L3-INFO Algorithmique et Modélisation

UFR IM²AG

UNIVERSITÉ Grenoble Alpes

Annales d'examens et de quicks

Pour tous ces examens sont interdits : les documents, les ordinateurs, les téléphones (incluant smartphone, tablettes,... tout ce qui contient un dispositif électronique).
Seuls les dictionnaires papier pour les personnes de langue étrangère sont autorisés.
Il sera tenu compte de la qualité de la rédaction et de la clarté de la présentation (2 pts).

Table des matières

1 Découpage *(Examen du 25/04/2017)*

On dispose d'une tige de longueur L qu'on souhaite découper pour produire des tiges plus petites de différentes longueur ; il y a k types de telles tiges. Chaque tige de type i ($1 \leqslant i \leqslant k$) a une longueur l_i que l'on suppose entière. Le nombre de tiges de chaque type que l'on peut produire n'est pas limité mais on cherche à éviter les pertes.

Par exemple, si $k = 3$, $l_1 = 7$; $l_2 = 3$; $l_3 = 6$, pour $L = 20$, on pourra produire deux tiges de type 1 et une de type 3. Pour une tige de longueur $L = 11$, on ne peut éviter une perte, avec par exemple une pièce de type 1 et une de type 2.

Étant donné des longueurs $l_1, \cdots l_k$ le problème est de décider s'il existe une découpe sans perte d'une tige de longueur L, c'est à dire s'il existe k entiers $n_1, \cdots n_k$ tels que

$$L = \sum_{i=1}^{k} n_i l_i.$$

1. **Algorithme naïf**
   Démontrer que l'algorithme glouton consistant à découper la tige en prenant en priorité la plus grande longueur d'abord ne minimise pas la perte.
2. **Algorithme récursif**
   Exprimer ce problème sous la forme d'une équation de récurrence (à la fois sur k et sur L) et en déduire un algorithme récursif. Calculer son coût (donner un ordre de grandeur).
3. **Algorithme de décision**
   En s'inspirant de l'algorithme de *rendu de monnaie*, proposer un algorithme ayant une complexité de l'ordre de $k \times L$. Modifier cet algorithme pour qu'il fournisse également une découpe sans perte $n_1, \cdots, n_k$ lorsqu'elle existe.
4. **(Bonus) Algorithme minimisant la perte**
   Modifier cet algorithme pour qu'il fournisse maintenant une découpe $n_1, \cdots, n_k$ qui minimise la perte.

2 Anagrammes *(Examen du 25/04/2017)*

Une anagramme d'un mot est un autre mot ayant les mêmes lettres, mais dans un ordre différent. Par exemple `GUERISON` est une anagramme de `SOIGNEUR`, `ASPIRINE` de `PARISIEN` ou `MINISTRE` de `INTERIMS`.

L'objectifs de cet exercice est d'écrire un algorithme d'énumération de toutes les anagrammes d'un mot. Un mot sera représenté par un tableau de caractères `M[1...n]` de taille n.

1. Si le mot M possède n lettres différentes, donner le nombre total de configurations possibles des lettres pouvant représenter une anagramme. Donner l'ordre de grandeur de ce nombre pour $n = 8$.
2. Algorithme générique
   (a) Écrire un algorithme récursif qui énumère, c'est à dire `Visite` une et une seule fois, toutes les configurations possibles. (On utilisera un espace mémoire de l'ordre de grandeur de la taille du mot, la fonction `Visite` est supposée fournie et pourra par exemple tester l'appartenance de la configuration à un dictionnaire).
   (b) Justifier et illustrer cet algorithme en l'exécutant sur l'exemple `RIME`.
   (c) Faire la preuve de l'algorithme.

On dispose maintenant d'une fonction `PrefixeDico` qui indique si une configuration est préfixe d'un mot du dictionnaire

3. Adapter l'algorithme afin de n'explorer que les configurations dont le préfixe a été reconnu et faire sa preuve.

Les mots proposés peuvent maintenant comporter plusieurs fois la même lettre, par exemple `ETINCELLE` anagramme de `CLIENTELE` comporte plusieurs `L` et plusieurs `E`, cependant le mot `ETINCELLE` ne doit être `Visite` qu'une seule fois.

4. **(Bonus)** Comment adapter l'algorithme pour ne visiter qu'une seule fois chaque anagramme du mot ?

3 Tourner en rond *(Examen du 25/04/2017)*

On se donne un graphe orienté avec des arcs pondérés $\mathcal{G} = (\mathcal{X}, \mathcal{A}, \mathcal{P})$, $\mathcal{X}$ est l'ensemble des sommets, $\mathcal{A}$ l'ensemble des arcs et $\mathcal{P}$ les poids des arcs, $p(u, v)$ est le poids de l'arc entre le sommet u et le sommet v. L'algorithme de Dijkstra, vu en cours et en TD1/2, permet de construire l'ensemble des chemins de poids minimal issus d'un sommet s à tous les autres sommets accessibles à partir de s. [1]

Une des hypothèses pour que l'algorithme de Dijkstra soit correct est que les poids soient tous positifs.

