

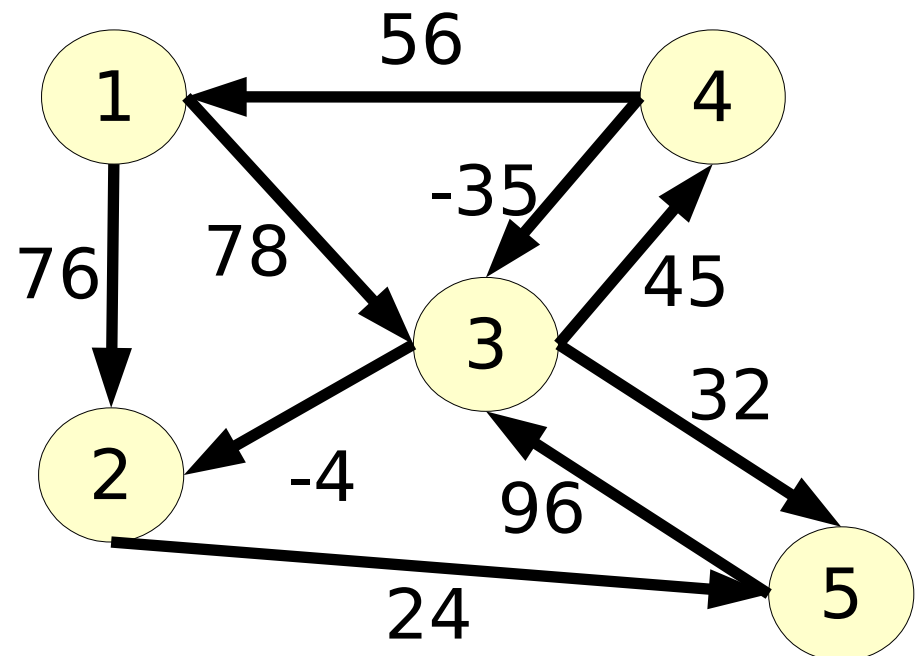
Optimisation dans les graphes

- Algorithmes de parcours d'un graphe
 - comment accéder successivement à tous les sommets d'un graphe ?
- Algorithmes de plus court chemin :
 - comment aller d'un sommet à un autre en minimisant la longueur des chemins visités ?
- Arbre de poids minimum :
 - comment déterminer les arêtes absolument nécessaires pour préserver la connexité d'un graphe.
- Flots

Problèmes de cheminement

- Donner une longueur $l(u)$ à chaque arc (ou arête) du graphe
- Plus court chemin entre 2 sommets i et j :
chemin μ t.q $\sum_{u \in \mu} l(u)$ est minimum

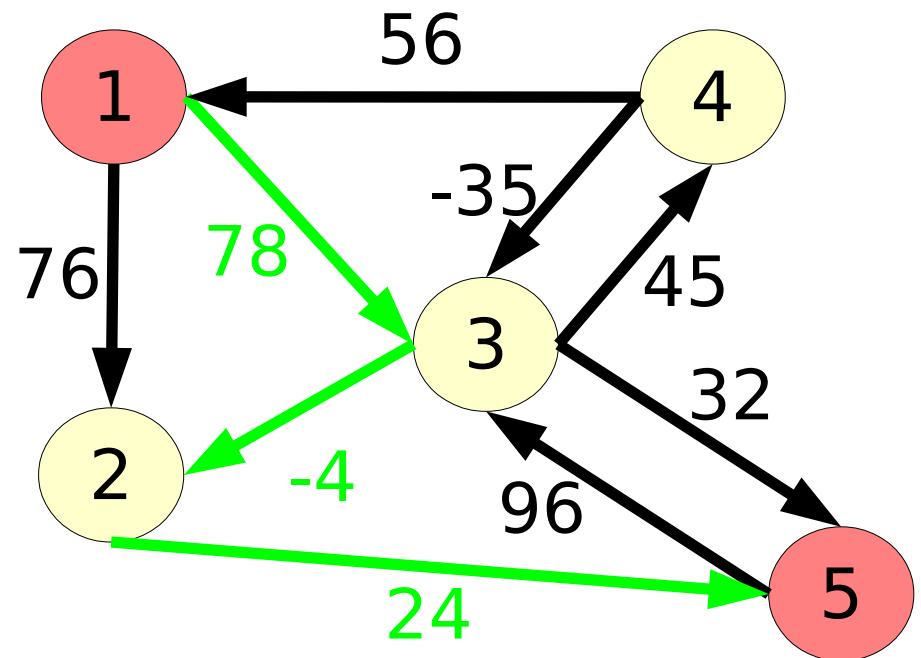
- Plus court chemin entre 1 et 5 ?



Problèmes de cheminement

- Donner une longueur $l(u)$ à chaque arc (ou arête) du graphe
- Plus court chemin entre 2 sommets i et j :
chemin μ t.q $\sum_{u \in \mu} l(u)$ est minimum

- Plus court chemin entre 1 et 5 ? **98**



Problèmes de cheminement

- Plus courts chemins de un vers tous :

- longueurs positives **Dijkstra**

- dans le plan (1 vers 1):

- Sedgewick et Vitter**

- longueurs réelles **Ford-Bellman**

Solution si aucun cycle de longueur < 0 à partir
du sommet initial

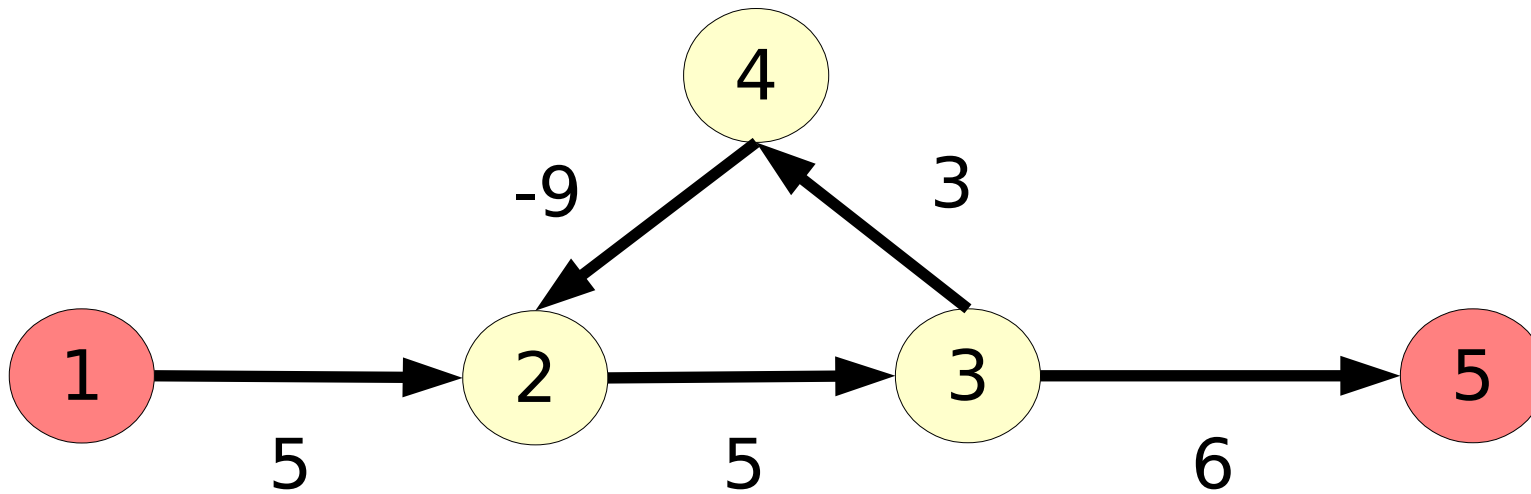
- graphes acycliques **Bellman**

- Plus courts chemins de tous vers tous **Floyd**

Solution si aucun cycle de longueur < 0 dans le
graphe

Cycles de longueur négative (cycles absorbants)

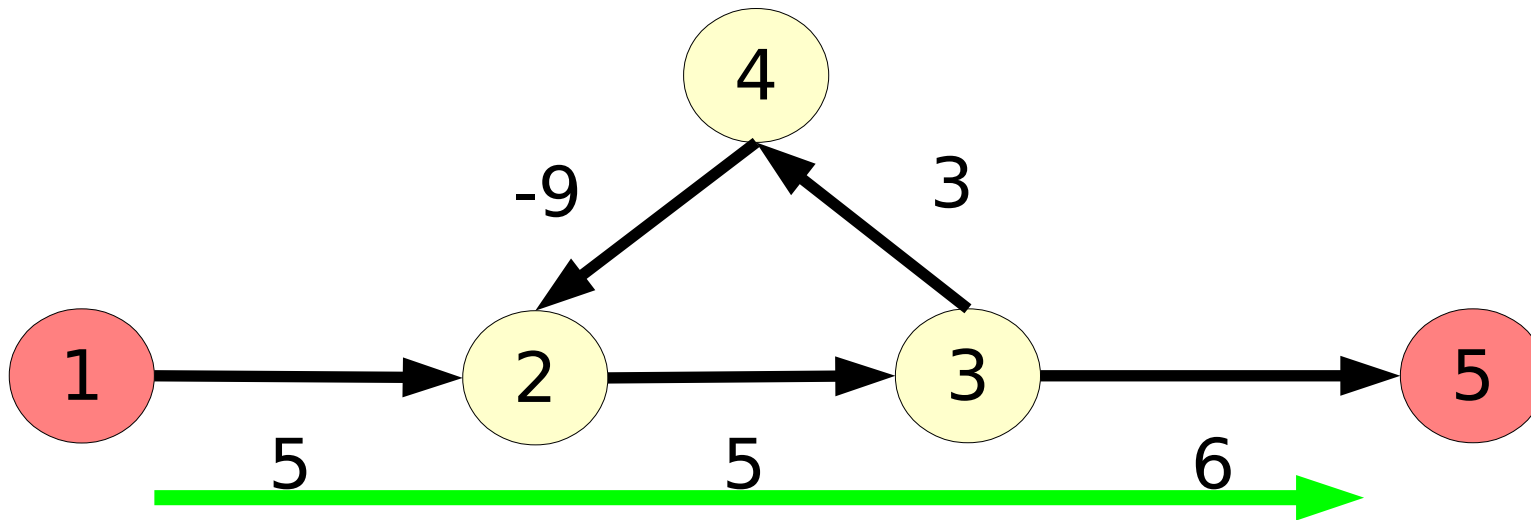
- Plus court chemins S_1 vers S_5 ?



Cycles de longueur négative

- Plus court chemins S_1 vers S_5 ?

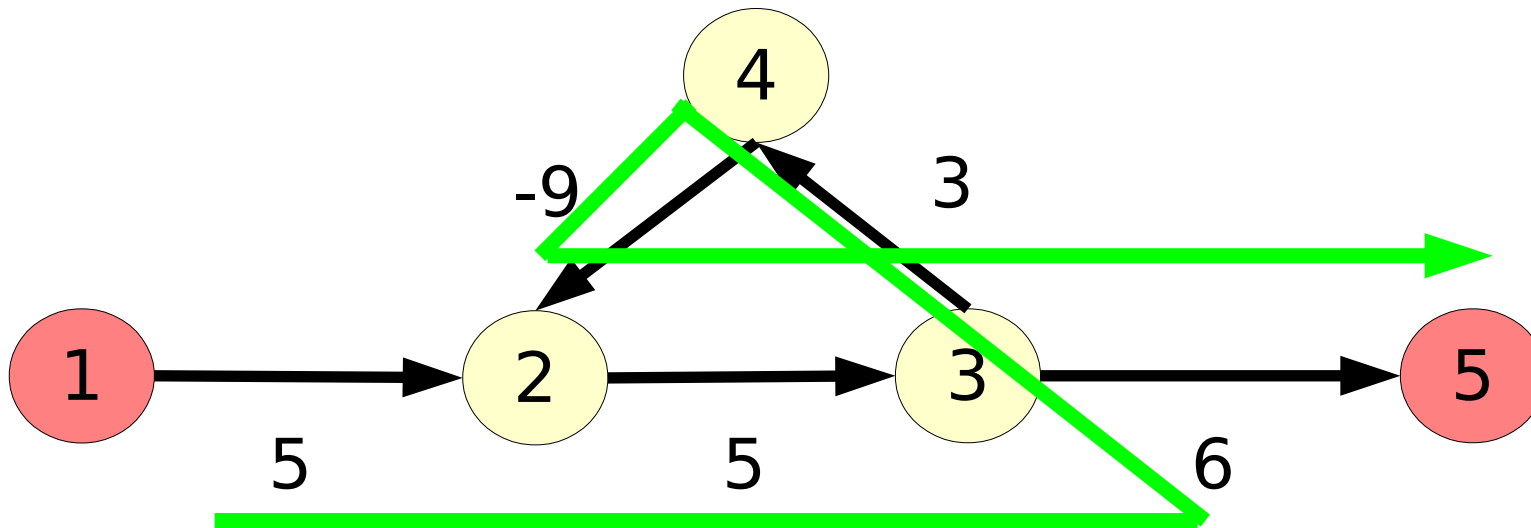
16 ?



Cycles de longueur négative

- Plus court chemins S_1 vers S_5 ?

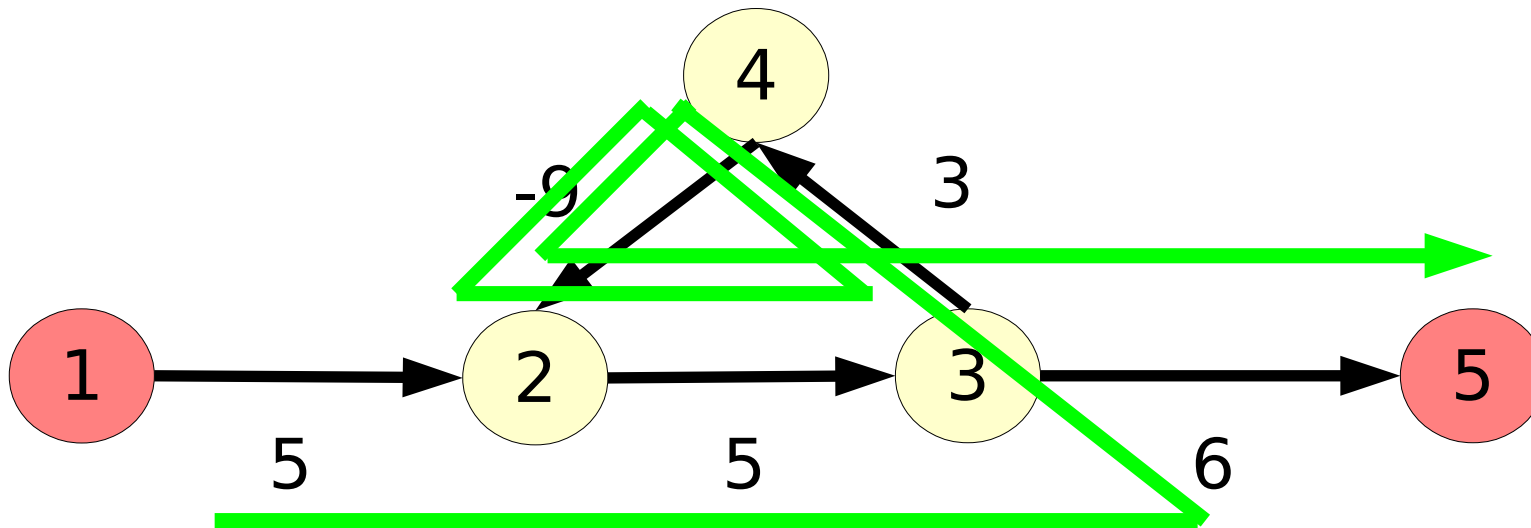
15 ?



Cycles de longueur négative

- Plus court chemins S_1 vers S_2 ?

14 ?

