Algorithmique et Complexité
TD 4/7 – Programmation dynamique

CentraleSupélec – Gif

ST2 – Gif

Introduction

L'objectif de ce TD est de s'exercer à la modélisation de problèmes et à leur résolution par programmation dynamique.

Ce TD est en 3h et comprend des parties de pratique en Python.

Plan

Plan

Exercice 1

Nous disposons d'un euro que nous voulons placer d'une manière optimale pendant 10 périodes. Nous connaissons avec certitude le coefficient de gain c_i pour un placement en début de période i.

période i	1	2	3	4	5	6	7	8	9	10
c_i	2	4	1	2	6	2	2	4	1	4

A chaque période, on a deux options, soit on place la somme totale disponible, soit on garde toute la somme et on n'en place rien. Supposons qu'au début de la période i on dispose d'une somme x. S'il n'y a pas de placement en période i, la somme x sera disponible en début de période $i+1$. Si on place une somme x en début de période i on reçoit la valeur $c_i x$, mais l'argent sera immobilisé pendant cette période et pendant la période $i+1$ et la nouvelle valeur sera disponible seulement en début de période $i+2$ pour être réinvestie ou simplement récupérée.

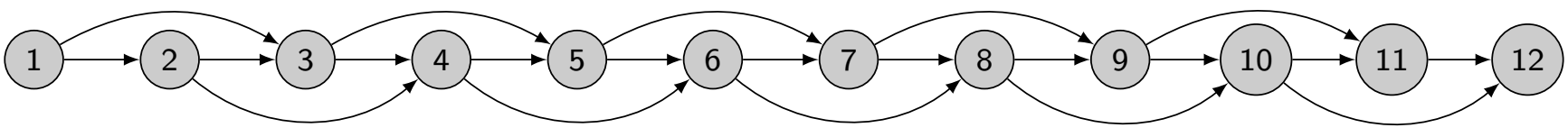
Exercice 1 : question 1

Il faut trouver la politique de placement total qui rapportera le plus d'argent en début de la période 12 (ce qui signifie qu'il n'y aura plus de placement en début de la période 11).

Question : En ignorant dans un premier temps les coefficients de gains, modéliser le problème par un graphe non-pondéré (les sommets et les arcs) où chaque politique de placement totale correspondra à un chemin de ce graphe. De quelle nature est-il ? Dessiner le graphe.

Exercice 1 : question 1 (correction)

Le graphe est **orienté**. Il y a 12 sommets qui représentent 12 moments (début de période) où on décide de placer ou non la somme disponible. Le sommet terminal de chaque arc indique le plus proche début de période où l'argent sera disponible. Chaque sommet a donc le degré sortant 2 (sauf pour les deux dernières périodes) et le degré entrant 2 (sauf les deux premières périodes). On obtient ainsi un **DAG**.



Exercice 1 : question 2

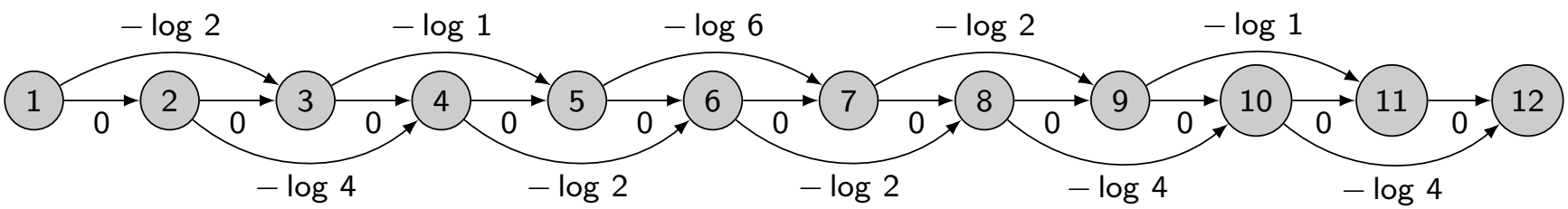
Question : Si on veut se ramener à un problème du plus court chemin, quels poids faut-il affecter aux arcs ? Complétez le graphe.

Exercice 1 : question 2 (correction)

Il s'agit de **maximiser** une fonction de taux c_i **multiplicative** !

1. $c_i \mapsto \log(c_i)$ pour passer à une fonction (de distances) **additive**
2. $\log(c_i) \mapsto -\log(c_i)$ pour passer à une **minimisation**

La valuation d'un arc entre deux sommets non-successifs est $-\log(c_i)$ où c_i est le coefficient de gain prévu par la période du sommet i de départ. 0 entre deux sommets successifs.



