ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Final Exam January 16, 2013

Algorithms

January 16, 2013

- You are only allowed to have an $A4$ page written on both sides.
  (Vous avez droit à un formulaire manuscrit sur une page $A4$ recto-verso.)
- Calculators, cell phones, computers, etc... are not allowed.
  (Aucun matériel électronique (calculatrices, téléphones portables, ordinateurs, etc) n'est permis.)

Last name :

First name :

Section :

Exercise 1 / 5 points	Exercise 2 / 5 points	Exercise 3 / 10 points	Exercise 4 / 20 points	Exercise 5 / 20 points	Exercise 6 / 20 points

Exercise 7 / 20 points

Total / 100

Problem 1 [5 points]. Consider the natural implementation of the three sorting algorithms (Insertion-Sort, Merge-Sort, Heap-Sort) that we studied in class. Which one of these implementations is *not* "in-place"?

French translation. Considérez les implémentations naturelles des trois algorithmes de tri (Insertion-Sort, Merge-Sort, Heap-Sort) que nous avons étudiés en classe. Laquelle de ces implémentations n'est pas "in-place"?

Problem 2 [5 points]. Simplify and arrange the following functions in increasing order according to asymptotic growth.

$$3^N, \sqrt{4^N}, \log^2 N, \sqrt{N}, N^2, \log N, 20N$$

French translation. Simplifier et ordonner les fonctions ci-dessus dans l'ordre croissant de leurs ordres de croissance.

Problem 3 [10 points]. Let $f(n)$ and $g(n)$ be the functions defined for positive integers as follows:

```
Function f(n):
1: ans ← 0
2: for i = 1, 2, ..., n − 1 do
3:     for j = 1, 2, ..., n − i do
4:         ans ← ans + 1
5:     end for
6: end for
7: return ans
```

```
Function g(n):
1: if n = 1 then
2:     return 1
3: else
4:     return g(⌊n/2⌋) + g(⌊n/2⌋)
5: end if
```

1. What is, in Θ notation, the running time of these algorithms, given that addition runs in time $\Theta(1)$? *(6 points)*

2. What is, in Θ notation, the running time of algorithm $g(n)$ if we at line 4 replace $g(\lfloor n/2 \rfloor) + g(\lfloor n/2 \rfloor)$ by $2g(\lfloor n/2 \rfloor)$? *(4 points)*

   (Assume that multiplication runs in time $\Theta(1)$.)

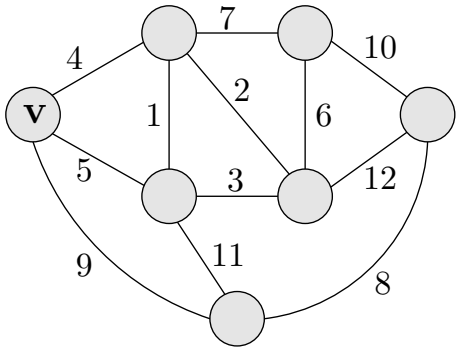
French translation. Soient $f(n)$ et $g(n)$ les fonctions définies ci-dessus sur les entiers positifs.

1. En notation Θ, quel est le temps de parcours de ces algorithmes, étant donné que le temps de parcours d'une addition est $\Theta(1)$? *(6 points)*

2. En notation Θ, quel est le temps de parcours de l'algorithme $g(n)$, si on remplace, à la ligne 4, $g(\lfloor n/2 \rfloor) + g(\lfloor n/2 \rfloor)$ par $2g(\lfloor n/2 \rfloor)$? *(4 points)*
   (On suppose que le temps de parcours d'une multiplication est $\Theta(1)$.)

Problem 4 [20 points]. **Spanning Trees**

Consider the following minimum spanning tree instance, i.e., an undirected connected graph with weights on the edges.



1. Write the weights of the edges of the minimum spanning tree in the order they are added by Prim's algorithm starting from vertex v. *(5 points)*

2. Write the weights of the edges of the minimum spanning tree in the order they are added by Kruskal's algorithm. *(5 points)*

3. A bottleneck edge is an edge of highest weight in a spanning tree. A spanning tree is a minimum bottleneck spanning tree if the graph does not contain a spanning tree with a smaller bottleneck edge weight. *Show that a minimum spanning tree is also a minimum bottleneck spanning tree.* *(10 points)*

French translation. **Arbres couvrants**

On considère l'instance ci-dessus d'un arbre couvrant minimal, c'est-à-dire un graphe connexe avec des poids sur les arêtes.

