Recherche Opérationnelle

Programmation Dynamique

Nadia Brauner & Alice Joffard

25 mars 2021

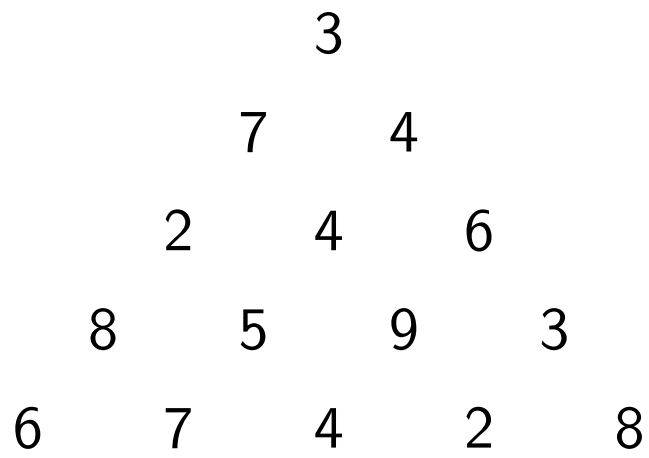
Enregistrement du cours

https://youtu.be/5nYqEsMHlO0

Attention, sur l'algorithme de Bellman-Ford, j'ai oublié de préciser qu'on peut ne faire qu'une itération des sommets uniquement parce qu'on prend l'ordre des sommets "naturel" (on a toujours parcouru tous les voisins entrant d'un sommet avant d'arriver à ce sommet), qui n'existe que parce qu'on s'est restreint au cas des DAG (si il y a un cycle, un tel ordre n'existe pas toujours). En pratique on se restreindra toujours aux DAGs dans ce cours, mais si on a des cycles, on peut avoir à faire $n-1$ itérations de tous les sommets avant d'arriver à la solution optimale

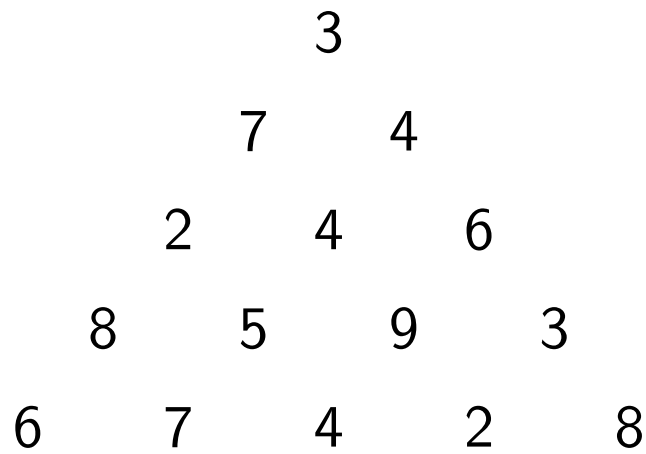
A vous de jouer

- Enoncé: trouvez un chemin de haut en bas qui maximise la somme des nombres traversés



A vous de jouer

- Enoncé: trouvez un chemin de haut en bas qui maximise la somme des nombres traversés



- Solutions:

A vous de jouer

A vous de jouer

- Preuve:

A vous de jouer

- Preuve:
  - En chaque nombre i, la somme maximale d'en haut jusqu'à i est: le maximum entre celle d'au dessus à droite et celle d'au dessus à gauche, auquel on ajoute i.

A vous de jouer

- Preuve:
  - En chaque nombre i, la somme maximale d'en haut jusqu'à i est: le maximum entre celle d'au dessus à droite et celle d'au dessus à gauche, auquel on ajoute i.
  - On calcule de haut en bas la somme maximale en appliquant cette formule:

A vous de jouer

- Preuve:
  - En chaque nombre i, la somme maximale d'en haut jusqu'à i est: le maximum entre celle d'au dessus à droite et celle d'au dessus à gauche, auquel on ajoute i.
  - On calcule de haut en bas la somme maximale en appliquant cette formule:

$$
\begin{array}{ccccccccc}
 & & & 3_{3} & & & & & \\
 & & 7_{10} & & 4_{7} & & & & \\
 & 2_{12} & & 4_{14} & & 6_{13} & & & \\
8_{20} & & 5_{19} & & 9_{23} & & 3_{16} & & \\
6_{26} & & 7_{27} & & 4_{27} & & 2_{25} & & 8_{26}
\end{array}
$$

A vous de jouer

- Preuve:
  - En chaque nombre i, la somme maximale d'en haut jusqu'à i est: le maximum entre celle d'au dessus à droite et celle d'au dessus à gauche, auquel on ajoute i.
  - On calcule de haut en bas la somme maximale en appliquant cette formule:
  - On garde la somme maximale parmi les nombres d'en bas

$$
\begin{array}{ccccccccc}
 & & & & 3_{3} & & & & \\
 & & & 7_{10} & & 4_{7} & & & \\
 & & 2_{12} & & 4_{14} & & 6_{13} & & \\
 & 8_{20} & & 5_{19} & & 9_{23} & & 3_{16} & \\
6_{26} & & 7_{27} & & 4_{27} & & 2_{25} & & 8_{26}
\end{array}
$$

A vous de jouer

A vous de jouer

- Enoncé: Partager ces n pièces en deux ensembles égaux:
  - $\{5 \quad 9 \quad 3 \quad 8 \quad 2 \quad 5\}$?

