Date
June 3, 2018

Final report

CPPS track

Barenholz, D. (0998941)
d.barenholz@student.tue.nl

Where innovation starts

Technische Universiteit
Eindhoven
University of Technology

Table of contents

Title
Final report

Where innovation starts

Contents of the report

Within this report I will write about the CPPS track. In the first section I will give an overview of what CPPS means and explain how the Honors track is organised. Afterwards, an outline of practising tools and practice done will be given. Then, an explanation of different types of competitions is given with reports on competitions that I participated with. Finally, an explanation of my seminar topic can be found.

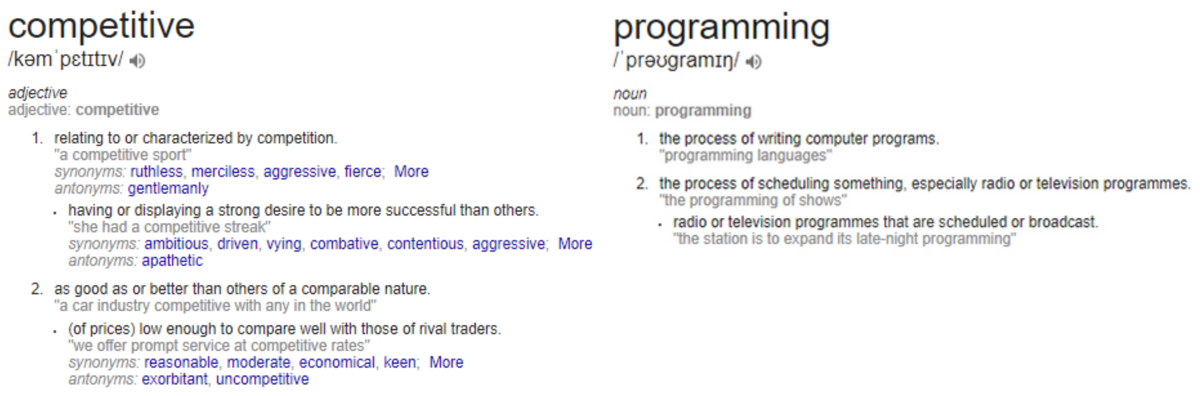
1 Introduction to CPPS

CPPS, which stands for *Competitive Programming and Problem Solving*, is one of the many tracks that the Honors Academy at Eindhoven University of Technology has to offer. I applied for the CPPS track since I wanted, and still want, to learn more about how people solve algorithmical problems and because I really enjoyed participating in my first programming contest. Another reason, although absent in my letter of application, is because I feel that programming should be a skill that everyone has the opportunity to acquire to some degree. Competitive programming at its core can be used, for example in secondary school, to introduce how fun programming (and problem solving) can be. This way, there will hopefully be more people who can write clean code, which should increase coding quality overall. I will now explain *what* competitive programming actually is, as well as the idea of problem solving.

1.1 Competitive Programming

As Shakespeare once said, *"What's in a name?"*, one may be wondering what competitive programming actually is. The mainstream idea of what programming is, is to write code for some kind of program. This program can be many things. Your web browser, e-mail clients and applications such as Microsoft Word, Adobe After Effects and Angry Birds are just a few of the millions of possible examples. Still, for every single one of these programs, there is one universal thing that holds. Someone, a programmer, had to write some code in order for it to work. However, *programming* itself doesn't have to be necessarily for an application per se. There can be other use cases for programming, for example, showing off your skills by generating a supposedly infinite number of prime numbers using only 53 bytes[1]or by a one-liner that converts any decimal number X to binary.[2]Another part of programming is that isn't always necessarily serious. A tribute to that fact are the programming languages called *Whitespace* (where one programs using spaces, tabs and enters) and *Brainfuck* (which works using only 8 characters). Of course, a programmer doesn't know all of these things by hard. We have Google. Perhaps the search engine can help out here to explain what competitive programming is?

[1] C code to do this: `main(m,k){for( m%k-?:(k=m++);k^1?printf("%i|",m));}`
[2] JavaScript code to do this: `(_$=($,_=[]+[])=>$?_$($»+!![],($&+!![])+_) _)(X)`


competitive
/kəmˈpɛtɪtɪv/

adjective
adjective: competitive

1. relating to or characterized by competition.
   "a competitive sport"
   synonyms: ruthless, merciless, aggressive, fierce; More
   antonyms: gentlemanly
   - having or displaying a strong desire to be more successful than others.
     "she had a competitive streak"
     synonyms: ambitious, driven, vying, combative, contentious, aggressive; More
     antonyms: apathetic
2. as good as or better than others of a comparable nature.
   "a car industry competitive with any in the world"
   - (of prices) low enough to compare well with those of rival traders.
     "we offer prompt service at competitive rates"
     synonyms: reasonable, moderate, economical, keen; More
     antonyms: exorbitant, uncompetitive

programming
/ˈprəʊɡramɪŋ/

noun
noun: programming

1. the process of writing computer programs.
   "programming languages"
2. the process of scheduling something, especially radio or television programmes.
   "the programming of shows"
   - radio or television programmes that are scheduled or broadcast.
     "the station is to expand its late-night programming"

(a) `define:competitive` (b) `define:programming`

Figure 1.1: Search queries in Google.

When combining both definitions, you may get that competitive programming is writing a *better* piece of code than others. This isn't too far from its actual meaning, but it raise the question what *better code* means. Is the shortest piece of code the best? Or is it the most human-readable piece of code? There's no real definition for *better code*, which is where you, as an aspiring programmer, decide to take action yourself and Google competitive programming. You immediately stumble upon Wikipedia [1] and find that competitive programming is *a mind sport usually held over the internet or a local network, involving participants trying to program according to provided specifications.* I won't continue to bore you by a Google query of *specifications* and then any other unknown words in this definition, however, I will tell you that this explanation of competitive programming is pretty close to its actual meaning. In essence, competitive programming is a contest or competition with multiple participants or competitors. The tasks and setup of a contest may vary, but with all of them there is some kind of problem to solve, and the participants write their code, either in group or alone, and submit their solution to the problem. The *better* piece of code is then the code that gives a better solution to the problem, or it is the piece of code that runs fastest. Of course, you can't participate in a programming competition and expect to do amazing if all you know is a programming language. You need the skill of problem solving to solve the problem(s) that are presented to you during a contest.

1.2 Problem Solving

I will explain the idea of problem solving, in the context of competitive programming, based on what I did to explain, or rather find a definition for, competitive programming itself. The *problem* that I *solved* was to *find a definition for competitive programming*. The way I solved that problem, is by doing certain things, in a certain order. This is the basis for any algorithm. An algorithm to code is what a recipe is to your favourite dish. It explains, step-by-step, how to solve a problem, which can be making your favourite dish. Within the world of competitive programming, an algorithm is a recipe that you can follow to solve a problem. Note that algorithms are usually written in pseudo-code. Examples will be encountered throughout this report, but here is the pseudo-code of the algorithm used to find a definition for competitive programming.

Algorithm 1: `myAlgo(String findADefForThis)`

Input: A compound sentence of one or multiple words.

Output: A definition for `findADefForThis`.

```
Procedure myAlgo(String findADefForThis)
    $L \leftarrow$ List of definitions of words
    for word $W$ in findADefForThis do
        Find definition $D$ of word $W$
        Add $D$ to $L$
    end
    return new definition created using $L$
```

Of course, above example is not what CPPS offers, it is merely an introduction to the concept of problem solving. I now present a possible problem that CPPS students can solve, in the same format as we get them.

Problem: BAPC10D - Collatz

Problem

In the process to solve the *Collatz conjecture*, better known as the $3n+1$ *problem*, Carl created a physical model with wood and ropes. A wooden bar contains a hole for every natural number from 1 to infinity from left to right. For every even number m there is a rope connecting the mth hole with hole $\frac{m}{2}$. For every odd number n there is a rope connecting the nth hole with hole $3n+1$. For an important conference where Carl plans to elaborate on his results, he wants to bring his structure, but it is too large to fit in his bag. So he decided to saw off the part of the bar containing the first N holes only. How many ropes will he need to cut?

Input

The first line of the input contains a single number: the number of test cases to follow. Each test case has the following format:

- One line with an integer N, satisfying $0 \leq N \leq 10^9$.

Output

For every test case in the input, the output should contain a single number, on a single line: the number of ropes that need to be cut.

Example

Input	Output
3	10
12	200
240	3000
3600	

Solving: BAPC10D - Collatz

When solving this kind of problem, there are some steps that you usually go through.

1. *Read the problem.*
   By reading the problem, you know what information there is and what you're trying to solve. In this case, you know there is some structure Carl made that consists of ropes that need to be cut, and you're trying to find the minimum number of ropes that he needs to cut.

2. *Guess the time complexity.*
   When you look at the input, you see that $0 \leq N \leq 10^9$. This means that a linear $O(n)$ solution is likely too slow. Thus, you're supposed to find a constant time $O(1)$ solution. In this case, it means that you're looking for some kind of function of m, n and N.

3. *Find a solution.*
   The next logical step would be to find, in this case, the function. Depending on how good you are with logical thinking, this may take a very long time to figure out, or you may see it immediately. However you find the solution, one thing should be clear, which is the fact that you need to compute how many ropes there are in the structure. But even this computation isn't straightforward at first glance. You have to differentiate between odd n and even m in following way:

   - Compute the number of ropes from m to $2m$ with $m \leq N$ and $N < 2m$.
   - Compute the number of ropes from n to $3n + 1$ with $n \leq N$ and $N < 3n + 1$.

   By going through both of these cases, you compute the total number of ropes in the structure. Now, you can, with some thinking, find that in total there will be k ropes cut, where k equals to:

$$\left\lceil \frac{N}{2} \right\rceil + \left\lceil \frac{N}{2} \right\rceil - \left\lceil \frac{\left\lfloor \frac{N-1}{3} \right\rfloor}{2} \right\rceil$$

4. *Code the solution.*
   The next and final step is to put your solution into code. This isn't always straightforward. In the case of the Collatz problem, it is straightforward. In fact, it's even easier than I described the solution to be. There is, in fact, no need to compute any number of ropes. The only thing that should be done is computing the above equation. Please see the appendix for Java and Python code on how this is done.

1.3 The Honors Track

The CPPS track gives you, as student, the opportunity to pursue competitive programming and problem solving. In contrast with other Honors tracks offered by the Honors Academy, the CPPS track doesn't have a team-based project. In stead, students are challenged to take the lead in their own development of problem solving. You are given a great amount of freedom due to the lack of a project, which comes with its respective responsibility.

The CPPS track is organised in following way. Throughout the year, there are several general meetings. At the first general meeting, all students are divided into teams of 3 people. Two teams make up a *supergroup*. Each supergroup has biweekly meetings with their respective coach, where they discuss their current progress. These meetings are the perfect time for students from different teams to learn from each other. Team A may have solved some problems that team B hasn't. Within these meeting solutions to these problem are explained, together with the thought process behind them.

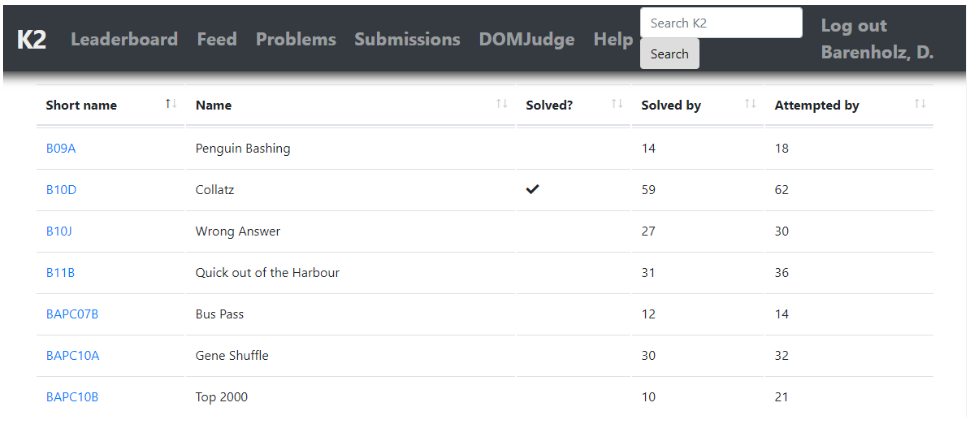
In the first quarter, the general meetings exists of preparation for the EAPC (a programming contest) and lectures on specific programming topics, such as dynamic programming, graph algorithms and computational geometry. General tips on how to program during a contest are also given. In the second and third quarter, students get to choose from a variety of topics and are required to give a seminar on their chosen topic. Finally, in quarter four, students write their final report and will be assessed.

Throughout the whole year, students are expected to be training for, and participating with, programming contests. In the following sections I will explain how you can train for contests and what types of contests there are.

2 Practice

2.1 Practising tools

1. **K2 Server @ `https://compprog.win.tue.nl/problems`**

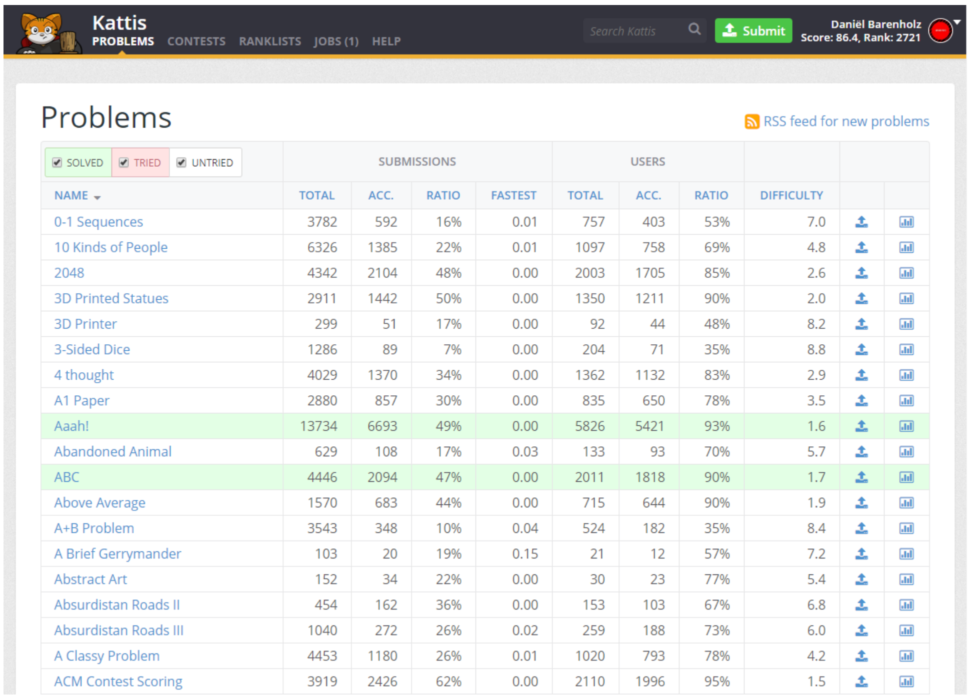


Short name	Name	Solved?	Solved by	Attempted by
B09A	Penguin Bashing		14	18
B10D	Collatz	✓	59	62
B10J	Wrong Answer		27	30
B11B	Quick out of the Harbour		31	36
BAPC07B	Bus Pass		12	14
BAPC10A	Gene Shuffle		30	32
BAPC10B	Top 2000		10	21

Figure 2.1: The problems page of the K2 server

The K2 server, also known as the `compprog` server, is a server specifically designed for students from the CPPS honors track. It contains most of the problems of all EAPC, BAPC and NWERC competitions, as well as the problems of competitions organised by the honors students themselves.

2. **Kattis @ `https://open.kattis.com/problems`**

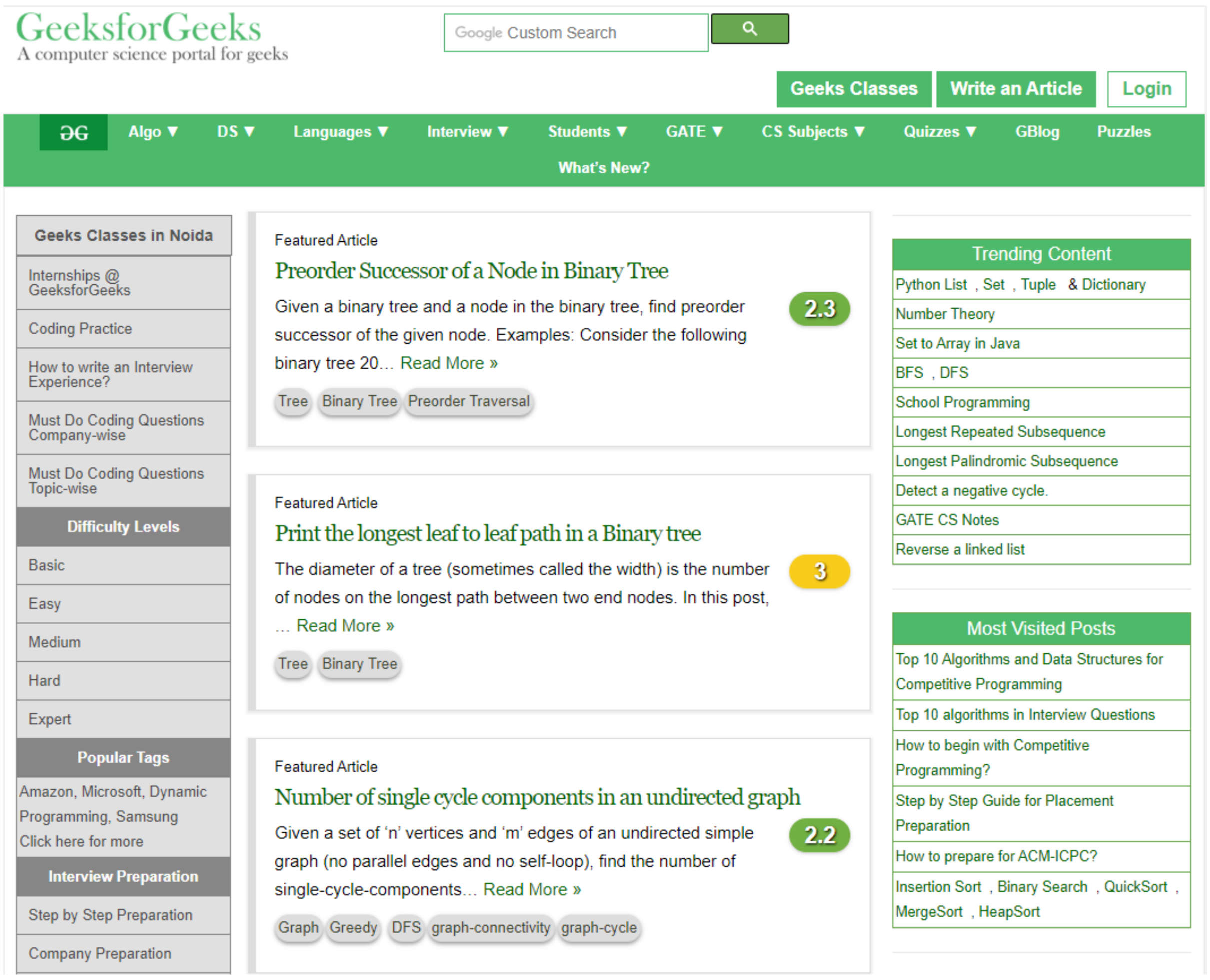


	SUBMISSIONS				USERS					
NAME	TOTAL	ACC.	RATIO	FASTEST	TOTAL	ACC.	RATIO	DIFFICULTY		
0-1 Sequences	3782	592	16%	0.01	757	403	53%	7.0		
10 Kinds of People	6326	1385	22%	0.01	1097	758	69%	4.8		
2048	4342	2104	48%	0.00	2003	1705	85%	2.6		
3D Printed Statues	2911	1442	50%	0.00	1350	1211	90%	2.0		
3D Printer	299	51	17%	0.00	92	44	48%	8.2		
3-Sided Dice	1286	89	7%	0.00	204	71	35%	8.8		
4 thought	4029	1370	34%	0.00	1362	1132	83%	2.9		
A1 Paper	2880	857	30%	0.00	835	650	78%	3.5		
Aaah!	13734	6693	49%	0.00	5826	5421	93%	1.6		
Abandoned Animal	629	108	17%	0.03	133	93	70%	5.7		
ABC	4446	2094	47%	0.00	2011	1818	90%	1.7		
Above Average	1570	683	44%	0.00	715	644	90%	1.9		
A+B Problem	3543	348	10%	0.04	524	182	35%	8.4		
A Brief Gerrymander	103	20	19%	0.15	21	12	57%	7.2		
Abstract Art	152	34	22%	0.00	30	23	77%	5.4		
Absurdistan Roads II	454	162	36%	0.00	153	103	67%	6.8		
Absurdistan Roads III	1040	272	26%	0.02	259	188	73%	6.0		
A Classy Problem	4453	1180	26%	0.01	1020	793	78%	4.2		
ACM Contest Scoring	3919	2426	62%	0.00	2110	1996	95%	1.5		

Figure 2.2: The problems page of the Kattis server

The Kattis server is one I include on this page because of its huge database of problems. All problems come from some kind of programming competition. Apart from that, it hosts competitions on a regular basis.

3. **Geeks for Geeks** @ `https://www.geeksforgeeks.org/`



Featured Article
Preorder Successor of a Node in Binary Tree
Given a binary tree and a node in the binary tree, find preorder successor of the given node. Examples: Consider the following binary tree 20... Read More »
2.3
Tree | Binary Tree | Preorder Traversal

Featured Article
Print the longest leaf to leaf path in a Binary tree
The diameter of a tree (sometimes called the width) is the number of nodes on the longest path between two end nodes. In this post, ... Read More »
3
Tree | Binary Tree

Featured Article
Number of single cycle components in an undirected graph
Given a set of 'n' vertices and 'm' edges of an undirected simple graph (no parallel edges and no self-loop), find the number of single-cycle-components... Read More »
2.2
Graph | Greedy | DFS | graph-connectivity | graph-cycle

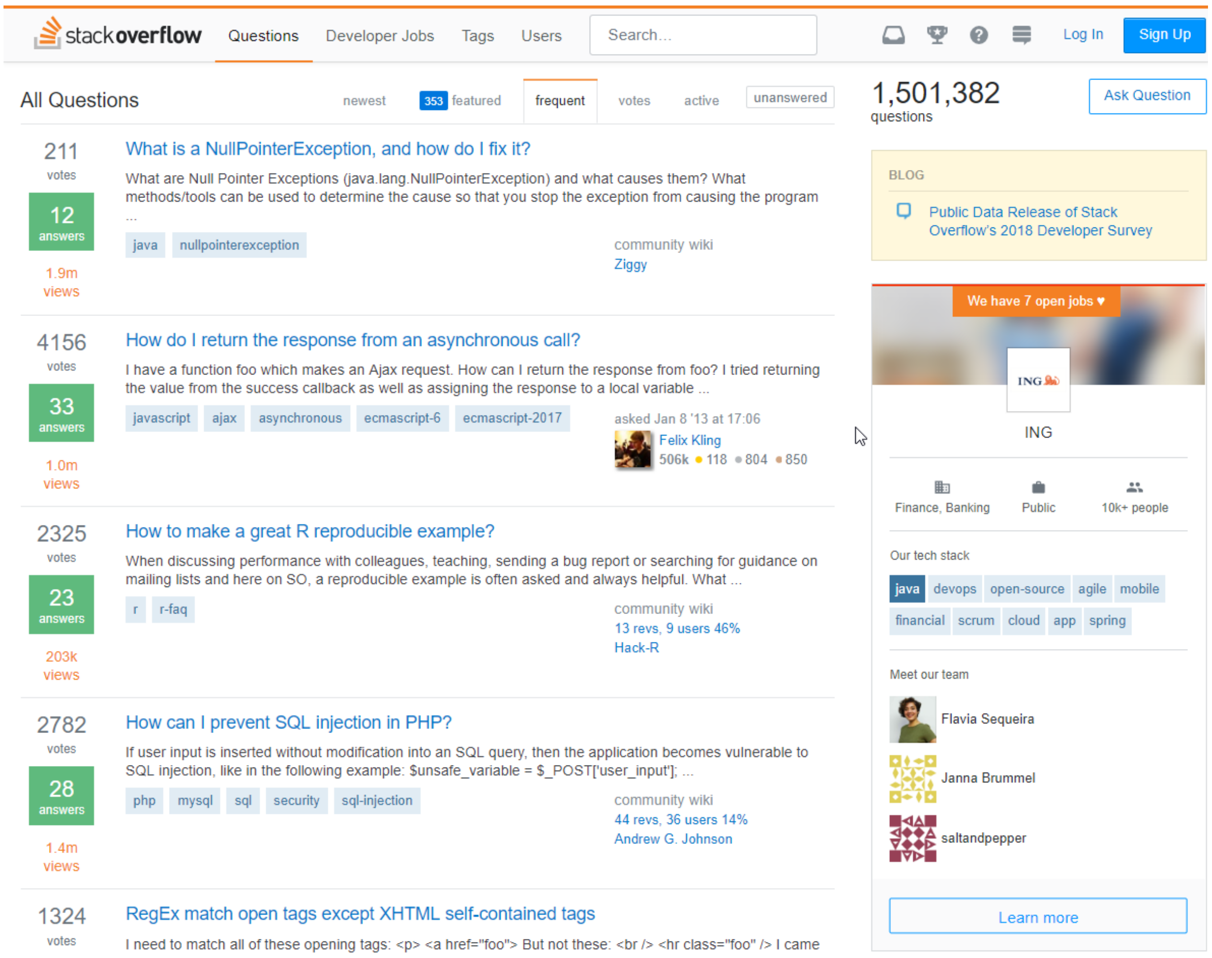
Trending Content
- Python List , Set , Tuple & Dictionary
- Number Theory
- Set to Array in Java
- BFS , DFS
- School Programming
- Longest Repeated Subsequence
- Longest Palindromic Subsequence
- Detect a negative cycle.
- GATE CS Notes
- Reverse a linked list

Most Visited Posts
- Top 10 Algorithms and Data Structures for Competitive Programming
- Top 10 algorithms in Interview Questions
- How to begin with Competitive Programming?
- Step by Step Guide for Placement Preparation
- How to prepare for ACM-ICPC?
- Insertion Sort , Binary Search , QuickSort , MergeSort , HeapSort

Figure 2.3: The home page of `www.geeksforgeeks.org`.

Geeks for geeks is a site with a ton of information. It contains articles on many known problems, of various types. Solutions are most of the times provided in C++, Java and Python. This site will popup in your Google searches for a problem.

4. **Stack Overflow** @ `https://stackoverflow.com/questions?sort=frequent`


stackoverflow Questions Developer Jobs Tags Users Search... Log In Sign Up

All Questions — newest, 353 featured, frequent, votes, active, unanswered — 1,501,382 questions — Ask Question

211 votes, 12 answers, 1.9m views — **What is a NullPointerException, and how do I fix it?**
What are Null Pointer Exceptions (java.lang.NullPointerException) and what causes them? What methods/tools can be used to determine the cause so that you stop the exception from causing the program ...
java nullpointerexception — community wiki Ziggy

4156 votes, 33 answers, 1.0m views — **How do I return the response from an asynchronous call?**
I have a function foo which makes an Ajax request. How can I return the response from foo? I tried returning the value from the success callback as well as assigning the response to a local variable ...
javascript ajax asynchronous ecmascript-6 ecmascript-2017 — asked Jan 8 '13 at 17:06 Felix Kling 506k ● 118 ● 804 ● 850

2325 votes, 23 answers, 203k views — **How to make a great R reproducible example?**
When discussing performance with colleagues, teaching, sending a bug report or searching for guidance on mailing lists and here on SO, a reproducible example is often asked and always helpful. What ...
r r-faq — community wiki 13 revs, 9 users 46% Hack-R

2782 votes, 28 answers, 1.4m views — **How can I prevent SQL injection in PHP?**
If user input is inserted without modification into an SQL query, then the application becomes vulnerable to SQL injection, like in the following example: $unsafe_variable = $_POST['user_input']; ...
php mysql sql security sql-injection — community wiki 44 revs, 36 users 14% Andrew G. Johnson

1324 votes — **RegEx match open tags except XHTML self-contained tags**
I need to match all of these opening tags: <p> <a href="foo"> But not these: <br /> <hr class="foo" /> I came

BLOG — Public Data Release of Stack Overflow's 2018 Developer Survey

We have 7 open jobs ♥ — ING — Finance, Banking · Public · 10k+ people
Our tech stack: java devops open-source agile mobile financial scrum cloud app spring
Meet our team: Flavia Sequeira, Janna Brummel, saltandpepper
Learn more

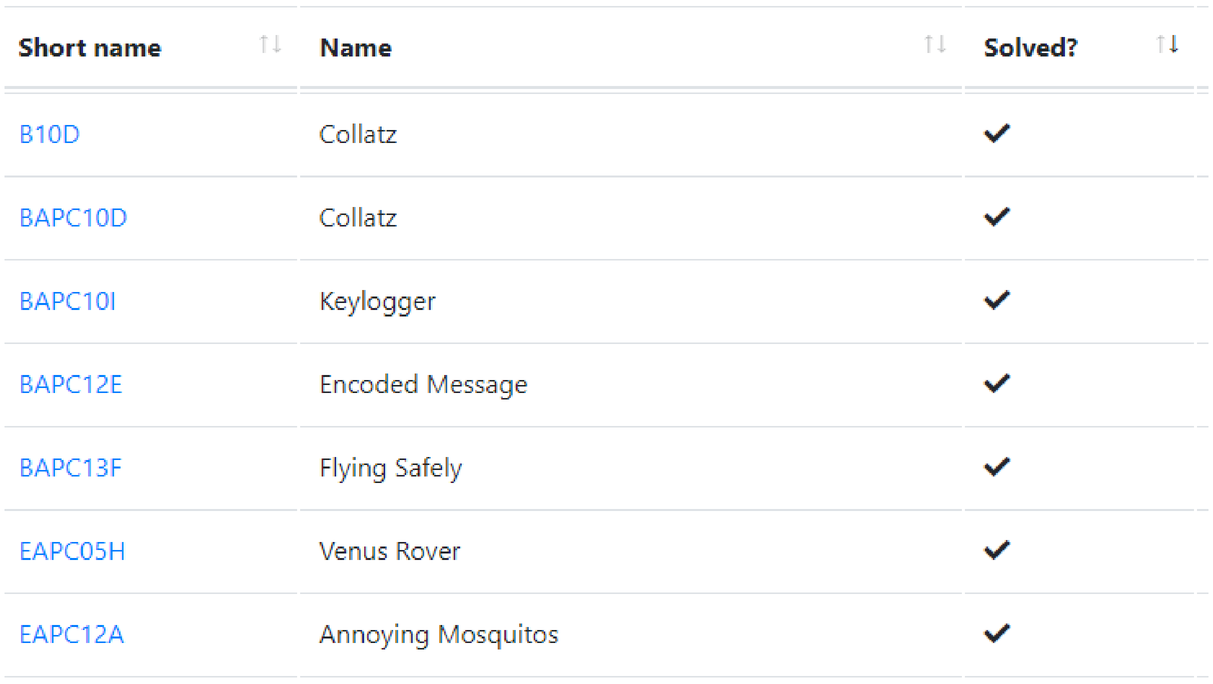
Figure 2.4: Most frequent questions on Stack Overflow.

