

## TD5 : Programmation dynamique

**Exercice 1.***Coefficients binomiaux*

Pour  $k \leq n$  donnés, on veut calculer le coefficient binomial  $\binom{n}{k} = \frac{n!}{k!(n-k)!}$  (correspondant au nombre de choix de  $k$  éléments parmi  $n$ ). On rappelle pour cela les formules de récurrence suivantes :

- pour tout entier  $n$ , on a  $\binom{n}{0} = \binom{n}{n} = 1$  et
- pour  $0 < k < n$ , on a  $\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}$ .

1. Proposer un algorithme récursif `COEF-BIN`( $n, k$ ) pour le calcul de  $\binom{n}{k}$ . Borner le nombre total d'appels récursifs pour calculer  $\binom{n}{k}$ .
2. On souhaite s'inspirer d'une procédure de type programmation dynamique en stockant pour chaque couple  $(i, j)$  avec  $i \leq n$  et  $j \leq k$  la valeur de  $\binom{i}{j}$ . Écrire l'algorithme correspondant et calculer sa complexité.

**Exercice 2.***Sim city*

Au cours d'un jeu, il faut choisir un emplacement pour construire une ville. Le plateau de jeu est formé d'une grille carrée divisée en cellules et une ville occupe un ensemble de cellules libres formant un carré. Le but est de construire la plus grande ville possible et on souhaite mettre au point un algorithme pour décider de l'emplacement optimal. Le problème se modélise ainsi comme suit.

**Entrée :** une matrice  $A$  de taille  $n \times n$  dont les entrées sont des 1 (occupée) et des 0 (libre).

**Sortie 1 :** la taille de la plus grande sous-matrice carrée de  $A$  ne contenant que des 0.

**Sortie 2 :** l'emplacement de cette sous-matrice.

On note  $T_A$  la taille de la plus grande sous-matrice carrée ne contenant que des 0. Pour  $0 \leq i \leq n-1$  et  $0 \leq j \leq n-1$ , on note aussi  $t_A(i, j)$  la taille  $k$  de la plus grande sous-matrice carrée  $k \times k$  de  $A$  ne contenant que des 0 et dont le coin inférieur droit est la cellule  $A_{[i,j]}$  de  $A$ . On dit qu'un tel carré est *basé en*  $A_{[i,j]}$ .

$$A = \begin{array}{|c|c|c|c|c|} \hline 1 & 0 & 1 & 1 & 0 \\ \hline 0 & 0 & 0 & 1 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 \\ \hline 1 & 1 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 \\ \hline \end{array}$$

1. Sur l'exemple proposé, que vaut  $t_A(0, 0)$ ,  $t_A(0, 4)$ ,  $t_A(2, 2)$ ? Et que vaut  $T_A$ ?
2. On cherche à établir un premier algorithme pour le calcul de  $T_A$ , pas forcément optimal en complexité.
  - (a) Écrire un algorithme `EXTENSIBLE`( $A, i, j, k$ ) qui prend en entrée  $i, j$  et  $k$  tels que  $t_A(i, j) \geq k$  et teste si  $t_A(i, j) \geq k + 1$ . En entrée de l'algorithme, on suppose (sans le vérifier) que  $t_A(i, j) \geq k$ . Évaluer sa complexité.
  - (b) En déduire un algorithme `CARRÉ`( $A, i, j$ ) qui étant donné  $i$  et  $j$  calcule la taille du plus grand carré de 0 basé en  $A_{[i,j]}$ . Évaluer sa complexité.
  - (c) En déduire un algorithme pour calculer  $T_A$ . Quel est la complexité de votre algorithme?
3. On va fournir maintenant un algorithme plus rapide, basé sur de la programmation dynamique. On donne la relation de récurrence suivante. Pour  $i \geq 1$  et  $j \geq 1$ ,  $t_A(i, j) = 0$  si  $A_{[i,j]} = 1$  et  $t_A(i, j) = 1 + \min\{t_A(i, j-1), t_A(i-1, j), t_A(i-1, j-1)\}$  sinon.
  - (a) Donner les cas de base  $t_A(0, j)$  et  $t_A(i, 0)$  et exprimer  $T_A$  en fonction des  $t_A(i, j)$ .
  - (b) Sur l'exemple proposé, calculer  $T_A$  à l'aide de la formule proposée.
  - (c) Prouver la relation de récurrence donnée.
  - (d) Écrire un algorithme implantant la formule ci-dessus (sans preuve) et établir sa complexité.
  - (e) Comment peut-on modifier l'algorithme pour renvoyer également la position du plus grand carré libre?

**Exercice 3.***Stratégie étudiante*

Un étudiant veut établir son programme de travail du semestre en fonction des points qu'il espère gagner sur sa moyenne. Chaque semaine  $i$  du semestre, il doit décider s'il fait une *semaine tranquille*, à l'issue de laquelle il

pense gagner  $t(i)$  points sur sa moyenne, ou une *semaine harassante*, qui lui permettra, d'après lui, de gagner  $h(i)$  points. Son énergie étant limitée, avant chaque semaine harassante, il doit se reposer la semaine précédente, ce qui lui rapporte 0 point. Le but est d'obtenir un planning qui maximise le nombre de points espérés.

Par exemple sur les semaines suivantes, sa stratégie optimale consiste à prendre une semaine de repos, puis faire une semaine harassante en Semaine 2 et enfin deux semaines tranquilles, ce qui lui rapportera 2,2 points.

