Eidgenössische Technische Hochschule Zürich
Ecole polytechnique fédérale de Zurich
Politecnico federale di Zurigo
Federal Institute of Technology at Zurich

Department Informatik
Markus Püschel
David Steurer

Johannes Lengler
Alen Stojanovic
Tyler Smith
Gleb Novikov

Exam
Algorithmen und Datenstrukturen
D-INFK

August 24, 2019

DO NOT OPEN!

Last name, first name: ______________________

Student number: ______________________

With my signature I confirm that I can participate in the exam under regular conditions. I will act honestly during the exam, and I will not use any forbidden means.

Signature: ______________________

Good luck!

Score:

P1	P2	T1	T2	T3	**Σ**

/ 20 P

Programming Task P1.

Enrollment Key: `ExamTime2019`

Submission: see Section 3 of the Technical Guide

Grid

Given is a square grid of size $N \times N$. Rows and columns are indexed from 0 to $N-1$ and each cell at index (row, column) contains a positive value (its cost) as shown in the example below.

You are asked to traverse the grid from the top row to the bottom row, one row at a time. In each step, you can move from a cell (i, j) in row i to either of the cells $(i+1, j-1)$, $(i+1, j)$, or $(i+1, j+1)$ in row $i+1$ as depicted below. As start you can pick any cell in the first row. The overall cost of a traversal is the sum of the costs of all visited cells, including start and end.

Devise an algorithm to find a path from top to bottom of the grid that minimizes the overall traversal cost.

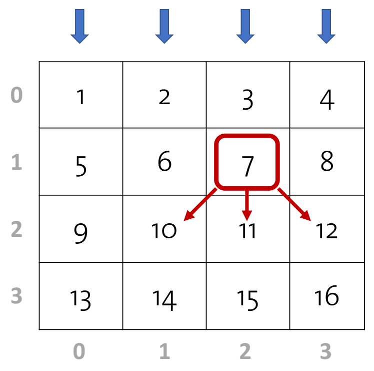
Complete the implementation of the following method:

- **`solveGrid(`*grid*`)`**: calculates the minimal cost to traverse the grid from top to bottom.

To test your implementation, the computed cost is written to the output. Your implementation can assume an $N \times N$ grid, where $1 \leq N \leq 10^3$, and the cost of each cell is an integer p with $1 \leq p \leq 10^6$.

Example

Consider a 4×4 grid, as illustrated below. You can start the traversal at each of the 4 columns, beginning at cell $(0, 0)$, $(0, 1)$, $(0, 2)$ or $(0, 3)$.



From each cell (i, j) you can move to cell $(i+1, j)$, $(i+1, j-1)$ or $(i+1, j+1)$. For example from cell $(1, 2)$, you can either move to cell $(2, 1)$, $(2, 2)$ or $(2, 3)$, as illustrated above.

The minimal cost to reach the bottom of the grid is through cells $(0, 0) \rightarrow (1, 0) \rightarrow (2, 0) \rightarrow (3, 0)$, and the price is $1 + 5 + 9 + 13 = 28$.

Grading

Overall, you can obtain a maximum of 20 judge points for this programming task. Assuming $N \times N$ is the size of the grid, you can obtain up to:

- **20 points** for time $O(N^2)$ solution.
- **15 points** for time $O(N^2 \cdot \log(N))$ solution.
- **10 points** for time $O(N \cdot 3^N)$ solution.

We assume that each solution is provided with reasonable hidden constants.

Instructions

For this exercise, we provide a program template as an Eclipse project in your workspace that helps you reading the input and writing the output. Importing any additional Java class is **not allowed** (with the exception of the already imported ones `java.io.{InputStream, OutputStream}` and `java.util.Scanner` class).

The project also contains data for your local testing and a `JUnit` program that runs your `Main.java` on all the local tests – just open and run `GridTest.launch` in the project. The local test data are different and generally smaller than the data that are used in the online judge.

Submit only your `Main.java`.

The input and output are handled by the template – you should not need the rest of this text.

Input The input of this problem consists of a number of test-cases. The first line contains T, the number of test-cases. Each of the T cases is independent of the others, and contains several lines:

1. The first line of each test case contains the amount of rows/columns of the grid $N \in \{1, \ldots, 10^3\}$.
2. Then it is followed by $N \times N$ numbers, that correspond to the the price $p \in \{1, \ldots, 10^6\}$ at each cell in the grid. The numbers are distributed in N lines, each having N integers separated by one or several white space characters. Each such line corresponds to one row of the grid.

Output For every case, the output is the minimal cost of the traversing the grid from top to bottom.

The output contains one line for each test-case. More precisely, the i-th line of the output contains an integer number that represents the minimal cost of traversing the grid from top to bottom. The output is terminated with an end-line character.