A vous de jouer

- Enoncé: Partager ces n pièces en deux ensembles égaux:
  - $\{5 \quad 9 \quad 3 \quad 8 \quad 2 \quad 5\}$?

- Solution: $\{5 \quad 9 \quad 2\}|\{3 \quad 8 \quad 5\}$

A vous de jouer

- Enoncé: Partager ces n pièces en deux ensembles égaux:
  - $\{5 \quad 9 \quad 3 \quad 8 \quad 2 \quad 5\}$?
  - Trouver toutes les solutions. Et quand on a un grand nombre de pièces ?
- Solution: $\{5 \quad 9 \quad 2\}|\{3 \quad 8 \quad 5\}$

A vous de jouer

- Enoncé: Partager ces n pièces en deux ensembles égaux:
  - {5 9 3 8 2 5} ?
  - Trouver toutes les solutions. Et quand on a un grand nombre de pièces ?
- Solution: {5 9 2}|{3 8 5}
- Méthode: On veut un ensemble de pièce dont la somme soit la moitié de la somme totale. On remplit m:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	V					V											
2	V					V				V					V		
3	V			V		V			V	V			V		V		
4	V			V		V			V	V		V	V		V		V

  - $m(i,j) = V$ si je peux avoir j avec les i premières pièces
  - $m(i,0) = V$ pour $i = 0$ à n
  - $m(i,j) = m(i-1,j)$ ou $m(i-1,j-piece(i))$

    $i = 1$ à n et $j = piece(i)$ à 16

Programmation Dynamique

Programmation Dynamique

Définition:

Programmation Dynamique

Définition:

Wikipedia

La *Programmation Dynamique* est une méthode algorithmique pour résoudre des problèmes d'optimisation. Elle consiste à résoudre un problème en le décomposant en sous-problèmes, puis à résoudre les sous-problèmes, des plus petits aux plus grands en stockant les résultats intermédiaires.

Programmation Dynamique

Définition:

Wikipedia

La *Programmation Dynamique* est une méthode algorithmique pour résoudre des problèmes d'optimisation. Elle consiste à résoudre un problème en le décomposant en sous-problèmes, puis à résoudre les sous-problèmes, des plus petits aux plus grands en stockant les résultats intermédiaires.

- Terme introduit par Bellman en 1950

Programmation Dynamique

Définition:

Wikipedia

La *Programmation Dynamique* est une méthode algorithmique pour résoudre des problèmes d'optimisation. Elle consiste à résoudre un problème en le décomposant en sous-problèmes, puis à résoudre les sous-problèmes, des plus petits aux plus grands en stockant les résultats intermédiaires.

- Terme introduit par Bellman en 1950
- *Programmation*, à la mode

Programmation Dynamique

Définition:

Wikipedia

La *Programmation Dynamique* est une méthode algorithmique pour résoudre des problèmes d'optimisation. Elle consiste à résoudre un problème en le décomposant en sous-problèmes, puis à résoudre les sous-problèmes, des plus petits aux plus grands en stockant les résultats intermédiaires.

- Terme introduit par Bellman en 1950
- *Programmation*, à la mode
- *Dynamique*, modélise des processus évolutifs

Programmation Dynamique

Définition:

Wikipedia

La *Programmation Dynamique* est une méthode algorithmique pour résoudre des problèmes d'optimisation. Elle consiste à résoudre un problème en le décomposant en sous-problèmes, puis à résoudre les sous-problèmes, des plus petits aux plus grands en stockant les résultats intermédiaires.

- Terme introduit par Bellman en 1950
- *Programmation*, à la mode
- *Dynamique*, modélise des processus évolutifs
- Méthode générale (pas dédiée à un problème particulier)

Programmation Dynamique

Définition:

Wikipedia

La *Programmation Dynamique* est une méthode algorithmique pour résoudre des problèmes d'optimisation. Elle consiste à résoudre un problème en le décomposant en sous-problèmes, puis à résoudre les sous-problèmes, des plus petits aux plus grands en stockant les résultats intermédiaires.

- Terme introduit par Bellman en 1950
- *Programmation*, à la mode
- *Dynamique*, modélise des processus évolutifs
- Méthode générale (pas dédiée à un problème particulier)
- Pour des problèmes ayant des propriétés structurelles

Principe de sous-optimalité

Principe de sous-optimalité

- On veut résoudre un problème P sur une instance I

Principe de sous-optimalité

- On veut résoudre un problème P sur une instance I

Structure spécifique de P

- Les "morceaux" d'une solution optimale sont optimaux

Principe de sous-optimalité

- On veut résoudre un problème P sur une instance I

Structure spécifique de P

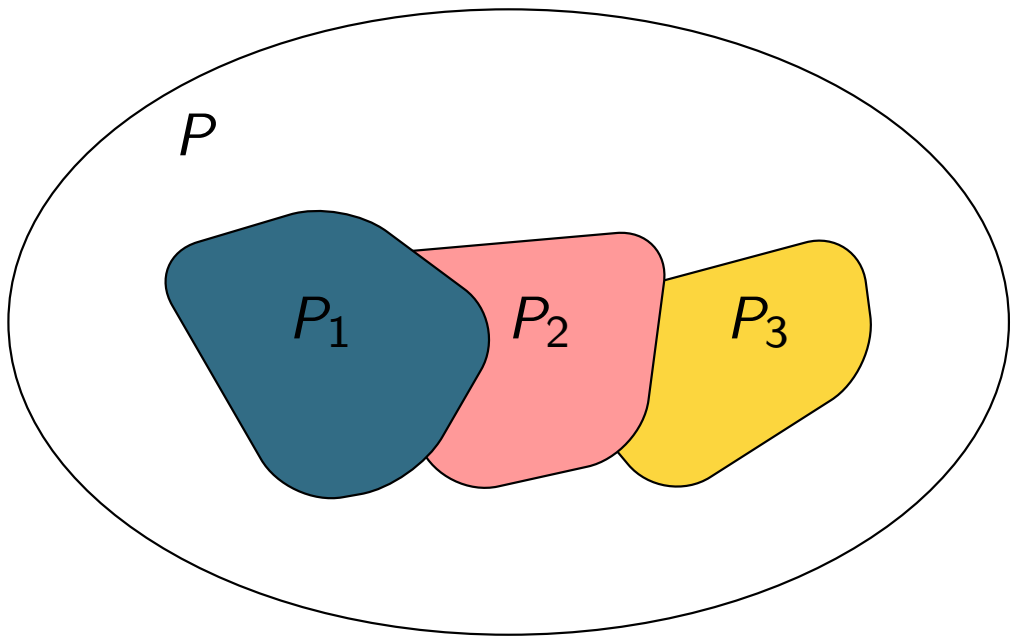
- Les "morceaux" d'une solution optimale sont optimaux
  - Le problème P se décompose en sous-problèmes $P_1, \ldots, P_k$
    L'optimum sur P s'obtient à partir des optimaux des sous-problèmes

Principe de sous-optimalité

- On veut résoudre un problème P sur une instance I

Structure spécifique de P

- Les "morceaux" d'une solution optimale sont optimaux
  - Le problème P se décompose en sous-problèmes $P_1, \ldots, P_k$
    L'optimum sur P s'obtient à partir des optimaux des sous-problèmes



Principe de sous-optimalité

Principe de sous-optimalité

- L'optimum sur une instance I peut se construire à partir de solutions optimales sur des instances plus "simples" $I_1, \ldots, I_k$

$$I^* = f(I_1^*, \ldots, I_k^*)$$

Principe de sous-optimalité

- L'optimum sur une instance I peut se construire à partir de solutions optimales sur des instances plus "simples" $I_1, \ldots, I_k$

$$I^* = f(I_1^*, \ldots, I_k^*)$$

  - On a une formulation récursive de I^*

Principe de sous-optimalité

- L'optimum sur une instance I peut se construire à partir de solutions optimales sur des instances plus "simples" $I_1, \ldots, I_k$

$$I^* = f(I_1^*, \ldots, I_k^*)$$

  - On a une formulation récursive de I^*
  - Il suffit de calculer l'optimum pour $I_1^*, \ldots, I_k^*$ puis d'appliquer f

Principe de sous-optimalité

- L'optimum sur une instance I peut se construire à partir de solutions optimales sur des instances plus "simples" $I_1, \ldots, I_k$

$$I^* = f(I_1^*, \ldots, I_k^*)$$

  - On a une formulation récursive de I^*
  - Il suffit de calculer l'optimum pour $I_1^*, \ldots, I_k^*$ puis d'appliquer f
  - Chaque I_j^* s'exprime à son tour en fonction d'instances plus simples