1. Donner un exemple de graphe pondéré (avec certaines pondérations négatives) et d'exécution de l'algorithme de Dijkstra qui illustre la non correction de l'algorithme.

Lorsque les pondérations peuvent être négatives il n'existe pas forcément un plus court chemin.

2. Donner un exemple de graphe pondéré pour lequel il n'existe pas de chemin de poids minimal entre s et un autre sommet accessible à partir de s.

On suppose qu'il existe un chemin de poids minimal entre s à chacun des autres sommets du graphe (il n'y a pas de circuit de poids négatif, un tel circuit est dit *absorbant*), on note $d^{\min}(x)$ le poids minimal d'un chemin de s à x

3. Démontrer que les variables $d^{\min}(.)$ vérifient l'équation de point fixe

$$d^{\min}(v) = \min_{(u,v)\in\mathcal{A}} \left\{ d^{\min}(u) + p(u, v) \right\} \text{ pour tout } v \in \mathcal{X}\backslash\{s\} \text{ avec } d^{\min}(s) = 0. \tag{1}$$

Indication : on pourra d'abord démontrer que $d^{\min}(v) \leqslant d^{\min}(u) + p(u, v)$ pour tout arc (u, v), puis montrer qu'il existe un arc (w, v) tel que $d^{\min}(v) \leqslant d^{\min}(w) + p(w, v)$.

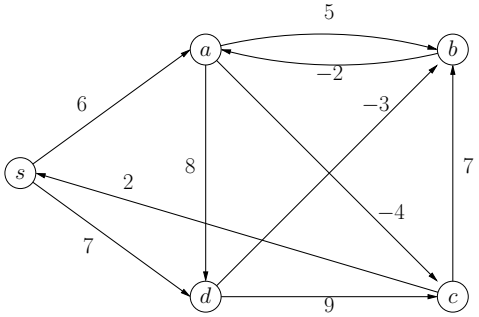
Considérons l'algorithme suivant :

Algorithme `CalculePoidsMin` ($\mathcal{G}$,s)
Données : Graphe pondéré $\mathcal{G} = (\mathcal{X}, \mathcal{A}, \mathcal{P})$,
$p(u, v)$ étant le poids de l'arc (u, v)
Résultat : Renvoie $d^{\min}(.)$

1 **foreach** $x \in \mathcal{X}\backslash\{s\}$ **do** $d(x) = +\infty$
2 $d(s) = 0$
3 **while** *il existe un arc* (u, v) *tel que* $d(v) > d(u) + p(u, v)$ ***do***
4 $\quad d(v) = \min\{d(v), d(u) + p(u, v)\}$
5 `Retourne` d

Algorithme 1 : Algorithme de calcul de $d^{\min}(.)$

4. Exécuter l'algorithme sur le graphe ci-dessous, on notera la séquence des arcs choisis et la suite des vecteurs d obtenus à la fin de chaque itération (ligne 4).



5. Si l'algorithme termine, démontrer que le vecteur d retourné est le point fixe de l'équation ci-dessus.
6. Démontrer que l'algorithme se termine. *Indication : trouver un variant de l'itération et utiliser le fait que le nombre de chemins "potentiellement de poids minimal" est fini.*

1. On rappelle que le poids d'un chemin est la somme des poids des arcs qui le composent.

7. L'algorithme permet de calculer le poids minimal des chemins d'origine s. Modifier l'algorithme afin d'avoir le routage, c'est à dire, pour chaque sommet x, un chemin de poids minimal s à x.

*On remplace les lignes 3 et 4 (boucle **While**) de l'algorithme par*

```
6 repeat |X| − 1 fois
7 |   foreach (u, v) ∈ A do
8 |   |   d(v) = min {d(v), d(u) + p(u, v)}
```

8. (**Bonus**) S'il n'y a pas de circuit absorbant, montrer que le coût de cet algorithme peut être réduit à $|\mathcal{X}|.|\mathcal{A}|$.
9. Que se passe-t-il si le graphe possède un circuit absorbant ? Modifier l'algorithme pour qu'il détecte s'il existe un circuit absorbant ou renvoie le $d^{\min}(.)$.
10. Comparer cet algorithme avec l'algorithme de Dijkstra (on se donnera au préalable des critères de comparaison).

4 Vocabulaire *(Quick du 07/03/2017)*

On souhaite analyser la richesse de vocabulaire d'un auteur littéraire. On dispose pour cela d'un texte et on souhaite compter le nombre de mots différents utilisés par l'auteur.

On suppose que le texte est segmenté en mots et que l'on accède au $k^{\text{ième}}$ mot par `texte[k]`. On notera n la taille du texte en nombre de mots.

1. Proposer (et justifier) une structure de donnée appropriée pour ce type de problème.
2. Proposer un algorithme de coût moyen $\mathcal{O}(n)$.

