Teacher : Ola Svensson
Algorithms CS-250 - (n/a)
21 Jan 2022
Duration : 180 minutes

n/a

n/a

SCIPER : **999999**

Do not turn the page before the start of the exam. This document is double-sided, has 24 pages, the last ones possibly blank. Do not unstaple.

- Place your student card on your table.
- You are only allowed to have a handwritten A4 page written on both sides.
- Communication, calculators, cell phones, computers, etc... are not allowed.
- The exam consists of two parts. The first part consists of four multiple choice questions and the second part consists of four open-ended questions.
- Use a **black or dark blue ballpen** and clearly erase with **correction fluid** if necessary.
- If a question is wrong, the teacher may decide to nullify it.

Good luck!

Respectez les consignes suivantes | Read these guidelines | Beachten Sie bitte die unten stehenden Richtlinien

choisir une réponse | select an answer Antwort auswählen	ne PAS choisir une réponse | NOT select an answer NICHT Antwort auswählen	Corriger une réponse | Correct an answer Antwort korrigieren
☒ ☑ ■	☐	☐ ☐

ce qu'il ne faut **PAS** faire | what should **NOT** be done | was man **NICHT** tun sollte

▨ ☒ ◯ ⊡ ▨ ☐

First part: multiple choice questions

For each question, mark the box corresponding to the correct answer. Each question has **exactly one** correct answer. You do not need to justify your answers in this part.

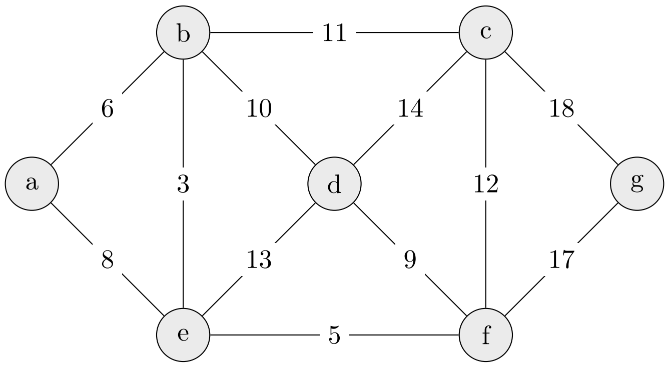
Question [SCQ-01] : *(8 pts)* Arrange the following functions in increasing order according to asymptotic growth.

$$50\log n, \quad 10^{\sqrt{n}}, \quad 100\sqrt{n}\cdot\log\log n, \quad n/2, \quad n^3\cdot n^{\log n}, \quad (\log\log n)^{10}$$

The correct increasing order is

- [✓] $(\log\log n)^{10}$, $50\log n$, $100\sqrt{n}\cdot\log\log n$, $n/2$, $n^3\cdot n^{\log n}$, $10^{\sqrt{n}}$
- ☐ $(\log\log n)^{10}$, $50\log n$, $100\sqrt{n}\cdot\log\log n$, $n/2$, $10^{\sqrt{n}}$, $n^3\cdot n^{\log n}$
- ☐ $(\log\log n)^{10}$, $50\log n$, $n/2$, $100\sqrt{n}\cdot\log\log n$, $n^3\cdot n^{\log n}$, $10^{\sqrt{n}}$
- ☐ $(\log\log n)^{10}$, $50\log n$, $n/2$, $100\sqrt{n}\cdot\log\log n$, $10^{\sqrt{n}}$, $n^3\cdot n^{\log n}$
- ☐ $50\log n$, $(\log\log n)^{10}$, $100\sqrt{n}\cdot\log\log n$, $n/2$, $n^3\cdot n^{\log n}$, $10^{\sqrt{n}}$
- ☐ $50\log n$, $(\log\log n)^{10}$, $100\sqrt{n}\cdot\log\log n$, $n/2$, $10^{\sqrt{n}}$, $n^3\cdot n^{\log n}$
- ☐ $50\log n$, $(\log\log n)^{10}$, $n/2$, $100\sqrt{n}\cdot\log\log n$, $n^3\cdot n^{\log n}$, $10^{\sqrt{n}}$
- ☐ $50\log n$, $(\log\log n)^{10}$, $n/2$, $100\sqrt{n}\cdot\log\log n$, $10^{\sqrt{n}}$, $n^3\cdot n^{\log n}$

Question [SCQ-02] : *(8 pts)*
Consider the following undirected graph with edge weights:



What are the weights of the edges (in the correct order) added to the solution by Dijkstra's and Prim's algorithms starting from a, i.e., with source a? Recall that Dijkstra's algorithm calculates a shortest path tree with source a and Prim's algorithm calculates a minimum spanning tree.

