ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Final Exam, Algorithms 2015-2016

- You are only allowed to have a handwritten A4 page written on both sides.
- Communication, calculators, cell phones, computers, etc... are not allowed.
- Your explanations should be clear enough and in sufficient detail so that a fellow student can understand it. In particular, do not only give pseudocode without explanations. A good guideline is that a description of an algorithm should be so that a fellow student can easily implement the algorithm following the description.
- **Do not touch until the start of the exam.**

Good luck!

Name: _____ **N° Sciper:** _____

Problem 1 / 23 points	Problem 2 / 10 points	Problem 3 / 15 points	Problem 4 / 16 points	Problem 5 / 17 points	Problem 6 / 19 points

Total / 100

1 *(23 pts)* **Basic questions.** This problem consists of four subproblems (1a-1d) for which you do *not* need to motivate your answers.

1a *(6 pts)* Let $A[1\ldots9] =$ | 3 | 9 | 5 | 4 | 8 | 6 | 11 | 7 | 2 | be an array consisting of 9 numbers. Illustrate how A looks like after executing the code

MAX-HEAPIFY$(A, 4, 9)$, MAX-HEAPIFY$(A, 3, 9)$, MAX-HEAPIFY$(A, 2, 9)$, MAX-HEAPIFY$(A, 1, 9)$.

Solution:

Resulting $A =$ | 11 | 9 | 6 | 7 | 8 | 3 | 5 | 4 | 2 |

1b (6 pts) A brilliant and highly motivated student at EPFL has decided to improve upon Strassen's algorithm for matrix multiplication. He strongly believes that after partitioning the matrices into 16 submatrices each of dimension $n/4 \times n/4$ — similarly to Strassen's algorithm (recall that Strassen's algorithm partitions the matrices into 4 submatrices each of dimension $n/2 \times n/2$) — one can obtain the final answer by using 32 matrix multiplications (each multiplying two matrices of dimension $n/4 \times n/4$). If the student modifies Strassen's algorithm using this idea, what is the running time obtained for multiplying two matrices of dimension $n \times n$?

Solution:

The answer is $\Theta(n^{2.5})$.

1c *(5 pts)* Suppose you have n distinct keys that you wish to store in a hash table. We have six different alternatives for the size of the hash table:

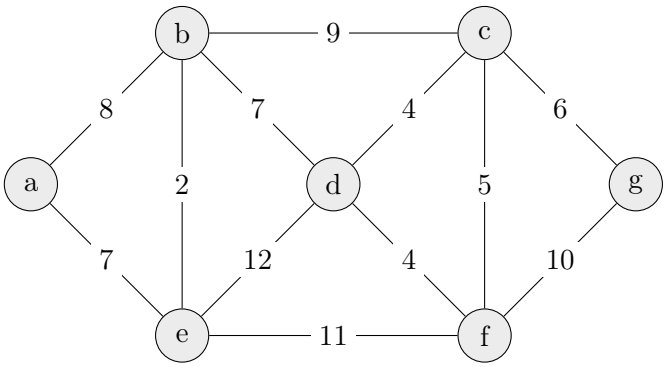
A: n **B:** $n \log n$ **C:** n^2 **D:** $n^2 \log n$ **E:** n^3 **F:** $n^3 \log n$

Assuming simple uniform hashing, what is the smallest size of the hash table (among the above alternatives) so that the expected number of collisions is less than 1 ?

Solution:

The choice is C.

1d *(6 pts)* Consider the following undirected graph with edge weights:



Write the weights of the edges (in the correct order) added to the shortest path tree by Dijkstra's algorithm starting from a, i.e., with source a.

Solution:

$7, 8, 7, 9, 11, 6$

2 *(10 pts)* **Probabilistic analysis.** Consider the following algorithm RANDOM-CUT that takes as input an undirected unweighted graph $G = (V, E)$.

```
RANDOM-CUT(G)
1. Let s and t be two vertices in G.
2. Let S = {s} and T = {t}.
3. for each vertex v in G different from s and t
4.       Add v to S or to T with equal probability 1/2.
5. return the cut (S, T)
```

2a *(6 pts)* Recall that an edge crosses a cut (S, T) if one of its end points is in S and the other one is in T. **Prove** that in expectation the number of edges crossing the cut returned by RANDOM-CUT is at least $|E|/2$, i.e., at least half of the edges crosses the cut in expectation.