5 Jeu de Dames *(Quick du 07/03/2017)*

Dans le jeu de dames, une case noire d'un damier $n \times n$ porte soit un pion ou une dame blanc, soit un pion ou une dame noire, soit ne porte pas de pion. L'objectif est d'écrire un algorithme qui énumère toutes les configurations possibles de jeu (sans prise en compte d'autres contraintes que le nombre maximum de pions/dame blancs ou noir (au plus 20 de chaque dans un jeu de dames classique).

On représente le damier par un ensemble de cases, numérotées de 1 à N et on code l'état de la case par l'une des 3 valeurs : 1 =blanc, -1=noir, 0 s'il n'y a pas de pion ou de dame (pour l'instant on seule compte la couleur). On suppose qu'il y a au maximum K pions/dames blancs et K pions/dames noirs.

1. Donner, en fonction de K et de N un majorant et un minorant du nombre de configurations possibles.
2. Proposer un algorithme qui énumère toutes les configurations possibles du jeu.
3. Modifier votre algorithme pour prendre en compte le type d'occupation d'une case (pion ou dame).

6 Tri par dénombrement *(Quick du 07/03/2017)*

extrait de l'ouvrage de Cormen et al, Algorithmique (Dunod 2012)

Le ***tri par dénombrement*** suppose que chacun des n éléments de l'entrée est un entier de l'intervalle 0 à k, k étant un certain nombre entier. Lorsque $k = O(n)$, le tri s'exécute en un temps $\Theta(n)$.

Le principe du tri par dénombrement est de déterminer, pour chaque élément x de l'entrée, le nombre d'éléments inférieurs à x. Cette information peut servir à placer l'élément x directement à sa position dans le tableau de sortie. Par exemple, s'il existe 17 éléments inférieurs à x, alors x se trouvera en sortie à la position 18. Ce schéma doit être légèrement modifié pour gérer la situation dans laquelle plusieurs éléments ont la même valeur, puisqu'on ne veut pas tous les placer à la même position.

Dans le code du tri par dénombrement, on suppose que l'entrée est un tableau $A[1\,..\,n]$ et donc que $longueur[A] = n$. Nous avons besoin de deux autres tableaux : le tableau $B[1\,..\,n]$ contient la sortie triée et le tableau $C[0\,..\,k]$ sert d'espace de stockage temporaire.

```
TRI-DÉNOMBREMENT(A, B, k)
 1  pour i ← 0 à k
 2      faire C[i] ← 0
 3  pour j ← 1 à longueur[A]
 4      faire C[A[j]] ← C[A[j]] + 1
 5  ▷ C[i] contient maintenant le nombre d'éléments égaux à i.
 6  pour i ← 1 à k
 7      faire C[i] ← C[i] + C[i − 1]
 8  ▷ C[i] contient maintenant le nombre d'éléments inférieurs ou égaux à i.
 9  pour j ← longueur[A] jusqu'à 1
10      faire B[C[A[j]]] ← A[j]
11           C[A[j]] ← C[A[j]] − 1
```

1. Exécuter cet algorithme sur le tableau A suivant :

   A :

2	5	3	0	2	3	0	3

   On précisera en particulier la valeur du tableau C à la ligne 6 et à la ligne 9. On illustrera également le remplissage du tableau B durant l'exécution de la boucle de la ligne 10.
2. Calculer le coût de cet algorithme.
3. Analyser ce coût et comparer avec les différents algorithmes de tri déjà rencontrés dans votre formation. Qu'en pensez-vous ?

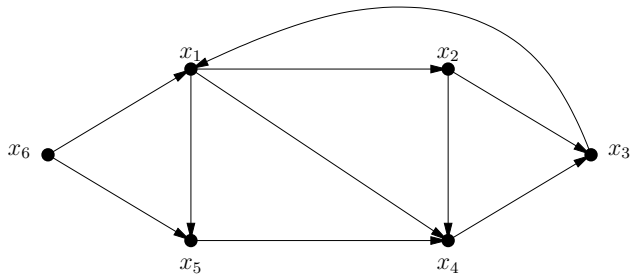
7 Cheminements *(Examen du 03/05/2016)*

Soit un graphe orienté $\mathcal{G} = (\mathcal{X}, \mathcal{A})$ avec $\mathcal{X} = \{x_1, \cdots, x_n\}$ l'ensemble des sommets et $\mathcal{A}$ l'ensemble des arêtes. n est le nombre de sommets du graphe et A la matrice d'adjacence associée (matrice booléenne).