Example input:

```
2
4
1   2   3   4
5   6   7   8
9   10  11  12
13  14  15  16
4
14  7   9   16
4   10  12  13
5   1   2   15
8   6   11  3
```

Example output:

```
28
18
```

Space for your notes. These will not be graded. Only what was submitted to the judge counts for this exercise.

Space for your notes. These will not be graded. Only what was submitted to the judge counts for this exercise.

/ 20 P

Programming Task P2.

Enrollment Key: `ExamTime2019`

Submission: see Section 3 of the Technical Guide

Structure

In this exercise, you should implement an algorithm to find the minimum spanning tree (MST) for a given graph $G = (V, E)$, where G is a *connected*, *weighted*, *undirected* and *simple* graph. One way of finding the MST of a given graph is by using Kruskal's algorithm. Your task is to complete the implementation of the following method:

- **`kruskal(`G`)`**: finds a minimum spanning tree for a given graph $G = (V, E)$, and outputs its cost, i.e., the sum of the weights of all edges in the computed MST.

We provide a template for this exercise (along with code to read the input and write the cost of the MST to the output) where you can provide the implementation of the algorithm.

A successful implementation of Kruskal's algorithm depends on a correct and efficient implementation of a **Union-Find** structure. A Union-Find data structure maintains a *family* of N disjoint nonempty sets (one set per connected component). Each set in the family has one designated element called its *label* and is identified by that label. The data structure supports the following three operations:

1. **`create(`N`)`**: creates a new set $\{x\}$ for all $x \in \{0, \ldots, N-1\}$ and adds it to the family.
2. **`union(`x`, `y`)`**: changes the family by replacing two sets, the one containing x and the one containing y, by a single set that is the union of these two sets. The label of the union is either `find(`x`)` or `find(`y`)`.
3. **`find(`x`)`**: returns the label of the set containing x.

In the code template we also provide a correct, but **inefficient** implementation of the Union-Find structure. To obtain full points for this exercise, your task is to modify the implementation of this structure such that:

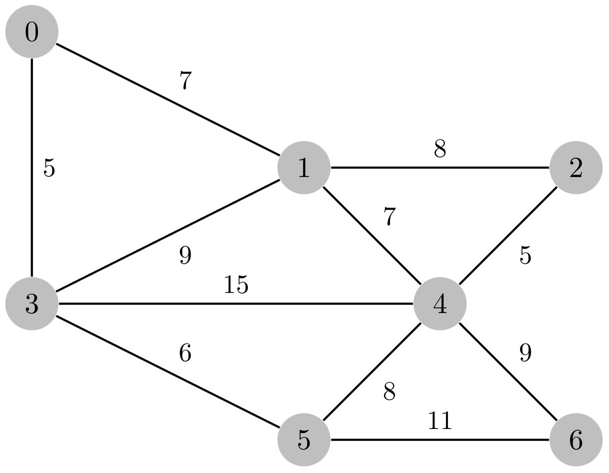
- **`find(`x`)`** and **`union(`x`, `y`)`** operate in time $O(\log(N))$ (amortized over one run of the algorithm), where N is the number of elements stored in the family.
- **`create(`N`)`** initializes the structure and any additional fields that might be used in the `find(`x`)` and `union(`x`, `y`)` routines. The modified routine should operate in time $O(N)$, where N is number of elements stored in the family.

The provided template is tailored towards the implementation of the Kruskal's algorithm. However, any other algorithm that will compute the MST in $O(|E| \cdot \log(|V|))$ will also be accepted, assuming it passes all automatic tests.

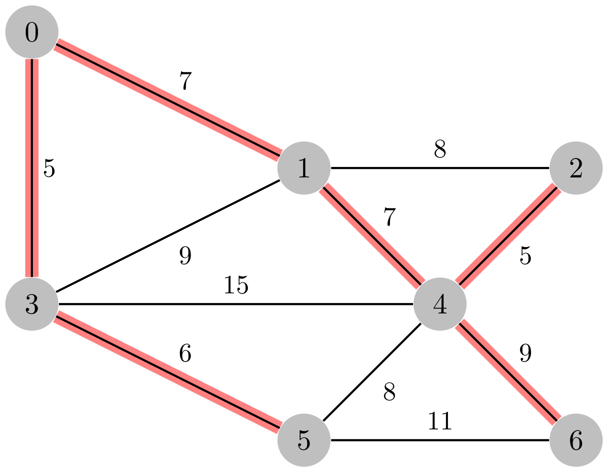
To validate the correctness of your implementation, the cost of the computed MST is written to the output. Your implementation can assume $|V|$ vertices, such that $2 \leq |V| \leq 10^6$, and $|E|$ edges such that $|V| - 1 \leq |E| \leq 10^6$. The weight of each edge is specified with an integer number w such that $1 \leq w \leq 10^6$.

