CentraleSupélec

1re année 2018–2019
ST2, campus de Saclay

Vendredi 25 janvier 2018, matin

Examen écrit de

Algorithme et Compléxité

Durée : 3 heures

Numéro étudiant :

Nom : Prénom :

— Tous les documents sont autorisés.
— Les outils électroniques (calculatrices, ordinateurs et téléphones) sont interdits.
— Tous vos algorithmes peuvent être écrits en pseudo-code ou en Python, selon ce que vous préférez.
— Le barème est donné à titre indicatif.
— Ce sujet comporte 12 pages.
— Vous pouvez utiliser les pages supplémentaires en fin de sujet si vous manquez de place.
— Les exercices sont indépendants.

Exercice 1 : Les amis de mes amis — 3 point(s)

On considère un réseau social qui, pour chaque utilisateur, stocke sa liste d'amis. On dispose ainsi d'une fonction `friends(u)` qui retourne la liste des amis de l'utilisateur `u`.

On souhaite écrire un algorithme qui propose de nouveaux contacts à partir des amis de nos amis.

Question 1 — 2 point(s)

Écrivez, en pseudo-code ou en Python, une fonction `propose(u)` qui prend en argument un utilisateur `u` et qui propose un **autre** utilisateur qui :

1. ne figure pas déjà dans la liste des amis de `u` ;
2. apparaît le plus fréquemment dans les listes d'amis des amis de `u`.

Question 2 1 point(s)

Pour chaque utilisateur nous souhaitons connaître l'ensemble des utilisateurs qui sont soit ses amis directs, soit pour lesquels il existe une suite des amis qui peuvent servir d'intermédiaires. Autrement dit, nous voulons construire la *fermeture transitive* du graphe de notre réseau social.

Écrivez un algorithme qui fait cette construction.

Exercice 2 : Couverture par les sommets — 4 point(s)

La RATP souhaite installer des caméras dans la station de métro Châtelet de manière à pouvoir suivre les déplacements des voyageurs au sein de la station. Les caméras seront installées aux intersections des couloirs : elles permettent de repérer toute personne passant par l'un des couloirs reliés à cette intersection.

L'objectif est de pouvoir surveiller tous les couloirs de la station. Pour des raisons de coût, on souhaite placer le moins de caméras possibles.

Question 1 — 1.5 point(s)

Modélisez ce problème. Quelle est la nature du problème ? Quelles sont les entrées et quelle est la question posée ?

Nous considérons l'algorithme suivant qui permet de construire une solution au problème posé. Il prend en paramètre la liste des couloirs *corridors* dont les éléments sont des couples d'intersections $(i1, i2)$. Chaque couloir est ainsi identifié par les deux points de la station qu'il relie entre eux.

```
def compute_solution(corridors)

  R = corridors.copy()
  S = []

  while len(R)>0:
    (i1,i2) = R.pop()

    S.append(i1)
    S.append(i2)

    R2 = []
    for r in R:
      if r[0]!=i1 and r[1]!=i1 and r[0]!=i2 and r[1]!=i2:
        R2.append(r)
    R = R2

  return S
```

Question 2 — 1.5 point(s)

Que fait cet algorithme ? Expliquez son fonctionnement en français en utilisant des termes de votre propre modélisation. Indiquez à quelle famille d'algorithmes il appartient. Calculez sa complexité en temps. Justifiez chacune de vos réponses.

Question 3 1 point(s)

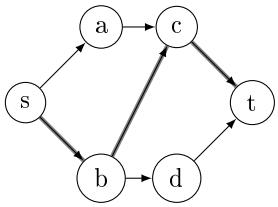
Montrez que cet algorithme est une 2-approximation, c'est-à-dire qu'il ne donne jamais une solution S de taille supérieure à 2 fois la taille des solutions optimales.

Exercice 3 : Recherche de chemins **3 point(s)**

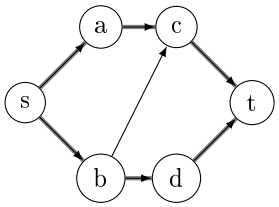
On s'intéresse au problème suivant :

> **MAX-EDGE-DISJOINT-PATHS**
> **Entrée** : Un graphe orienté $G = (V, E)$ et deux sommets s et t de G.
> **Question** : Construire le plus grand nombre possible de chemins de s à t sans utiliser deux fois le même arc.

Chaque arc ne peut donc être utilisé **que dans un seul chemin**. Le but est d'obtenir le plus de chemins possibles. Sur l'exemple ci-dessous, il est possible de construire une solution à deux chemins (c'est le maximum).



Solution à 1 chemin



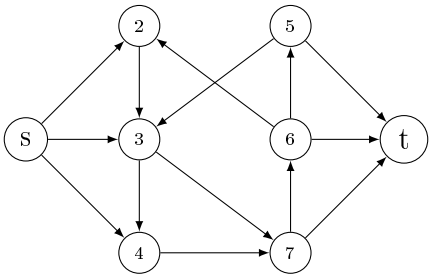
Solution à 2 chemins

Question 1 2 point(s)

Donnez un algorithme polynomial qui calcule le **nombre** maximum de chemins possibles. Justifiez votre réponse.

Question 2 1 point(s)

Déroulez cet algorithme sur le problème suivant en détaillant bien toutes les étapes. Quel est le nombre maximum de chemins?



Exercice 4 : Une soirée « fun » 5 point(s)

Une entreprise organise une soirée pour ses employés. Les organisateurs veulent que la soirée soit la plus divertissante possible.

Note de « fun ». La capacité à plaisanter et à être de bonne humeur de chaque employé est connue et répertoriée par les organisateurs. Elle est exprimée par une note « fun » attribuée à chaque employé.

