Programmation dynamique

Consignes générales

- Le sujet est composé de deux problèmes complètement indépendants. Si vous repérez ce qui semble être une coquille, n'hésitez pas à la modifier en expliquant les initiatives que vous êtes amené à prendre.
- Les programmes doivent être systématiquement **précédés** d'une brève explication de leur fonctionnement (sauf éventuellement ceux de moins de 6 lignes). Inutile en revanche de surcharger vos codes à l'aide de commentaires inutiles, par exemple pour m'indiquer que la ligne qui commence par `for` consiste à démarrer une boucle.
- Vous êtes libres de définir autant de fonctions intermédiaires que vous le souhaitez pour répondre à une question. La longueur idéale d'un code se situe entre 4 et 10 lignes : en dessous, le programme ne sert sans doute à rien, au-dessus il est envisageable de le découper en plusieurs morceaux.

I. Planning de TD

Soit $n \in \mathbb{N}^*$. Un élève de PC* désire organiser son planning de passage en TD pour les n séances de l'année. Les séances sont numérotées par les entiers de 0 à $n-1$ (le premier TD est donc le TD numéro 0).

(**i**) A chaque séance, il peut choisir entre deux exercices : un exercice facile et un exercice difficile.

(**ii**) Pour $i < n-1$, s'il choisit l'exercice difficile à la i-ième séance, il doit impérativement traiter l'exercice facile à la $(i+1)$-ième séance pour se reposer.

(**iii**) Chaque exercice lui rapporte un certain nombre de points d'expérience. Pour tout $i \in [\![0; n-1]\!]$, on note f_i le nombre de points rapportés par l'exercice facile de la i-ième séance, et d_i celui rapporté par l'exercice difficile.

L'objectif de l'exercice est de l'aider à faire ses choix de manière à maximiser ses points d'expérience à l'issue des séances de TD. Dans tous les codes python du sujet, on dispose de deux listes `f` et `d` tels que `f[i]` (resp. `d[i]`) soit égale à f_i (resp. d_i).

1. Dans cette question uniquement, $n = 4$ et on considère les valeurs suivantes :

   ```
   f = [2;1;3;5]      et      d = [7;2;4;6]
   ```

   Donner un planning maximisant le total de points d'expérience en expliquant brièvement le raisonnement utilisé. A-t-on unicité d'un tel planning maximisant l'expérience ?

2. On représente en python un planning par une liste `p` de booléens de longueur n telle que `p[(i]` prend la valeur `True` si et seulement si le candidat prend l'épreuve difficile au i-ième TD.

   (a) Ecrire une fonction `score` qui prend en argument un planning `p` et les deux tableaux `f` et `d` et qui renvoie le nombre de points d'expérience récoltés à l'issue de ce planning.

   (b) Ecrire une fonction `verifie` qui prend en argument un planning `p` et renvoie `True` si et seulement si le planning `p` respecte la contrainte (**ii**).

3. Pour tout entier $n \in \mathbb{N}^*$, on note u_n le nombre de plannings possibles sur n TDs respectant la condition (**2**).

   (a) Donner les valeurs de u_1, u_2 et u_3.

   (b) Justifier que pour tout entier n, $\quad u_{n+2} = u_{n+1} + u_n$

   (c) Une solution au problème consiste à générer tous les plannings possibles puis à calculer pour chacun d'entre eux leur score afin de déterminer celui de score maximal. Que pensez-vous en pratique d'une telle méthode ? On argumentera soigneusement la réponse par des arguments de complexité, avec des ordres de grandeur cohérents.

4. On propose pour répondre à la question la méthode (heuristique) suivante :

$i \longleftarrow 0$
Tant que $i+1 < n$
  Si $d_{i+1} > f_{i+1} + d_i$
    Prendre l'exercice facile au TD i et l'exercice diffcile au TD $i+1$
    $i \longleftarrow i+2$
  Sinon
    Prendre l'exercice facile aux TD i
    $i \longleftarrow i+1$
Fin Tant que

   (a) Puisque i augmente de 1 ou 2 à chaque boucle, cet algorithme termine soit lorsque $i = n$, soit lorsque $i = n-1$. Dans ce deuxième cas, on n'a pas encore décidé quoi faire lors du dernier TD. Expliquer comment faire ce choix de manière cohérente.

   (b) A l'aide d'un exemple bien choisi, démontrer que l'algorithme ne fournit pas toujours un planning de score optimal.

Pour tout entier $j \in [\![0; n-1]\!]$, on note m_j le score maximal que l'on peut obtenir lors des j premières séances de TD.

5. Justifier soigneusement que pour tout $j \in [\![1; n-2]\!]$,

$$m_{j+1} = \max\{m_j + f_{j+1}, m_{j-1} + f_j + d_{j+1}\}$$

6. En déduire une fonction `score_max` qui prend en argument `f` et `d` et qui renvoie le score maximal qu'il est possible d'obtenir en préparant les n TDs. Evaluer proprement sa complexité.

