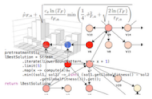
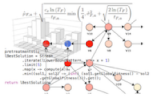


PLAN

1. Introduction
2. Généralités sur les graphes
3. Représentation d'un graphe en machine
4. Parcours dans les graphes
5. Arbre recouvrant
6. **Plus court chemin dans un graphe**
7. Coloration d'un graphe
8. Graphes planaires
9. Flots et réseaux de transports
10. Réseaux d'interactions



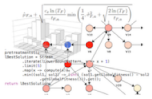
OPTIMISER DES DÉPLACEMENTS

Développer des stratégies efficaces pour résoudre des problèmes de cheminement dans le métro Parisien, comme :

Trouver le « meilleur » trajet entre deux stations

Trouver le meilleur itinéraire de visite

Trouver le meilleur emplacement pour minimiser les déplacements

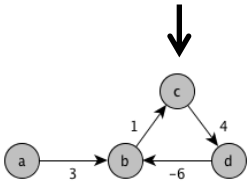


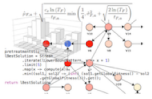
PRINCIPE

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.

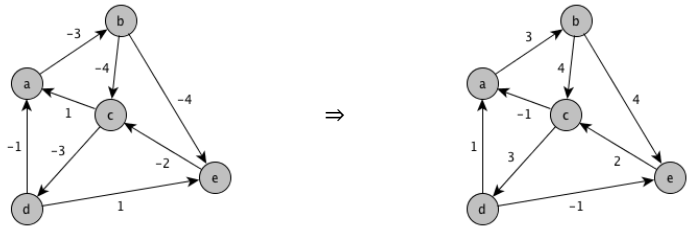
Théorème
Ce problème a une solution si et seulement si il n'existe pas de circuit absorbant (de longueur négative) entre « i » et « j ».





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).

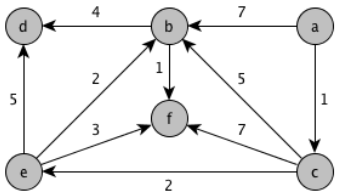


ALGORITHME DE DIJKSTRA

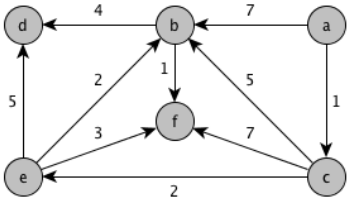
Cas où les valuations sont toujours positives ($\forall u \in U, l(u) \geq 0$).

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 ».



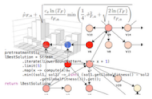
ALGORITHME DE DIJKSTRA



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)) ;
}
}
}

$O(n^2)$

ou moins …



ALGORITHME DE DIJKSTRA

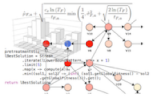
La complexité dépend de la manière d'implémenter l'accès au sommet qui possède la plus petite valeur de λ.

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

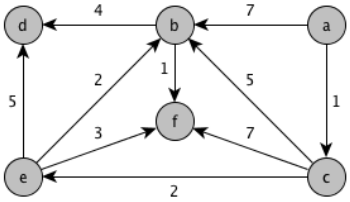
Opérations	Binomial Heap	Binary Heap	Fibonacci Heap
MakeHeap	$O(1)$	$O(1)$	$O(1)$
Insert	$O(log(n))$	$O(log(n))$	$O(1)$
Minimum	$O(log(n))$	$O(1)$	$O(1)$
ExtraitMin	$O(log(n))$	$O(log(n))$	$O(log(n))$
DecreaseKey	$O(log(n))$	$O(log(n))$	$O(1)$
Delete	$O(log(n))$	$O(log(n))$	$O(log(n))$
Union	$O(log(n))$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

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 :

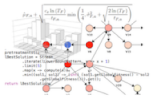
Frederickson (1986) : **O**(n√log(n)) grâce à la décomposition
Henzinger (1997) : **O**(n) grâce à la décomposition
…



ALGORITHME DE DIJKSTRA

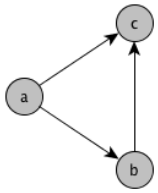


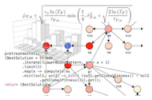
Dijkstra(G=(X,U,l),s,λ,pere) ;
{ Z = X-{s} ;
λ(s)=0 ;
pour tout i∈Z **faire**
{ **si** (s,i)∈U **faire** { λ(i)=l((s,i)) ; pere(i)=s ; }
sinon λ(i)=+∞ ;
}
tant que Z≠∅ **faire**
{ prendre x∈Z tel que λ(x)=min{ λ(j) / j∈Z } ;
Z = Z-{x} ;
pour tout (i∈Γ^+(x) et i∈Z) **faire**
{ **si** λ(x)+l((x,i))<λ(i)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



ALGORITHME DE DIJKSTRA

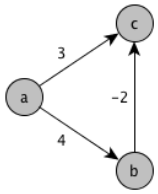
Montrer que l'algorithme de Dijkstra donne un résultat faux avec des valuations négatives.

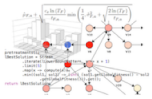




ALGORITHME DE DIJKSTRA

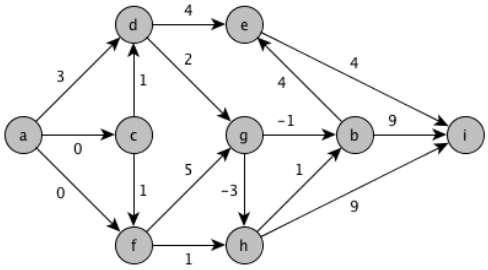
Montrer que l'algorithme de Dijkstra donne un résultat faux avec des valuations négatives.

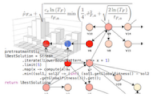




ALGORITHME DE DIJKSTRA

Peut-on transformer le graphe ci-dessous de manière à ne plus avoir de valuations négatives afin de pouvoir utiliser l'algorithme de Dijkstra ?



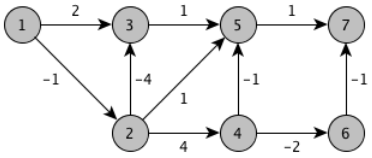


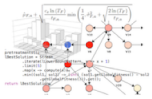
ALGORITHME DE BELLMAN

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.

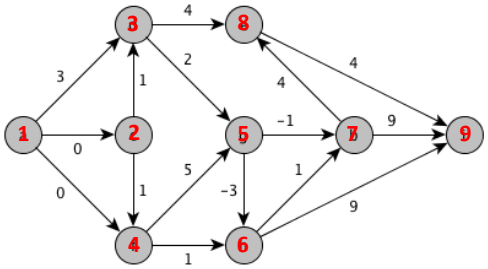
L'algorithme retourne un tableau λ(i) indiquant la longueur du plus court chemin de « 1 » à « i ».



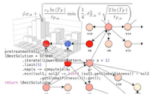


ALGORITHME DE BELLMAN

Bellman(G=(X,U,l),λ,pere) ;
{ Réaliser une numérotation topologique du graphe ;
λ(1)=0 ;
pour i=2 **à** n **faire**
{ λ(i) = min{λ(j) + l((j,i)) / j∈Γ⁻(i)} ;
pere(i) = prédécesseur j qui a permis de fixer λ(i) ;
}
}



O(n+m)



ALGORITHME DE FORD

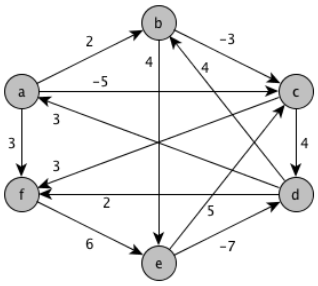
Aucune contrainte sur le graphe.

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 λ(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∈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 ;
      λ(0,s)=0 ;
      pour tout i∈X-{s} faire    λ(0,i)=+∞ ;
      k = 1 ;
      tant que ((k ≤ n) et (Test)) faire
      {     λ(k,s)=0 ;
            pour tout i∈X-{s} faire
            {     λ(k,i) = min{λ(k-1,j) + l((j,i)) / j∈Γ⁻(i)} ;
                  si le minimum est obtenu grâce au sommet x alors pere(i) = x ;
            }
            si ∀i (λ(k,i) = λ(k-1,i)) faire Test = faux ;
            k = k+1 ;
      }
      Return(Test) ;
}
```

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 ».