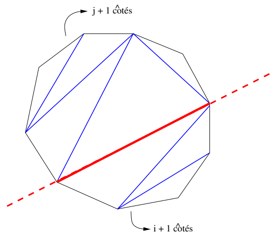
Correction TD 3 - Programmation dynamique

Exercice 1 - Triangulation de polygônes

1 - On procède par récurrence sur $n \geq 3$.
Pour $n = 3$, le polygône est un triangle, qui a bien $n-2=1$ triangle et $n-3=0$ corde.
Soit $n \in \mathbb{N}, n \geq 3$. On suppose le résultat démontré pour tout polygône convexe possédant un nombre de côtés strictement inférieur à n. Soit alors un polygône à n côtés et une triangulation de ce polygône. Considérons une corde qui divise ce polygône en deux polygônes, respectivement à $(i+1)$ et $(j+1)$ côtés avec $i+j=n, i \geq 2, j \geq 2$ (cf figure ci-dessous).

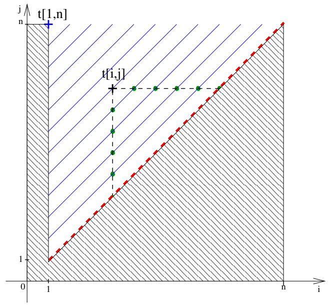


Par hypothèse de récurrence, on a $(i-2)$ cordes dans le premier polygône et $j-2$ dans le second, ce qui fait pour le polygône entier $(i-2)+(j-2)+1 = n-3$ cordes en tout (le $+1$ représente la corde qui séparait le polygône en deux).

De même, on obtient un total de $(i-1)+(j-1) = n-2$ triangles dans le polygône en tout, ce qui achève la démonstration.

2 - Pour trouver une formule de récurrence sur t, il suffit de s'aperçevoir que le $v_{i-1}v_j$ fait partie d'un triangle dont le troisième sommet est v_k. On en déduit la formule : $t[i,j] = min_{i \leq k \leq j-1}(t[i,k] + t[k+1,j] + w(i-1,k,j))$ (*On remarque que la convention* $t[i,i]$ *est bien adaptée puisque la formule ci-dessus reste vraie quand* $j = i+1$ *: on a bien* $t[i,i+1] = w(i-1,i,i+1)$.)

Nous pouvons maintenant écrire un algorithme qui calcule $t[1,n]$, et ce à partir des $t[k,j], i+1 \leq k \leq j$ et des $t[i,k], i \leq k \leq j-1$, c'est à dire, si l'on représente les valeurs à calculer dans un diagramme (i,j), à partir des valeurs situées "en-dessous" et "à droite" du point $(1,n)$. On s'aperçoit alors qu'il suffit de calculer les $t[i,j]$ pour i de 1 à n et pour $j \geq i$, et ce par $(j-i)$ croissant.



Ceci donne l'algorithme :

début
  pour i ***de*** 0 ***à*** n **faire**
    $t[i,i] \leftarrow 0$
  pour d ***de*** 1 ***à*** $n-1$ **faire**
    pour i ***de*** 1 ***à*** $n-d$ **faire**
      $t[i,i+d] \leftarrow min_{i \leq k \leq j-1}(t[i,k] + t[k+1,j] + w(i-1,k,j))$
  Retourner $t[1,n]$
fin

Cet algorithme calcule $t[i,j]$ en $2(j-i)$ additions et en $j-i-1$ comparaisons pour calculer le min.

Le nombre total d'additions est donc : $A_n = \sum_{d=1}^{n-1}((n-d)\cdot(2d))$ où $(n-d)$ correspond au nombre de $t[i,j]$ à calculer sur la diagonale $j-i=d$ et $2d$ au nombre d'additions dans le calcul de chaque $t[i,j]$ où $j-i=d$, ce qui nous donne :

$$\begin{aligned} A_n &= 2n \cdot \sum_{d=1}^{n-1} d - 2\sum_{d=1}^{n-1} d^2 \\ &= 2n \cdot \frac{n\cdot(n-1)}{2} - 2\frac{(n-1)\cdot n\cdot(2n-1)}{6} \\ &= \frac{(n-1)\cdot n\cdot(n+1)}{3} \\ &\sim n^3/3 = \Theta(n^3) \end{aligned}$$

Pour calculer le nombre T_n de tests à effectuer, il suffit de remarquer qu'on fait deux fois moins de tests que d'additions dans le calcul du minimum, ce qui nous donne $T_n = A_n/2 - C_n$ où C_n représente le nombre total de $t[i,j]$ calculés (ie le nombre d'affectations) On en déduit :

$$\begin{aligned} T_n &= A_n/2 - \sum_{d=1}^{n-1}(n-p) \\ &= \Theta(n^3) \end{aligned}$$

La complexité globale de l'algorithme est donc en $\Theta(n^3)$.

3 - Dans le cas général, il faut autant de valeurs pour définir la fonction w qu'il n'y a de triangles possibles, soit $C_{n+1}^3 = \Theta(n^3)$. Comme il faudra bien lire chacune de ces valeurs pour construire une triangulation optimale, n'importe quel algorithme doit fonctionner en une complexité d'au moins $\Theta(n^3)$. Notre algorithme est donc optimal en complexité du point de vue de l'ordre de grandeur.