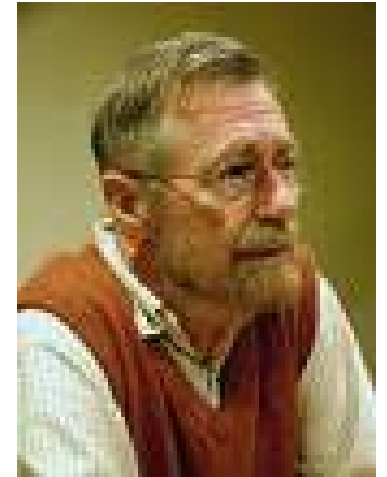


Pas de solution

- Dijkstra ne le détecte pas
- Ford-Bellman et Floyd le signalent

Algorithme de Dijkstra

- Graphe $G=(X, U)$. $l(u)>0 \forall u \in U$
- Plus court chemins de S_0 vers tous les autres sommets de X
- Marque (nombre) **D(i)** associée à tous les sommets du graphe :
 - représente la longueur du meilleur chemin de S_0 vers S à chaque étape
 - contient la longueur minimale à la fin de l'algorithme



Algorithme de Dijkstra

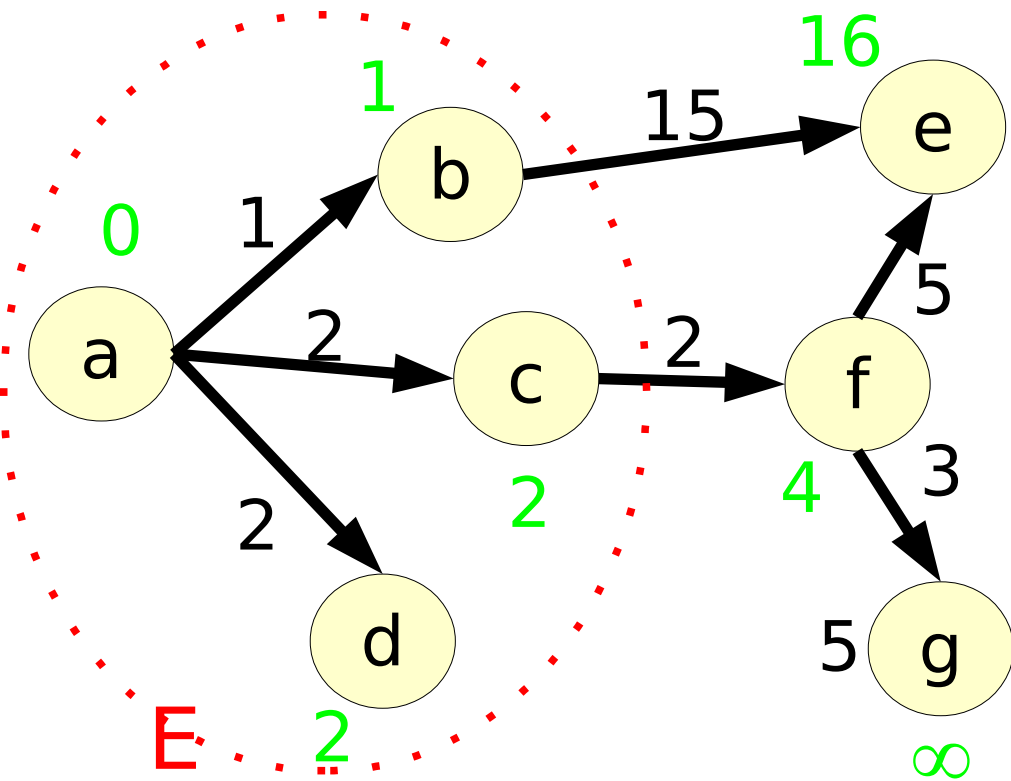
- Ensemble **E** des sommets dont la plus courte distance à S_0 est déjà connue
 - un sommet est ajouté dans E à chaque étape
 - il y a $|X|$ étapes
 - pour chaque sommet S, $D(S)$ est éventuellement amélioré à une étape donnée

$D(S_3)$ peut il passer de 18 à 23 ?

$E = \{S_0, S_2, S_5\}$

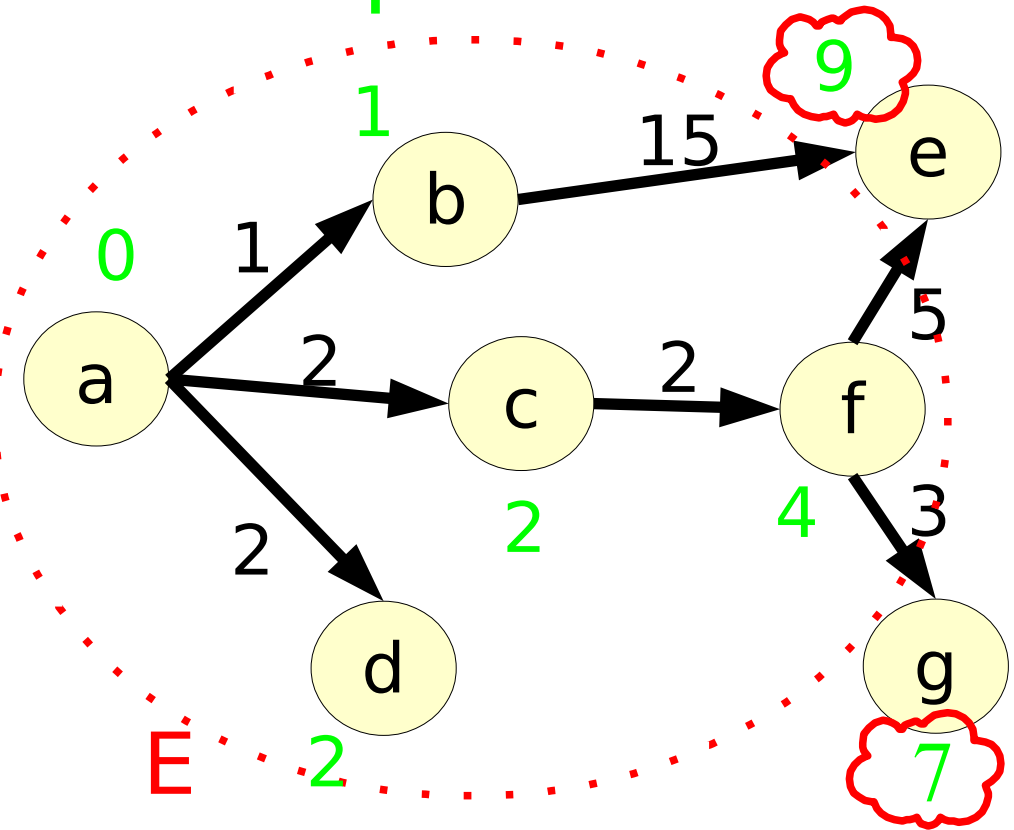
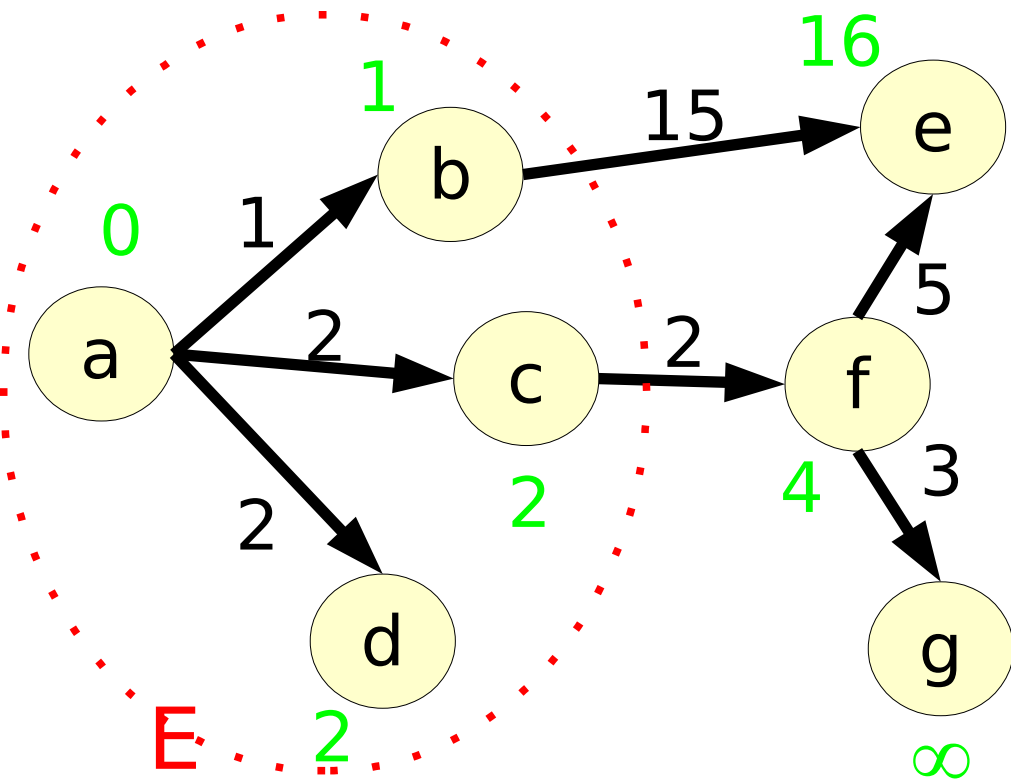
D					
S_0	S_1	S_2	S_3	S_4	S_5
0	45	7	18	25	6

Dijkstra - Une étape



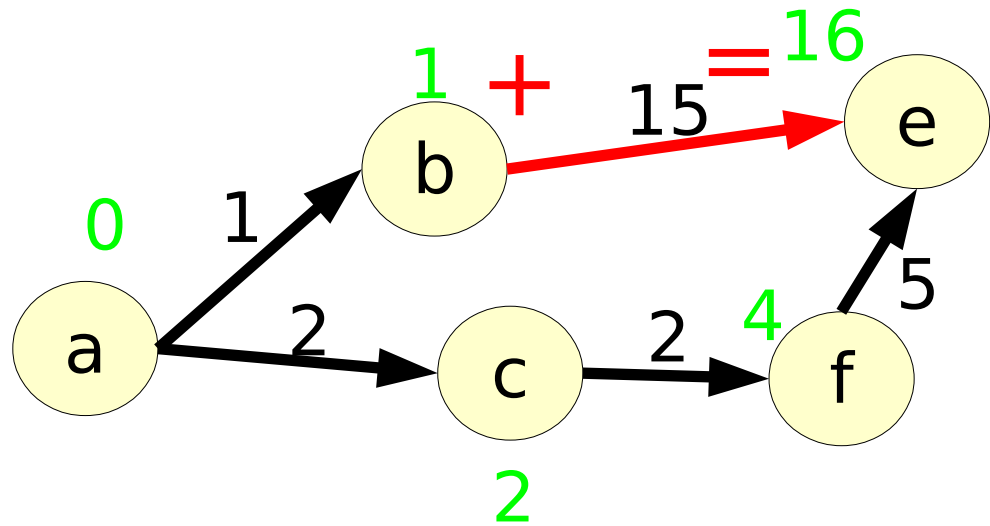
- f est le sommet choisi
 $f \notin E, D(f) = 4$
- e et g sont adjacents à f
mise à jour ?

Dijkstra - Une étape

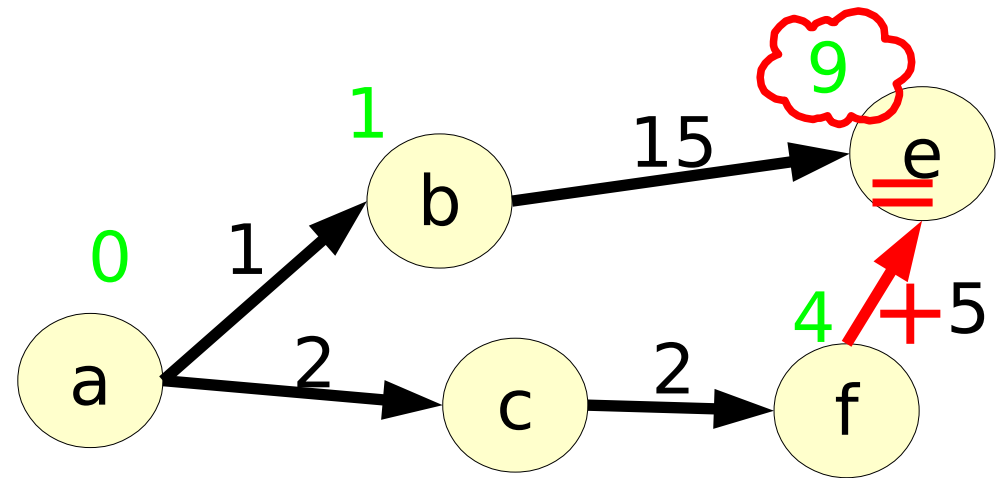


- f inclus dans E
- e et g effectivement mis à jour

Dijkstra - mise à jour



aller de a à b + arc $b \rightarrow e$



aller de a à f + arc $f \rightarrow e$

Dijkstra - algorithme

Dijkstra(Graphe G, Sommet S_0)(Tableau D)

1-Initialisation

Ensemble $E \leftarrow \{\}$

Tableau D, $D[S_0] \leftarrow 0$ et $\forall S_i \neq S_0, D[i] \leftarrow \infty$

2-Itération courante

choisir $S_j \notin E$ t.q. $D[j]$ minimum

$E \leftarrow E \cup \{S_j\}$

pour chaque k t.q. $S_k \notin E$

Si $D[j] + l_{j \rightarrow k} < D[k]$

$D[k] \leftarrow D[j] + l_{j \rightarrow k}$

Donne les longueurs
mais pas les
chemins !

Dijkstra - calcul des chemins

1-Initialisation

$$\forall S_i, C[i] \leftarrow S_0$$

2-Itération courante

...

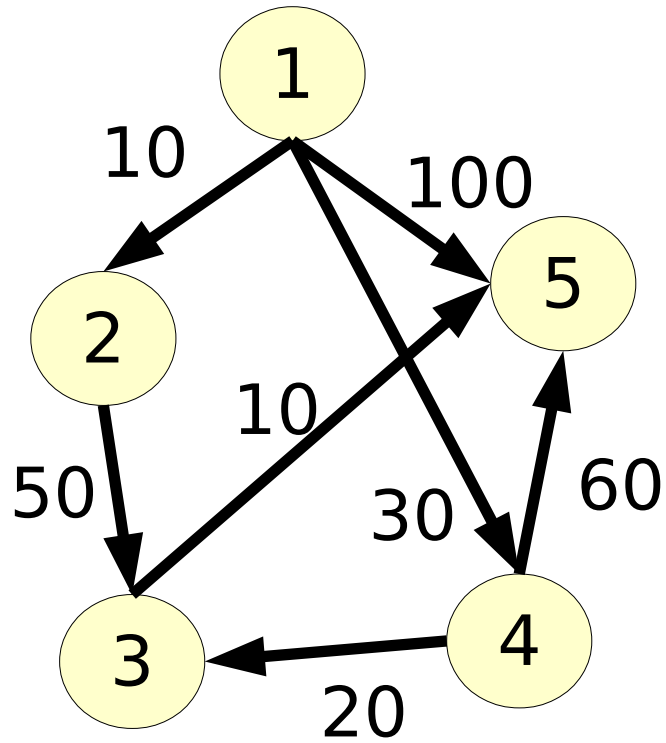
$$\text{Si } D[j] + l_{j \rightarrow k} < D[k]$$

$$C[k] \leftarrow S_j$$

affChemin(Sommets S_0 S_i ,
Tableau C)
Si $S_i \neq S_0$
 affChemin(S_0 , $C[S_i]$, C)
afficher ($\rightarrow S_i$)

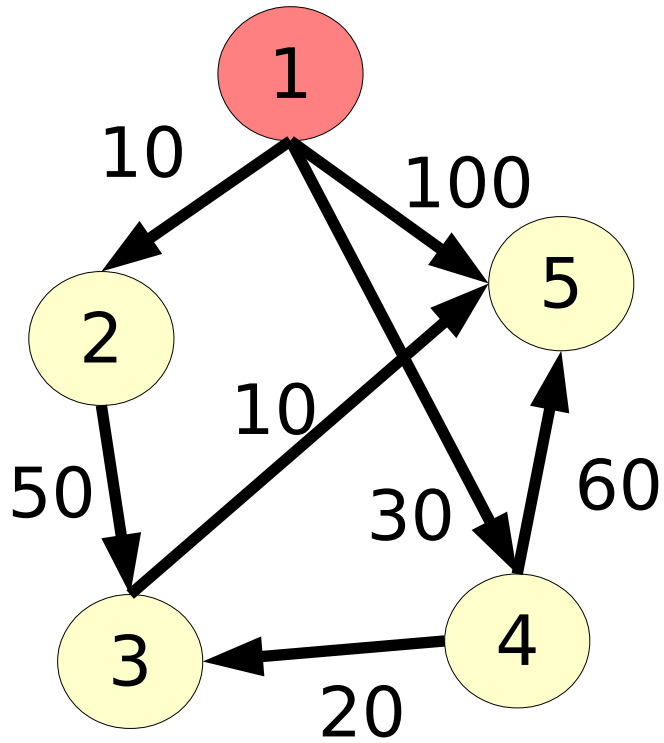
$C[i]$ est le prédécesseur
de i sur le chemin $S_0 \rightarrow S_i$

Dijkstra exemple



		D				
#	Choisi	D[1]	D[2]	D[3]	D[4]	D[5]
Init	-	0	∞	∞	∞	∞

Dijkstra exemple

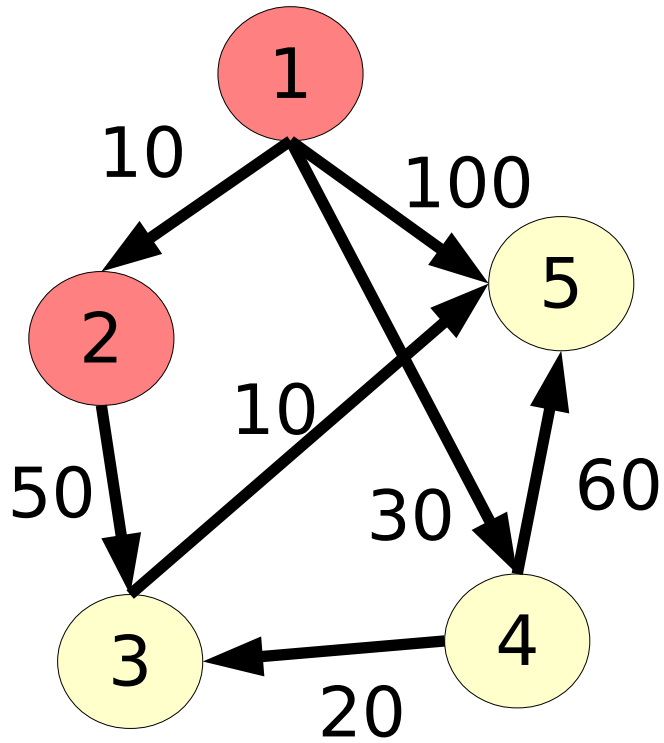


E

		D				
#	Choisi	D[1]	D[2]	D[3]	D[4]	D[5]
Init	-	0	∞	∞	∞	∞
1	S_1	0	10	∞	30	100

MAJ

Dijkstra exemple

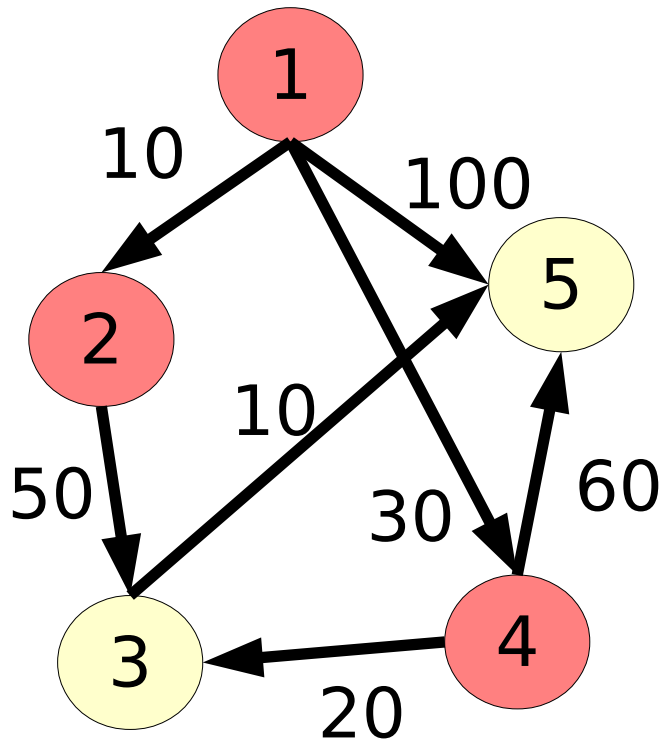


E

		D				
#	Choisi	D[1]	D[2]	D[3]	D[4]	D[5]
Init	-	0	∞	∞	∞	∞
1	S_1	0	10	∞	30	100
2	S_2	0	10	60	30	100

MAJ

Dijkstra exemple

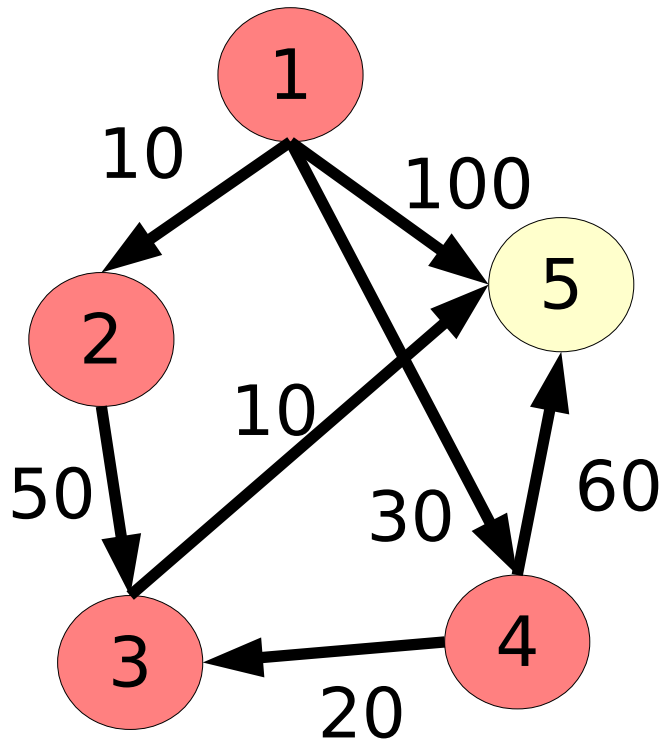


E

		D				
#	Choisi	D[1]	D[2]	D[3]	D[4]	D[5]
Init	-	0	∞	∞	∞	∞
1	S_1	0	10	∞	30	100
2	S_2	0	10	60	30	100
3	S_4	0	10	50	30	90

MAJ

Dijkstra exemple

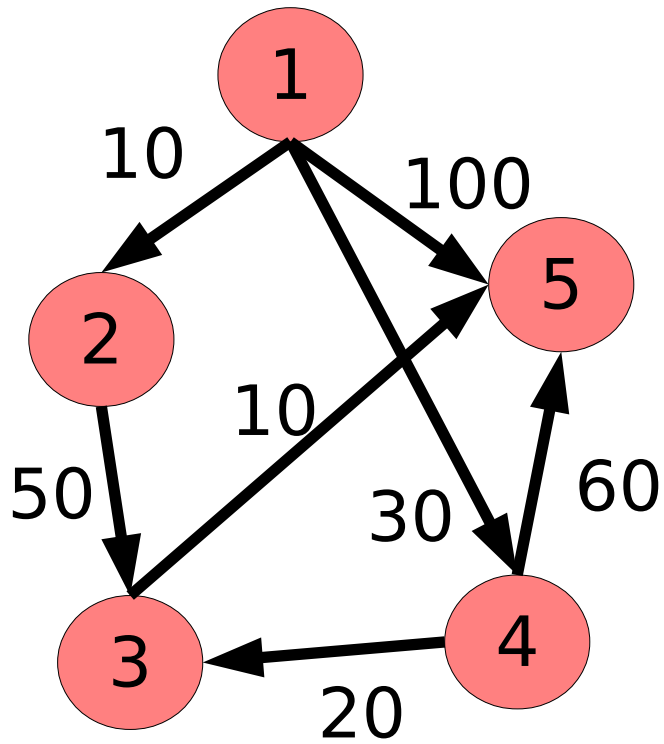


E

		D				
#	Choisi	D[1]	D[2]	D[3]	D[4]	D[5]
Init	-	0	∞	∞	∞	∞
1	S_1	0	10	∞	30	100
2	S_2	0	10	60	30	100
3	S_4	0	10	50	30	90
4	S_3	0	10	50	30	60

MAJ

Dijkstra exemple



E

		D				
#	Choisi	D[1]	D[2]	D[3]	D[4]	D[5]
Init	-	0	∞	∞	∞	∞
1	S_1	0	10	∞	30	100
2	S_2	0	10	60	30	100
3	S_4	0	10	50	30	90
4	S_3	0	10	50	30	60
5	S_5	0	10	50	30	60

MAJ

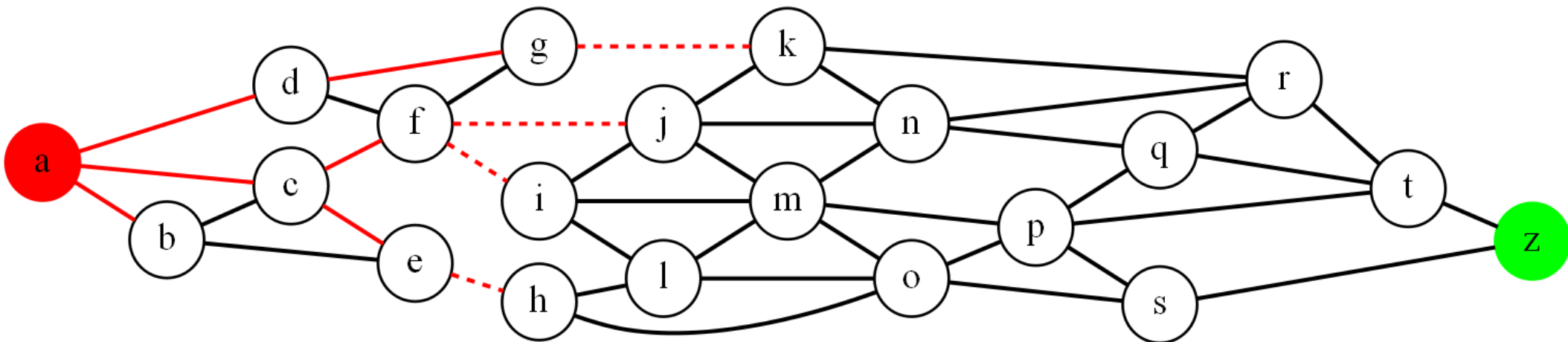
Adapter Dijkstra paire de points dans le plan

- Adaptation : plus court chemin d'un point s à un point t
Géographie, routage GPS
- Pas besoin de voir tous les points
- Arrêt de l'algorithme lorsque t est trouvé



Algorithme Sedgwick et Vitter paire de points dans le plan

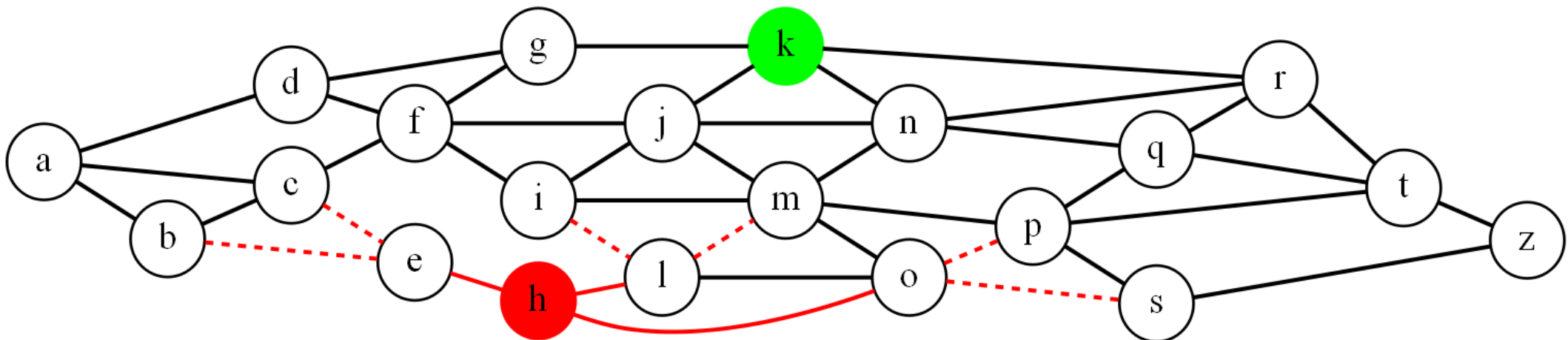
- Adaptation : plus court chemin d'un point s à un point t
Géographie, routage GPS
- Pas besoin de voir tous les points
- Aller dans la bonne direction
 - Aller de a à z ... ok



Algorithme Sedgwick et Vitter paire de points dans le plan

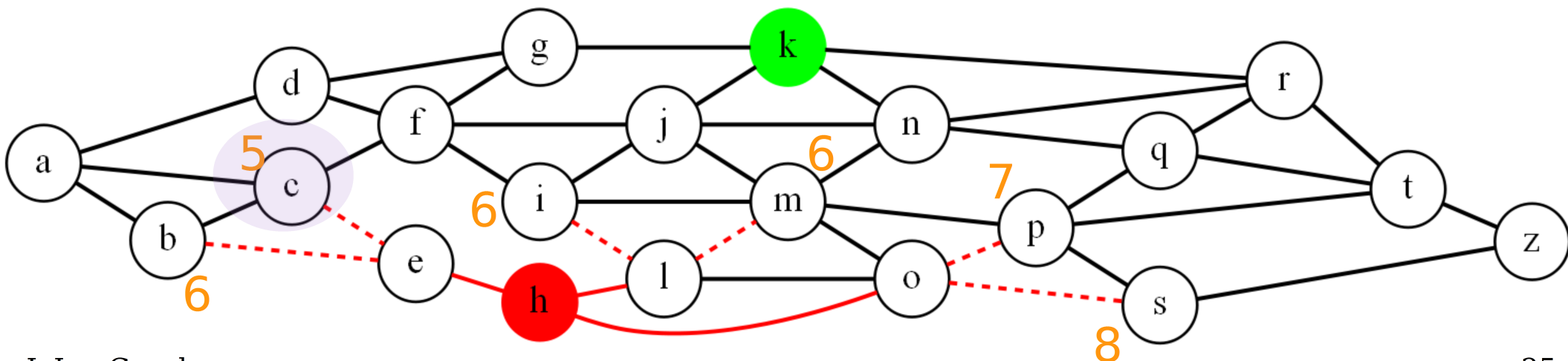
- Adaptation : plus court chemin d'un point s à un point t
Géographie, routage GPS
- Pas besoin de voir tous les points
- Aller dans la bonne direction

- Aller de h à k ? arbre très large !



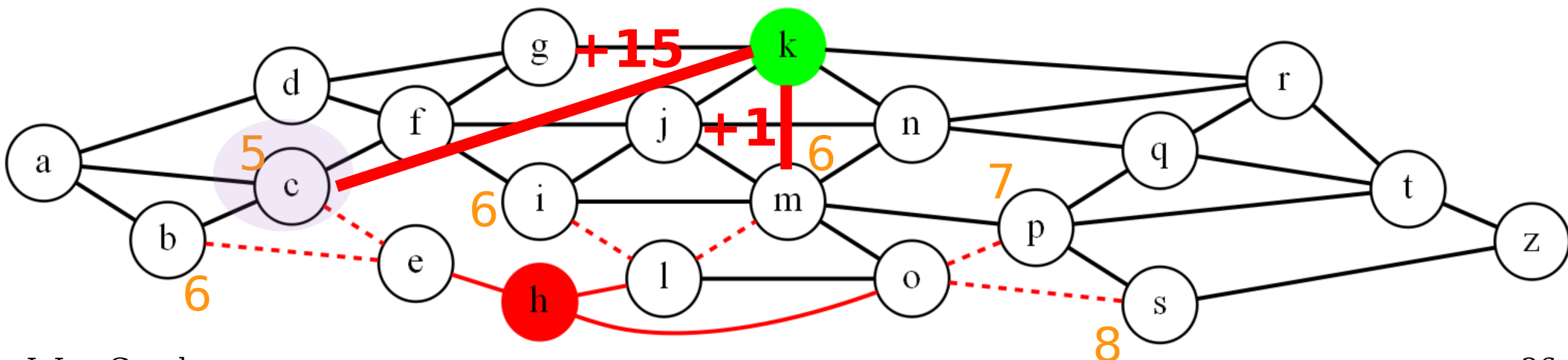
Algorithme Sedgwick et Vitter paire de points dans le plan

- Normalement, avec Dijkstra, le choix du prochain sommet fixé dépend de $D[s]$. Ici sommet **e** choisi
- Pas de choix lie à la connaissance géographique !



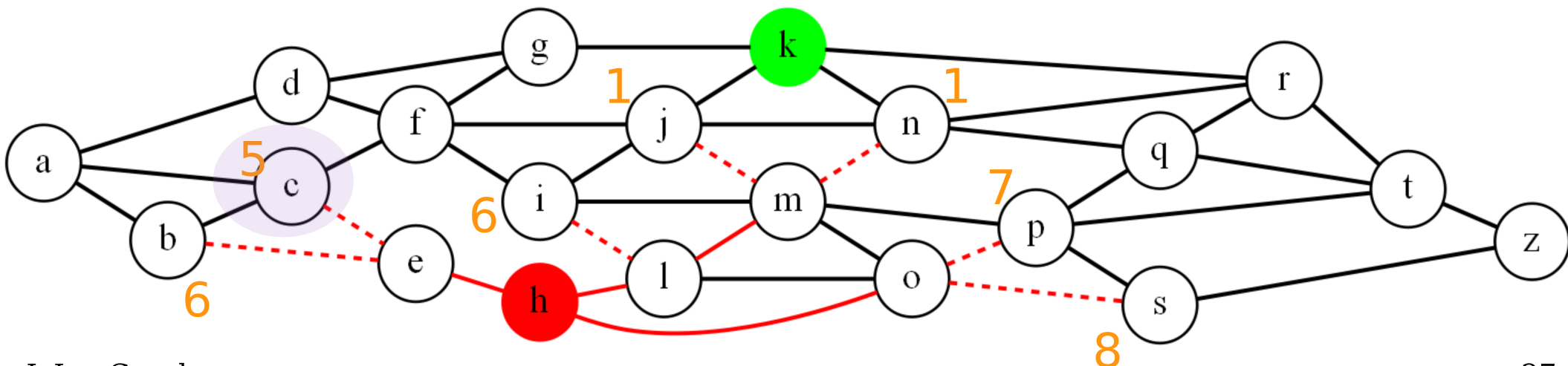
Algorithme Sedgwick et Vitter paire de points dans le plan

- Normalement, avec Dijkstra, le choix du prochain sommet fixé dépend de $D[s]$. Ici sommet **e** choisi
- Pas de choix lie à la connaissance géographique !
- Ajout dans D de la distance **restante estimée**
 - $D[m] = 6 + 1$
 - $D[c] = 5 + 15$



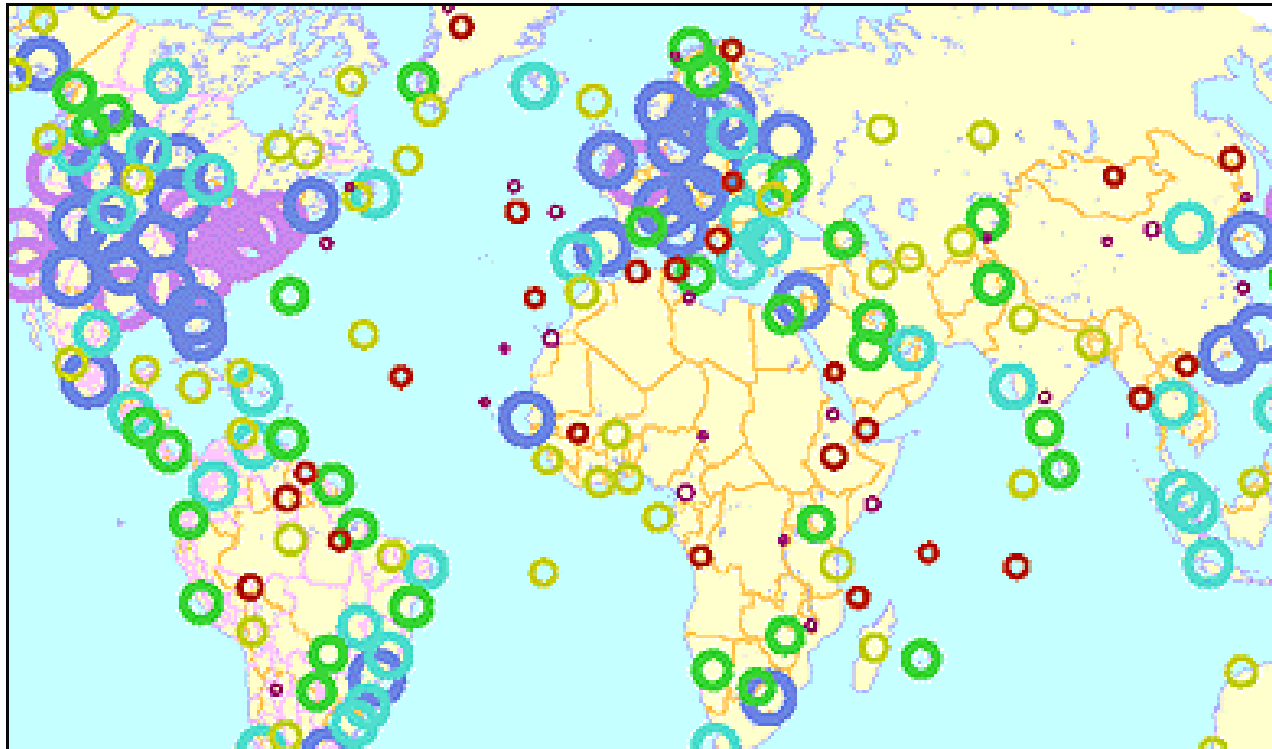
Algorithme Sedgwick et Vitter paire de points dans le plan

- Normalement, avec Dijkstra, le choix du prochain sommet fixé dépend de $D[s]$. Ici sommet **e** choisi
- Pas de choix lie à la connaissance géographique !
- Ajout dans D de la distance **restante estimée**
 - $D[m] = 6 + 1$
 - On progresse dans la bonne direction



Algorithme de Dijkstra

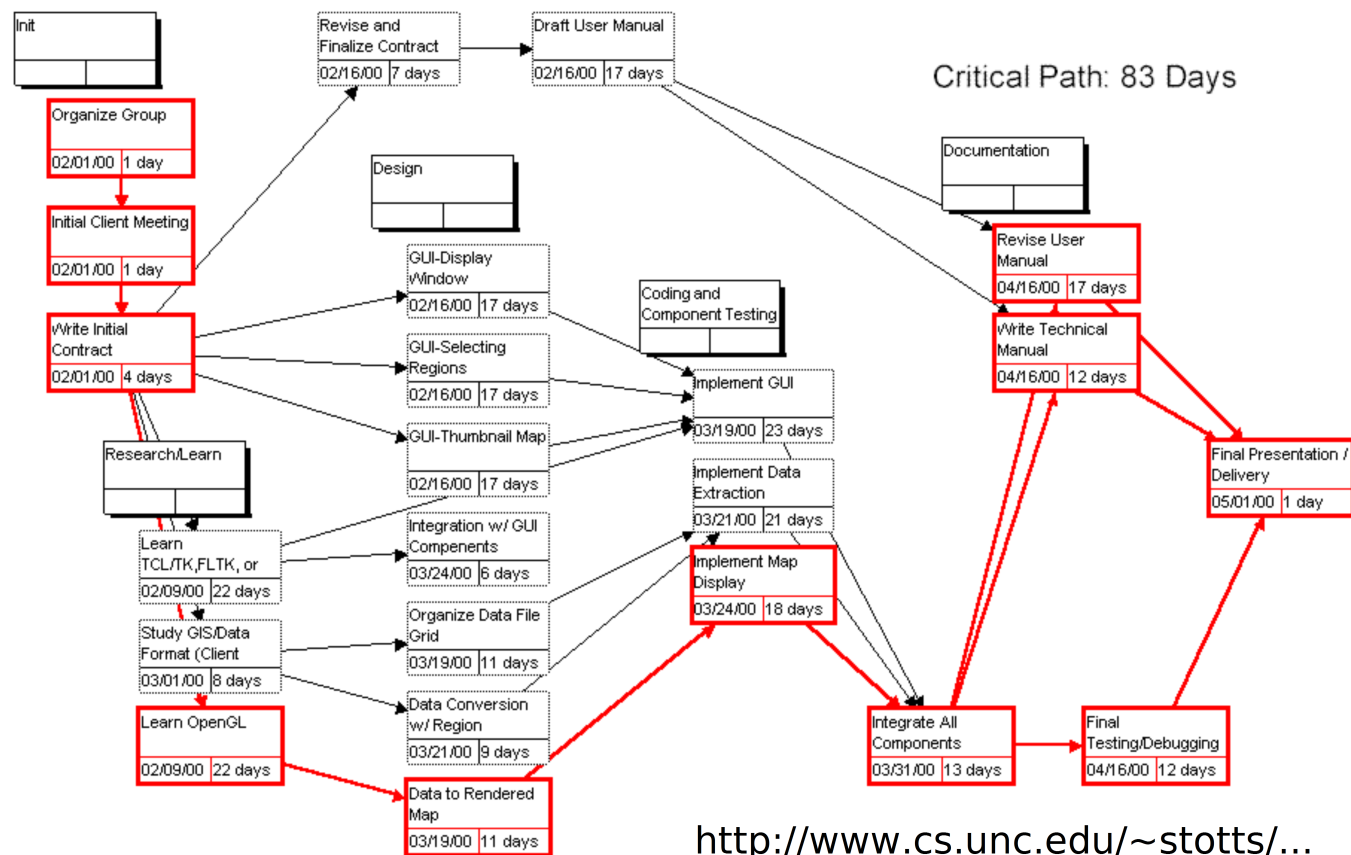
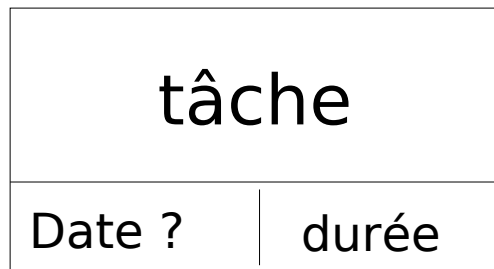
- Application : *Open shortest Path First*
Topologie de réseau connue par tous les routeurs
- Plus court chemin de machine i à machine j
(en nombre de routeurs)
- Equilibrage de la charge (routes engorgées plus longues)



Algorithme de Dijkstra

- Application : *PERT*
ensemble de tâches avec contraintes de
précédence et durée d'exécution
Dijkstra : dates au plus tôt

- Critical path :
plus long
chemin



Algorithme de Dijkstra

Complexité

- Complexité : $N * O(N + A/N)$

pour $G=(X, U)$

$N=|X|$

$A=|U|$

$$O(N^2 + A)$$

Mise à jour successeurs

Calcul du minimum

Etapes

Itération courante Dijkstra

choisir $S_j \notin E$ t.q. $D[j]$ minimum

$E \leftarrow E \cup \{S_j\}$

pour chaque k t.q. $S_k \notin E$

$$D[k] = \min(D[k], [D[j] + l_{j \rightarrow k}])$$

Algorithme de Dijkstra

Complexité

- Complexité : $N * O(N + A/N)$

pour $G=(X, U)$

$N=|X|$

$A=|U|$

$$O(N^2 + A)$$

Mise à jour successeurs

Calcul du minimum

Etapes

Itération courante Dijkstra

choisir $S_j \notin E$ t.q. $D[j]$ minimum

$E \leftarrow E \cup \{S_j\}$

pour chaque k t.q. $S_k \notin E$

$$D[k] = \min(D[k], [D[j] + l_{j \rightarrow k}])$$

Tas de Fibonacci

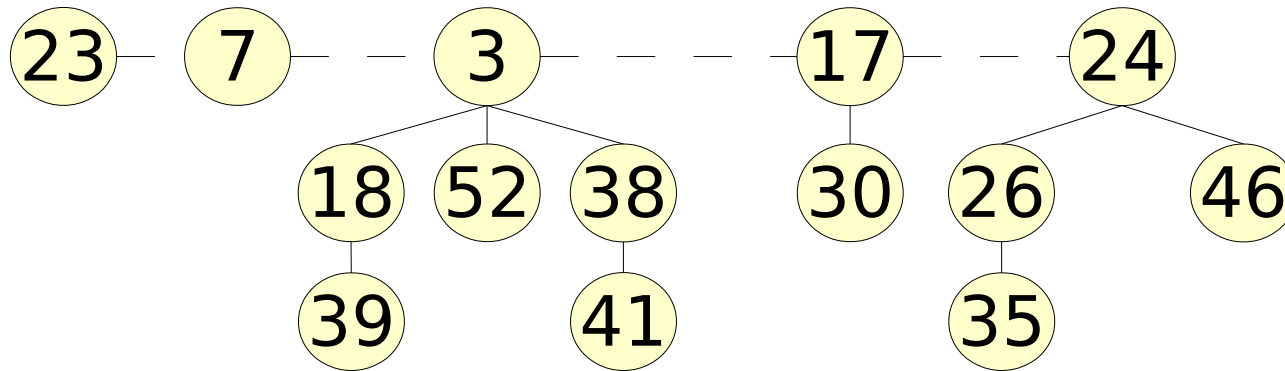
pour le calcul du minimum en $O(\log_2 N)$

$$O(N \cdot \log_2 N + A)$$

Algorithme de Dijkstra

Tas de Fibonacci

- Collection d'au plus $\log_2 N$ arbres contenant des clés. Un fils à toujours une clé \geq à son parent.



- Opérations principalement utilisées, en $O(\log_2 N)$
 - Extraire-Min(T) à chaque itération
 - Diminuer-clé(S_i) quand m.à.j des successeurs

Algorithme de Ford-Bellman

- Graphe $G=(X, U)$. $l(u)$ quelconques
- Plus court chemins de S_0 vers tous les autres sommets de X
- Marquage **D(i)** comme pour Dijkstra
- Principe : arriver à vérifier

$$\forall S_i, S_j \in X, D^*(i) + l_{i \rightarrow j} \leq D^*(j)$$

car si $\exists j$ t.q. $D(i) + l_{i \rightarrow j} \geq D^*(j)$

alors : $D^*(j) = D^*(i) + l_{i \rightarrow j}$

$D^*(i) = \text{optimum}$

Algorithme de Ford-Bellman

Mise à jour de $D(i)$

- A chaque étape, pour l'ensemble des sommets

$$\forall S_i \in X, \forall S_j \in \Gamma^{S_i}$$

S_j est un successeur de S_i

$$v \leftarrow D(i) + l_{i \rightarrow j}$$

si $v < D(j)$ alors

$$D(j) \leftarrow v$$

$$C(j) \leftarrow S_i$$

mise à jour du chemin
vers S_j en passant par S_i

Algorithme de Ford-Bellman

Nombre d'étapes

- A chaque étape **k**, mise à jour des chemins de longueur k :
 - Si $D(i)$ correspond à un chemin de longueur $k-1$
 - Si $\exists s_j \in \Gamma^{s_i}$ t.q. $D(j) > D(i) + l_{i \rightarrow j}$
Alors $D(j) \leftarrow D(i) + l_{i \rightarrow j}$ **Chemin de longueur k**
 - vrai à l'ordre 1.



