ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Final Exam, Algorithms 2014-2015

- 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 / 15 points	Problem 2 / 15 points	Problem 3 / 5 points	Problem 4 / 11 points	Problem 5 / 29 points	Problem 6 / 25 points

Total / 100

1 *(15 pts)* **Asymptotics and basic runtime analysis.**

1a *(5 pts)* Arrange the following functions in increasing order according to asymptotic growth.

$$2^n, (\log\log n)^{10}, n^{300}, \sqrt{n}, n/\log n, 2^{2^n}, n^{\sqrt{n}}, \log n$$

(In this problem, you only need to give the right order, i.e., you do not need to explain your answer.)

Solution:

1b *(10 pts)* Consider the following four recursive functions. We assume $f_i(0) = 0$ and they are defined as follows for $n \geq 1$:

$$f_1(n) = \max_{1 \leq i \leq n} (p_i + f_1(n-i)), \qquad f_2(n) = \max_{1 \leq i \leq \lfloor \log n \rfloor} (p_i + f_2(n-i))$$

$$f_3(n) = p_n + f_3(n-1), \qquad f_4(n) = \max_{\substack{1 \leq i \leq n \\ 1 \leq k \leq i}} (k^2 p_i + f_4(n-k)),$$

where $p_1, \ldots, p_n$ are positive integers.

Give tight asymptotic running times (using the $\Theta(\cdot)$ notation) of the bottom-up dynamic programming implementations for calculating $f_1(n), f_2(n), f_3(n)$, and $f_4(n)$. We assume that the implementation of the bottom-up does not use any special properties of the recursions, i.e., it fills in each cell in the table using exactly the definition of the recursion.

(In this problem, you only need to give the answer, i.e., no explanations are needed. In particular, you do not need to explain the bottom-up dynamic programs.)

Solution:

Using the bottom-up dynamic programming technique (without using any special properties of the recursions)

- the asymptotic running time for calculating $f_1(n)$ is _____
- the asymptotic running time for calculating $f_2(n)$ is _____
- the asymptotic running time for calculating $f_3(n)$ is _____
- the asymptotic running time for calculating $f_4(n)$ is _____

2 *(15 pts)* **Hash tables.**

2a *(7 pts)* **Illustrate/draw the hash table** $T[0, \ldots, 6]$ obtained after inserting the keys $3, 9, 7, 24, 0, 14$ in the given order using the hash function $h(k) = k \mod 7$. Collisions are resolved using chaining with double-linked lists.

(Note that you only need to draw *one* hash table: the one obtained after inserting *all* the above keys.)

Solution:

2b *(8 pts)* Suppose you have a hash table with $2n$ slots and suppose that n distinct keys are inserted into the table. Each key is equally likely to be hashed into each slot (*simple uniform hashing*).

Prove that the expected number of collisions is $(n-1)/4$. Recall that we say that two keys k_i and k_j with $i \neq j$ collide if they are hashed to the same slot. The number of collisions is the number of pairs of keys that collide.

Solution:

3 *(5 pts)* **Spanning trees.** Let $G = (V, E)$ be a connected undirected graph with edge weights $w : E \to \mathbb{R}$. Consider an arbitrary edge $e \in E$. Professor Homer Simpson thinks the following statement is always true:

"e is contained in a minimum spanning tree **or** e is contained in a maximum spanning tree."

Show that Homer is wrong by **giving a counterexample** to this statement.

Solution:

4 *(11 pts)* **Basic algorithm design.** It is well-known that finding the minimum in an array of n numbers requires time $\Theta(n)$. However, if we are given more information about the structure of this array, we might be able to do it more efficiently. For instance, if the array is sorted, then one can find the minimum in constant time.

In this problem, we are given an array $A = [a_1, \ldots, a_n]$ of n numbers, with the promise that there exists an index $\ell \in \{1, 2, \ldots, n\}$ such that

$$a_1 > a_2 > \cdots > a_\ell \qquad \text{and} \qquad a_\ell < a_{\ell+1} < \cdots < a_n$$

Note that in this case, the minimum in this array is a_ℓ.

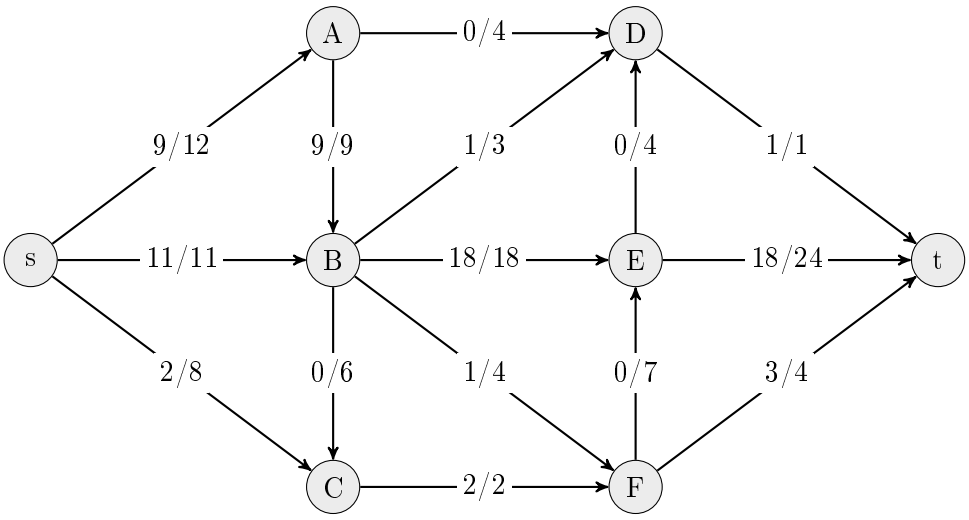
Design and analyze an algorithm that, given such an array A, returns the minimum element a_ℓ in time $O(\log n)$ (no points will be given for worse running times). You may assume that the array is always in the *correct* format, and the elements in the array are distinct, and hence the minimum is unique.

