RO03
Problèmes de cheminement

Exemples d'applications

- Rechercher un meilleur itinéraire (le plus court, le moins long, le plus beau, le plus intéressant, le plus sûr) : GPS, sites internet, etc.
- Quand remplacer sa voiture en fonction du prix de revente (qui diminue d'année en année) et des frais d'entretient (qui augmente d'année en année) ?
- Comment optimiser ses placements financiers en fonction des rendements des différentes possibilités ?
- Trouver la suite d'actions pour réussir un jeu solitaire déterministe de stratégie.

Applications

- Sous-problème de nombreux problèmes d'optimisation.
- Applications dans les transports:
  - Tournées de véhicules;
  - Détermination du trajet le plus rapide ou le plus court.
- Théorie des jeux.
- Applications dans les réseaux télécom.
- *etc.*

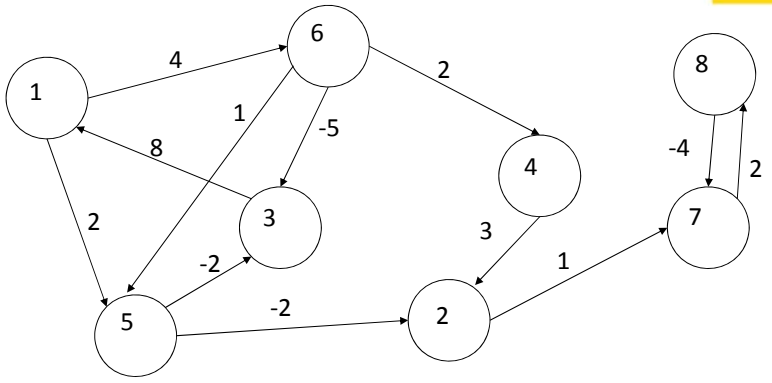
Problèmes de chemins optimaux

- G=(X,U,v) avec:
  - $X=\{x_0, x_1, x_2,\ldots, x_{n-1}\}$ et $v : U \rightarrow \Re$
- Longueur d'un chemin : nombre d'arcs du chemin
- Valeur d'un chemin : somme des valuations du chemin
- Un chemin de x_i à x_k est de valeur minimale si sa valeur est plus petite (inférieure ou égale) que celle de tout autre chemin allant de x_i à x_k.

Problèmes de chemins optimaux

Trois types de problèmes:

- Etant donné deux sommets x_i et x_k, trouver un chemin de valeur minimale (s'il existe);
- Etant donné un sommet x_s, trouver les chemins de valeurs minimales (s'ils existent) allant de x_s à tout autre sommet x_i;
- Trouver un chemin de valeur minimale (s'ils existent) entre tout couple de sommets.



- Trouver un chemin de valeur minimale entre les sommets 1 et 2.
- Existe-t-il un chemin de valeur minimale entre 1 et 7 ?

Propriétés des chemins minimaux

- Tout sous-chemin d'un chemin de valeur minimale est un chemin de valeur minimale.

Propriétés des chemins minimaux

- Supposons qu'il existe un chemin de x_0 à x_i. S'il existe un circuit de valeur négative passant par x_i, alors il n'existe pas de chemin de valeur minimale de x_0 à x_i.
- $\Rightarrow$ Il est nécessaire qu'il n'existe pas de circuit de valeur négative passant par x_i pour qu'il existe un chemin de valeur minimale de x_0 à x_i.

Propriétés des chemins minimaux

- En supposant qu'il existe au moins un chemin allant de x_0 à x_i quelque soit i (x_0 est une racine de G), si G est sans circuit de valeur strictement négative, alors il existe un chemin de valeur minimale de x_0 à tout autre sommet x_i.
- $\Rightarrow$Une condition suffisante pour que, pour tout i, il existe un chemin de valeur minimale allant de x_0 à x_i est que le graphe G soit sans circuit de valeur strictement négative (ces circuits sont dit absorbants).