Principe de sous-optimalité

- L'optimum sur une instance I peut se construire à partir de solutions optimales sur des instances plus "simples" $I_1, \ldots, I_k$

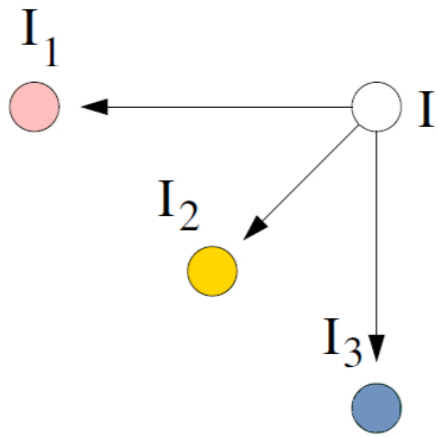
$$I^* = f(I_1^*, \ldots, I_k^*)$$

  - On a une formulation récursive de I^*
  - Il suffit de calculer l'optimum pour $I_1^*, \ldots, I_k^*$ puis d'appliquer f
  - Chaque I_j^* s'exprime à son tour en fonction d'instances plus simples
  - Jusqu'à obtenir une instance de base $\underline{I}$ directement calculable

Principe de sous-optimalité

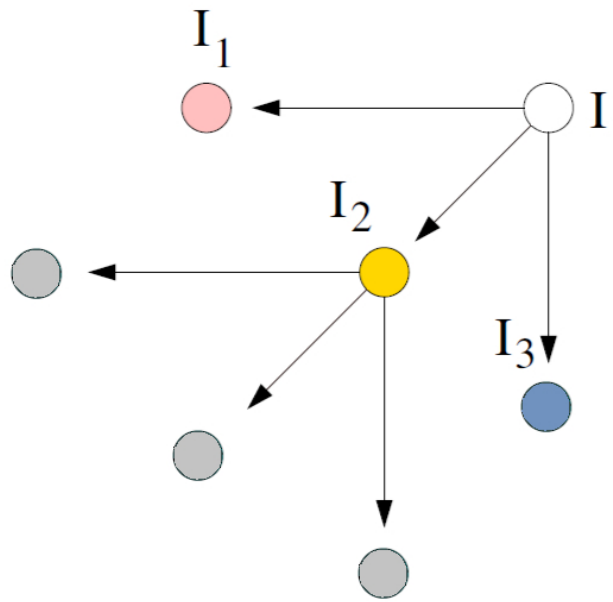
Principe de sous-optimalité

- Calcul récursif de l'optimum:



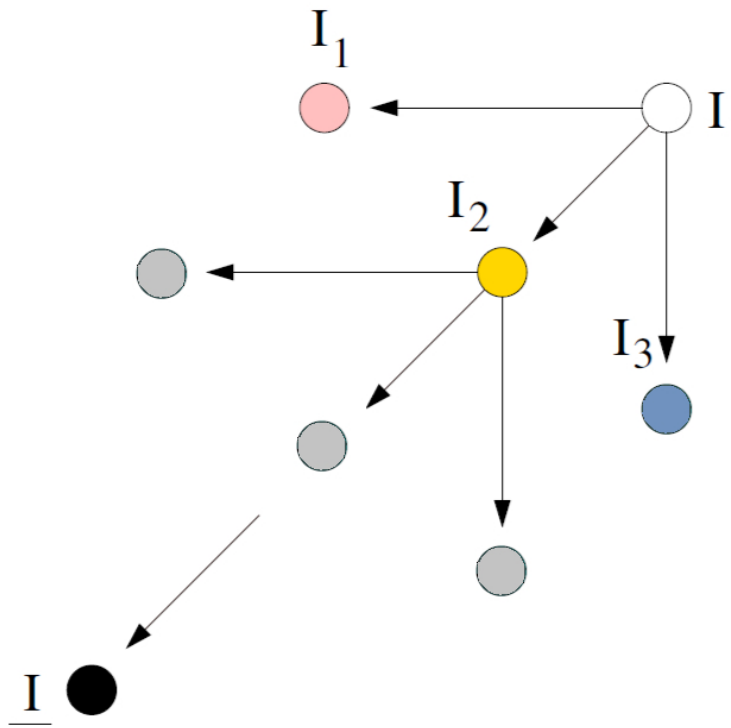
Principe de sous-optimalité

- Calcul récursif de l'optimum:



Principe de sous-optimalité

- Calcul récursif de l'optimum:



Programmation dynamique

Programmation dynamique

- DAG:

Programmation dynamique

- DAG:
  - DAG : *Directed Acyclic Graph*

Programmation dynamique

- DAG:
  - DAG : *Directed Acyclic Graph*
  - Les noeuds peuvent étre mis sur une ligne

