PLAN

- Généralités sur les graphes
- Représentation d'un graphe en machine
- Parcours dans les graphes
- Arbre recouvrant
- **Plus court chemin dans un graphe**
- Coloration d'un graphe
- Graphes planaires
- Flots et réseaux de transports
- Analyse des réseaux d'interactions

EXEMPLE DE PROBLÉMATIQUE

- **Trouver le plus court trajet (en temps) entre 2 lieux**

  Modélisation du réseau des transports parisiens par un graphe

EXEMPLE DE PROBLÉMATIQUE

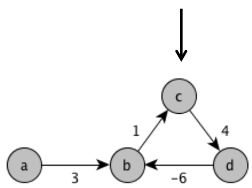
- **Trouver le plus court trajet (en temps) entre 2 lieux**

PRINCIPE

- Soit G=(X,U,l) un graphe orienté et valué.
  l(.) est une fonction qui associe à tout arc une valeur réelle.

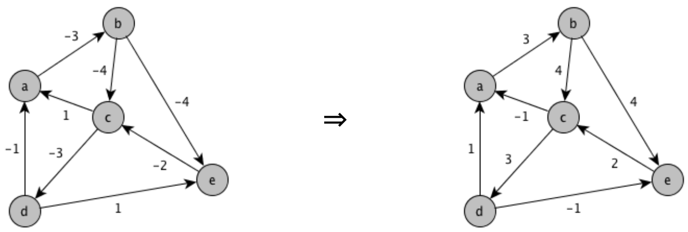
- **Objectif**
  Déterminer, parmi l'ensemble de tous les chemins reliant « i » à « j », le chemin μ qui rend la longueur totale $l(\mu)=\sum_{u\in\mu} l(u)$ minimale.

- **Théorème**
  Ce problème a une solution si et seulement si il n'existe pas de **circuit absorbant** (de longueur négative) entre « i » et « j ».



PLUS LONG CHEMIN

- Si on recherche les chemins les plus longs dans le graphe G=(X,U,l), il faut chercher les chemins les plus courts dans le graphe G'=(X,U,-l).

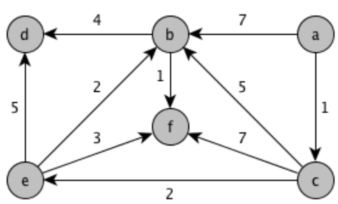


ALGORITHME DE DIJKSTRA

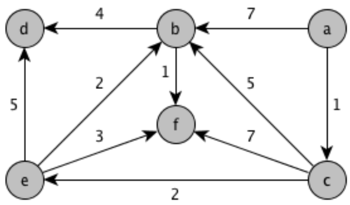
- Cas où les valuations sont **toujours positives** ($\forall u \in U$, $l(u) \geq 0$).

  L'algorithme de Dijkstra permet de déterminer la longueur du plus court chemin d'un sommet « s » donné à tous autres sommets du graphe.

  L'algorithme retourne un tableau $\lambda(i)$ indiquant la longueur du plus court chemin de « s » à « i ».



ALGORITHME DE DIJKSTRA




```
Dijkstra(G=(X,U,l),s,λ) ;
{     Z = X-{s} ;
      λ(s)=0 ;
      pour tout i∈Z faire
      {     si  (s,i)∈U  faire λ(i)=l((s,i)) ;
            sinon  λ(i)=+∞ ;
      }
      tant que Z≠∅ faire
      {     prendre  x∈Z  tel que λ(x)=min{ λ(j) / j∈Z } ;
            Z = Z-{x} ;
            pour tout ( i∈Γ+(x) et i∈Z ) faire
            {     si  λ(x)+l((x,i))<λ(i)  faire  λ(i)=λ(x)+l((x,i)) ;
            }
      }
}
```

$\boxed{O(n^2)}$

ou moins …

ALGORITHME DE DIJKSTRA

- La complexité dépend de la manière d'implémenter l'accès au sommet qui possède la plus petite valeur de λ.

  En utilisant un **tas de Fibonacci** ou une **skip-list**, la complexité de l'algorithme de Dijkstra passe en **O**(m+nlog(n)).

Opérations	Binomial Heap	Binary Heap	Fibonacci Heap
MakeHeap	$O(1)$	$O(1)$	$O(1)$
Insert	$O(log(n))$	$O(log(n))$	$O(1)$
Minimum	$O(log(n))$	$O(1)$	$O(1)$
ExtraitMin	$O(log(n))$	$O(log(n))$	$O(log(n))$
DecreaseKey	$O(log(n))$	$O(log(n))$	$O(1)$
Delete	$O(log(n))$	$O(log(n))$	$O(log(n))$
Union	$O(log(n))$	$O(n)$	$O(1)$

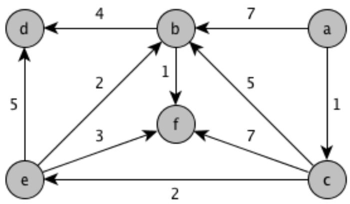
- Dans le cas de graphes planaires à valuations positives, il existe des algorithmes de construction de structures particulières ou de décompositions récursives du graphe qui permettent d'améliorer cette complexité :