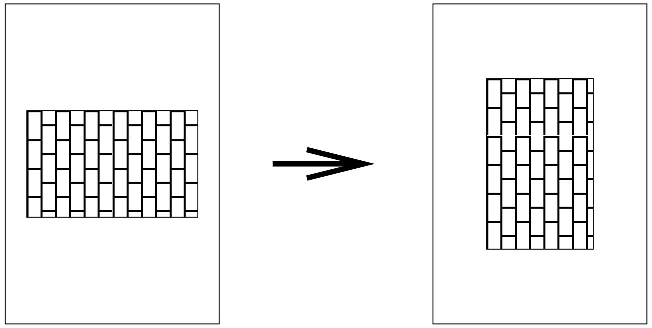
4 - Dans le cas où le poids d'un triangle est égal à son aire, toutes les triangulations donnent le même poids qui est l'aire du polygône. L'algorithme précédent est donc inadapté, et on peut à la place procéder par exemple comme suit :
Algorithme
- Prendre une triangulation quelconque (par exemple avec les cordes $(v_0v_i)_{2 \leq i \leq n-1}$).
- Faire la somme des poids des triangles.

Cet algorithme n'effectue aucun test et $n-3$ additions (bien en dessous du $\Theta(n^3)$!).

Exercice 2 - Jeu de construction

Si on prend un parallélépipède quelconque $(x_i,\ y_i,\ z_i)$, on s'aperçoit qu'on peut le poser sur la tour de 6 manières différentes.

On peut commencer par remarquer que si on peut poser une brique sur une autre de façon à ce que la longueur de la 1^{re} brique soit parallèle à la largeur de la 2^{e}, alors on peut aussi poser la 1^{re} brique sur la 2^{e} de façon à ce que les largeurs (et donc aussi les longueurs) des deux briques soient parallèles. On peut donc faire l'hypothèse que les briques seront posées les unes sur les autres de telle sorte que les largeurs des briques soient toutes parallèles entre elles. (*Le nombre de configurations par parallélépipède est ainsi réduit à 3.*)



Remarquons ensuite qu'un même parallélépipède ne peut être utilisé au plus que deux fois (dans deux configurations différentes). En effet, considérons le parallélépipède (x_i, y_i, z_i) avec $x_i \leq y_i \leq z_i$: si on la pose une première fois, alors, dans le meilleur des cas, on peut encore poser sur la tour toutes briques (x_j, y_j, z_j) telles que $x_j < y_i$ et $y_j < z_i$ (en supposant que $x_j \leq y_j \leq z_j$). Ainsi, si les inégalités sont strictes pour x_i, y_i, et z_i, on peut encore poser le parallélépipède. Pour le poser une troisième fois, il faudrait qu'il posséde un côté de longueur strictement inférieure à x_i. Ce n'est pas le cas, est donc on ne peut le poser que deux fois. Celà a pour conséquence que la tour de hauteur maximale sera composée d'au plus $2n$ parallélépipèdes.

Dorénavant, nous allons considérer les briques (L_i, l_i, h_i) avec $L_i \geq l_i$. Nous devons alors considérer $3n$ briques au lieu de n parallélépipèdes, mais il ne reste plus qu'une configuration possible par brique. (*Si le parallélépipède est (x_i, y_i, z_i) avec $x_i < y_i < z_i$, les 3 briques associées sont (z_i, y_i, x_i), (z_i, x_i, y_i) et (y_i, x_i, z_i).*)

Ainsi, on peut poser la brique (L_i, l_i, h_i) sur la brique (L_j, l_j, h_j) si et seulement si $\begin{cases} L_i < L_j \\ l_i < l_j \end{cases}$

On pose $\forall i \in \{1, \ldots, 3n\}, H_i =$ hauteur maximale parmi les tours se terminant par la i^{eme} brique. On a la relation de récurrence suivante :

$$\forall i \in \{1, \ldots, 3n\}, H_i = h_i + Max_{1 \leq j \leq 3n}(H_j / L_j > L_i \text{ et } l_j > l_i) \tag{1}$$

L'idée est de commencer par trier les $3n$ briques par (L_i, l_i) décroissant. Celà se fait en $O(nlog(n))$ comparaisons. L'équation précédente peut alors se réécrire :

$$\forall i \in \{1, \ldots, 3n\}, H_i = h_i + Max_{1 \leq j < i}(H_j / L_j > L_i \text{ et } l_j > l_i) \tag{2}$$

On calcule ainsi chaque H_i pour i allant de 2 à $3n$ en partant de $H_1 = h_1$. Le calcul de H_i nécessite 1 addition et $3i - 3$ comparaions dans le pire des cas. Le calcul de tous les $H_i, i \in 1, \ldots, 3n$ est donc linéaire en addition et quadratique en comparaison.

Enfin, il faut calculer $H = Max_{1 \leq i \leq 3n}(H_i)$ qui est la solution de notre problème, ce qui coûte $3n - 1$ comparaisons. La complexité totale est par conséquent en $O(n^2)$ pour les comparasions et en $O(n)$ pour les additions.

Et l'algorithme est :

---

début
- liste $\leftarrow$ []
- **pour** i ***de*** 1 ***à*** n **faire**
  - Ajouter les 3 configurations possibles du parallélépipède $(x_i,\ y_i,\ z_i)$ à *liste*
- Trier $liste = (L_i, l_i, h_i)_{i \in \{1, \ldots, 3n\}}$ par (L_i, l_i) décroissant.
- $H_1 \leftarrow h_1$
- **pour** i ***de*** 2 ***à*** $3n$ **faire**
  - $H_i \leftarrow h_i + Max_{1 \leq j < i}(H_j / L_j > L_i \text{ et } l_j > l_i)$
- $H \leftarrow Max_{1 \leq i \leq 3n}(H_i)$
- Renvoyer H

fin