Programmation dynamique

- DAG:
  - DAG : *Directed Acyclic Graph*
  - Les noeuds peuvent étre mis sur une ligne
  - Tous les arcs vont de gauche à droite

Programmation dynamique

- DAG:
  - DAG : *Directed Acyclic Graph*
  - Les noeuds peuvent étre mis sur une ligne
  - Tous les arcs vont de gauche à droite
- en Programmation dynamique:

Programmation dynamique

- DAG:
  - DAG : *Directed Acyclic Graph*
  - Les noeuds peuvent étre mis sur une ligne
  - Tous les arcs vont de gauche à droite
- en Programmation dynamique:
  - Le DAG est implicite

Programmation dynamique

- DAG:
  - DAG : *Directed Acyclic Graph*
  - Les noeuds peuvent étre mis sur une ligne
  - Tous les arcs vont de gauche à droite
- en Programmation dynamique:
  - Le DAG est implicite
  - Noeuds : sous-Problèmes

Programmation dynamique

- DAG:
  - DAG : *Directed Acyclic Graph*
  - Les noeuds peuvent étre mis sur une ligne
  - Tous les arcs vont de gauche à droite
- en Programmation dynamique:
  - Le DAG est implicite
  - Noeuds : sous-Problèmes
  - Arcs : dépendances entre les sous Problèmes

Programmation dynamique

- DAG:
  - DAG : *Directed Acyclic Graph*
  - Les noeuds peuvent étre mis sur une ligne
  - Tous les arcs vont de gauche à droite
- en Programmation dynamique:
  - Le DAG est implicite
  - Noeuds : sous-Problèmes
  - Arcs : dépendances entre les sous Problèmes
- Si on voit les sommets de gauche à droite, on est sûr que les prédécesseurs ont été vus: l'information nécessaire est disponible

Programmation dynamique

- DAG:
  - DAG : *Directed Acyclic Graph*
  - Les noeuds peuvent étre mis sur une ligne
  - Tous les arcs vont de gauche à droite
- en Programmation dynamique:
  - Le DAG est implicite
  - Noeuds : sous-Problèmes
  - Arcs : dépendances entre les sous Problèmes
- Si on voit les sommets de gauche à droite, on est sûr que les prédécesseurs ont été vus: l'information nécessaire est disponible
- Trouver le plus court/long chemin dans un DAG est facile grâce à l'algorithme de Bellman-Ford

Algorithme de Bellman-Ford

Algorithme de Bellman-Ford

- Problème du plus court chemin:

  Instance : Un graphe dirigé, une source s, un puit p, des distances sur les arcs, pas de cycle de poids négatif.
  Solutions réalisables : Chemins de s à p
  Critère : Minimiser la distance d'un chemin de s à p

Algorithme de Bellman-Ford

- Problème du plus court chemin:

  INSTANCE : Un graphe dirigé, une source s, un puit p, des distances sur les arcs, pas de cycle de poids négatif.
  SOLUTIONS RÉALISABLES : Chemins de s à p
  CRITÈRE : Minimiser la distance d'un chemin de s à p