Propriétés des chemins minimaux

- Une condition nécessaire et suffisante pour que, pour tout i, il existe un chemin de valeur minimale allant de x_0 à x_i est que le graphe G soit sans circuit de valeur strictement négative (ces circuits sont dit absorbants).

Propriétés des chemins minimaux

- Soit G un graphe sans circuit de valeur strictement négative et λ_i des valeurs de chemins entre x_0 et tout sommet x_i.
- Une condition nécessaire et suffisante pour que $\{\lambda_i / 0 \leq i \leq n-1\}$ soit l'ensemble des valeurs des chemins minimaux issus de x_0 est que :

  1) $\lambda_0 = 0$;

  2) $\lambda_j \leq \lambda_i + v_{ij}$, pour tout arc $(x_i, x_j) \in U$.

- La condition est nécessaire : si elle n'est pas vérifiée, il existe un λ_i qui ne pas être la valeur d'un chemin optimale de x_0 à x_i.
- La condition est suffisante : si elle est vérifiée, $\forall i$, il ne peut pas y avoir un chemin $[z_0 = x_0, z_1, z_2, \ldots, z_k = x_i]$ de valeur λ^*_i avec $\lambda^*_i < \lambda_i$.

Propriétés des chemins minimaux

- Corollaire :

  l'ensemble des arcs (x_i, x_j) pour lesquels $\lambda_j - \lambda_i = v_{ij}$ est l'ensemble des arcs appartenant à des chemins minimaux.

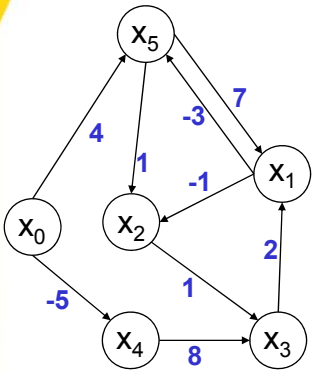
Algorithmes de cheminements

- Algorithme de FORD :
  - Valide avec des valuations quelconque
  - O(n m)
  - Algorithme à correction d'étiquettes
- Algorithme de DIJKSTRA :
  - Valide uniquement avec des valuations positives ou nulles
  - O(n²)
  - Algorithme à fixation d'étiquette
- Algorithme de BELLMAN :
  - Valide avec des valuations quelconque mais uniquement s'il n'existe pas de circuit
  - O(m)
  - Algorithme à correction d'étiquettes

Algorithme de FORD

```
λ[0] ←0; P[0] ←0;
pour i←1 à n-1 faire { λ[i] ← ∞; P[i] ← -1; }
faire {
    modification←faux;
    pour i←0 à n-1 faire {
        si (P[i]≠-1) alors {
            pour chaque successeurs x_j de x_i faire {
                si (λ[j]> λ[i]+v_ij) alors {
                    P[j] ←i; // x_j est atteint à partir de x_i
                    modification ←vrai;
                    λ[j] ← λ[i]+v_ij;
                }
            }
        }
    }
} tant que (modification=vrai)
```

Algorithme de FORD




```
λ[0] ←0; P[0] ←0;
pour i←1 à n-1 faire { λ[i] ← ∞; P[i] ← -1; }
faire {
    modification←faux;
    pour i←0 à n-1 faire {
        si (P[i]≠-1) alors {
            pour tout xj∈U+(xi) faire {
                si (λ[j]> λ[i]+vij) alors {
                        P[j] ←i;
                        modification ←vrai;
                        λ[j] ← λ[i]+vij;
                }
            }
        }
    }
} tant que (modification=vrai)
```


Algorithme de FORD : preuve

- Théorème : si le graphe est sans circuit absorbant, l'algorithme calcule les valeurs des chemins minimaux.
- Démonstration de l'algorithme par récurrence :
  - On appelle λ_i^k, la valeur minimale d'un chemin de x_0 à x_i empruntant au plus k arcs.

