4. Arbres couvrants et théorie de la complexité

25 Octobre 2023

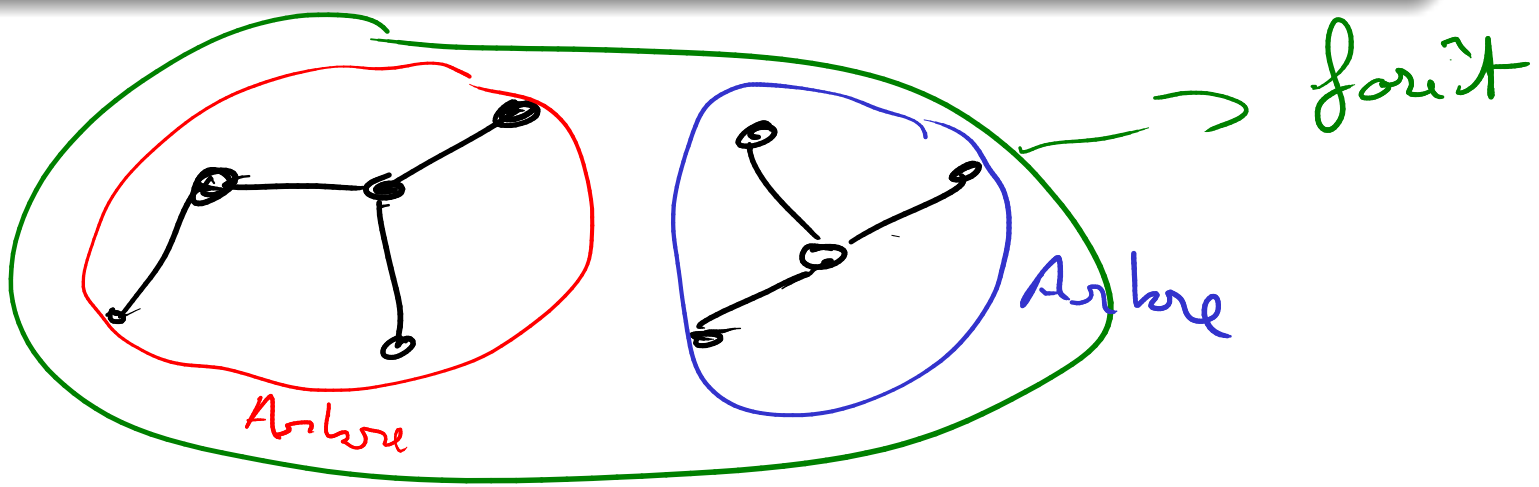
Rappel sur les arbres et forêts

$\forall (u,v) \in V$
$\exists$ u-v chemin

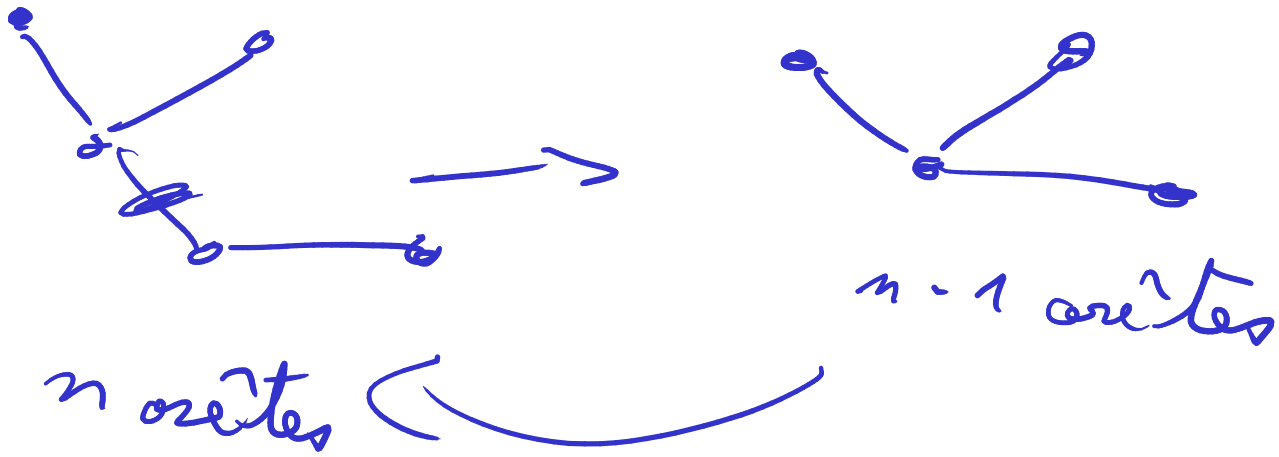
- **Forêt** : graphe (non orienté) acyclique
- **Arbre** : graphe (non orienté) acyclique connexe

Proposition.

- Chaque composante connexe d'une forêt est un arbre.
- Un arbre avec n sommets possède exactement $n-1$ arêtes.