Solution:

For each edge $e \in E$, let X_e be the indicator random variable for the event that "e crosses the cut (S, T) returned by RANDOM-CUT". Then the expected number of edges that crosses the cut is

$$\mathbb{E}\left[\sum_{e \in E} X_e\right] = \sum_{e \in E} \mathbb{E}[X_e]$$
$$= \sum_{e \in E} \Pr_{(S,T)}[e \text{ crosses the cut } (S, T)].$$

We complete the proof by showing that $\Pr_{(S,T)}[e \text{ crosses the cut } (S, T)]$ is at least $1/2$ for any edge e. If $e = \{s, t\}$ then the probability is one. In the other cases, when $e = \{s, u\}$ or $e = \{u, t\}$ or $e = \{u, v\}$, the probability equals $1/2$ (because after fixing the side of one of the vertices, the probability that the other vertex goes to the opposite side of the cut is exactly $1/2$). This completes the proof since this implies that $\sum_{e \in E} \Pr_{(S,T)}[e \text{ crosses the cut } (S, T)] \geq |E|/2$.

2b *(4 pts)* **Give a small modification** to the algorithm RANDOM-CUT so that *strictly more* than half of the edges crosses the returned cut in expectation. Here, we assume that the input graph contains at least one edge.

Solution:

The small modification is that instead of chosing s and t to be any two vertices in G, we select an edge $e = \{u, v\}$ in G and let $s = u$ and $t = v$. Then that edge will be cut with probability 1 and all other edges in the graph will be cut with probability $1/2$ by the same analysis as in the previous subproblem. Thus the expected number of edges crossing the edges with this modification is

$$1 + \frac{|E| - 1}{2} = \frac{|E| + 1}{2},$$

which is strictly greater than $|E|/2$.

3 *(15 pts)* **Job assignments.** Consider the following job assignment problem:

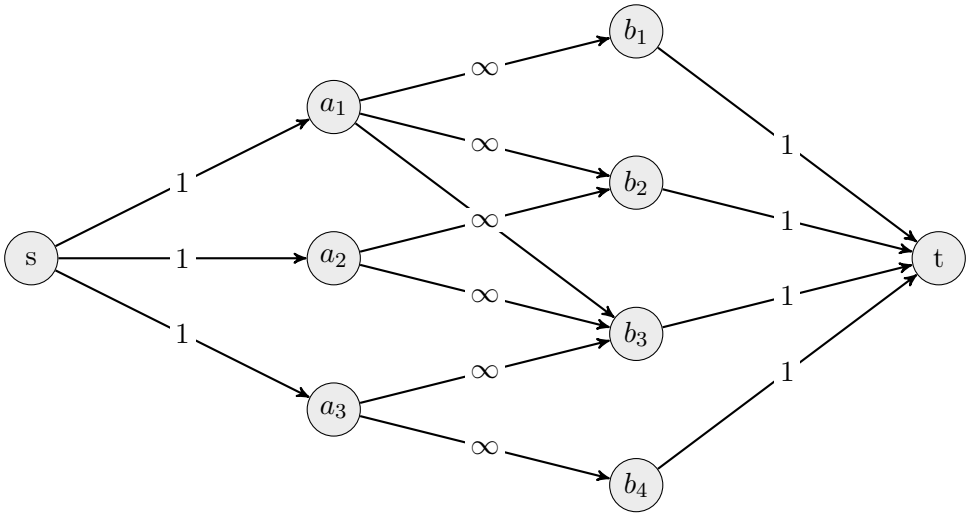
> There is a set $A = \{a_1, a_2, \ldots, a_n\}$ of n students and a set $B = \{b_1, b_2, \ldots, b_m\}$ of m jobs. Each student $a_i \in A$ is interested in a subset $N(a_i) \subseteq B$ of the jobs. The goal is to find (if possible) an assignment of jobs to students so that each student $a_i \in A$ gets one job b_j in $N(a_i)$ and each job is assigned to at most one student.

In class, we saw how to formulate this problem as a max flow problem of a flow network G so that the maximum flow value is equal to n if and only if there exists a job assignment to the students. The flow network G is defined as follows (see also example below):

- The vertices consist of a source s, a sink t, a vertex $a_i \in A$ for each student, and a vertex $b_j \in B$ for each job.
- There is an arc of capacity 1 from the source s to a_i for each student $a_i \in A$. Similarly, there is an arc of capacity 1 from b_j to the sink t for each job $b_j \in B$.
- Finally, there is an arc of capacity ∞ from a_i to b_j if and only if $b_j \in N(a_i)$, i.e., if student a_i is interested in the job b_j.

The task, in the following two subproblems, is to analyze for which instances there is a job assignment (or equivalently, for which instances, the flow network has a max flow value of n).