Example

Consider the graph below with 7 vertices and 11 edges (weights are specified next to each edge in the graph):



A minimum spanning tree is shown with the highlighted edges:



The cost of the tree is the sum of edges $\{0,3\},\{3,5\},\{0,1\},\{1,4\},\{2,4\}$ and $\{4,6\}$, which is $5+6+7+7+5+9=39$.

Grading

Overall, you can obtain a maximum of 20 judge points for this programming task. To get full points your program should require $O(|E|\cdot\log(|V|))$ time to compute the cost of the MST of the graph (with reasonable hidden constants).

Slower solutions can obtain partial points, namely an $O(|E|\cdot|V|)$ solution can obtain up to 10 points (with reasonable hidden constants). This solution can be achieved by implementing Kruskal's algorithm with the provided Union-Find data-structure.

Instructions and assistance for programming task P2

For this exercise, we provide a program template as an Eclipse project in your workspace that helps you reading the input and writing the output. Importing any additional Java class is **not allowed** (with the exception of the already imported ones `java.io.{InputStream, OutputStream}` and `java.util.{Arrays, Comparator, Scanner}` class).

In this exercise we permit the use of `java.util.Arrays` library, that provides various methods for manipulating arrays (such as sorting and searching). One such function is the `Arrays.sort` method that provides efficient sorting based on merge-sort in $O(N \cdot \log(N))$ time:

```
Arrays.sort(T[] a, Comparator<? super T> c)
```

The methods works such that it sorts the specified array of objects according to the order induced by the specified comparator. To illustrate usage, consider the class *Employee* that represents employees with their name, age and salary:

```
public class Employee {
  public String name;
  public int age;
  public int salary;
  public Employee(String name, int age, int salary) {
    this.name   = name;
    this.age    = age;
    this.salary = salary;
  }
}
```

To sort the array of employees `listOfEmployees` by their salaries, we can invoke the method providing a `Comparator` instance, that compares the salaries:

```
public void sortEmployeesBySalary (Employee[] listOfEmployees) {
  //
  // Create an instance of Comparator for a given Employee class
  //
  Comparator<Employee> employeeComparator = new Comparator<Employee>() {
    //
    // Implement the comparison routine of two Employee instances
    //
    public int compare(Employee e1, Employee e2) {
      return e1.salary - e2.salary;
    }
  }
  //
  // Use the Comparator instance to perform the sorting.
  //
  Arrays.sort(listOfEmployees, employeeComparator);
}
```

Use the `Array.sort` or any other available method in `java.util.Arrays` on your convenience.

The project also contains data for your local testing and a `JUnit` program that runs your `Main.java` on all the local tests – just open and run `StructureTest.launch` in the project. The local test data are different and generally smaller than the data that are used in the online judge.

Submit only your `Main.java`.

The input and output are handled by the template – you should not need the rest of this text.

Input The input of this problem consists of a number of test-cases. The first line contains T, the number of test-cases. Each of the T cases is independent of the others, and contains several lines:

1. The first line of each test case contains the number of vertices in the graph $|V| \in \{2, \ldots, 10^6\}$ and the number of edges $|E| \in \{|V| - 1, \ldots, 10^6\}$.
2. Then it is followed by E lines, such that each contains 3 integer numbers: u, v and w. The numbers u, $v \in \{0, \ldots, |V| - 1\}$ (such that $u \neq v$) represent the end points of an edge and $w \in \{1, \ldots, 10^6\}$ represents its weight. As we have undirected and simple graph each (u, v) pair is commutative (i.e. (u, v) and (v, u) represent the same edge $\{u, v\}$), and each pair is distinct (the input does not contain the same edge more than once).

Output For every case, the output is the cost of the minimum spanning tree of the input graph.

The output contains one line for each test-case. More precisely, the i-th line of the output contains an integer number that represents the cost of the minimum spanning tree of the input graph. The output is terminated with an end-line character.

Example input:

```
2
7 11
1 0 7
2 1 8
3 0 5
3 1 9
4 1 7
4 2 5
4 3 15
5 3 6
5 4 8
6 4 9
6 5 11
4 6
0 1 3
0 2 2
0 3 5
1 2 4
1 3 6
2 3 1
```

Example output:

```
39
6
```

Space for your notes. These will not be graded. Only what was submitted to the judge counts for this exercise.

Space for your notes. These will not be graded. Only what was submitted to the judge counts for this exercise.

Theory Task T1. / 14 P

Notes:

1) In this problem, you have to provide **solutions only**. You should write them directly on this sheet.
2) We assume letters to be ordered alphabetically and numbers to be ordered ascendingly, according to their values.

/ 1 P

a) *Min-Heap*: Draw the Min-Heap that is obtained when inserting into an empty heap the keys `8, 3, 2, 7, 4, 1` in this order.