■ The weights of the edges added by Dijkstra's algorithm (in the correct order) is $6, 8, 5, 10, 11, 17$.
The weights of the edges added by Prim's algorithm (in the correct order) is $6, 3, 5, 9, 11, 17$.

☐ The weights of the edges added by Dijkstra's algorithm (in the correct order) is $6, 3, 5, 9, 11, 17$.
The weights of the edges added by Prim's algorithm (in the correct order) is $6, 8, 5, 10, 11, 17$.

☐ The weights of the edges added by Dijkstra's algorithm (in the correct order) is $6, 8, 5, 10, 11, 17$.
The weights of the edges added by Prim's algorithm (in the correct order) is $3, 5, 6, 9, 11, 17$.

☐ The weights of the edges added by Dijkstra's algorithm (in the correct order) is $3, 5, 6, 9, 11, 17$.
The weights of the edges added by Prim's algorithm (in the correct order) is $6, 8, 5, 10, 11, 17$.

☐ The weights of the edges added by Dijkstra's algorithm (in the correct order) is $5, 6, 8, 10, 11, 17$.
The weights of the edges added by Prim's algorithm (in the correct order) is $6, 3, 5, 9, 11, 17$.

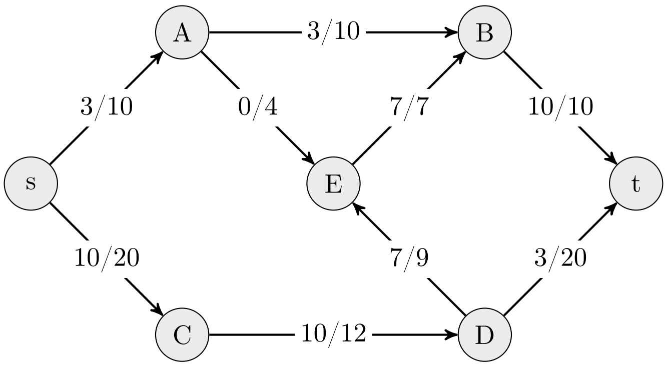
☐ The weights of the edges added by Dijkstra's algorithm (in the correct order) is $6, 3, 5, 9, 11, 17$.
The weights of the edges added by Prim's algorithm (in the correct order) is $5, 6, 8, 10, 11, 17$.

☐ The weights of the edges added by Dijkstra's algorithm (in the correct order) is $5, 6, 8, 10, 11, 17$.
The weights of the edges added by Prim's algorithm (in the correct order) is $3, 5, 6, 9, 11, 17$.

☐ The weights of the edges added by Dijkstra's algorithm (in the correct order) is $3, 5, 6, 9, 11, 17$.
The weights of the edges added by Prim's algorithm (in the correct order) is $5, 6, 8, 10, 11, 17$.

☐ The weights of the edges added by Dijkstra's algorithm (in the correct order) is $6, 3, 5, 9, 11, 17$.
The weights of the edges added by Prim's algorithm (in the correct order) is $6, 3, 5, 9, 11, 17$.

Question [SCQ-03] : *(8 pts)* Assume the following flow network and corresponding flow of value 13 (the numbers on an edge determine its current flow and its capacity).



Starting with the depicted flow, what is the value of the flow obtained by running **one iteration** of the Ford-Fulkerson algorithm that chooses the *fattest augmenting path* (the one with the largest bottleneck).

- ■ The value of the obtained flow is 20.
- ☐ The value of the obtained flow is 15.
- ☐ The value of the obtained flow is 13.
- ☐ The value of the obtained flow is 17.
- ☐ The value of the obtained flow is 21.
- ☐ The value of the obtained flow is 14.
- ☐ The value of the obtained flow is 16.
- ☐ The value of the obtained flow is 18.
- ☐ The value of the obtained flow is 19.