On considère l'algorithme suivant :

Algorithme Mystère(A)
Données : A matrice d'adjacence (booléenne) d'un graphe $\mathcal{G}$ de taille $n \geqslant 1$
Résultat : Une matrice B de booléens

```
1  b_{1,1} = 1
2  for k = 2 to n do
       // Point d'observation de B
3      b_{k,k} = 1
4      for i = 1 to k − 1 do
5          b_{i,k} = a_{i,k}
6          for j = 1 to k − 1 do  b_{i,k} = b_{i,k} ou (b_{i,j} et a_{j,k})
       // Commentaire 1 :
7      for j = 1 to k − 1 do
8          b_{k,j} = a_{k,j}
9          for i = 1 to k − 1 do  b_{k,j} = b_{k,j} ou (a_{k,i} et b_{i,j})
       // Commentaire 2 :
10     for i = 1 to k − 1 do
11         for j = 1 to k − 1 do
12             b_{i,j} = b_{i,j} ou (b_{i,k} et b_{k,j})
       // Commentaire 3 :
```

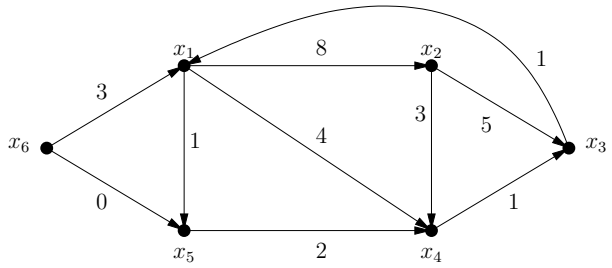


1. Évaluer le coût de cet algorithme en fonction de la taille du graphe, on précisera les opérations considérées pour le calcul du coût.
2. Exécuter l'algorithme sur l'exemple ci-dessus. On écrira la matrice B à chaque itération de la boucle externe au "point d'observation de B".
3. Analyse de l'algorithme
   (a) Que fait l'algorithme mystère ? Expliquer en français le principe et faire un dessin explicatif.
   (b) Écrire les commentaires 1, 2 et 3.
   (c) Donner un invariant de la boucle externe ligne 2.

On suppose maintenant que les arcs du graphe sont valués, avec des valeurs positives.

4. Modifier l'algorithme mystère afin de fournir une matrice D de valeur minimale de chemin entre les sommets du graphe. [2]
5. Avec les valuations suivantes sur le graphe d'exemple, exécuter votre algorithme de calcul de D. (écrire la matrice D au point d'observation pour chaque itération).

2. On rappelle que la valeur associée à un chemin est la somme des valeurs de ses arcs.



6. Modifier l'algorithme proposé afin de fournir simultanément une matrice de routage R ayant la propriété suivante $r_{i,j}$ est un sommet voisin de i sur un chemin de valeur minimale allant de i à j.
7. On suppose que les valuations peuvent être positives ou négatives. Quelle difficulté peut survenir ? Comment modifier votre algorithme pour pallier cette difficulté ?
8. Quels sont les avantages et les inconvénients de l'algorithme proposé par rapport aux algorithmes vus en cours tels que l'algorithme de Dijkstra ou l'algorithme de Floyd-Warshall ?

8 Mise en boite *(Examen du 03/05/2016)*

Soit $\mathcal{O}$ un ensemble de n objets. À chaque objet o_i de $\mathcal{O}$ on associe une taille $t(o_i)$. On souhaite ranger les objets dans une boite de taille T.

Cependant la boite est trop petite pour contenir tous les objets, on cherche donc à mettre un maximum d'objets dans la boite. Annabelle propose l'algorithme suivant :

Algorithme `Remplir` (O)
Données : Un ensemble $O = \{o_1, \cdots, o_n\}$ de n objets avec leur taille
Une taille de boite T
Résultat : Un ensemble maximal d'éléments entrant dans la boite

```
1 if O est non vide
2     i = argmin{t(o_i)}                    // o_i est un objet de taille minimale
3     if t(o_i) ⩽ T
4         o_i est dans la solution
5         Remplir (O\{o_i}, T − t(o_i))
```

Algorithme 2 : Algorithme de remplissage

1. Démontrer que l'algorithme glouton proposé par Annabelle produit bien une solution maximisant le nombre d'objets dans la boite.

Bertrand demande maintenant de trouver un sous-ensemble d'objets qui remplissent au mieux la boite (il n'y a pas d'objet qui dépasse), c'est à dire que la somme des tailles des objets dans la boite soit la plus proche possible de T.

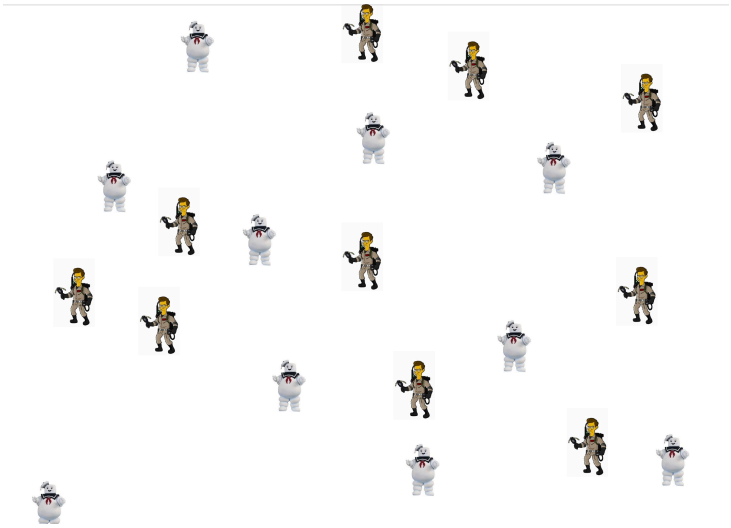
2. Montrer que l'algorithme proposé par Annabelle de donne pas une solution optimale pour la boite la mieux remplie.
3. En vous inspirant de l'algorithme d'énumération des parties, proposer un algorithme qui fournit une solution optimale. On justifiera les choix associés aux branchements.
4. Faire la preuve de votre algorithme.