Solution:

$$\begin{tikzcd}
 & & & 1 \arrow[dll] \arrow[drr] & & \\
 & 4 \arrow[dl] \arrow[dr] & & & & 2 \arrow[dl] \\
8 & & 7 & & 3 &
\end{tikzcd}$$

/ 1 P

b) *Max-Heap*: Draw the resulting Max-Heap obtained from the following Max-Heap by performing the operation `DELETE-MAX` **twice**.

$$\begin{tikzcd}
 & & & & & & 20 \arrow[dlll] \arrow[drr] & & & \\
 & & & 12 \arrow[dll] \arrow[drr] & & & & & 10 \arrow[dl] \arrow[dr] & \\
 & 7 \arrow[dl] \arrow[dr] & & & & 11 \arrow[dl] & & 8 & & 1 \\
5 & & 6 & & 9 & & & & &
\end{tikzcd}$$

Solution:

$$\begin{tikzcd}
 & & & & 11 \arrow[dll] \arrow[drr] & & & \\
 & & 9 \arrow[dl] \arrow[dr] & & & & 10 \arrow[dl] \arrow[dr] & \\
 & 7 \arrow[dl] & & 6 & & 8 & & 1 \\
5 & & & & & & &
\end{tikzcd}$$

/ 1 P

c) *Bubblesort*: Bubblesort is implemented as two nested loops. Draw how the following array looks like after the first iteration of bubblesort's outer loop.

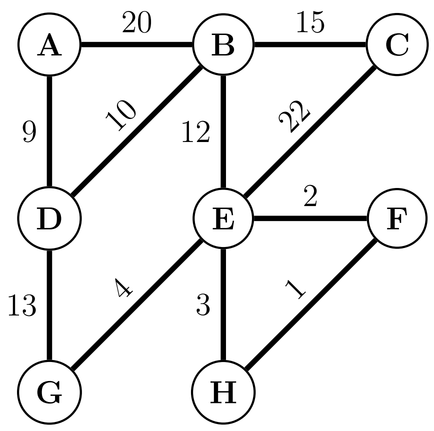
19 18 20 9 7 33 1 2 6 5

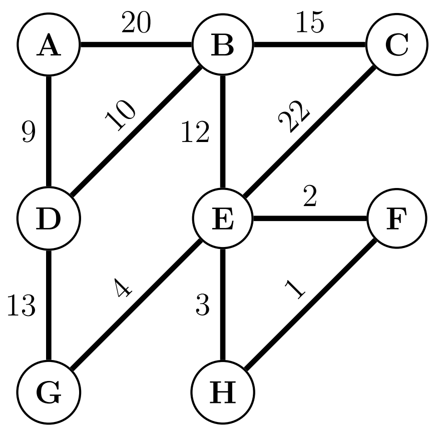
Solution:

18 19 9 7 20 1 2 6 5 33

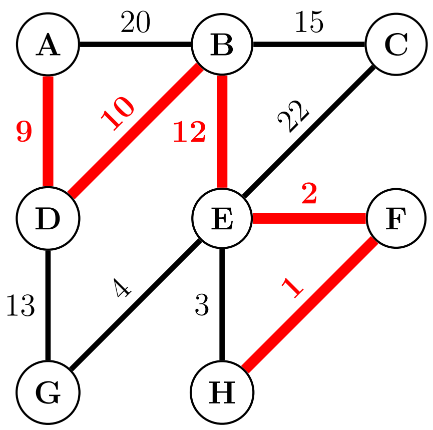
/ 2 P

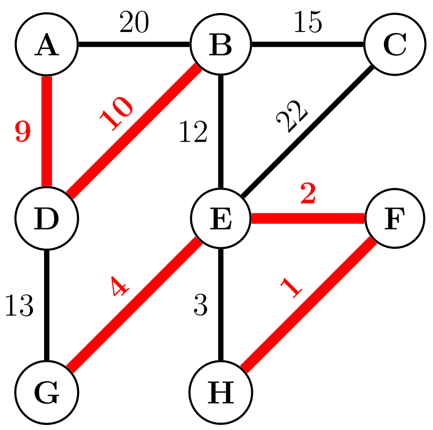
d) Below you find two copies of the same graph. In the left graph, highlight the first 5 edges that Prim's algorithm adds to the minimum spanning tree, when starting at iftogglesolution *B*. In the right graph, highlight the first 5 edges that Kruskal's algorithm adds to the minimum spanning tree.