Question [`SCQ-04`] : *(8 pts)* Consider an undirected graph $G = (V, E)$ where each vertex has degree three, i.e., each vertex is incident to exactly three edges. Now suppose we color the edges of G as follows:

> Each edge is independently colored red, green, or blue uniformly at random (i.e., each of the three colors is equally likely).

We say that a vertex is *monochromatic* if all its three incident edges are colored with the same color. What is the expected number of monochromatic vertices?

- [✓ The expected number of monochromatic vertices is $|V|/9$.]
- ☐ The expected number of monochromatic vertices is $|V|/3$.
- ☐ The expected number of monochromatic vertices is $|V|/27$.
- ☐ The expected number of monochromatic vertices is $|E|/9$.
- ☐ The expected number of monochromatic vertices is $|E|/3$.
- ☐ The expected number of monochromatic vertices is $|E|/27$.

Second part, open questions

This part consists of four questions, each worth 17 points. Please follow the following instructions:

- Your explanations should be clear enough and in sufficient detail that a fellow student can understand them. In particular, do not only give pseudocode without explanations. A good guideline is that a description of an algorithm should be such that a fellow student can easily implement the algorithm following the description.
- You are allowed to refer to material covered in the lectures including algorithms and theorems (without reproving them). You are however *not* allowed to simply refer to material covered in exercises.
- Please answer all questions within the designated boxes (otherwise your answer may not be accurately scanned). At the end of the exam there are five extra pages if you need additional space for your answers.
- Leave the check-boxes empty, they are used for the grading.

Question 5: Recurrences. *(17 pts)*

☐ 0 ☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐ 6 ☐ 7 ☐ 8 ☐ 9
☐ 10 ☐ 11 ☐ 12 ☐ 13 ☐ 14 ☐ 15 ☐ 16 ■ 17

Do not write here.

Consider the following procedure Unknown that takes as input an integer n:

```
Unknown($n$):
1. if $n < 100$
2.       return
3. $q = \lfloor n/5 \rfloor$
4. Unknown($3 \cdot q$)
5. Unknown($4 \cdot q$)
6. for $i = 1, 2, \ldots, n$
7.       PRINT “Almost done!”
```

Let $T(n)$ be the time it takes to execute Unknown(n). Give the recurrence relation of $T(n)$ and prove that $T(n) = O(n^2)$. To simplify notation and calculations, you may assume that $n/5$ always evaluates to an integer.

(In this question you are asked to give the recurrence relation of $T(n)$ and to give a mathematically rigorous proof that $T(n) = O(n^2)$. Recall that you are allowed to refer to material covered in the lectures.)

Begin writing your answer on the next page.

The recurrence relation is given by $T(n) = T(3n/5) + T(4n/5) + cn$ (where c is a constant) and $T(n) = 1$ if $n < 100$.

Let us prove by induction that $T(n) \leq an^2 + bn$ for some constants a, b that we will fix.

Assume this is true for all inputs up to $n-1$. Then we can write, by the recurrence relation,

$$\begin{aligned} T(n) &= T(3n/5) + T(4n/5) + cn \\ &\leq a\left(\frac{3n}{5}\right)^2 + b\frac{3n}{5} + a\left(\frac{4n}{5}\right)^2 + b\frac{4n}{5} + cn \\ &= an^2 \cdot \left(\left(\frac{3}{5}\right)^2 + \left(\frac{4}{5}\right)^2\right) + n \cdot \left(\frac{7b}{5} + c\right) \\ &= an^2 + n \cdot \left(\frac{7b}{5} + c\right). \end{aligned}$$

By choosing b such that $\frac{7b}{5} + c \leq b \iff b \leq -\frac{5c}{2}$, we obtain that the induction hypothesis works. Hence we fix $b = -\frac{5c}{2}$.

For the base case to be correct, we need that $an^2 - \frac{5cn}{2} \geq 1$ for all $1 \leq n < 100$. Choosing $a = 3c$ is enough.

Therefore, we have proven that $T(n) \leq 3cn^2 - \frac{5cn}{2}$ for all $n \geq 1$. This shows that $T(n) = O(n^2)$.

Catalog

Question 6: Check shortest path distances. *(17 pts)*

☐0 ☐1 ☐2 ☐3 ☐4 ☐5 ☐6 ☐7 ☐8 ☐9
☐10 ☐11 ☐12 ☐13 ☐14 ☐15 ☐16 ■17

Do not write here.