7. Ecrire une fonction `panning_optimal` qui prend en argument `f` et `d` et qui renvoie un planning `p` qui réalise ce score maximal.

II. Plus long chemin dans un graphe ordonné

Soit $n \in \mathbb{N}^*$. Dans tout l'énoncé, on considère un graphe $G = (S, A)$ où $S = [\![0; n-1]\!]$ est l'ensemble des sommets et A l'ensemble des arêtes. Pour tout sommet $x \in S$, on note

$$a(x) = \{y \in S, \quad (y, x) \in A\} \qquad \text{et} \qquad s(x) = \{y \in S, \quad (x, y) \in A\}$$

appelés respectivement ensemble des antécédents et des successeurs du sommet x. Le graphe est supposé ordonné, ce qui signifie que

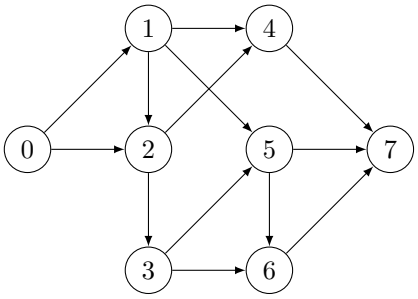
(**i**) Si (i, j) est une arête du graphe, nécessairement $i < j$.

En d'autres termes, aucune arête ne relie un sommet à un sommet d'indice inférieur.. On suppose pour finir que

(**ii**) A l'exception du sommet 0, tout sommet x de G admet au moins un arc entrant, c'est-à-dire que $a(x)$ est non vide.

(**iii**) A l'exception du sommet $n-1$, tout sommet x de G admet un arc sortant, c'est-à-dire que $s(x)$ est non vide.

Voici un exemple de tel graphe.



Sur cet exemple, on a donc notamment $a(5) = \{1, 3\}$ et $s(5) = \{6, 7\}$.

On rappelle qu'un chemin de x vers y est une suite de sommets $(s_0, \ldots, s_k)$ tels que $s_0 = x$, $s_k = y$ et pour tout $i < k$, $(s_i, s_{i+1}) \in A$. Un tel chemin est alors de longueur k (c'est le nombre d'arcs qui le constitue). L'objectif du problème est de concevoir un algorithme pour trouver le plus **long** chemin du sommet 0 vers le sommet $n-1$.

1. Donner un exemple de plus long chemin depuis 0 vers 7 dans l'exemple précédent.
2. Démontrer que pour tout sommet $s \in S$, il existe au moins un chemin depuis le sommet 0 jusqu'au sommet i.
   Indication : on pourra raisonner par récurrence forte sur i.
   On justifiera brièvement de même qu'il existe pour tout i un chemin de i vers $n-1$.
3. On considère l'algorithme glouton suivant pour résoudre le problème :

   $L \longleftarrow [0], i \longleftarrow 0$
   Tant que $i \neq n-1$
   Trouver j minimal tel que $(i, j) \in A$
   Ajouter j à L, $i \longleftarrow j$
   Fin Tant que
   Renvoyer L

   A l'aide d'un exemple bien choisi, démontrer que cet algorithme ne fournit pas toujours un plus long chemin du sommet 0 vers le sommet $n-1$.

Dans toute la suite, le graphe est implémenté en Python par listes d'adjacence. On rappelle qu'on représente alors le graphe par une liste `L` telle que pour tout i, `L[i]` est une liste qui contient l'ensemble des éléments de $s(i)$.

4. Donner la représentation par listes d'ajacences du graphe donné en exemple par l'énoncé.
5. Ecrire une fonction `est_ordonné` qui prend en argument une liste `L` représentant un graphe G quelconque et teste si la condition (**i**) est vérifiée.
6. Ecrire une fonction `listes_antécédents` qui prend en argument une liste L représentant un graphe G quelconque et renvoie une nouvelle liste `L2` telle que pour tout i, `L2[i]` est cette fois l'ensemble des éléments de $a(i)$ (dans un ordre arbitraire).

Pour tout entier $i \in [\![0; n-1]\!]$, on note m_i la longueur d'un plus long chemin depuis le sommet 0 vers le sommet i. Cette quantité est bien définie d'après le résultat de la question 2. Il est clair que $m_0 = 0$.

7. Soit $i \geq 1$. Démontrer soigneusement que $$m_i = 1 + \max_{j \in a(i)} m_j$$

8. En déduire une fonction `longueur_max` qui prend en argument une liste `L` représentant un graphe satisfaisant les conditions (**i**), (**ii**) et (**iii**) et renvoie la valeur de m_n. Evaluer sa complexité.

9. Modifier la fonction précédente pour obtenir une fonction `chemin_max` qui prend en argument `L` et renvoie la liste des sommets d'un plus long chemin depuis le sommet 0 vers le sommet $n-1$ dans G.