Another website which will most definitely show up in Google searches whenever you occur an error, or when you're looking for something specific. It's part of a network of extremely useful sites. There is a math overflow for your math related problems and a tex overflow for your LaTeX related problems.

2.2 Practice done

Throughout the year I have solved a few problems on the K2 server. I first give an overview of all solved questions this server, and then go into detail for some of them. Afterwards I show my progress on Kattis in the same manner.

K2 Server



Short name	Name	Solved?
B10D	Collatz	✓
BAPC10D	Collatz	✓
BAPC10I	Keylogger	✓
BAPC12E	Encoded Message	✓
BAPC13F	Flying Safely	✓
EAPC05H	Venus Rover	✓
EAPC12A	Annoying Mosquitos	✓

Figure 2.5: Solved questions on K2 server

I have already explained the Collatz problem (BAPC10D) earlier, so we won’t look at it again. In stead, I will look at the *Keylogger* and *Encoded Message* problems and provide pseudocode to solve both.

Problem: BAPC10I - Keylogger

Problem

As a malicious hacker you are trying to steal your mother’s password, and therefore you have installed a keylogger on her PC (or Mac, so you like). You have a log from your mother typing the password, but unfortunately the password is not directly visible because she used the left and right arrows to change the position of the cursor, and the backspace to delete some characters. Write a program that can decode the password from the given keylog.

Input

The first line of the input contains a single number: the number of test cases to follow. Each test case has the following format:

- One line with a string L, satisfying $1 \leq Length(L) \leq 1.000.000$, consisting of:
  - $'-'$ representing backspace: the character directly before the cursor position is deleted, if there is any.
  - $'<'$ (and $'>'$) representing the left (right) arrow: the cursor is moved 1 character to the left (right), if possible.
  - alphanumeric characters, which are part of the password, unless deleted later. We assume âĂŸinsert mode’: if the cursor is not at the end of the line, and you type

an alphanumeric character, then all characters after the cursor move one position to the right.

Every decoded password will be of length > 0.

Output

For every test case in the input, the output should contain a single string, on a single line: the decoded password.

Example

Input	Output
2	BAPC
«BP<A»Cd-	ThIsIsS3Cr3t
ThIsIsS3Cr3t	

Solution: BAPC10I - Keylogger

The solution to this problem is rather trivial. You simply have to simulate what the user was typing as password. To do this, you have to store the typed string, and iterate over it character by character, which results in a linear time algorithm. Below you find an algorithm which solves the problem for one case.

```
Algorithm 2: Keylogger(String L)
Input:    A keylogged password $L$.
Output:   The original password.

Procedure Keylogger(String $L$)
    $R \leftarrow$ Empty list of characters
    for character $c$ in $L$ do
        switch $c$ do
            case '-' do
                Delete character at current index and update index
            end
            case '>' do
                Increment index
            end
            case '<' do
                Decrement index
            end
            case default do
                Copy $c$ from $L$ to $R$ at current index
            end
        end
    end
    return String $R$
```

Java code for this problem can be found as appendix.

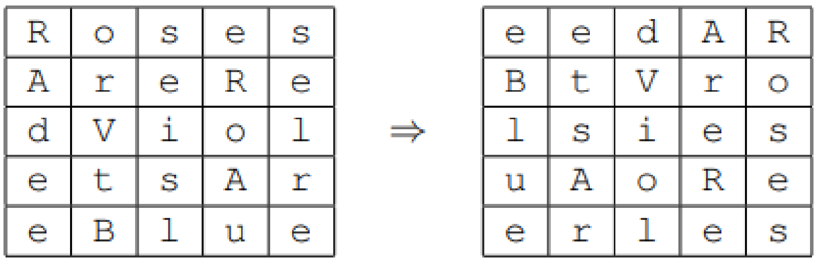
Problem: BAPC 12E - Encoded Message

Problem

Alex wants to send a love poem to his girlfriend Bridget. Unfortunately, she has a nosy friend, Ellen, who might intercept his message and invade their privacy.

To prevent this, Alex has invented a scheme to make his missives indecipherable to Ellen. He arranges the letters into a square, which is rotated a quarter-turn clockwise, and then he puts the resulting letters on a single line again. (For simplicity's sake, Alex doesn't use whitespace or punctuation in his poems.)

For example, the text `RosesAreRedVioletsAreBlue` would be encoded as `eedARBtVrolsiesuAoReerles` using the following intermediate steps:



R	o	s	e	s
A	r	e	R	e
d	V	i	o	l
e	t	s	A	r
e	B	l	u	e

⇒

e	e	d	A	R
B	t	V	r	o
l	s	i	e	s
u	A	o	R	e
e	r	l	e	s

Ellen has intercepted some of Alex's messages but they make no sense to her. Can you write a program to help her decode them?

Input

On the first line one positive number: the number of test cases, at most 100. After that per test case:

- one line with an encoded message: a string consisting of upper-case and lower-case letters only. The length of the message is a square between 1 and 10000 characters.

Output

Per test case:

- one line with the original message.

Sample in- and output

Input

```
3
RSTEEOTCP
eedARBtVrolsiesuAoReerles
EarSvyeqeBsuneMa
```

Output

```
TOPSECRET
RosesAreRedVioletsAreBlue
SquaresMayBeEven
```

Solution: BAPC 12E - Encoded Message

It is quite clear how to solve this problem. I first present pseudo-code on how to decode a message, and then proceed to explain where necessary.

Algorithm 3: `Decode(String M)`

Input: An encoded message M

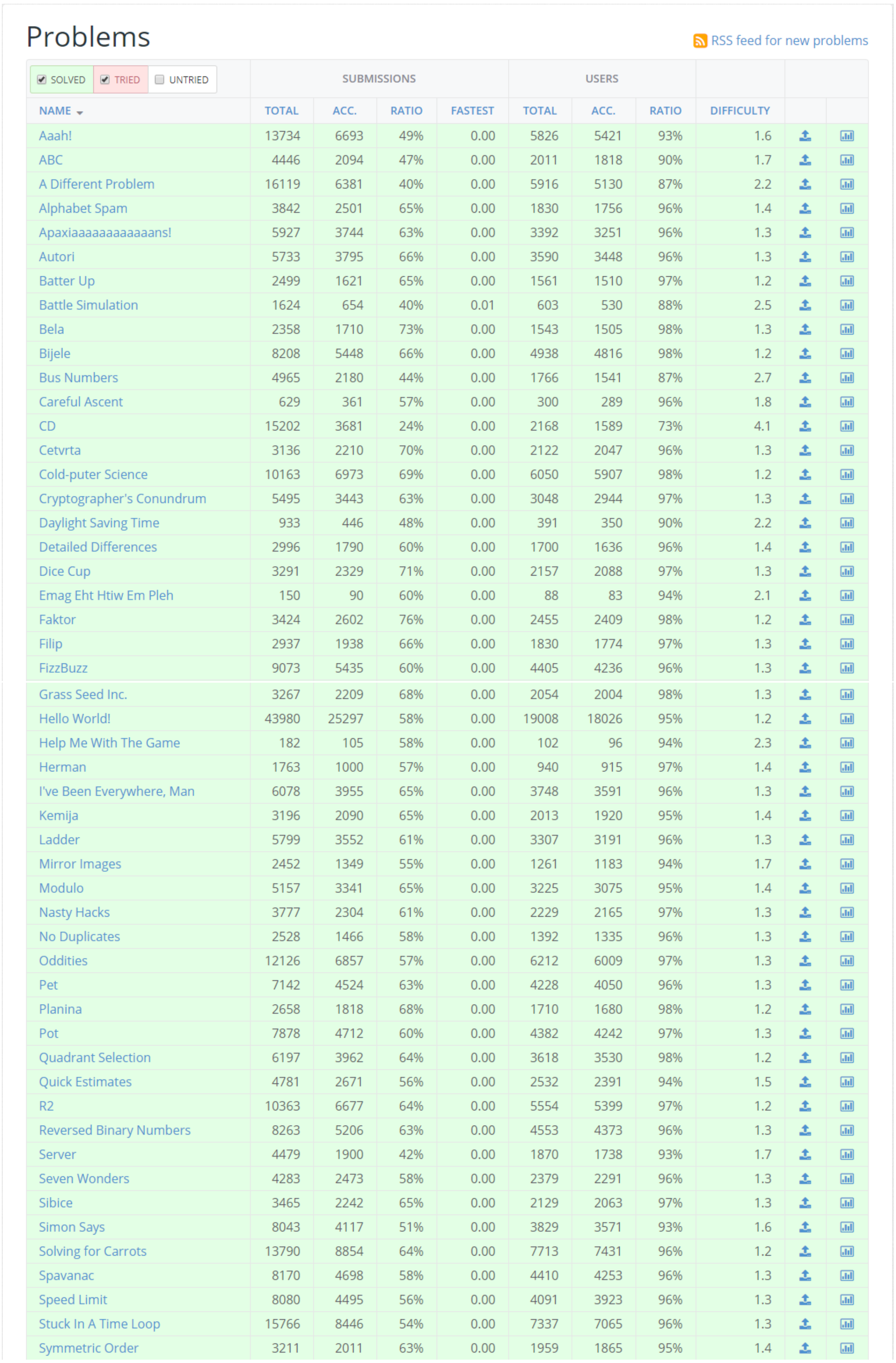
Output: A decoded message of M

```
Procedure Decode (String M)
    out ← empty string
    length ← length(M)
    squareSize ← sqaure root of length
    for i = 0 to squareSize do
        for j = 0 to squareSize do
            add character at index "squareSize - i - 1 + j * squaresize" to out
        end
    end
    return String out
```

There are a couple things that I could explain. First, since we know that the length of the message is a square, we know that the root of this length is the size of the table. Second and last, line 7. I add the character at index `squareSize - i - 1 + j * squareSize` to the output. Initially, both i and j are 0, hence we add the character at index *squareSize - 1*. When you take a look at the table in the problem description, you can see that this is the first letter of the message. Next, the character at index *squareSize - 1 - 1* is added. This is the second letter of the message. By going through the double for loop, every single letter gets added in the correct order, effectively decoding the encoded message and solving the problem.

Java code for this problem can be found as appendix.

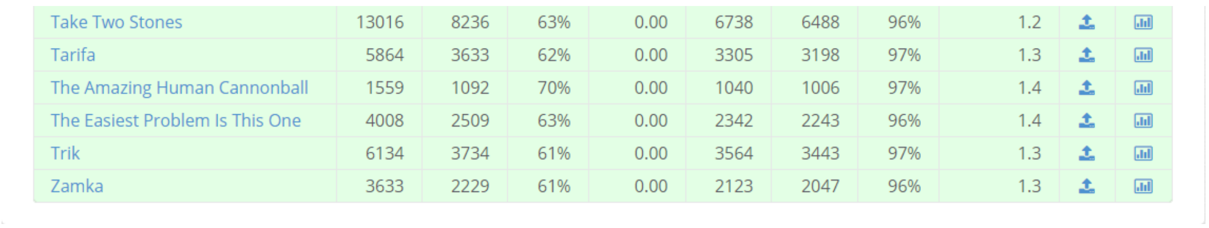
Kattis Server



Problems

RSS feed for new problems

☑ SOLVED ☑ TRIED ☐ UNTRIED	SUBMISSIONS				USERS			
NAME	TOTAL	ACC.	RATIO	FASTEST	TOTAL	ACC.	RATIO	DIFFICULTY
Aaah!	13734	6693	49%	0.00	5826	5421	93%	1.6
ABC	4446	2094	47%	0.00	2011	1818	90%	1.7
A Different Problem	16119	6381	40%	0.00	5916	5130	87%	2.2
Alphabet Spam	3842	2501	65%	0.00	1830	1756	96%	1.4
Apaxiaaaaaaaaaaaans!	5927	3744	63%	0.00	3392	3251	96%	1.3
Autori	5733	3795	66%	0.00	3590	3448	96%	1.3
Batter Up	2499	1621	65%	0.00	1561	1510	97%	1.2
Battle Simulation	1624	654	40%	0.01	603	530	88%	2.5
Bela	2358	1710	73%	0.00	1543	1505	98%	1.3
Bijele	8208	5448	66%	0.00	4938	4816	98%	1.2
Bus Numbers	4965	2180	44%	0.00	1766	1541	87%	2.7
Careful Ascent	629	361	57%	0.00	300	289	96%	1.8
CD	15202	3681	24%	0.00	2168	1589	73%	4.1
Cetvrta	3136	2210	70%	0.00	2122	2047	96%	1.3
Cold-puter Science	10163	6973	69%	0.00	6050	5907	98%	1.2
Cryptographer's Conundrum	5495	3443	63%	0.00	3048	2944	97%	1.3
Daylight Saving Time	933	446	48%	0.00	391	350	90%	2.2
Detailed Differences	2996	1790	60%	0.00	1700	1636	96%	1.4
Dice Cup	3291	2329	71%	0.00	2157	2088	97%	1.3
Emag Eht Htiw Em Pleh	150	90	60%	0.00	88	83	94%	2.1
Faktor	3424	2602	76%	0.00	2455	2409	98%	1.2
Filip	2937	1938	66%	0.00	1830	1774	97%	1.3
FizzBuzz	9073	5435	60%	0.00	4405	4236	96%	1.3
Grass Seed Inc.	3267	2209	68%	0.00	2054	2004	98%	1.3
Hello World!	43980	25297	58%	0.00	19008	18026	95%	1.2
Help Me With The Game	182	105	58%	0.00	102	96	94%	2.3
Herman	1763	1000	57%	0.00	940	915	97%	1.4
I've Been Everywhere, Man	6078	3955	65%	0.00	3748	3591	96%	1.3
Kemija	3196	2090	65%	0.00	2013	1920	95%	1.4
Ladder	5799	3552	61%	0.00	3307	3191	96%	1.3
Mirror Images	2452	1349	55%	0.00	1261	1183	94%	1.7
Modulo	5157	3341	65%	0.00	3225	3075	95%	1.4
Nasty Hacks	3777	2304	61%	0.00	2229	2165	97%	1.3
No Duplicates	2528	1466	58%	0.00	1392	1335	96%	1.3
Oddities	12126	6857	57%	0.00	6212	6009	97%	1.3
Pet	7142	4524	63%	0.00	4228	4050	96%	1.3
Planina	2658	1818	68%	0.00	1710	1680	98%	1.2
Pot	7878	4712	60%	0.00	4382	4242	97%	1.3
Quadrant Selection	6197	3962	64%	0.00	3618	3530	98%	1.2
Quick Estimates	4781	2671	56%	0.00	2532	2391	94%	1.5
R2	10363	6677	64%	0.00	5554	5399	97%	1.2
Reversed Binary Numbers	8263	5206	63%	0.00	4553	4373	96%	1.3
Server	4479	1900	42%	0.00	1870	1738	93%	1.7
Seven Wonders	4283	2473	58%	0.00	2379	2291	96%	1.3
Sibice	3465	2242	65%	0.00	2129	2063	97%	1.3
Simon Says	8043	4117	51%	0.00	3829	3571	93%	1.6
Solving for Carrots	13790	8854	64%	0.00	7713	7431	96%	1.2
Spavanac	8170	4698	58%	0.00	4410	4253	96%	1.3
Speed Limit	8080	4495	56%	0.00	4091	3923	96%	1.3
Stuck In A Time Loop	15766	8446	54%	0.00	7337	7065	96%	1.3
Symmetric Order	3211	2011	63%	0.00	1959	1865	95%	1.4



Take Two Stones	13016	8236	63%	0.00	6738	6488	96%	1.2	⬆	▥
Tarifa	5864	3633	62%	0.00	3305	3198	97%	1.3	⬆	▥
The Amazing Human Cannonball	1559	1092	70%	0.00	1040	1006	97%	1.4	⬆	▥
The Easiest Problem Is This One	4008	2509	63%	0.00	2342	2243	96%	1.4	⬆	▥
Trik	6134	3734	61%	0.00	3564	3443	97%	1.3	⬆	▥
Zamka	3633	2229	61%	0.00	2123	2047	96%	1.3	⬆	▥

Figure 2.6: Solved questions on Kattis server

I will now go over a few fun and interesting problems.

Problem: CD (Waterloo Programming Contest 2010-09-26)

Problem

Jack and Jill have decided to sell some of their Compact Discs, while they still have some value. They have decided to sell one of each of the CD titles that they both own. How many CDs can Jack and Jill sell?

Neither Jack nor Jill owns more than one copy of each CD.

Input

The input consists of a sequence of test cases. The first line of each test case contains two non-negative integers N and M, each at most one million, specifying the number of CDs owned by Jack and by Jill, respectively. This line is followed by N lines listing the catalog numbers of the CDs owned by Jack in increasing order, and M more lines listing the catalog numbers of the CDs owned by Jill in increasing order. Each catalog number is a positive integer no greater than one billion. The input is terminated by a line containing two zeros. This last line is not a test case and should not be processed.

Output

For each test case, output a line containing one integer, the number of CDs that Jack and Jill both own.

Example

Sample Input 1

```
3 3
1
2
3
1
2
4
0 0
```

Sample Output 1

```
2
```

Solution: CD (Waterloo Programming Contest 2010-09-26)

When reading the problem it becomes clear that the number of CDs that Jack and Jill both own is a very simply calculation. The problem however, lays in the time constraint of this question, which is 2 seconds. If in Java you use a Scanner in stead of a `BufferedWriter` with my proposed solution, you will run out of time. In fact, my solution is *barely* fast enough, coming in at 1.93 seconds.

My proposed solution is to use sets. The number of CDs that can be sold is precisely the number of CDs that Jill owns added with the number of CDs that Jack owns, subtracted by the number of CDs that both own. So, by storing three sets one can very easily solve this.

```
Procedure CD()
    $N \leftarrow$ CDs owned by Jack
    $M \leftarrow$ CDs owned by Jill
    Jill ← empty set
    Jack ← empty set
    Both ← empty set
    for i = 0 to N do
        Add CD to Jack
        Add CD to Both
    end
    for i = 0 to M do
        Add CD to Jill
        Add CD to Both
    end
    return Jack.Size + Jill.Size - Both.Size
```

Java code for this problem can be found as appendix.

Problems: Help Me With The Game & Emag Eht Htiw Em Pleh (CTU Open 2005)

Problem

Help Me With The Game: Your task is to read a picture of a chessboard position and print it in the chess notation.

Emag Eht Htiw Em Pleh: This problem is a reverse case of the problem “Help Me With The Game". You are given the output from that problem, and your task is to find the corresponding input.

Input

Help Me With The Game: The input consists of an ASCII-art picture of a chessboard with chess pieces on positions described by the input. The pieces of the white player are shown in upper-case letters, while the black playerâĂŹs pieces are lower-case letters. The letters are one of “K" (King), “Q" (Queen), “R" (Rook), “B" (Bishop), “N" (Knight), or “P" (Pawn). The chessboard outline is made of plus (“+"), minus (“-"), and pipe (“|") characters. The black fields are filled with colons (“:"), white fields with dots (“.").

Emag Eht Htiw Em Pleh: The input follows the output specification of the problem “Help Me With The Game".

Output

Help Me With The Game: The output consists of two lines. The first line consists of the string “White: ", followed by the description of positions of the pieces of the white player. The second line consists of the string “Black: ", followed by the description of positions of the pieces of the black player.

The description of the position of the pieces is a comma-separated list of terms (without any spaces) describing the pieces of the appropriate player. The description of a piece consists of a single upper-case letter that denotes the type of the piece (except for pawns, for that this identifier is omitted). This letter is immediately followed by the position of the piece in the standard chess notation âĂŞ a lower-case letter between “a" and “h" that determines the column (“a" is the leftmost column in the input) and a single digit between 1 and 8 that determines the row (8 is the first row in the input).

The pieces in the description must appear in the following order: Kings (“K"), Queens (“Q"), Rooks (“R"), Bishops (“B"), Knights (“N"), and pawns.

Note that the numbers of pieces may differ from the initial position, and might not even correspond to a situation that could arise in a valid chess game.

In case two pieces of the same type appear in the input, the piece with the smaller row number must be described before the other one if the pieces are white, and the one with the larger row number must be described first if the pieces are black. If two pieces of the same type appear in the same row, the one with the smaller column letter must appear first.

Emag Eht Htiw Em Pleh: The output must be a valid input for the problem "Help Me With The Game", corresponding to the given input data.

Example

Sample Input 1

```
+---+---+---+---+---+---+---+---+
|.r.|:::|.b.|:q:|.k.|:::|.n.|:r:|
+---+---+---+---+---+---+---+---+
|:p:|.p.|:p:|.p.|:p:|.p.|:::|.p.|
+---+---+---+---+---+---+---+---+
|...|:::|.n.|:::|...|:::|...|:p:|
+---+---+---+---+---+---+---+---+
|:::|...|:::|...|:::|...|:::|...|
+---+---+---+---+---+---+---+---+
|...|:::|...|:::|.P.|:::|...|:::|
+---+---+---+---+---+---+---+---+
|:P:|...|:::|...|:::|...|:::|...|
+---+---+---+---+---+---+---+---+
|.P.|:::|.P.|:P:|...|:P:|.P.|:P:|
+---+---+---+---+---+---+---+---+
|:R:|.N.|:B:|.Q.|:K:|.B.|:::|.R.|
+---+---+---+---+---+---+---+---+
```

Sample Output 1

```
White:  Ke1,Qd1,Ra1,Rh1,
Bc1,Bf1,Nb1,a2,c2,d2,f2,
g2,h2,a3,e4
Black:  Ke8,Qd8,Ra8,Rh8,
Bc8,Ng8,Nc6,a7,b7,
c7,d7,e7,f7,h7,h6
```

Sample Output 2

```
White:
Black:  Kh5,Ke1
```

Sample Input 2

```
+---+---+---+---+---+---+---+---+
|...|:::|...|:::|...|:::|...|:::|
+---+---+---+---+---+---+---+---+
|:::|...|:::|...|:::|...|:::|...|
+---+---+---+---+---+---+---+---+
|...|:::|...|:::|...|:::|...|:::|
+---+---+---+---+---+---+---+---+
|:::|...|:::|...|:::|...|:::|.k.|
+---+---+---+---+---+---+---+---+
|...|:::|...|:::|...|:::|...|:::|
+---+---+---+---+---+---+---+---+
|:::|...|:::|...|:::|...|:::|...|
+---+---+---+---+---+---+---+---+
|...|:::|...|:::|...|:::|...|:::|
+---+---+---+---+---+---+---+---+
|:::|...|:::|...|:k:|...|:::|...|
+---+---+---+---+---+---+---+---+
```

Commented solutions (in Java) have been provided as appendices. The comments that were added should make everything clear.

3 Contests

3.1 Types of contests

There are a few broad types of contests. Most common are the algorithmical kind of contests, where you have a problem that you're trying to solve optimally with some time and memory constraints. With these contest, generally, you gather your team in real life together in a hub. When you then solve a problem, you'll get a balloon. After the contest (which typically takes 5 hours), solutions to the problems are presented by the organisators. During a contest, generally, there will be one laptop provided running a Linux distribution. Programming IDEs such as Eclipse and IntelliJ are installed on these laptops, as well as documentation (usually). This means that there can only be one person coding at the same time. Teams, which usually consist of up to 3 people, are supposed to come up with a strategy on how to work around this fact. A possible solution is pen-coding, where you write out actual code on paper, which is in my opinion really valuable. It shows the deficiencies in the solution you have in mind, which you need to solve somehow. By doing this on paper, in stead of behind the laptop, you save time.

A second type is an Artificial Intelligence contest. With these contests, participants are supposed to write an AI for a specially designed game. The duration of these contests may vary greatly, but usually these contests take around 1 month as they are tournament based. Generally, these contests are online and you can participate from anywhere. It is up to the participating team to stay organised and focused on creating a solution, within the provided framework of the game (usually in a few languages including Java, C++ and Python).

Finally, there are engineering contests, where you're presented with a difficult problem, and you have to write some code that gives a *good* output. Mostly, these engineering challenges do not have some optimal solution. Usually, the only thing that the organisator of the challenge is interested in, is the output that your code creates. This means that you can use any programming language you want, as long as you create the output in the correct format (`.txt` file).

3.2 Algorithmical: EAPC

The EAPC - Eindhoven Algorithmical Programming Contest - took place 23th of September 2017. I participated together with Storm de Zeeuw and Nihal Goel, but unfortunately Nihal ended up being sick. Our teamname was $e^{i \cdot \pi} + 1 = 0$.

To prepare for the contest we did several things. First of all, there were lectures from Kevin on general algorithmic problems and how to tackle them, as I explained earlier. These general meetings also provided you with practice questions. We held a meeting to discuss what should be in our *cheatsheet*, which we could use during the competition for pointers on how to approach various problems. The cheatsheet contained not only relevant algorithmic information but also the raw code for the algorithms themselves so that we could quickly replicate an algorithm whenever the need arose. Daniël took it upon himself to create this cheatsheet. We also had individual meetings where we would get together and discuss the EAPC problems from the previous years and work on them as a team to be able to replicate the work flow in the competition itself.

The contest ended up being extremely hard in comparison with previous years, where we had solved more than 5 problems. We now ended up solving only 2 problems. This was, however, enough to land us in 6th place out of 22 or so teams with the 4th place having solved 4 problems and the best team having solved 6 problems. A major problem we encountered during this contest was efficiency and communication. Storm and Daniël had a hard time communicating, and a basic structure of command wasn't really present. We both did the best we could individually but there was next to no teamwork except for the double checking of each others code. We had discussed some teamwork tactics beforehand but due to Nihal being sick all plans were rather useless and we were thrown off guard. We decided that next time we should have a backup plan and prepare for harder problems by meeting more and discussing the problems from this EAPC in particular and their solutions.

I will now walk you through solving a problem that we had solved during the EAPC, *Disastrous Doubling*.

Problem: EAPC17D - Disastrous Doubling

Problem

A scientist, E. Collie, is going to do some experiments with bacteria. Right now, she has one bacterium. She already knows that this species of bacteria doubles itself every hour. Hence, after one hour there will be 2 bacteria. E. Collie will do one experiment every hour, for n consecutive hours. She starts the first experiment exactly one hour after the first bacterium starts growing. In experiment i she will need b_i bacteria. How many bacteria will be left directly after starting the last experiment? If at any point there are not enough bacteria to do the experiment, print `error`. Since the answer may be very large, please print it modulo $10^9 + 7$.

Input

The input consist of two lines.

- The first line contains an integer1 $\leq n \leq 10^5$, the number of experiments.
- The second line contains n integers $b_1, \cdot, b_n$ where $0 \leq b_i \leq 2^{60}$ is the number of bacteria used in the $i$the experiment.

Output

Output a single line containing the number of bacteria that remains after doing all the experiments, or `error`.

Sample Input 1	Sample Output 1
3 0 0 0	8

Sample Input 2	Sample Output 2
5 1 1 1 1 1	1

Sample Input 3	Sample Output 3
5 0 2 2 4 0	0

Sample Input 4	Sample Output 4
5 0 2 2 4 1	error

Solution: EAPC17D - Disastrous Doubling

One thing to notice is that the numbers become very, very large. In fact, they become so large that they won't fit in a long (Java). The proposed solution was that one needed to calculate in modulo, however, Java has a BigInteger class which *can* hold numbers this big.

```
Procedure getBacteriaAmount()
    Let $n \leftarrow$ number of experiments
    Let $amOfBact \leftarrow 1$
    for $i = 0$ to $n$ do
        Subtract the next integer from amOfBact
        if $amOfBact < 0$ then
            return "error"
        end
        Multiply amOfBact by 2.
    end
    return amOfBact modulo $10^9 + 7$
```

Java code has been added as appendix.

3.3 AI: Battlecode

The Battlecode competition was a seed-based tournament, taking place from the 8th of January 2018 to the 24th of January 2018. I participated together with Koen Degelingen, Rowin Versteeg and Storm de Zeeuw. Our teamname was `Dako RowStorm`.

We participated in Battlecode which is a contest revolving around creating a working AI for a specific game made by them with a specific ruleset and then battling your AI with other group their AIs. The game itself was a turn based strategy game in which the goal was the build a spaceship to Mars since Earth was going to be destroyed. Each game consisted of two teams, a red team and a blue team who had to compete against each other. In the game you could collect resources to build units, factories or to research certain abilities. The units were either workers (who gather resources), or fighting units (who attack the enemy). Factories could produce units.

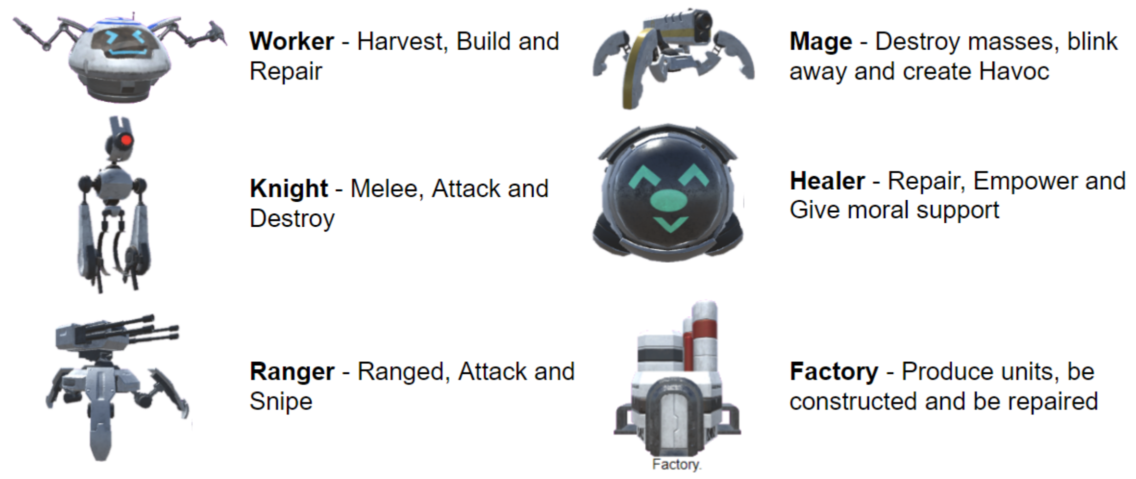

Worker - Harvest, Build and Repair
Mage - Destroy masses, blink away and create Havoc
Knight - Melee, Attack and Destroy
Healer - Repair, Empower and Give moral support
Ranger - Ranged, Attack and Snipe
Factory - Produce units, be constructed and be repaired
Factory

Figure 3.1: The produce-able units from the Battlecode game.

To develop a rocket to go to Mars you first had to research the ability to build the rocket and then actually build the rocket. Below is an 3D representation of the game. Both teams are fighting on Earth, you can see this because of the green ground. The red team has focused on attacking and has conquered one half of Earth and is trying to overtake blue. The blue team on the other hand has focused on defending by creating a lot of factories, but will probably lose due to its lack of fighting units.