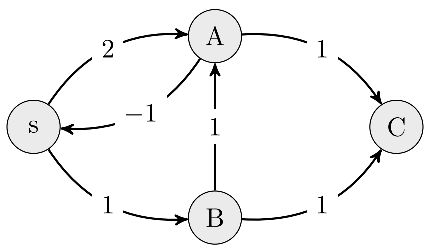
Design and analyze an algorithm that **runs in linear time**, i.e., in time $O(|V| + |E|)$, for the following problem:

INPUT: A directed graph $G = (V, E)$, a source $s \in V$, edge-weights $w : E \to \mathbb{Z}$, and a value $v.d \in \mathbb{Z}$ for each vertex $v \in V$.

OUTPUT: Print "CORRECT" if $v.d$ equals the shortest path distance from s to v for every vertex $v \in V$; otherwise print "INCORRECT."

You may assume that every vertex is reachable from s and that the graph only contains non-negative cycles but it may contain negative edge weights.
We remark that there is no assumption on the values $v.d$. A solution that is fully correct under the additional assumption that, for every vertex $v \in V$, $v.d$ is at least the shortest path distance from s to v is rewarded 10 points.

Example: Consider the following input (the values $v.d$ are not depicted):



The output of your algorithm should be "CORRECT" if and only if $s.d = 0, A.d = 2, B.d = 1$, and $C.d = 2$.

(In this question you are asked to design and analyze an algorithm that runs in time $O(|V| + |E|)$ for the problem of verifying that the values $v.d$ equals the shortest path distance from s to v for every vertex $v \in V$. Recall that you are allowed to refer to material covered in the lectures.)

We call edge $e = (u, v)$ tight if $u.d + w(u, v) = v.d$. We call a path between two vertices tight if it consists entirely of tight edges.

The solution is as follows:

1. Check $s.d = 0$.
2. For each edge $e = (u, v)$:
   (a) Check that $v.d \geq u.d + w(u, v)$.
   (b) If e is tight mark it as tight.
3. Check that each vertex is reachable from root through tight paths, using DFS.

If any check fails, output "Incorrect". Otherwise, output "Correct".

Time analysis. The first two steps take time $\Theta(1)$. The loop runs for $|E|$ iterations and each iteration takes time $\Theta(1)$. Thus the total running time of the loop is $\Theta(|E|)$. Finally, the running time of DFS is $\Theta(|V| + |E|)$ and so the total running time is $\Theta(|V| + |E|)$ as required.

Correctness. It is pretty obvious that if the distance assignment is correct, the algorithms outputs "Correct".

If the assignment is not correct, let S be the set of all vertices that have the correct assignment, and let $dist(v)$ denote the true distances to each of the vertices. Notice that if $s \notin S$, we will detect it when we check $s.d = 0$. Therefore, we can assume $s \in S$. Then, there exists $v \in V \setminus S, v \neq s$. Let C be the set of all vertices in $V \setminus S$, such that $\forall v \in C$ there is some shortest path from s to v where v is the only vertex in that path not in S. Suppose that there is a $v \in C$ such that $dist(v) < v.d$. Let u be the second-to-last vertex in the shortest path to v. Then $u \in S$ and $dist(u) = u.d$. Therefore, we will output "Incorrect" when we check edge (u, v) in step 2, since $dist(v) = dist(u) + w(u, v)$ since it's an edge on the shortest path to v.

Now, suppose that for all of the vertices v in C $dist(v) > v.d$. Then there is no tight path from s to any vertex v in C, because otherwise $v.d$ would be equal to the length of some path from s, which is at least $dist(v) > v.d$. Therefore, if C is not empty, the algorithm will not reach vertices in C in step 3 and output "Incorrect".

Finally, notice that if $S \neq V$, C is not empty, because we can always consider a shortest path from s to some vertex in $V \setminus S$, and the first vertex not in S on that path would be in C. Therefore, if the assignment $v.d$ is incorrect, the algorithm always outputs "Incorrect".

