Examen d'algorithmique et complexité, M1 informatique, Dijon, janvier 2010. Répondez dans l'ordre aux questions, et donnez le numéro de la question. Vous répondrez en 2 lignes au plus à toutes les questions, sauf mention contraire : **les réponses trop longues sont notées 0**. Chaque question est notée sur 1 point. Toutes les questions ont été vues en cours (CM/TD/TP).

1. Citez 3 algorithmes (distincts...) en temps $O(n^3)$.

2. Citez les 2 algorithmes pour calculer l'arbre couvrant optimal vus en cours.

3. Citez 3 représentations pour les graphes.

4. Citez 3 algorithmes pour les problèmes de plus courts chemins (5 ont été vus et programmés)

5. Si un algorithme de tri ne peut utiliser que des comparaisons entre éléments, quelle est sa meilleure complexité possible ?

6. Nous avons vu un algorithme de tri d'un ensemble d'entiers, qui utilisait autre chose que des comparaisons entre les entiers. Quel est son nom, et quelle est sa complexité ?

7. Citer 1 algorithme pour résoudre les problèmes de programmation linéaire.

8. Citez 3 problèmes pour lesquels il n'existe pas ou on ne connait pas d'algorithme en temps polynomial.

9. La programmation dynamique a été utilisée en TP pour résoudre 2 problèmes. Lesquels ? (Indication : les tables de hachage ont aussi été utilisées à cette occasion).

10. Combien d'additions et de multiplications (de grands entiers) nécessite le calcul de fibonacci(n), dans le meilleur algorithme programmé en TP ?

11. Citer 2 représentations utilisées pour les polynômes.

12. Une entreprise de chimie doit produire en 5 jours (1, 2, 3, 4, 5) plusieurs produits chimiques P1, P2, ... Pn. Pour des raisons de sécurité, la législation interdit de fabriquer certains produits le même jour. Il y a donc une liste de paires interdites de produits. L'entreprise doit décider quels produits fabriquer chaque jour 1, 2, 3, 4, 5. Modéliser le problème par un graphe : quels sont les sommets ? quelles sont les arêtes ? A quelles conditions sur le graphe l'entreprise peut elle produire les P_i ? **Réponse en 4 lignes au plus.**

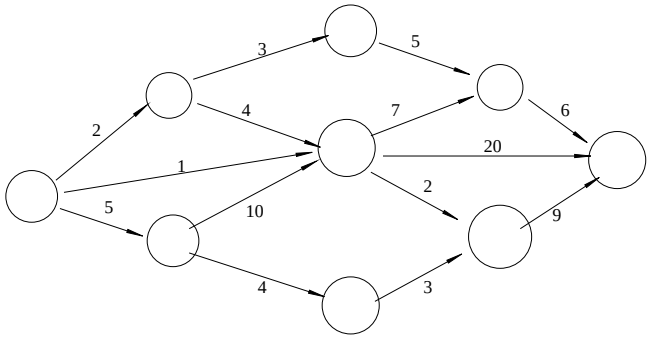
13. Suite de la question précédente : certains produits ne peuvent être fabriqués qu'après certains autres. Un deuxième graphe, orienté, représente ces relations : $P_i \rightarrow P_j$ ssi P_i doit être fabriqué avant P_j. Quelle sont les conditions sur les graphes, pour que l'entreprise puisse fabriquer les produits ? Remarque : il est interdit de fabriquer certains produits le même jour, mais pas de les utiliser ensemble après qu'ils sont fabriqués. **Réponse en 4 lignes au plus.**

14. Le calcul de a^k, avec k un entier naturel, peut il être fait avec moins de $k-1$ multiplications ? Si oui, combien ?

15. Quels sont les noms (français ou anglais) de la méthode que nous avons utilisée pour résoudre le problème des reines ?

16. Citer 3 méthodes efficaces de tri, qui n'utilisent que des comparaisons entre éléments à trier (ceci exclut le tri par bulles, et les tris naifs).