Invariant:

À la fin de la k^e itération, si $P[i] \neq -1$, $\lambda[i]$ est la valeur d'un chemin de x_0 à x_i telle que $\lambda[i] \leq \lambda_i^k$.

Algorithme de FORD : complexité

- **L'invariant est vrai à la fin de la n-1ème itération :**
  A la fin de la n-1ème itération, $\lambda[i]$ est la valeur d'un chemin de x_0 à x_i telle que $\lambda[i] \leq \lambda_i^{n-1}$. Donc $\lambda[i]$ est la valeur d'un chemin de valeur minimale.

- **Complexité** : O(n m)

- On peut continuer l'algorithme après l'itération n-1. Si les valeurs changent encore, cela indique la présence d'un circuit absorbant.

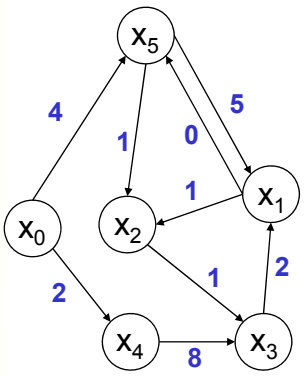
Algorithme de FORD modifié

```
λ[0] ←0; P[0] ←0; Nb_iterations ←0;
pour i←1 à n-1 faire { λ[i] ← ∞; P[i] ← -1; }
faire {
    modification←faux; Nb_iterations ← Nb_iterations+1;
    pour i←0 à n-1 faire {
        si (P[i]≠-1) alors {
            pour chaque successeurs j de i faire {
                si (λ[j]> λ[i]+v_ij) alors {
                    P[j] ←i;  modification ←vrai;
                    λ[j] ← λ[i]+v_ij;
                }
            }
        }

    }
} tant que (modification=vrai et Nb_iterations<n)
si (Nb_iterations=n) alors écrire(« circuit absorbant ») ;
```

Algorithme de DIJKSTRA

Attention : valide uniquement si les valuations sont ≥ 0




```
pour i←0 à n-1 faire {
    λ[i] ← ∞; visité[i] ← faux; P[i] ← -1;
}
λ[0] ←0; P[0] ←0; nb_visités←0;
tant que (nb_visités < n){
    min ← -1;
    pour i←1 à n-1 faire {
        si (visité[i]=faux et (min=-1 ou λ[i] < λ[min] )
        alors min ← i;
    }
    visité[min] ←vrai;  nb_visités← nb_visités +1;
    pour j ∈ U+(min) faire {
        si (visité[j]=faux et λ[j]> λ[min]+vij) alors {
            λ[j] ← λ[min]+vmin,j;  P[j] ←min;
        }
    }
}
```


Algorithme de DIJKSTRA

- valuations ≥ 0 : il n'existe pas de circuit absorbant

- Démonstration de la validité de l'algorithme par récurrence :

Invariant:

A la fin de l'itération **k**,

1- Si $x_i \in S$ (visité[i]=vrai) : **$\lambda[i] = \lambda^*_i$**.

2- Si $x_i \notin S$ (visité[i]=faux) : **$\lambda[i] = \min_{z \in U^-(i) \cap S} \lambda[z]+v_{zi}$**

- **Complexité** : $O(n^2)$

Edsger Wybe Dijkstra (1930-2002)

Mathématicien et informaticien néerlandais.

- A formalisé le concept de sémaphore et introduit le concept de section critique
- A publié un article célèbre contre le GOTO et qui en a fait une instruction marginalisée
- A joué un rôle important dans le développement du langage Algol.
- A développé ensuite « la science et l'art des langages de programmation », contribuant grandement à notre compréhension de leur structure, de leur représentation et de leur implémentation ».