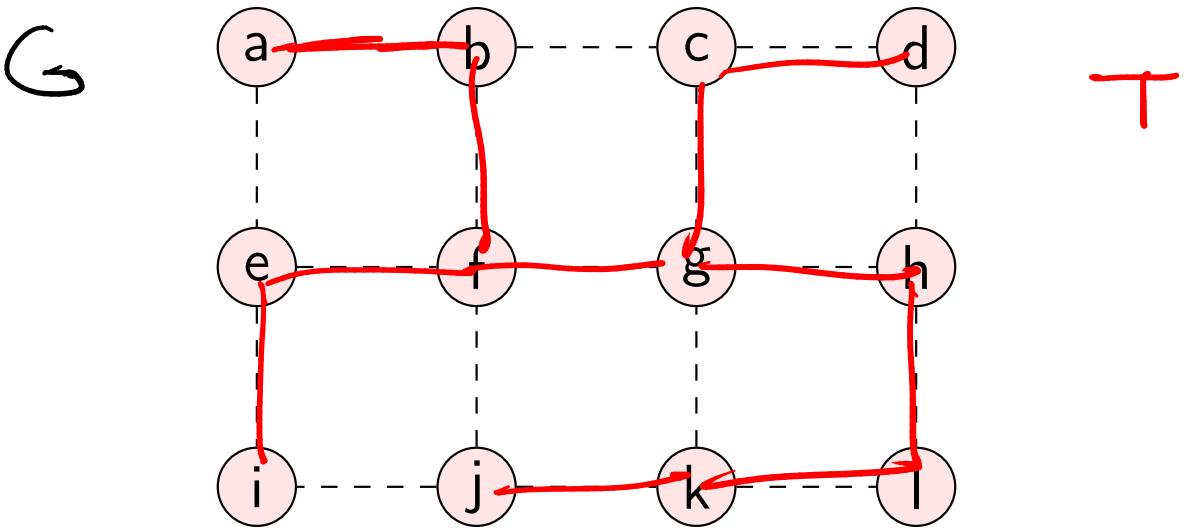


- $n = 1$: • 0 arêtes
- $n \geq 1$. Arbre avec $n+1$ sommets



Arbre couvrant

Définition. Un **arbre couvrant** dans un graphe $G = (V, E)$ est un arbre $T = (V, E')$ tel que $E' \subset E$.

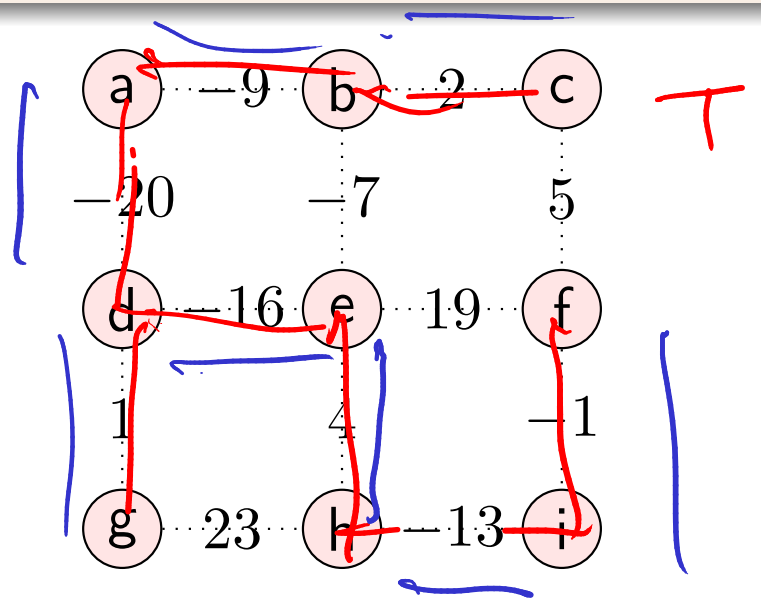


Problème d'arbre couvrant de poids minimum

Arbre couvrant de poids minimum

- **Instance**. Graphe non orienté $G = (V, E)$, poids $(c(e))_{e \in E}$.
- **Question**. Trouver un arbre couvrant $T = (V, E')$ de poids minimum

$$c(T) = \sum_{e \in E'} c(e)$$



Algorithme de Kruskal

$G = (V, E)$

1. Trier les arêtes $e \in E$ par coût croissant : $E = \{e_1, \ldots, e_m\}$, avec $c(e_i) \leq c(e_j)$ si $i < j$

Algorithme de Kruskal

$G = (V, E)$

1. Trier les arêtes $e \in E$ par coût croissant : $E = \{e_1, \ldots, e_m\}$, avec $c(e_i) \leq c(e_j)$ si $i < j$
2. $E'_0 = \emptyset$

Algorithme de Kruskal

$G = (V, E)$

1. Trier les arêtes $e \in E$ par coût croissant : $E = \{e_1, \ldots, e_m\}$, avec $c(e_i) \leq c(e_j)$ si $i < j$
2. $E'_0 = \emptyset$
3. Pour tout $i \in \{1, \ldots, m\}$
   - Si $E'_{i-1} \cup \{e_i\}$ n'a pas de cycles : $E'_i = E'_{i-1} \cup \{e_i\}$
   - Sinon, $E'_i = E'_{i-1}$

Algorithme de Kruskal

$G = (V, E)$

1. Trier les arêtes $e \in E$ par coût croissant : $E = \{e_1, \ldots, e_m\}$, avec $c(e_i) \leq c(e_j)$ si $i < j$
2. $E'_0 = \emptyset$
3. Pour tout $i \in \{1, \ldots, m\}$
   - Si $E'_{i-1} \cup \{e_i\}$ n'a pas de cycles : $E'_i = E'_{i-1} \cup \{e_i\}$
   - Sinon, $E'_i = E'_{i-1}$
4. Retourner $T = (V, E'_m)$

