Plus courts chemins, programmation dynamique

1. Plus courts chemins à partir d'un sommet
2. Plus courts chemins entre tous les sommets
3. Semi-anneau
4. Programmation dynamique
5. Applications à la bio-informatique

Graphes orientés avec longueur

Soit un graphe orienté $G = (X, A, or, ext, \ell)$

- $X = \{x_1, x_2, \ldots, x_n\}$ Ensemble de sommets
- A Ensemble d'arcs
- $or : \ A \to X, ext : \ A \to X$ origine et extrémité de chaque arc
- $\ell : \ A \to \mathcal{Z}$ longueur des arcs

1. Une *marche* est une suite d'arcs $a_1, a_2, \ldots, a_p$ telle que pour tout $1 \leq i < p$:

$$or(a_{i+1}) = ext(a_i)$$

2. Un *chemin* est une marche $a_1, a_2, \ldots, a_p$ telle que pour tout $i \neq j$ on ait:

$$or(a_i) \neq or(a_j)$$

3. Un *circuit* est un chemin tel que $or(a_1) = ext(a_p)$

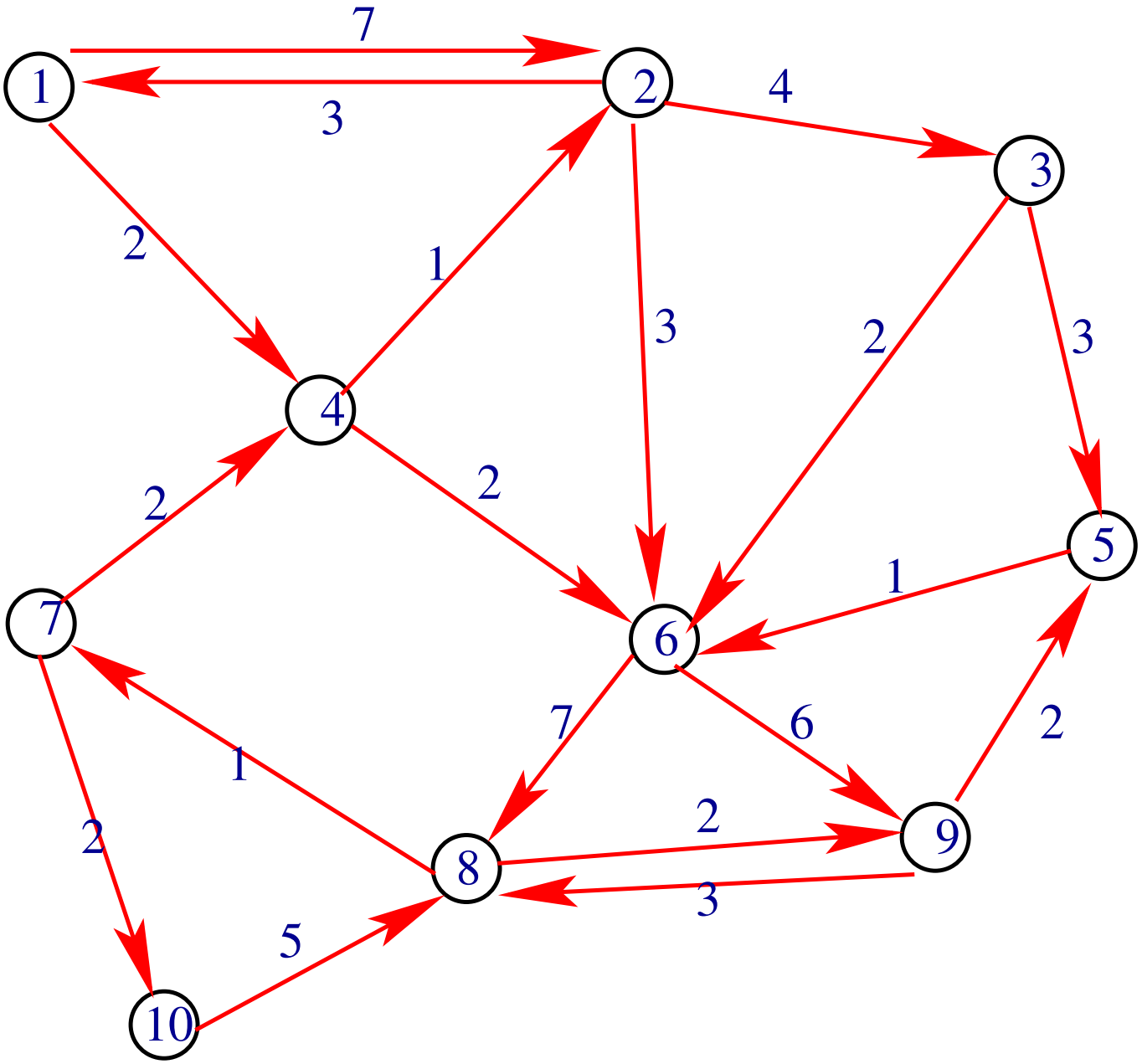
Plus courts chemins à partir d'un sommet

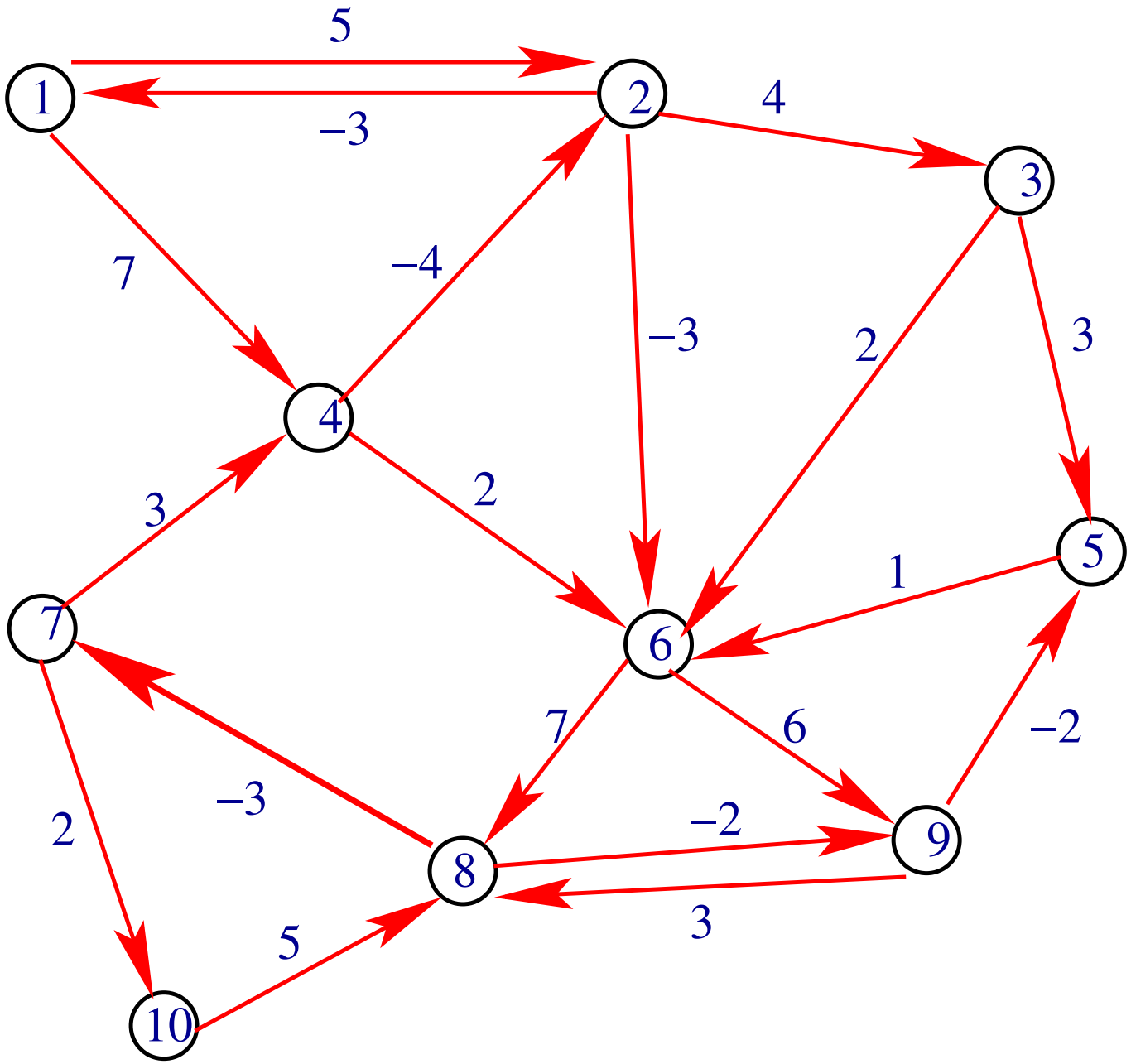
On se donne un graphe orienté G aux arcs valués par $l(x, y)$ et un sommet x_0, trouver pour chaque sommet x :

1. La longueur du plus court chemin de x_0 à x
2. La suite des arcs constituant ce plus court chemin

Structures de données :

- `d[x]` la longueur du plus court chemin de x_0 à x (avec mise à jour)
- `pere[x]` sommet qui précède x sur le plus court chemin de x_0 à x


1
2
3
4
5
6
7
8
9
10
7
3
4
2
1
3
2
3
2
2
1
2
7
6
2
1
2
2
3
5


5
1
2
4
−3
3
7
−4
−3
2
3
4
2
3
5
1
7
6
7
6
−2
2
−3
−2
9
8
3
5
10

Relaxation

Relaxation de l'arc x, y

- Si $d[y] > d[x] + l(x, y)$
- Alors
- $d[y] = d[x] + l(x, y)$
- $pere[y] = x$

Algorithme de Dijkstra

- Les distances sont supposées positives
- Un ensemble F initialisé à X
- les valeurs de `d[x]` sont initialisées à ∞ sauf : `d[x0] = 0`.

- Tant que F non vide :
- Soit y l'élément de F ayant un `d[y]` minimal
- Pour tout arc `a` d'origine `y` Faire

  `Relaxation(a)`
- Supprimer y de F.

Algorithme de Dijkstra: preuve de correction

- $\texttt{d[x]} \geq \delta(x_0, x)$
- lorsqu'un sommet x quitte F on ne peut pas trouver un chemin plus court entre x_0 et x par la suite
- Si le chemin le plus court entre x_0 et x passe par y, alors il est composé d'un chemin le plus court entre x_0 et y et d'un plus court y et x.
- point fixe

Implantation

1. Gérer F comme une file?
2. Trouver le plus petit `d[x]`
3. Heaps, Fibonacci Heaps
4. Retrouver le chemin le plus court

Algorithme de Dijkstra dans le cas d'arcs de longueurs négatives

La version précédente ne donne pas le bon résultat

On propose la modification suivante:

- Tant que F non vide :
- Soit y l'élément de F ayant un `d[y]` minimal
- Pour tout arc `a` d'origine `y` Faire
  `Relaxation(a)` Si relaxation modifie la valeur de $ext(a)$ alors ajouter $ext(a)$ à F.
- Supprimer y de F

Analyse de cette modification

Donne le bon résultat s'il n'y a pas de circuit de longueur négative

Mais un exemple montre que la complexité peut être exponentielle

Exemple de Johnson

Un graphe D_n ayant n sommets $x_1, x_2, \ldots, x_n$ et des arcs entre tous les sommets (dans les deux sens)

Les longueurs des arcs (x_i, x_j) pour $i < j$ est 2^n

Exemple de Johnson (suite)

Les longueurs des arcs (x_i, x_j) pour $i > j$ sont toutes négatives et sont données par les relations suivantes

$$\ell(x_2, x_1) = -2, \ \forall i > 2, \quad \ell(x_{i+1}, x_1) = \ell(x_i, x_1) - 2^{i-2} - 1$$

$$\forall i \geq 2, \quad 1 < j < i, \ \ \ell(x_i, x_j) = \ell(x_i, x_{j-1}) + 1$$

Exemple $n = 5$

$$\ell(x_5, x_1) = \ell(x_4, x_1) - 5 = \ell(x_3, x_1) - 3 - 5 = \ell(x_2, x_1) - 2 - 8 = -12$$

On obtient facilement la matrice des longueurs :

	x_1	x_2	x_3	x_4	x_5
x_1	0	32	32	32	32
x_2	-2	0	32	32	32
x_3	-4	-3	0	32	32
x_4	-7	-6	-5	0	32
x_5	-12	-11	-10	-9	0

$F = \{x_1, x_2, x_3, x_4, x_5\}$	∞	∞	∞	∞	0
$F = \{x_1, x_2, x_3, x_4\}$	-12	-11	-10	-9	0
$F = \{x_2, x_3, x_4\}$	-12	-11	-10	-9	0
$F = \{x_1, x_3, x_4\}$	-13	-11	-10	-9	0
$F = \{x_3, x_4\}$	-14	-13	-10	-9	0
$F = \{x_1, x_2, x_4\}$	-14	-13	-10	-9	0
$F = \{x_2, x_4\}$	-14	-13	-10	-9	0
$\ldots$	$\ldots$				
$F = \{x_4\}$	-15	-13	-10	-9	0
$F = \{x_1, x_2, x_3\}$	-16	-15	-14	-9	0
$\ldots$	$\ldots$				
$F = \{x_1\}$	-19	-17	-14	-9	0

Analyse de l'exécution

- Il n'y a pas de circuit de longueur négative
- x_1 sort une fois sur deux
- x_2 une fois sur quatre
- Au total 2^{n-1} étapes de calcul

Algorithme de Ford

- Les distances peuvent être négatives

- Pour i= 1 à n-1 faire

- Pour tout arc `a` du graphe Faire

  `Relaxation(a)`

Remarque Pour vérifier qu'il n'y a pas de circuit négatif on refait un tour de relaxation: si une nouvelle valeur est modifiée c'est qu'il existe un circuit négatif

Plus courts chemins entre tous les couples de sommets

On calcule les $\delta_k(x, y)$ par l'algorithme suivant :

- Pour tous les couples x, y on pose : $\delta_0(x, y) = l(a)$ s'il existe un arc a entre x et y et ∞ sinon.

- Pour $k = 1, 2, 3, \ldots, n$ faire
- Pour tout couple x, y
- Pour tout sommet z successeur de x
- $$\delta_{k+1}(x, y) = min(\delta_k(x, y), \delta_0(x, z) + \delta_k(z, y))$$

Preuve de valdité

- Il suffit de vérifier que $\delta_k(x, y)$ est la longueur du plus court chemin composé de k arcs au plus entre x et y
- Par récurrence sur k

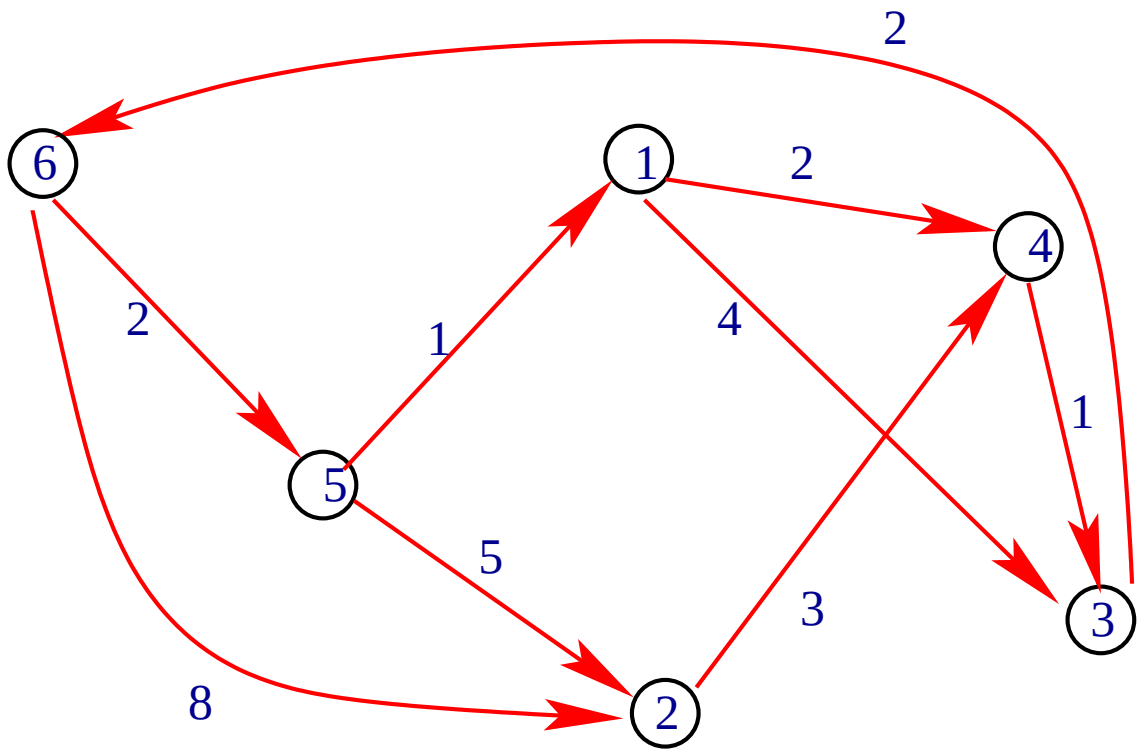
Recherche du chemin

- Maintenir un tableau `suivant[x][y]` qui contient le sommet qui suit x dans le chemin qui mène de x à y

Algorithme de Roy-Warshall (en n^3)

- Pour k de 1 à n faire
- Pour i de 1 à n faire
- Pour j de 1 à n faire
- Si `delta[i][j] > delta[i][k] + delta[k][j]`
- Alors `delta[i][j] = delta[i][k] + delta[k][j]`

Un exemple d'illustration



Semi-anneau

Un ensemble muni de deux lois notées $\odot, \oplus$*:*

- La loi $\oplus$ est associative, commutative
- Elle possède un élémént neutre $\overline{0}$
- La loi $\odot$ est associative
- La loi $\odot$ est distributive par rapport à $\oplus$
- $\overline{0}$ est un zéro pour $\odot$

Exemples de semi-anneaux

1. Un anneau, par exemple $\mathcal{Z}$
2. Les entiers naturels $\mathcal{N}$ avec $+$ et $\times$
3. Le semi-anneau de Boole à deux éléments 0, 1 avec $(1 + 1 = 1)$
4. Les opérations min + sur $\mathcal{N}$
5. Les opérations max $\times$ sur l'intervalle réel $[0, 1]$
6. Union et concaténation pour les ensembles de mots

Graphes à arcs valués sur un semi-anneau

Le semi anneau a pour lois $\odot, \oplus$:

- Sommets X, Arcs A, Valuation λ
- Valuation d'un chemin $f = a_1, a_2, \ldots, a_k$=
- $$\lambda(f) = \lambda(a_1) \odot \lambda(a_2) \odot \ldots \odot \lambda(a_k)$$
- Valuation d'un ensemble $\Gamma = \{f, g, \ldots u, v\}$ de chemins
- $$\lambda(f) = \lambda(f) \oplus \lambda(g) \oplus \ldots \oplus \lambda(u) \oplus \lambda(v)$$

Algorithme sur un semi anneau quelconque

On calcule les $\lambda_k(x, y)$ par l'algorithme suivant :

- Pour tous les couples x, y

  $\lambda_0(x, y) = \lambda(a)$ s' il existe un arc a entre x et y et ε sinon.

- Pour $k = 1, 2, 3, \ldots, n$ faire
- Pour tous les x, y faire

$$\lambda_{k+1}(x, y) = \lambda_k(x, y) \oplus \lambda_k(x, z) \odot \lambda_k(z, y)$$

Applications

- Min et $+$: Les plus courts chemins
- Max et $*$: La fiabilité maximale dans un réseau
- Semi-anneau de Boole: Existence de chemins
- Union, concaténation: Forme rationnelle d'un langage reconnu par un automate

Principe de la programmation dynamique

1. La solution d'un problème P_k contient les solutions de problèmes ayant des tailles plus petites
2. La construction de la solution pour P_k s'effectue à l'aide d'un faible nombre d'informations sur P_j, $j < k$

Exemple 1: faire un produit de matrices de tailles différentes

On se donne n matrices $M_1, M_2, \ldots, M_n$ chaque M_i a m_i lignes et m_{i+1} colonnes, les entrées sont des nombres (réels ou entiers). On cherche la matrice produit:

$$M_1 M_2 \ldots M_n$$

Il faut trouver l'ordre des produits à effectuer de façon à minimiser le nombre total d'opérations.

Remarque: Le calcul de $M_{i-1}M_i$ demande $O(m_{i-1}m_i m_{i+1})$ opérations.

Exemple Trois matrices M_1, M_2, M_3 de tailles respectives 10×100, 100×5 et 5×50

1. Le calcul de $M_1 M_2$ demande 5 000 multiplications de nombres le produit ensuite par M_3 fait faire 2 500 de plus soit un total de 7 500
2. Le calcul de $M_2 M_3$ demande 25 000 multiplications de nombres, la détermination ensuite de $M_1 M_2 M_3$ utilise 50 000 multiplications soit un total de 75 000

Conclusion Il vaut mieux faire $((M_1 M_2) M_3)$ que calculer $(M_1 (M_2 M_3))$

Cas général

- On note $c_{i,j}$ le nombre minimum de multiplications pour calculer $M_i M_{i+1} \ldots M_j$
- On a alors $c_{i,i} = 0$, $c_{i-1,i} = m_{i-1} m_i m_{i+1}$
- On découpe $M_i M_{i+1} \ldots M_j$ en $M_i \ldots M_k$ et $M_{k+1} \ldots M_j$ en choisissant le meilleur k:

$$c_{i,j} = \min_{k=i,j-1} [c_{i,k} + c_{k+1,j} + m_i m_{k+1} m_{j+1}]$$

- On doit déterminer $M_{1,n}$

Un exemple:

i	1	2	3	4	5	6	7
m_i	30	35	15	5	10	20	25

$c_{i,j}$	1	2	3	4	5	6
1	0	-	-	-	-	-
2	-	0	-	-	-	-
3	-	-	0	-	-	-
4	-	-	-	0	-	-
5	-	-	-	-	0	-
6	-	-	-	-	-	0

Un exemple:

i	1	2	3	4	5	6	7
m_i	30	35	15	5	10	20	25

$c_{i,j}$	1	2	3	4	5	6
1	0	15 750	-	-	-	-
2	-	0	2 625	-	-	-
3	-	-	0	750	-	-
4	-	-	-	0	1 000	-
5	-	-	-	-	0	5 000
6	-	-	-	-	-	0