- $N-1$ étapes (chemins de longueur maxi $N-1$ entre S_0 et les autres sommets)
- Si chemin $> N-1$ arcs, boucle de longueur négative

Ford-Bellman - énoncé

FordBellman(Graphe G, Sommet S_0)(Tableaux D, C)

1-Initialisation

entier $k \leftarrow 0$

Tableaux D, C $\forall S_i C[i] \leftarrow S_i$
 $D[S_0] \leftarrow 0$, et $\forall S_i \neq S_0 D[i] \leftarrow \infty$

2-Itération courante

modif \leftarrow faux

$\forall S_i \in X, \forall S_j \in \Gamma^{S_i}$

$v \leftarrow D(i) + l_{i \rightarrow j}$

si $v < D(j)$ alors

$D(j) \leftarrow v$

$C(j) \leftarrow S_i$

modif \leftarrow vrai

3- Test de fin

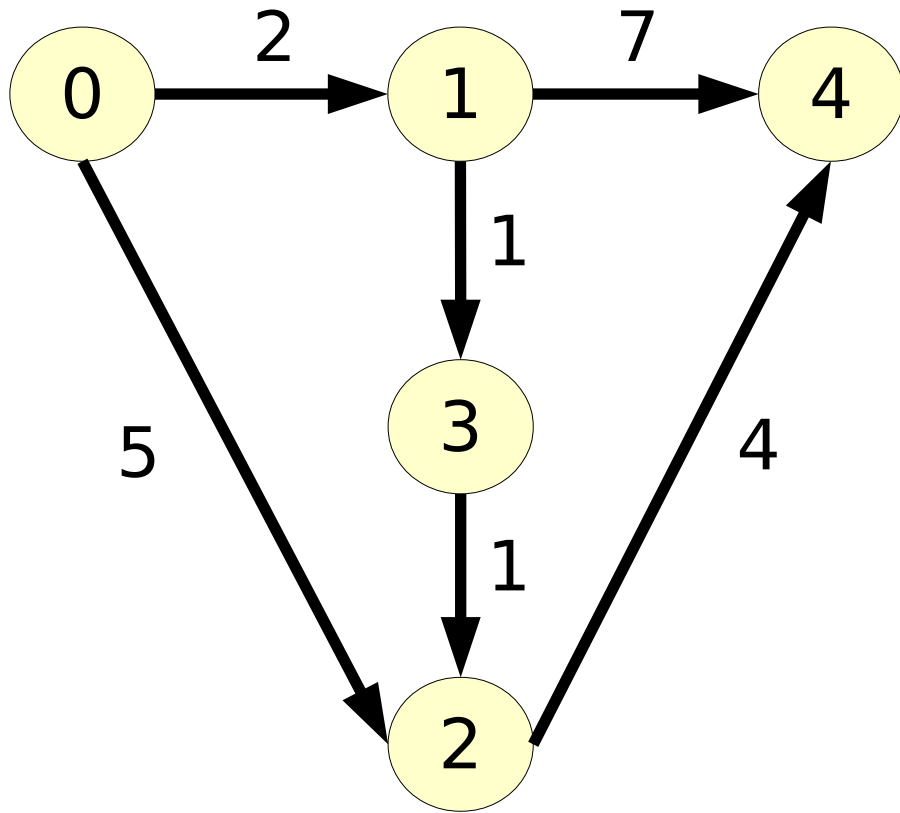
si modif = faux **fin**

$k \leftarrow k + 1$

si $k = N$ **fin**

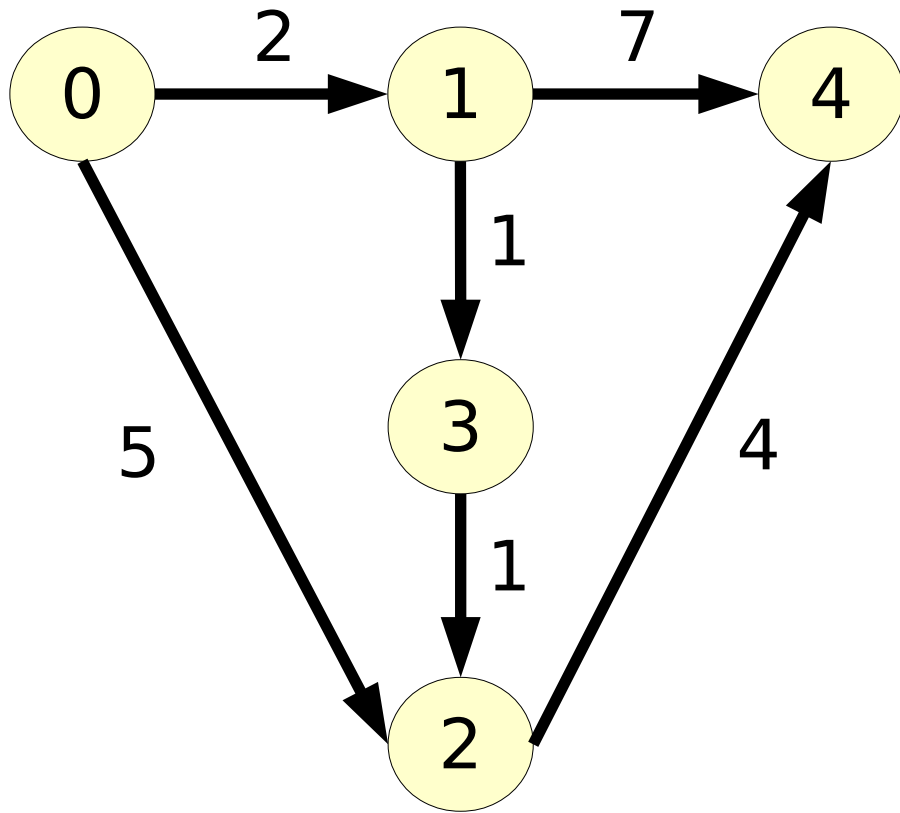
sinon aller en **2-**

Ford-Bellman exemple



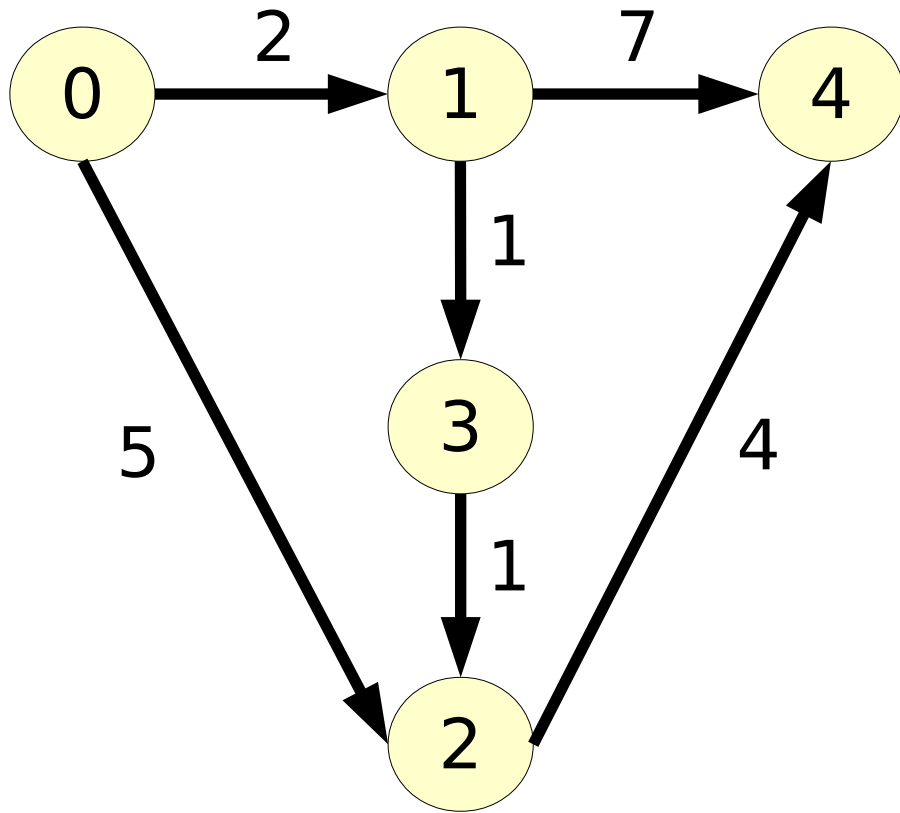
		D				
#	Succs de	D[0]	D[1]	D[2]	D[3]	D[4]
0	S_0	0	2	5	∞	∞

Ford-Bellman exemple



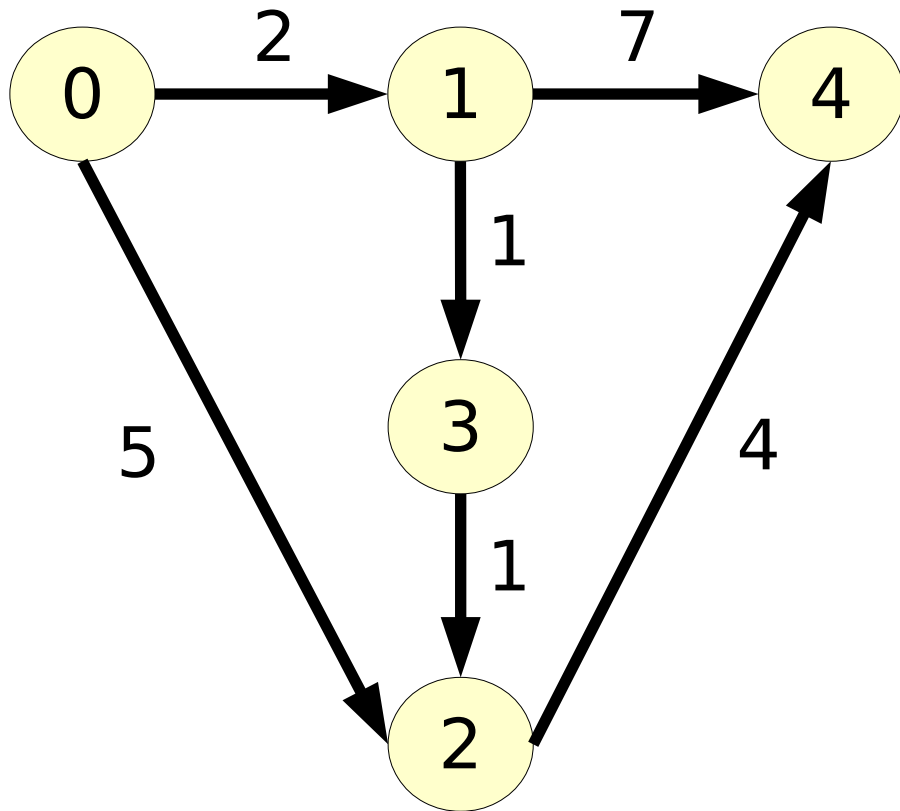
		D				
#	Succs de	D[0]	D[1]	D[2]	D[3]	D[4]
0	S_0	0	2	5	∞	∞
1	S_0	0	-	-	-	-
	S_1	0	-	-	3	9
	S_2	0	-	-	-	-
	S_3	0	-	4	-	-

Ford-Bellman exemple



		D				
#	Succs de	D[0]	D[1]	D[2]	D[3]	D[4]
0	S_0	0	2	5	∞	∞
1	S_0	0	-	-	-	-
	S_1	0	-	-	3	9
	S_2	0	-	-	-	-
	S_3	0	-	4	-	-
2	S_0	0	-	-	-	-
	S_1	0	-	-	-	-
	S_2	0	-	-	-	8
	S_3	0	-	-	-	-

Ford-Bellman exemple



		D				
#	Succs de	D[0]	D[1]	D[2]	D[3]	D[4]
0	S_0	0	2	5	∞	∞
1	S_0	0	-	-	-	-
	S_1	0	-	-	3	-
	S_2	0	-	-	-	-
	S_3	0	-	4	-	-
2	S_0	0	-	-	-	-
	S_1	0	-	-	-	-
	S_2	0	-	-	-	8
	S_3	0	-	-	-	-
3	K = 3 et modif = faux					

Chemin de capacité maximale

- Capacité c_u d'un arc : nombre positif ($c_u \geq 0$)
- Capacité d'un chemin P :
minimum des capacités des arcs du chemin

$$C_P = \min_{\{u \in C\}} C_u$$

- Le problème du chemin de capacité maximum (CCM) est analogue au problème du plus court chemin (PCC) :

$$\text{CCM : } \forall S_j \neq S_0 \in \Gamma^{S_i}, C_j^* = \max_{\{i \in \Gamma^{-1}S_j\}} \min(C_i^*, C_{i \rightarrow j})$$

$$\text{et } C_0^* = \infty$$

$$\text{PCC : } \forall S_j \neq S_0 \in \Gamma^{S_i}, D_j^* = \min_{\{i \in \Gamma^{-1}S_j\}} D_i^* + l_{i \rightarrow j} \text{ et } D_0^* = 0$$

Capacité maximale - algorithme

CapacitéMax(Graphe G, Sommet S_0)(Tableau D)

1-Initialisation

Ensemble $E \leftarrow \{S_0\}$

Tableau D, $D[i] \leftarrow c_{0 \rightarrow i}$ (si l'arc $S_0 \rightarrow S_i$ existe, 0 sinon)

2-Itération courante

choisir $S_j \notin E$ t.q. $D[j]$ maximum

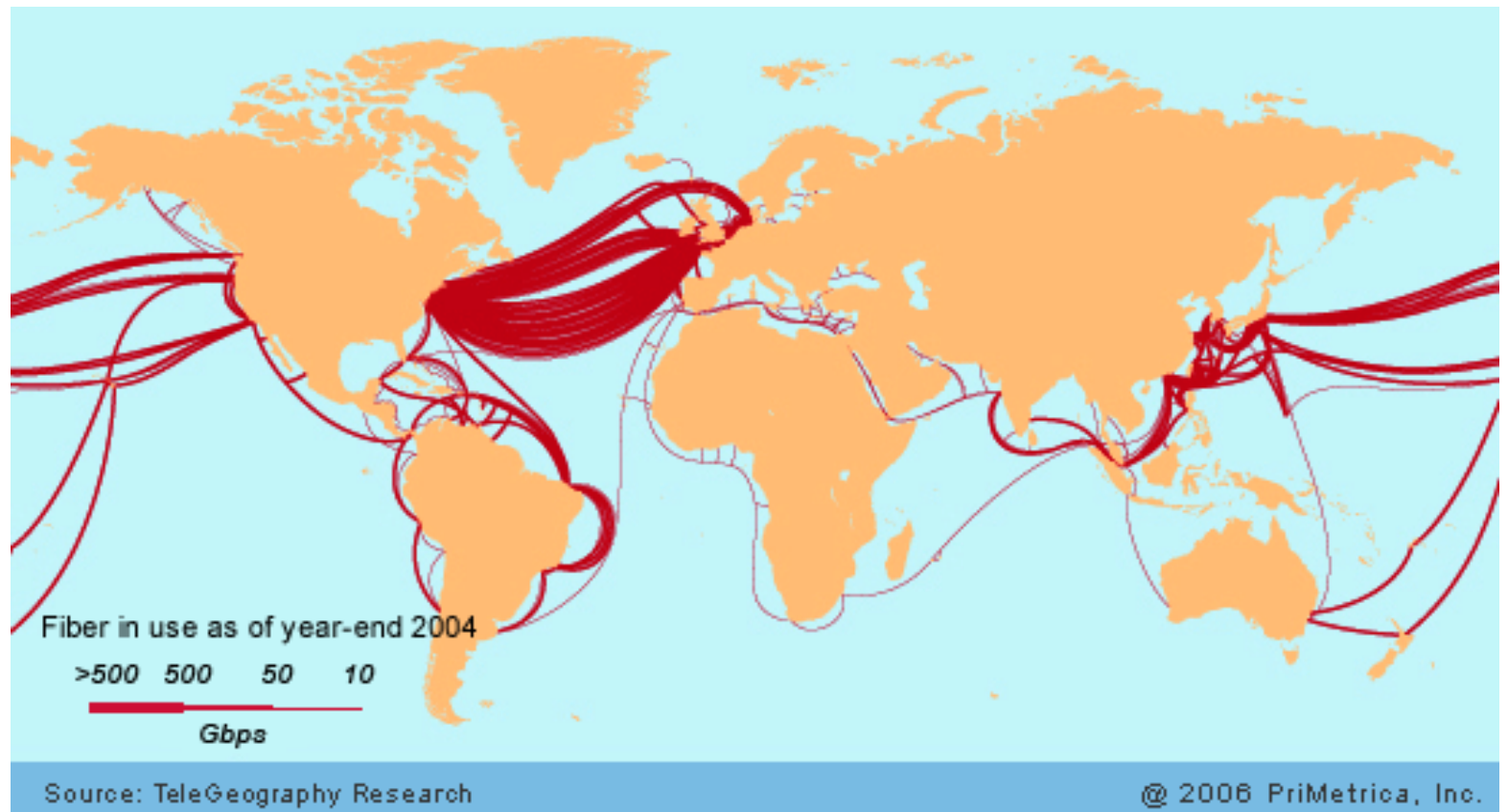
$E \leftarrow E \cup \{S_j\}$

pour chaque k t.q. $S_k \notin E$

$$D[k] = \max(D[k], \min([D[j], c_{j \rightarrow k}]))$$

Exercice

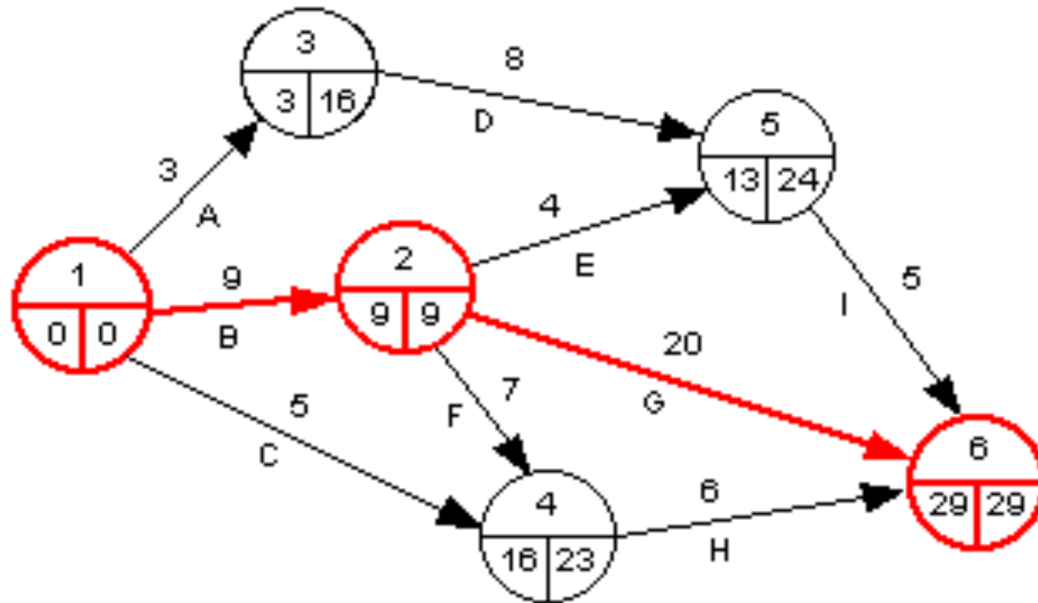
- Par où obtient on le meilleur débit pour télécharger un film australien (si les liaisons terrestres sont de capacités très supérieures aux liaisons sous marines) ?