Solution:

Continuation of the solution to 4:

5 *(29 pts)* **Flows and Cuts.**

5a *(10 pts)* Consider the flow of value 22 in the following flow network (the numbers on an edge determine its current flow and its capacity).



Starting with the above flow, **find a max flow by running the Ford-Fulkerson method**. In each iteration, draw the residual network, and if there exists an augmenting path, indicate which one you selected and explain how the flow is updated along this path.

Solution:

Continuation of the solution to 5a:

5b *(6 pts)* Consider the same flow network as in Subproblem 5a and consider the min cut "corresponding" to the max flow that you found in that problem. Use the structure of this cut to **prove** (in a couple of sentences) that the max flow value must decrease if the capacity of arc (C, F) is decreased from 2 to 1.

Solution:

5c *(13 pts)* Thanks to the many excellent students (and professors :)), EPFL has seen a rapid increase in size and quality. However, there is one worry: are there enough professors and PhD students to cover all courses? In order to ensure high quality teaching we have the following constraints in assigning teaching staff to courses. Professors and PhD students can be assigned to at most one course in their expertise. Moreover, each course needs a course-dependent number of staff members, one of which needs to be a professor.

Formally, we wish to solve the following teaching assignment problem:

Input: a set C of courses where each course $c \in C$ has a requirement $r(c) \geq 1$, a set T of teaching staff, partitioned into a set P of professors and a set S of PhD students, where each staff member $t \in T$ can be assigned to a subset $C_t \subseteq C$ of courses (corresponding to his/her expertise).

Output: If possible, an assignment of teaching staff to the courses so that

- Each staff member $t \in T$ is assigned to at most one course and, if assigned, the assignment must be to a course in C_t.
- Each course $c \in C$ is assigned at least $r(c)$ teaching staff members.
- At least one professor is assigned to each course (note that it is also fine if more than one professor is assigned to a course).

If no such assignment exists, simply output "We need to hire!".

Your task is to **design and analyze** a polynomial time algorithm for the above problem.

(Hint: formulate a flow problem so that an (integral) max flow corresponds to an assignment if one exists. However, do not forget to explain how to turn a solution to the flow problem into a solution to the original problem.)

Solution:

Continuation of the solution to 5c:

Continuation of the solution to 5c:

6 *(25 pts)* **Shortest paths and dynamic programming.** Consider a directed graph $G = (V, E)$ with *positive* edge weights $w : E \to \mathbb{R}_{>0}$, i.e., $w(e) > 0$ for $e \in E$, and a vertex $s \in V$. We shall design and analyze an efficient algorithm that finds, for each $v \in V \setminus \{s\}$, the *number* of shortest paths from s to v. We divide this task into that of solving two subproblems.

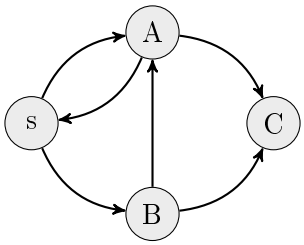
(Note that you can solve the second subproblem even if you did not manage to solve the first one.)

6a *(12 pts)* **Design and analyze** a polynomial time algorithm that finds a permutation $\pi : V \to \{1, 2, \ldots, n\}$ of the vertices so that

- $\pi(s) = 1$; and
- no shortest path from s to another vertex uses a "back-edge", that is, an edge $(u, v) \in E$ such that $\pi(u) > \pi(v)$.

For full score, you should argue why your algorithm returns a permutation that satisfies the above properties.

Example: Consider the following graph where all edges have weight 1:



A permutation satisfying the required properties would be $\pi(s) = 1, \pi(A) = 2, \pi(B) = 3, \pi(C) = 4$ because no shortest path from s uses the edges (B, A) and (A, s). Also note that the number of shortest paths from s to A is 1, from s to B is 1, and from s to C is 2.

Solution:

Continuation of the solution to 6a:

6b *(13 pts)* Use the permutation π of the vertex set V from Subproblem 6a to **design and analyze** a polynomial time dynamic programming algorithm that fills in a table/array c indexed by the vertices in V so that $c[s] = 1$ and, for $v \in V \setminus \{s\}$, $c[v]$ equals the number of shortest paths from s to v.

(Hint: suppose that you calculated $c[u]$ for all u with $\pi(u) < \pi(v)$. How would you use this information to calculate $c[v]$?)

Solution:

Continuation of the solution to 6b: