Corrigé du TD 2

1. Sous-arbre de poids maximal

On suppose que l'arbre A est enraciné et que chaque sommet de l'arbre pointe vers ces fils. On note $S(v)$ le poids maximal d'un sous-arbre de A ayant pour racine v (c'est-à-dire, dont tous les sommet sont des descendant de v). La fonction $S(v)$ satisfait la relation de récurrence

$$S(v) = \alpha(v) + \sum_{u \text{ fils de } v} \max(0, S(u)),$$

qui se traduit immédiatement en un algorithme récursif pour le calcul des poids $S(v)$.

Après le calcul de la fonction S, un parcours de l'arbre permet de trouver un sommet v_0 maximisant S. On détermine ensuite un sous-arbre B de racine v_0 ayant un poids $S(v_0)$ en initialisant $B = \{v_0\}$ puis en ajoutant récursivement à B tous les fils u des sommets de B vérifiant $S(u) > 0$.

On remarque que cet algorithme peut etre réalisé en temps linéaire et cela sans recourrir à la programmation dynamique.

2. Reconstruction de texte

1. On note $s[i,j]$ la partie du texte allant de la position i à la position j (incluse), de sorte que $s = s[1,n]$.

On considère la table booléenne $t(i)$ qui dit si le facteur gauche $s[1,i]$ de longueur i du texte peut être décomposé : cette table vérifie $t(0) =$ VRAI et satisfait la récurrence

$$t(i) = \bigvee_{1 \le j \le i} (\mathtt{dict}(s[j,i]) \wedge t(j-1)).$$

Le remplissage de cette table prend donc un temps $O(n^2)$, sauf si on a une borne M sur la taille du plus long mot du dictionnaire, auquel cas la complexité est $O(Mn)$.

2. Lors de la construction de la table $t(i)$ on remplit en parallèle une table $p(i)$ qui indique pour chaque position i telle que $t(i)$ est VRAI un indice j tel que $\mathtt{dict}(s[j,i]) \wedge t(j-1)$. Il suffit alors de suivre les indices en partant de $p(n)$ pour reconstruire une décomposition. L'espace utilisé est linéaire ainsi que le temps de calcul une fois les tables constituées.

3. Algorithme de Viterbi

i. Il suffit de parcourir le graphe en largeur ; partant du sommet v_0, on suit toutes les arêtes d'étiquette s_1, et on construit la liste des sommets d'arrivée ; pour chacun de ces sommets, on suit alors les arêtes d'étiquette s_2, etc.

Si le graphe a $|S|$ sommets, la complexité est en $O(n|S|^2)$.

ii. Soit $c = v_0 v_1 \ldots v_n$ un chemin le plus probable partant de v_0 d'étiquette s ; alors le chemin $v_1 \ldots v_n$ est un chemin le plus probable partant de v_1 et d'étiquette $s_2 \ldots s_n$.

Posons alors $f(v,i)$ le chemin le plus probable issu de v et dont l'étiquette est $s[i..n]$. On a dans ce cas la relation de récurrence

$$f(v,i) = \max_{u;\text{étiq}(v,u)=s_i} p(v,u) f(u,i+1),$$

avec la condition initiale $f(v,n+1) = 1$ pour tout v, qui nous donne un schéma de programmation dynamique. La complexité est dans ce cas $O(n|S|^2)$, car le calcul d'un terme de la table (qui est de taille $O(n|S|)$) prend un temps $O(|S|)$.

4. Ordonnancement pour minimiser le nombre de tâches en retard

i. Les tâches en retard peuvent être exécutées dans n'importe quel ordre après que toutes les tâches non en retard ont été exécutées. De plus, lorsque l'ordre des tâches a été fixé, un ordonnancement au plus tôt, c'est-à-dire dans lequel la première tâche commence à l'instant 0 et les autres tâches commencent à la fin de leur prédécesseur, est clairement optimal. Il faut donc trouver l'ordre des opérations non en retard. Dans un ordonnancement optimal, s'il y a deux tâches non en retard consécutives T_i et T_j telles que T_i précède T_j et $e_i \geq e_j$, on peut inverser ces deux tâches sans modifier la position des autres tâches et, dans ce nouvel ordonnancement T_i et T_j ne sont pas en retard. Il existe donc un ordonnancement optimal dans lequel les tâches non en retard sont ordonnées selon leur date d'échéance e_i.