Le graphe étant sans circuits (absorbants), la notion de plus court chemin reste bien définie et on pourra ainsi le calculer dans la question suivante.

Exercice 1 : question 3

Question : Quel algorithme vu en cours peut-on utiliser pour répondre à l'exercice ? Trouver la solution de l'exemple numérique.

Exercice 1 : question 3 (correction)

En vu des arcs négatifs, on peut opter pour l'algorithme de Ford-Bellman.

La formule de récurrence, pour $i \in 1 \dots |V|$:

$$\mathrm{OPT}(i, v) = \min\left(\mathrm{OPT}(i-1, v), \min_{u \in V}(\mathrm{OPT}(i-1, u) + \omega((u, v)))\right)$$

Algorithme Bellman-Ford (1956, 1958)

Principe

- Basé sur le principe de la programmation dynamique
- Calcule le coût du plus court chemin

  mais on peut retrouver le chemin à partir de la table de mémoïsation. . .

Rappel : données du problème

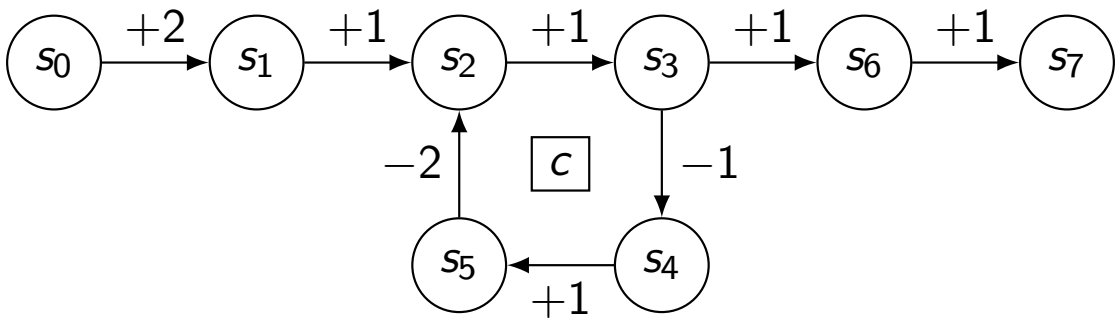
Un graphe orienté pondéré G quelconque, deux sommets s et t

y compris des poids négatifs. . .

➜ Quelle est la longueur du plus court chemin reliant s à t ?

Attention aux circuits absorbants !

Définition : circuit absorbant



Le circuit c dans cet exemple est *un circuit absorbant*, parce que $\sum_{e=(v,u)\in c} \omega(e) < 0$.

Plus court chemin avec poids négatifs

Besoin d'une formulation plus précise :

➜ On cherche le plus court chemin sans circuit !

Algorithme Bellman-Ford (1956, 1958)

Principe

- Basé sur le principe de la programmation dynamique
- Calcule le coût du plus court chemin

  mais on peut retrouver le chemin à partir de la table de mémoïsation. . .

Propriétés

✓ Supporte les poids négatifs (contrairement à Dijkstra)

✓ Détecte s'il y a un circuit absorbant

Principes de l'algorithme Bellman-Ford

Décomposer en sous-problèmes

Soit $\mathrm{OPT}(i, v)$ la longueur du chemin le plus court vers le nœud cible t depuis un nœud $v, v \neq t$ qui contient au plus i arcs.

Construction récursive d'une solution

- $\mathrm{OPT}(i, v) = \min_{(v,u)\in E}\big(\mathrm{OPT}(i-1, u) + \omega((v, u))\big)$

  $\rightarrow$ Pour rejoindre t, aller d'abord à u en prenant le plus court chemin en $i-1$ étapes.
- Sauf s'il y a déjà un chemin en $i-1$ étapes depuis v qui est plus court que tout le reste !

  $\rightarrow$ Auquel cas $\mathrm{OPT}(i, v) = \mathrm{OPT}(i-1, v)$

Principes de l'algorithme Bellman-Ford

Décomposer en sous-problèmes

Soit $\mathrm{OPT}(i, v)$ la longueur du chemin le plus court vers le nœud cible t depuis un nœud $v, v \neq t$ qui contient au plus i arcs.

Construction récursive d'une solution

$$\mathrm{OPT}(i, v) = \min\left(\mathrm{OPT}(i-1, v), \min_{u \in V}(\mathrm{OPT}(i-1, u) + \omega((u, v)))\right)$$

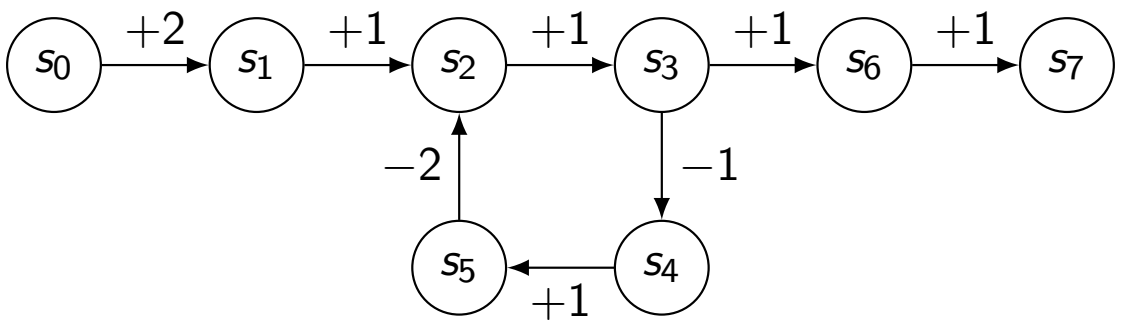
Stocker les $\mathrm{OPT}(i, v) \rightarrow$ tableau à 2 dimensions.

Quand s'arrêter ?

Propriété

- Dans un graphe sans circuit, le plus court chemin contient au plus $|V| - 1$ arcs.

➜ C'est vrai aussi dans un graphe quelconque mais sans circuit absorbant.

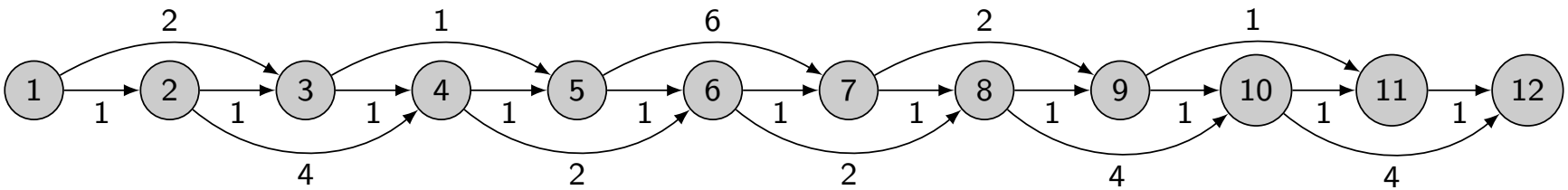


➜ On s'arrête lorsqu'on a atteint les chemins comprenant $|V| - 1$ arcs.

Une implémentation de Bellman-Ford (matrice d'adjacence)

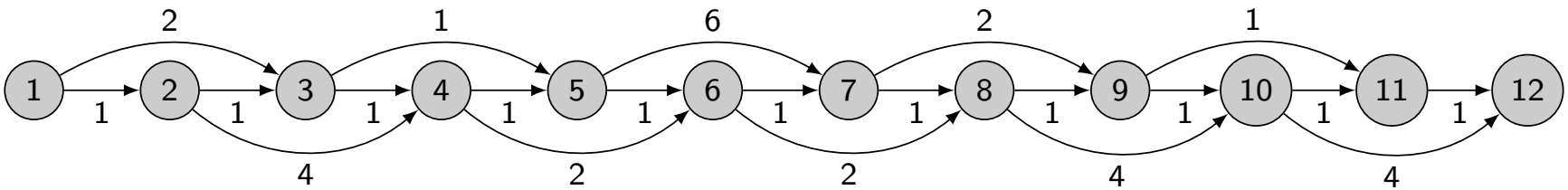
```
import math

# graph est une matrice d'adjacence
n = len(graph)

# initialisation du tableau OPT
OPT = [[math.inf for _ in range(n)] for _ in range(n+1)]
OPT[0][n-1] = 0

# remplissage iteratif du tableau OPT
for i in range(1,n):
    for v in range(n):
        OPT[i][v] = OPT[i-1][v]
        for u in range(n):
            if graph[v][u] != None and \
                    OPT[i][v] > OPT[i-1][u] + graph[v][u]:
                OPT[i][v] = OPT[i-1][u] + graph[v][u]
```

Exercice 1 : question 3 (correction)



	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9	s_{10}	s_{11}	s_{12}
0												
1												
2												
3												
4												
5												
6												
7												
8												
9												
10												
11												

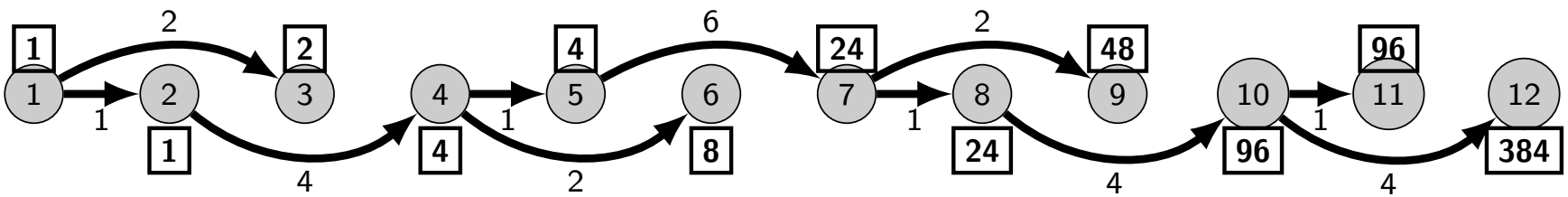
Exercice 1 : question 3 (correction)



	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9	s_{10}	s_{11}	s_{12}
0												$1,\_$
1										$4,s_{12}$	$1,s_{12}$	$1,\_$
2								$16,s_{10}$	$4,s_{10}$	$4,s_{12}$	$1,s_{12}$	$1,\_$
3						$32,s_8$	$16,s_8$	$16,s_{10}$	$4,s_{10}$	$4,s_{12}$	$1,s_{12}$	$1,\_$
4				$64,s_6$	$96,s_7$	$32,s_8$	$16,s_8$	$16,s_{10}$	$4,s_{10}$	$4,s_{12}$	$1,s_{12}$	$1,\_$
5		$256,s_4$	$96,s_5$	$96,s_5$	$96,s_7$	$32,s_8$	$16,s_8$	$16,s_{10}$	$4,s_{10}$	$4,s_{12}$	$1,s_{12}$	$1,\_$
6	$256,s_2$	$384,s_4$	$96,s_5$	$96,s_5$	$96,s_7$	$32,s_8$	$16,s_8$	$16,s_{10}$	$4,s_{10}$	$4,s_{12}$	$1,s_{12}$	$1,\_$
7	$384,s_2$	$384,s_4$	$96,s_5$	$96,s_5$	$96,s_7$	$32,s_8$	$16,s_8$	$16,s_{10}$	$4,s_{10}$	$4,s_{12}$	$1,s_{12}$	$1,\_$
8	$384,s_2$	$384,s_4$	$96,s_5$	$96,s_5$	$96,s_7$	$32,s_8$	$16,s_8$	$16,s_{10}$	$4,s_{10}$	$4,s_{12}$	$1,s_{12}$	$1,\_$
9	$384,s_2$	$384,s_4$	$96,s_5$	$96,s_5$	$96,s_7$	$32,s_8$	$16,s_8$	$16,s_{10}$	$4,s_{10}$	$4,s_{12}$	$1,s_{12}$	$1,\_$
10	$384,s_2$	$384,s_4$	$96,s_5$	$96,s_5$	$96,s_7$	$32,s_8$	$16,s_8$	$16,s_{10}$	$4,s_{10}$	$4,s_{12}$	$1,s_{12}$	$1,\_$
11	$384,s_2$	$384,s_4$	$96,s_5$	$96,s_5$	$96,s_7$	$32,s_8$	$16,s_8$	$16,s_{10}$	$4,s_{10}$	$4,s_{12}$	$1,s_{12}$	$1,\_$

Exercice 1 : question 3 (correction)

Voici le résultat final de la version multiplicative de Bellman et Ford :



Un chemin optimal de la période 1 à 12 a comme coeff de gain 384 ! et emprunte les sommets [2, 4, 5, 7, 8, 10]
Il faut noter que s'agissant d'un DAG c-à-d pas de circuits et en particulier pas de circuits absorbants, l'algo de Ford-Bellman aboutit donc à la bonne solution.

Plan

Exercice 2 : question 1

Un magasin de ski possède en stock un ensemble de n paires de skis de différentes longueurs. Il reçoit une demande d'un club de vacances pour une location de m paires pour des clients dont la taille est connue avec $m < n$. Le propriétaire du magasin souhaite optimiser le confort des skieurs. Pour cela, il propose de minimiser la somme des écarts (en valeur absolue) entre la taille du client et la taille de ses skis.