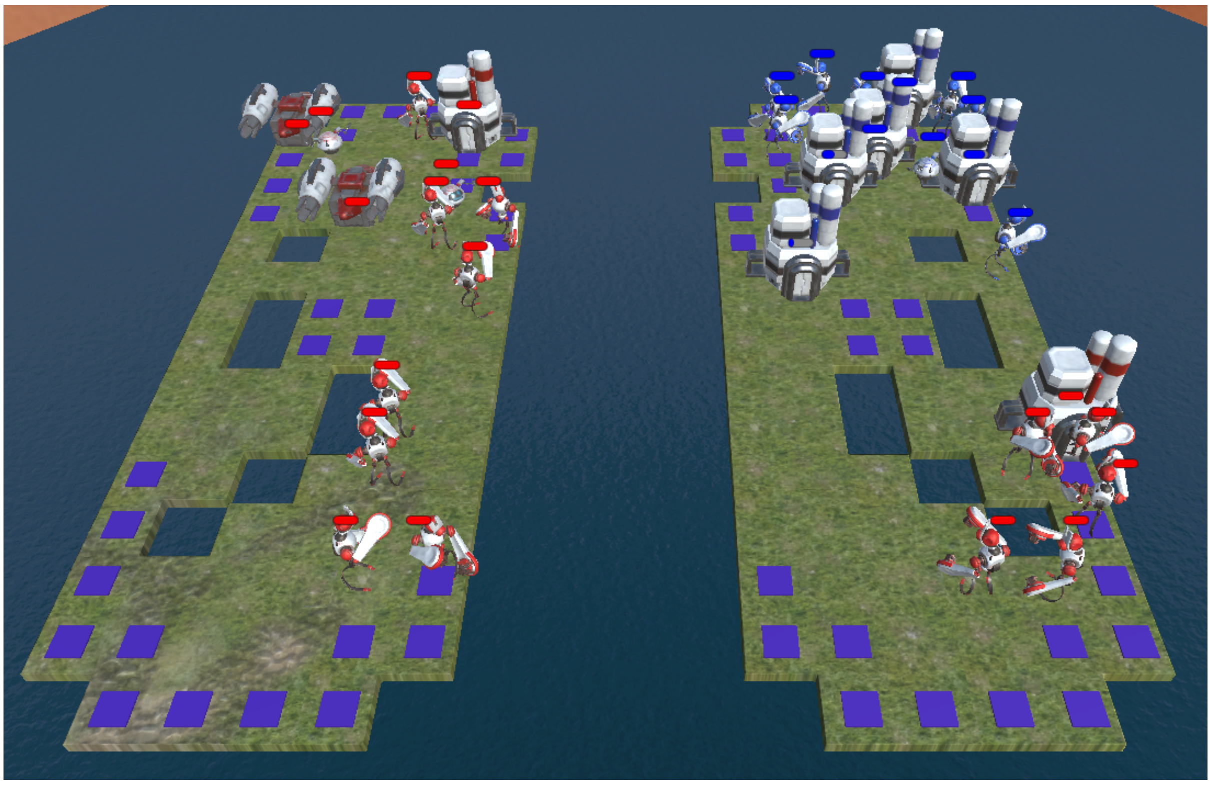


Figure 3.2: A visualisation of the Battlecode game.

The contest lasted about 2 weeks, in which you could create and improve your AI. The goal is to train against other teams in preparation for the big competition at the end. AI and games, of course sounded interesting to the group members. It is also organised by MIT which is a highly respected institution in the STEM science fields so the lectures and information provided by them can be seen as being of high quality so we were excited to start. We ended up having 2 meetings, in the first we just wanted to go through the general rules, we watched some video lectures provided by MIT and get everyone on the same level of understanding. In the second meeting we installed everything and got to work on a general strategy, which was a rush strategy: to get to Mars as quickly as possible without focusing on the enemy.

The contest itself went very wrong, the competition was planned during the exam weeks of our university, so no one really had time to meet or work on the AI. In the end we ended up coding a general framework and got the basics of our first strategy in code to a certain degree. We learned that motivation and getting motivated by team members is really important in long lasting projects / contests. Without contact no one will feel the need to do anything. Also, meeting in person is way better than meeting online.

For next time we should have either more meetings or we should not participate if we know we will not have the time and energy to perform well. If we are not motivated for the contest then participating is of no use.

3.4 Engineering: Google Hashcode

Google Hashcode took place on the 1st of March 2018. I planned to participate together with Rowin Versteeg, Bogdan Enache and Storm de Zeeuw, but unfortunately I ended up being sick and missing the contest. The teamname was *The Weeaboys*. I am really unhappy with the fact that I missed this contest, as it was the one I was looking forward to the most.

This year we participated in the qualification round of the Google Hash Code in which we had to solve an engineering problem as optimally as possible. It is a competition hosted by Google for students and industry professional across Europe, the Middle East and Africa. The problem presented would be an NP-hard problem, which could not be easily solved by one simple algorithm. Multiple algorithms; greedy, random, brute-force and more were needed for an estimation of the most optimal solution. We chose to do this competition as it was something we have never tried before, and it was one of the recommended contests by our track coordinator, Kevin Verbeek.

To prepare we met a week before to try out the judge system which had been put online by the Hashcode team. We had to solve a (much smaller) optimisation problem, which was about optimising the amount of pizza slices given a large pizza with specific ingredients.

It was of a great help since we did not have any experience with this kind of problems and contests before. The Hashcode worked differently than most competitions, instead of uploading code, you had to upload input and output files as a solution. We also read the document that Kevin Verbeek (our track coordinator) sent us, which contained a description of a solution for the Hashcode of last year, it also gave us a great insight of what to expect.

The competition was hosted by Gehack at the Tu/e for the hub of Eindhoven on the 1st of March. Unfortunately, Daniel was sick and was unable to participate. The problem presented was a taxi scheduling problem, coincidentally we had to solve a similar problem in the course DBL Algorithms a few weeks before that. The main idea of this problem was that given a list of n taxis and m taxi requests, decide on which taxi picks up which request (if any). This decision should take into account the distance between the current position of the taxi and the

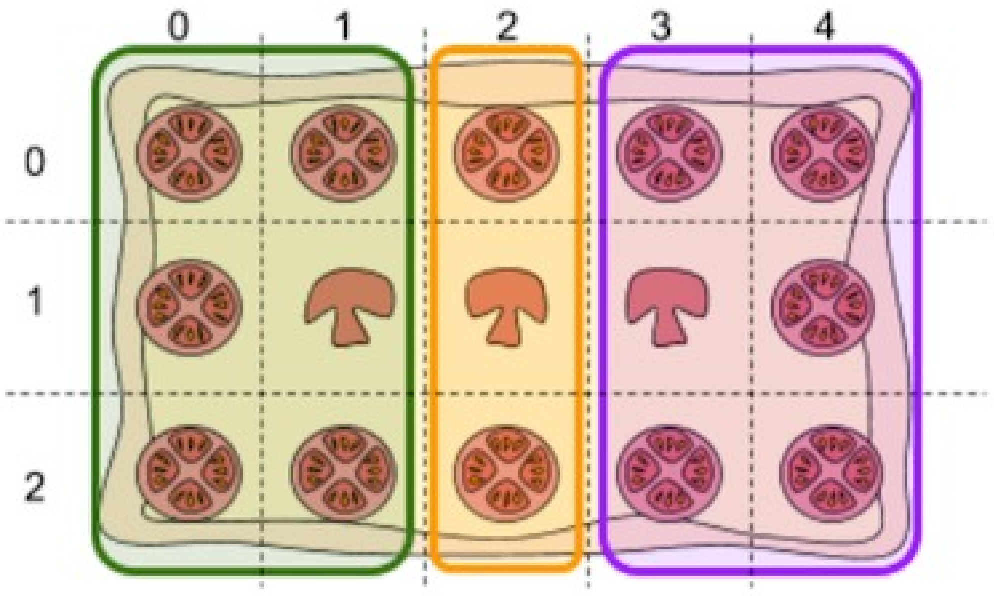


Figure 3.3: A possible slicing of a pizza for HashCode

location of the taxi request, the availability of the taxi and the request, the income generated by that request if it is fulfilled and what other requests can be fulfilled by the taxi in the same amount of time. During the contest, Rowin was implementing the basic features and brute force methods of the code, so that the code was working as fast as possible. Storm was inventing new easy to implement tricks to optimise the scheduling. Bogdan was trying to implement his own greedy algorithm, but more complex using our basic framework. As a result, our team was one of the first with working basic code (brute force) which gradually improved over time as the greedy strategy was being used. The greedy strategy had a simple but efficient idea: once a taxi is available, assign the most profitable request to it, where profitable is represented by the greatest amount of money that the respective taxi can make by taking into consideration the money gained by serving the request (determined by how long that taxi trip is) as well as the costs for moving from its current position to the location of the request (i.e. avoid going all across the town for a 2 minute taxi trip). In the end we ended up in 7th place of the hub and rank 1280 of the world, which is pretty good for participating for the first time in this contest.

We learnt that it is better to work on different strategies and merge them together, as working on one specific strategy may limit your view and leave you stuck with a sub-optimal solution. Teamwork is very important for this, you need to communicate the changes you made and try to combine these, for this you should use the same framework which was implemented at the start so that combining later takes no time and effort.

4 Seminar

As hinted at in the section about the organisation of the track, students are required to give a seminar on a specific topic. In this context, giving a seminar means giving a mini-lecture, explaining your topic. My topic was *Subset DP*. In order to explain what subset DP is, one first needs to fully understand *DP - Dynamic Programming*.

Dynamic Programming is a way to solve certain types of optimisation problems. The classroom example of a DP problem is the 0/1 Knapsack. In this problem, you have a knapsack which can hold at most a total of weight W. There are some items, which have a value p_i and a weight w_i. The goal of solving a 0/1 Knapsack is to figure out how much value you can take with you. An initial solution would be to try every single combination of items until you find the best one possible, but that takes a lot of time. By using DP, you can still try every single combination, but you do so in a clever way. Before I explain how to solve the Knapsack problem, one should note that in order to use DP on a problem, the problem needs the property that it can be split into subproblems (within the knapsack case you can define these subproblems in terms of items and weight used). In order to split into subproblems, it needs the optimal substructure property. Now, onto knapsack. The way we solve the problem with dynamic programming is as follows. You define a subproblem in terms of the input (W and the items). You keep track a table consisting of all subproblems. You figure out the base cases (if there are no items or the maximum allowed weight is 0, then you cannot take anything) and fill these in your table. Then, you find a recurrence and with the recurrence fill in the rest of the table (you either take item i, and add its value and weight, or you don't). In pseudocode, this all becomes:

Algorithm 4: `DP()`

Input: Maximum weight W and n items with weights and values.

Output: The maximum value you can take with you in the knapsack.

```
Procedure DP()
    $n \leftarrow$ total amount of items
    $T \leftarrow$ a table of dimensions $n + 1 \times W + 1$
    for All items i to n do
        for All weights w to W do
            if i == 0 || w == 0 then
                $T[i][w] \leftarrow 0$
            else if weight of item $i - 1 \leq w$ then
                $T[i][w] \leftarrow \max(T[i-1][w - w_i] + p_i, T[i-1][w])$
            else
                $T[i][w] \leftarrow T[i-1][w])$
            end
        end
    end
```

Subset DP is in essence the same thing, but one uses all subsets of some set when defining subproblems.

To prepare for the seminar, I did research on what kind of subset DP problems there were. Then I read up on the specifics for those problems, so that I understood them completely. Afterwards, I spent a lot of time on slides, in the hope that they would be clear and usable as standalone. I practised by seminar a few times by myself, and once with a friend, who pointed out some obvious though not yet noticed by me mistakes and missing information, which I added in later.

I believe that the talk itself went pretty well. I was able to answer all questions that I got after the seminar. The only comment I got was that I didn't really give an overview of the topic, but that was a conscious choice made by me.

By giving this seminar, I learned that I *can* actually give presentations, as long as I put a lot of time into the presentation by myself. I need to know all the slides pretty much by hard, and understand what I have to talk about. But, once I know the slides and know the topic very well, I believe I can explain it adequately.

Of course, I also learned how to solve problems with subset DP.

For the next time, I should practise even more. I ran out of time during my talk, which I literally said. This is not done. I should have just continued talking at a normal pace, just skip certain slides without mentioning, and get to the point.

List of Figures

A Written Code

A.1 Code for BAPC10D - Collatz

Java:

```java
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;

public class bapc10d {

    public static void main(String[] args) throws IOException {
        BufferedReader r = new BufferedReader(new InputStreamReader(System.
    in));
        BufferedWriter w = new BufferedWriter(new OutputStreamWriter(System
    .out));

        int cases = Integer.parseInt(r.readLine());

        while(cases-->0) {
            double N = Double.parseDouble(r.readLine());
            int k = (int) (2 * Math.ceil(N / 2) - Math.ceil( Math.floor( (N
    -1)/3) / 2));
            w.write(k + "\n");
        }
        w.flush();
    }
}
```

Python:

```python
import math as m

cases = int(input())

while cases > 0:
    N = int(input())
    print(2 * m.ceil(N / 2)  - m.ceil( m.floor((N - 1) / 3) / 2 ))
    cases -= 1
```

A.2 Code for BAPC10I - Keylogger

Java:

```java
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.util.LinkedList;

/**
```

```java
 *
 * @author s165839
 */
public class keylogger {
  void run() throws NumberFormatException, IOException {
    BufferedReader r = new BufferedReader(new InputStreamReader(System.in))
   ;
    BufferedWriter w = new BufferedWriter(new OutputStreamWriter(System.out
   ));
    int cases = Integer.parseInt(r.readLine());
    while (cases-- > 0) {
      char[] in = r.readLine().toCharArray();
      LinkedList<Character> out = new LinkedList<Character>();
      StringBuilder sb = new StringBuilder("");
      // keep a counter for the index of the password
      int index = 0;
      for (int i = 0; i < in.length; i++) {
        // for every character, make a decision
        switch (in[i]) {
        case '-':
          if (index > 0) {
            out.remove(--index);
          }
          break;
        case '>':
          if (index < out.size()) {
            index++;
          }
          break;
        case '<':
          if (index > 0) {
            index--;
          }
          break;
        default: {
          out.add(index, in[i]);
          index++;
        }
        }
      }
      // for every character in the output
      for (Character c : out) {
        // add to string
        sb.append(c);
      }
      sb.append("\n");
      w.write(sb.toString());
      w.flush();
    }
  }

  public static void main(String[] args) throws NumberFormatException,
   IOException {
    new keylogger().run();
  }
}
```

A.3 Code for BAPC12E - Encoded Message

Java:

```java
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;

/**
 *
 * @author s165839
 */
public class beapc12e {

  void run() throws NumberFormatException, IOException {
    BufferedReader r = new BufferedReader(new InputStreamReader(System.in))
    ;
    BufferedWriter w = new BufferedWriter(new OutputStreamWriter (System.
    out));

    int cases = Integer.parseInt(r.readLine());
    // for every case
    while(cases-->0) {
      // process the word
      char[] in = r.readLine().toCharArray();
      String out = "";
      int length = in.length;
      // square root of length times 2 is the encoding square
      int squareSize = (int) Math.sqrt((double) length);
      // process horizontally and vertically
      for (int i = 0 ; i < squareSize ; i++) {
        for (int j = 0 ; j < squareSize ; j++){
          out += in[squareSize - i - 1 + j * squareSize];
        }
      }
      // write the word
      w.write(out + "\n");
    }
    w.flush();
  }




    public static void main(String[] args) throws NumberFormatException,
    IOException {
        new beapc12e().run();

    }
}
```

A.4 Code for CD

Java:

```java
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.util.HashSet;
import java.util.Set;
import java.util.StringTokenizer;

public class CD {
  // use buffered reader and writer
  BufferedReader r = new BufferedReader(new InputStreamReader(System.in));
  BufferedWriter w = new BufferedWriter(new OutputStreamWriter(System.out))
    ;

  void run() throws IOException {
    while (true) {
      // StringTokenizer : gives all tokens one by one
      StringTokenizer s = new StringTokenizer(r.readLine());
      // Number CD from person 1
      int n = Integer.parseInt(s.nextToken());
      // Number CD from person 2
      int m = Integer.parseInt(s.nextToken());
      // check if we're at the end of the test cases
      if (n == 0 && m == 0) {
        break;
      }
      // create three sets to keep track of the numbers
      Set<Integer> setOne = new HashSet<>();
      Set<Integer> bothSets = new HashSet<>();
      Set<Integer> setTwo = new HashSet<>();
      // add for the first set
      for (int i = 0; i < n; i++) {
        int cd = Integer.parseInt(r.readLine());
        setOne.add(cd);
        bothSets.add(cd);
      }
      // add for the second set
      for (int i = 0; i < m; i++) {
        int cd = Integer.parseInt(r.readLine());
        setTwo.add(cd);
        bothSets.add(cd);
      }
      // write the answer
      w.write(setOne.size() + setTwo.size() - bothSets.size() + "\n");
    }
    // needed for bufferedwriter
    w.flush();
  }

  public static void main(String[] a) throws IOException {
    (new CD()).run();
  }
}
```

A.5 Code for Help Me With The Game

Java:

```
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.util.ArrayList;
import java.util.Arrays;

public class helpme {
  // We create a list of the pieces we have
  public ArrayList<Piece> pieces = new ArrayList<Piece>();

  /*
   * Removes the last character of a string.
   */
  private static String removeLastChar(final String str) {
    return str.substring(0, str.length() - 1);
  }

  /*
   * Reads the input, and saves any pieces you encounter in the list of
    pieces
   */
  private void readInput() throws IOException {
    // initialize a buffered reader to read the input
    BufferedReader r = new BufferedReader(new InputStreamReader(System.in))
   ;
    // skip first line
    r.readLine();
    // for every line on the board
    for (int i = 8; i >= 1; i--) {
      // read a line
      char[] l = r.readLine().toCharArray();
      // for every character
      for (int c = 0; c < l.length; c++) {
        // if it is a letter
        if (Character.isLetter(l[c])) {
          Piece p = new Piece(l[c], i, c);
          // add said piece to all pieces
          pieces.add(p);
        }
      }
      // skip a line
      r.readLine();
    }
  }

  /*
   * Writes the output.
   */
  private void writeOutput() throws IOException {
    // initialize a buffered writer to write the output
    BufferedWriter w = new BufferedWriter(new OutputStreamWriter(System.out
   ));

    // create two strings for each colour
    String white = "White: ";
    String black = "Black: ";

    // create a list of all pieces, in sorted order
    ArrayList<Piece> sortedPieces = Piece.sort(pieces);

    // for every single piece
    for (Piece piece : sortedPieces) {
      if (piece.isWhite()) {
        white += (piece.toString() + ",");
      } else {
        black += (piece.toString() + ",");
      }
    }

    // remove the last character from the string; this is a ','.
    white = removeLastChar(white);
    black = removeLastChar(black);

    // write both strings with a newline
    w.write(white + "\n");
    w.write(black + "\n");

    // flush your buffered writer
    w.flush();
  }

  /*
   * Runs the program
   */
  void run() throws IOException {
    readInput();
    writeOutput();
  }

  /*
   * Main method
   */
  public static void main(String[] a) throws IOException {
    (new helpme()).run();
  }
}

/*
 * This is a class for a piece on the board. This piece has a name (e.g. R
    for
 * rook), a rownumber, a columnnumber, a colour and it is either a pawn or
    no
 * pawn.
 */
class Piece {

```

```java
  char name;
  int row;
  int col;
  boolean colour;
  boolean pawn;

  /*
   * Constructor of a piece. With this one can create new pieces.
   */
  Piece(char name, int row, int col) {
    this.name = name;
    this.row = row;
    this.col = col;
    colour = Character.isLowerCase(name) ? false : true;
    pawn = ((name == 'p') || (name == 'P')) ? true : false;
  }

  /*
   * Returns a list sorted on index of the pieces for white pieces
   */
  private static ArrayList<Piece> whiteIndexSort(final ArrayList<Piece>
    list) {
    // create a list to return on Index
    ArrayList<Piece> onIndex = new ArrayList<Piece>();

    // create an array with rowIndices
    int[] rowIndices = new int[list.size()];
    int i = 0;
    for (Piece p : list) {
      rowIndices[i] = p.row();
      i++;
    }
    // Sort on the indexes from LOW to HIGH
    Arrays.sort(rowIndices);
    // for every row index,add the corresponding piece to the list on index
    .
    for (int ind : rowIndices) {
      for (Piece p : list) {
        if (ind == p.row()) {
          if (!onIndex.contains(p)) {
            onIndex.add(p);
          }
        }
      }
    }
    // then return said list
    return onIndex;
  }

  /*
   * Returns a list sorted on index of the pieces for black pieces
   */
  private static ArrayList<Piece> blackIndexSort(final ArrayList<Piece>
    list) {
    // create a list to return on Index
    ArrayList<Piece> onIndex = new ArrayList<Piece>();

    // create an array with rowIndices
    int[] rowIndices = new int[list.size()];
    int i = 0;
    for (Piece p : list) {
      rowIndices[i] = p.row();
      i++;
    }
    // System.err.print("Elements: " + p.toString() + "\n");

    // Sort on the indexes from HIGH to LOW
    Arrays.sort(rowIndices);
    // for every row index,add the corresponding piece to the list on index
  .
    for (int ind = rowIndices.length - 1; ind >= 0; ind--) {
      for (Piece p : list) {
        if (rowIndices[ind] == p.row()) {
          if (!onIndex.contains(p)) {
            onIndex.add(p);
          }
        }
      }
    }
    // then return said list
    return onIndex;
  }

  /*
   * Sorts pieces as wanted by the question
   */
  public static ArrayList<Piece> sort(ArrayList<Piece> pieces) {
    // A semi-sorted list to return
    ArrayList<Piece> sorted = new ArrayList<Piece>();

    // Create sublists
    ArrayList<Piece> whiteKings = new ArrayList<Piece>();
    ArrayList<Piece> whiteQueens = new ArrayList<Piece>();
    ArrayList<Piece> whiteRooks = new ArrayList<Piece>();
    ArrayList<Piece> whiteBishops = new ArrayList<Piece>();
    ArrayList<Piece> whiteKnights = new ArrayList<Piece>();
    ArrayList<Piece> whitePawns = new ArrayList<Piece>();
    ArrayList<Piece> blackKings = new ArrayList<Piece>();
    ArrayList<Piece> blackQueens = new ArrayList<Piece>();
    ArrayList<Piece> blackRooks = new ArrayList<Piece>();
    ArrayList<Piece> blackBishops = new ArrayList<Piece>();
    ArrayList<Piece> blackKnights = new ArrayList<Piece>();
    ArrayList<Piece> blackPawns = new ArrayList<Piece>();

    // First add the piece to their lists (also add Kings to the sorted
    // list, they are first)
    for (Piece piece : pieces) {
      switch (piece.name) {
      case 'K':
        whiteKings.add(piece);
      case 'Q':
        whiteQueens.add(piece);
      case 'R':
        whiteRooks.add(piece);
      case 'B':
        whiteBishops.add(piece);
      case 'N':
        whiteKnights.add(piece);
      case 'P':
        whitePawns.add(piece);
      case 'k':
        blackKings.add(piece);
      case 'q':
        blackQueens.add(piece);
      case 'r':
        blackRooks.add(piece);
      case 'b':
        blackBishops.add(piece);
      case 'n':
        blackKnights.add(piece);
      case 'p':
        blackPawns.add(piece);
      }
    }

    // Sort all sublists
    whiteKings = whiteIndexSort(whiteKings);
    for (Piece whiteKing : whiteKings) {
      if (!sorted.contains(whiteKing)) {
        sorted.add(whiteKing);
      }
    }

    whiteQueens = whiteIndexSort(whiteQueens);
    for (Piece whiteQueen : whiteQueens) {
      if (!sorted.contains(whiteQueen)) {
        sorted.add(whiteQueen);
      }
    }

    whiteRooks = whiteIndexSort(whiteRooks);
    for (Piece whiteRook : whiteRooks) {
      if (!sorted.contains(whiteRook)) {
        sorted.add(whiteRook);
      }
    }

    whiteBishops = whiteIndexSort(whiteBishops);
    for (Piece whiteBishop : whiteBishops) {
      if (!sorted.contains(whiteBishop)) {
        sorted.add(whiteBishop);
      }
    }

    whiteKnights = whiteIndexSort(whiteKnights);
    for (Piece whiteKnight : whiteKnights) {
      if (!sorted.contains(whiteKnight)) {
        sorted.add(whiteKnight);
      }
    }

    whitePawns = whiteIndexSort(whitePawns);
    for (Piece whitePawn : whitePawns) {
      if (!sorted.contains(whitePawn)) {
```

```
                sorted.add(whitePawn);
            }
        }

        blackKings = blackIndexSort(blackKings);
        for (Piece blackKing : blackKings) {
            if (!sorted.contains(blackKing)) {
                sorted.add(blackKing);
            }
        }

        blackQueens = blackIndexSort(blackQueens);
        for (Piece blackQueen : blackQueens) {
            if (!sorted.contains(blackQueen)) {
                sorted.add(blackQueen);
            }
        }

        blackRooks = blackIndexSort(blackRooks);
        for (Piece blackRook : blackRooks) {
            if (!sorted.contains(blackRook)) {
                sorted.add(blackRook);
            }
        }

        blackBishops = blackIndexSort(blackBishops);
        for (Piece blackBishop : blackBishops) {
            if (!sorted.contains(blackBishop)) {
                sorted.add(blackBishop);
            }
        }

        blackKnights = blackIndexSort(blackKnights);
        for (Piece blackKnight : blackKnights) {
            if (!sorted.contains(blackKnight)) {
                sorted.add(blackKnight);
            }
        }

        blackPawns = blackIndexSort(blackPawns);
        for (Piece blackPawn : blackPawns) {
            if (!sorted.contains(blackPawn)) {
                sorted.add(blackPawn);
            }
        }

        return sorted;
    }

    /*
     * Returns the size of a Piece Array List
     */
    @SuppressWarnings("unused")
    private int size(final ArrayList<Piece> list) {
        return list.size();
    }

    /*
```

```
     * Returns true if a piece is a pawn
     */

    public boolean isPawn() {
      return pawn;
    }

    /*
     * Returns true if a piece is a white piece
     */
    public boolean isWhite() {
      return colour;
    }

    /*
     * Return the name of the piece
     */
    private char name() {

      return Character.toUpperCase(name);
    }

    /*
     * Return the row index of the piece
     */
    private int row() {
      return row;
    }

    /*
     * Return the col name of the piece
     */
    private char col() {
      switch (col) {
      case 2:
        return 'a';
      case 6:
        return 'b';
      case 10:
        return 'c';
      case 14:
        return 'd';
      case 18:
        return 'e';
      case 22:
        return 'f';
      case 26:
        return 'g';
      case 30:
        return 'h';

      default:
        System.err.println("Something went wrong with finding column
   information...");
      }
      // something went wrong
      return ' ';
    }
```

```

  /*
   * Overrides the toString method. We want to print out only our relevant
   * information.
   */
  @Override
  public String toString() {
    String ret = "";
    if (!isPawn()) {
      ret += name();
    }
    ret += col();
    ret += row();
    return ret;
  }
}
```