  Frederickson (1986) : **O**(n√log(n)) grâce à la décomposition

  Henzinger (1997) : **O**(n) grâce à la décomposition

  …

ALGORITHME DE DIJKSTRA

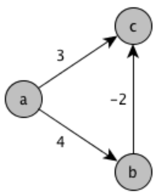



```
Dijkstra(G=(X,U,l),s,λ,pere) ;
{    Z = X-{s} ;
     λ(s)=0 ;
     pour tout i∈Z faire
     {    si (s,i)∈U faire { λ(i)=l((s,i)) ; pere(i)=s ; }
          sinon λ(i)=+∞ ;
     }
     tant que Z≠∅ faire
     {    prendre x∈Z tel que λ(x)=min{ λ(j) / j∈Z } ;
          Z = Z-{x} ;
          pour tout ( i∈Γ+(x) et i∈Z ) faire
          {    si λ(x)+l((x,i))<λ(i) faire { λ(i)=λ(x)+l((x,i)) ; pere(i)=x ; }
          }
     }
}
```


PLUS COURT CHEMIN

ALGORITHME DE DIJKSTRA

- Montrer que l'algorithme de Dijkstra donne un résultat faux avec des valuations négatives.

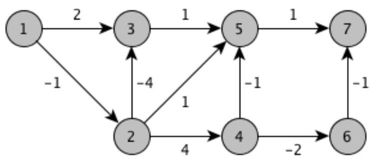


ALGORITHME DE BELLMAN

- Cas où le graphe est **sans circuit**.

  L'algorithme de Bellman permet de déterminer la longueur du plus court chemin du sommet « 1 » (de la numérotation topologique) à tous autres sommets du graphe.

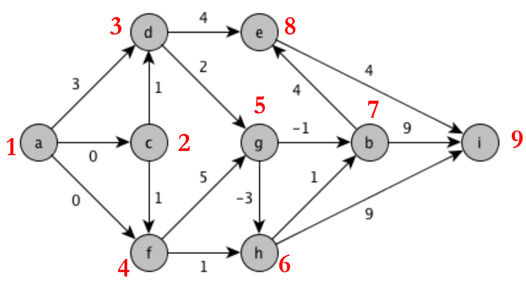
  L'algorithme retourne un tableau $\lambda(i)$ indiquant la longueur du plus court chemin de « 1 » à « i ».



PLUS COURT CHEMIN

ALGORITHME DE BELLMAN

- **Bellman**(G=(X,U,l),λ,pere) ;
  {   Réaliser une numérotation topologique du graphe ;
      $\lambda(1)=0$;
      pour i=2 **à** n **faire**
      { $\lambda(i) = \min\{\lambda(j) + l((j,i)) \ / \ j \in \Gamma^{-}(i)\}$;
          pere(i) = prédécesseur j qui a permis de fixer $\lambda(i)$;
      }
  }



O(n+m)

ALGORITHME DE FORD

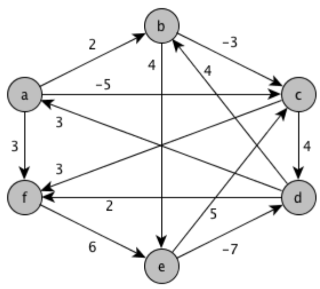
- **Aucune contrainte** sur le graphe.

Principe
A chaque étape k et pour chaque sommet « i », l'algorithme cherche les plus courts chemins de longueur k entre « s » et « i ».

L'algorithme retourne un tableau $\lambda(k,i)$ indiquant la longueur du plus court chemin de « s » à « i » à l'étape k.

Si à l'étape k, pour tout $i \in X-\{s\}$, la longueur du chemin entre « s » et « i » est la même qu'à l'étape k-1, alors le graphe est sans circuit absorbant; le problème est résolu.

ALGORITHME DE FORD



- **Ford**(G=(X,U,l),s,λ,pere) : boolean ;
{ Test = vrai ;
$\lambda(0,s)=0$;
pour tout $i \in X-\{s\}$ **faire** $\lambda(0,i)=+\infty$;
k = 1 ;
tant que (($k \leq n$) **et** (Test)) **faire**
{ $\lambda(k,s)=0$;
pour tout $i \in X-\{s\}$ **faire**
{ $\lambda(k,i) = \min\{\lambda(k-1,j) + l((j,i)) \;/\; j \in \Gamma^{-}(i)\}$;
si le minimum est obtenu grâce au sommet x **alors** pere(i) = x ;
}
si $\forall i$ ($\lambda(k,i) = \lambda(k-1,i)$) **faire** Test = faux ;
k = k+1 ;
}
Return(Test) ;
}

$\boxed{O(nm)}$

ALGORITHME A*

- **Objectif**
  Déterminer, le plus court chemin entre une source « s » et une destination « p ».

- **Principe**
  Se déplacer prioritairement « en direction » de la destination

- **Coût**
  Le coût d'un sommet x =
  longueur du meilleur chemin de s à x + estimation du cout de x à p.

  ⇒ imaginer une heuristique propre au problème modélisé.

- Variante de Dijsktra qui permet de limiter la recherche du sommet suivant aux sommets « potentiellement intéressants ».