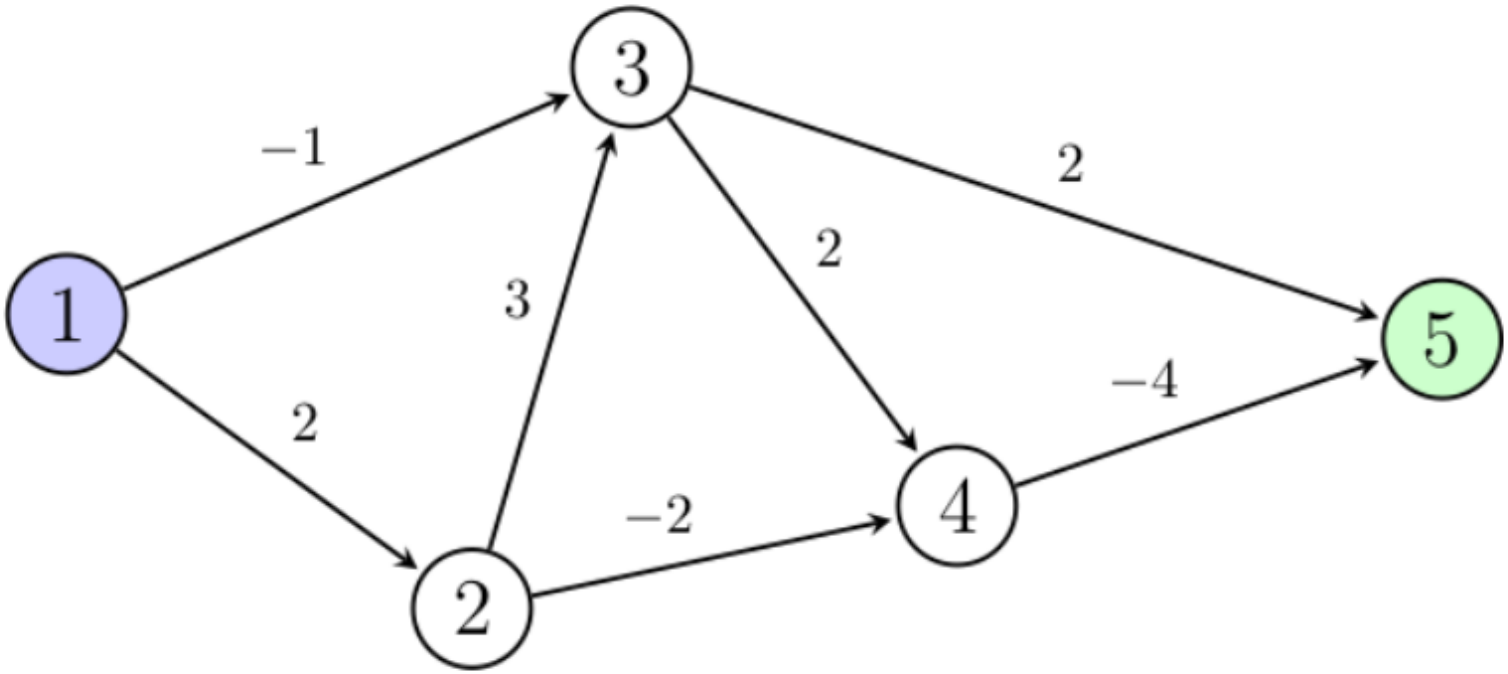
- Algorithme de Bellman-Ford:

```
fonction Bellman_Ford(G, s)
  pour chaque sommet u du graphe
  |         d[u] = +∞
  d[s] = 0

 pour chaque sommet u du graphe
   |    pour chaque arc (u, t) du graphe faire
   |      |    d[t] := min(d[t], d[u] + poids(u, t))

  retourner d
```