Example of flow network: Consider the instance when $S = \{a_1, a_2, a_3\}$, $J = \{b_1, b_2, b_3, b_4\}$, and $N(a_1) = \{b_1, b_2, b_3\}$, $N(a_2) = \{b_2, b_3\}$ and $N(a_3) = \{b_3, b_4\}$. The corresponding flow network can then be depicted as follows:



3a *(5 pts)* Suppose there is a subset $A' \subseteq A$ of students so that $|A'| > |N(A')|$, where $N(A') = \bigcup_{a_i \in A'} N(a_i)$. Then there cannot be an assignment of jobs to the students because the total number of jobs that interest the students in A' is less than the number of students in A'.

In this case, which of the following cuts is guaranteed to have capacity less than n?

A: The cut defined by $S = \{s\}$ and $T = \{t\} \cup A \cup B$.

B: The cut defined by $S = \{s\} \cup A \cup B$ and $T = \{t\}$.

C: The cut defined by $S = \{s\} \cup A'$ and $T = \{t\} \cup (A \setminus A') \cup B$.

D: The cut defined by $S = \{s\} \cup N(A')$ and $T = \{t\} \cup A \cup (B \setminus N(A'))$.

E: The cut defined by $S = \{s\} \cup A' \cup N(A')$ and $T = \{t\} \cup (A \setminus A') \cup (B \setminus N(A'))$.

F: The cut defined by $S = \{s\} \cup A' \cup (B \setminus N(A'))$ and $T = \{t\} \cup (A \setminus A') \cup N(A')$.

Solution:

The cut which is guaranteed to have capacity less than n is E

(You are *not* required to motivate your answer in this subproblem.)

3b *(10 pts)* **Prove** that if every $A' \subseteq A$ satisfies

$$|A'| \leq |N(A')| \qquad \text{(where again } N(A') = \textstyle\bigcup_{a_i \in A'} N(a_i)\text{)}$$

then there is a job assignment, i.e., the max flow has value n.

Solution:

We prove that the max flow has value $\geq n$ by showing that any s,t-cut has capacity at least n. Consider any cut (S,T) where $s \in S$ and $t \in T$. Let

$$A' = A \cap S \qquad \text{and} \qquad B' = B \cap S.$$

First observe that if $N(A') \not\subseteq B'$ then there exists an arc from a student $a_i \in A'$ to a job $b_j \in N(a_i) \setminus B'$, i.e., this arc crosses the cut. Since these arcs have capacity ∞, the cut has clearly capacity $\geq n$ in this case.

Therefore we may from now on assume that $N(A') \subseteq B'$. In that case the capacity of the cut is

$$\sum_{a_i \notin A'} 1 + \sum_{b_j \in B'} 1,$$

where the first sum comes from the arcs from the source s to the students not in A' and the second sum comes from the arcs from the jobs in B' to the sink. The capacity of the cut can thus be rewritten as

$$n - |A'| + |B'| \geq n - |A'| + |N(A')| \geq n,$$

where the first inequality follows from that we now assume $N(A') \subseteq B'$ and the second inequality follows from the assumption of the statement.

We have thus verified that every cut has capacity $\geq n$ and since the cut $S = \{s\}, T = \{A, B, t\}$ has capacity n, we can conclude from the max-flow min-cut Theorem that the max-flow has value n.

4 *(16 pts)* **Shortest paths.**

4a *(3 pts)* Consider a directed graph $G = (V, E)$ with edge-weights $w : E \to \mathbb{R}$ and a vertex/source $s \in V$. We assume that G does not contain any negative cycles. In addition, we have the following facts:

- The edges $(u_1, v), (u_2, v) \in E$ are both part of shortest paths from s to v and $w(u_1, v) = 10, w(u_2, v) = 20$.
- The length of a shortest path from s to v is 1981.

What are the lengths of shortest paths from s to u_1 and from s to u_2?

Solution:

The length of a shortest path from s to u_1 is 1971.

The length of a shortest path from s to u_2 is 1961.

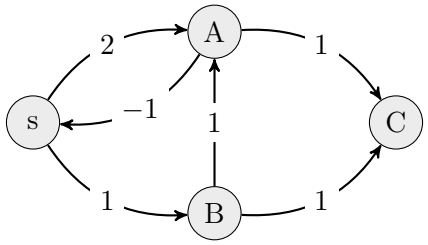
(You are *not* required to motivate your answers in this subproblem.)

4b *(13 pts)* **Design and analyze** an efficient algorithm for the following problem:

> Given a directed graph $G = (V, E)$ with edge-weights $w : E \to \mathbb{R}$ and a vertex/source $s \in V$, return all the edges that are part of a shortest path starting at s.

You may assume that the graph only contains non-negative cycles but it may contain negative edge weights.

Example: Consider the following input:



Then the shortest paths from s are $s \to A$, $s \to B$, $s \to B \to A$, $s \to B \to C$. Therefore the correct output of the algorithm would be the edges $(s, A), (s, B), (B, A), (B, C)$.