9 Ghostbusters *(Examen du 03/05/2016)*

> *There's something very important I forgot to tell you ! Don't cross the streams. . . It would be bad. . .Try to imagine all life as you know it stopping instantaneously and every molecule in your body exploding at the speed of light.* Dr. Egon Spengler

Les chasseurs de fantômes Ghostbusters utilisent des rayons pour neutraliser les fantômes. Comme l'indique la citation ci-dessus, il est primordial que ces rayons ne se croisent jamais. Imaginons dix fantômes chassés par dix

Ghostbusters. Ces derniers peuvent-ils se répartir les fantômes afin que chacun tire sur un fantôme sans qu'aucun rayon ne se croise.



On suppose que les positions des ghostbusters et des fantômes sont quelconques, c'est à dire que 3 d'entre eux ne sont jamais alignés. On dispose d'une fonction `coté` indiquant si un point est à gauche ou à droite d'une droite orientée.

1. Proposer un algorithme de division du problème en 2 sous-problèmes. On pourra utiliser les idées de l'algorithme de balayage de Graham, utilisé pour le calcul d'enveloppe convexe, avec le choix adéquat d'un chasseur et d'un fantôme cible.
2. Écrire un algorithme qui donne une association admissible entre les chasseurs et les fantômes.
3. Donner la preuve et la complexité de l'algorithme. On donnera des exemples de pire et meilleurs cas.

10 Calcul du pic *(Quick du 02/03/2016)*

On dispose d'un tableau T de taille n d'éléments comparables ayant un pic, c'est à dire qu'il existe $p \in \{1, \dots, n\}$ tel que

$$T[1] < T[2] < \cdots < T[p] > T[p+1] > \cdots > T[n].$$

1. Proposer un algorithme naïf pour calculer la position du pic et calculer sa complexité.
2. Proposer un algorithme de type diviser pour régner pour calculer la position du pic et calculer sa complexité.

11 Opérations sur les ensembles *(Quick du 02/03/2016)*

On se donne deux tableaux T et U de tailles n et m, on note $N = \max\{m, n\}$. On veut produire un tableau V contenant tous les éléments apparaissant dans T mais pas dans U.

1. Proposer un algorithme de coût $\mathcal{O}(N^2)$ pour effectuer cette opération.
2. On peut améliorer l'opération précédente en commençant par trier les tableaux, calculer la complexité d'une telle solution ?
3. Peut-on calculer le tableau V en coût $\mathcal{O}(N)$? Si oui, écrire l'algorithme.

12 Élément de rang k *(Quick du 02/03/2016)*

On se donne un tableau T comprenant n éléments distincts. On appelle élément de rang k du tableau T, le $k$ième plus grand élément du tableau T (l'élément de rang 1 est donc le plus petit élément du tableau tandis que l'élément de rang n est le plus grand élément du tableau).

1. Écrire un algorithme de complexité $\mathcal{O}(n)$ qui calcule les éléments de rang 1 et de rang n.
2. Écrire un algorithme naïf permettant de calculer l'élément de rang k, calculer la complexité de votre algorithme et, si ce n'est pas le cas, modifier votre algorithme pour obtenir une complexité $\mathcal{O}(n \log n)$ dans le pire cas.

On rappelle l'algorithme *quicksort* randomisé :
- Choisir un élément au hasard (uniformément) parmi $T[1] \dots T[n]$ comme pivot
- Partitionner les éléments en plaçant les éléments plus petit que le pivot en début du tableau et les éléments plus grand que le pivot en fin de tableau.
- Appeler récursivement *quicksort* sur les deux sous-tableaux.

3. En s'inspirant de l'algorithme de *quicksort*, écrire un algorithme `QuickSelect`(T,k) qui calcule l'élément de rang k (*indication :* après avoir partitionné le tableau en deux, il est facile de tester si l'élément de rang k est plus petit ou plus grand que le pivot.)