Edsger Wybe Dijkstra (1930-2002)

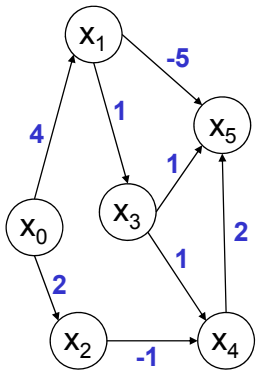
Quelques citations célèbres:

- « Tester un programme peut démontrer la présence de bugs, jamais leur absence ».
- « Se demander si un ordinateur peut penser est aussi intéressant que de se demander si un sous-marin peut nager. »
- « La programmation par objets est une idée exceptionnellement mauvaise qui ne pouvait naître qu'en Californie. »
- « Les progrès ne seront possibles que si nous pouvons réfléchir sur les programmes sans les imaginer comme des morceaux de code exécutable. »
- « Autrefois les physiciens répétaient les expériences de leurs collègues pour se rassurer. Aujourd'hui ils adhèrent à FORTRAN et s'échangent leurs programmes, bugs inclus. »
- « À propos des langages : il est impossible de tailler un crayon avec une hache émoussée. Il est vain d'essayer, à la place, de le faire avec dix haches émoussées. »

Algorithme de BELLMAN

Attention : valide uniquement si le graphe est sans circuit

Numéroter les sommets du graphe;
$\lambda[0] \leftarrow 0$; $P[0] \leftarrow 0$;
pour $j \leftarrow 1$ à n-1 **faire {**
  $P[j] \leftarrow -1$; $\lambda[j] \leftarrow \infty$;
  pour chaque prédécesseur i de j **faire {**
    si ($\lambda[j] > \lambda[i]+v_{ij}$) **alors {**
      $\lambda[j] \leftarrow \lambda[i]+v_{i,j}$;
      $P[j] \leftarrow i$;
    }
  }
}



Algorithme de Bellman

- le graphe est sans circuit donc sans circuit absorbant

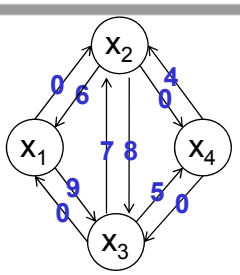
- Démonstration de l'algorithme par récurrence :

Invariant:
A la fin de l'itération **j**, $\lambda[j] = \lambda^*_j$.

- **Complexité** : O(m)

Méthode matricielle

```
pour i←1 à n faire {
    pour j←1 à n faire {
        si (j∈U+(i)) alors V0[i][j] ← vij sinon V0[i][j] ← ∞;
    }
}
pour k←1 à n faire {
    pour i←1 à n faire {
        pour j←1 à n faire {
            Vk[i][j] ← min(Vk-1[i][j] , Vk-1[i][k]+Vk-1[k][j])
        }
    }
}
```


```
pour i←1 à n faire {
    pour j←1 à n faire {
        si (j∈U+(i)) alors V0[i][j] ← vij
        sinon V0[i][j] ← ∞;
    }
}
pour k←1 à n faire {
  pour i←1 à n faire {
    pour j←1 à n faire {
      Vk[i][j] ← min(Vk-1[i][j] , Vk-1[i][k]+Vk-1[k][j])
    }
  }
}
```

	1	2	3	4
1	0	0	9	∞
2	6	0	8	0
3	0	7	0	5
4	∞	4	0	0

	1	2	3	4
1	0	0	9	∞
2	6	0	8	0
3	0	0	0	5
4	∞	4	0	0

	1	2	3	4
1	0	0	8	0
2	6	0	8	0
3	0	0	0	0
4	10	4	0	0

	1	2	3	4
1	0	0	8	0
2	6	0	8	0
3	0	0	0	0
4	0	0	0	0

	1	2	3	4
1	0	0	0	0
2	0	0	0	0
3	0	0	0	0
4	0	0	0	0

Méthode matricielle

- L'absence de circuit absorbant permet de se limiter à la recherche de chemins élémentaires de valeur minimale.