A.6 Code for Emag Eht Htiw Em Pleh

Java:

```
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.util.ArrayList;

public class emplah {
  // We create a list of the pieces we have
  public ArrayList<ReversedPiece> reversedPieces = new ArrayList<
    ReversedPiece>();

  /*
   * Reads the input, and saves any pieces you encounter in the list of
    pieces
   */
  private void readInput() throws IOException {
    BufferedReader r = new BufferedReader(new InputStreamReader(System.in))
    ;
    String[] temp = { "" };
    r.skip(7);
    String w = r.readLine();
    String[] white = (w.length()) > 1 ? w.split(",") : temp;
    r.skip(7);
    String b = r.readLine();
    String[] black = (b.length()) > 1 ? b.split(",") : temp;

    for (String whitePiece : white) {
      boolean colour = true;
      boolean pawn;
      ReversedPiece p;// = new ReversedPiece(' ', -1, -1, false, false);
      switch (whitePiece.length()) {
      case 3:
        pawn = false;
        p = new ReversedPiece(whitePiece.charAt(0), convertCol(whitePiece.charAt(1)),
            convertRow(whitePiece.charAt(2)), colour, pawn);
        reversedPieces.add(p);
        break;
      case 2:
        pawn = true;
        p = new ReversedPiece('P', convertCol(whitePiece.charAt(0)), convertRow(whitePiece.charAt(1)), colour,
            pawn);
        reversedPieces.add(p);
        break;
      default:
        break;
      }
    }

    for (String blackPiece : black) {
      boolean colour = false;
      boolean pawn;
      ReversedPiece p;// = new ReversedPiece(' ', -1, -1, false, false);
      switch (blackPiece.length()) {
      case 3:
        pawn = false;
        p = new ReversedPiece(blackPiece.charAt(0), convertCol(blackPiece.charAt(1)),
            convertRow(blackPiece.charAt(2)), colour, pawn);
        reversedPieces.add(p);
        break;
      case 2:
        pawn = true;
        p = new ReversedPiece('P', convertCol(blackPiece.charAt(0)), convertRow(blackPiece.charAt(1)), colour,
            pawn);
        reversedPieces.add(p);
        break;
      default:
        break;
      }
    }
  }

  /*
   * Writes the output.
   */
  private void writeOutput() throws IOException {
    // initialize a buffered writer to write the output
    BufferedWriter w = new BufferedWriter(new OutputStreamWriter(System.out));

    // code to build the board
    char[][] board = {
        // 0
        { '+', '-', '-', '-', '+', '-', '-', '-', '+', '-', '-', '-', '+', '-', '-', '-', '+', '-', '-', '-',
            '+', '-', '-', '-', '+', '-', '-', '-', '+', '-', '-', '-', '+' },
        // 1
        { '|', '.', '.', '.', '|', ':', ':', ':', '|', '.', '.', '.', '|', ':', ':', ':', '|', '.', '.', '.',
            '|', ':', ':', ':', '|', '.', '.', '.', '|', ':', ':', ':', '|'
    },
        // 2
        { '+', '-', '-', '-', '+', '-', '-', '-', '+', '-', '-', '-', '+', '-', '-', '-', '+', '-', '-', '-',
            '+', '-', '-', '-', '+', '-', '-', '-', '+', '-', '-', '-', '+'
    },
        // 3
        { '|', ':', ':', ':', '|', '.', '.', '.', '|', ':', ':', ':', '|', '.', '.', '.', '|', ':', ':', ':',
            '|', '.', '.', '.', '|', ':', ':', ':', '|', '.', '.', '.', '|'
    },
        // 4
        { '+', '-', '-', '-', '+', '-', '-', '-', '+', '-', '-', '-', '+', '-', '-', '-', '+', '-', '-', '-',
            '+', '-', '-', '-', '+', '-', '-', '-', '+', '-', '-', '-', '+'
    },
        // 5
        { '|', '.', '.', '.', '|', ':', ':', ':', '|', '.', '.', '.', '|', ':', ':', ':', '|', '.', '.', '.',
            '|', ':', ':', ':', '|', '.', '.', '.', '|', ':', ':', ':', '|'
    },
        // 6
        { '+', '-', '-', '-', '+', '-', '-', '-', '+', '-', '-', '-', '+', '-', '-', '-', '+', '-', '-', '-',
            '+', '-', '-', '-', '+', '-', '-', '-', '+', '-', '-', '-', '+'
    },
        // 7
        { '|', ':', ':', ':', '|', '.', '.', '.', '|', ':', ':', ':', '|', '.', '.', '.', '|', ':', ':', ':',
            '|', '.', '.', '.', '|', ':', ':', ':', '|', '.', '.', '.', '|'
    },
        // 8
        { '+', '-', '-', '-', '+', '-', '-', '-', '+', '-', '-', '-', '+', '-', '-', '-', '+', '-', '-', '-',
            '+', '-', '-', '-', '+', '-', '-', '-', '+', '-', '-', '-', '+'
    },
        // 9
        { '|', '.', '.', '.', '|', ':', ':', ':', '|', '.', '.', '.', '|', ':', ':', ':', '|', '.', '.', '.',
            '|', ':', ':', ':', '|', '.', '.', '.', '|', ':', ':', ':', '|'
    },
        // 10
        { '+', '-', '-', '-', '+', '-', '-', '-', '+', '-', '-', '-', '+', '-', '-', '-', '+', '-', '-', '-',
            '+', '-', '-', '-', '+', '-', '-', '-', '+', '-', '-', '-', '+'
    },
        // 11
        { '|', ':', ':', ':', '|', '.', '.', '.', '|', ':', ':', ':', '|', '.', '.', '.', '|', ':', ':', ':',
            '|', '.', '.', '.', '|', ':', ':', ':', '|', '.', '.', '.', '|'
    },
        // 12
        { '+', '-', '-', '-', '+', '-', '-', '-', '+', '-', '-', '-', '+', '-', '-', '-', '+', '-', '-', '-',
                '+', '-', '-', '-', '+', '-', '-', '-', '+', '-', '-', '-', '+'
      },
          // 13
          { '|', '.', '.', '.', '|', ':', ':', ':', '|', '.', '.', '.', '|',
   ':', ':', ':', '|', '.', '.', '.',
                '|', ':', ':', ':', '|', '.', '.', '.', '|', ':', ':', ':', '|'
      },
          // 14
          { '+', '-', '-', '-', '+', '-', '-', '-', '+', '-', '-', '-', '+',
   '-', '-', '-', '+', '-', '-', '-',
                '+', '-', '-', '-', '+', '-', '-', '-', '+', '-', '-', '-', '+'
      },
          // 15
          { '|', ':', ':', ':', '|', '.', '.', '.', '|', ':', ':', ':', '|',
   '.', '.', '.', '|', ':', ':', ':',
                '|', '.', '.', '.', '|', ':', ':', ':', '|', '.', '.', '.', '|'
      },
          // 16
          { '+', '-', '-', '-', '+', '-', '-', '-', '+', '-', '-', '-', '+',
   '-', '-', '-', '+', '-', '-', '-',
                '+', '-', '-', '-', '+', '-', '-', '-', '+', '-', '-', '-', '+'
      },

    };

    // Add the pieces to the board at their place
    for (ReversedPiece p : reversedPieces) {
      board[p.getCol()][p.getRow()] = p.isWhite() ? p.getName() : Character
   .toLowerCase(p.getName());
    }

    // Code to print the board that was made
    String[] boardPrint = new String[board[0].length];
    for (int i = 0; i < board.length; i++) {
      // boardPrint[i] = board[i].toString();
      boardPrint[i] = makeString(board[i]);
    }
    for (String line : boardPrint) {
      if (line != null) {
        w.write(line + "\n");
      }
    }

    // flush your buffered writer
    w.flush();
  }

  // Converts character to column index
  private int convertCol(char x) {
    switch (x) {
    case 'a':
      return 2;
    case 'b':
      return 6;
    case 'c':
      return 10;
    case 'd':
      return 14;
    case 'e':
      return 18;
    case 'f':
      return 22;
    case 'g':
      return 26;
    case 'h':
      return 30;
    }
    return -1;
  }

  // Converts character to row index
  private int convertRow(char x) {
    switch (x) {
    case '1':
      return 15;
    case '2':
      return 13;
    case '3':
      return 11;
    case '4':
      return 9;
    case '5':
      return 7;
    case '6':
      return 5;
    case '7':
      return 3;
    case '8':
      return 1;
    }
    return -1;
  }

  private String makeString(char[] carr) {
    String ret = "";
    for (char c : carr) {
      ret += c;
    }
    return ret;
  }

  /*
   * Runs the program
   */
  void run() throws IOException {
    readInput();
    writeOutput();
  }

  /*
   * Main method
   */
  public static void main(String[] a) throws IOException {
    (new emplah()).run();
  }
}
```

```java

class ReversedPiece {
  private char name;
  private int row;
  private int col;
  private boolean colour;
  private boolean pawn;

  ReversedPiece(char name, int row, int col, boolean colour, boolean pawn)
    {
    this.name = name;
    this.row = row;
    this.col = col;
    this.colour = colour;
    this.pawn = pawn;
  }

  public void setColour(boolean x) {
    this.colour = x;
  }

  public void setPawn(boolean x) {
    this.pawn = x;
  }

  public void setName(char x) {
    this.name = x;
  }

  public void setCol(int x) {
    this.col = x;
  }

  public void setRow(int x) {
    this.row = x;
  }

  public char getName() {
    return this.name;
  }

  public int getCol() {
    return this.col;
  }

  public int getRow() {
    return this.row;
  }

  public boolean getPawn() {
    return this.pawn;
  }

  public boolean isWhite() {
    return this.colour;
  }
}
```

A.7 Code for EAPC17D - Disastrous Doubling

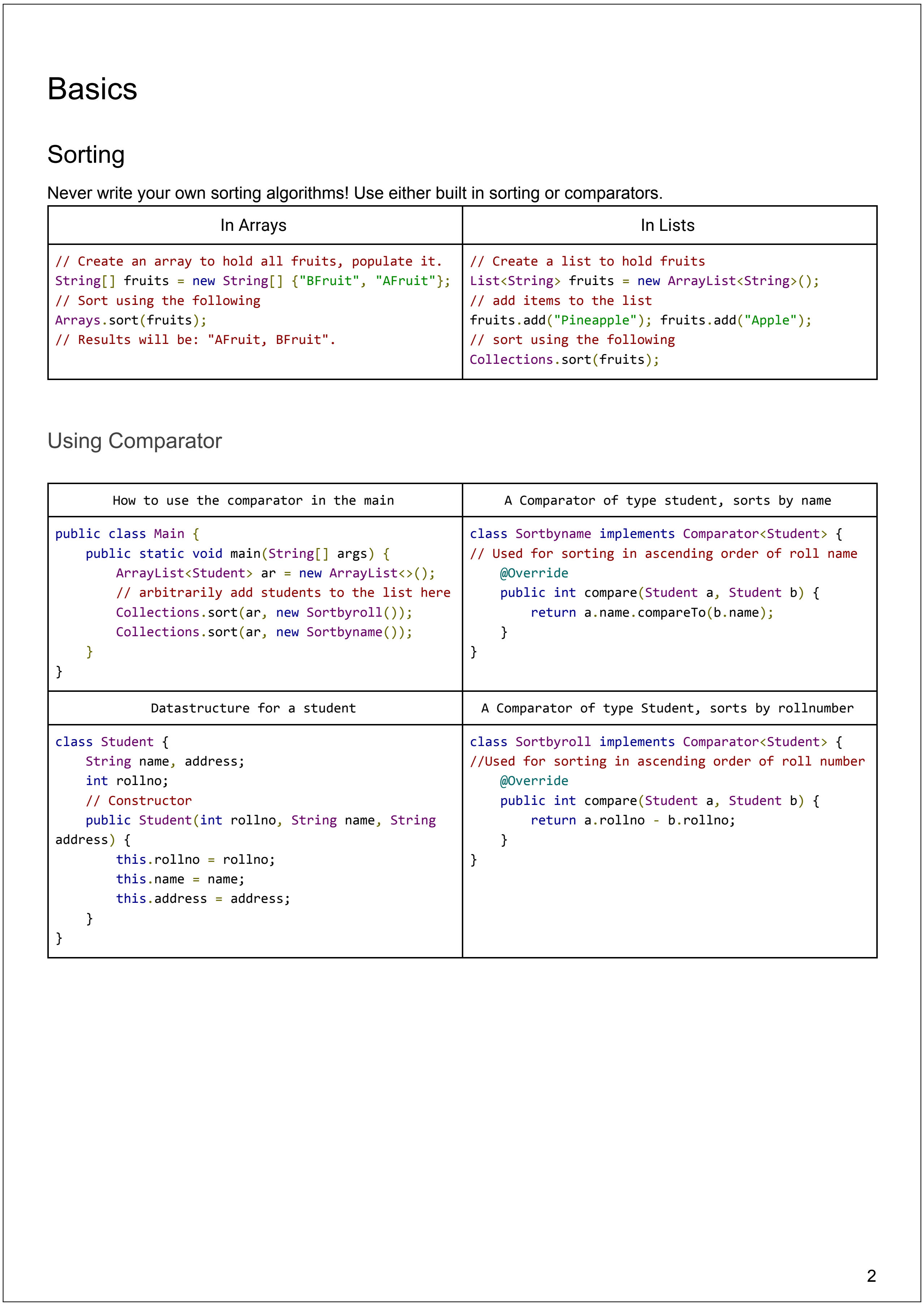
Java:

```java
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.math.BigInteger;
import java.util.StringTokenizer;

public class eapc17d {
    public static void main(String[] args) throws IOException {
        BufferedReader r = new BufferedReader(new InputStreamReader(System.
    in));
        BufferedWriter w = new BufferedWriter(new OutputStreamWriter(System
    .out));

        BigInteger amOfBact = new BigInteger("1");
        String err = "error\n";

        // Number of experiments
        int n = Integer.parseInt(r.readLine());

        // Go through each experiment
        StringTokenizer s = new StringTokenizer(r.readLine());
        for (int i = 0; i <= n; i++) {

            // s.nextToken() bacteria used for the next experiment
            amOfBact = amOfBact.subtract(new BigInteger(s.nextToken()));

            // Test if there are still enough bacteria
            if (amOfBact.compareTo(BigInteger.ZERO) == -1) {
                w.write(err);
                break;
            }

            // Multiply the bacteria by 2
            amOfBact = amOfBact.multiply(new BigInteger("2"));
        }

        w.write(amOfBact.mod(new BigInteger("1000000007")) + "\n");
        w.flush();
    }
}
```

B EAPC Cheatsheet

CheatSheet EAPC 2017 - Team $e^{i\pi} + 1 = 0$.

1



Basics

Sorting

Never write your own sorting algorithms! Use either built in sorting or comparators.

In Arrays

```java
// Create an array to hold all fruits, populate it.
String[] fruits = new String[] {"BFruit", "AFruit"};
// Sort using the following
Arrays.sort(fruits);
// Results will be: "AFruit, BFruit".
```

In Lists

```java
// Create a list to hold fruits
List<String> fruits = new ArrayList<String>();
// add items to the list
fruits.add("Pineapple"); fruits.add("Apple");
// sort using the following
Collections.sort(fruits);
```

Using Comparator

How to use the comparator in the main

```java
public class Main {
    public static void main(String[] args) {
        ArrayList<Student> ar = new ArrayList<>();
        // arbitrarily add students to the list here
        Collections.sort(ar, new Sortbyroll());
        Collections.sort(ar, new Sortbyname());
    }
}
```

A Comparator of type student, sorts by name

```java
class Sortbyname implements Comparator<Student> {
// Used for sorting in ascending order of roll name
    @Override
    public int compare(Student a, Student b) {
        return a.name.compareTo(b.name);
    }
}
```

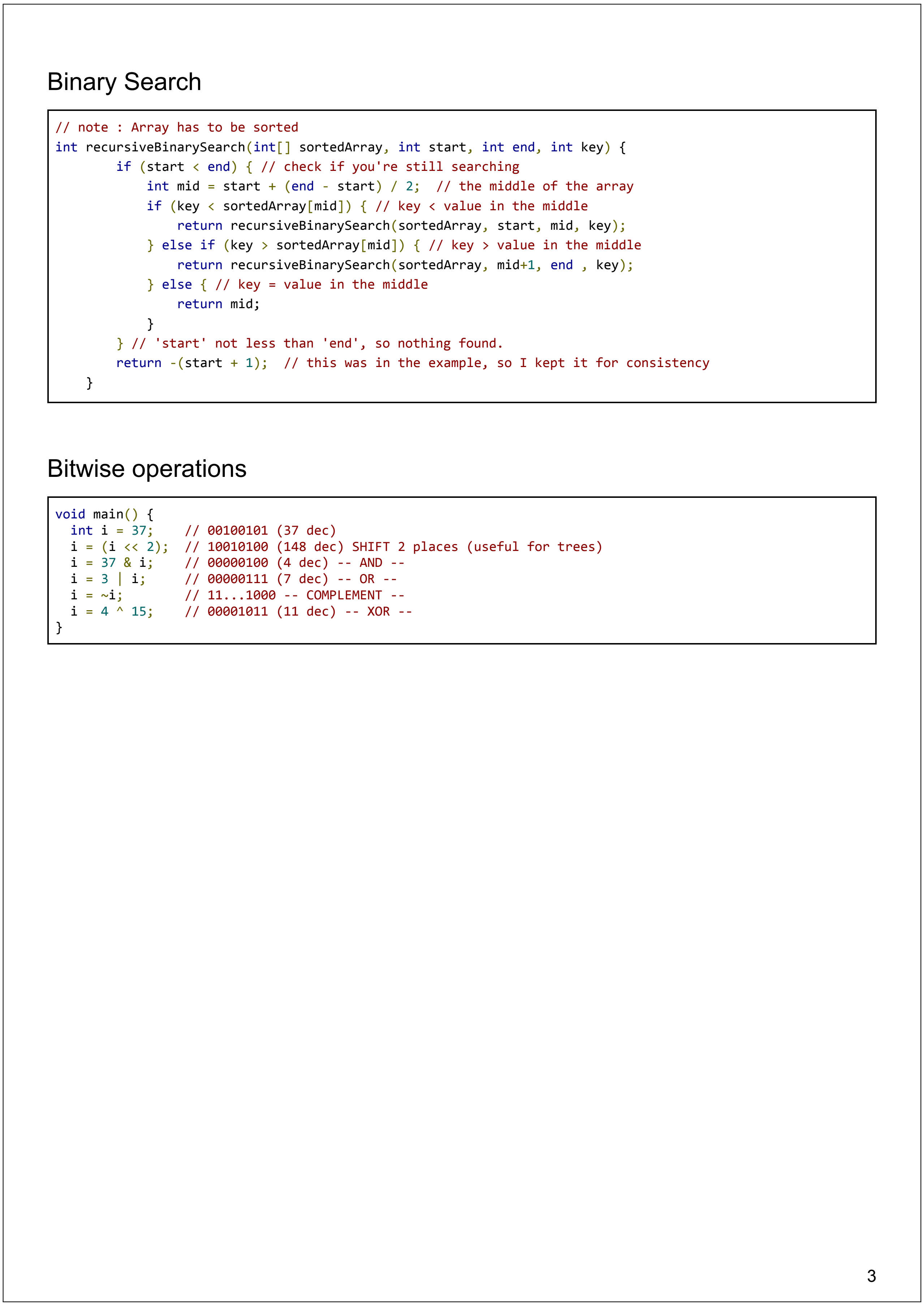
Datastructure for a student

```java
class Student {
    String name, address;
    int rollno;
    // Constructor
    public Student(int rollno, String name, String
address) {
        this.rollno = rollno;
        this.name = name;
        this.address = address;
    }
}
```

A Comparator of type Student, sorts by rollnumber

```java
class Sortbyroll implements Comparator<Student> {
//Used for sorting in ascending order of roll number
    @Override
    public int compare(Student a, Student b) {
        return a.rollno - b.rollno;
    }
}
```

2



Binary Search

```
// note : Array has to be sorted
int recursiveBinarySearch(int[] sortedArray, int start, int end, int key) {
        if (start < end) { // check if you're still searching
            int mid = start + (end - start) / 2;  // the middle of the array
            if (key < sortedArray[mid]) { // key < value in the middle
                return recursiveBinarySearch(sortedArray, start, mid, key);
            } else if (key > sortedArray[mid]) { // key > value in the middle
                return recursiveBinarySearch(sortedArray, mid+1, end , key);
            } else { // key = value in the middle
                return mid;
            }
        } // 'start' not less than 'end', so nothing found.
        return -(start + 1);  // this was in the example, so I kept it for consistency
    }
```

Bitwise operations

```
void main() {
  int i = 37;     // 00100101 (37 dec)
  i = (i << 2);   // 10010100 (148 dec) SHIFT 2 places (useful for trees)
  i = 37 & i;     // 00000100 (4 dec) -- AND --
  i = 3 | i;      // 00000111 (7 dec) -- OR --
  i = ~i;         // 11...1000 -- COMPLEMENT --
  i = 4 ^ 15;     // 00001011 (11 dec) -- XOR --
}
```

3

Number Theory

Greatest Common Divisor, Least Common Multiplier

The extended Euclidian algorithm is used in two cases, as described above the algorithms

$aX + bY = \gcd(a, b)$

```
static long euclidesX, euclidesY, gcd;
static void extendedEuclides(long a, long b) {
  long x = 0, y = 1, lastX = 1, lastY = 0, temp;
  while(b != 0) {
    long q = a / b;
    long r = a % b;

    a = b;
    b = r;

    temp = x;
    x = lastX - q * x;
    lastX = temp;

    temp = y;
    y = lastY - q * y;
    lastY = temp;
  }
  gcd = a;
  euclidesX = lastX;
  euclidesY = lastY;
}
```

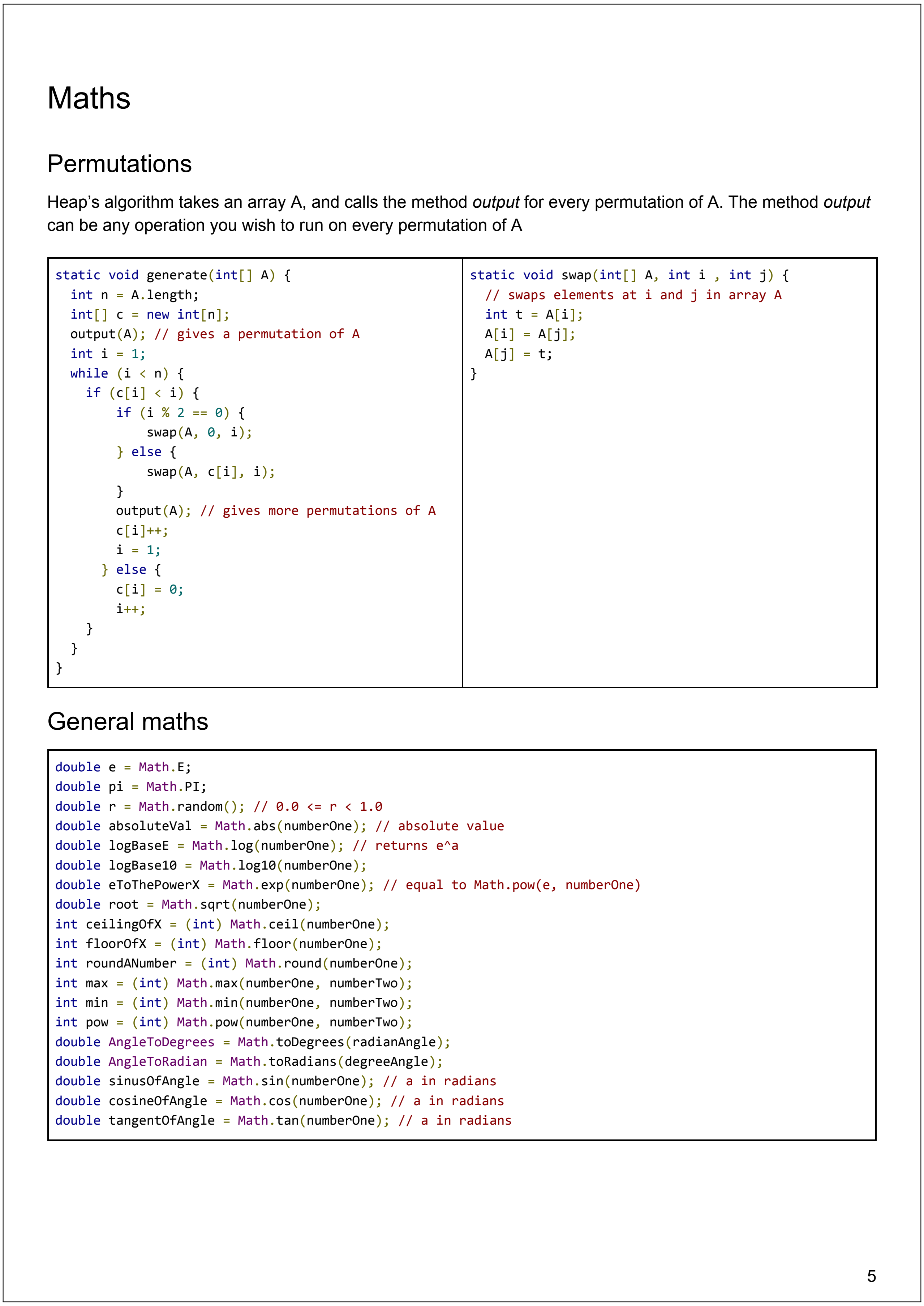
$aX + bY = c$ ONLY IF $\gcd(a, b) = 1$

```
static long euclidesX, euclidesY, gcd;
static void extendedEuclides(long a, long b, long c)
{
  extendedEuclides(a, b);
  euclidesX *= c;
  euclidesY *= c;
  long ag = a/gcd;
  long bg = b/gcd;
  if (euclidesX < 0) {
    long steps = (-euclidesX-1)/bg+1;
    euclidesX += bg*steps;
    euclidesY -= ag*steps;
  }
  if (euclidesY < 0) {
    long steps = (-euclidesY-1)/ag+1;
    euclidesX -= bg*steps;
    euclidesY += ag*steps;
  }
}
```

Sieve of Eratosthenes

The sieve of Eratosthenes finds all prime numbers up to some value n. Use it when one needs to check if a number is prime. **Runningtime: O(n(logn)(loglogn))**

```
final int n = 1000; // find it up to n
boolean[] A = new boolean[n]; // boolean array
for (int i = 0; i < n; i++) {
  A[i] = true;
}
// set all duplicates to false
for (int i = 2; i < Math.sqrt(n); i++) {
  if (A[i] == true) {
    for (int multiple = i * i; multiple < n; multiple += i) {
      A[multiple] = false;
    }
  }
} // if A[i] then i is a prime number
```



Maths

Permutations

Heap's algorithm takes an array A, and calls the method *output* for every permutation of A. The method *output* can be any operation you wish to run on every permutation of A

```java
static void generate(int[] A) {
  int n = A.length;
  int[] c = new int[n];
  output(A); // gives a permutation of A
  int i = 1;
  while (i < n) {
    if (c[i] < i) {
        if (i % 2 == 0) {
            swap(A, 0, i);
        } else {
            swap(A, c[i], i);
        }
        output(A); // gives more permutations of A
        c[i]++;
        i = 1;
      } else {
        c[i] = 0;
        i++;
    }
  }
}
```

```java
static void swap(int[] A, int i , int j) {
  // swaps elements at i and j in array A
  int t = A[i];
  A[i] = A[j];
  A[j] = t;
}
```

General maths

```java
double e = Math.E;
double pi = Math.PI;
double r = Math.random(); // 0.0 <= r < 1.0
double absoluteVal = Math.abs(numberOne); // absolute value
double logBaseE = Math.log(numberOne); // returns e^a
double logBase10 = Math.log10(numberOne);
double eToThePowerX = Math.exp(numberOne); // equal to Math.pow(e, numberOne)
double root = Math.sqrt(numberOne);
int ceilingOfX = (int) Math.ceil(numberOne);
int floorOfX = (int) Math.floor(numberOne);
int roundANumber = (int) Math.round(numberOne);
int max = (int) Math.max(numberOne, numberTwo);
int min = (int) Math.min(numberOne, numberTwo);
int pow = (int) Math.pow(numberOne, numberTwo);
double AngleToDegrees = Math.toDegrees(radianAngle);
double AngleToRadian = Math.toRadians(degreeAngle);
double sinusOfAngle = Math.sin(numberOne); // a in radians
double cosineOfAngle = Math.cos(numberOne); // a in radians
double tangentOfAngle = Math.tan(numberOne); // a in radians
```

5

Rounding to n decimal digits:

```
DecimalFormat fourDigitsRound = new DecimalFormat("#.0000");
double x = 21341241.154951345;
System.out.println(fourDigitsRound.format(x)); // returns 21341241.1550
```

Geometry Implementations

Point

```
class Point {
  double x, y;
  Point(double nx, double ny) {
    this.x = nx; this.y = ny;
  }
  Point() {}
  // implement methods as seen fit
  Point sub(Point p,   Point q) {
    return new Point (p.x - q.x, p.y - q.y);
  }
  double inp(Point p, Point q, Point O) {
   return (p.x - O.x) * (q.x - O.x) + (p.y - O.x) * (q.y - O.x);
  }
  double inp(Point p, Point q) {
    return Point.inp(p, q, new Point(0, 0));
  }
  double hat(Point p, Point q){
    return p.x * q.y - p.y * q.x;
  }
  double cross(Point A, Point B, Point O) {
    return (A.x - O.x) * (B.y - O.y) - (A.y - O.y) * (B.x - O.x);
  }
  double cross(Point A, Point B) {
    return cross(A, B, new Point(0, 0));
  }
}
```

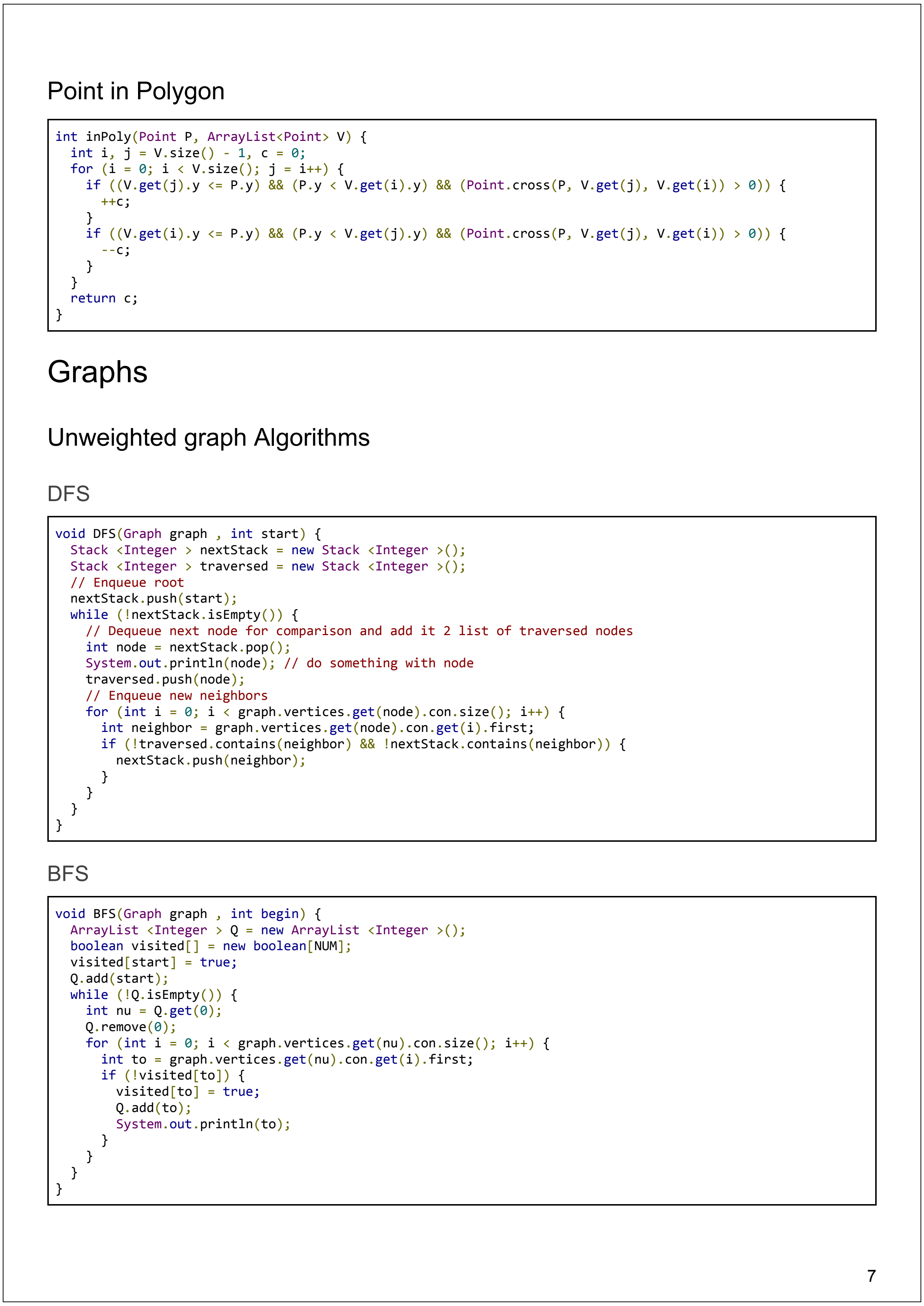
Line / Circle

```
class Line {
  Point p1, p2;
  Line(double p1x, double p1y,
double p2x, double p2y)
  {
    p1 = new Point(p1x, p1y);
    p2 = new Point(p2x, p2y);
  }
  Line() {
    p1 = new Point(0, 0);
    p2 = new Point(0, 0);
  }
}

class Circle {
  Point center;
  double radius;
  Circle(double cx, double cy,
double r)
  {
    center = new Point(cx, cy);
    radius = r;
  }
  Circle() {
    center = new Point(0, 0);
    radius = 1;
  }
}
```

Distance between point and line:

```
double dist(Point p, Line l, boolean line) {
  Point q1, q2;
  q1 = l.p1;
  q2 = l.p2;
  if (line) {
    if (Point.inp(Point.sub(p, q1), Point.sub(q2, q1)) < 0) {
      return Math.sqrt(Point.inp(Point.sub(p, q1), Point.sub(p, q1)));
    }
    if (Point.inp(Point.sub(p, q2), Point.sub(q1, q2)) < 0) {
      return Math.sqrt(Point.inp(Point.sub(p, q2), Point.sub(p, q2)));
    }
  }
  return Math.abs(Point.hat(Point.sub(p, q1), Point.sub(q2, q1)))
               / Math.sqrt(Point.inp(Point.sub(q2, q1), Point.sub(q2, q1)));
  }
double dist(Point p, Line l) {
  return dist(p, l, false);
}
```

6



Point in Polygon

```
int inPoly(Point P, ArrayList<Point> V) {
  int i, j = V.size() - 1, c = 0;
  for (i = 0; i < V.size(); j = i++) {
    if ((V.get(j).y <= P.y) && (P.y < V.get(i).y) && (Point.cross(P, V.get(j), V.get(i)) > 0)) {
      ++c;
    }
    if ((V.get(i).y <= P.y) && (P.y < V.get(j).y) && (Point.cross(P, V.get(j), V.get(i)) > 0)) {
      --c;
    }
  }
  return c;
}
```