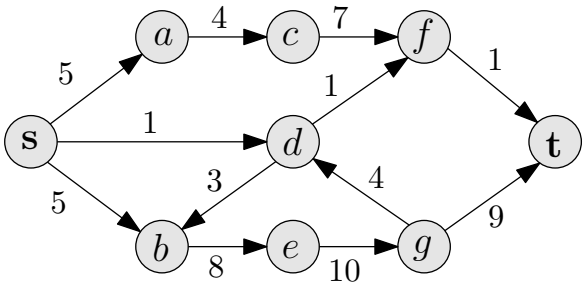
1. Donner les poids des arêtes de l'arbre couvrant minimal dans l'ordre où ces arêtes sont ajoutées par l'algorithme de Prim commençant au sommet v. *(5 points)*

2. Donner les poids des arêtes de l'arbre couvrant minimal dans l'ordre où ces arêtes sont ajoutées par l'algorithme de Kruskal. *(5 points)*

3. Une arête de poids maximal dans un arbre couvrant est dite "arête bottleneck". Un arbre couvrant est dit "bottleneck-minimal" si le graphe ne contient aucun autre arbre couvrant avec une arête bottleneck de plus petit poids. *Montrer qu'un arbre couvrant minimal est aussi bottleneck-minimal.* *(10 points)*

Problem 5 [20 points]. **Flows and Cuts**

Consider the following flow network:



1. Find a max-flow and a min-cut by running the Ford-Fulkerson algorithm that in each iteration chooses a shortest augmenting path (a shortest path can be found by using BFS). *(15 points)*

   (In each iteration draw the residual network and explain how you found the min-cut.)

2. What is the value of the max-flow? What is the capacity of the min-cut? *(5 points)*

French translation. **Flux et Cuts**

Considérer le réseau de flux ci-dessus.

1. Trouver un flux maximal et un cut minimal en utilisant l'algorithme de Ford-Fulkerson, qui choisit à chaque itération un chemin augmentant le plus court (un chemin le plus court peut être obtenu avec l'algorithme BFS). *(15 points)*

   (A chaque itération, donner le réseau résiduel et expliquer comment vous avez trouvé le cut minimal.)

2. Quelle est la valeur du flux maximal? Quelle est la capacité du cut minimal? *(5 points)*

Problem 6 [20 points]. Let $x_1...x_m$ and $y_1...y_n$ be two strings. For $0 \leq i \leq m$ and $0 \leq j \leq n$, let $c[i,j]$ be the length of the longest common subsequence of $x_1...x_i$ (or the empty string if $i = 0$) and $y_1...y_j$ (or the empty string if $j = 0$). In other words, $c[i,j]$ is defined to be the maximum k such that there exist indices $1 \leq i_1 < i_2 < \cdots < i_k \leq i$ and $1 \leq j_1 < j_2 < \cdots < j_k \leq j$ satisfying $x_{i_\ell} = y_{j_\ell}$ for all $\ell = 1, \ldots, k$.

1. Complete the recurrence relation for $c[i,j]$ that can be used for dynamic programming:

$$c[i,j] = \begin{cases} \_\_\_\_\_ & \text{if } i = 0 \text{ or } j = 0 \\ \_\_\_\_\_ & \text{if } i,j > 0 \text{ and } x_i = y_j \\ \_\_\_\_\_ & \text{if } i,j > 0 \text{ and } x_i \neq y_i \end{cases} \qquad \textit{(10 points)}$$

2. Use the recurrence relation to return the length of the longest common subsequence of the two strings BABDBA and DACBCBA by filling in the table of $c[i,j]$ values below (in a bottom-up dynamic programming fashion). *(10 points)*

	j	0	1	2	3	4	5	6	7
i		y_j	D	A	C	B	C	B	A
0	x_i								
1	B								
2	A								
3	B								
4	D								
5	B								
6	A								

French translation. Soient $x_1...x_m$ et $y_1...y_n$ deux mots. Pour $0 \leq i \leq m$ et $0 \leq j \leq n$, soit $c[i,j]$ la longueur de la plus longue sous-séquence commune de $x_1...x_i$ (ou le mot vide si $i = 0$) et $y_1...y_j$ (ou le mot vide si $j = 0$). En d'autres termes, c[i,j] est défini comme le plus grand k tel qu'il existe des indices $1 \leq i_1 < i_2 < \cdots < i_k \leq i$ et $1 \leq j_1 < j_2 < \cdots < j_k \leq j$ avec $x_{i_\ell} = y_{j_\ell}$ pour tout $\ell = 1, \ldots, k$.