Question : Formalisez ce problème d'optimisation.

Exercice 2 : question 1 (correction)

Entrées :

- un ensemble de n entiers $s_1, \ldots, s_n$ représentant les longueurs de chaque paire de ski
- un ensemble de m entiers $c_1, \ldots, c_m$ représentant les tailles de chaque client

Sortie : une fonction d'affectation injective $f : [1, m] \to [1, n]$ telle que $\sum_{i \in [1,m]} |c_i - s_{f(i)}|$ est minimale.

On peut constater que la complexité d'un algo bête et exhaustif sera exponentielle $\mathcal{O}(n^m)$.

Exercice 2 : question 2

Considérons un algorithme glouton qui alloue les skis à l'aide de la méthode suivante : on cherche d'abord le couple (ski,client) qui a le plus petit écart *en valeur absolue*, on alloue la paire de ski au client et on recommence avec les skis et les clients restants. Écrivez en Python le code de cet algorithme. Pour vous aider, rendez-vous sur la page de pratique du TD :

https://wdi.centralesupelec.fr/1CC2000/TD4ProgEn

Exercice 2 : question 3

Question : Quelle est la complexité de cet algorithme ?

Exercice 2 : question 3 (correction)

L'algorithme est en $\mathcal{O}(m^2 \times n)$. En effet, la recherche du meilleur appariement est en $\mathcal{O}(m \times n)$ (deux boucles for imbriquées) et le tout est fait exactement m fois (un client est servi à chaque passage dans la boucle while).

Exercice 2 : question 4

Question : L'algorithme glouton donne-t-il une solution optimale ?

Exercice 2 : question 4 (correction)

Voici un contre exemple avec 2 clients et 2 paires de ski :
$C = \{170, 140\}$ et $S = \{160, 200\}$.

On fera le test en Python pour vérifier. (Démo)

Exercice 2 : question 5

On suppose sans perte de généralité que les n paires de skis et les m skieurs sont classés par ordre croissant de taille. On note $Sol[i, j]$ le coût de la solution optimale (c'est-à-dire la somme des écarts de taille) pour les i premiers skis et les j premiers clients. On souhaite utiliser une méthode de type « programmation dynamique » pour trouver le coût de la solution optimale $Sol[n, m]$.

On peut constater que, pour 2 paires de skis et 2 skieurs, mieux vaut attribuer la plus petite paire au plus petit skieur et la plus grande paire au plus grand skieur. Ainsi, si on généralise ce résultat à i paires de skis et j skieurs, lorsque les $j - 1$ premiers skieurs sont servis, le skieur j peut choisir sa paire de ski et on ne trouvera pas de meilleure solution en permutant sa paire avec un skieur plus petit (servi avant lui).

C'est cette observation qui va nous permettre de définir notre algorithme de résolution par programmation dynamique.

Exercice 2 : question 5

Question : Donnez la formule de récurrence qui définit la valeur de $Sol[i, j]$ en fonction des valeurs de $Sol[i-1, j]$ et $Sol[i-1, j-1]$.

Exercice 2 : question 5 (correction)

- Soit le skieur j ne prend pas la paire de ski numéro i. Il a donc déjà pris une paire plus petite et on ne gagnerait pas à donner cette paire i à un skieur plus petit que j (propriété énoncée plus haut). Dans ce cas, $Sol[i, j] = Sol[i - 1, j]$.
- Soit le skieur j prend la paire de ski i et dans ce cas, $Sol[i, j] = Sol[i - 1, j - 1] + |C[j] - S[i]|$.

La formule de récurrence est donc :

$$Sol[i, j] = min(Sol[i - 1, j], Sol[i - 1, j - 1] + |C[j] - S[i]|)$$

Il faut initialiser tous les $Sol[i_{\geq 0}, 0]$ à 0 (ça ne coûte rien tant que personne n'a pris de skis) et tous les $Sol[0, j_{\geq 1}]$ à ∞ (coût infini si on n'a alloué de skis à aucun skieur).

Exercice 2 : question 6

Question : Retournez sur la page de pratique du TD pour écrire en Python un algorithme qui utilise la sous-structure optimale explicitée par la question précédente. Donnez la complexité de l'algorithme.

Exercice 2 : question 6 (correction)

La complexité de tri des paires de skis et des skieurs est $\mathcal{O}(n \log n) + \mathcal{O}(m \log m)$ et la complexité de l'algo de prog dynamique est $\mathcal{O}(n \times m)$