17. Notez les dates au plus tôt et au plus tard sur ce graphe orienté. Encerclez les sommets critiques.



18. Embêter le fisc. Vous devez payer une somme c, entière, avec des pièces de monnaie. Vous avez n types de pièces, de valeurs $a_0, a_1, \ldots a_{n-1}$, toutes différentes. 1 fait partie des a_i. Le nombre de vos pièces de valeur a_i est suffisant pour payer c ou davantage ; par exemple, vous pouvez payer c avec c pièces de 1. Mais cela maximise le nombre de pièces nécessaires. Or vous souhaitez minimiser le nombre de pièces utilisées (le coût de la poste est à votre charge). Pour un ordre donné des a_i, l'algorithme glouton utilise autant de pièces de valeurs a_i que possible, sans dépasser la somme qu'il reste à payer. Complétez la fonction rendre dans cette session ocaml, sachant qu'elle utilise l'algorithme glouton :

```
$ ocaml
# let rec rendre s pieces = match pieces with
| [] -> []
| p1::qpieces -> if s < p1 then rendre s qpieces
                 else (.......):: (rendre (.......) ....) ;;
val rendre : int -> int list -> (int * int) list = <fun>
# rendre  26 [20; 10; 5; 2; 1 ];;
- : (int * int) list = [(1, 20); (1, 5); (1, 1)]
# rendre  26 [1; 2; 5; 10; 20 ];;
- : (int * int) list = [(26, 1)]
# rendre 263 [100; 10; 1];;
- : (int * int) list = [(2, 100); (6, 10); (3, 1)]
# rendre 263 [1; 10; 100] ;;
- : (int * int) list = [(263, 1)]
```

19. (Suite) La solution de l'algorithme glouton pour $c = 26$, $a = [20; 10; 5; 2; 1]$ est elle optimale ? Dire pourquoi **(5 lignes max)**.

20. (Suite) Le problème des pièces de monnaie peut-il s'exprimer comme un problème de sac à dos (vu en cours) ? Si oui, donnez cette formulation. Sinon, pourquoi ? **(3 lignes max)**.

1. Citez 3 algorithmes en temps $O(n^3)$.

Produit de 2 matrices n par n par la méthode usuelle. Inversion d'une matrice par la méthode de Gauss. Plus courts chemins par Dantzig, Floyd, Ford (ou Bellman-Ford). Méthode CYK pour l'analyse syntaxique.

2. Citez les 2 algorithmes pour calculer l'arbre couvrant optimal vus en cours.

Kruskal. Prim

3. Citez 3 représentations pour les graphes

Matrice. Tableau de listes des voisins (ou des arcs). Fonction (exemple : un sommet est un échiquier partiellement rempli dans le problème des reines ; un sommet est un sommet du polytope en programmation linéaire, et les arêtes du graphe sont les arêtes du polytope).

4. Citez 3 algorithmes pour les problèmes de plus courts chemins (5 ont été vus et programmés)

Dijkstra, Dantzig, Floyd (ou Floyd - Warshall), Ford (ou Bellman-Ford), élévation au carré de la matrice avec une multiplication spéciale : $M[l][c] = \min_k A[l][k] + B[k][c]$

5. Si un algorithme de tri ne peut utiliser que des comparaisons entre éléments, quelle est sa meilleure complexité possible ?

$O(n \log n)$. En effet, il y a $n!$ ordres possibles, et chaque comparaison élimine au mieux la moitié des ordres encore possibles. Or $\log n!$ croît comme $n \log n$.

6. Nous avons vu un algorithme de tri d'un ensemble d'entiers, qui utilisait autre chose que des comparaisons entre les entiers. Quel est son nom, et quelle est sa complexité ?

radix sort, ou tri par base. Il est en $O(kbn)$: n est le nombre d'entiers à trier. k est le nombre max de chiffres des entiers à trier dans la base b. Pour des entiers machines ($k = 32, b = 2$ par exemple) cette méthode est linéaire : $O(64n) = O(n)$.

7. Citer 1 algorithme pour résoudre les problèmes de programmation linéaire.

Le simplexe de Dantzig (primal, ou dual) et ses variantes. La méthode de l'ellipsoïde. Les méthodes par point intérieur. Le simplexe englobant (l'ellipsoïde est remplacé par un simplexe).

8. Citez 3 problèmes pour lesquels il n'existe pas ou on ne connait pas d'algorithme en temps polynomial.

Le problème 3-SAT. La clique max (or de cardinal donné) dans un graphe. Le stable max (or de cardinal donné) dans un graphe. Le coloriage des sommets d'un graphe en 3 couleurs (ou plus). L'isomorphisme entre 2 graphes. La factorisation des grands entiers. La résolution de $a^x = k$ dans un corps fini (modulo un premier par exemple). L'arrêt de la machine de Turing. Chemin hamiltonien. Circuit hamiltonien. Sac à dos. Problème du voyageur de commerce.

9. La programmation dynamique a été utilisée en TP pour résoudre 2 problèmes. Lesquels ? (Indication : les tables de hachage ont aussi été utilisées à cette occasion).

Le calcul optimal (minimisant la quantité de multiplications entre 2 nombres flottants) du produit de matrices : M1 M2 ..Mn. Le sac à dos avec des poids entiers. La plus longue sous séquence commune entre 2 séquences peut aussi être calculée par programmation dynamique, mais cela n'a pas été vu en cours.

10. Combien d'additions et de multiplications (de grands entiers) nécessite le calcul de fibonacci(n), dans le meilleur algorithme programmé en TP ?