Solution:





/ 2 P

e) For each of the following claims, state whether it is true or false. You get 0.5P for a correct answer, -0.5P for a wrong answer, 0P for a missing answer. You get at least 0 points in total.

claim	true	false
$\frac{n}{\log n} \geq \Omega(n^{1/2})$	☐	☐
$\log_7(n^8) = \Theta(\log_3(n^{\sqrt{n}}))$	☐	☐
$3n^4 + n^2 + n \geq \Omega(n^2)$	☐	☐
$n! \leq O(n^{n/2})$	☐	☐

Solution:

claim	true	false
$\frac{n}{\log n} \geq \Omega(n^{1/2})$	☒	☐
$\log_7(n^8) = \Theta(\log_3(n^{\sqrt{n}}))$	☐	☒
$3n^4 + n^2 + n \geq \Omega(n^2)$	☒	☐
$n! \leq O(n^{n/2})$	☐	☒

/ 3 P f) Let $T : \mathbb{N} \to \mathbb{R}$ be a function that satisfies the following two conditions:

$$\begin{aligned} &T(n) \geq 4 \cdot T(\tfrac{n}{2}) + 3n \qquad \text{whenever } n \text{ is divisible by 2;} \\ &T(1) = 4. \end{aligned}$$

Prove by mathematical induction that

$$T(n) \geq 6n^2 - 2n$$

holds whenever n is a power of 2, i.e., $n = 2^k$ with $k \in \mathbb{N}_0$.

Solution:

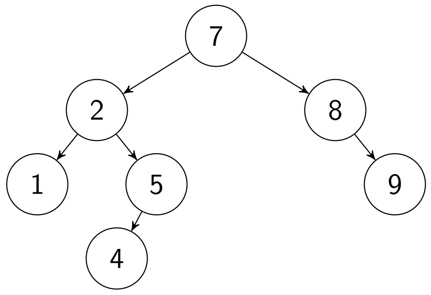
Base case (n=1): *(1pt)*
$T(1) = 4 \geq 6n^2 - 2n = 6 - 2 = 4$
Induction hypothesis: *(1pt also for mentioning in the induction step where it was used)*
$T(n) \geq 6n^2 - 2n$ for an arbitrary but fixed n with $n = 2^k$ and $k \in \mathbb{N}_0$.
Induction step ($n \to 2n$): *(1pt)*

$$\begin{aligned} T(2n) &\geq 4 \cdot T(\tfrac{2n}{2}) + 3 \cdot 2n \\ &= 4 \cdot T(n) + 6n \\ &\overset{IH}{\geq} 4 \cdot (6n^2 - 2n) + 6n \\ &= 24n^2 - 8n + 6n \\ &= 24n^2 - 2n \\ &\geq 24n^2 - 4n \\ &= 6 \cdot (2n)^2 - 2 \cdot (2n) \end{aligned}$$

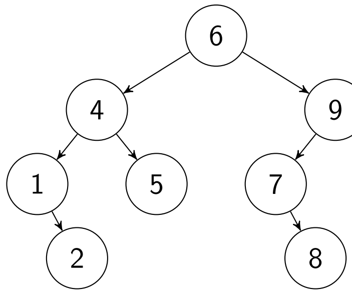
■

/ 1 P g) *Binary search trees*: Draw the binary search tree that is obtained when inserting into an empty tree the keys `7`, `2`, `1`, `5`, `4`, `8`, `9` in this order.

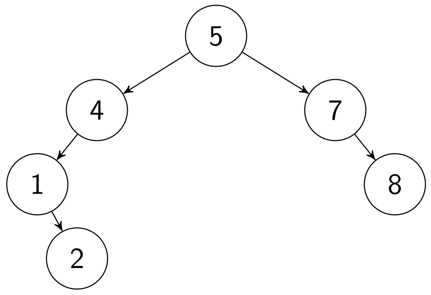
Solution:



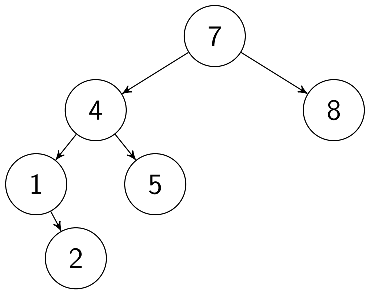
/ 1 P h) *Binary search trees*: Draw the resulting binary search tree obtained by deleting the keys `6` and `9` in this order from the following binary search tree.



Solution:

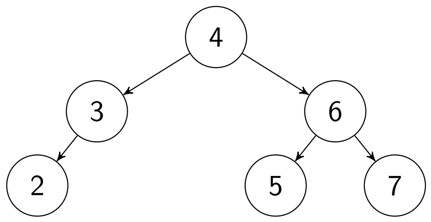


or:



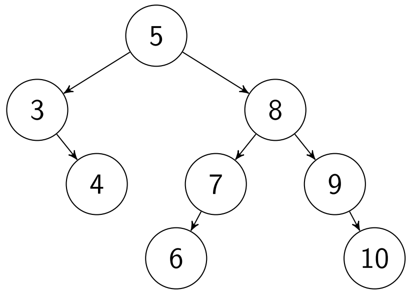
/ 1 P i) *AVL-trees*: Draw the AVL-tree that is obtained when inserting in an empty tree the keys `7`, `5`, `6`, `4`, `3`, `2` in this order.