1. Compléter la relation de récurrence pour $c[i,j]$ qui pourra être utilisée pour la programmation dynamique:

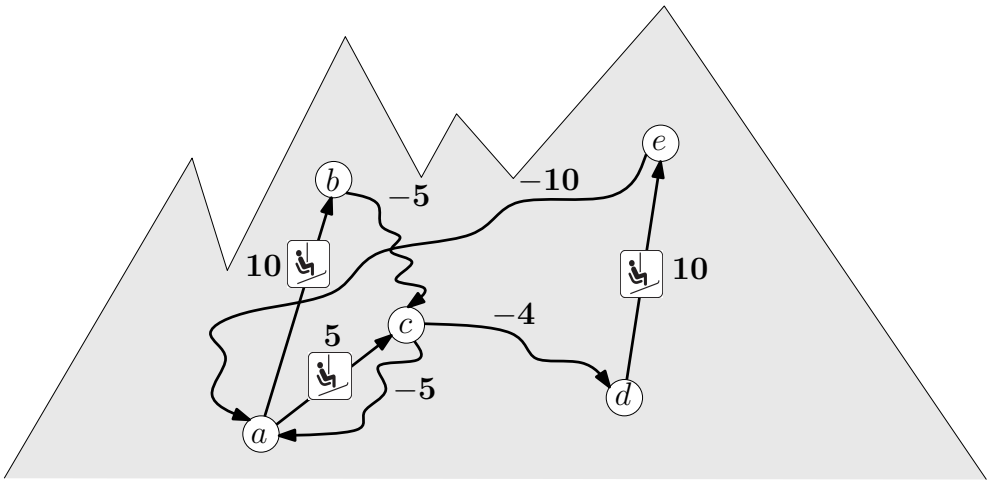
$$c[i,j] = \begin{cases} \_\_\_\_\_ & \text{si } i = 0 \text{ ou } j = 0 \\ \_\_\_\_\_ & \text{si } i,j > 0 \text{ et } x_i = y_j \\ \_\_\_\_\_ & \text{si } i,j > 0 \text{ et } x_i \neq y_i \end{cases} \qquad \textit{(10 points)}$$

2. Utiliser la relation de récurrence pour donner la longueur de la plus longue sous-séquence commune des deux mots BABDBA and DACBCBA en remplissant la table ci-dessus avec les valeurs de $c[i, j]$ (utiliser la programmation dynamique bottom-up). *(10 points)*

.....

Problem 7 [20 points]. The famous alpine ski racer Lindsey Vonn has sent her assistant to measure the altitude differences of the ski lifts and slopes of a famous ski resort in Switzerland. The assistant returns after several days of hard work with a map of all ski lifts and all slopes together with the altitude difference between the start station and the end station of each lift and the altitude difference between the start station and end station of each slope. A slope starts from the end station of a lift and ends at the start station of a potentially different lift (see the figure below).

Lindsey wants to verify the map of her assistant by performing the following sanity check: starting from any point A, no matter how we ski (using lifts and slopes), the altitude change when we return to A should be 0 (i.e., neither strictly negative nor strictly positive). Design an algorithm to perform this sanity check and analyze its running time in terms of the number of slopes and ski lifts ($|E|$) and the number of start and end stations ($|V|$). Your algorithm should run in polynomial time (in $|V|$ and $|E|$). *(20 points)*



French translation. La célèbre skieuse de ski alpin Lindsey Vonn envoie son assistant mesurer la différence d'altitude des remontées mécaniques et des pistes d'une certaine station de ski en Suisse. Après plusieurs jours de travail, l'assistant revient avec une carte de toutes les remontées mécaniques et de toutes les pistes, avec la différence d'altitude entre la première et la dernière station de chaque remontée, et la différence d'altitude entre la première et la dernière station de chaque piste. Une piste commence à la dernière station d'une certaine remontée, et finit à la première station d'une certaine remontée (pas nécessairement la même). (voir la figure ci-dessus).

Lindsey veut vérifier la carte de son assistant de la manière suivante: si on commence à n'importe quel point A, de quelque manière qu'on skie (en utilisant les remontées et les pistes), la différence d'altitude quand on retourne en A doit être nulle (donc ni strictement positive, ni strictement négative). Créer un algorithme qui performe cette vérification. Analyser le temps de parcours de votre algorithme en fonction du nombre de pistes et de remontées ($|E|$) et du nombre de stations de départ et d'arrivée ($|V|$). Votre algorithme doit avoir un temps de parcours polynomial (en $|V|$ et $|E|$). *(20 points)*