L'algorithme nécessite $\log(n)$ produits de matrices carrées 2 par 2. Il a donc besoin de $\log(n)$ additions et multiplications. Par contre, le coût de ces opérations n'est pas constant.

Complément : d'après la formule de Binet, F_n vaut $(\phi^n - \phi'^n)/\sqrt{5}$ où ϕ est le

nombre d'or : $(1+\sqrt{5})/2 \approx 1.618...$, et $\phi' = (1-\sqrt{5})/2 \approx -0.618...$ est son congugué. On sait aussi que $F_n = (\phi^n - \phi'^n)/\sqrt{5} = \lfloor \phi^n/\sqrt{5} \rceil$ (indication : très vite, $|\phi'^n|$ tend vers 0). On peut donc approcher en temps constant ($O(1)$) la valeur de F_n en utilisant une de ces deux expressions avec l'arithmétique flottante : pour des valeurs de $n \leq 70$, l'approximation flottante est exacte. Après, la mantisse des flottants (format double) n'est plus assez longue pour représenter ϕ^n exactement.

11. Citer 2 représentations utilisées pour les polynômes.

Les graphes orientés sans cycles ou DAG (directed acyclic graphs) : ce sont "des arbres où les noeuds peuvent être partagés". Les arbres. Les listes (ordonnées ou non) de monômes. Les tableaux de coefficients à n dimensions, s'il y a n variables. Les fonctions. Les ensembles de paires (point, valeur du polynôme en ce point).

12. Une entreprise de chimie doit produire en 5 jours (1, 2, 3, 4, 5) plusieurs produits chimiques P1, P2, ... Pn. Pour des raisons de sécurité, la législation interdit de fabriquer certains produits le même jour. Il y a donc une liste de paires interdites de produits. L'entreprise doit décider quels produits fabriquer chaque jour 1, 2, 3, 4, 5. Modéliser le problème par un graphe : quels sont les sommets ? quelles sont les arêtes ? A quelles conditions sur le graphe l'entreprise peut elle produire les P_i ?

Chaque produit P_i est un sommet P_i. Une arête lie P_i et P_j quand P_i, P_j est une paire interdite. Il faut que les sommets du graphe puissent être coloriés en 5 couleurs (une pour chaque jour), 2 sommets liés par une arête doivent être de couleurs différentes.

13. Suite de la question précédente : certains produits ne peuvent être fabriqués qu'après certains autres. Un deuxième graphe, orienté, représente ces relations : $P_i \rightarrow P_j$ ssi P_i doit être fabriqué avant P_j. Quelle sont les conditions sur les graphes, pour que l'entreprise puisse fabriquer les produits ? Remarque : il est interdit de fabriquer certains produits le même jour, mais pas de les utiliser ensemble une fois qu'ils sont fabriqués.

Pas de cycle dans le graphe orienté. De plus l'arc $P_i \rightarrow P_j$ ajoute des paires interdites (donc des arêtes) dans le graphe non orienté. Dans le graphe non orienté, il faut pouvoir colorier les sommets avec des numéros de jours (dans 1–5) qui vérifient des contraintes supplémentaires : si $P_i \rightarrow P_j$ dans le graphe orienté, la couleur de P_i doit être inférieure à la couleur de P_j.

14. Le calcul de a^k, avec k un entier naturel, peut il être fait avec moins de $k-1$ multiplications ? Si oui, combien ?

$O(\log k)$

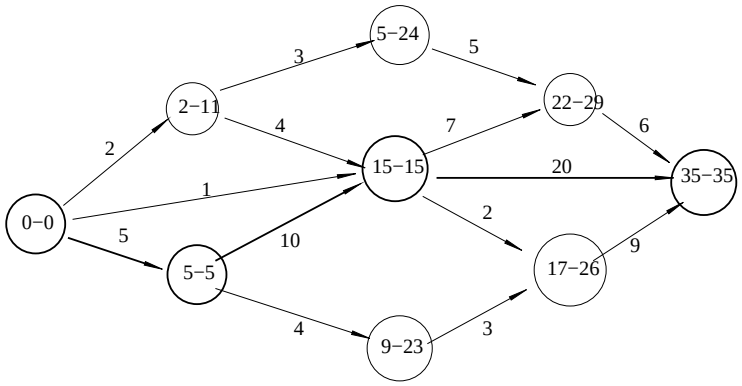
15. Quels sont les noms (français ou anglais) de la méthode que nous avons utilisée pour résoudre le problème des reines ?

Avec DM, recherche en profondeur avec retour arrière, depth first search, backtrack. Avec JJC, métaheuristiques : recuit simulé, tabou.

16. Citer 3 méthodes efficaces de tri, qui n'utilisent que des comparaisons entre éléments à trier (ceci exclut le tri par bulles, et les tris naifs).