Algorithme de Bellman-Ford

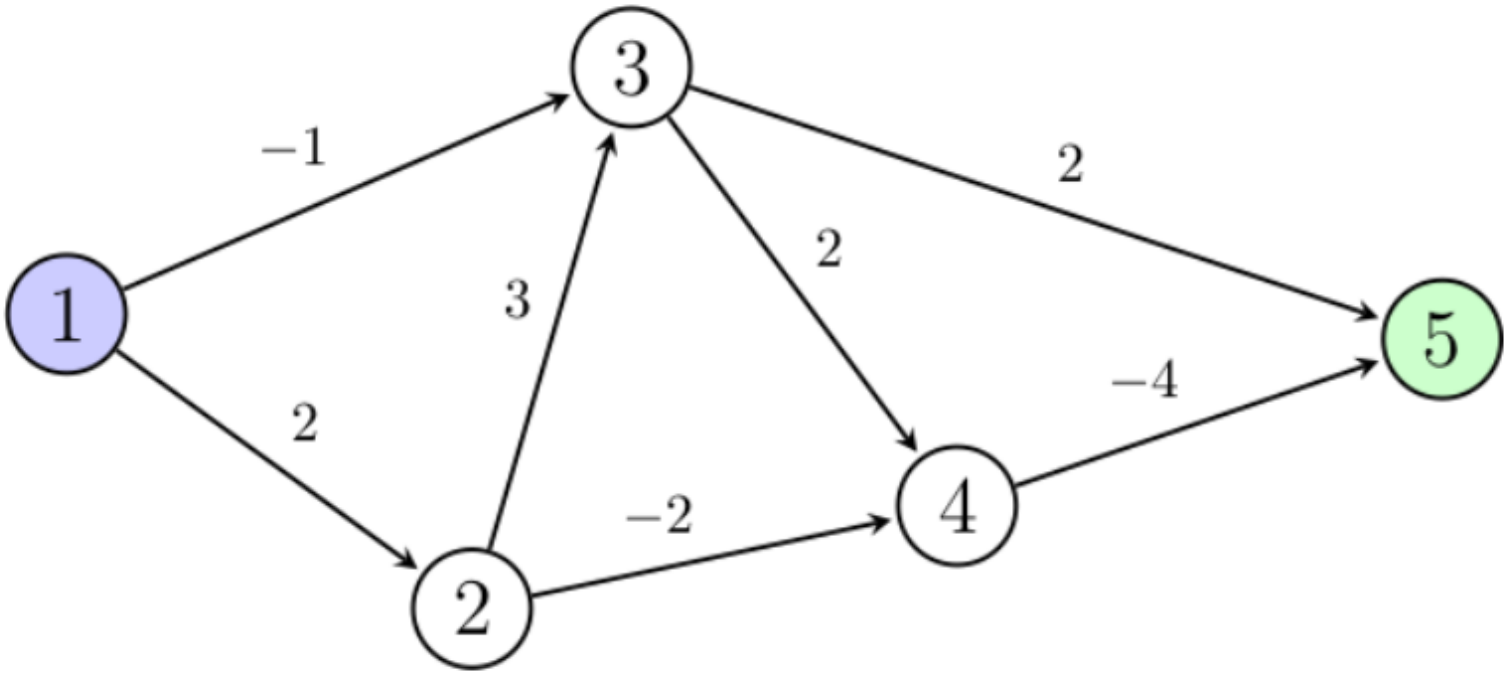
Algorithme de Bellman-Ford

- Exemple:



Algorithme de Bellman-Ford

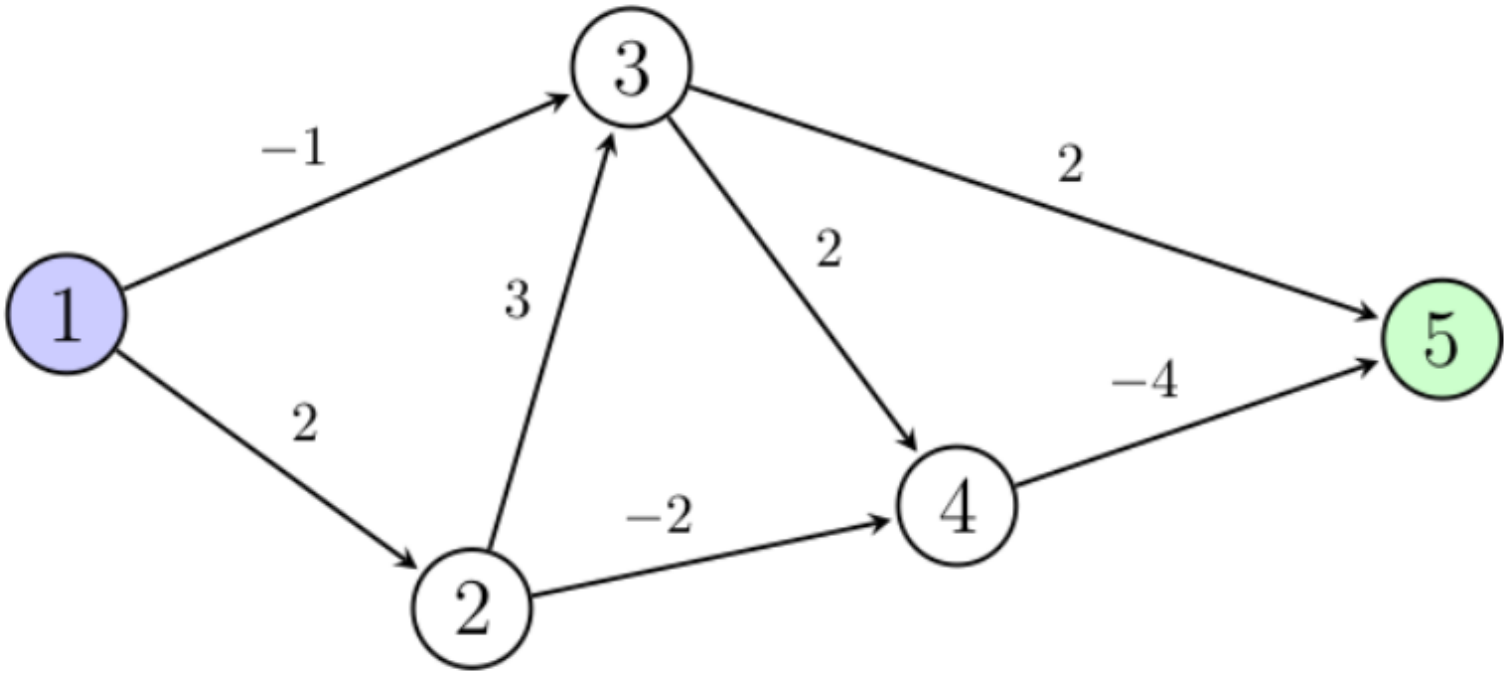
- Exemple:



- Et pour le plus long chemin ?

Algorithme de Bellman-Ford

- Exemple:



- Et pour le plus long chemin ?

