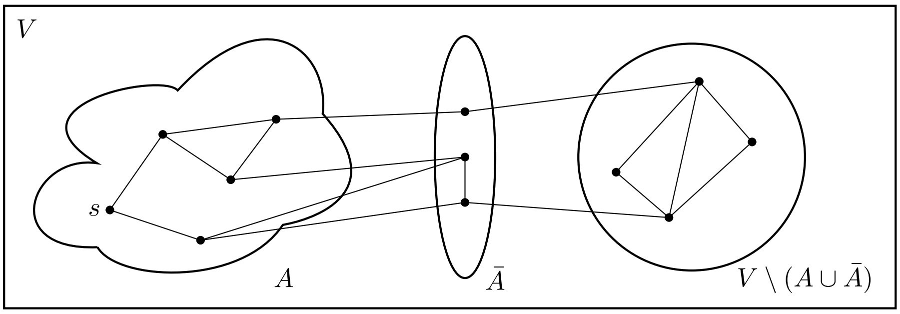
TD d'Algorithmique et structures de données
Implémentations de l'algorithme de Dijkstra

Équipe pédagogique Algo SD

L'algorithme de Dijkstra est un algorithme *glouton* qui prend en entrée un graphe pondéré $G = (V, E, w)$ avec poids positifs et un nœud initial $s \in V$ et qui renvoie le plus court chemin de s vers tout autre nœud $v \in V \setminus \{s\}$ du graphe. Durant l'exécution de l'algorithme, on garde en mémoire un ensemble A de nœuds pour lesquels le plus court chemin depuis s est déjà calculé, et un ensemble $\bar{A}$ de nœuds directement atteignables depuis A, i.e., pour tout nœud $v \in \bar{A}$ il existe un nœud $u \in A$ tel que l'arc (u, v) est dans E. A chaque étape, l'algorithme étend l'ensemble A en déplaçant le nœud de *plus grande priorité* de $\bar{A}$ vers A. La priorité est ici définie comme la *plus petite distance estimée* vers un nœud de $\bar{A}$ depuis s. A la fin de chaque étape, de nouveaux nœuds sont possiblement ajoutés à $\bar{A}$, tandis que les distances estimées vers les nœuds de $\bar{A}$ sont mises à jour.



Une description formelle de l'algorithm est la suivante.

Dijkstra(graph $G = (V, E)$, vertex s)

$\{$*distance*(v) : the current distance of the vertex v ; it can be stored in the struct Node$\}$
$\{$*predecessor*(v) : the predecessor of v in the shortest path from $s\}$
$\{$Initialization$\}$
$A := \emptyset$; $\bar{A} := \{s\}$;
distance$(s) := 0$; *predecessor*$(s) :=$ *null* ;
for each vertex $v \in V \setminus \{s\}$ **do**
   distance$(v) := +\infty$;
   predecessor$(v) :=$ *null* ;
end for
while $\bar{A} \neq \emptyset$ **do**
   $\{$Select the highest priority vertex in $\bar{A}\}$
   Let $u \in \bar{A}$ be the vertex with $distance(u) = \min\{distance(v) : v \in \bar{A}\}$;
   $A := A \cup \{u\}$;
   $\bar{A} := \bar{A} \setminus \{u\}$;
   for each vertex $v \in V \setminus A$ such that $(u, v) \in E$ **do**
      $\{$If needed, update the distance (priority) of a vertex$\}$
      if $distance(v) > distance(u) + w_{u,v}$ **then**
         if $distance(v) = \infty$ **then**
            $\bar{A} := \bar{A} \cup \{v\}$;
         end if
         $distance(v) := distance(u) + w_{u,v}$;
         $predecessor(v) := u$;

```
    end if
  end for
end while
```

1. Quelle est la complexité de l'algorithme de Dijkstra en fonction de la taille de G et des coûts des opérations dans $\bar{A}$?

1 Listes chaînées

L'algorithme initialement proposé par Dijkstra utilise une liste chaînée (non triée) pour représenter l'ensemble $\bar{A}$.

1. Quelle est la complexité de l'algorithme de Dijkstra dans ce cas ?

2 Implémentation de Dial

1. Montrer qu'à chaque itération, la distance de s vers le nouveau nœud ajouté à A est supérieure ou égale à la distance de s vers le nœud ajouté à A lors de la précédente itération.

   Soit W le poids maximum sur tous les arcs, i.e., $W = \max\{w_{u,v} : (u, v) \in E\}$. La distance de s vers tout nœud est donc au plus $(|V| - 1)W$ (au pire on passe par tous les nœuds).

   Soit un tableau c de taille $(|V| - 1)W + 1$. Chaque indice i du tableau contient une liste chaînée des nœuds dont la distance courante depuis s est égale à i (on part de l'indice 0).
2. Implémenter l'algorithme de Dijkstra en utilisant c (indication : ne pas utiliser les ensembles A et $\bar{A}$ pour l'instant).
3. Quelle est la complexité en temps et en place mémoire de cet algorithme ?

   Revenons maintenant à l'utilisation des ensembles A (nœuds déjà parcourus) et $\bar{A}$ (nœuds voisins des nœuds de A).
4. Montrer qu'à la fin de chaque itération de la boucle `Tant que` dans l'algorithme de Dijkstra (algorithme initial), on a $\max\{distance(v) : v \in \bar{A}\} - \min\{distance(v) : v \in \bar{A}\} \leq W$.
5. A partir de l'observation précédente, comment peut-on réduire la place mémoire nécessaire à l'algorithme ? Proposer une structure de données pour remplacer le tableau c.

3 Implémentation par tas

1. Implémenter l'algorithme de Dijkstra en utilisant un tas binaire pour représenter $\bar{A}$.
2. Quelle est la complexité en temps de cet algorithme ?

   A la place d'un tas binaire, on pourrait aussi utiliser un d-tas, $d \geq 2$, c'est-à-dire un tas où chaque nœud a jusqu'à d fils.
3. Quelle est la complexité de `insert`, `extract_max` et `increase_priority` pour un d-tas ?
4. Quelle est la complexité de l'algorithme de Dijkstra si on utilise un d-tas ?
5. Quelle valeur de d donne la meilleure complexité ?