Calcul de $c_{1,3}$:

i	1	2	3	4	5	6	7
m_i	30	35	15	5	10	20	25

$$c_{1,3} = \min \quad [c_{1,1} + c_{2,3} + m_1 m_2 m_4, \qquad c_{1,2} + c_{3,3} + m_1 m_3 m_4]$$

$$= \min \quad [0 + 2625 + 30 \times 35 \times 5, \quad 15750 + 0 + 30 \times 15 \times 5]$$

$$= \min \quad [7\,875, \qquad 18\,000]$$

Un exemple:

i	1	2	3	4	5	6	7
m_i	30	35	15	5	10	20	25

$c_{i,j}$	1	2	3	4	5	6
1	0	15 750	7 875	-	-	-
2	-	0	2 625	4 375	-	-
3	-	-	0	750	2 500	-
4	-	-	-	0	1 000	3 500
5	-	-	-	-	0	5 000
6	-	-	-	-	-	0

Un exemple:

i	1	2	3	4	5	6	7
m_i	30	35	15	5	10	20	25

$c_{i,j}$	1	2	3	4	5	6
1	0	15 750	7 875	9 375	-	-
2	-	0	2 625	4 375	7 125	-
3	-	-	0	750	2 500	5 375
4	-	-	-	0	1 000	3 500
5	-	-	-	-	0	5 000
6	-	-	-	-	-	0

Un exemple:

i	1	2	3	4	5	6	7
m_i	30	35	15	5	10	20	25

$c_{i,j}$	1	2	3	4	5	6
1	0	15 750	7 875	9 375	11 875	-
2	-	0	2 625	4 375	7 125	10 500
3	-	-	0	750	2 500	5 375
4	-	-	-	0	1 000	3 500
5	-	-	-	-	0	5 000
6	-	-	-	-	-	0

Un exemple:

i	1	2	3	4	5	6	7
m_i	30	35	15	5	10	20	25

$c_{i,j}$	1	2	3	4	5	6
1	0	15 750	7 875	9 375	11 875	15 125
2	-	0	2 625	4 375	7 125	10 500
3	-	-	0	750	2 500	5 375
4	-	-	-	0	1 000	3 500
5	-	-	-	-	0	5 000
6	-	-	-	-	-	0

Retrouver les étapes du calcul

un tableau pour les valeurs des k intermédiaires

$K_{i,j}$	1	2	3	4	5	6
1	-	-	1	3	3	3
2	-	-	-	3	3	3
3	-	-	-	-	3	3
4	-	-	-	-	-	5
5	-	-	-	-	-	-
6	-	-	-	-	-	-

Pour calculer $M_i \dots M_j$ on pose $k = K_{i,j}$ donné par le tableau puis on calcule $M_i \dots M_k$ et $M_{k+1} \dots M_j$ et on les multiplie entre elles.

Sac à dos

- Des objets $1, 2, \ldots, n$ de poids $w_1, w_2, \ldots, w_n$
- Chacun rapporte un bénéfice $b_1, b_2, \ldots, b_n$
- Trouver le bénéfice maximum réalisable sachant que la charge maximale est P_0

Sac à dos: idée pour P_0 entier pas trop grand

- On note $B(k,p)$ le bénéfice maximal reálisable avec des objets $1, 2, \ldots, k$ et le poids maximal p
- On a pour $k = 1$:

$$B(1,p) = \begin{cases} 0 & \text{si} \quad p < w_1 \\ b_1 & \text{si} \quad p \geq w_1 \end{cases}$$

- Pour $k > 1$

$$B(k,p) = \begin{cases} B(k-1,p) & \text{si} \quad p < w_k \\ \max(B(k-1,p), B(k-1,p-w_k) + b_k) & \text{si} \quad p \geq w_k \end{cases}$$

Exemple pour le sac à dos

Objets	1	2	3	4	5	6	7	8
Poids	2	3	5	2	4	6	3	1
Bénéfices	5	8	14	6	13	17	10	4

Poids total $P_0 = 12$

Sac à dos:

Objets	1	2	3	4	5	6	7	8
Poids	2	3	5	2	4	6	3	1
Bénéfices	5	8	14	6	13	17	10	4

$B(k,p)$	0	1	2	3	4	5	6	7	8	9	10	11	12
$k=1$	0	0	5	5	5	5	5	5	5	5	5	5	5

Sac à dos:

Objets	1	2	3	4	5	6	7	8
Poids	2	3	5	2	4	6	3	1
Bénéfices	5	8	14	6	13	17	10	4

$B(k,p)$	0	1	2	3	4	5	6	7	8	9	10	11	12
$k=1$	0	0	5	5	5	5	5	5	5	5	5	5	5
$k=2$	0	0	5	8	8	13	13	13	13	13	13	13	13

Sac à dos:

Objets	1	2	3	4	5	6	7	8
Poids	2	3	5	2	4	6	3	1
Bénéfices	5	8	14	6	13	17	10	4

$B(k,p)$	0	1	2	3	4	5	6	7	8	9	10	11	12
$k=1$	0	0	5	5	5	5	5	5	5	5	5	5	5
$k=2$	0	0	5	8	8	13	13	13	13	13	13	13	13
$k=3$	0	0	5	8	8	14	14	19	22	22	27	27	27

Sac à dos:

$B(k,p)$	0	1	2	3	4	5	6	7	8	9	10	11	12
$k=1$	0	0	5	5	5	5	5	5	5	5	5	5	5
$k=2$	0	0	5	8	8	13	13	13	13	13	13	13	13
$k=3$	0	0	5	8	8	14	14	19	22	22	27	27	27
$k=4$	0	0	6	8	11	14	14	20	22	25	28	28	33
$k=5$	0	0	6	8	13	14	19	21	24	27	28	33	35
$k=6$	0	0	6	8	13	14	19	21	24	27	30	33	36
$k=7$	0	0	6	10	13	16	19	23	24	29	31	34	37
$k=8$	0	4	6	10	14	17	20	23	27	29	33	35	38

Alignements de séquences

Pour deux mots sur l'alphabet $\{A, C, G, T\}$ on cherche un alignement qui optimise une fonction qui mesure la similarité

Exemple: **GATGCAT et** $ATCGAT$

G	A	T	-	G	C	A	T
-	A	T	C	G	-	A	T

Algorithme de calcul du meilleur alignement

$p(a,b)$ est d'autant plus grand que les lettres sont similaires.

$b \neq a$, $p(a,a) > 0 > p(a,b)$

$p(a,b) = p(b,a)$

$q < 0$ exprime la similarité d'une lettre avec l'espacement: $p(a,e) = p(e,a) = q$

$$ms_{i,j} = \max \begin{cases} ms_{i-1,j-1} + p(u_i, v_j) \\ ms_{i-1,j} + q \\ ms_{i,j-1} + q \end{cases}$$

Principe d'un algorithme en espace linéaire

- Le calcul des meilleurs scores peut se faire sur une seule ligne
- Le problème est de retrouver l'alignement une fois que l'on a le score
- On procède par divide and conquer
- Pour aligner u et v on prend i comme la moitié de la longueur de u
- On calcule le score a_j du meilleur alignement de $u_1 u_2 \ldots u_{i-1}$ avec chaque facteur gauche $v_1 v_2 \ldots v_j$ de v
- On calcule le score b_k du meilleur alignement de $u_{i+1} u_{i+2} \ldots u_n$ avec chaque facteur droit $v_k v_{k+1} \ldots v_n$ de v
- On cherche le $a_{j-1} + b_{j+1} + p(u_i, v_j)$ le meilleur

Calcul de la complexité de l'algorithme en espace linéaire

- On montre que le nombre de comparaison $C(m,n) \leq 2mn$
- Vrai pour $m = 1$
- $$C(m,n) \leq \frac{mn}{2} + \frac{mn}{2} + C(m/2, j) + C(m/2, n-j)$$
- $$C(m,n) \leq \frac{mn}{2} + \frac{mn}{2} + mj + m(n-j) \quad \leq 2mn$$