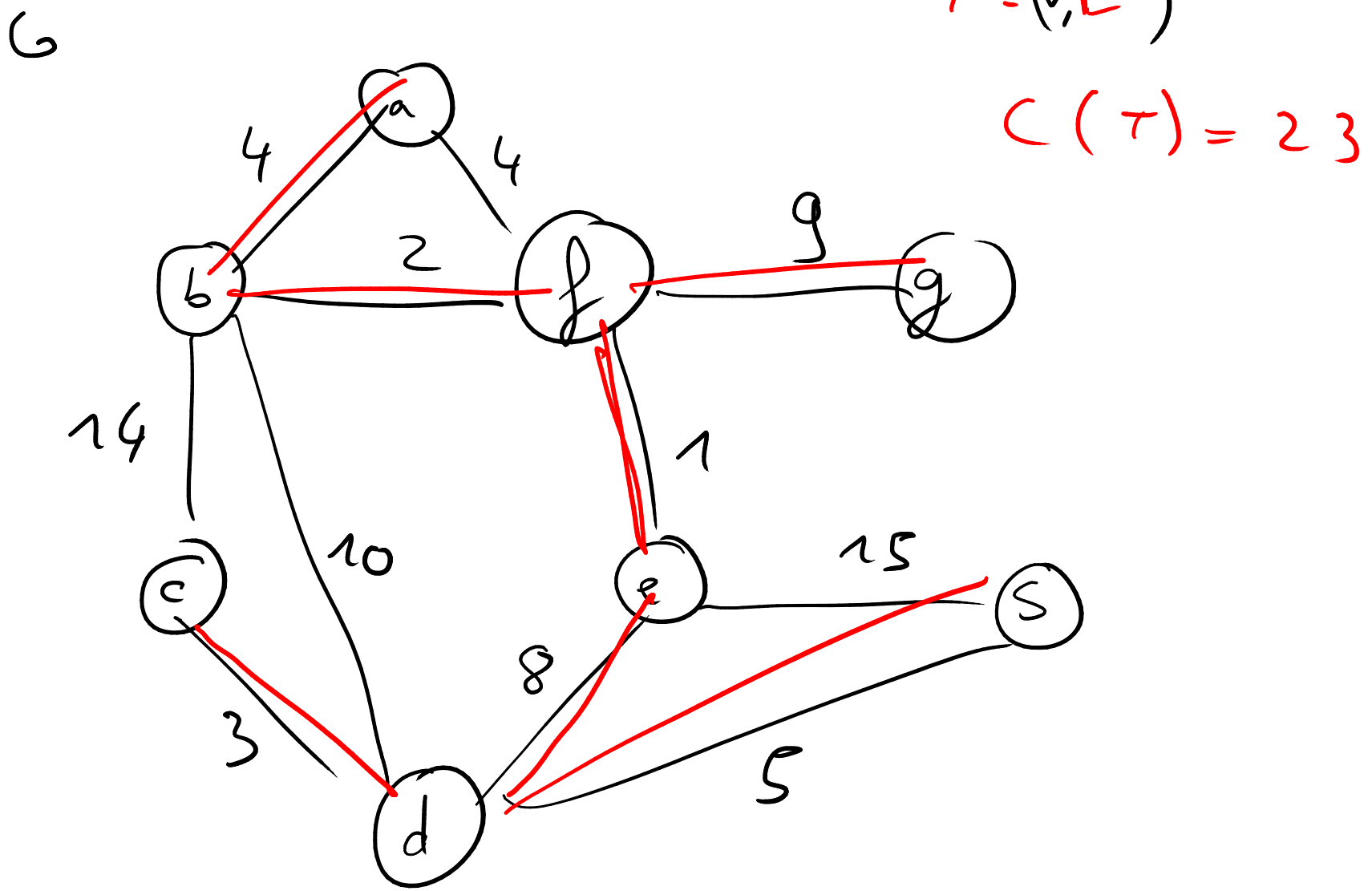
Algorithme de Kruskal

$G = (V, E)$

1. Trier les arêtes $e \in E$ par coût croissant : $E = \{e_1, \ldots, e_m\}$, avec $c(e_i) \leq c(e_j)$ si $i < j$
2. $E'_0 = \emptyset$
3. Pour tout $i \in \{1, \ldots, m\}$
   - Si $E'_{i-1} \cup \{e_i\}$ n'a pas de cycles : $E'_i = E'_{i-1} \cup \{e_i\}$
   - Sinon, $E'_i = E'_{i-1}$
4. Retourner $T = (V, E'_m)$

Proposition. Si G est connexe, l'algorithme de Kruskal retourne un arbre couvrant de poids minimum en $\mathcal{O}(|E| \cdot \log |E|)$
$(= \mathcal{O}(|E| \cdot \log |V|))$