Soit $C(n,k)$ le nombre moyen de comparaisons qu'effectue `Quickselect` pour trouver l'élément de rang k d'un tableau de taille n

4. Montrer que (il est fortement suggéré de faire des dessins)

$$C(n,k) = (n-1) + \frac{1}{n}\sum_{i=1}^{k-1} C(n-i, k-i) + \frac{1}{n}\sum_{i=k+1}^{n} C(i,k)$$

On notera $M(n)$ et on admettra (question bonus) que

$$M(n) \leq \frac{2}{n}\sum_{p=n/2}^{n-1} M(p) + \mathcal{O}(n).$$

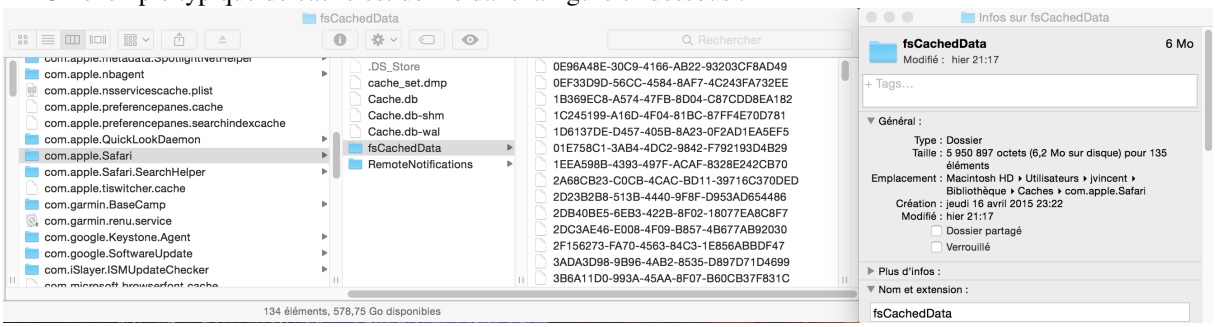
5. En déduire que $M(n) = \mathcal{O}(n)$.

13 Navigation *(Examen du 06/05/2015)*

Lors de l'observation du comportement de surfers sur le Web on constate que les pages visitées sont visitées plusieurs fois, qu'un même objet peut apparaitre sur plusieurs pages visitées (logos, images, ...). Pour améliorer les performances d'un navigateur internet en évitant de télécharger plusieurs fois un même objet, on implémente un mécanisme de cache qui stocke localement une partie du contenu des pages visitées.

Afin de retrouver rapidement les objets dans le cache, on nomme les objets à partir de leur url ce qui permet un accès direct à la copie locale. Ce nommage est effectué par une fonction de hachage qui prend en argument l'url et qui renvoie une chaîne de caractères de longueur fixe.

Un exemple typique de cache est donné dans la figure ci-dessous :



Pour simplifier l'analyse on suppose que seuls les 8 premiers caractères sont générés par une fonction de hachage (chiffres et lettres A,B,C,D,E,F).

1. En supposant que la taille du cache est de 6Mo et que la taille moyenne d'un objet caché est de 50Ko calculer la probabilité d'avoir une collision, c'est à dire la probabilité que 2 objets aient la même valeur de hachage. On donnera juste l'ordre de grandeur de cette probabilité. Quelle serait la taille du cache, en gardant des objets de même taille moyenne, pour que cette probabilité soit de l'ordre de 1% ?

2. Commenter le résultat. Est-il pertinent de tester si l'on a une collision ? Comment s'améliore cette probabilité si on concatène au résultat de la première fonction de hachage, le résultat d'une deuxième fonction de hachage générant une chaîne de taille 12 (fin du nom de fichier dans l'exemple ci-dessus) ?

14 Compter les chemins *(Examen du 06/05/2015)*

L'objectif de cet exercice est de concevoir un algorithme qui compte le nombre de chemins de longueur l allant de i à j dans un graphe orienté $\mathcal{G} = (X, A)$. Les chemins peuvent passer plusieurs fois par le même sommet ou le même arc

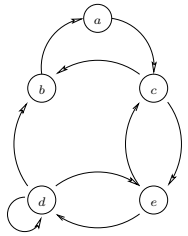
$\mathcal{G} = (X, A)$ avec $|X| = 5$ et $|A| = 9$

$X = \{a, b, c, d, e\}$

$A = \{(a, c), (b, a), (c, b), (c, e), (d, b), (d, d), (d, e), (e, c), (e, d)\}$

$C = (a, c, e, d, e, d, d, b)$

C est un chemin de longueur 7 de a à b



1. Donner la matrice d'adjacence du graphe ci-dessus et le nombre de chemins de a à b de longueur l pour $l = 0, 1, 2, 3, 4, 5$. On note $n_{i,j}^{(l)}$ le nombre de chemins de i à j de longueur l et $N^{(l)}$ la matrice associée
2. Écrire les équations de récurrences exprimant les coefficients $n_{i,j}^{(l)}$ en fonction des coefficients $n_{i,k}^{(l-1)}$. Justifier en français ces équations et faire un dessin explicatif.
3. En vous inspirant des équations de récurrence écrire un algorithme qui calcule la matrice $N^{(l)}$ en $|X|^3.l$ opérations. Expliquer une méthode pour réduire le coût à moins de $C.|X|^{\log_2 7}.2 \log l$ opérations avec C un facteur constant.

15 Gros sous-tableau *(Examen du 06/05/2015)*

On se donne un tableau T de taille n (indicé de 1 à n), contenant des nombres (positifs ou négatifs). On cherche à trouver le sous-tableau contigu $T[i : j]$ de somme maximale. Autrement dit, on cherche les indices i et j qui maximisent $\sum_{k=i}^{j} T[k]$.