Catalog

Question 7: Short walk of maximum value. *(17 pts)*

☐0 ☐1 ☐2 ☐3 ☐4 ☐5 ☐6 ☐7 ☐8 ☐9
☐10 ☐11 ☐12 ☐13 ☐14 ☐15 ☐16 ■17 *Do not write here.*

Consider a $(n+1) \times (n+1)$ grid of cells with coordinates (i, j) for $0 \le i \le n$ and $0 \le j \le n$. We are going to analyze walks from the bottom-left cell $(0,0)$ to the top-right cell (n,n) where the available moves are:

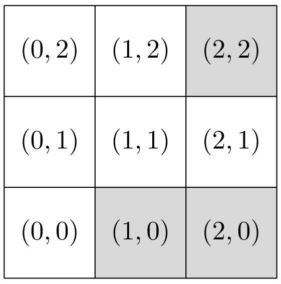
UP: Standing in a cell (i, j) with $j < n$, we can go to $(i, j+1)$.

RIGHT: Standing in a cell (i, j) with $i < n$, we can go to $(i+1, j)$.

UP-RIGHT: Standing in a cell (i, j) with $i < n$ and $j < n$, we can go to $(i+1, j+1)$.

Notice that any such walk makes at least n moves and at most $2n$ moves. We say that the walk is **short** if it makes at most $3n/2$ moves. In addition, each cell (i, j) has a value $v(i, j)$, and the value of a walk is the total value of the cells it visited. Your task is to design and analyze an efficient algorithm that calculates the maximum value any short walk from $(0,0)$ to (n,n) can achieve.
The running time of your algorithm should be polynomial in n. While your algorithm should run in polynomial time, it is not important that you achieve an optimal running time in this question. We also remark that partial credits will be given to solutions that output the maximum value that any (not necessarily short) walk can achieve.

Example: Consider the following $(n+1) \times (n+1)$ grid with $n = 2$:



The value of the gray cells is 1 and the value of the white cells is 0 (in this example cells take values 0 and 1 but in general cells can take any value). Hence, the maximum value of any walk from $(0,0)$ to $(2,2)$ is 3. This value is obtained by the walk that visits the cells $(0,0), (1,0), (2,0), (2,1), (2,2)$. However, this walk makes $2n = 4$ moves and is thus not short. The maximum value of a short walk (that makes at most $3n/2 = 3$ moves) is 2. This is achieved by the short walk that makes 3 moves and visits the cells $(0,0), (1,0), (2,1), (2,2)$. On this input, the output of your algorithm should thus be 2.

(In this question you are asked to design and analyze an algorithm that runs in time polynomial in n for the problem of calculating the maximum value of any short walk from $(0,0)$ to (n,n). Recall that you are allowed to refer to material covered in the lectures.)

Begin writing your answer on the next page.

We will solve this problem using the dynamical programming approach. Let $dp[i][j][k]$ denote the maximal value of a walk that starts in $(0,0)$, ends in (i,j) and makes maximally k moves. The main observation is that the maximum value for a walk ending in (i,j) making at most k moves is equal to $V[i][j]$ plus the maximum of short walks ending in $(i,j-1),(i-1,j),(i-1,j-1)$ making at most $k-1$ moves. This gives us a recurrence relation

$$dp[i][j][k] = V[i][j] + \max(d[i-1][j][k-1], dp[i][j-1][k-1], dp[i-1][j-1][k-1]). \tag{1}$$

One needs to be careful about corner cases, for instance $k=0$ or $i=0, j=0$. One needs to take them into account when writing down the full recurrence relation. One way to take care of these corner cases is given in the pseudocode below.

Algorithm 1 FindMaximumShortWalk(V, n)

```
1: $dp[0][0][0] = V[0][0]$
2: for $i$ from 0 to $n$ do
3:     for $j$ from 0 to $n$ do
4:         for $k$ from 1 to $\frac{3}{2}n$ do
5:             if $i = 0$ and $j = 0$ then
6:                 $dp[i][j][k] = dp[i][j][k-1]$
7:             else if $i = 0$ then
8:                 $dp[i][j][k] = V[i][j] + dp[i][j-1][k-1]$
9:             else if $j = 0$ then
10:                $dp[i][j][k] = V[i][j] + dp[i-1][j][k-1]$
11:            else
12:                $dp[i][j][k] = V[i][j] + \max(dp[i-1][j][k-1], dp[i][j-1][k-1], dp[i-1][j-1][k-1])$
13: return $dp[n][n][\frac{3}{2}n]$
```

The first condition says that the maximum value when we end in cell $(0,0)$ in 0 moves is $V[0][0]$. In line 6 we say that to end up in cell $(0,0)$ with the number of moves at most k is the same as with the number of moves at most $k-1$. Lines 8,10 express the fact that if we are at the edges of the grid then we can only do either up or right moves. Line 12 represents (1).