- Démonstration de l'algorithme par récurrence :

Invariant:

A la fin de l'itération **k**, **$V^k[j][j]$** est la valeur minimale d'un chemin élémentaire allant de **i** à **j** et ne passant que par les sommets **{1, 2, …, k}+{i, j}.**

- **Complexité** : $O(n^3)$

Méthode matricielle en pratique

```
pour i←1 à n faire {
   pour j←1 à n faire {
        si (j∈U+(i)) alors V[i][j] ← v_ij sinon V[i][j] ← ∞;
   }
}
pour k←1 à n faire {
   pour i←1 à n faire {
        pour j←1 à n faire {
              V^tmp[i][j] ← min(V[i][j]  , V [i][k]+V[k][j])
        }
   }
   V← V^tmp;
}
```

Autres problèmes de cheminement

- Chemins de valeur maximale: FORD et BELLMAN sont valides en remplaçant **min** par **max**, en initialisant les λ_i à $-\infty$ et λ_0 à 0.
- Existence d'un chemin entre i et j : méthode matricielle ou puissance d'une matrice d'adjacence (CF TD)
- Dénombrement du nombre de chemins entre i et j.
- Chemin de capacité maximale qui est la plus petite des valuations du chemin. FORD et BELLMAN sont utilisables en remplaçant **+** par **max** et **min** par **max** et en initialisant les λ_i à -1 et λ_0 à ∞.

Chemins de fiabilité maximale

- Dans une région en proie à un conflit ou une catastrophe naturelle, un convoi doit être acheminé d'une ville s à une ville t.
- Le réseau routier est donné par un graphe valué:
  - les sommets : l'ensemble des villes
  - Les arcs : les tronçons de route
  - La valuation d'un arc (i,j) est la probabilité de parcourir sans dommage la route de i à j.
- On souhaite déterminer un itinéraire maximisant la probabilité pour le convoi d'arriver sans dommage en t.
- La valeur d'un chemin = le produit des valuations de ses arcs : adaptation des algorithmes ou prendre le log des valuations.

Le problème du caboteur

- Une compagnie maritime veut définir une ligne côtière desservant cycliquement N ports.
- On connaît pour chaque voyage d'un port x à un port y:
  - le bénéfice b(x,y);
  - la durée du trajet d(x,y).
- La compagnie souhaite maximiser le bénéfice par unité de temps, égal pour un parcours P au bénéfice cumulé en suivant P, divisé par la durée de P.
- Le problème consiste donc à trouver un cycle P maximisant ce rapport.
- La valeur du chemin est la valeur moyenne ou moyenne pondérée des chemins.

Algorithme de DIJKSTRA

Attention : valide uniquement si les valuations sont ≥ 0

```
λ[0] ←0; visité[0] ←vrai; P[0] ←0;
pour i←1 à n-1 faire { λ[i] ← ∞; visité[i] ← faux; P[i] ← -1; }
pour chaque successeurs j de 0 faire { P[j] ← 0; λ[j] ← v0,j ; }
nb_visités←1;
tant que (nb_visités < n){
    min ← -1;
    pour i←1 à n-1 faire {
        si (visité[i]=faux et (min=-1 ou λ[i] < λ[min] ) alors min ← i;
    }
    visité[min] ←vrai;  nb_visités← nb_visités +1;
    pour chaque successeurs j de min faire {
        si (visité[j]=faux et λ[j]> λ[min]+vij) alors {
                λ[j] ← λ[min]+vmin,j;
                P[j] ←min;
        }
    }
}
```

Algorithme de DIJKSTRA modifié

```
pour i←0 à n-1 faire {
    λ[i] ← ∞; visité[i] ← faux; P[i] ← -1;
}
λ[0] ←0; P[0] ←0; nb_visités←0; insérer(TAS,(0,0));

tant que (nb_visités < n){
    (min,v_min) ← extraire_min_tas(TAS);
     λ[min] ←v_min;
    nb_visités← nb_visités +1;
    pour chaque successeurs j de min faire {
          si (λ[j]> λ[min]+v_min,j) alors {
                λ[j] ← λ[min]+v_min,j;
                P[j] ←min;
                rétablir_tas(TAS,j);
          }
    }
}
```