Algorithme de Kruskal : exercice 4.2

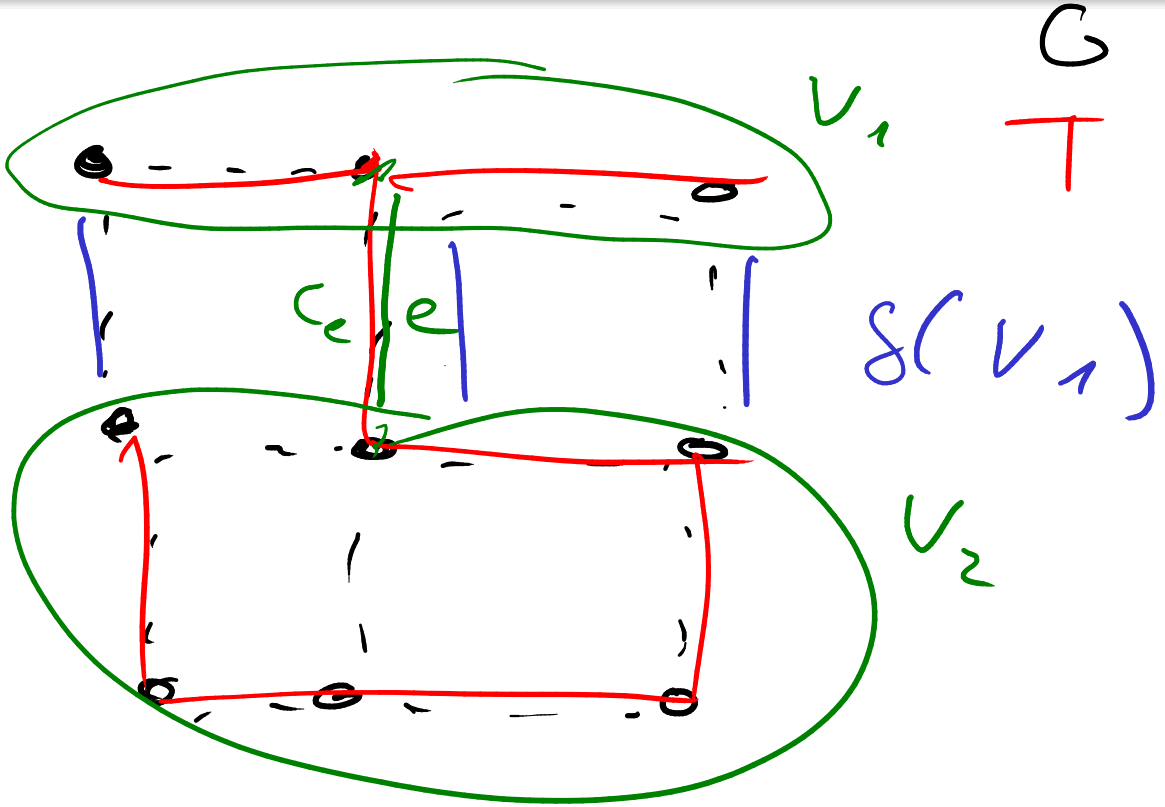


$G = (V, E)$

Proposition intermédiaire

$\delta(V_1) = \{ (u,v) \in E \mid u \in V_1, v \notin V_1 \}$

Proposition. Un arbre couvrant $T = (V, E')$ est de coût minimum ssi pour tout $e \in E'$, e est l'arête de poids minimale de la coupe $\delta(V_1)$ entre les deux composantes connexes V_1 et V_2 de $(V, E' \setminus \{e\})$.



Preuve $= (V, E')$

($\Rightarrow$) T de coût minimum.

Si $e \in E'$, $e' \in \delta(V_1)$ tq $c_{e'} < c_e$

$T' = [V, E' \setminus \{e\} \cup \{e'\}]$

$c(T') = c(T) - c_e + c_{e'} < c(T)$

Absurde

($\Leftarrow$) T_0 arbre couvrant de coût min

k arêtes différentes par rapport à T

- itérativement $T_k = T$ de coût min

Preuve de Kruskal

- A chaque itération $T_i = (V, E_i)$ est une forêt.
- A la fin T_m est un arbre couvrant
- T_m est de poids minimal d'après la proposition

Preuve de Kruskal

Quelques définitions

Problème de décision (P)

- **Instance**. données/input
- **Question**. réponse par oui ou non pour l'instance

Quelques définitions

$2^{p-1} \leq n \leq 2^p - 1$

$\log_2(n+1) \leq p$

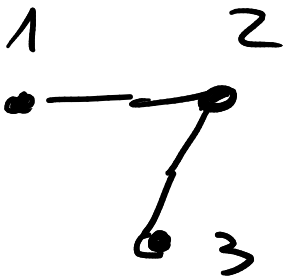
10 1000
2^{3-1} 2^3

Problème de décision (P)

- **Instance**. données/input
- **Question**. réponse par oui ou non pour l'instance

Taille d'une instance. Taille binaire $|I|_2$

- Taille d'un entier naturel n ? → $\lceil \log_2(n+1) \rceil$
- Taille d'un graphe $G = (V, E)$? → $|V|^2$
- Taille d'un graphe $G = (V, E)$ pondéré par c ?



Matrice d'adjacence : M de taille $|V| \times |V|$

$M_{ij} = 1$ si $(i, j) \in E$

0 sinon

$$M = \begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{ccc} 1 & 2 & 3 \end{array} \\ \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \end{array}$$

Quelques définitions

Problème de décision (P)

- **Instance**. données/input
- **Question**. réponse par oui ou non pour l'instance

Taille d'une instance. Taille binaire $|I|_2$

- Taille d'un entier naturel n ? $\rightarrow \lceil \log_2(n+1) \rceil$
- Taille d'un graphe $G = (V, E)$? $\rightarrow |V|^2$
- Taille d'un graphe $G = (V, E)$ pondéré par c ?
  $\rightarrow \sum_{(i,j)\in A} \lceil \log_2(c_{i,j}+1) \rceil$ $(+\ |V|^2)$

Un algorithme $\mathcal{A}$ est dit **polynomial** s'il trouve la solution de toute instance I en $\mathcal{O}(Q(|I|_2))$, avec Q un polynôme.

Exemples : Dijkstra

Algorithme de Dijkstra : $\mathcal{O}(n^2)$ $(n = |V|)$

$$|I|_2 = \sum_{i,j} \lceil \log_2(c_{ij} + 1) \rceil$$

Exemples : Dijkstra

Algorithme de Dijkstra : $\mathcal{O}(n^2)$

$$|I| = \sum_{i=1}^{n}\sum_{j=1}^{n} \underbrace{\lceil \log_2(c_{i,j}+1) \rceil}_{\geq 1} \geq n^2$$

Soit $Q(x) = x$.

On a $Q(n^2) \leq Q(|I|)$, i.e. $n^2 \leq Q(|I|)$

donc complexité$(\mathcal{A}) = \mathcal{O}(Q(|I|))$

$\Longrightarrow$ Dijkstra est polynomial.