The algorithm runs in time $\Theta(n^3)$ as we have three for loops inside each other, where each iterates over $\Theta(n)$ values, and we perform $\Theta(1)$ operations inside of them.

Catalog

Catalog

Question 8: Many spanning trees. *(17 pts)*

☐0 ☐1 ☐2 ☐3 ☐4 ☐5 ☐6 ☐7 ☐8 ☐9
☐10 ☐11 ☐12 ☐13 ☐14 ☐15 ☐16 ■17

Do not write here.

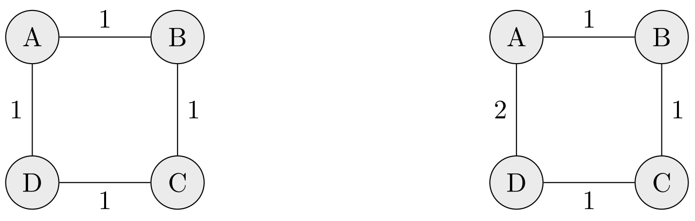
Design and analyze an efficient algorithm for the following problem:

INPUT: A connected undirected graph $G = (V, E)$ with integer edge-weights $w : E \to \mathbb{Z}$.

OUTPUT: Print "ONE" if G with edge-weights w has a unique minimum spanning tree; otherwise print "MANY."

The running time of your algorithm should be polynomial in $|V|$. While your algorithm should run in polynomial time, it is not important that you achieve an optimal running time in this question. For full credit, you should also explain in words why your algorithm is correct.

Example: Consider the following two inputs:



The output of your algorithm should be "MANY" on the left input since that graph contains four minimum spanning trees; on the right instance, the output should be "ONE" since that graph contains a unique minimum spanning tree.

(In this question you are asked to design and analyze an algorithm that runs in time polynomial in $|V|$ for the problem of verifying whether the graph $G = (V, E)$ with integer edge-weights w has a unique minimum spanning tree. While you do not need to give a rigorous proof of correctness, you should argue why your algorithm is correct in addition to analyzing its running time. Recall that you are allowed to refer to material covered in the lectures.)

In the following, for a graph $G = (V, E)$ with edge weights $w : E \to \mathbb{Z}$ we will denote by $w(G)$ the sum of the weights of edges in G, that is $w(G) = \sum_{e \in E} w(e)$. The following algorithm solves the problem.

```
**Input:** a graph $G = (V, E)$
Run Kruskal algorithm on graph $G$ to get an MST $T$.
**for all** $e \in T$ **do**
  **if** graph $G' = (V, E \setminus \{e\})$ is connected **then**
    Run Kruskal on $G'$ to get the corresponding MST solution $T'$.
    **if** $w(T) = w(T')$ **then**
      Return "MANY".
    **end if**
  **end if**
**end for**
Return "ONE".
```

Proof:

- Assume there exist two different MST solutions T, T' of graph G. Then, we know that $w(T) = w(T')$. In addition, let $e \in T \setminus T'$ then T' is an MST of graph $G' = (V, E \setminus \{e\})$. Thus, our algorithm will compute an MST of value $w(T)$ after running Kruskal on graph $G' = (V, E \setminus \{e\})$ and it will output "MANY".
- Assume that there exists a unique MST T of graph G. Then for all edges $e \in G$ the graph $G' = (V, E \setminus \{e\})$ will either be disconnected or every MST T' of that graph will have a larger total weight, i.e., $w(T') > w(T)$. Consequently our algorithm will correctly output "ONE".

Time complexity: The time complexity of Kruskal is $|E| \log |E| \leq |V|^2 \log |V|^2 = O(|V|^3)$. In addition finding if a graph is connected requires time $O(|E| + |V|)$. Consequently, our algorithm has time complexity $O(|V|^4)$ since the number of edges in an MST solution T is $O(|V|)$.

Catalog

Catalog