→ On remplace le minimum par un maximum

Sac-à-dos

Sac-à-dos

Instance : n objets avec des poids w_i et des valeur c_i, une capacité entière W du sac-à-dos

Solutions réalisables : Un ensemble d'objets dont le poids total n'excède pas W

Critère : Maximiser la valeur d'une solution (i.e. la somme des valeurs des objets présents)

Sac-à-dos

INSTANCE : n objets avec des poids w_i et des valeur c_i, une capacité entière W du sac-à-dos

SOLUTIONS RÉALISABLES : Un ensemble d'objets dont le poids total n'excède pas W

CRITÈRE : Maximiser la valeur d'une solution (i.e. la somme des valeurs des objets présents)

- Les vidéos suivantes expliquent comment exprimer un problème de sac-à-dos à l'aide d'un problème de plus long chemin et d'utiliser la programmation dynamique pour le résoudre:

Sac-à-dos

INSTANCE : n objets avec des poids w_i et des valeur c_i, une capacité entière W du sac-à-dos

SOLUTIONS RÉALISABLES : Un ensemble d'objets dont le poids total n'excède pas W

CRITÈRE : Maximiser la valeur d'une solution (i.e. la somme des valeurs des objets présents)

- Les vidéos suivantes expliquent comment exprimer un problème de sac-à-dos à l'aide d'un problème de plus long chemin et d'utiliser la programmation dynamique pour le résoudre:
  - https://youtu.be/yUwTvJiO3dw
  - https://youtu.be/4TWkUFvpwqY
  - https://youtu.be/nBX6gz7Ssj8

Méthode générale

Méthode générale

- Trouver les sous Problèmes

Méthode générale

- Trouver les sous Problèmes
- Trouver la relation (écrire la formule de récurrence)

Méthode générale

- Trouver les sous Problèmes
- Trouver la relation (écrire la formule de récurrence)
- Trouver un ordre qui respecte la récurrence

Méthode générale

- Trouver les sous Problèmes
- Trouver la relation (écrire la formule de récurrence)
- Trouver un ordre qui respecte la récurrence
- Définir les cas de base

Méthode générale

- Trouver les sous Problèmes
- Trouver la relation (écrire la formule de récurrence)
- Trouver un ordre qui respecte la récurrence
- Définir les cas de base
- Remplir le tableau (la matrice) des valeurs

Méthode générale

- Trouver les sous Problèmes
- Trouver la relation (écrire la formule de récurrence)
- Trouver un ordre qui respecte la récurrence
- Définir les cas de base
- Remplir le tableau (la matrice) des valeurs
- Trouver la valeur optimale et retrouver le chemin pris pour l'obtenir: on a la solution optimale

Applications

Applications

Applications à de nombreux problèmes:

Applications

Applications à de nombreux problèmes:

- Plus court chemin

Applications

Applications à de nombreux problèmes:

- Plus court chemin
- Sac à dos

Applications

Applications à de nombreux problèmes:

- Plus court chemin
- Sac à dos
- Plus longue sous-sequence croissante

Applications

Applications à de nombreux problèmes:

- Plus court chemin
- Sac à dos
- Plus longue sous-sequence croissante
- Alignement de sequences et génétique

Applications

Applications à de nombreux problèmes:

- Plus court chemin
- Sac à dos
- Plus longue sous-sequence croissante
- Alignement de sequences et génétique
- Multiplication de matrices

Applications

Applications à de nombreux problèmes:

- Plus court chemin
- Sac à dos
- Plus longue sous-sequence croissante
- Alignement de sequences et génétique
- Multiplication de matrices
- Ensembles indépendants dans des arbres

Applications

Applications à de nombreux problèmes:

- Plus court chemin
- Sac à dos
- Plus longue sous-sequence croissante
- Alignement de sequences et génétique
- Multiplication de matrices
- Ensembles indépendants dans des arbres
- Etc

A vous de jouer !

- Exercices 1, 2 et 3 de la fiche de TD:
  https://moodle.caseine.org/pluginfile.php/113485/mod_resource/content/1/TD.pdf

A vous de jouer !

- Exercices 1, 2 et 3 de la fiche de TD: https://moodle.caseine.org/pluginfile.php/113485/mod_resource/content/1/TD.pdf
- Correction de l'exercice 1: https://youtu.be/2Riuf79aO-0

A vous de jouer !

- Exercices 1, 2 et 3 de la fiche de TD: https://moodle.caseine.org/pluginfile.php/113485/mod_resource/content/1/TD.pdf
- Correction de l'exercice 1: https://youtu.be/2Riuf79aO-0
- Correction de l'exercice 2: https://youtu.be/hHDHOwhMzDY

A vous de jouer !

- Exercices 1, 2 et 3 de la fiche de TD: https://moodle.caseine.org/pluginfile.php/113485/mod_resource/content/1/TD.pdf
- Correction de l'exercice 1: https://youtu.be/2Riuf79aO-0
- Correction de l'exercice 2: https://youtu.be/hHDHOwhMzDY
- Correction de l'exercice 3: https://youtu.be/HP4JoXZKCsM