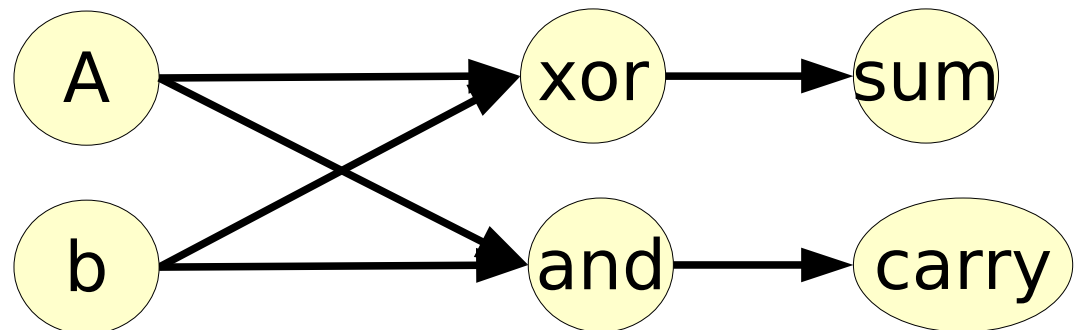
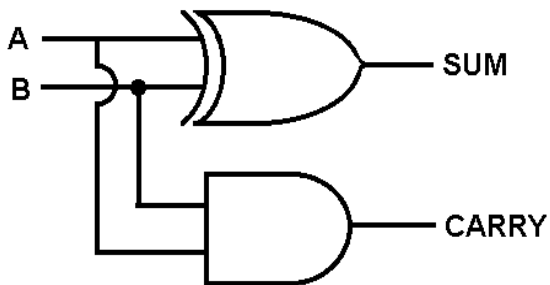
Graphes acycliques orientés

- Certains graphes ne peuvent contenir de cycles par construction

- PERT



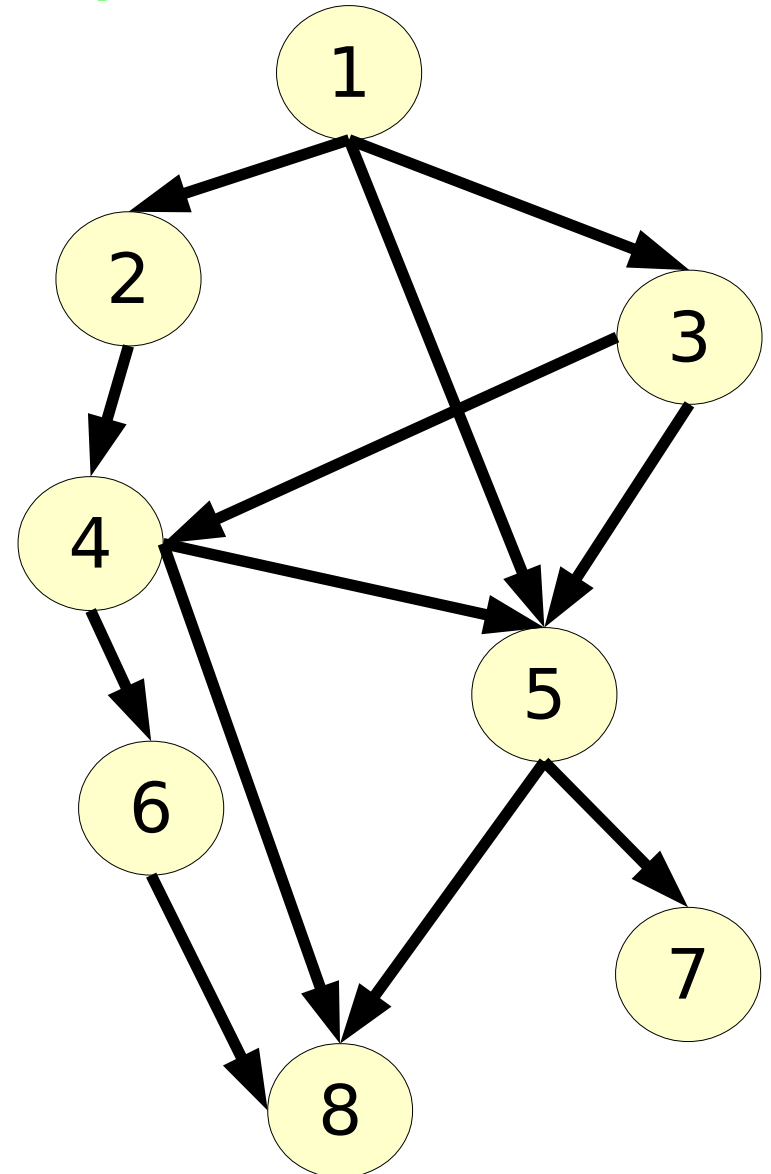
- Réseau booléen



Graphes acycliques orientés tri topologique

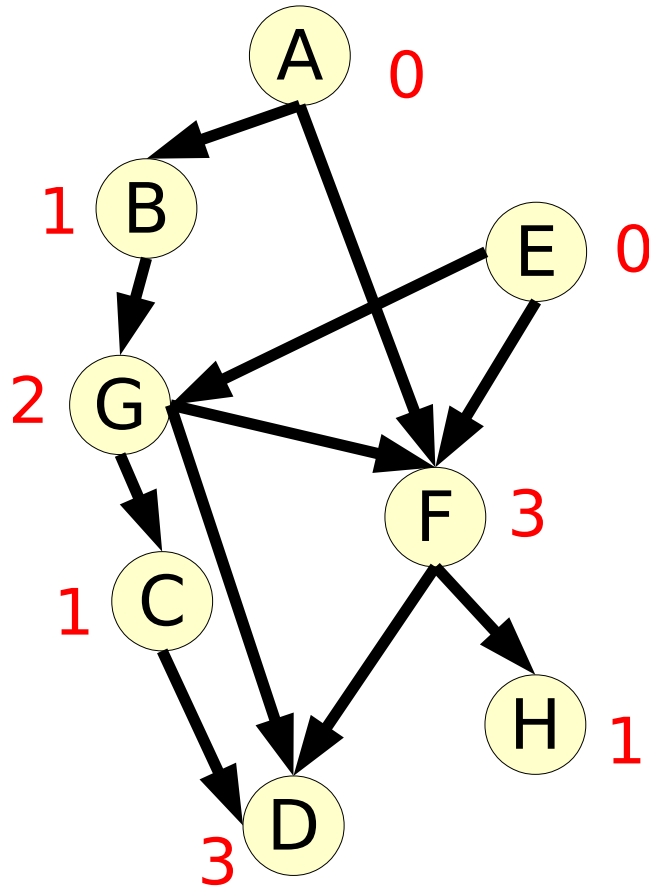
- $\forall S_i \in X, \forall S_j \in \Gamma^{S_i},$
 $\text{index}[S_i] < \text{index}[S_j]$
**possible uniquement si
aucun cycle !**
- L'examen des sommets
suivant l'ordre topologique
permet de calculer des
plus courts chemins en
 $O(n+m)$, même si $lg < 0$

(Dijkstra: $O(n \cdot \log(n))$)



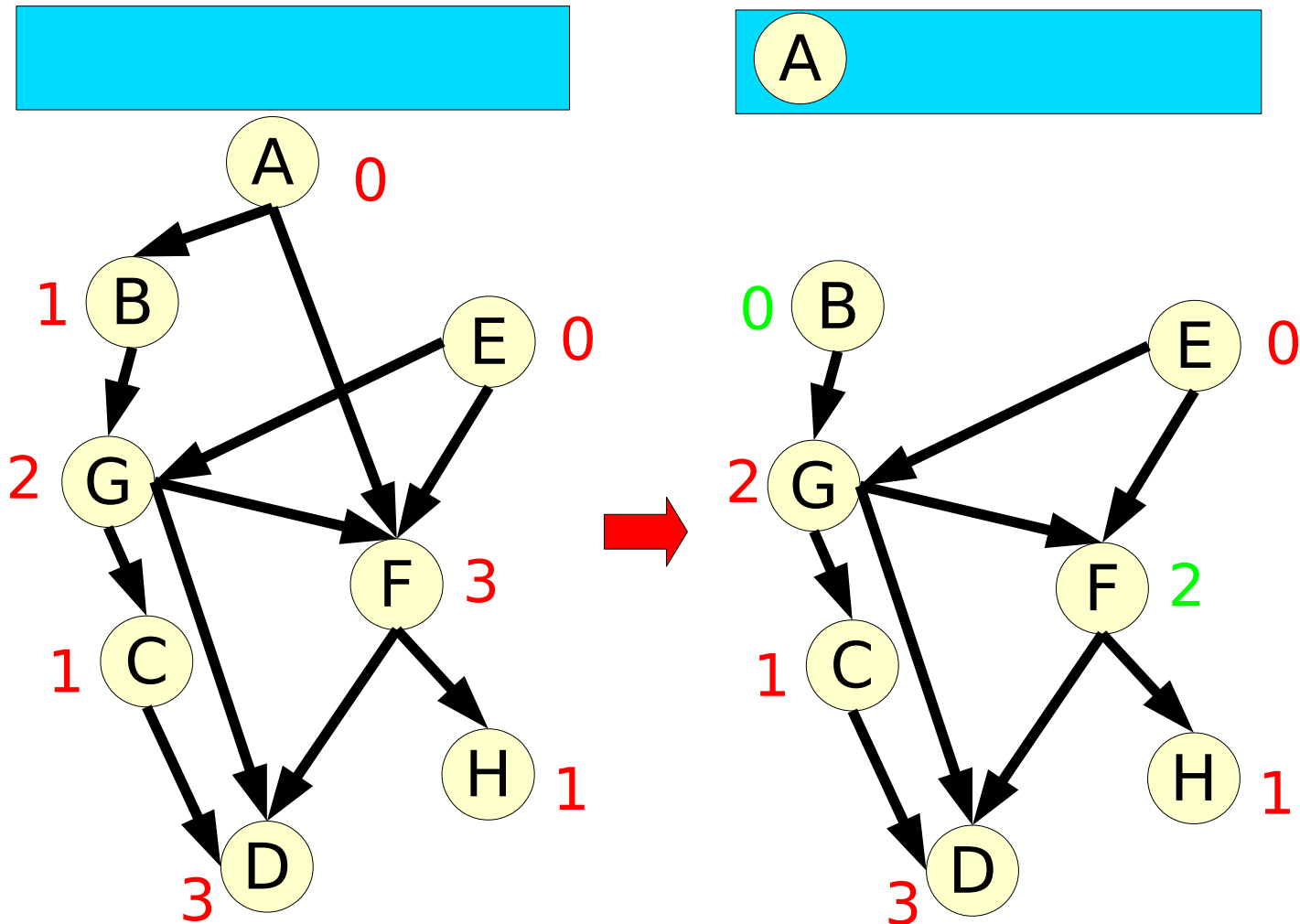
Tri topologique - principe

- Les sommets de **degré négatif** nuls sont les premiers dans l'ordre topologique



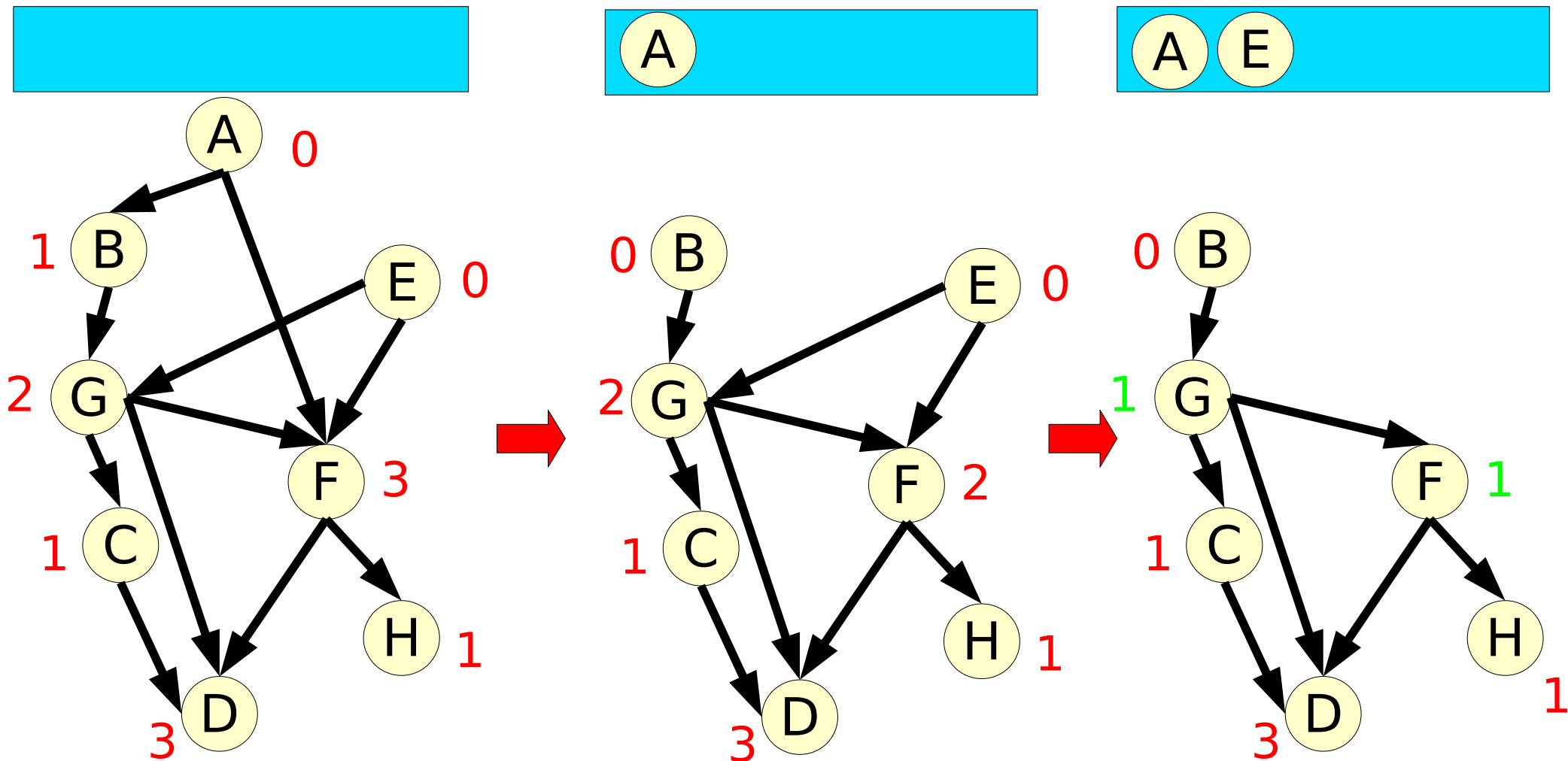
Tri topologique - principe

- On en choisit un et on le supprime.
on met à jour les degrés



Tri topologique - principe

- On en choisit un et on le supprime.
on met à jour les degrés



Tri topologique - algorithme

TriTopologique(Graphe G)(Tableau topologique T)