1. Calculer (à la main), une solution pour les tableaux suivants (à chaque fois, donner les indices d'un sous-tableau de somme maximale et la somme).

   $T_1 = [1, 4, -2, 3, 3, -4, -2, -2, 2, -1]$ $\quad T_2 = [1, -2, 3, 3, 1, -5, -1, -3, 8, -5]$
2. Évaluer le coût d'un algorithme naïf calculant les sommes de tous les sous-tableaux.
3. Proposer un algorithme en $\mathcal{O}(n^2)$ pour résoudre le problème (indication : on pourra se servir de programmation dynamique pour calculer toutes les valeurs de $\sum_{k=i}^{j} T[k]$).
4. Pour résoudre ce problème, on utilise une approche diviser pour régner. On découpe le tableau T en deux sous-tableaux T_L et T_H de taille moitié : T_L contient les éléments de 1 à $\frac{n}{2}$ et T contient les éléments de $\frac{n}{2}$ à n, on considèrera dans un premier temps que n est une puissance de 2. Le sous-tableau de somme maximale est un des trois tableaux suivants :
   T_L^{max} le sous-tableau de somme maximale de T_L ;
   T_H^{max} le sous-tableau de somme maximale de T_H ;
   T_{LH} la concaténation du sous-tableau de somme maximale de T_L terminant à $\frac{n}{2}$ et du sous-tableau de somme maximale de T_H commençant à $\frac{n}{2} + 1$.

   Les sous-tableaux T_L et T_H sont calculés en appelant la fonction récursivement.
   (a) Faire un dessin explicatif de l'algorithme.
   (b) Montrer que l'on peut calculer le tableau T_{LH} en temps $\mathcal{O}(n)$.
   (c) Soit $C(n)$ la complexité de l'algorithme sur un tableau de taille n. Donner une formule de récurrence pour $C(n)$ et donner l'ordre de grandeur de $C(n)$.
   (d) Comment adapter l'algorithme lorsque n n'est pas une puissance de 2 ?
   (e) Écrire l'algorithme récursif.

5. Bart prétend que l'on peut résoudre le problème en utilisant l'algorithme suivant :

```
Entrées : Un tableau T de taille n
Sorties : La somme des valeurs du sous-tableau de somme maximale
somme_courante ← 0;
meilleure_somme ← 0;
pour k = 0 à n − 1 faire
    somme_courante ← somme_courante + T[k];
    si somme_courante > meilleure_somme alors
        meilleure_somme ← somme_courante;
    sinon
        si somme_courante ≤ 0 alors
            somme_courante ← 0
Retourner meilleure_somme
```

   (a) Quelle est la complexité de cet algorithme ?

   (b) Cet algorithme rend-il la bonne somme ? Justifier votre réponse, c'est-à-dire : si oui, démontrer que l'algorithme est correct, si non donner un contre-exemple.

   (c) Si l'algorithme donne la somme maximale, proposer une modification de l'algorithme pour qu'il retourne également les indices du sous-tableau de somme maximale.

6. Synthèse : Parmi les quatre algorithmes proposés, lequel recommander, et pourquoi ?

16 Complexité diviser pour régner *(Quick du 07/04/2015)*

Pour un problème donné, un algorithme naïf a une complexité en $O(n^3)$. On a aussi trouvé deux solutions de type diviser pour régner :

Div1 Découper le problème de taille n en deux sous-problèmes de taille $n/2$ et les recombiner en temps $O(n^2)$.

Div2 Découper le problème en quatre sous-problèmes de taille $n/3$ et les recombiner en temps $O(n)$.

1. Quelle est la complexité de Div1 ?
2. Quelle est la complexité de Div2 ?
3. Quelle solution choisir : naïf, div1 ou div2 ?

17 Programmation dynamique *(Quick du 07/04/2015)*

On se donne une matrice C de taille $n \times m$ (n et m sont deux entiers supérieurs ou égaux à 1). On chercher à calculer la quantité $D_{n,m}$, définie par récurrence par la formule suivante :

$$D_{i,j} = \begin{cases} i \text{ si } j = 0 \\ j \text{ si } i = 0 \\ \min(D_{i-1,j} + 1, D_{i,j-1} + 1, D_{i-1,j-1} + C_{ij}) \text{ sinon} \end{cases}$$

1. Écrire un algorithme récursif qui prend en entrée une matrice C de taille $n \times m$ et calcule $D_{n,m}$. Quelle est sa complexité ?
2. La relation de récurrence précédente se prête particulièrement bien à l'expression d'une implémentation par programmation dynamique. Écrire l'algorithme correspondant à cette solution. Quelle est la complexité de votre algorithme ?

18 Plus courts chemins *(Quick du 07/04/2015)*

On se donne un graphe orienté (S, A) ayant $|S| = n$ sommets et $|A| = m$ arcs. Soit $o \in S$ un sommet de ce graphe. On cherche à calculer l'ensemble des sommets pour lesquels il existe un chemin partant de o. Pour cela, on utilise la propriété suivante :

(P) il existe un chemin de o à j de longueur inférieure ou égale à k si :

- il existe un chemin de o à j de longueur inférieure ou égale à $k - 1$;

ou

- il existe un arc $(i, j) \in A$ et un chemin de o à i de longueur inférieure ou égale à $k - 1$.