ii. Soit t_{pj} la date de fin au plus tôt des sous-ensembles de $\{T_1, \cdots, T_j\}$ de poids supérieur ou égal à p. On conviendra que $t_{pj} = \infty$ s'il n'y a pas au moins un tel sous-ensemble. On a ainsi $t_{0j} = 0$ et $t_{p0} = \infty$ si $p > 0$. On s'intéresse maintenant à la relation de récurrence. On considère le sous-ensemble S de $\{T_1, \cdots, T_{j+1}\}$ et de poids supérieur ou égal à p et qui peut être exécuté le plus rapidement (S se termine par définition à $t_{p,j+1}$). Soit T_{j+1} est dans S, soit il ne n'est pas. Si $T_{j+1} \notin S$, alors $S \subseteq \{T_1, \cdots, T_j\}$ et S est donc le sous-ensemble de $\{T_1, \cdots, T_j\}$ et de poids supérieur ou égal à p qui peut être exécuté le plus rapidement, S se termine donc à t_{pj}. Si $T_{j+1} \in S$, cette tâche a la plus grande date d'échéance et on peut l'ordonnancer en dernier. L'ensemble des tâches qui la précèdent a un poids d'au moins $p - p_{j+1}$ et cet ensemble doit pouvoir être exécuter le plus rapidement possible. On a donc :

$$t_{p,j+1} = \begin{cases} \min(t_{pj}, t_{p-p_{j+1},j} + d_{j+1}) & \text{si } t_{p-p_{j+1},j} + d_{j+1} \leq e_{j+1} \\ t_{pj} & \text{sinon} \end{cases}$$

Puisqu'il n'y a pas de sous-ensemble de poids strictement supérieur à $\sum p_j$, on peut arrêter la récurrence quand p atteint cette valeur, ce qui donne un algorithme en $O(n \sum p_i)$. La plus grande valeur p pour laquelle $t_{pn} < \infty$ donne le poids maximal des tâches sans retard. L'ensemble correspondant peut être déduit des valeurs t_{pj} en prenant les j tels que $t_{pj} \neq t_{p,j-1}$.

5. Problème du voyageur de commerce

i. Une énumération exhaustive prend en compte toutes les tournées possibles. Une tournée est entièrement caractérisée par l'ordre de parcours des sommets $\{2, \ldots, n\}$. Il y a donc $(n-1)!$ telles tournées.

ii. Soit un chemin réalisant l'optimum $C(S,k)$, et soit m le dernier sommet visité avant k par cette tournée. Il est alors bien clair que le chemin auquel on a enlevé k réalise l'optimum

$C(S\setminus\{k\},m)$, sans quoi on peut améliorer le chemin de 1 à k. On a donc $C(S\setminus\{k\},m)+d_{mk} = C(S,k)$. Par suite, il est facile de déduire la relation de récurrence

$$C(S,k) = \min_{\ell\in S}(C(S\setminus\{k\},\ell) + d_{\ell k}).$$

On initialise la récurrence par $C(\{k\},k) = d_{1k}$.

Cette relation de récurrence doit s'exploiter via une technique de programmation dynamique, dans la mesure où il faut s'attendre à ce que les sous-problèmes soient rencontrés plusieurs fois.

Le traitement d'un sous-problème (S,k) prend un temps $|S|-1 < n$, et le nombre de sous-problèmes est $\sum_{P\subset\{2,\ldots,n\}} |P| \leqslant n2^n$. La complexité du calcul est donc $O(n^2 2^n)$, ce qui est bien plus favorable que $(n-1)!$.

Reste à expliquer comment trouver la tournée optimale. Il suffit de prendre

$$\min_{r\in\{2,\ldots,n\}} (C(\{2,\ldots,n\},r) + d_{r1}).$$

Pour le pseudo-code, on suppose qu'on dispose d'une fonction de hachage qui associe à chaque partie de $\{2,\ldots,n\}$ un entier de $[0,2^{n-1}-1]$. Cela peut se faire facilement en associant à S la somme $\sum_{i\in S} 2^{i-2}$.

1. Pour y de 2 à n faire
   1.1. $C[\{y\},y] \leftarrow d[1,y]$
   1.2. $P[\{y\},y] \leftarrow 1$
2. Pour ℓ de 1 à $n-1$ faire
   2.1. Pour $S \subset \{2,\ldots,n\}$, $|S| = \ell$, faire
      2.1.1. Pour $k \in S$ faire
         2.1.1.1. $C[S,k] \leftarrow \infty$
         2.1.1.2. Pour $j \in S\setminus\{k\}$ faire
            2.1.1.2.1. Si $C[S\setminus\{k\}),j] + d[j,k] < C[S,k]$ alors
               2.1.1.2.1.1. $C[S,k] \leftarrow C[S\setminus\{k\}),j] + d[j,k)]$
               2.1.1.2.1.2. $P[S,k] \leftarrow j$
3. $m \leftarrow \infty$.
4. Pour r de 2 à n faire
   4.1. Si $C[\{2,\ldots,n\},r] + d_{r1} < m$ alors
      4.1.1. $m \leftarrow C[\{2,\ldots,n\},r] + d_{r1}$;
      4.1.2. $P[\{2,\ldots,n\},1] \leftarrow r$.
5. Renvoyer m.

Pour afficher le circuit optimal à l'aide de la table P, on procède comme suit :

Affiche(P,S,x)
1. Affiche $(P,S\setminus\{x\},P[S,x])$
2. print(x)

et on appelle Affiche$(P,\{2,\ldots,n\},1)$.