TD5 : Algorithme de Dijkstra

1 Application de l'algorithme

L'algorithme de Dijkstra est l'un des algorithmes les plus célèbres permettant de calculer des plus courts chemins dans les graphes. Cet algorithme est adapté pour connaître les plus courts chemins depuis un nœud donné (la source) vers tous les autres nœuds du graphe. Attention, la garantie que les chemins obtenus soient les plus courts ne tient que si tous les poids des arêtes du graphe sont positifs. L'algorithme de Dijkstra peut être décrit par :

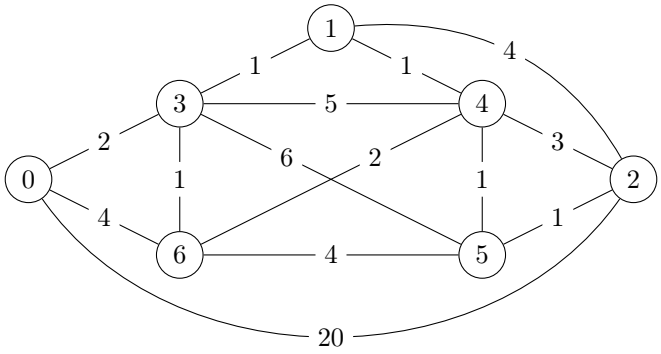
données : G : un graphe, s un sommet source
résultat : une distance et un nœud de provenance pour chaque nœud de G
Algorithme
- **pour chaque** nœud v de G **faire**
  - $d_v \leftarrow \infty$ // distance de s à v
  - $p_v \leftarrow \emptyset$ // prédécesseur de v dans le chemin minimal depuis s
- $d_s \leftarrow 0$ // distance de s à s
- $p_s \leftarrow \emptyset$ // prédécesseur de s dans le chemin depuis s
- ajouter s dans la File avec la priorité 0
- **tant que** la File n'est pas vide **faire**
  - $v \leftarrow$ un élément de la File ayant une distance depuis s minimale
  - retirer v de la File
  - **pour chaque** nœud w voisin de v **faire**
    - $d_{v,w} \leftarrow$ poids de l'arête de v à w
    - **si** $d_w > d_v + d_{v,w}$ **alors**
      - $d_w \leftarrow d_v + d_{v,w}$ // mise à jour de la distance de s à w
      - $p_w \leftarrow v$ // mise à jour du prédécesseur de w dans le chemin depuis s
      - **si** w n'est pas dans la File **alors**
        - ajouter w dans la File avec la priorité d_w
      - **sinon**
        - mettre à jour la priorité de w dans la File avec d_w

Notez que dans cet algorithme, la gestion de la File n'est pas triviale. En termes de type abstrait, les requêtes nécessaires sont :

— insérer un élément avec une distance donnée ;
— retirer un élément de distance minimale ;
— modifier à la baisse la distance d'un élément présent ;
— déterminer si un élément est présent ou non dans la file.

Ce type abstrait est un classique, dénommé « File à priorité ». Plusieurs structures de données ont été proposées pour le réaliser la plus célèbre étant le « Tas binaire ». Aucune de ces structures n'offre de complexité en temps constant sur toutes les requêtes.

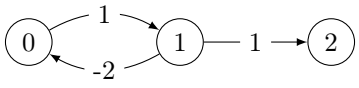
a) Appliquez l'algorithme de Dijkstra sur le graphe suivant pour déterminer le plus court chemin depuis le nœud 0 vers tous les autres nœuds.



2 Limitations

2.1 Définition du problème

b) Que dire du graphe suivant ? Pouvez vous calculer le plus court chemin de 0 vers 2 dans ce cas ?

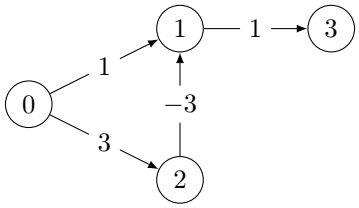


c) Que se passera-t-il si vous appliquez l'algorithme de Dijkstra sur ce graphe ?

2.2 Gestion de la file avec des poids négatifs

L'algorithme proposé n'interdit pas à un nœud qui est passé dans la file à priorité d'y retourner par la suite.

d) Quels seraient les plus courts chemin calculés par Dijkstra à partir du nœud 0 dans le graphe suivant si un nœud n'était pas autorisé à retourner dans la file une seconde fois ?



3 Pire cas pour des poids positifs

Pour étudier la complexité de l'algorithme de Dijkstra, nous nous placerons dans le cas où la structure de donnée utilisée pour la file à priorité est un tas binaire. Dans ce cas, les bornes maximales de complexité des opérations sont les suivantes :

— insertion d'un nouveau nœud dans la file : $O(\log n)$;
— retrait du nœud ayant la distance la plus faible : $O(\log n)$;
— diminution de la distance d'un nœud : $O(\log n)$.

Pour déterminer si un nœud est présent ou non dans la file, chaque nœud est marqué par un booléen qui est mis à vrai lors d'une insertion et à faux lors d'un retrait. L'opération peut donc être réalisée en temps constant.

e) Élaborez des graphes à trois, quatre, cinq sommets et des ensembles de poids positifs pour leurs arêtes demandant le plus d'opérations possibles à l'algorithme.

f) Si n est le nombre de sommets du graphe, et m le nombre d'arêtes, pouvez-vous écrire une borne maximale de complexité pour l'algorithme de Dijkstra ?

g) En autorisant des arêtes de poids négatif, mais pas de cycles négatifs, quelle est la pire complexité que vous pensez pouvoir obtenir ?