On note $C_{j,k}$ une variable qui vaut *vrai* s'il existe un chemin de o à j de longueur inférieure ou égale à k et qui vaut faux sinon.

1. Formaliser une relation de récurrence sur $C_{j,k}$ utilisant la propriété (P).
2. La relation de récurrence précédente se prête particulièrement bien à l'expression d'une implémentation par programmation dynamique. Écrire l'algorithme correspondant à cette solution. L'algorithme prendra en entrée un sommet o et l'ensemble des arcs A et rendra en sortie un vecteur C tel que $C_i = vrai$ s'il existe un chemin entre o et i et $C_i = faux$ sinon.
3. Calculer la complexité de votre algorithme en fonction de n et m.
4. On associe à chaque arc $(i, j) \in A$ un poids $P_{ij} \geq 0$ et on cherche à calculer un chemin de poids minimal entre deux sommets o et d. Écrire un algorithme utilisant la propriété (P) qui affiche un chemin de poids minimal de o à d.

19 Random-SAT *(Quick du 03/04/2015)*

On se donne une expression booléenne sous forme normale conjonctive $\Phi(x_1, \cdots, x_n)$ portant sur n variables notées $\{x_1, x_2, \cdots x_n\}$, ayant m clauses, chaque clause ayant k littéraux. C'est une instance du problème k-SAT.

$$\Phi(x_1, \cdots, x_n) = \bigwedge_{i=1}^{m} (x_{i_1} \vee x_{i_2} \cdots \vee x_{i_k});$$

avec x_{i_j} étant l'une des variables x_l ou sa négation. On suppose, sans perte de généralité qu'une clause ne contient pas deux fois la même variable.

Pour résoudre ce problème, on génère aléatoirement et uniformément les valeurs des n variables et on évalue les m clauses. On suppose que la génération des valeurs des variables x_i se fait de manière indépendantes et uniforme : probabilité $\frac{1}{2}$ d'avoir la valeur 1 (VRAI) et $\frac{1}{2}$ d'avoir la valeur 0 (FAUX).

Soit Z_i la variable aléatoire modélisant la valeur de la clause i et soit N la variable aléatoire modélisant le nombre de clauses vérifiées, on a :

$$N = Z_1 + Z_2 + \cdots + Z_m.$$

1. Calculer la probabilité que la clause i ne soit pas vérifiée ($\mathbb{P}(Z_i = 0)$), en déduire la probabilité que la clause i soit vérifiée ($\mathbb{P}(Z_i = 1)$).
2. Calculer $\overline{m}$ le nombre moyen de clauses vérifiées.
3. En déduire que si $\overline{m} > m - 1$ alors la formule Φ est satisfiable.
   Indication : remarquer que $\overline{m} \leqslant m$
4. Sous la condition ci-dessus $\overline{m} > m-1$, proposer un algorithme randomisé calculant une solution à $\Phi(x_1, \cdots, x_n) = 1$.
5. *(plus difficile)* Soit p la probabilité de générer une solution. Montrer que

   $$p.m + (1 - p)(m - 1) \geqslant \overline{m}.$$

   En déduire un minorant de p. Donner un majorant de la moyenne du nombre de tirages nécessaires pour trouver une solution.
6. Que pensez-vous de cette approche ?

20 SUM *(Quick du 03/04/2015)*

Soit $\mathcal{E}$ un ensemble de n éléments. À chaque élément i de $\mathcal{E}$ on associe une valeur entière v_i. Pour éviter les cas particuliers on supposera les v_i distincts.

On recherche dans un premier temps tous les couples (i, j) avec $i \neq j$ tels que

$v_i + v_j = S$ pour un S donné ;

ou *pas de couple* s'il n'en existe pas.

1. Écrire un algorithme naïf qui résout le problème en $\mathcal{O}(n^2)$ opérations.
2. Écrire un algorithme qui résout le problème en $\mathcal{O}(n \log n)$ opérations.
3. En utilisant une structure de donnée adaptée, écrire un algorithme qui résout le problème en $\mathcal{O}(n)$ opérations.

On recherche maintenant tous les sous-ensembles de $\mathcal{E}$ tels que la somme des valeurs des éléments du sous-ensemble soit égale à S, sans contrainte sur la taille des sous-ensembles. Un peu d'étude bibliographique indique que ce problème est dans $\mathcal{NP}$.

Cas où toutes les valeurs $\{v_i\}$ sont positives

4. En vous inspirant de l'algorithme d'énumération des parties proposer un algorithme qui visite tous les sous-ensembles de somme S. On précisera avec soin les conditions sous lesquelles on effectuera les appels récursifs.
5. Écrire la preuve de votre algorithme.

Cas où les valeurs v_i peuvent être positives ou négatives

6. Adapter l'algorithme précédent en changeant les conditions d'appels récursifs.