Graphs

Unweighted graph Algorithms

DFS

```
void DFS(Graph graph , int start) {
  Stack <Integer > nextStack = new Stack <Integer >();
  Stack <Integer > traversed = new Stack <Integer >();
  // Enqueue root
  nextStack.push(start);
  while (!nextStack.isEmpty()) {
    // Dequeue next node for comparison and add it 2 list of traversed nodes
    int node = nextStack.pop();
    System.out.println(node); // do something with node
    traversed.push(node);
    // Enqueue new neighbors
    for (int i = 0; i < graph.vertices.get(node).con.size(); i++) {
      int neighbor = graph.vertices.get(node).con.get(i).first;
      if (!traversed.contains(neighbor) && !nextStack.contains(neighbor)) {
        nextStack.push(neighbor);
      }
    }
  }
}
```

BFS

```
void BFS(Graph graph , int begin) {
  ArrayList <Integer > Q = new ArrayList <Integer >();
  boolean visited[] = new boolean[NUM];
  visited[start] = true;
  Q.add(start);
  while (!Q.isEmpty()) {
    int nu = Q.get(0);
    Q.remove(0);
    for (int i = 0; i < graph.vertices.get(nu).con.size(); i++) {
      int to = graph.vertices.get(nu).con.get(i).first;
      if (!visited[to]) {
        visited[to] = true;
        Q.add(to);
        System.out.println(to);
      }
    }
  }
}
```

7

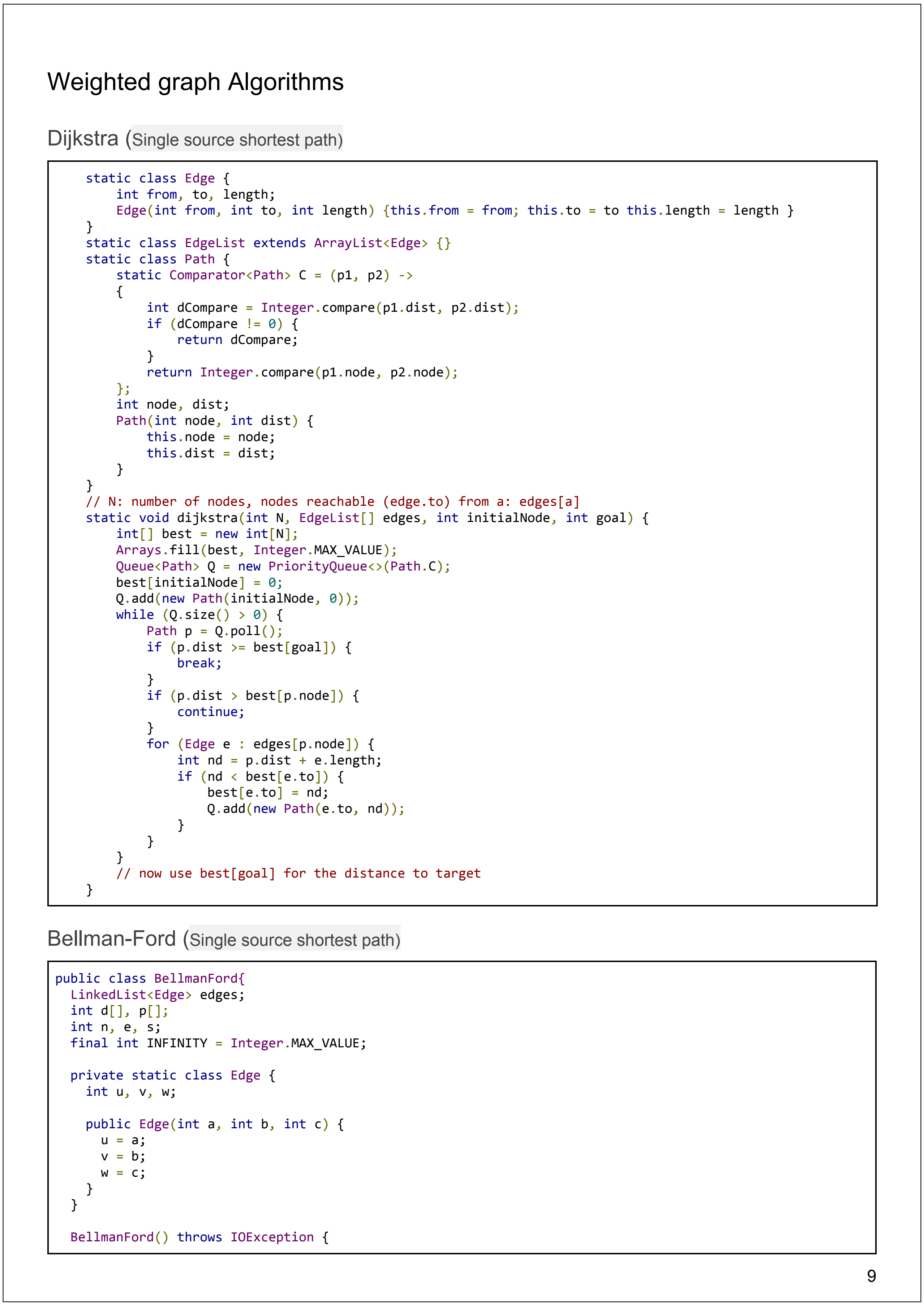
Tarjan's Algorithm

```
//algorithm for searching all strongly connected components in a graph
int index = 0;
ArrayList <Node> stack = new ArrayList <Node >();
ArrayList <ArrayList <Node>> SCC = new ArrayList <ArrayList <Node >>();
ArrayList <ArrayList <Node>> tarjan(Node v, AdjacencyList list){
  v.index = index;
  v.lowlink = index;
  index++;
  stack.add(0, v);
  for(Edge e : list.getAdjacent(v)){
    Node n = e.to;
    if(n.index == -1){
      tarjan(n, list);
      v.lowlink = Math.min(v.lowlink , n.lowlink);
    } else if(stack.contains(n)){
      v.lowlink = Math.min(v.lowlink , n.index);
    }
  }
  if(v.lowlink == v.index){
    Node n;
    ArrayList <Node> component = new ArrayList <Node >();
    do {
      n = stack.remove(0);
      component.add(n);
    } while(n != v);
    SCC.add(component);
  }
  return SCC;
}
```

Topological Sorting (cycle detection)

```
static int N; // input: number of nodes
static IntegerList[] edges; // input: edge nodes from a are in edges[a]
static List<Integer> L; // output: ordered list of nodes
static int[] marked; // used by topological sort

static boolean topologicalSort() { // returns false if graph has cycle
  L = new ArrayList<>(N);
  marked = new int[N];
  for (int i = 0; i < N; i++) {
    if (marked[i] == 0) {
      if (!visit(i)) {
        return false;
      }
    }
  }
  return true;
}
static boolean visit(int n) {
  if (marked[n] == 1) {
    return false;
  }
  if (marked[n] == 0) {
    marked[n] = 1;
    for (int m : edges[n]) {
      if (!visit(m)) {
        return false;
      }
    }
    marked[n] = 2;
    L.add(n);
  }
  return true;
}
```

8



Weighted graph Algorithms

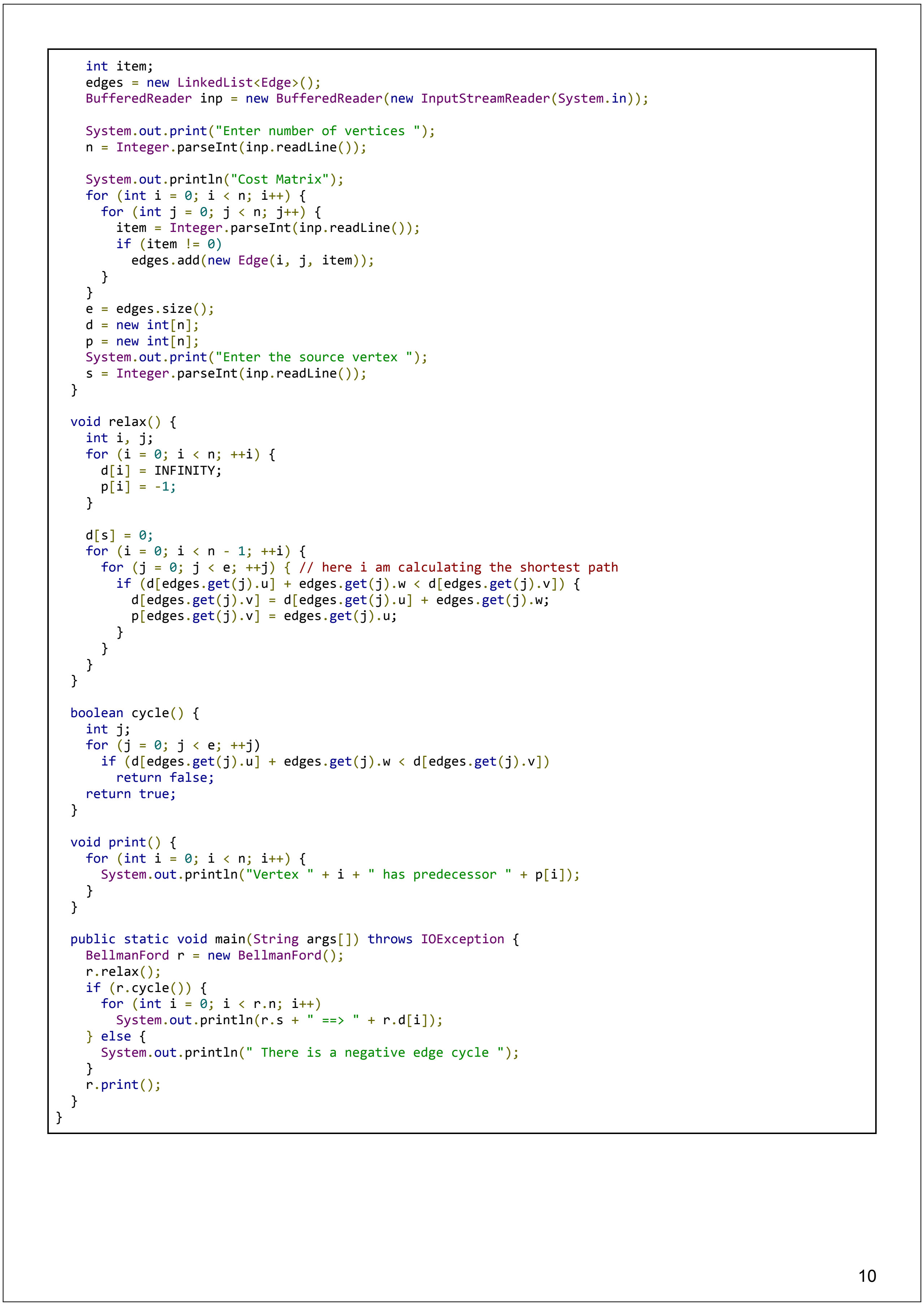
Dijkstra (Single source shortest path)

```
    static class Edge {
        int from, to, length;
        Edge(int from, int to, int length) {this.from = from; this.to = to this.length = length }
    }
    static class EdgeList extends ArrayList<Edge> {}
    static class Path {
        static Comparator<Path> C = (p1, p2) ->
        {
            int dCompare = Integer.compare(p1.dist, p2.dist);
            if (dCompare != 0) {
                return dCompare;
            }
            return Integer.compare(p1.node, p2.node);
        };
        int node, dist;
        Path(int node, int dist) {
            this.node = node;
            this.dist = dist;
        }
    }
    // N: number of nodes, nodes reachable (edge.to) from a: edges[a]
    static void dijkstra(int N, EdgeList[] edges, int initialNode, int goal) {
        int[] best = new int[N];
        Arrays.fill(best, Integer.MAX_VALUE);
        Queue<Path> Q = new PriorityQueue<>(Path.C);
        best[initialNode] = 0;
        Q.add(new Path(initialNode, 0));
        while (Q.size() > 0) {
            Path p = Q.poll();
            if (p.dist >= best[goal]) {
                break;
            }
            if (p.dist > best[p.node]) {
                continue;
            }
            for (Edge e : edges[p.node]) {
                int nd = p.dist + e.length;
                if (nd < best[e.to]) {
                    best[e.to] = nd;
                    Q.add(new Path(e.to, nd));
                }
            }
        }
        // now use best[goal] for the distance to target
    }
```

Bellman-Ford (Single source shortest path)

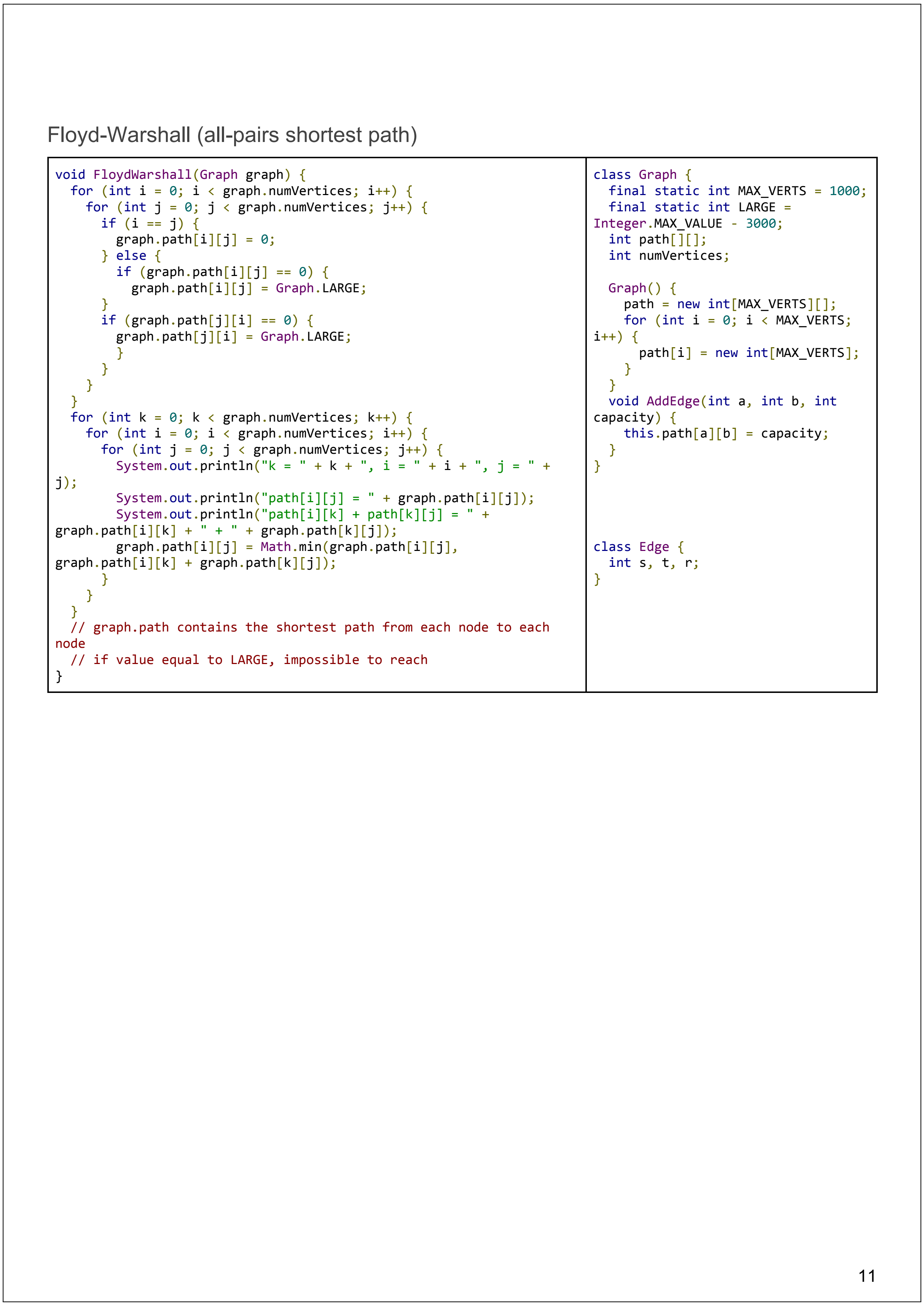
```
public class BellmanFord{
  LinkedList<Edge> edges;
  int d[], p[];
  int n, e, s;
  final int INFINITY = Integer.MAX_VALUE;

  private static class Edge {
    int u, v, w;

    public Edge(int a, int b, int c) {
      u = a;
      v = b;
      w = c;
    }
  }

  BellmanFord() throws IOException {
```

9



```
    int item;
    edges = new LinkedList<Edge>();
    BufferedReader inp = new BufferedReader(new InputStreamReader(System.in));

    System.out.print("Enter number of vertices ");
    n = Integer.parseInt(inp.readLine());

    System.out.println("Cost Matrix");
    for (int i = 0; i < n; i++) {
      for (int j = 0; j < n; j++) {
        item = Integer.parseInt(inp.readLine());
        if (item != 0)
          edges.add(new Edge(i, j, item));
      }
    }
    e = edges.size();
    d = new int[n];
    p = new int[n];
    System.out.print("Enter the source vertex ");
    s = Integer.parseInt(inp.readLine());
  }

  void relax() {
    int i, j;
    for (i = 0; i < n; ++i) {
      d[i] = INFINITY;
      p[i] = -1;
    }

    d[s] = 0;
    for (i = 0; i < n - 1; ++i) {
      for (j = 0; j < e; ++j) { // here i am calculating the shortest path
        if (d[edges.get(j).u] + edges.get(j).w < d[edges.get(j).v]) {
          d[edges.get(j).v] = d[edges.get(j).u] + edges.get(j).w;
          p[edges.get(j).v] = edges.get(j).u;
        }
      }
    }
  }

  boolean cycle() {
    int j;
    for (j = 0; j < e; ++j)
      if (d[edges.get(j).u] + edges.get(j).w < d[edges.get(j).v])
        return false;
    return true;
  }

  void print() {
    for (int i = 0; i < n; i++) {
      System.out.println("Vertex " + i + " has predecessor " + p[i]);
    }
  }

  public static void main(String args[]) throws IOException {
    BellmanFord r = new BellmanFord();
    r.relax();
    if (r.cycle()) {
      for (int i = 0; i < r.n; i++)
        System.out.println(r.s + " ==> " + r.d[i]);
    } else {
      System.out.println(" There is a negative edge cycle ");
    }
    r.print();
  }
}
```

10

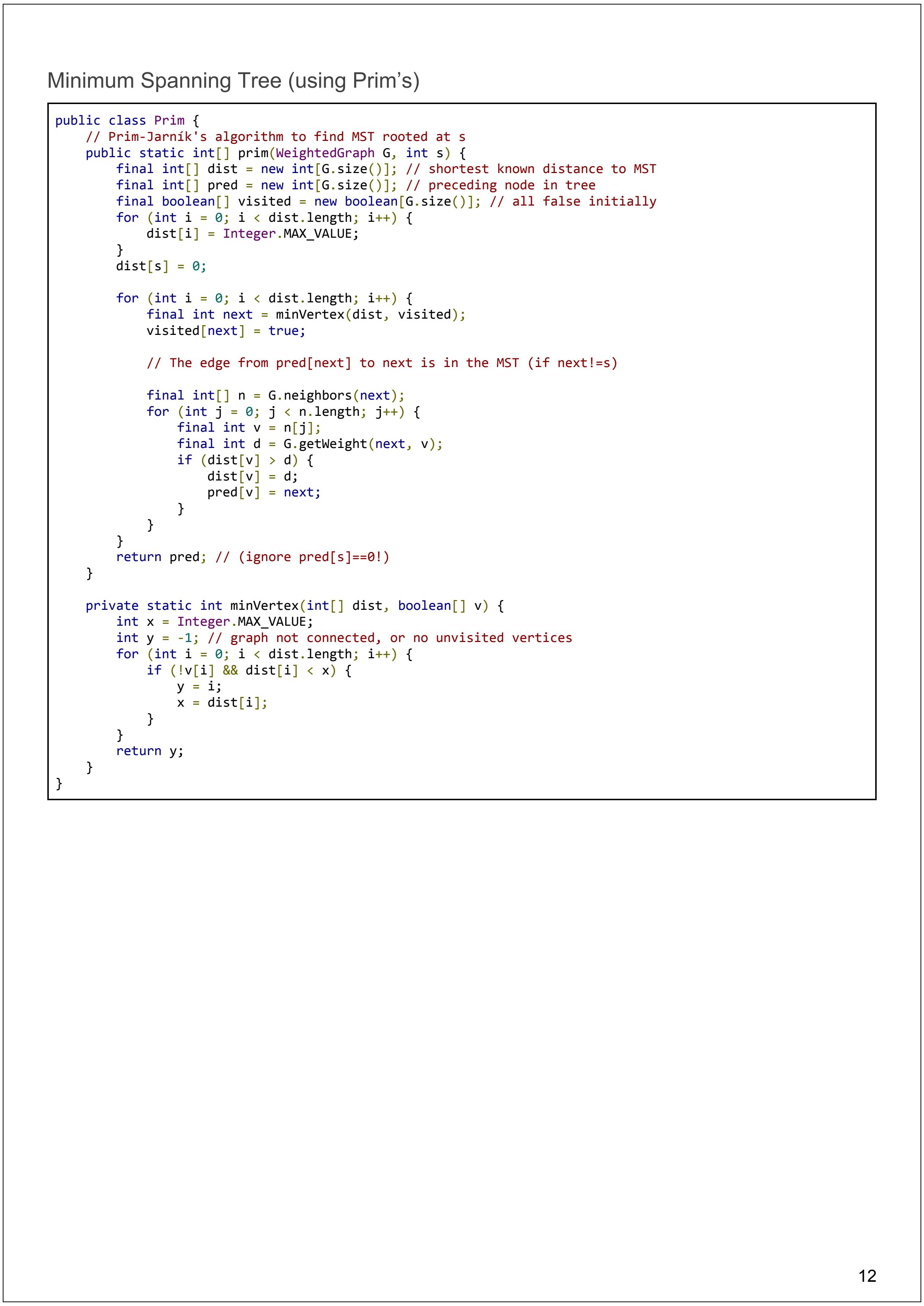


Floyd-Warshall (all-pairs shortest path)

```
void FloydWarshall(Graph graph) {
  for (int i = 0; i < graph.numVertices; i++) {
    for (int j = 0; j < graph.numVertices; j++) {
      if (i == j) {
        graph.path[i][j] = 0;
      } else {
        if (graph.path[i][j] == 0) {
          graph.path[i][j] = Graph.LARGE;
        }
      if (graph.path[j][i] == 0) {
        graph.path[j][i] = Graph.LARGE;
        }
      }
    }
  }
  for (int k = 0; k < graph.numVertices; k++) {
    for (int i = 0; i < graph.numVertices; i++) {
      for (int j = 0; j < graph.numVertices; j++) {
        System.out.println("k = " + k + ", i = " + i + ", j = " +
j);
        System.out.println("path[i][j] = " + graph.path[i][j]);
        System.out.println("path[i][k] + path[k][j] = " +
graph.path[i][k] + " " + " " + graph.path[k][j]);
        graph.path[i][j] = Math.min(graph.path[i][j],
graph.path[i][k] + graph.path[k][j]);
      }
    }
  }
  // graph.path contains the shortest path from each node to each
node
  // if value equal to LARGE, impossible to reach
}
```

```
class Graph {
  final static int MAX_VERTS = 1000;
  final static int LARGE =
Integer.MAX_VALUE - 3000;
  int path[][];
  int numVertices;

  Graph() {
    path = new int[MAX_VERTS][];
    for (int i = 0; i < MAX_VERTS;
i++) {
      path[i] = new int[MAX_VERTS];
    }
  }
  void AddEdge(int a, int b, int
capacity) {
    this.path[a][b] = capacity;
  }
}



class Edge {
  int s, t, r;
}
```

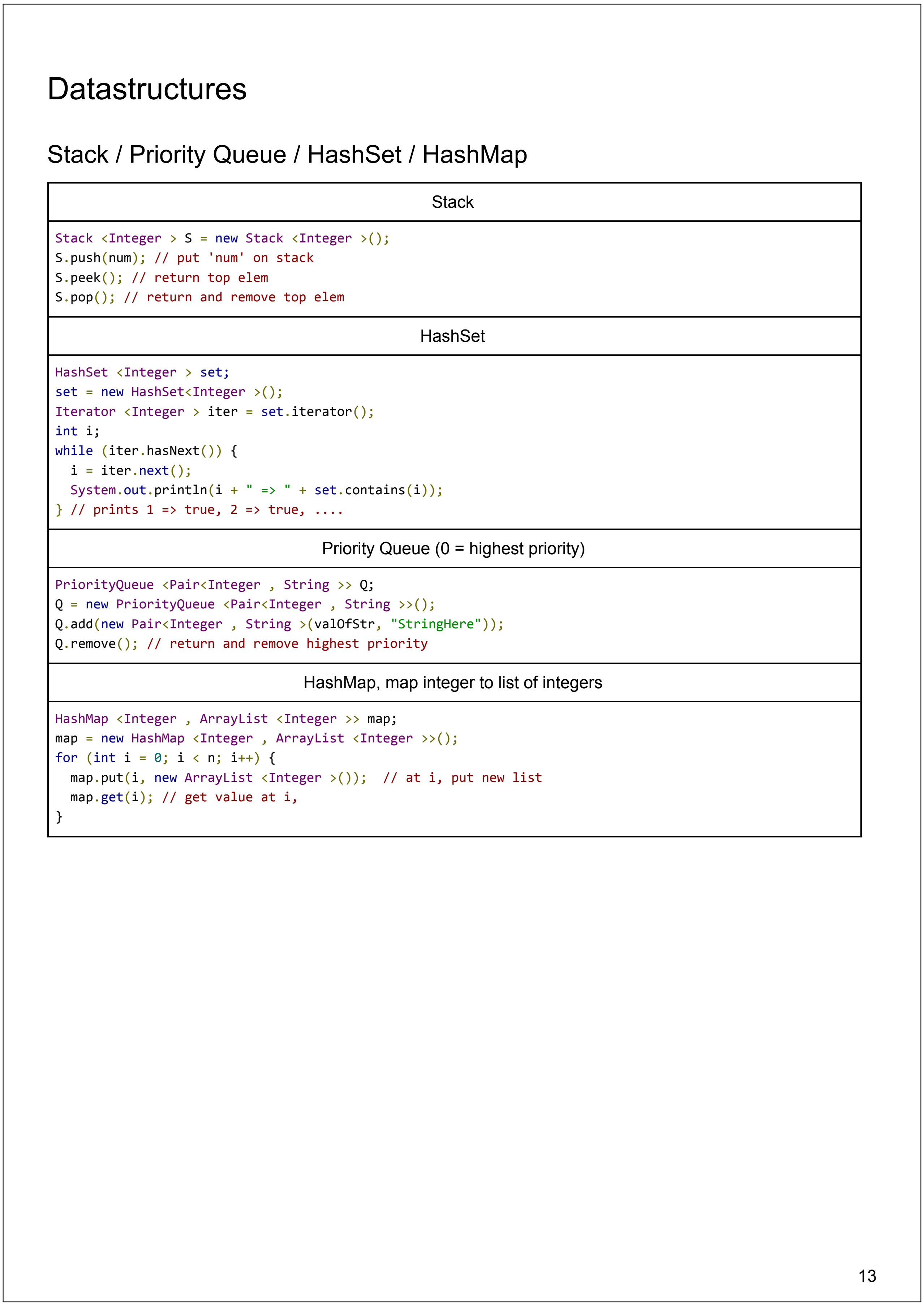
11



Minimum Spanning Tree (using Prim's)

```
public class Prim {
    // Prim-Jarník's algorithm to find MST rooted at s
    public static int[] prim(WeightedGraph G, int s) {
        final int[] dist = new int[G.size()]; // shortest known distance to MST
        final int[] pred = new int[G.size()]; // preceding node in tree
        final boolean[] visited = new boolean[G.size()]; // all false initially
        for (int i = 0; i < dist.length; i++) {
            dist[i] = Integer.MAX_VALUE;
        }
        dist[s] = 0;

        for (int i = 0; i < dist.length; i++) {
            final int next = minVertex(dist, visited);
            visited[next] = true;

            // The edge from pred[next] to next is in the MST (if next!=s)

            final int[] n = G.neighbors(next);
            for (int j = 0; j < n.length; j++) {
                final int v = n[j];
                final int d = G.getWeight(next, v);
                if (dist[v] > d) {
                    dist[v] = d;
                    pred[v] = next;
                }
            }
        }
        return pred; // (ignore pred[s]==0!)
    }

    private static int minVertex(int[] dist, boolean[] v) {
        int x = Integer.MAX_VALUE;
        int y = -1; // graph not connected, or no unvisited vertices
        for (int i = 0; i < dist.length; i++) {
            if (!v[i] && dist[i] < x) {
                y = i;
                x = dist[i];
            }
        }
        return y;
    }
}
```

12



Datastructures

Stack / Priority Queue / HashSet / HashMap

Stack

```
Stack <Integer > S = new Stack <Integer >();
S.push(num); // put 'num' on stack
S.peek(); // return top elem
S.pop(); // return and remove top elem
```

HashSet

```
HashSet <Integer > set;
set = new HashSet<Integer >();
Iterator <Integer > iter = set.iterator();
int i;
while (iter.hasNext()) {
  i = iter.next();
  System.out.println(i + " => " + set.contains(i));
} // prints 1 => true, 2 => true, ....
```