1-Initialisation

$\forall S_i \in X, d[S_i] \leftarrow \text{deg}^-(S_i)$

$\text{idx} \leftarrow 0$

$\text{liste} \leftarrow \{ S_i \in X, \mid d[S_i] = 0 \}$

2-Itération courante

si liste vide aller en **3-**
retirer un sommet S de liste

$\text{idx} \leftarrow \text{idx} + 1$

$T[\text{idx}] \leftarrow S$

$\forall S_j \in \Gamma^s,$

$d[S_j] \leftarrow d[S_j] - 1$

si $d[S_j] = 0$

$\text{liste} \leftarrow \text{liste} \cup \{S_j\}$

3-fin

si $\text{idx} < |X|$ cycle sinon
retourner T

Plus court chemin sur GAO algorithme de Bellman

BellmanGAO(Graphe G, sommet S_A)(Tableau D)

1-Initialisation

renuméroter les sommets par ordre topologique

Soit i_0 l'index de S_A .

$$\forall 1 \leq j < i_0, D[S_j] \leftarrow \infty$$

$$D[S_{i_0}] \leftarrow 0$$

pas de chemin $S_{i_0} \rightarrow S_j$

2-Itération courante

pour j de $i_0 + 1$ à n

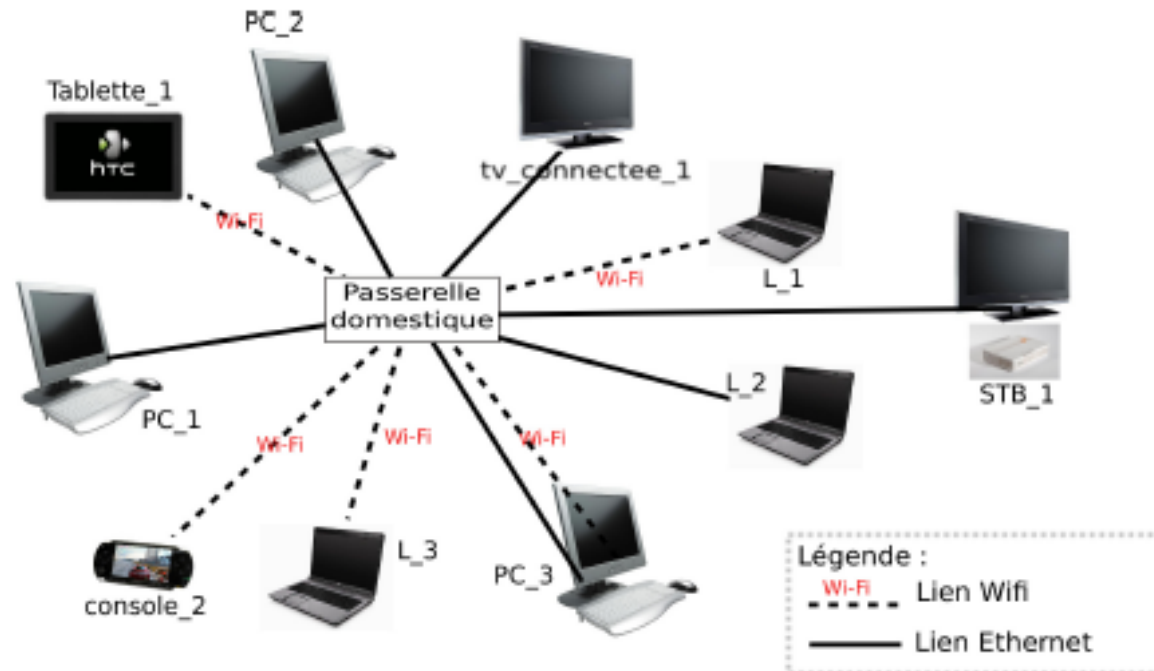
$$D[S_j] \leftarrow \min_{\{S_i \in \Gamma^{-1}(S_j)\}} D[S_i] + l_{i \rightarrow j}$$

Plus court chemins

Application en QoS

Domaine de la Qualité de service (QoS)

- Diffusion de vidéo sur réseau domestique
- Priorité de la vidéo / navigation web
- Réservation de bande passante

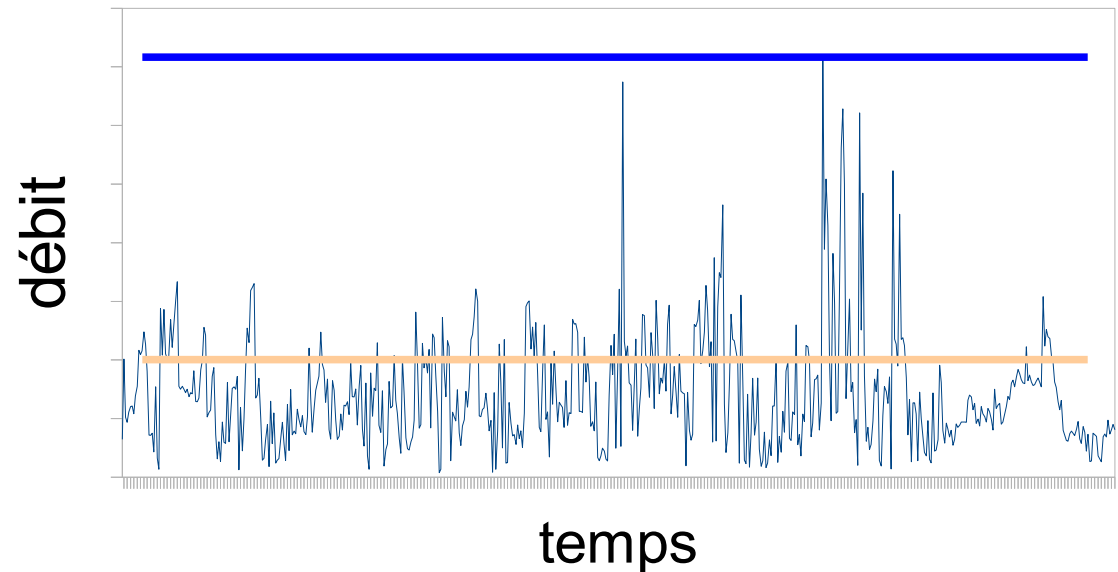


Plus court chemins

Bande passante variable

Nombreux encodages à débit variable (VBR)

- Réserver le **max** ? **Perte de BP**
- Réserver la **moyenne** ? **Perte de QoS**




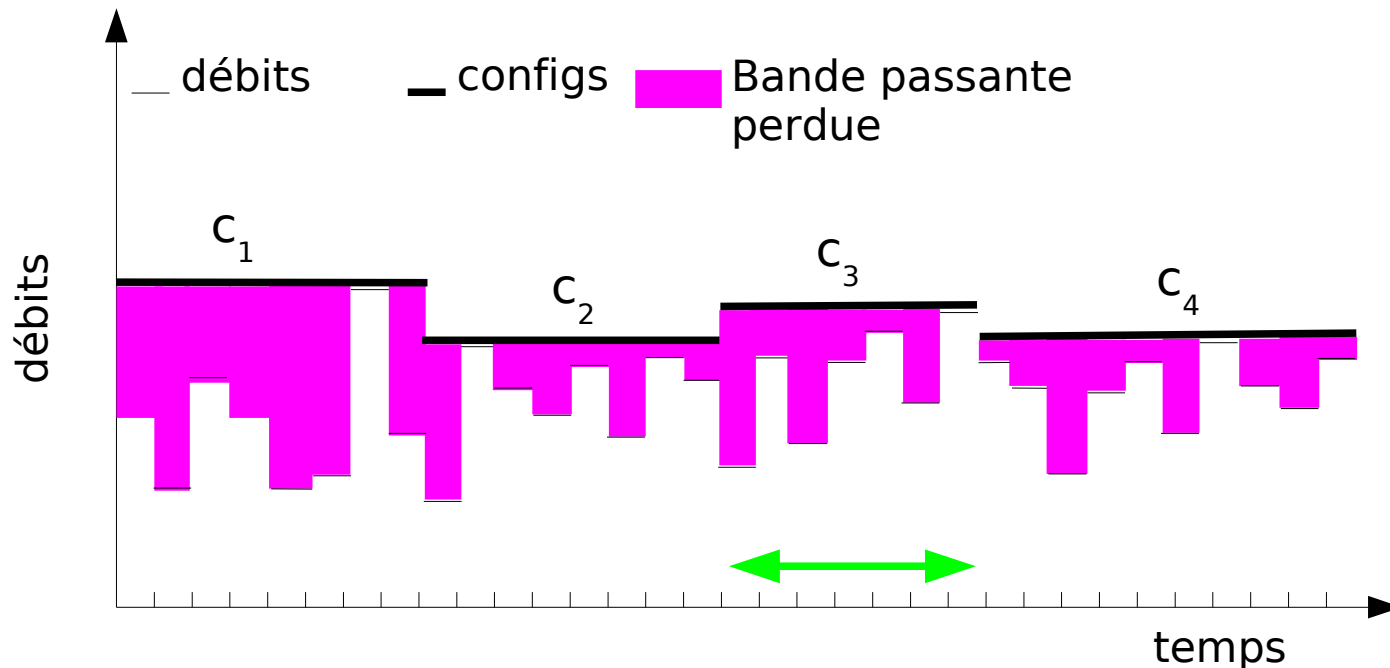
Réservation variable sous contraintes

Plus court chemins

Bande passante variable

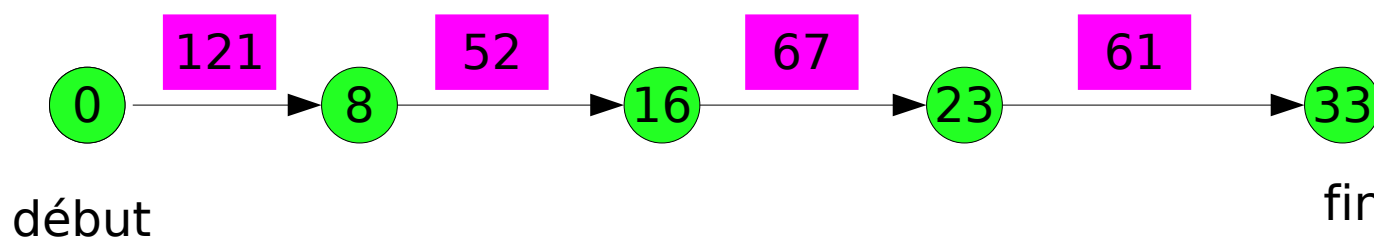
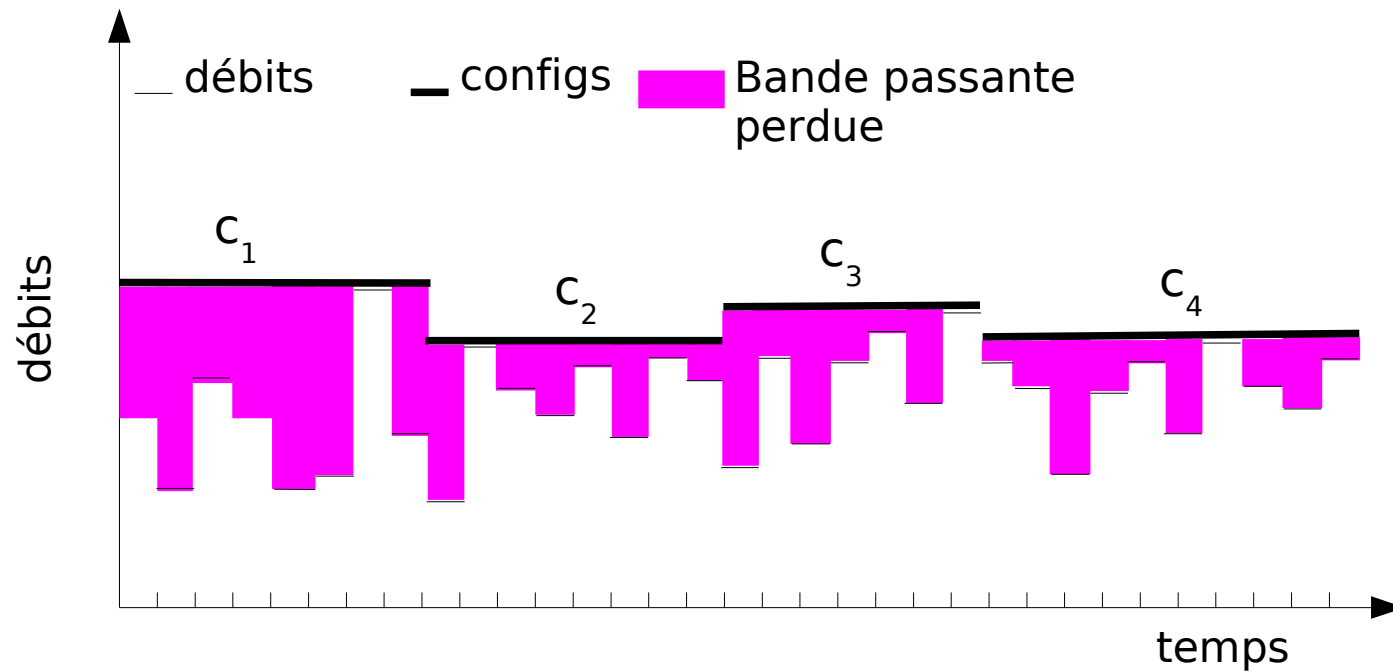
Contraintes sur les réservations de bande passante

- M : Nombre de configs c_i différentes limité
- P : Temps minimum entre 2 reconfigurations 
(durée min d'une configuration)



graphe

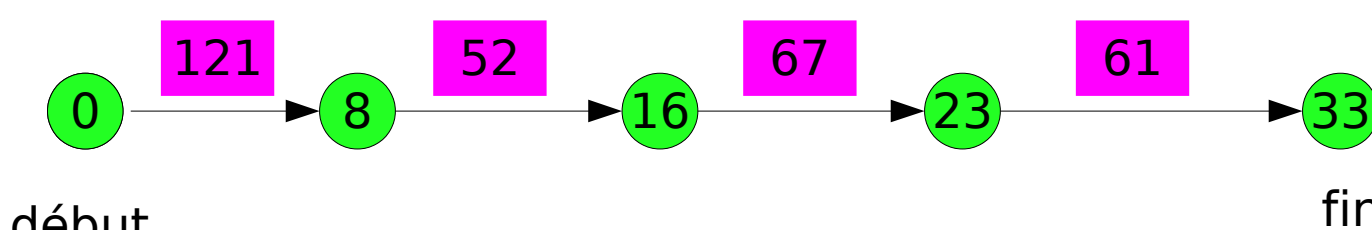
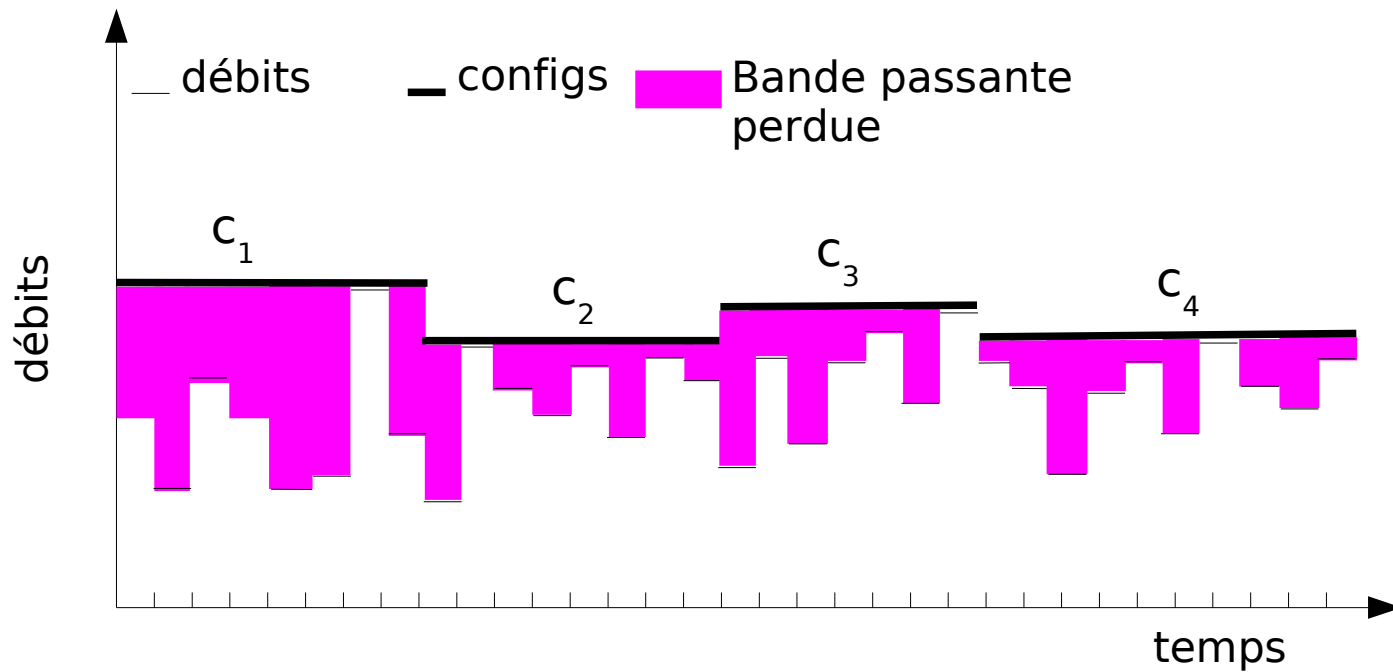
Bande passante variable