Le tri par fusion (mergesort). Le tri par tas (heapsort). Insertion dans un arbre équilibré quelconque (il y en a beaucoup), et suppression itérée du plus petit élément de l'arbre. Le tri rapide (quicksort), mais il peut être quadratique si on est malchanceux.

17. Notez les dates au plus tôt et au plus tard sur ce graphe orienté. Encerclez les sommets critiques.



18. Vous devez payer une somme c, entière avec des pièces de monnaie. Vous avez n types de pièces, de valeurs $a_0, a_1, \ldots a_{n-1}$, toutes différentes. 1 fait partie des a_i. Le nombre de vos pièces de valeur a_i est suffisant pour payer c ou davantage ; par exemple, vous pouvez payer c avec c pièces de 1. Mais vous souhaitez minimiser le nombre de pièces utilisées.

Pour un ordre donné des a_i, l'algorithme glouton utilise autant de pièces de valeurs a_i que possible, sans dépasser la somme qu'il reste à payer. Complétez la fonction rendre dans cette session ocaml (**4 lignes**) :

```
$ ocaml
        Objective Caml version 3.10.0
# let rec rendre s pieces = match pieces with
| [] -> []
| p1::qpieces -> if s < p1 then rendre s qpieces
                 else (s / p1, p1 ):: (rendre (s mod p1) qpieces) ;;
val rendre : int -> int list -> (int * int) list = <fun>
# rendre  26 [20; 10; 5; 2; 1 ];;
- : (int * int) list = [(1, 20); (1, 5); (1, 1)]
# rendre  26 [1; 2; 5; 10; 20 ];;
- : (int * int) list = [(26, 1)]
# rendre  26 [1; 6; 7];;
- : (int * int) list = [(26, 1)]
# rendre  26 [7; 6; 1];;
- : (int * int) list = [(3, 7); (5, 1)]
# rendre 263 [100; 10; 1];;
- : (int * int) list = [(2, 100); (6, 10); (3, 1)]
# rendre 263 [1; 10; 100] ;;
- : (int * int) list = [(263, 1)]
```

19. (Suite) La solution de l'algorithme glouton pour $c = 26$, $a = [20; 10; 5; 2; 1]$ est elle optimale ? Dire pourquoi **(5 lignes max)**.

La méthode gloutonne trouve une solution en 3 pièces ; elle est optimale car il n'existe pas de solution en 1 pièce ($26 \notin [20; 10; 5; 2; 1]$), ni de solution en 2 pièces : l'ensemble des $x + y$ avec $x \in [20; 10; 5; 2; 1], y \in [20; 10; 5; 2; 1]$) est :

$$[40; 30; 25; 22; 21; 30; 20; 15; 12; 11; 25; 15; 10; 7; 6; 22; 12; 7; 4; 3; 21; 11; 6; 3; 1]$$

20. (Suite) Le problème des pièces de monnaie peut-il s'exprimer comme un

problème de sac à dos (vu en cours) ? Si oui, donnez cette formulation. Sinon, pourquoi ? (Réponse en 3 lignes maximum).

Oui.

Le sac à dos consiste à maximiser l'utilité $\sum_{i=0}^{n-1} u_i x_i$ d'un sac à dos d'alpiniste sous les contraintes $\sum_{i=0}^{n-1} p_i x_i \leq P$, et $x_i \in \{0, 1\}$. Les u_i sont des utilités, les p_i sont les poids, P est le poids maximum pour l'alpiniste.

Il faut se mettre à la place de Harpagon. Sa fortune est F, très supérieure à c, ce qu'il doit payer. Harpagon veut garder le maximum d'argent, et le maximum de pièces. Il veut maximiser $\sum x_i p_i + \sum x_i = \sum x_i(p_i + 1)$, avec $x_i \in 0, 1$, et sous la contrainte $\sum x_i p_i \leq F - c$ (il s'agit bien d'un $\leq$, Harpagon ne peut pas garder tout son argent !). C'est bien un problème de sac à dos. Il y a un point délicat, cependant : Harpagon souhaite maximiser 2 choses (l'argent qui lui reste, et le nombre de pièces) ; suffit-il simplement de maximiser la somme des deux ? Comment être sûr que le maximum se produit bien pour $\sum x_i p_i = F - c$? Un raisonnement par l'absurde le prouve, pour les personnes qui ne savent pas se mettre à la place de Harpagon :

Supposons que le maximum se produise pour $\sum x_i p_i < F - c$, plus précisément $\sum x_i p_i = F - c - k$. Ici k est un entier positif ; en d'autres termes Harpagon donne k euros de trop... Alors, en respectant la contrainte, Harpagon peut, par exemple, garder k pièces de 1 euro, ce qui accroît la valeur du maximum, de $2k$ (il peut sûrement faire mieux, mais nous n'en avons pas besoin pour la preuve...). C'est absurde. CQFD.