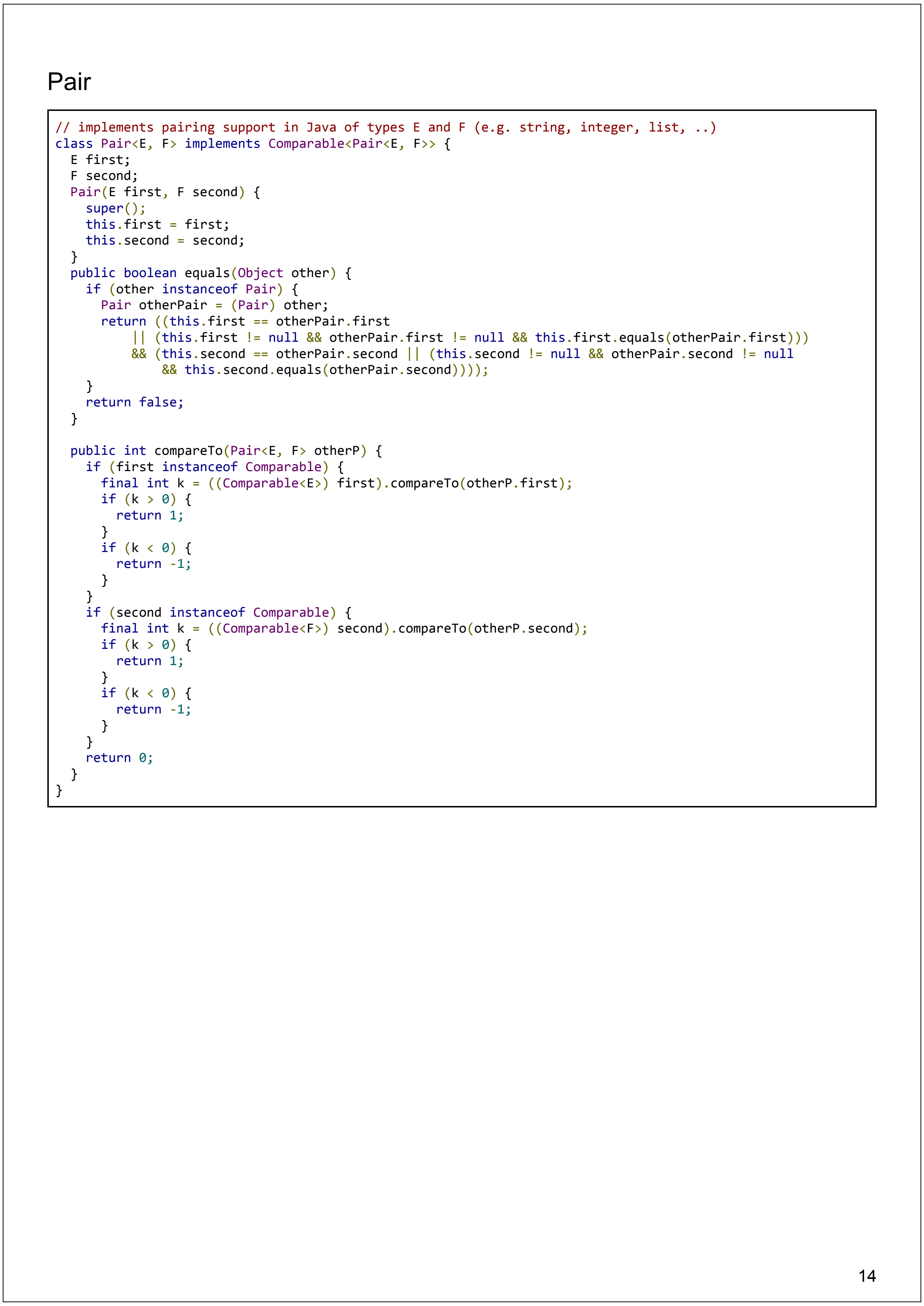
Priority Queue (0 = highest priority)

```
PriorityQueue <Pair<Integer , String >> Q;
Q = new PriorityQueue <Pair<Integer , String >>();
Q.add(new Pair<Integer , String >(valOfStr, "StringHere"));
Q.remove(); // return and remove highest priority
```

HashMap, map integer to list of integers

```
HashMap <Integer , ArrayList <Integer >> map;
map = new HashMap <Integer , ArrayList <Integer >>();
for (int i = 0; i < n; i++) {
  map.put(i, new ArrayList <Integer >());  // at i, put new list
  map.get(i); // get value at i,
}
```

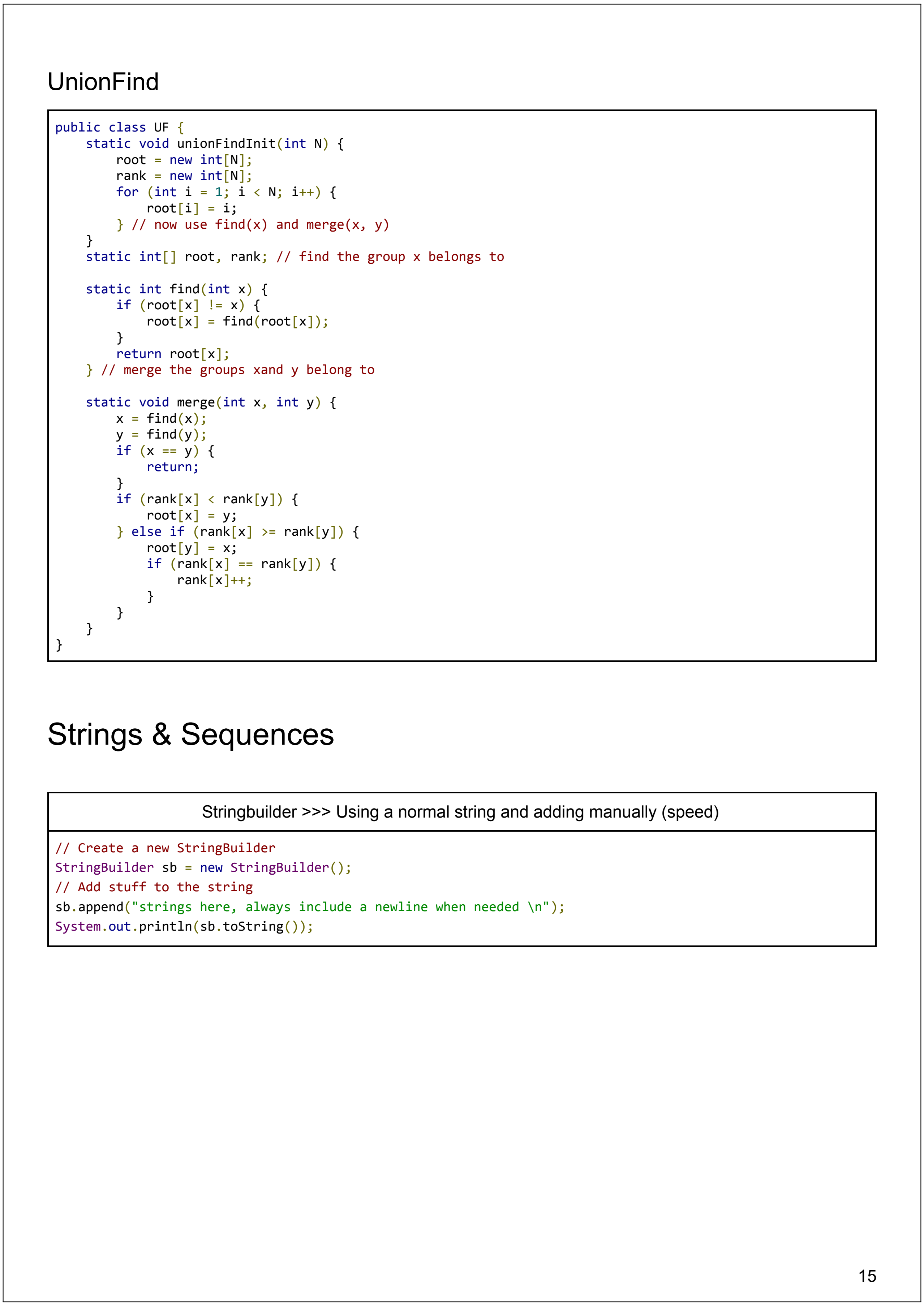
13



Pair

```java
// implements pairing support in Java of types E and F (e.g. string, integer, list, ..)
class Pair<E, F> implements Comparable<Pair<E, F>> {
  E first;
  F second;
  Pair(E first, F second) {
    super();
    this.first = first;
    this.second = second;
  }
  public boolean equals(Object other) {
    if (other instanceof Pair) {
      Pair otherPair = (Pair) other;
      return ((this.first == otherPair.first
          || (this.first != null && otherPair.first != null && this.first.equals(otherPair.first)))
          && (this.second == otherPair.second || (this.second != null && otherPair.second != null
              && this.second.equals(otherPair.second))));
    }
    return false;
  }

  public int compareTo(Pair<E, F> otherP) {
    if (first instanceof Comparable) {
      final int k = ((Comparable<E>) first).compareTo(otherP.first);
      if (k > 0) {
        return 1;
      }
      if (k < 0) {
        return -1;
      }
    }
    if (second instanceof Comparable) {
      final int k = ((Comparable<F>) second).compareTo(otherP.second);
      if (k > 0) {
        return 1;
      }
      if (k < 0) {
        return -1;
      }
    }
    return 0;
  }
}
```

14



UnionFind

```
public class UF {
    static void unionFindInit(int N) {
        root = new int[N];
        rank = new int[N];
        for (int i = 1; i < N; i++) {
            root[i] = i;
        } // now use find(x) and merge(x, y)
    }
    static int[] root, rank; // find the group x belongs to

    static int find(int x) {
        if (root[x] != x) {
            root[x] = find(root[x]);
        }
        return root[x];
    } // merge the groups xand y belong to

    static void merge(int x, int y) {
        x = find(x);
        y = find(y);
        if (x == y) {
            return;
        }
        if (rank[x] < rank[y]) {
            root[x] = y;
        } else if (rank[x] >= rank[y]) {
            root[y] = x;
            if (rank[x] == rank[y]) {
                rank[x]++;
            }
        }
    }
}
```

Strings & Sequences

Stringbuilder >>> Using a normal string and adding manually (speed)

```
// Create a new StringBuilder
StringBuilder sb = new StringBuilder();
// Add stuff to the string
sb.append("strings here, always include a newline when needed \n");
System.out.println(sb.toString());
```

15

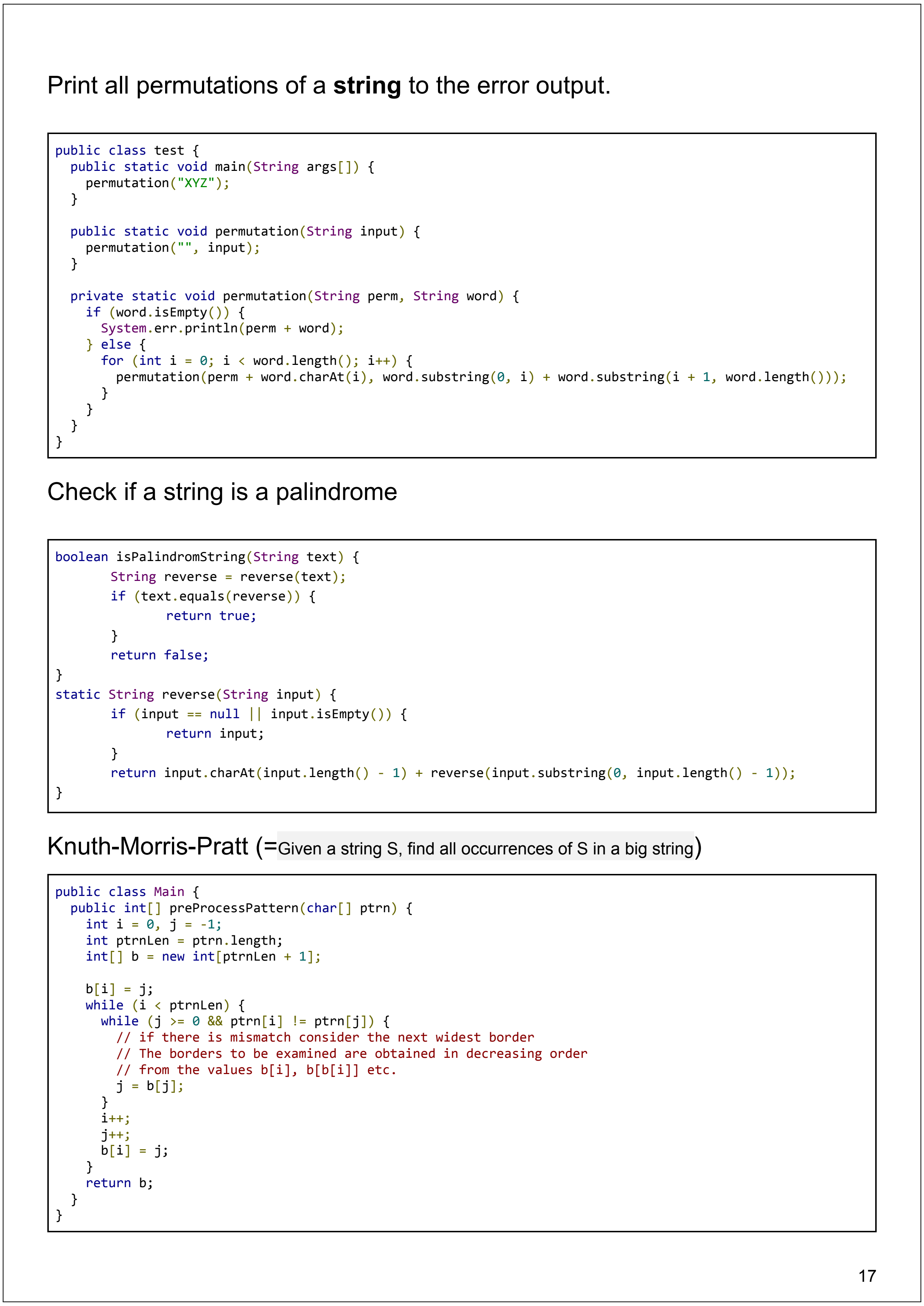
Check for duplicate characters in a string and print them.

```java
public class FindDuplicateCharacters {
        public static void main(String args[]) {
                printDuplicateCharacters("Programming");
                printDuplicateCharacters("Combination");
                printDuplicateCharacters("Java");
        }
        /*
         * Find all duplicate characters in a String and print each of them.
         */
        public static void printDuplicateCharacters(String word) {
                char[] characters = word.toCharArray();
                // build HashMap with character and number of times they appear in
                // String
                Map<Character, Integer> charMap = new HashMap<Character, Integer>();
                for (Character ch : characters) {
                        if (charMap.containsKey(ch)) {
                                charMap.put(ch, charMap.get(ch) + 1);
                        } else {
                                charMap.put(ch, 1);
                        }
                }
                // Iterate through HashMap to print all duplicate characters of String
                Set<Map.Entry<Character, Integer>> entrySet = charMap.entrySet();
                System.out.printf("List of duplicate characters in String '%s' %n", word);
                for (Map.Entry<Character, Integer> entry : entrySet) {
                        if (entry.getValue() > 1) {
                                System.out.printf("%s : %d %n", entry.getKey(), entry.getValue());
                        }
                }
        }
}
```

Check if a string is an anagram

```java
boolean isAnagram(String word, String anagram) {
                char[] charFromWord = word.toLowerCase().toCharArray();
                char[] charFromAnagram = anagram.toLowerCase().toCharArray();
                Arrays.sort(charFromWord);
                Arrays.sort(charFromAnagram);
                return Arrays.equals(charFromWord, charFromAnagram);
}
```

16



Print all permutations of a **string** to the error output.

```java
public class test {
  public static void main(String args[]) {
    permutation("XYZ");
  }

  public static void permutation(String input) {
    permutation("", input);
  }

  private static void permutation(String perm, String word) {
    if (word.isEmpty()) {
      System.err.println(perm + word);
    } else {
      for (int i = 0; i < word.length(); i++) {
        permutation(perm + word.charAt(i), word.substring(0, i) + word.substring(i + 1, word.length()));
      }
    }
  }
}
```

Check if a string is a palindrome

```java
boolean isPalindromString(String text) {
       String reverse = reverse(text);
       if (text.equals(reverse)) {
              return true;
       }
       return false;
}
static String reverse(String input) {
       if (input == null || input.isEmpty()) {
              return input;
       }
       return input.charAt(input.length() - 1) + reverse(input.substring(0, input.length() - 1));
}
```

Knuth-Morris-Pratt (=Given a string S, find all occurrences of S in a big string)

```java
public class Main {
  public int[] preProcessPattern(char[] ptrn) {
    int i = 0, j = -1;
    int ptrnLen = ptrn.length;
    int[] b = new int[ptrnLen + 1];

    b[i] = j;
    while (i < ptrnLen) {
      while (j >= 0 && ptrn[i] != ptrn[j]) {
        // if there is mismatch consider the next widest border
        // The borders to be examined are obtained in decreasing order
        // from the values b[i], b[b[i]] etc.
        j = b[j];
      }
      i++;
      j++;
      b[i] = j;
    }
    return b;
  }
}
```

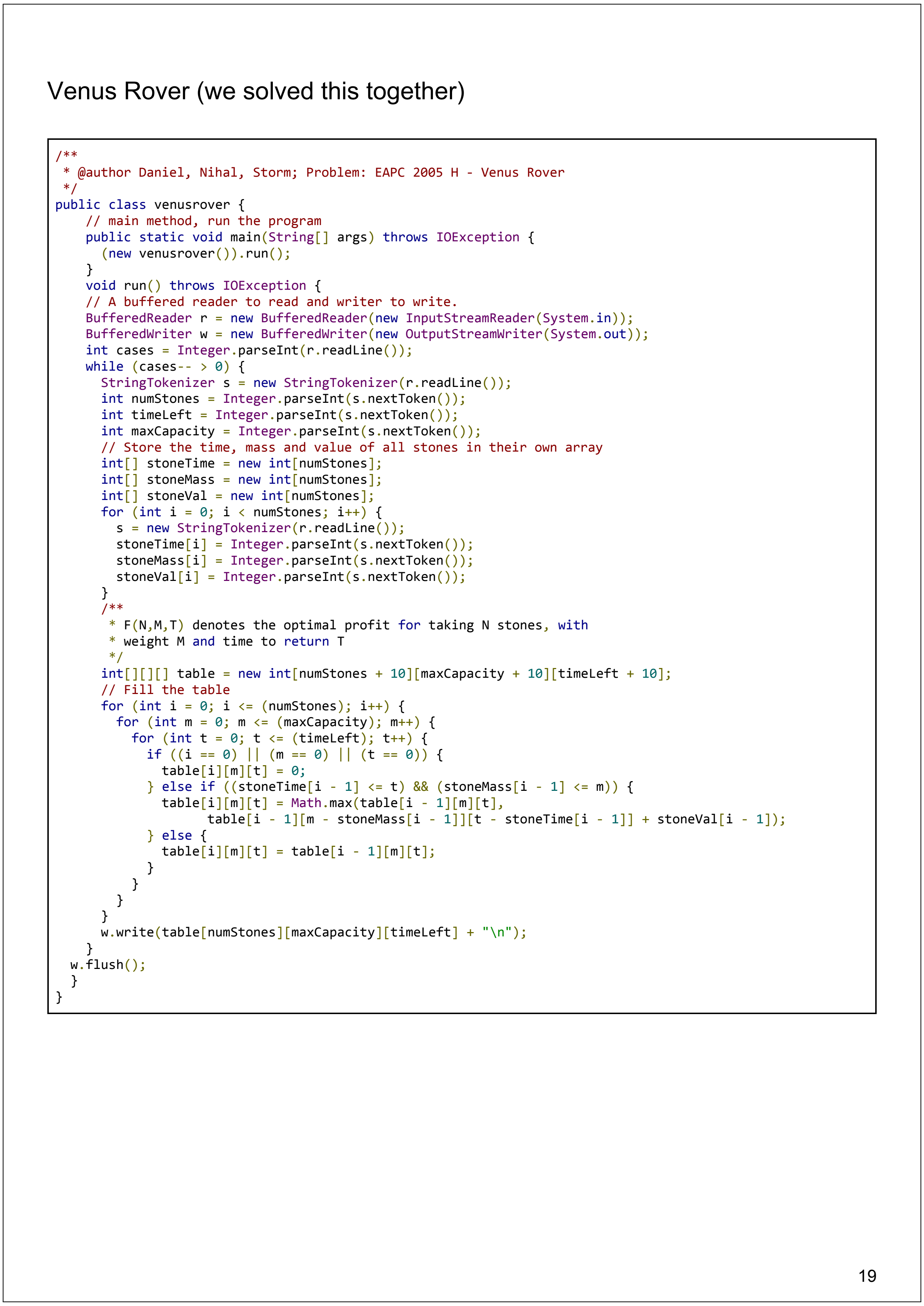
17

Dynamic Programming

Longest Common Subsequence

```
/* Dynamic Programming Java implementation of LCS problem */
public class LCS {

  /* Returns length of LCS for X[0..m-1], Y[0..n-1] */
  int lcs(char[] X, char[] Y, int m, int n) {
    int L[][] = new int[m + 1][n + 1];

    /*
     * Following steps build L[m+1][n+1] in bottom up fashion. Note that
     * L[i][j] contains length of LCS of X[0..i-1] and Y[0..j-1]
     */
    for (int i = 0; i <= m; i++) {
      for (int j = 0; j <= n; j++) {
        if (i == 0 || j == 0)
          L[i][j] = 0;
        else if (X[i - 1] == Y[j - 1])
          L[i][j] = L[i - 1][j - 1] + 1;
        else
          L[i][j] = max(L[i - 1][j], L[i][j - 1]);
      }
    }
    return L[m][n];
  }

  /* Utility function to get max of 2 integers */
  int max(int a, int b) {
    return (a > b) ? a : b;
  }

  public static void main(String[] args) {
    LCS lcs = new LCS();
    String s1 = "AGGTAB";
    String s2 = "GXTXAYB";

    char[] X = s1.toCharArray();
    char[] Y = s2.toCharArray();
    int m = X.length;
    int n = Y.length;

    System.out.println("Length of LCS is" + " " + lcs.lcs(X, Y, m, n));
  }
}
```

18



Venus Rover (we solved this together)

```
/**
 * @author Daniel, Nihal, Storm; Problem: EAPC 2005 H - Venus Rover
 */
public class venusrover {
    // main method, run the program
    public static void main(String[] args) throws IOException {
      (new venusrover()).run();
    }
    void run() throws IOException {
    // A buffered reader to read and writer to write.
    BufferedReader r = new BufferedReader(new InputStreamReader(System.in));
    BufferedWriter w = new BufferedWriter(new OutputStreamWriter(System.out));
    int cases = Integer.parseInt(r.readLine());
    while (cases-- > 0) {
      StringTokenizer s = new StringTokenizer(r.readLine());
      int numStones = Integer.parseInt(s.nextToken());
      int timeLeft = Integer.parseInt(s.nextToken());
      int maxCapacity = Integer.parseInt(s.nextToken());
      // Store the time, mass and value of all stones in their own array
      int[] stoneTime = new int[numStones];
      int[] stoneMass = new int[numStones];
      int[] stoneVal = new int[numStones];
      for (int i = 0; i < numStones; i++) {
        s = new StringTokenizer(r.readLine());
        stoneTime[i] = Integer.parseInt(s.nextToken());
        stoneMass[i] = Integer.parseInt(s.nextToken());
        stoneVal[i] = Integer.parseInt(s.nextToken());
      }
      /**
       * F(N,M,T) denotes the optimal profit for taking N stones, with
       * weight M and time to return T
       */
      int[][][] table = new int[numStones + 10][maxCapacity + 10][timeLeft + 10];
      // Fill the table
      for (int i = 0; i <= (numStones); i++) {
        for (int m = 0; m <= (maxCapacity); m++) {
          for (int t = 0; t <= (timeLeft); t++) {
            if ((i == 0) || (m == 0) || (t == 0)) {
              table[i][m][t] = 0;
            } else if ((stoneTime[i - 1] <= t) && (stoneMass[i - 1] <= m)) {
              table[i][m][t] = Math.max(table[i - 1][m][t],
                    table[i - 1][m - stoneMass[i - 1]][t - stoneTime[i - 1]] + stoneVal[i - 1]);
            } else {
              table[i][m][t] = table[i - 1][m][t];
            }
          }
        }
      }
      w.write(table[numStones][maxCapacity][timeLeft] + "\n");
    }
  w.flush();
  }
}
```

19

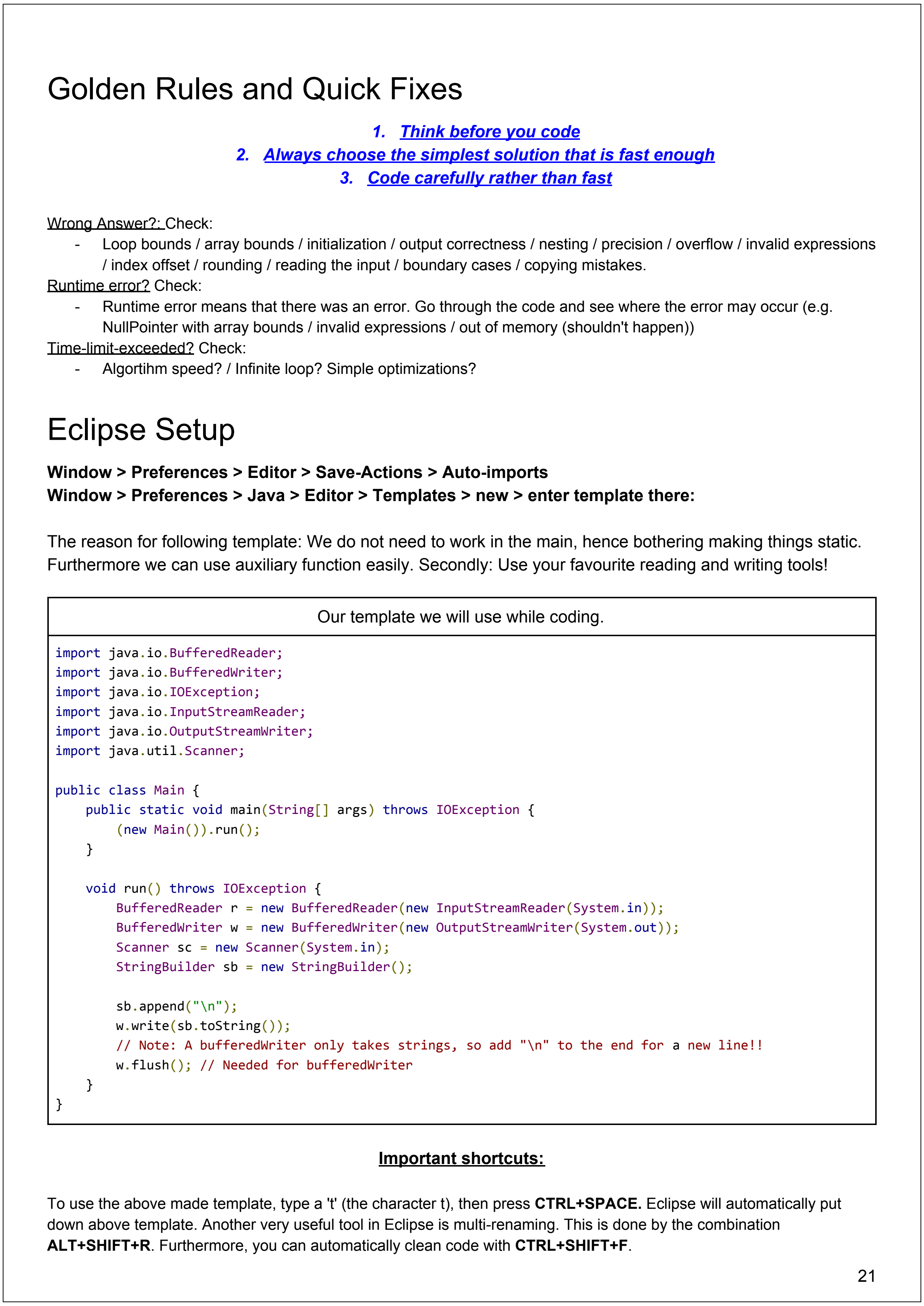
Knapsack 0/1

```
static List<Integer> knapsack(int capacity, int n, int[] values, int[] weights) {
  int[][] bestValue = new int[n + 1][capacity + 1];
  for (int i = 1; i <= n; i++) {
    int itemIndex = i - 1;
    for (int j = 0; j <= capacity; j++) {
      int weightIfNotIncluded = bestValue[i - 1][j];
      if (j >= weights[itemIndex]) {
        int weightIfIncluded = bestValue[i - 1][j - weights[itemIndex]] + values[itemIndex];
        bestValue[i][j] = Main.max(weightIfIncluded, weightIfNotIncluded);
      } else {
        bestValue[i][j] = weightIfNotIncluded;
      }
    }
  }
  // you can check the bestValue[n][capacity] here for example
  // backtrack the chosen items (optional)
  List<Integer> chosen = new ArrayList<>();
  int j = capacity;
  for (int i = n; i > 0; i--) {
    if (bestValue[i][j] != bestValue[i - 1][j]) {
      int itemIndex = i - 1;
      chosen.add(itemIndex);
      j -= weights[itemIndex];
    }
  }
  return chosen;
}

static int max(int a, int b) {
  return (a > b) ? a : b;
}
```

Common problems with DP implementation

- The dimensions of the table in a DP should be 1 bigger than the actual values, since the 0th row and columns are initial values.
- The loopbounds when filling in the table should be "i <= X", not "i < X"
- We should account for indexing offset (in venusrover: in stoneTime, stoneMass and stoneVal) in the recursion.

20



Golden Rules and Quick Fixes

1. ***Think before you code***
2. ***Always choose the simplest solution that is fast enough***
3. ***Code carefully rather than fast***

Wrong Answer?: Check:
- Loop bounds / array bounds / initialization / output correctness / nesting / precision / overflow / invalid expressions / index offset / rounding / reading the input / boundary cases / copying mistakes.

Runtime error? Check:
- Runtime error means that there was an error. Go through the code and see where the error may occur (e.g. NullPointer with array bounds / invalid expressions / out of memory (shouldn't happen))

Time-limit-exceeded? Check:
- Algortihm speed? / Infinite loop? Simple optimizations?

Eclipse Setup

Window > Preferences > Editor > Save-Actions > Auto-imports
Window > Preferences > Java > Editor > Templates > new > enter template there:

The reason for following template: We do not need to work in the main, hence bothering making things static. Furthermore we can use auxiliary function easily. Secondly: Use your favourite reading and writing tools!

Our template we will use while coding.

```java
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.util.Scanner;

public class Main {
    public static void main(String[] args) throws IOException {
        (new Main()).run();
    }

    void run() throws IOException {
        BufferedReader r = new BufferedReader(new InputStreamReader(System.in));
        BufferedWriter w = new BufferedWriter(new OutputStreamWriter(System.out));
        Scanner sc = new Scanner(System.in);
        StringBuilder sb = new StringBuilder();

        sb.append("\n");
        w.write(sb.toString());
        // Note: A bufferedWriter only takes strings, so add "\n" to the end for a new line!!
        w.flush(); // Needed for bufferedWriter
    }
}
```

Important shortcuts:

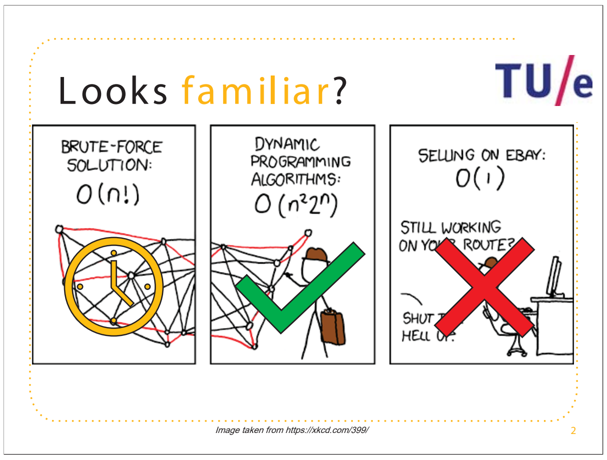
To use the above made template, type a 't' (the character t), then press **CTRL+SPACE.** Eclipse will automatically put down above template. Another very useful tool in Eclipse is multi-renaming. This is done by the combination **ALT+SHIFT+R**. Furthermore, you can automatically clean code with **CTRL+SHIFT+F**.

21

C Seminar Slides

TU/e

Seminar subset DP
by Daniël Barenholz


Looks familiar?