	Semaine 1	Semaine 2	Semaine 3	Semaine 4
$t$	0,3	0,5	0,4	0,6
$h$	0,5	1,2	1	0,8

1. Montrer que l'algorithme suivant n'est pas optimal ( $n$  correspond au nombre de semaines dans le semestre).

```

1  $i \leftarrow 1$ 
2 tant que  $i \leq n$  :
3   si  $i \leq n-1$  et  $h(i+1) > t(i) + t(i+1)$  :
4     Se reposer semaine  $i$  et faire une semaine harassante semaine  $i+1$ 
5      $i \leftarrow i+2$ ;
6   sinon :
7     Faire une semaine tranquille semaine  $i$ 
8      $i \leftarrow i+1$ ;

```

2. Proposer une formule récursive pour calculer le nombre  $p(i)$  maximal de points qu'on peut obtenir en travaillant jusqu'à la semaine  $i$ .
3. Donner un algorithme de type programmation dynamique pour calculer  $p(n)$  et aussi produire la stratégie de travail associée.

#### Exercice 4.

*Le retour du sac-à-dos*

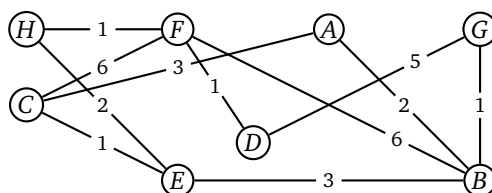
On revient sur le problème du sac-à-dos. Pour une instance  $(T, (t_0, v_0), \dots, (t_{n-1}, v_{n-1}))$  de ce problème, où toutes les valeurs sont entières, on dispose d'un sac-à-dos ayant une taille  $T$  et d'un ensemble d'objets  $O_0, \dots, O_{n-1}$ , chaque objet  $O_i$  ayant une valeur  $v_i$  et une taille  $t_i$ . Le but est toujours de charger le sac-à-dos en maximisant la valeur emportée, mais cette fois, les objets ne sont fractionnables : pour tout  $i = 1, \dots, n$ , soit on prend l'objet  $O_i$ , soit on ne le prend pas.


Pour résoudre cette version du problème, on va proposer deux solutions basées sur la programmation dynamique. Pour la première, on note  $V(m, t)$  la valeur maximale que l'on peut atteindre en ne prenant que des objets parmi  $O_0, \dots, O_m$ , pour une taille totale de  $t$ , pour  $m < n$  et  $t \leq T$ .

1. Que vaut  $V(m, t)$  si  $t < t_m$  ? Et si  $m = 0$  ?
2. Établir une relation de récurrence exprimant  $V(m, t)$  en fonction de valeurs  $V(m', t')$  avec  $m' < m$  et  $t' < t$ . On pourra étudier deux cas, selon que l'on choisisse l'objet  $O_m$  ou non.
3. À l'aide de la formule précédente, écrire un algorithme résolvant le problème. Estimer la complexité de votre algorithme. Cette complexité est-elle polynomiale en la taille de l'entrée ?
4. On souhaite maintenant fournir, en plus de la valeur optimale, le choix d'objets permettant d'obtenir cette valeur optimale. À l'aide d'une valeur  $C_{[m,t]}$ , on encodera le choix fait lors du calcul de  $V(m, t)$ .
  - (a) Compléter votre algorithme afin de calculer les valeurs  $C_{[m,t]}$ .
  - (b) Répondre ensuite à la question en donnant un algorithme pour calculer un choix optimal d'objets.
5. On peut faire une variante de l'algorithme précédent, en définissant  $t(m, v)$  comme étant la taille minimale d'un sac-à-dos qui atteindrait la valeur  $v$  à l'aide des objets  $O_0$  à  $O_m$ . Décrire la solution du problème (valeur maximale) à partir des  $t(m, v)$ , une formule récursive pour  $t(m, v)$  (avec les cas de base) et un algorithme correspondant (version sans et avec reconstruction de la solution). Quelle est la complexité obtenue ? Est-elle meilleure ou moins bonne que l'algorithme précédent ?

### Exercice 5.

Algorithme de Dijkstra



 Appliquer l'algorithme de Dijkstra pour calculer la distance de A à tous les autres sommets du graphe.

### Exercice 6.

Algorithme de Bellman-Ford

L'algorithme de Bellman-Ford permet de calculer les distances d'un sommet du graphe à tous les autres, même en présence d'arêtes de poids négatif. On interdit cependant des cycles dont le poids total (somme des poids des arêtes) serait négatif : tourner en rond permettrait de faire des chemins de longueur aussi petite que souhaitée, et il n'existerait pas de *plus court chemin*.

1. Montrer qu'en l'absence de cycle négatif, pour tous  $s$  et  $v$ , il existe un *plus court chemin* de  $s$  à  $v$  ne passe pas deux fois par le même sommet.

On considère donc un graphe pondéré  $G = (S, A, p)$  avec  $p : A \rightarrow \mathbb{R}$  (poids positifs et négatifs), et un sommet  $s \in G$ . Pour tout sommet  $v \in G$ , on note  $\text{dist}_\ell(v)$  le poids de plus court chemin *empruntant au plus  $\ell$  arêtes* entre  $s$  et  $v$ .

2. Que vaut  $\text{dist}_0(v)$  pour chaque  $v \in G$  ?
3. Montrer que la distance de  $s$  à  $v$  est  $\text{dist}_{n-1}(v)$  où  $n$  est le nombre de sommets de  $G$ .
4. Montrer que pour  $\ell > 0$ ,

$$\text{dist}_\ell(v) = \min\{\text{dist}_{\ell-1}(v), \min_{uv \in A} (\text{dist}_{\ell-1}(u) + p(u, v))\}.$$

5. En déduire un algorithme de calcul de la distance de  $s$  à n'importe quel sommet  $v$  et analyser sa complexité.
6. (*bonus*) Généraliser au calcul des distances entre chaque couple de sommets  $u$  et  $v$ .