Exercice 7-1: Programmation Dynamique

Trouver le chemin le plus court de A à B. Les distances sont: $d(A,C) = 5, d(A,D) = 3, d(A,G) = 14, d(D,C) = 11, d(D,G) = 6, d(D,E) = 7, d(C,E) = 3, d(C,F) = 2, d(F,B) = 7, d(G,E) = 7, d(E,B) = 5, d(G,B) = 6$.

Solution

1) Initialiser $Q(N) = 200$ pour tous N.

2) $Q(B) = 0$

3) Les voisins de B sont mis à jour: $Q(F) = 7, Q(E) = 5, Q(G = 6)$.

4) mise-à-jour des voisins $Q(C) = 8, Q(D) = 12, Q(A) = 13$

5) Le plus court chemin: commencer à A. Choisir un voisin n avec $Q(n) = d(A,n) = Q(A)$. Continuer. Le plus court chemin est A-C-E-B.

Exercice 7-2: Alignement de séquences

On cherche à aligner les séquences `ABCXXDAB` et `ABCDAB`. Appliquer l'algorithme de Needleman et Wunsch et donner la mise en correspondance optimale.

Solution

Il s'agit d'une insertion de 'XX' dans la première séquence.

Exercice 7-3: Alignement de séquences

On cherche à aligner les séquences `ABCNJRQCLCRPM` et `AJCJNRCKCRBP`. Donner la mise en correspondance optimale obtenue à l'aide de l'algorithme de Needleman et Wunsch.

Solution

	A	B	C	N	J	R	Q	C	L	C	R	P	M
A	1	0	0	0	0	0	0	0	0	0	0	0	0
J	0	0	0	0	1	0	0	0	0	0	0	0	0
C	0	0	1	0	0	0	0	1	0	1	0	0	0
J	0	0	0	0	1	0	0	0	0	0	0	0	0
N	0	0	0	1	0	0	0	0	0	0	0	0	0
R	0	0	0	0	0	1	0	0	0	0	1	0	0
C	0	0	1	0	0	0	0	1	0	1	0	0	0
K	0	0	0	0	0	0	0	0	0	0	0	0	0
C	0	0	1	0	0	0	0	1	0	1	0	0	0
R	0	0	0	0	0	1	0	0	0	0	1	0	0
B	0	1	0	0	0	0	0	0	0	0	0	0	0
P	0	0	0	0	0	0	0	0	0	0	0	1	0

	A	B	C	N	J	R	Q	C	L	C	R	P	M
A	8	7	6	6	5	4	4	3	3	2	1	0	0
J	7	7	6	6	6	4	4	3	3	2	1	0	0
C	6	6	7	6	5	4	4	4	3	3	1	0	0
J	6	6	6	5	6	4	4	3	3	2	1	0	0
N	5	5	5	6	5	4	4	3	3	2	1	0	0
R	4	4	4	4	4	5	4	3	3	2	2	0	0
C	3	3	4	3	3	3	3	4	3	3	1	0	0
K	3	3	3	3	3	3	3	3	3	2	1	0	0
C	2	2	3	2	2	2	2	3	2	3	1	0	0
R	2	1	1	1	1	2	1	1	1	1	2	0	0
B	1	2	1	1	1	1	1	1	1	1	1	0	0
P	0	0	0	0	0	0	0	0	0	0	0	1	0

Le meilleur alignement est le suivant:

```
ABCNJ-RQCLCR-PM
AJC-JNR-CKCRBP
```

Exercice 7-4: Ordre de Complexité

A) Nombre de pas de calcul pour remplir toute la matrice: - nombre de cases au total: $N * M$. - nombre de cases à comparer pour mettre à jour la case (i, k) : $(i-1+k-1)$ - nombre de cases à comparer dans le pire des cas (c'est-à-dire quand on se retrouve dans la case en haut à gauche de la matrice): $(N-1+M-1)$ - au

final, on a donc $(N * M) * (N + M - 2)$ comparaisons à effectuer, ce qui est à peu près égal à $(N^2) * (2 * N)$, ce qui équivaut a un ordre de complexité de $O(N^3)$

B) Nombre de possibilités de chemins différentes pour un algorithme **naif**:

- à la première ville, on a N possibilités, alors N chemins différents.

- à la deuxième, on a de nouveau N choix différents ce qui fait N^2 chemins différents pour le moment.

- à la troisième ville, on en a encore N possibilités, =¿ N^3 chemins différents pour le moment.

- ... - a la M_e ville, on aura donc N^M chemins possibles.

Cet algorithme a donc un ordre de complexité exponentiel en N^M si on augmente le nombre de couche M, qui peut mener à une explosion combinatoire.

Si on utilise un algorithme de **programmation dynamique**, a chaque couche k, on aura à calculer N^2 possibilités. Par contre, ces possibilités s'additionnent (et ne se multiplient pas).

- N pas de calcul à faire à la couche M: on note la distance d pour chaque ville de la couche M et met $Q = d$.

- Pour calculer les valeur Q d'une ville de la couche $M - 1$, on regarde les valeurs Q de toutes les villes de la couche M, et ajoute la distance d entre les 2 villes. En prenant le minimum de ces N possibilités on obtient la valeur Q de cette ville. Ainsi, on a N calcul/comparaisons pour chaque ville de la couche $M - 1$, ou N^2 comparaisons.

- Ainsi de suite, pour calculer les valeur Q d'une ville de la couche $M - 2$, on regarde les valeurs Q de toutes les villes de la couche $M - 1$, et ajoute la distance d entre les 2 villes. En prenant le minimum de ces N possibilités on obtient la valeur Q de cette ville. Ainsi, on a N calcul/comparaisons pour chaque ville de la couche $M - 2$, ou N^2 comparaisons.

- au final, nous aurons effectué $(M-1)*N^2$ comparaisons pour calculer les valeurs Q de chaque ville. L'ordre de complexité de cet algorithme est donc de $M * N^2$

- Pour trouver le chemin le plus court on choisit ensuite à chaque couche la ville avec la valeurs Q minimale.

Du coup, pour que le 2e algo devienne plus efficace, il faut que le nombre M de couches satisfasse la condition $M > 2$. Ainsi, pour tout nombre de couches supérieur à 2, l'algo dynamique est plus efficace.