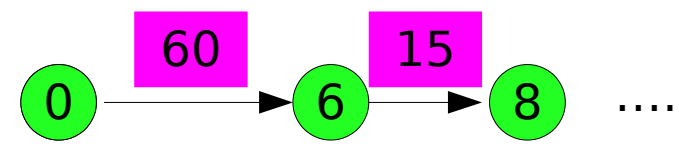
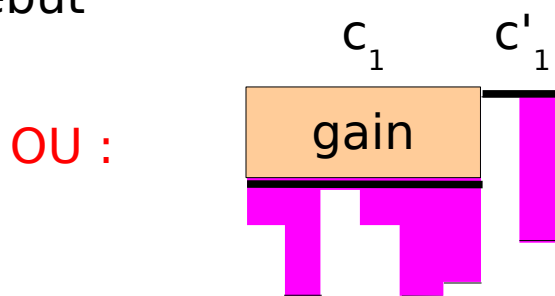
Coût
Total :
301

graphe

Bande passante variable



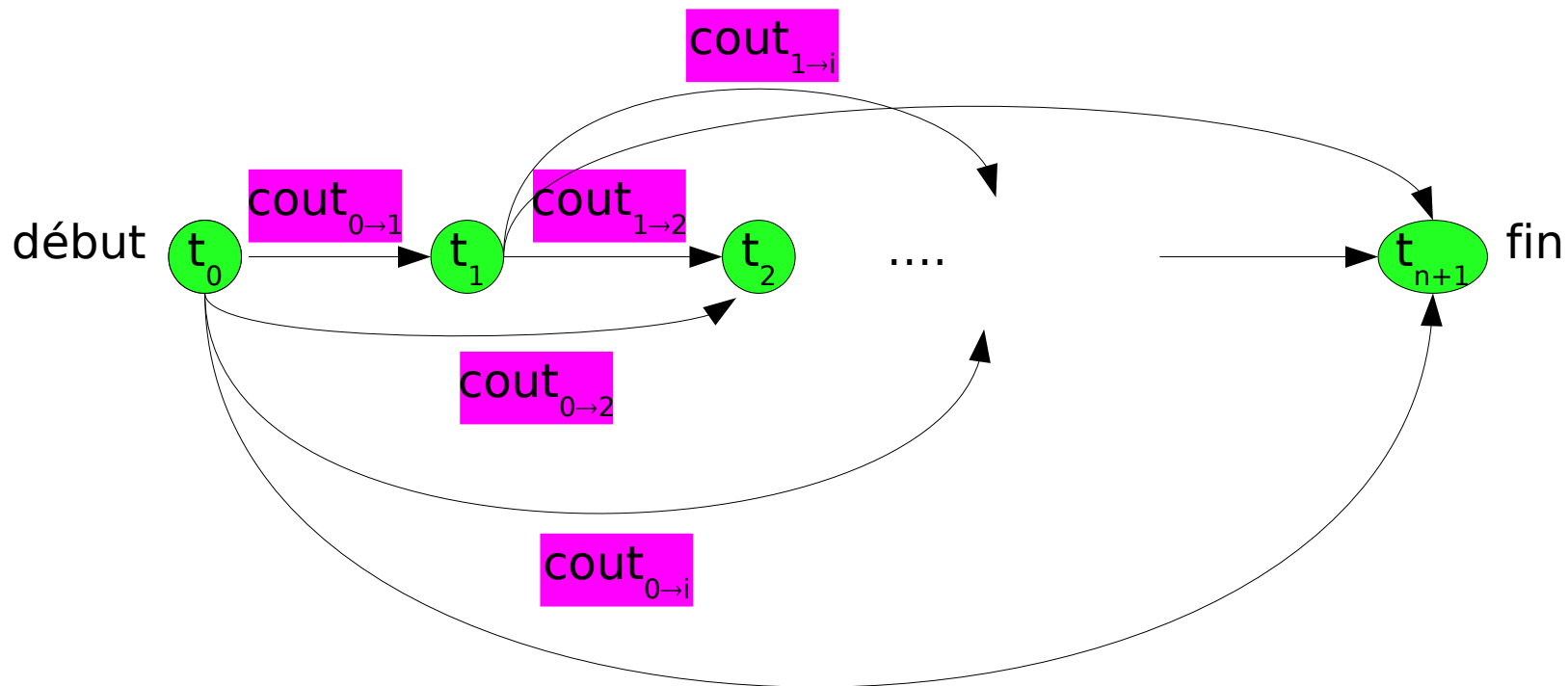
Coût Total : 301



Coût Total : 255

graphe

Bande passante variable



- Un sommet par date
- $\forall j > i$ un arc $t_i \rightarrow t_j$: config à date t_i et reconfig à t_j
- Poids correspondant au surcoût
- Un sommet supplémentaire à la fin

graphe

Bande passante variable

Meilleure solution : coût total minimal

- plus court chemin

Bellman sur GAO

Contraintes sur les réservations de bande passante

- P : Temps minimum entre 2 reconfigurations
(durée min d'une configuration)

enlever les arcs $i \rightarrow j$ t.q $j - i < P$

- M : Nombre de configs c_i différentes limité

M premières étapes de Ford-Bellman

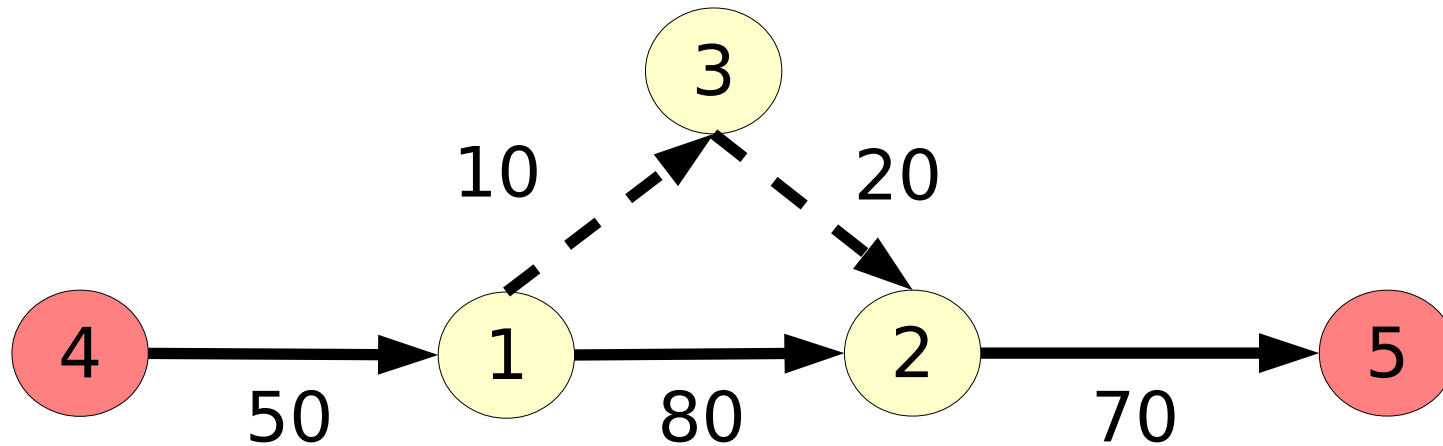
Toutes paires de sommets algorithme de Floyd-Warshall

- Graphe $G=(X, U)$
 $N=|X|$
- Dijkstra, Ford-Bellman :
1 vers N
 $C = O(N^2 + |U|)$
- Tous vers tous avec
Dijkstra
 $O(N * C) = O(N^3 + N)$
- Floyd : algorithme
matriciel
 $O(N^3)$



Algorithme de Floyd - Exemple

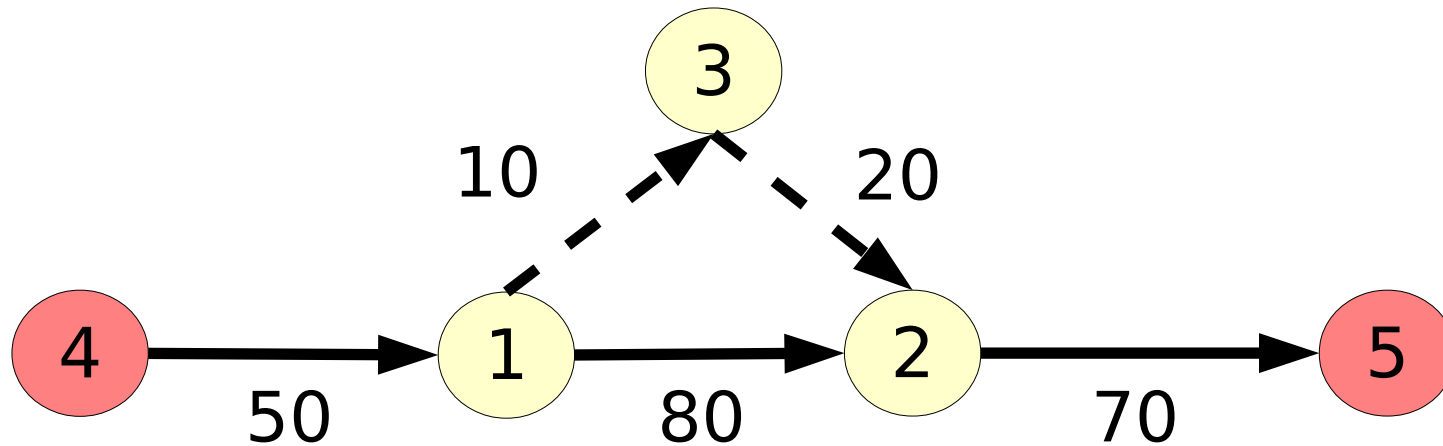
- Chemins entre toutes paires de sommets
- Idée : à l'étape **k**, on connaît les chemins $i \rightarrow j$ optimaux *ne passant que* par S_1, S_2, \dots, S_k



- **k=2**. Entre autres :
 $D_{45} = 50 + 80 + 70 = 200$
 $D_{43} = 50 + 10 = 60$
 $D_{35} = 20 + 70 = 90$

Algorithme de Floyd - Exemple

- Chemins entre toutes paires de sommets
- Idée : à l'étape **k**, on connaît les chemins $i \rightarrow j$ optimaux *ne passant que* par S_1, S_2, \dots, S_k



- **k=2.** Entre autres :

$$D_{45} = 50 + 80 + 70 = 200$$

$$D_{43} = 50 + 10 = 60$$

$$D_{35} = 20 + 70 = 90$$

- **k=3.** D_{43} et D_{35} inchangés ...

Si D_{45} ne passe pas par 3

$$D_{45}^3 = D_{45}^2 = 200$$

Sinon

$$D_{45}^3 = D_{43} + D_{35} = 150$$

Algorithme de Floyd - Principe

- En généralisant, pour $G = (X, U)$

$$\forall S_i, S_j \in X, \quad L_{ij}^k = \min \{ L_{ij}^{k-1}, L_{ik}^{k-1} + L_{kj}^{k-1} \}$$

- L est une matrice $N \times N$ ($= |X|$)
- L^0 est défini par :
 - $L_{ij}^0 = D_{ij}$ si $i \neq j$ ($D_{ij} = \infty$ si l'arc $i \rightarrow j$ n'existe pas)
 - $L_{ii}^0 = D_{ii} = 0$
- A la $N^{\text{ième}}$ étape, L contient des chemins pour les sommets S_1 à S_N , ie les plus courts chemins

Floyd - énoncé

Floyd(Graphe $G = (X, U)$)(Tableaux D, A)

1-Initialisation

Pour $i \leftarrow 1..N$, Pour $j \leftarrow 1..N$

$D[i,j] \leftarrow l_{ij}$, $D[i,i] \leftarrow 0$

$A[i,j] \leftarrow S_i$

$k \leftarrow 0$

D : distances
 A : précédences

2-Itération courante $k \leftarrow 1..N$

Pour $i \leftarrow 1..N$, Pour $j \leftarrow 1..N$

$v \leftarrow D[i,k] + D[k,j]$

Si $v < D[i,j]$

$D[i,j] \leftarrow v$

$A[i,j] \leftarrow A[k,j]$

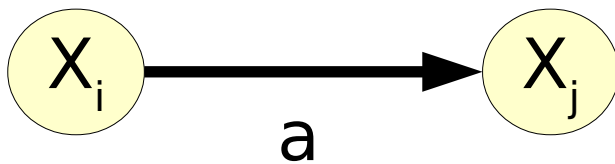
Si $(i=j)$ et $(v < 0)$

fin

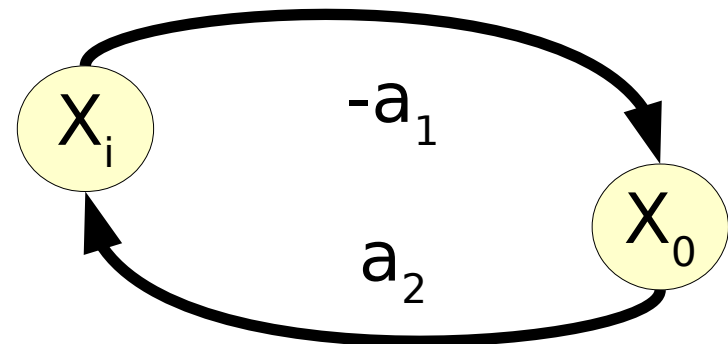
Algorithme de Floyd cohérence de graphes STP

- Floyd permet de détecter les cycles de longueur négative
 - $L_{ii}^N < 0$
- Vérifier la cohérence des problèmes temporels simples (STP)
 - ensembles de contraintes de dates/durées
 - correspondent à un graphe de distances

$$X_j - X_i \leq a$$



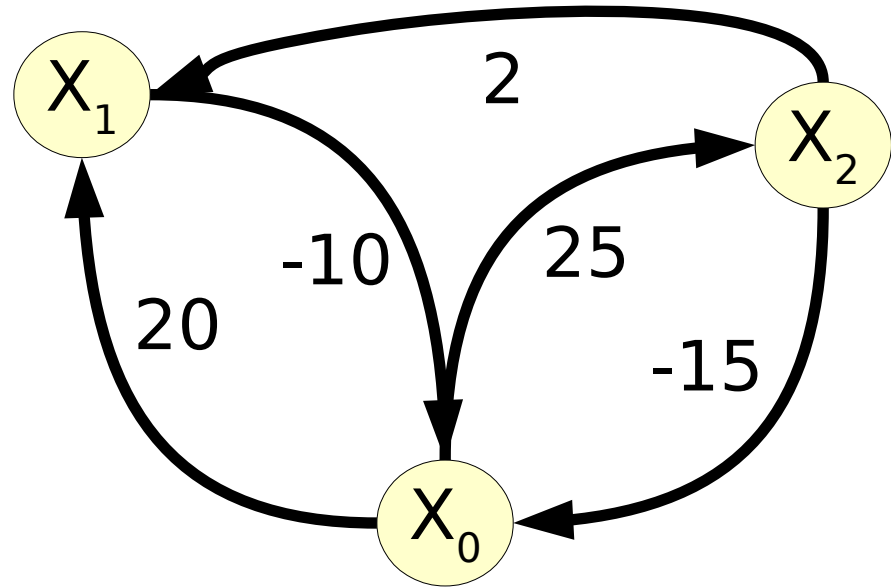
$$a_1 \leq X_i \leq a_2$$



STP - Exemple

- contraintes C :

$$\begin{cases} c_1: 10 \leq X_1 \leq 20 \\ c_2: 15 \leq X_2 \leq 25 \\ c_3: X_1 - X_2 \leq 2 \end{cases}$$



- Y a t il une date possible pour X_1 et X_2 respectant les contraintes ? **pas de cycle négatif !**
- Application** : si X_1 et X_2 sont des niveaux d'eau estimés sur des parcelles lors d'une inondation (c_1 et c_2) et que il y a des flux d'eau (c_3), y a t il des incohérences ?