Exemples : test de primalité

Pour tester si n est premier, on essaie de la diviser par tous les entier de 1 à $\sqrt{n}$: $\mathcal{O}(\sqrt{n})$

Exemples : test de primalité

Pour tester si n est premier, on essaie de la diviser par tous les entier de 1 à $\sqrt{n}$: $\mathcal{O}(\sqrt{n})$

$$|I| = \lceil \log_2(n+1) \rceil \simeq \log_2(n)$$

$$\sqrt{n} = \sqrt{2^{\log_2 n}} = 2^{\frac{|I|}{2}} \implies \text{non polynomial}$$

Classification des problèmes

On souhaite séparer les problèmes "faciles" et "difficiles", pour savoir quel type de méthode utiliser :

- Problème "facile" $\Longrightarrow$ algorithme exact
- Problème "difficile" $\Longrightarrow$ approximation/heuristique

Classification des problèmes

On souhaite séparer les problèmes "faciles" et "difficiles", pour savoir quel type de méthode utiliser :

- Problème "facile" $\Longrightarrow$ algorithme exact
- Problème "difficile" $\Longrightarrow$ approximation/heuristique

Facile	**Difficile**
Cycle eulérien	Cycle Hamiltonien
Plus court chemin avec $c > 0$	Plus court chemin avec $c < 0$
Arbre couvrant de poids min	Arbre de Steiner
Vehicle scheduling (flots)	Stochastic Vehicle Scheduling

Classe $\mathcal{P}$ (polynomial)

Un problème est dit **polynomial** s'il existe un algorithme polynomial le résolvant.

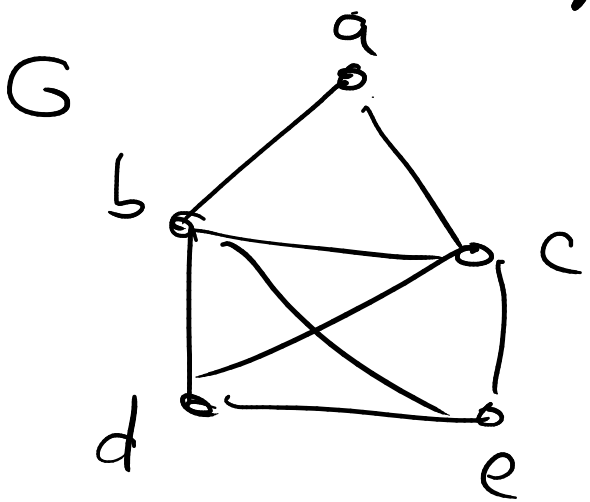
Définition. $\mathcal{P} = \{\text{problèmes polynomiaux}\}$ (problèmes "faciles").

Classe $\mathcal{NP}$ (non deterministic polynomial)

Définition. $\mathcal{NP} = \{$problèmes **de décision** pour lesquels on peut vérifier qu'une instance positive admet bien la réponse "oui" en **temps polynomial** à l'aide d'un **certificat** proposé par un devin$\}$

Cycle Hamiltonien $\in \mathcal{NP}$

Certificat = un cycle hamiltonien

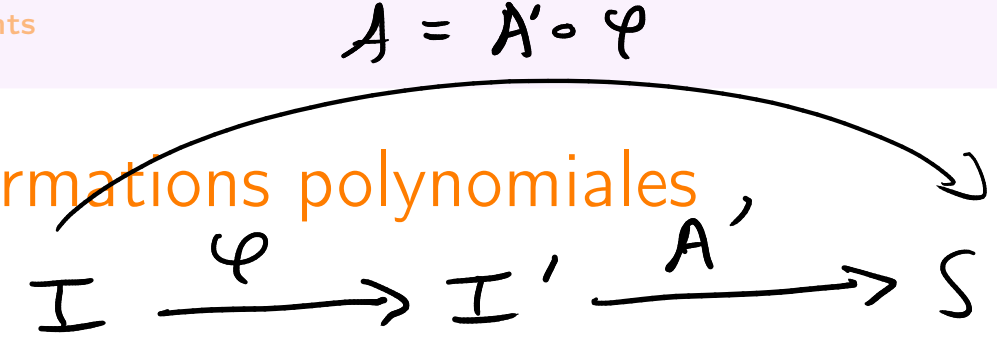


certificat : (a b d e c)

Conjecture $\mathcal{P} \neq \mathcal{NP}$

Théorème. $\mathcal{P} \subset \mathcal{NP}$

Transformations polynomiales



Soit D et D' deux problèmes de décision. On dit que D se transforme/réduit polynomialement en D' s'il existe une application φ vérifiant :

1. $\forall I$ instance de D, $\varphi(I)$ est une instance de D'
2. $\forall I$ instance de D, I et $\varphi(I)$ admettent la même réponse
3. φ est polynomial

On écrit alors $D \prec D'$.

Interprétation de $D \prec D'$: D' est au moins aussi difficile que D.

Proposition. Si D' est polynomial et $D \prec D'$, alors D est polynomial

Classe $\mathcal{NP}$-complet

Définition. Un problème de décision D est $\mathcal{NP}$-complet si :

1. $D \in \mathcal{NP}$
2. $\forall D' \in \mathcal{NP}$, $D' \prec D$ (i.e. D est au moins aussi difficile que tous les problèmes de $\mathcal{NP}$)

Comment montrer qu'un problème est $\mathcal{NP}$-complet ?

Classe $\mathcal{NP}$-complet

Définition. Un problème de décision D est $\mathcal{NP}$-complet si :

1. $D \in \mathcal{NP}$
2. $\forall D' \in \mathcal{NP}$, $D' \prec D$ (i.e. D est au moins aussi difficile que tous les problèmes de $\mathcal{NP}$)

Comment montrer qu'un problème est $\mathcal{NP}$-complet ?

Théorème. Soit D un problème de décision. Si on a :

1. $D \in \mathcal{NP}$
2. $D' \prec D$ avec D' un problème $\mathcal{NP}$-complet.

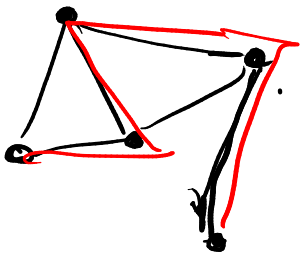
Alors D est $\mathcal{NP}$-complet.