Solution:

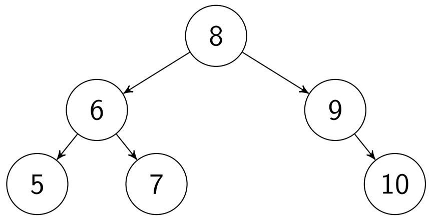


/ 1 P

j) *AVL-trees*: Draw the resulting AVL-tree obtained by deleting the keys `4` and `3` in this order from the following AVL-tree.



Solution:



Extra space. Please indicate clearly to which task your notes belong. Please cross out all notes that you do not want to be graded.

Extra space. Please indicate clearly to which task your notes belong. Please cross out all notes that you do not want to be graded.

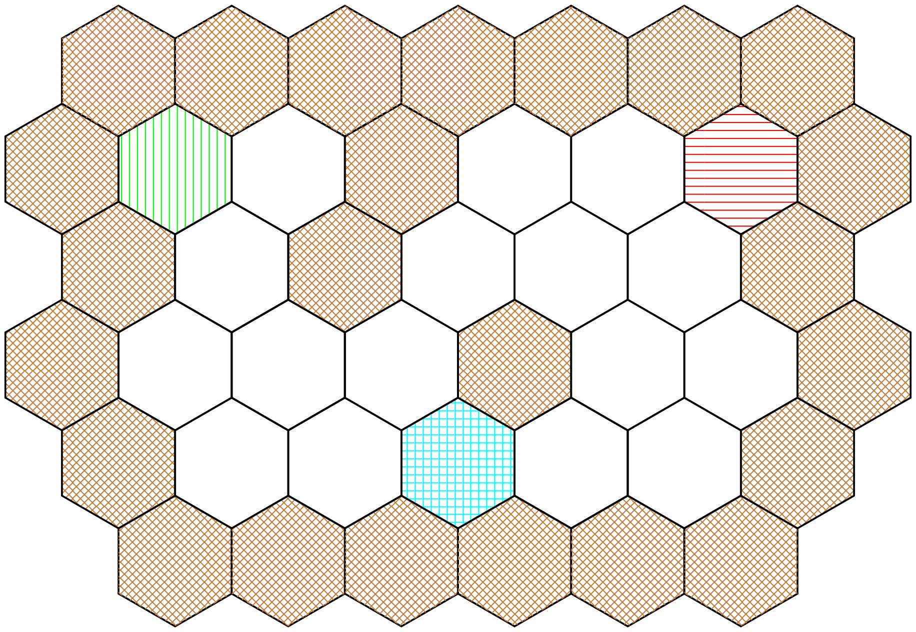
Theory Task T2.

/ 13 P

You are alone and stranded in a desert trying to reach the only city. The desert can be divided into hexagons and is surrounded by impassable cliffs. The desert terrain varies, and hexagons take differing amounts of time to traverse. There are oases in the desert.

- In this problem, you move from the center of one hexagon to the center of an adjacent hexagon.
- The amount of time spent traveling from the center of one hexagon to another is equal to half the amount of time needed to cross the first hexagon plus half the amount of time needed to cross the second.

Example:



Legend:

▦	Oasis	▥	Starting Location
▩	Impassable	▤	City

/ 6 P

a) Your goal is to minimize the amount of time it takes to get from the starting location to the city. Model the problem with a graph in such a way that an algorithm from the lecture computes a solution to the problem. Describe precisely the set of vertices and edges, and describe what the vertices and edges represent. Name an, as efficient as possible, algorithm from the lecture to solve this problem, and state the running time of the algorithm in terms of the number of hexagons, N.

Definition of the graph (if possible, in words and not formal):
We define an undirected graph with edge weights. The vertices are the hexagons, and there is an edge between two hexagons if and only if they are adjacent. The weight of an edge is equal to half the amount of time needed to cross one hexagon plus half the amount of time needed to cross the other.

Number of vertices and edges (in Θ-Notation in N, as concisely as possible):
There are N vertices. As each vertex has degree at most 6 and the graph is connected, there are between $6N/2 = 3N$ and $N-1$ edges. Therefore, there are $\Theta(N)$ vertices and $\Theta(N)$ edges.

Algorithm, as efficient as possible:
As all edge weights are non-negative travel times, we can use Dijkstra's algorithm for shortest paths to compute a shortest path from the starting location to the city.

Running time (as concisely as possible in Θ notation in terms of N). Justify your answer:
The (worst-case) running time of Dijkstra's algorithm for shortest paths is $\Theta(|V|\log|V|+|E|)$. (Note that this worst case can occur on a graph of our form.) We have $|V| = \Theta(N)$ and $|E| = \Theta(N)$. Therefore, the running time is $\Theta(N \log N)$.