Jamais avec mon « n+1 ». Chaque employé a une position hiérarchique stricte dans l'entreprise. Il est alors impossible qu'un employé et son « n+1 » (c'est-à-dire son supérieur hiérarchique direct) soient tous les deux présents à la soirée, car cela ne serait pas « fun » du tout...

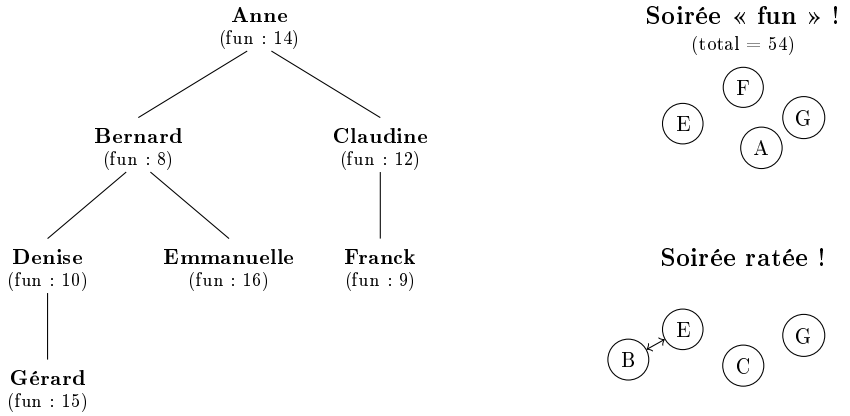


FIGURE 1 – Exemple simplifié de soirées dans une entreprise de 7 personnes

Le but est d'organiser une soirée telle que la somme des notes « fun » de tous les participants à la soirée soit maximale, tout en respectant la contrainte « jamais avec mon n+1 ». Pour ce faire, nous écrirons un algorithme en utilisant le principe de la **programmation dynamique**.

Fonctions utiles

Pour un employé i quelconque, on note :
- fun(i) la valeur de sa note de « fun » ;
- sub(i) la liste de ses subordonnés directs (ceux dont il est le « n+1 »).

Nous allons calculer deux valeurs :
- $\text{total}_{\text{yes}}(i)$ le score maximum de « fun » d'une soirée limitée à l'employé i et tous ses subordonnés directs et indirects, **lorsque i est présent à la soirée** ;
- $\text{total}_{\text{no}}(i)$ le score maximum de « fun » d'une soirée limitée à l'employé i et tous ses subordonnés directs et indirects, lorsque i **n'est pas présent** à la soirée.

Question 1

2 point(s)

Écrivez les formules de récurrence de $\text{total}_{\text{yes}}(i)$ et $\text{total}_{\text{no}}(i)$ en fonction des valeurs de $\text{total}_{\text{yes}}$ et total_{no} des subordonnés.

Question 2 3 point(s)

Écrivez un algorithme de programmation dynamique qui maximise la somme des notes « fun » des participants à la soirée. Vous pouvez écrire une ou plusieurs fonctions selon ce qui vous semble le plus judicieux.

Exercice 5 : Complexité de problème — 3 point(s)

On considère les deux problèmes suivants :

BI-PARTITION
Entrée : Un ensemble E de nombres.
Question : Existe-il un sous-ensemble $F \subseteq E$ tel que $\sum_{x \in F} x = \frac{1}{2} \sum_{x \in E} x$?

SUBSETSUM
Entrée : Un ensemble S de nombres et un nombre t
Question : Existe-il un sous-ensemble $T \subseteq S$ tel que $\sum_{x \in T} x = t$?

Question 1 — 1 point(s)

Montrez que SUBSETSUM est dans NP.

Question 2 — 2 point(s)

Sachant que BI-PARTITION est NP-complet, montrez que SUBSETSUM est lui aussi NP-complet.

Exercice 6 : Retour à l'hôpital **2 point(s)**

On s'intéresse à nouveau au problème de couverture d'un territoire par des hôpitaux que nous avons étudié pendant les deux séances de TP.

Rappels : Nous avons défini une fonction `combination_k` qui peut s'utiliser comme un itérateur dans une boucle pour écrire un algorithme de recherche exhaustive. À titre d'exemple, le code ci-dessous affiche successivement toutes les combinaisons de 10 villes parmi les villes candidates.

```
def test_combination(candidates):
    for c in combination_k(candidates,10):
        print(c) # prints a list of 10 cities
```

Nous avons aussi écrit une fonction `all_distances` qui construit un dictionnaire contenant toutes les distances entre deux villes du territoire. Les clefs du dictionnaire sont les couples de villes et les valeurs sont les distances entre les villes :

```
def test_distances(cities):
    dict_dist = all_distances(cities)
    print(dict_dist[('A','B')])
```

Le code ci-dessus affiche la distance entre les villes de noms A et B du territoire (si tant est qu'elles existent).

Définition : On appelle *diamètre* d'une liste de villes la plus grande distance entre deux villes de la liste.

Question 1 1.5 point(s)

Écrivez une fonction qui prend en argument un territoire, une liste de villes candidates et un nombre k et qui retourne la combinaison de k villes candidates qui a le plus petit diamètre (ou l'une de ces combinaisons s'il y en a plusieurs).

Question 2 0.5 point(s)

Quelle est la complexité de cet algorithme ? Justifiez votre réponse.

Exercice 7 : Bonus **1 point(s)**

On considère le problème suivant :

SATISFIABILITY
Entrée :
- Un ensemble l de variables booléennes ;
- Une formule logique $f(l)$ reliant les variables de l.

Question : Existe-il une affectation des valeurs de l telle que $f(l)$ soit vrai ?

Question 1 1 point(s)

Écrivez un algorithme polynomial pour résoudre ce problème.

— fin —

2019-02-02