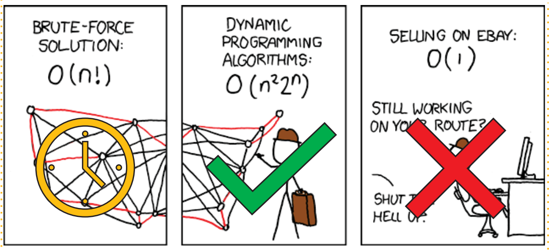
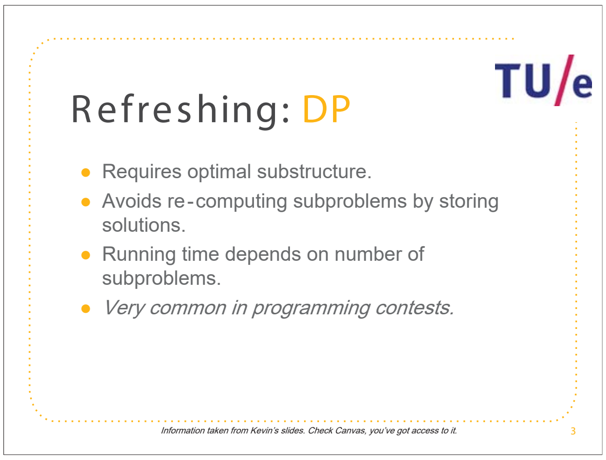
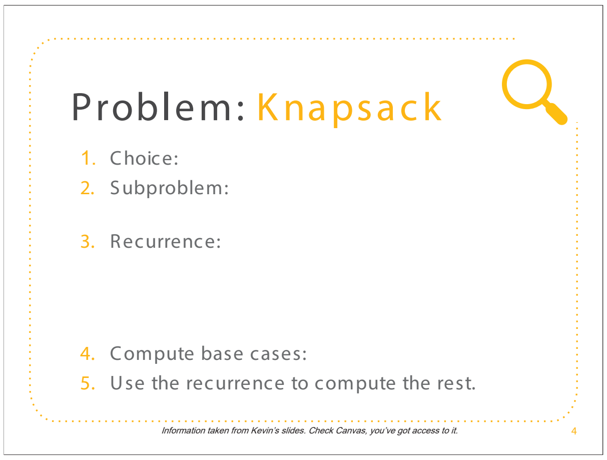

BRUTE-FORCE SOLUTION: O(n!)
DYNAMIC PROGRAMMING ALGORITHMS: O(n²2ⁿ)
SELLING ON EBAY: O(1)
STILL WORKING ON YOUR ROUTE?
SHUT THE HELL UP.

Image taken from https://xkcd.com/399/ 2


Refreshing: DP

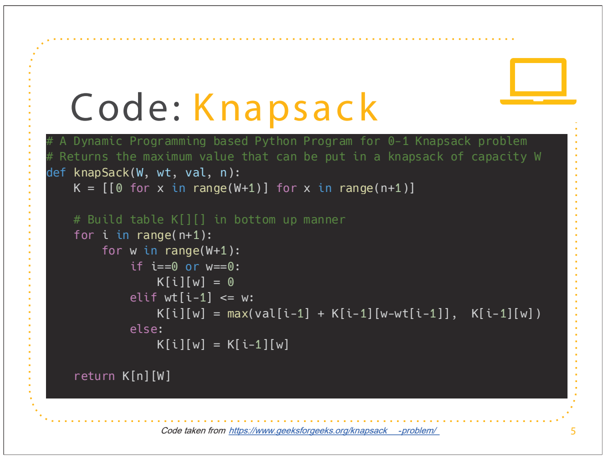
- Requires optimal substructure.
- Avoids re-computing subproblems by storing solutions.
- Running time depends on number of subproblems.
- *Very common in programming contests.*

Information taken from Kevin's slides. Check Canvas, you've got access to it. 3


Problem: Knapsack

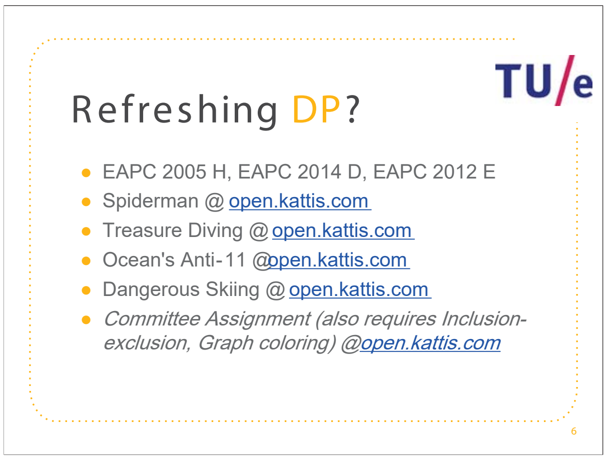
1. Choice:
2. Subproblem:
3. Recurrence:
4. Compute base cases:
5. Use the recurrence to compute the rest.

Information taken from Kevin's slides. Check Canvas, you've got access to it. 4


Code: Knapsack

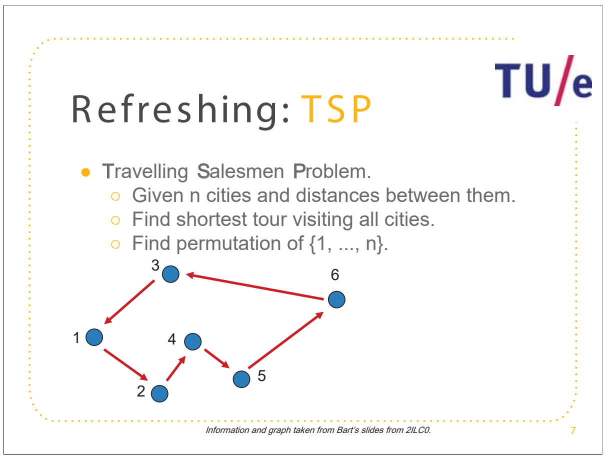
```python
# A Dynamic Programming based Python Program for 0-1 Knapsack problem
# Returns the maximum value that can be put in a knapsack of capacity W
def knapSack(W, wt, val, n):
    K = [[0 for x in range(W+1)] for x in range(n+1)]

    # Build table K[][] in bottom up manner
    for i in range(n+1):
        for w in range(W+1):
            if i==0 or w==0:
                K[i][w] = 0
            elif wt[i-1] <= w:
                K[i][w] = max(val[i-1] + K[i-1][w-wt[i-1]],  K[i-1][w])
            else:
                K[i][w] = K[i-1][w]

    return K[n][W]
```

Code taken from https://www.geeksforgeeks.org/knapsack -problem/ 5


Refreshing DP?

- EAPC 2005 H, EAPC 2014 D, EAPC 2012 E
- Spiderman @ open.kattis.com
- Treasure Diving @ open.kattis.com
- Ocean's Anti-11 @open.kattis.com
- Dangerous Skiing @ open.kattis.com
- *Committee Assignment (also requires Inclusion-exclusion, Graph coloring) @open.kattis.com*

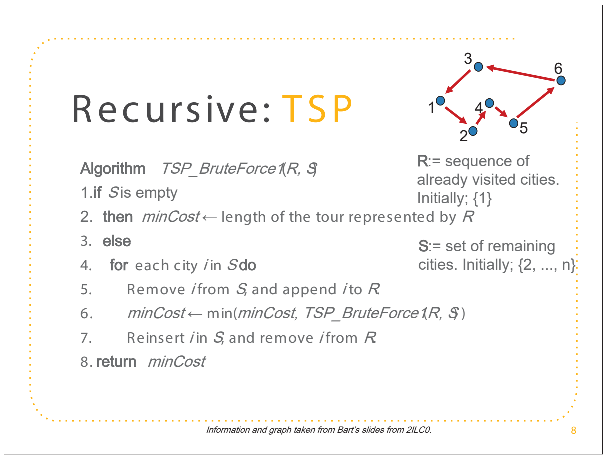
6



Refreshing: TSP

- Travelling Salesmen Problem.
  - Given n cities and distances between them.
  - Find shortest tour visiting all cities.
  - Find permutation of {1, ..., n}.

Information and graph taken from Bart's slides from 2ILC0. 7



Recursive: TSP

R:= sequence of already visited cities. Initially; {1}

S:= set of remaining cities. Initially; {2, ..., n}

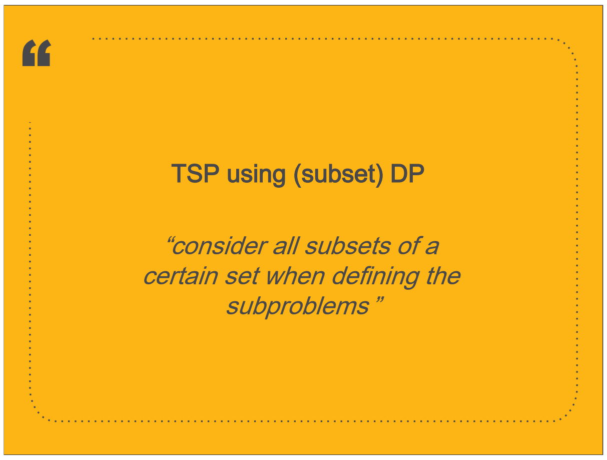
```
Algorithm  TSP_BruteForce1(R, S)
1. if S is empty
2.   then minCost ← length of the tour represented by R
3.   else
4.     for each city i in S do
5.         Remove i from S, and append i to R.
6.         minCost ← min(minCost, TSP_BruteForce1(R, S))
7.         Reinsert i in S, and remove i from R.
8. return minCost
```

Information and graph taken from Bart's slides from 2ILC0. 8

!

Can we do better?

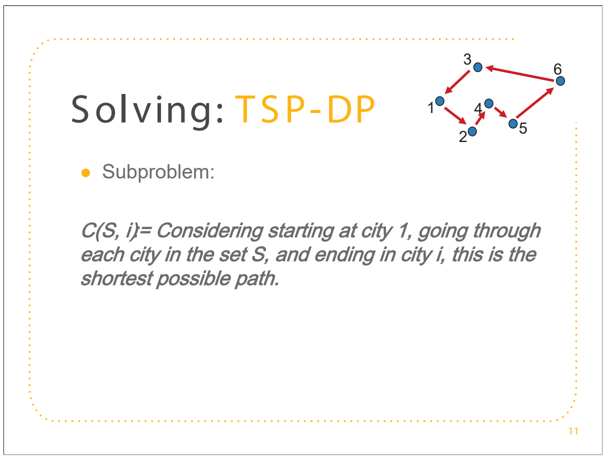
Yes.



“

TSP using (subset) DP

“consider all subsets of a certain set when defining the subproblems”

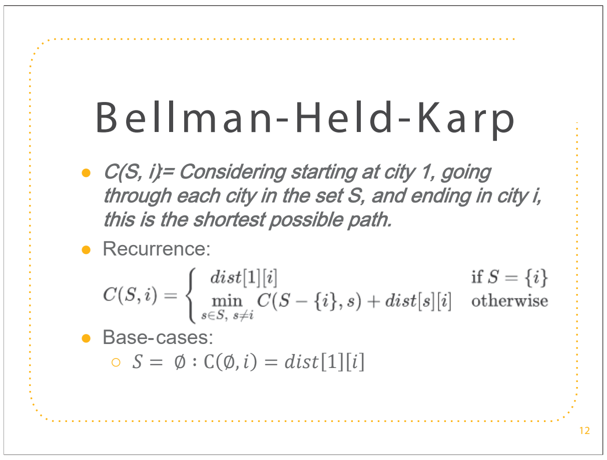


Solving: TSP-DP

- Subproblem:

C(S, i)= Considering starting at city 1, going through each city in the set S, and ending in city i, this is the shortest possible path.

11



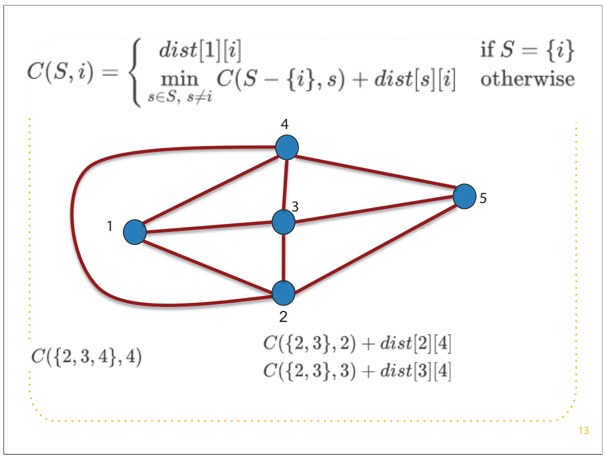
Bellman-Held-Karp

- *C(S, i)= Considering starting at city 1, going through each city in the set S, and ending in city i, this is the shortest possible path.*
- Recurrence:

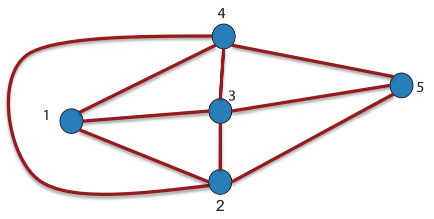
$$C(S,i) = \begin{cases} dist[1][i] & \text{if } S = \{i\} \\ \min\limits_{s \in S,\ s \neq i} C(S - \{i\}, s) + dist[s][i] & \text{otherwise} \end{cases}$$

- Base-cases:
  - $S = \emptyset : C(\emptyset, i) = dist[1][i]$

12



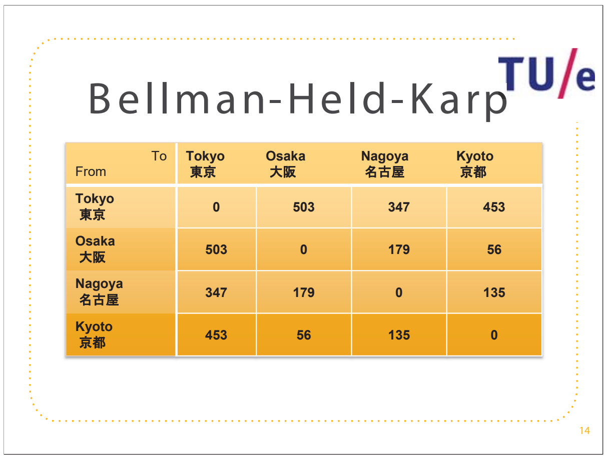
$$C(S,i) = \begin{cases} dist[1][i] & \text{if } S = \{i\} \\ \min\limits_{s \in S,\ s \neq i} C(S - \{i\}, s) + dist[s][i] & \text{otherwise} \end{cases}$$



$C(\{2,3,4\},4)$

$C(\{2,3\},2) + dist[2][4]$
$C(\{2,3\},3) + dist[3][4]$

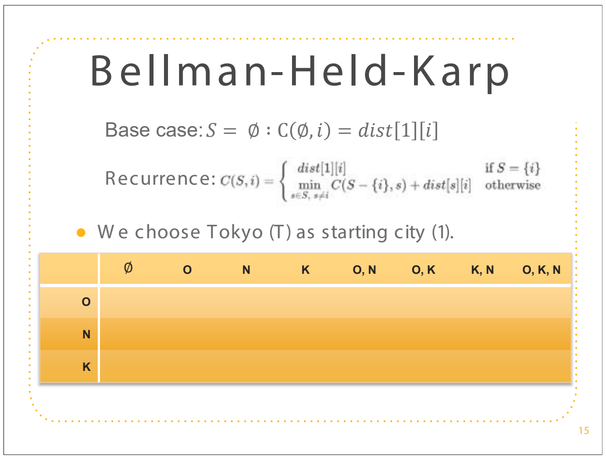
13



Bellman-Held-Karp

From \ To	Tokyo 東京	Osaka 大阪	Nagoya 名古屋	Kyoto 京都
Tokyo 東京	0	503	347	453
Osaka 大阪	503	0	179	56
Nagoya 名古屋	347	179	0	135
Kyoto 京都	453	56	135	0

14



Bellman-Held-Karp

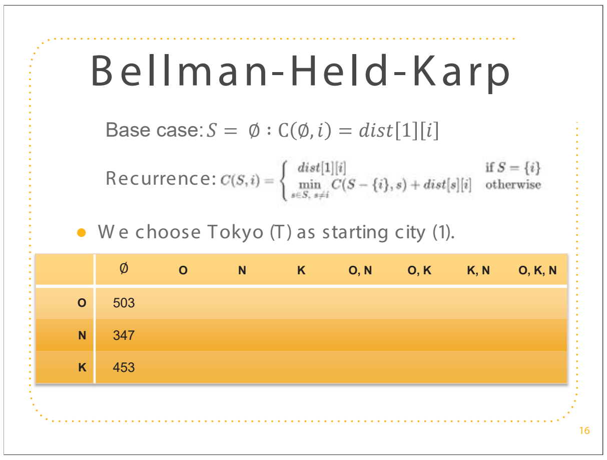
Base case: $S = \emptyset : C(\emptyset, i) = dist[1][i]$

Recurrence: $C(S,i) = \begin{cases} dist[1][i] & \text{if } S = \{i\} \\ \min\limits_{s \in S,\ s \neq i} C(S - \{i\}, s) + dist[s][i] & \text{otherwise} \end{cases}$

- We choose Tokyo (T) as starting city (1).

	∅	O	N	K	O, N	O, K	K, N	O, K, N
O								
N								
K								

15



Bellman-Held-Karp

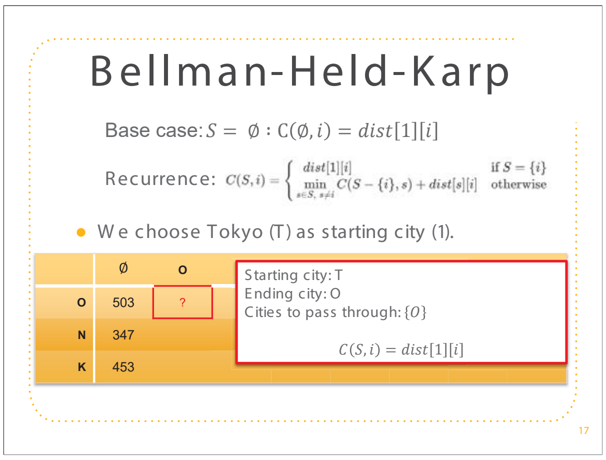
Base case: $S = \emptyset : C(\emptyset, i) = dist[1][i]$

Recurrence: $C(S,i) = \begin{cases} dist[1][i] & \text{if } S = \{i\} \\ \min\limits_{s \in S,\ s \neq i} C(S - \{i\}, s) + dist[s][i] & \text{otherwise} \end{cases}$

- We choose Tokyo (T) as starting city (1).

	∅	O	N	K	O, N	O, K	K, N	O, K, N
O	503							
N	347							
K	453							

16



Bellman-Held-Karp

Base case: $S = \emptyset : C(\emptyset, i) = dist[1][i]$

Recurrence: $C(S,i) = \begin{cases} dist[1][i] & \text{if } S = \{i\} \\ \min\limits_{s \in S,\ s \neq i} C(S - \{i\}, s) + dist[s][i] & \text{otherwise} \end{cases}$

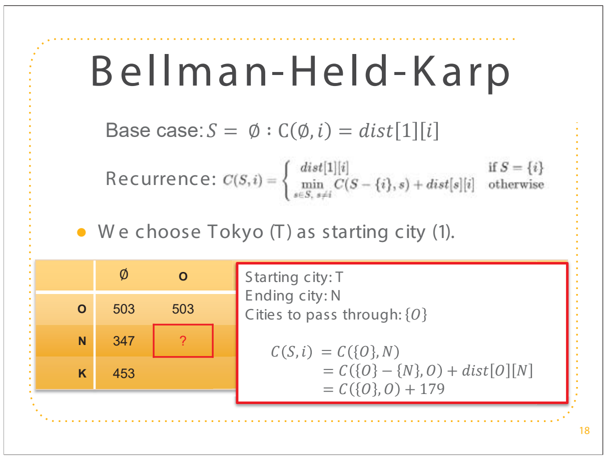
- We choose Tokyo (T) as starting city (1).

	∅	O
O	503	?
N	347	
K	453	

Starting city: T
Ending city: O
Cities to pass through: $\{O\}$

$C(S,i) = dist[1][i]$

17



Bellman-Held-Karp

Base case: $S = \emptyset : C(\emptyset, i) = dist[1][i]$

Recurrence: $C(S,i) = \begin{cases} dist[1][i] & \text{if } S = \{i\} \\ \min\limits_{s \in S,\ s \neq i} C(S - \{i\}, s) + dist[s][i] & \text{otherwise} \end{cases}$

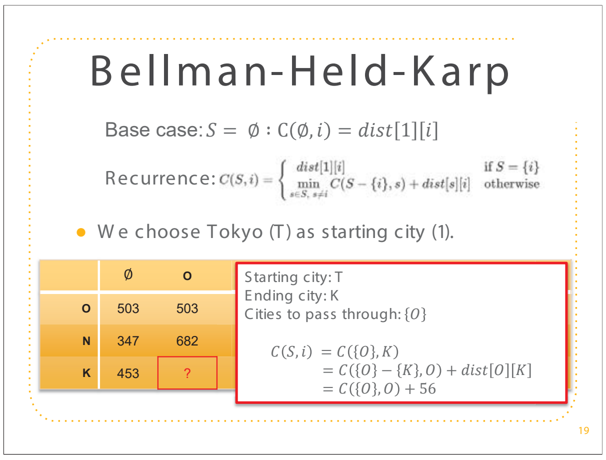
- We choose Tokyo (T) as starting city (1).

	∅	O
O	503	503
N	347	?
K	453	

Starting city: T
Ending city: N
Cities to pass through: $\{O\}$

$$\begin{aligned} C(S,i) &= C(\{O\},N) \\ &= C(\{O\} - \{N\}, O) + dist[O][N] \\ &= C(\{O\}, O) + 179 \end{aligned}$$

18



Bellman-Held-Karp

Base case: $S = \emptyset$: $C(\emptyset, i) = dist[1][i]$

Recurrence: $C(S,i) = \begin{cases} dist[1][i] & \text{if } S = \{i\} \\ \min\limits_{s \in S,\, s \neq i} C(S - \{i\}, s) + dist[s][i] & \text{otherwise} \end{cases}$

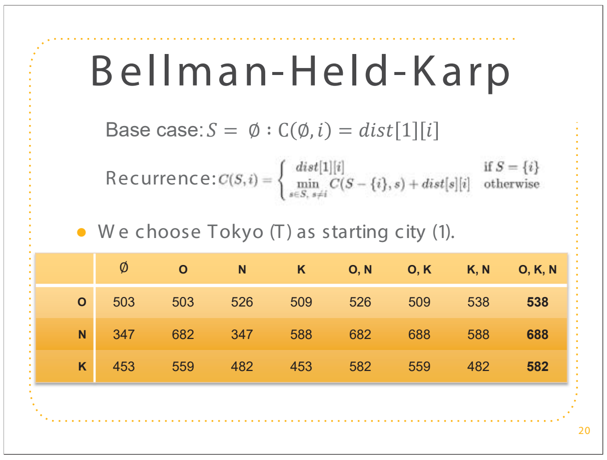
- We choose Tokyo (T) as starting city (1).

	∅	O
O	503	503
N	347	682
K	453	?

Starting city: T
Ending city: K
Cities to pass through: $\{O\}$

$$\begin{aligned} C(S,i) &= C(\{O\}, K) \\ &= C(\{O\} - \{K\}, O) + dist[O][K] \\ &= C(\{O\}, O) + 56 \end{aligned}$$

19



Bellman-Held-Karp

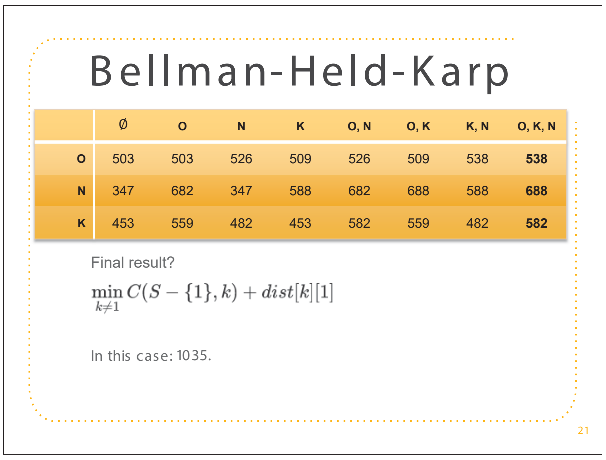
Base case: $S = \emptyset$: $C(\emptyset, i) = dist[1][i]$

Recurrence: $C(S,i) = \begin{cases} dist[1][i] & \text{if } S = \{i\} \\ \min\limits_{s \in S,\, s \neq i} C(S - \{i\}, s) + dist[s][i] & \text{otherwise} \end{cases}$

- We choose Tokyo (T) as starting city (1).

	∅	O	N	K	O, N	O, K	K, N	O, K, N
O	503	503	526	509	526	509	538	**538**
N	347	682	347	588	682	688	588	**688**
K	453	559	482	453	582	559	482	**582**

20



Bellman-Held-Karp

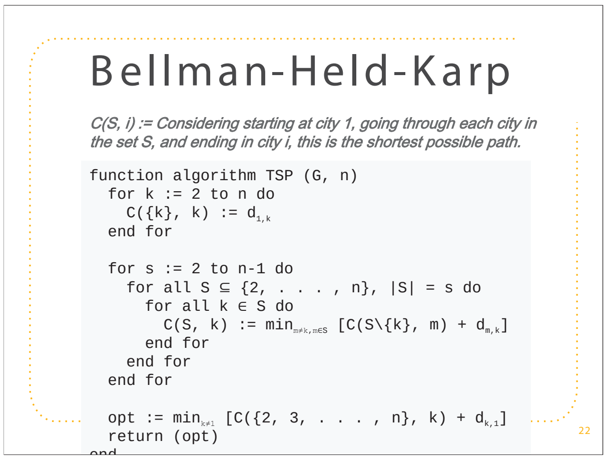
	∅	O	N	K	O, N	O, K	K, N	O, K, N
O	503	503	526	509	526	509	538	**538**
N	347	682	347	588	682	688	588	**688**
K	453	559	482	453	582	559	482	**582**

Final result?

$$\min_{k \neq 1} C(S - \{1\}, k) + dist[k][1]$$

In this case: 1035.

21

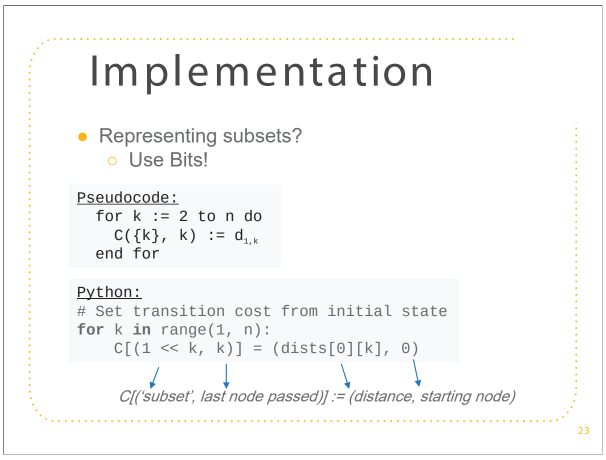


Bellman-Held-Karp

C(S, i) := Considering starting at city 1, going through each city in the set S, and ending in city i, this is the shortest possible path.

```
function algorithm TSP (G, n)
  for k := 2 to n do
    C({k}, k) := $d_{1,k}$
  end for

  for s := 2 to n-1 do
    for all S ⊆ {2, . . . , n}, |S| = s do
      for all k ∈ S do
        C(S, k) := $\min_{m \neq k, m \in S}$ [C(S\{k}, m) + $d_{m,k}$]
      end for
    end for
  end for

  opt := $\min_{k \neq 1}$ [C({2, 3, . . . , n}, k) + $d_{k,1}$]
  return (opt)
```

22



Implementation

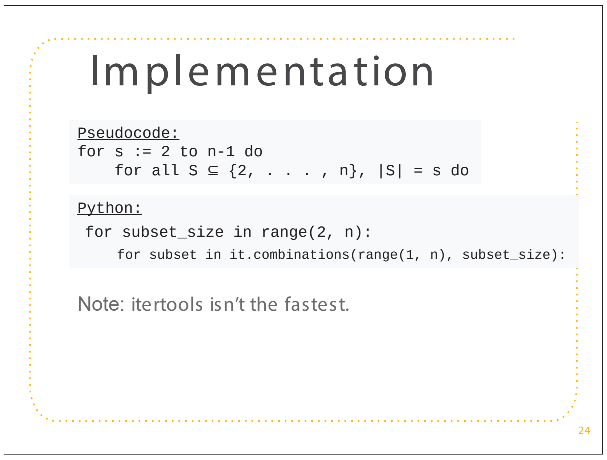
- Representing subsets?
  - Use Bits!

```
Pseudocode:
  for k := 2 to n do
    C({k}, k) := $d_{1,k}$
  end for
```

```
Python:
# Set transition cost from initial state
for k in range(1, n):
    C[(1 << k, k)] = (dists[0][k], 0)
```

C[('subset', last node passed)] := (distance, starting node)

23



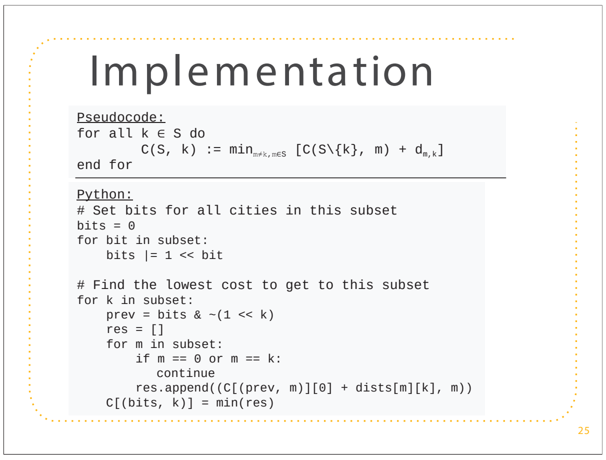
Implementation

```
Pseudocode:
for s := 2 to n-1 do
    for all S ⊆ {2, . . . , n}, |S| = s do
```

```
Python:
 for subset_size in range(2, n):
     for subset in it.combinations(range(1, n), subset_size):
```

Note: itertools isn't the fastest.