/ 3 P

b) We now modify the problem, adding the following rules:

- You begin a full water bottle that you can drink from D times, and you can refill it at each *oasis*.
- When you arrive at a hexagon that isn't an oasis or the city, you must drink water from your water bottle.
- If your water bottle is empty, you become dehydrated.

Your goal is to find the fastest route from the starting location to the city, without becoming dehydrated.

Model the modified problem with a graph in such a way that an algorithm from the lecture computes a solution to the problem. Describe precisely the set of vertices and edges, and describe what the vertices and edges represent. Name an, as efficient as possible, algorithm from the lecture to solve this problem, and state the running time of the algorithm in terms of the number of hexagons, N, and the size of your water bottle, D.

Definition of the graph (if possible, in words and not formal):
We define a directed graph with edge weights. The vertices are pairs of hexagons and bottle states. A bottle state is a number from 1 to D, representing the number of times that you can still drink from the bottle. There is an edge from one vertex to another vertex if the respective hexagons are adjacent and traveling from the first hexagon with the first bottle state to the second hexagon results in the second bottle state (we always refill the bottle when we reach an oasis).

The weight of an edge is again equal to half the amount of time needed to cross the first hexagon plus half the amount of time needed to cross the second.

Number of vertices and edges (in Θ-Notation in N and D, as concisely as possible):
We have $\Theta(N \cdot D)$ vertices and $\Theta(N \cdot D)$ edges.

Algorithm, as efficient as possible:
As the edge weights are still non-negative travel times, we can again use Dijkstra's algorithm for shortest paths to compute a shortest path from the starting location with bottle state D to the city with arbitrary bottle state.

Running time (as concisely as possible in Θ notation in terms of N and D). Justify your answer:
The (worst-case) running time of Dijkstra's algorithm is $\Theta(|V| \log|V| + |E|)$. (Note that this worst case can occur on a graph of our form.) We have $|V| = \Theta(N \cdot D)$ and $|E| = \Theta(N \cdot D)$. Therefore, the running time is $\Theta(N \cdot D \cdot \log(N \cdot D))$.

/ 4 P

c) We now further modify the problem from (b) so that there are magic portals at the center of some hexagons in the desert. When you enter such a portal, it teleports you back in time, and you exit the portal in a different location in the desert. The teleportation does not affect your thirst, only the time passed and the location. For a fixed portal, the destination and time lapse are always the same, and they are known to you. Your goal is to reach the city at a time as early as possible. You still must avoid dehydration.

Model the modified problem with a graph in such a way that an algorithm from the lecture computes a solution to the problem. Describe precisely the set of vertices and edges, and describe what the vertices and edges represent. Name an, as efficient as possible, algorithm from the lecture to solve this problem, and state the running time of the algorithm in terms of the number of hexagons, N, and the size of your water bottle, D. It is part of your solution to decide whether you can go back arbitrarily far in time by repeatedly using the same portals.

Definition of the graph (if possible, in words and not formal):
We modify the graph given as our solution for problem (b) by adding additional edges representing portals: For each portal and bottle state, we add an edge from the portal's entrance with the given bottle state to the portal's destination with that same bottle state. The weight of the edge is a negative number corresponding to the time lapse of the portal.

Number of vertices and edges (in Θ-Notation in N and D, as concisely as possible):
As there is at most one portal per hexagon, we still have $\Theta(N \cdot D)$ vertices and $\Theta(N \cdot D)$ edges.

Algorithm, as efficient as possible:
We use the Bellman–Ford algorithm, starting from the starting location with bottle state D, in order to either find a shortest path to the city with arbitrary bottle state or to determine that we can travel arbitrarily far back in time.

Running time (as concisely as possible in Θ notation in terms of N and D). Justify your answer:
The Bellman–Ford algorithm has running time $\Theta(|V| \cdot |E|)$. We have $|V| = \Theta(N \cdot D)$ and $|E| = \Theta(N \cdot D)$. Therefore, in terms of N and D, the running time is $\Theta((N \cdot D)^2)$.

Can you find out whether there is a limit on how far you can travel back in time? How? Relate your answer to the algorithm you chose to solve the above problem.
Yes. You can travel arbitrarily far back in time exactly if there is a negative cycle in our graph which is reachable from the starting location. The Bellman–Ford algorithm will converge after iterating through the edges at most $|V| - 1$ times (as there cannot be more edges in a shortest path) if and only if there is no such negative cycle. Therefore, we can run one additional iteration of Bellman–Ford's inner loop and check whether any of the distances decrease. If so, you can travel arbitrarily far back in time, otherwise you cannot.

Extra space. Please indicate clearly to which task your notes belong. Please cross out all notes that you don't want to be graded.

Theory Task T3. / 13 P

Assume that there are n towns $T_1, \dots, T_n$ in a country Examistan. Some of the towns are connected by high-speed roads. All of the high-speed roads in Examistan are one-way. Moreover, there is a unique way to get from the town T_1 to any other town that uses only high-speed roads.