Exemple : 3.20

Cycle Hamiltonien = CH
Chemin ——— = CHH

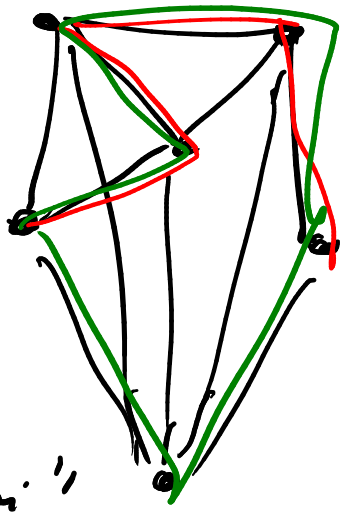
- CHH $\propto$ CH :

I



φ

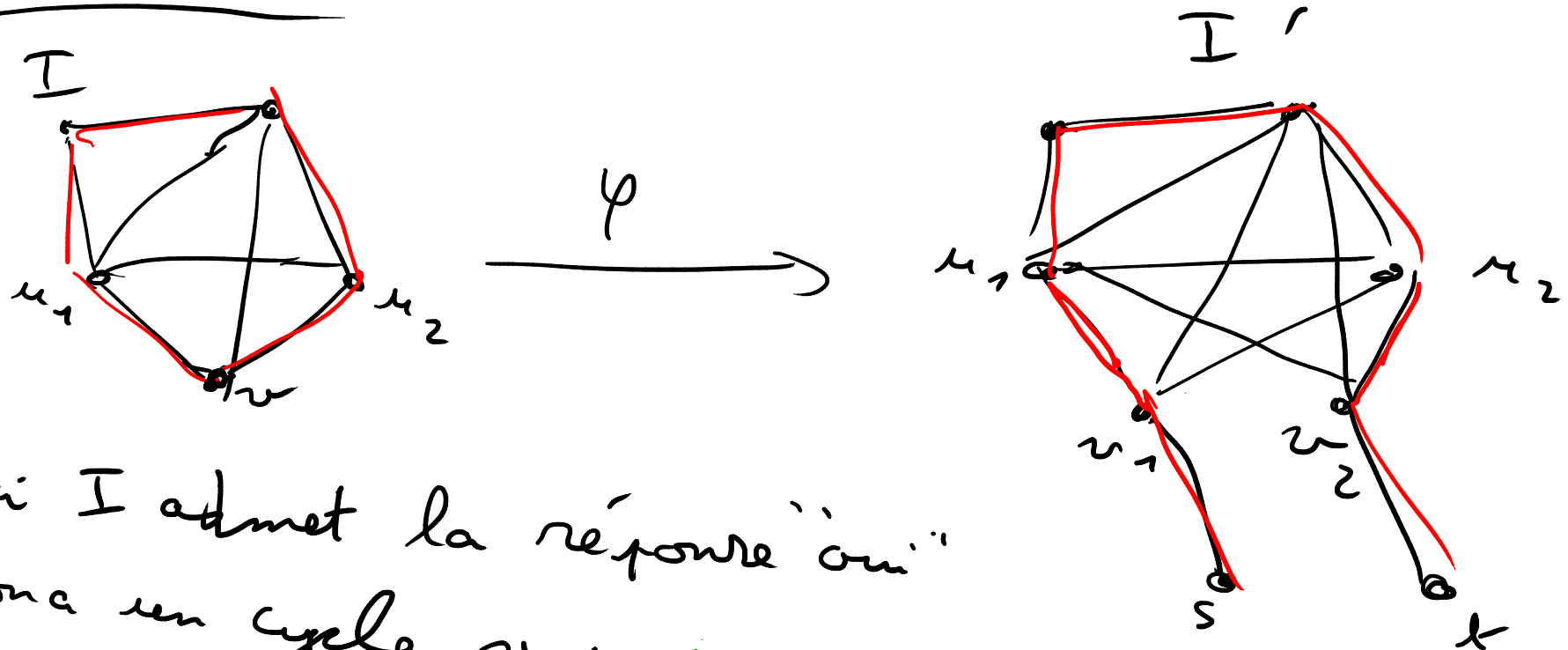
I'



① Si I admet la réponse "oui"
I' admet la réponse "oui"

② Si I' admet la réponse "oui"
I admet la réponse "oui"

CH ≤ ChH :



- Si I admet la réponse "oui"
on a un cycle $v u_1 \cdots u_2 v$
Dans I', on sélectionne $u_1 v_1$ et $u_2 v_2$
on construit le chemin $s v_1 u_1 \cdots u_2 v_2 t$
I' admet la réponse "oui".
-

Classe $\mathcal{NP}$-difficile

PO : I

: $\min_{x \in X(I)} c(x)$

PD : • I, K

• existe-t-il $x \in X(I)$ tq $c(x) \leq K$?

Définition. Un problème d'optimisation est $\mathcal{NP}$-difficile s'il est au moins aussi difficile que tous les problèmes de $\mathcal{NP}$.

Rappel : b-flot

Definition. Soit $D = (V, A)$ un graphe orienté, $\ell : A \to \mathbb{R}_+$ et $u : A \to \mathbb{R}_+$ des capacités telles que $\ell \leq u$, et $b : V \to \mathbb{R}$ une fonction telle que $\sum_{v \in V} b(v) = 0$.

Un b-**flot** est une application $f : A \to \mathbb{R}_+$ telle que :

$$\forall v \in V, \sum_{a \in \delta^+(v)} f(a) - \sum_{a \in \delta^-(v)} f(a) = b(v)$$

Et pour tout $a \in A$:

$$\ell(a) \leq f(a) \leq u(a)$$

Une **circulation** est un b-flot avec $b = 0$.