24



Implementation

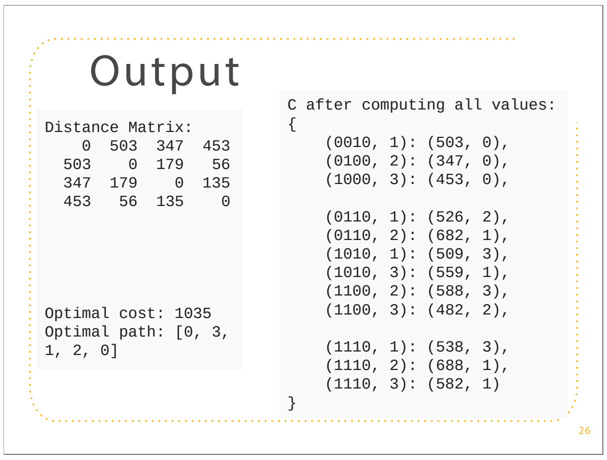
Pseudocode:

```
for all k ∈ S do
        C(S, k) := min_{m≠k,m∈S} [C(S\{k}, m) + d_{m,k}]
end for
```

Python:

```python
# Set bits for all cities in this subset
bits = 0
for bit in subset:
    bits |= 1 << bit

# Find the lowest cost to get to this subset
for k in subset:
    prev = bits & ~(1 << k)
    res = []
    for m in subset:
        if m == 0 or m == k:
           continue
        res.append((C[(prev, m)][0] + dists[m][k], m))
    C[(bits, k)] = min(res)
```

25

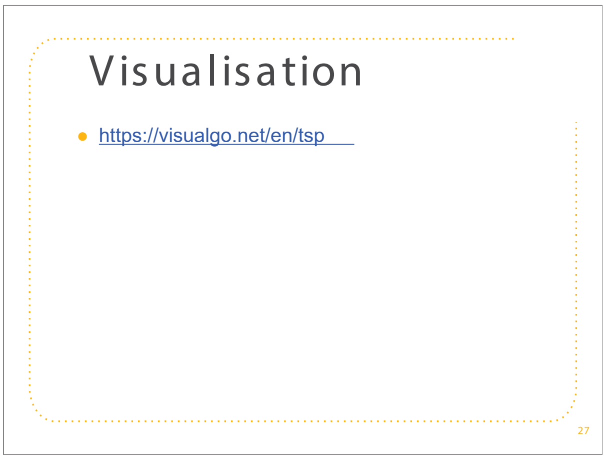


Output

```
Distance Matrix:
   0  503  347  453
 503    0  179   56
 347  179    0  135
 453   56  135    0


Optimal cost: 1035
Optimal path: [0, 3,
1, 2, 0]
```

```
C after computing all values:
{
     (0010, 1): (503, 0),
     (0100, 2): (347, 0),
     (1000, 3): (453, 0),

     (0110, 1): (526, 2),
     (0110, 2): (682, 1),
     (1010, 1): (509, 3),
     (1010, 3): (559, 1),
     (1100, 2): (588, 3),
     (1100, 3): (482, 2),

     (1110, 1): (538, 3),
     (1110, 2): (688, 1),
     (1110, 3): (582, 1)
}
```

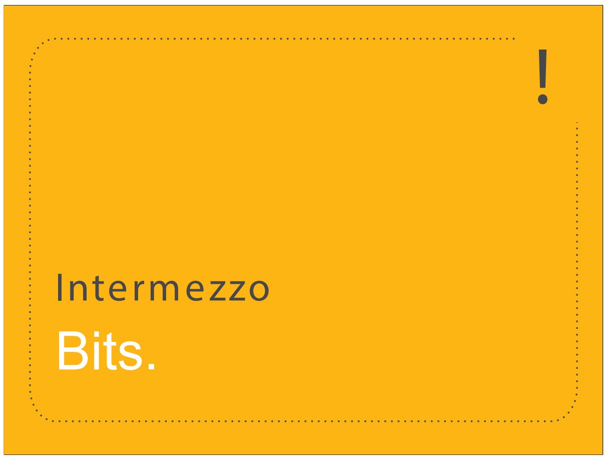
26



Visualisation

- https://visualgo.net/en/tsp

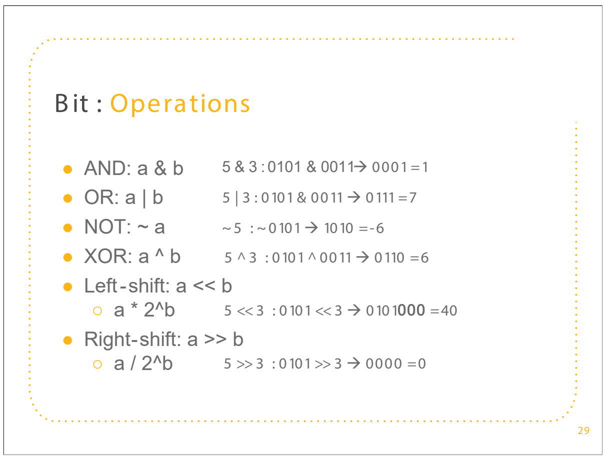
27



!

Intermezzo

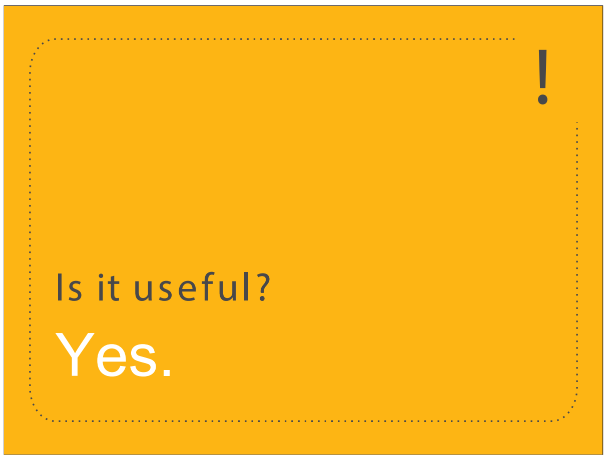
Bits.



Bit : Operations

- AND: a & b   5 & 3 : 0101 & 0011→ 0001 = 1
- OR: a | b   5 | 3 : 0101 & 0011 → 0111 = 7
- NOT: ~ a   ~5 : ~0101 → 1010 = -6
- XOR: a ^ b   5 ^ 3 : 0101 ^ 0011 → 0110 = 6
- Left-shift: a << b
  - a * 2^b   5 << 3 : 0101 << 3 → 0101000 = 40
- Right-shift: a >> b
  - a / 2^b   5 >> 3 : 0101 >> 3 → 0000 = 0

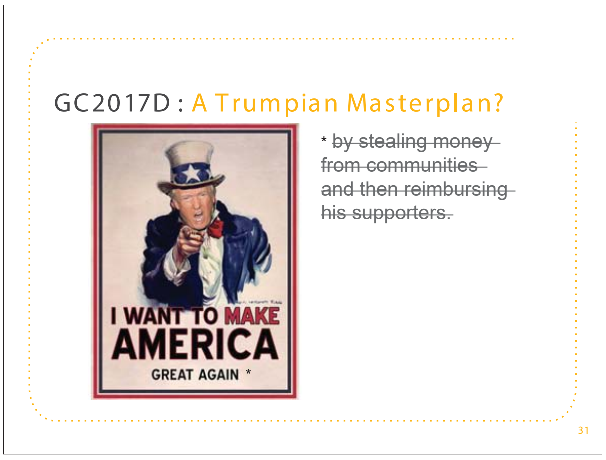
29



!

Is it useful?

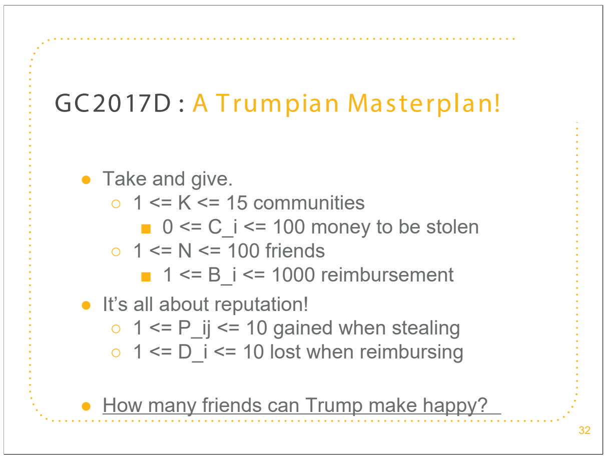
Yes.



GC2017D : A Trumpian Masterplan?

~~* by stealing money from communities and then reimbursing his supporters.~~

31

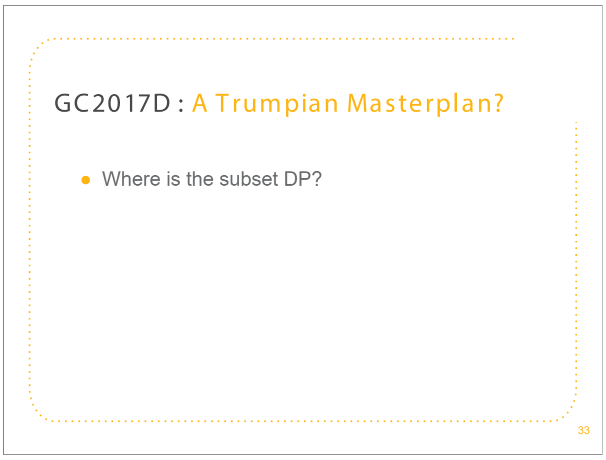


GC2017D : A Trumpian Masterplan!

- Take and give.
  - 1 <= K <= 15 communities
    - 0 <= C_i <= 100 money to be stolen
  - 1 <= N <= 100 friends
    - 1 <= B_i <= 1000 reimbursement
- It's all about reputation!
  - 1 <= P_ij <= 10 gained when stealing
  - 1 <= D_i <= 10 lost when reimbursing

- How many friends can Trump make happy?

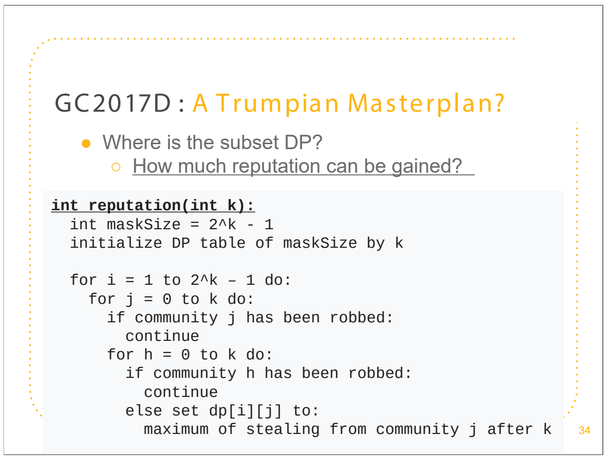
32



GC2017D : A Trumpian Masterplan?

- Where is the subset DP?

33

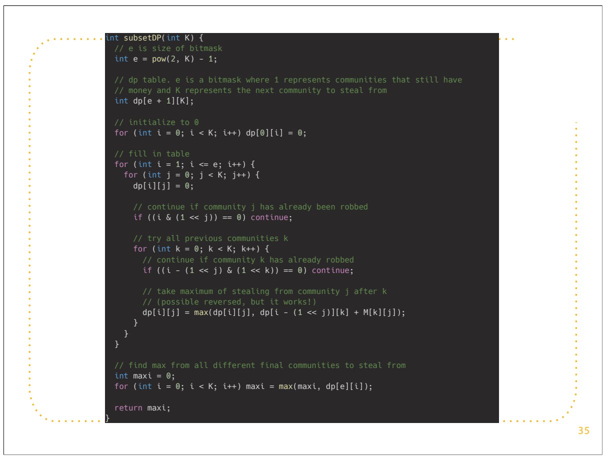


GC2017D : A Trumpian Masterplan?

- Where is the subset DP?
  - How much reputation can be gained?

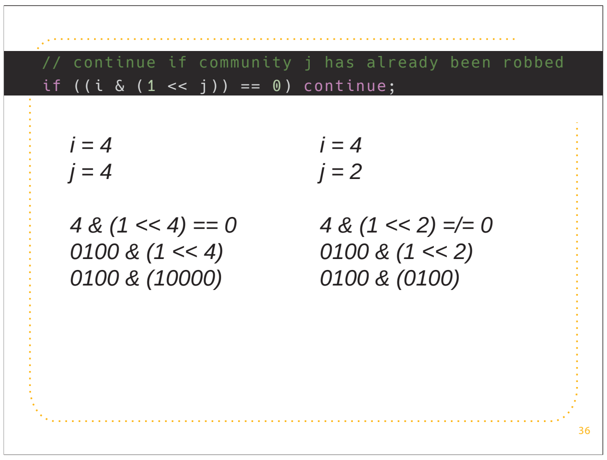
```
int reputation(int k):
  int maskSize = 2^k - 1
  initialize DP table of maskSize by k

  for i = 1 to 2^k – 1 do:
    for j = 0 to k do:
      if community j has been robbed:
        continue
      for h = 0 to k do:
        if community h has been robbed:
          continue
        else set dp[i][j] to:
          maximum of stealing from community j after k
```

34



```
int subsetDP(int K) {
  // e is size of bitmask
  int e = pow(2, K) - 1;

  // dp table. e is a bitmask where 1 represents communities that still have
  // money and K represents the next community to steal from
  int dp[e + 1][K];

  // initialize to 0
  for (int i = 0; i < K; i++) dp[0][i] = 0;

  // fill in table
  for (int i = 1; i <= e; i++) {
    for (int j = 0; j < K; j++) {
      dp[i][j] = 0;

      // continue if community j has already been robbed
      if ((i & (1 << j)) == 0) continue;

      // try all previous communities k
      for (int k = 0; k < K; k++) {
        // continue if community k has already robbed
        if ((i - (1 << j) & (1 << k)) == 0) continue;

        // take maximum of stealing from community j after k
        // (possible reversed, but it works!)
        dp[i][j] = max(dp[i][j], dp[i - (1 << j)][k] + M[k][j]);
      }
    }
  }

  // find max from all different final communities to steal from
  int maxi = 0;
  for (int i = 0; i < K; i++) maxi = max(maxi, dp[e][i]);

  return maxi;
}
```

35



```
// continue if community j has already been robbed
if ((i & (1 << j)) == 0) continue;
```

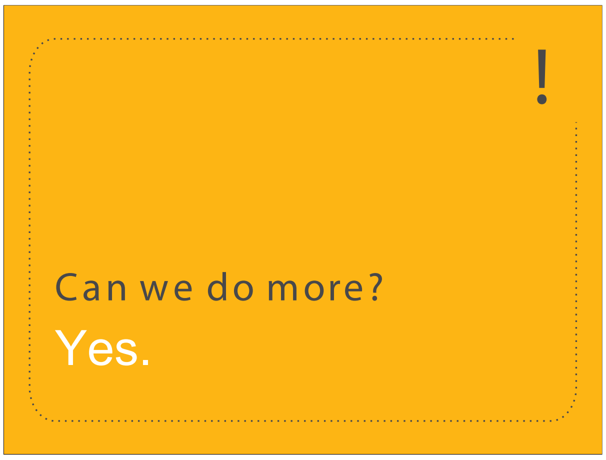
i = 4
j = 4

4 & (1 << 4) == 0
0100 & (1 << 4)
0100 & (10000)

i = 4
j = 2

4 & (1 << 2) =/= 0
0100 & (1 << 2)
0100 & (0100)

36

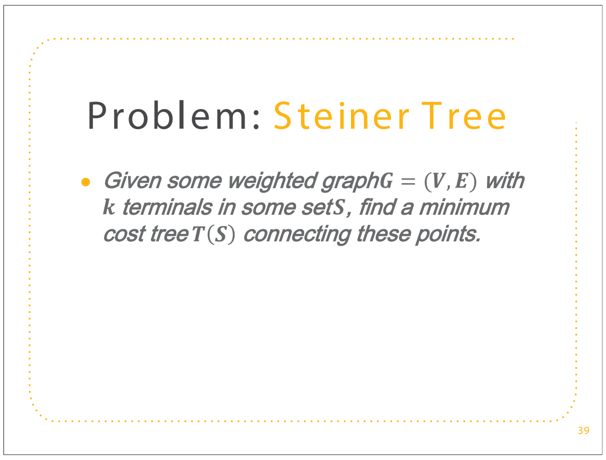

!

Can we do more?

Yes.

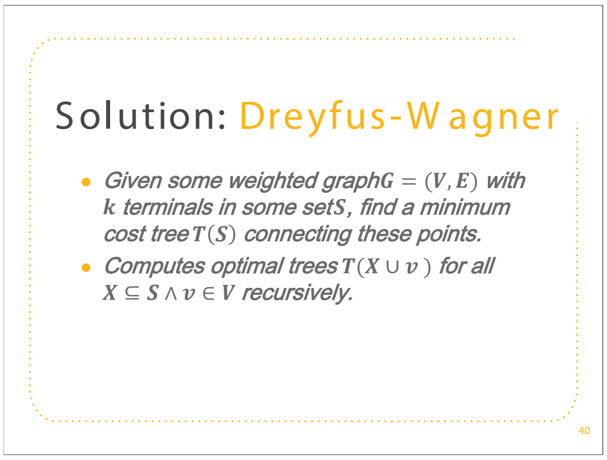
“

1. Steiner Tree
2. Treewidth


Problem: Steiner Tree

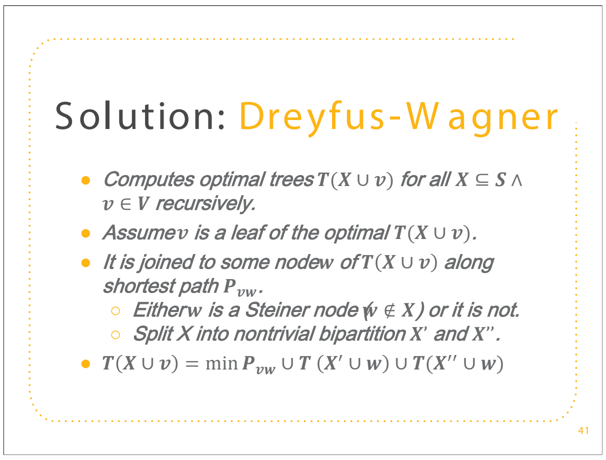
- *Given some weighted graph$G = (V, E)$ with k terminals in some set$S$, find a minimum cost tree $T(S)$ connecting these points.*

39


Solution: Dreyfus-Wagner

- *Given some weighted graph$G = (V, E)$ with k terminals in some set$S$, find a minimum cost tree $T(S)$ connecting these points.*
- *Computes optimal trees $T(X \cup v)$ for all $X \subseteq S \wedge v \in V$ recursively.*

40

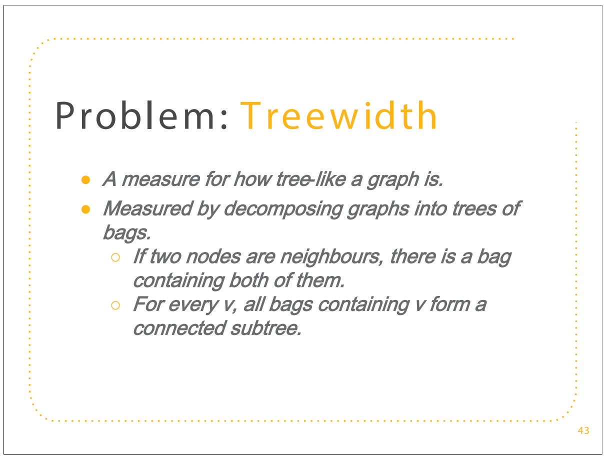

Solution: Dreyfus-Wagner

- *Computes optimal trees $T(X \cup v)$ for all $X \subseteq S \wedge v \in V$ recursively.*
- *Assume$v$ is a leaf of the optimal $T(X \cup v)$.*
- *It is joined to some node$w$ of $T(X \cup v)$ along shortest path P_{vw}.*
  - *Either$w$ is a Steiner node ($w \notin X$) or it is not.*
  - *Split X into nontrivial bipartition X' and X''.*
- $T(X \cup v) = \min P_{vw} \cup T(X' \cup w) \cup T(X'' \cup w)$

41

“

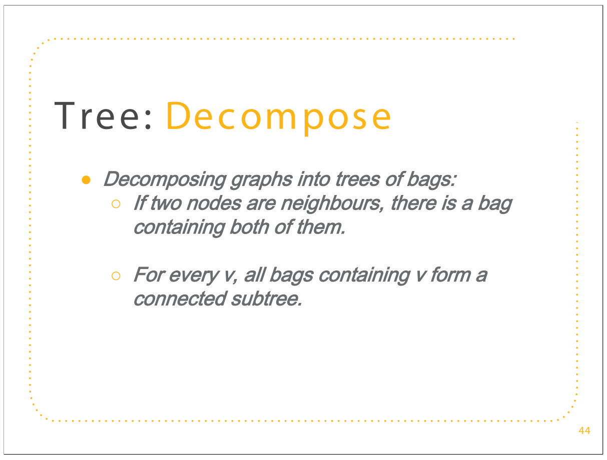
1. Steiner Tree
2. Treewidth



Problem: Treewidth

- *A measure for how tree-like a graph is.*
- *Measured by decomposing graphs into trees of bags.*
  - *If two nodes are neighbours, there is a bag containing both of them.*
  - *For every v, all bags containing v form a connected subtree.*

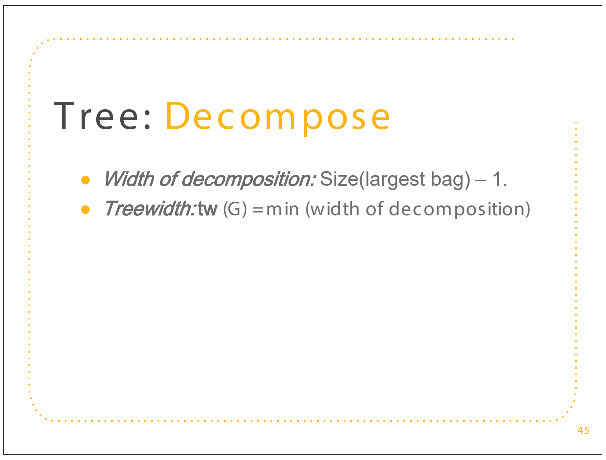
43



Tree: Decompose

- *Decomposing graphs into trees of bags:*
  - *If two nodes are neighbours, there is a bag containing both of them.*
  - *For every v, all bags containing v form a connected subtree.*

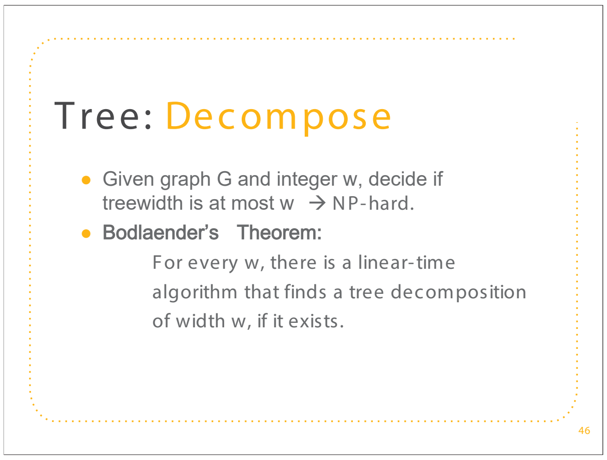
44



Tree: Decompose

- *Width of decomposition:* Size(largest bag) – 1.
- *Treewidth:*tw (G) =min (width of decomposition)

45

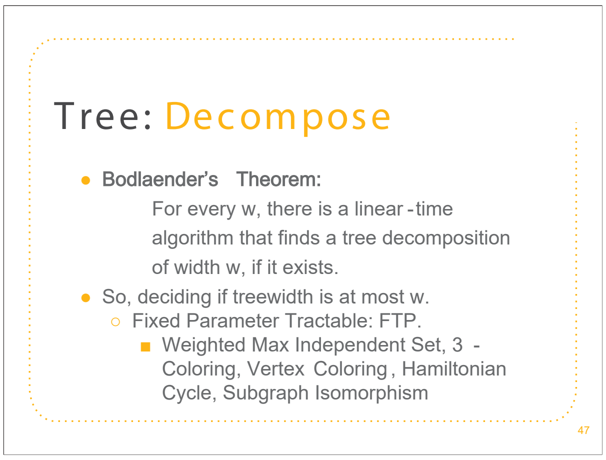


Tree: Decompose

- Given graph G and integer w, decide if treewidth is at most w → NP-hard.
- Bodlaender's Theorem:

  For every w, there is a linear-time algorithm that finds a tree decomposition of width w, if it exists.

46

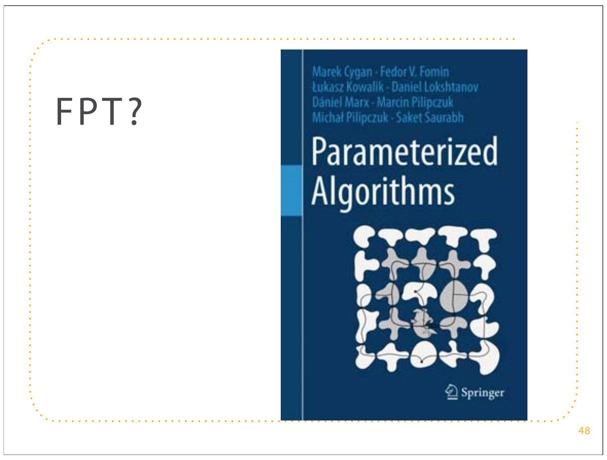


Tree: Decompose

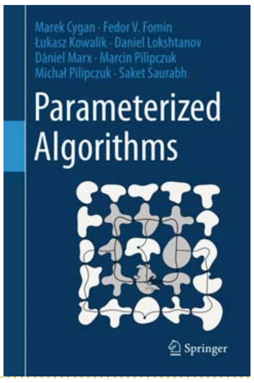
- Bodlaender's Theorem:

  For every w, there is a linear -time algorithm that finds a tree decomposition of width w, if it exists.
- So, deciding if treewidth is at most w.
  - Fixed Parameter Tractable: FTP.
    - Weighted Max Independent Set, 3 - Coloring, Vertex Coloring, Hamiltonian Cycle, Subgraph Isomorphism

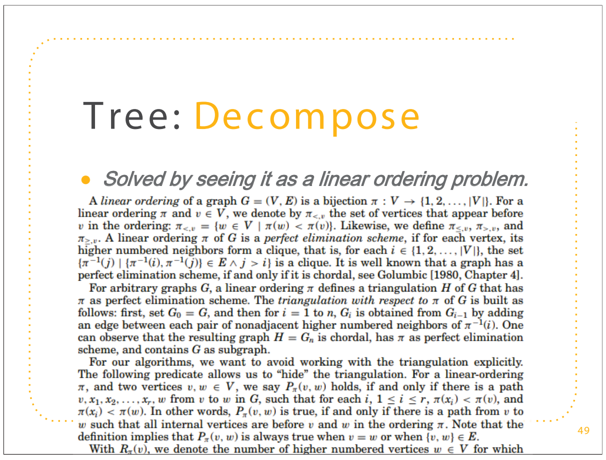
47



FPT?


Marek Cygan · Fedor V. Fomin
Łukasz Kowalik · Daniel Lokshtanov
Dániel Marx · Marcin Pilipczuk
Michał Pilipczuk · Saket Saurabh
Parameterized Algorithms
Springer

48



Tree: Decompose

- *Solved by seeing it as a linear ordering problem.*

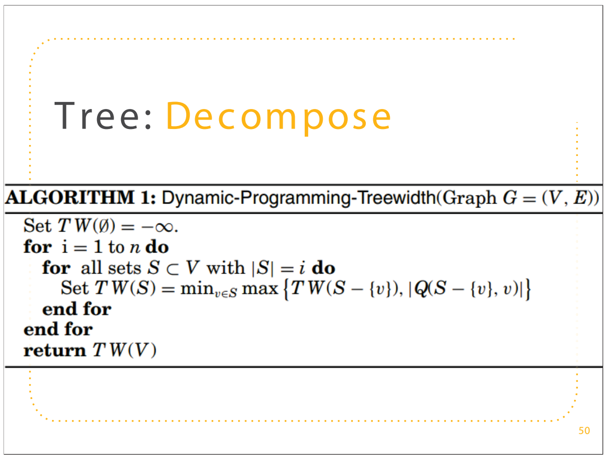
A *linear ordering* of a graph $G = (V, E)$ is a bijection $\pi : V \to \{1, 2, \ldots, |V|\}$. For a linear ordering π and $v \in V$, we denote by $\pi_{<,v}$ the set of vertices that appear before v in the ordering: $\pi_{<,v} = \{w \in V \mid \pi(w) < \pi(v)\}$. Likewise, we define $\pi_{\leq,v}$, $\pi_{>,v}$, and $\pi_{\geq,v}$. A linear ordering π of G is a *perfect elimination scheme*, if for each vertex, its higher numbered neighbors form a clique, that is, for each $i \in \{1, 2, \ldots, |V|\}$, the set $\{\pi^{-1}(j) \mid \{\pi^{-1}(i), \pi^{-1}(j)\} \in E \wedge j > i\}$ is a clique. It is well known that a graph has a perfect elimination scheme, if and only if it is chordal, see Golumbic [1980, Chapter 4].

For arbitrary graphs G, a linear ordering π defines a triangulation H of G that has π as perfect elimination scheme. The *triangulation with respect to* π of G is built as follows: first, set $G_0 = G$, and then for $i = 1$ to n, G_i is obtained from G_{i-1} by adding an edge between each pair of nonadjacent higher numbered neighbors of $\pi^{-1}(i)$. One can observe that the resulting graph $H = G_n$ is chordal, has π as perfect elimination scheme, and contains G as subgraph.

For our algorithms, we want to avoid working with the triangulation explicitly. The following predicate allows us to "hide" the triangulation. For a linear-ordering π, and two vertices $v, w \in V$, we say $P_\pi(v, w)$ holds, if and only if there is a path $v, x_1, x_2, \ldots, x_r, w$ from v to w in G, such that for each i, $1 \leq i \leq r$, $\pi(x_i) < \pi(v)$, and $\pi(x_i) < \pi(w)$. In other words, $P_\pi(v, w)$ is true, if and only if there is a path from v to w such that all internal vertices are before v and w in the ordering π. Note that the definition implies that $P_\pi(v, w)$ is always true when $v = w$ or when $\{v, w\} \in E$.

With $R_\pi(v)$, we denote the number of higher numbered vertices $w \in V$ for which

49

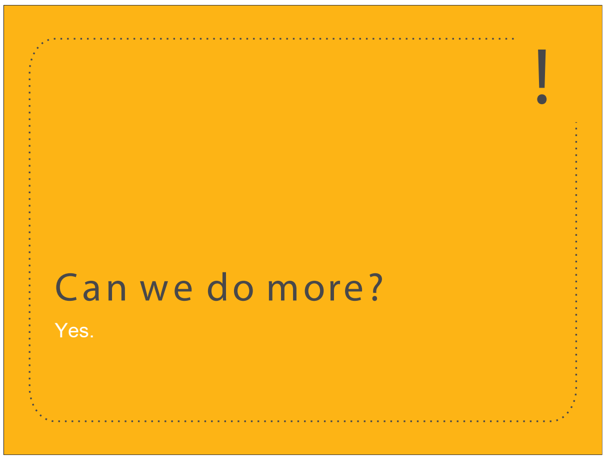


Tree: Decompose

ALGORITHM 1: Dynamic-Programming-Treewidth(Graph $G = (V, E)$)

```
Set $TW(\emptyset) = -\infty$.
for i = 1 to $n$ do
  for all sets $S \subset V$ with $|S| = i$ do
    Set $TW(S) = \min_{v \in S} \max \{TW(S - \{v\}), |Q(S - \{v\}, v)|\}$
  end for
end for
return $TW(V)$
```

50



!

Can we do more?

Yes.

Bibliography

[1] Wikipedia contributors. (2018, May 21). Competitive programming. In *Wikipedia, The Free Encyclopedia*. Retrieved June 1, 2018, from `https://en.wikipedia.org/w/index.php?title=Competitive_programming&oldid=842227680`