For simplicity you can assume that each town T_i is represented by its index i.

/ 2 P

a) Model the n towns and high-speed roads as a directed graph: give a precise description of the vertices and edges of this graph $G = (V, E)$ involved (if possible, in words and not formal). Which graph property is implied by the sentence "There is a unique way to get from the town T_1 to any other town that uses only high-speed roads"?

Solution. $G = (V, E)$ is a tree with vertices $V = \{T_1, \dots, T_n\}$, E is a set of high-speed roads. The sentence "There is a unique way to get from the town T_1 to any other town that uses only high-speed roads" means that G is a directed tree with root T_1.

In the following subtasks b) and c) you can assume that the directed graph in a) is represented by a data structure that allows you to traverse the direct successors and direct predecessors of a vertex u in time $\mathcal{O}(\deg_+(u))$ and $\mathcal{O}(\deg_-(u))$ respectively, where $\deg_-(u)$ is the in-degree of vertex u and $\deg_+(u)$ is the out-degree of vertex u.

/ 8 P

b) Assume that you are going to build a chain of hotels in Examistan. The competition laws in Examistan require that in any town there should be at most one hotel from your chain, and that you cannot build hotels in both of the towns that are connected by a high-speed road (that is, if T_i and T_j are connected by a high-speed road, you are not allowed to build hotels in both T_i and T_j).

For each town T_i you have a profit P_i you can get from building a hotel in T_i (P_i is a nonnegative integer).

Provide an, as efficient as possible, *dynamic programming* algorithm that takes as input a graph G from task a) and a sequence $P_1, \dots, P_n$ of profits and outputs the maximal total profit. More precisely, the algorithm should output

$$V = \max_{S \in \mathcal{F}} \sum_{i \in S} P_i \,,$$

where $\mathcal{F}$ is the family of all subsets $S \subseteq \{1, \dots n\}$ that satisfy the following property: building hotels in all T_i such that $i \in S$ is consistent with the competition laws of Examistan described above.

Address the following aspects in your solution and state the running time of your algorithm:

1) *Definition of the DP table: What are the dimensions of the table $DP[\dots]$?* What is the meaning of each entry?
2) *Computation of an entry*: How can an entry be computed from the values of other entries? Which entries do not depend on others?
3) *Calculation order*: In which order can entries be computed so that values needed for each entry have been determined in previous steps?
4) *Extracting the solution*: How can the final solution be extracted once the table has been filled?

Size of the DP table / Number of entries: The DP-table is one dimensional, its size is n.

Meaning of a table entry (in words):

$DP[i]$ is the maximal total profit in the subtree with root T_i.

Computation of an entry (initialization and recursion):

$$DP[i] = \max\left\{P_i + \sum_{(i,j)\in E}\ \sum_{(j,k)\in E} DP[k]\,,\ \sum_{(i,j)\in E} DP[j]\right\}.$$

Here we assume that empty sum is 0. In particular, if T_i is a leaf, $DP[i] = P_i$.

Order of computation: Reverse BFS order (BFS starting from T_1, so in order of computation T_1 is the last entry).

Computing the output: $DP[T_1]$ contains the result.

Running time in concise Θ-notation in terms of n and m, where n is the number of vertices and m is the number of edges in G. Justify your answer: We compute $\Theta(n)$ entries of DP table and each edge of G may appear at most twice when we fill the DP table. BFS requires $\Theta(n+m)$ time. Hence the running time is $\Theta(n+m) = \Theta(n)$ (since $\Theta(m) = \Theta(n)$ for trees).

c) Use the DP table from point b) to efficiently find the optimal set of towns. More precisely, provide an algorithm that takes as input the DP table from point b) and outputs some set S^* that maximizes $\sum_{i \in S} P_i$ over $\mathcal{F}$, where $\mathcal{F}$ is a family of subsets of $\{1, \dots n\}$ described in point b).

State the running time of your algorithm in Θ-notation in terms of n and m, where m is the number of edges in G.

Solution. We start with empty S^*. For i in BFS order (starting from T_1):

If

$$DP[i] = P_i + \sum_{(i,j) \in E} \sum_{(j,k) \in E} DP[k]\,,$$

and the predecessor of i is not in S^*, we add i to S^*.

The running time of the algorithm is $\Theta(n)$.

Extra space. Please indicate clearly to which task your notes belong. Please cross out all notes that you do not want to be graded.

Extra space. Please indicate clearly to which task your notes belong. Please cross out all notes that you do not want to be graded.

Extra space. Please indicate clearly to which task your notes belong. Please cross out all notes that you do not want to be graded.

Extra space. Please indicate clearly to which task your notes belong. Please cross out all notes that you do not want to be graded.

Extra space. Please indicate clearly to which task your notes belong. Please cross out all notes that you do not want to be graded.