Rappel : b-flot de coût minimum

Soit une fonction de coût $c : A \to \mathbb{R}$, le **coût d'un b-flot** f est :

$$\sum_{a \in A} c(a) f(a)$$

Minimum cost flow

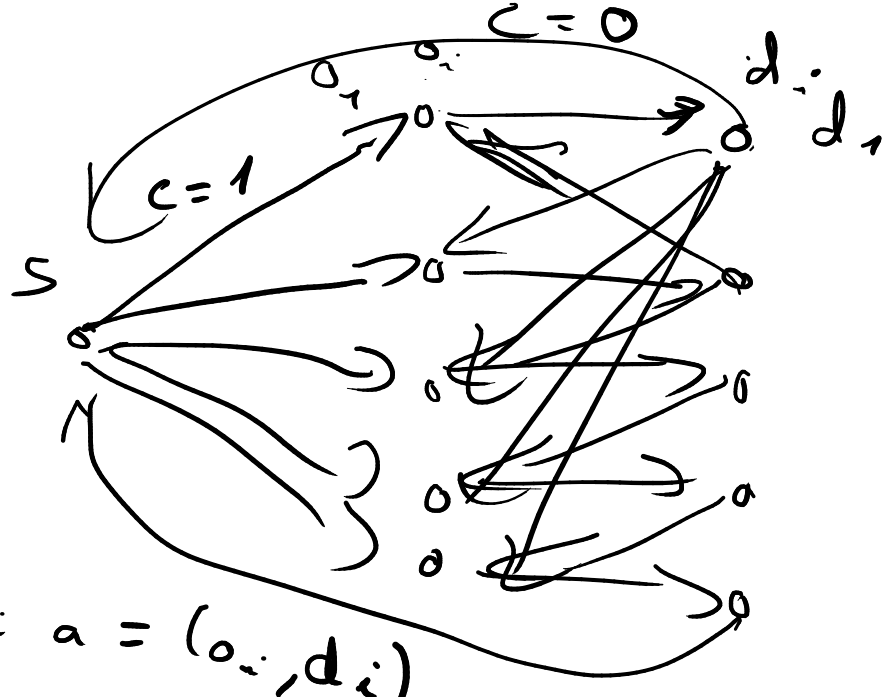
- **Instance**. Un graphe orienté $D = (V, A)$, $\ell : A \to \mathbb{R}_+$ et $u : A \to \mathbb{R}_+$ des capacités telles que $\ell \leq u$, $b : V \to \mathbb{R}$ une fonction telle que $\sum_v b(v) = 0$, et une fonction de coût $c : A \to \mathbb{R}$.
- **Question**. Un b-flot de coût minimum.

6.13/ p trajets

$i \in [p]$, départ o_i, arrivée d_i

heure de départ h_i, durée t_i

τ_{ji} : temps de trajet entre d_j et o_i



$\forall i$, $(o_i, d_i) \in A$

$\forall i$ $(s, o_i) \in A$, $(d_i, s) \in A$

$\forall i,j$, $(d_i, o_j) \in A$

si $\underbrace{h_i + t_i}_{\text{heure d'arrivée en } d_i} + \tau_{ij} \leq h_j$