Solution:

The algorithm is as follows:

1. Run Bellman-Ford to calculate the distance $d(u)$ from s all other vertices $u \in V$. (We use Bellman-Ford as the graph may have negative edges but no negative cycles.)
2. Let SOL $= \emptyset$. For each arc $(u, v) \in E$, add (u, v) to SOL if $d(v) = d(u) + w(u, v)$.
3. Return SOL.

First we calculate all the shortest path lengths from the source. Then we add an edge (u, v) to the solution SOL if and only if it is part of a shortest path from s to v. This is the case if a shortest path from s to u of length $d(u)$ plus the length of the edge (u, v) (i.e., $w(u, v)$) equals the length of a shortest path from s to v.

The running time of our algorithm is as follows. The first step takes $O(|V||E|)$ time. The second step takes $O(|E|)$ time. Hence the total running time of the algorithm is $O(|V||E|)$ which is the asymptotic running time of the Bellman-Ford algorithm.

5 *(17 pts)* **The funny array game. Design and analyze** an efficient algorithm for the following problem of optimally playing a game on an array:

> You are given a sequence of n positive numbers $a_1, a_2, ..., a_n$. Initially, they are all colored black. In one move, you can choose a black number a_k and color it and its immediate neighbors (if any) red (the immediate neighbors are the elements a_{k-1}, a_{k+1}). You get a_k points for this move. What is the maximum total number of points that you can get during the game?

For full credit, your algorithm should run in time $O(n)$.

Example: Given a sequence `1 2 6 4 7`, you can start by choosing `6`; when you do, you get 6 points and have to color `2 6 4` red. Next, you can choose `7`, which gives you 7 points; you color `7` red (its neighbor `4` was already red). Finally, you choose `1` (the only remaining black number), get 1 point and color it red. Now all numbers are red and you cannot make any more moves. You obtain $6 + 7 + 1 = 14$ points, which is the maximum possible score.

Solution:

First we observe that it does not matter in which order we choose the numbers. Instead a feasible selection of numbers can be described as a subset of numbers so that we have not chosen two adjacent indices.

We can now write a recursive formulation of our problem. Let $p[t]$ be the maximum number of points of an optimal solution to the game on the first t numbers, i.e., on $a_1, a_2, \ldots, a_t$. Then, we can express $p[t]$ as the following recurrence

$$p[t] = \begin{cases} 0 & \text{if } t = 0 \\ a_1 & \text{if } t = 1 \\ \max\{p[t-1], p[t-2] + a_t\} & \text{if } t \geq 2. \end{cases}$$

The maximum is over two options: either we don't pick the t:th number so we get a smaller instance of the first $t-1$ numbers; or we pick the t:th number and in this case we cannot pick the $t-1$:th number so we get a smaller instance of the first $t-2$ numbers.

We now can implement this recurrence either using the bottom-up or the top-down approach. We do bottom-up:

BOTTOM-UP$(a_1, a_2, \ldots, a_n)$

1. Let $p[n]$ be an array of n numbers.
2. $p[0] = 0$ and $p[1] = a_1$
3. For $t = 2, \ldots, n$:
4. $\quad p[t] = \max\{p[t-1], p[t-2] + a_t\}$
5. **return** $p[n]$.

It is clear that the loop dominates the running time. As it is executed $\leq n$ times, the total running time is $O(n)$.

6 *(19 pts)* **Median of two sorted arrays.**

Given two *sorted* arrays A and B that both contain n integers, **design and analyze** an efficient algorithm that returns the median of the array resulting from merging A and B. For full credit, your algorithm should run in time $O(\log n)$.

(Recall that the median of k numbers $a_1 \leq a_2 \leq \cdots \leq a_k$ is $a_{\lceil k/2 \rceil}$ if k is odd and $\frac{a_{(k/2)}+a_{(k/2+1)}}{2}$ if k is even.)

Solution:

The idea of the algorithm is as follows: Let C denote the merged array of A and B. Note that if we remove the x smallest numbers and the x largest numbers from C then the median does not change (as long as we do not remove the numbers that define the median). How can we efficiently remove smaller and larger elements in the merged array without merging it? Well let $\text{Med}(A)$ denote the median of A and let $\text{Med}(B)$ denote the median of B. Suppose that $\text{Med}(A) \leq \text{Med}(B)$, then we know that the numbers $A[1 \dots \lceil n/2 \rceil - 1]$ will not be part of the median because they are too small of C and $B[\lfloor n/2 \rfloor + 2 \dots n]$ will not be part of the median because they are too big. This means that we have thrown away $x = \lceil n/2 \rceil - 1 = n - (\lfloor n/2 \rfloor + 2) + 1$ smaller and bigger elements and we can recurse on the smaller instance of roughly half the size.