Si $a = (o_i, d_i)$

$\ell(a) = 1$

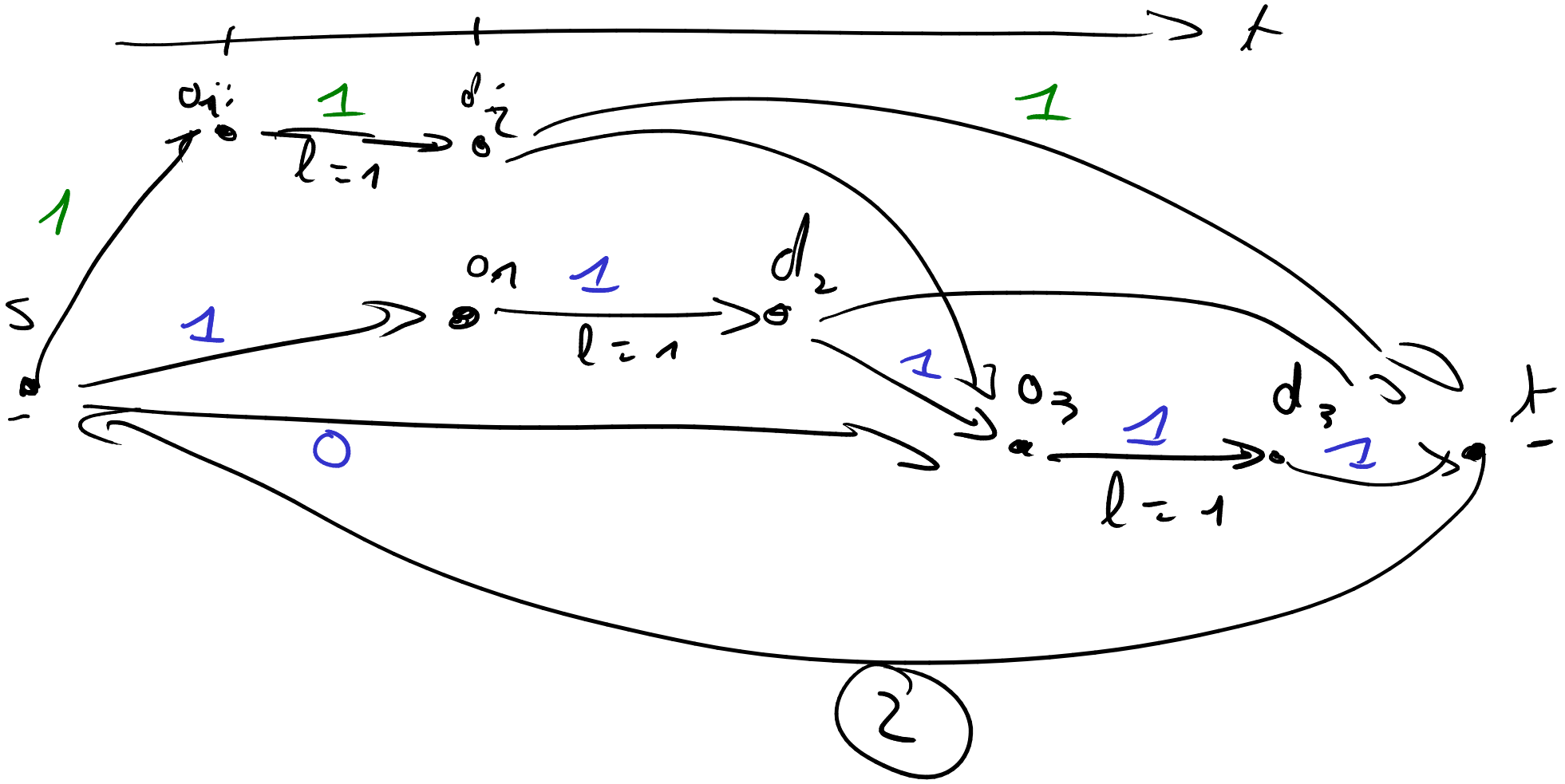
Sinon

$\ell(a) = 0$

$u = +\infty$

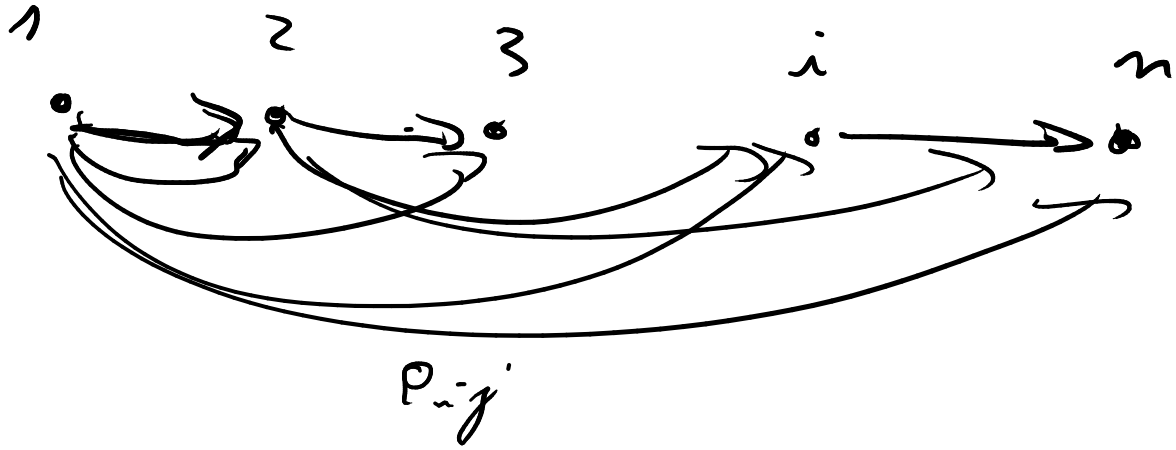
heure d'arrivée en o_j

$\forall i$, $b(o_i) = 0$, $b(d_i) = 0$


t
o_1
d_1
1
$l = 1$
1
1
s
o_2
1
d_2
1
$l = 1$
1
o_3
d_3
1
1
$l = 1$
t
0
2

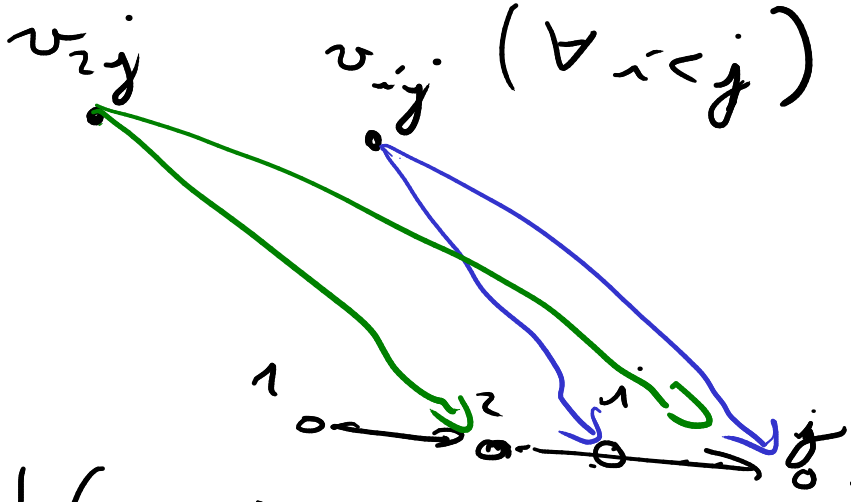
6.10/ $V = [\![1, n]\!]$ $\quad c(i,j) = p_{ij}$

$A = \{(i,j) \mid i<j\}$



$u(i,j) = d_{ij} \quad l(i,j) = 0$

$\Rightarrow$ ne marche pas



$\forall i<j,\ \begin{cases}(v_{ij},i)\in A\\(v_{ij},j)\in A\end{cases}$

$(i,i+1)\in A$

$b(v_{ij}) = d_{ij}$, $\begin{cases}c((v_{ij},i)) = -c_{ij}\\c((v_{ij},j)) = 0\end{cases}$

$u(i,i+1) = B$, $l = 0$

$$b(j) = -\sum_{i<j} d_{ij}$$

$\forall v\in V,\ \sum b(v) = 0$