6.3 Parcours en largeur

6.3.1 Motivation

On s'intéresse à la problématique du plus court chemin dans un graphe : trouver un itinéraire pour aller de la pharmacie à la boulangerie, réaliser mat en le moins de coups possibles aux échecs, etc.

Justifions que le plus court chemin est une notion bien définie. Soit $s, t \in S$. S'il existe un chemin de s à t alors

$$L_{s,t} = \{\text{longueur de } c \mid c \text{ chemin de } s \text{ à } t\} \subseteq \mathbb{N}$$

est non vide et admet un minimum $\ell_{s,t}$. Sinon, $\ell_{s,t} = +\infty$. On appelle plus court chemin un chemin c de s à t telle que la longueur de c est égale à $\ell_{s,t}$.

6.3.2 Algorithme

L'algorithme suivant calcule la plus courte distance depuis une source s à tout nœud t. Le calcul est stocké dans $d[t]$. $pred[t]$ est le prédécesseur dans un plus court chemin depuis s à t. On utilise une file F pour stocker les nœuds à traiter.

Algorithme 23 : Parcours en largeur

Entrées : Un graphe G et un sommet s

```
1  pour t ∈ S faire
2      d[t] := +∞
3  d[s] := 0
4  pred[s] := []
5  F :=créer_file([s])
6  tant que F ≠ ∅ faire
7      s' := F.défiler
8      pour t tel que s' → t faire
9          si d[t] = +∞ alors
10             d[t] := d[s'] + 1
11             enfile(F, t)
12             pred[t] := s'
```

6.3.3 Terminaison et complexité

Théorème. *Chaque sommet est enfilé au plus une fois.*

Démonstration.
Invariant : un sommet t qui a été dans la file est tel que $d[t]$ est fini. ■

Théorème. *On a une complexité en $O(|S| + |A|)$.*

Démonstration.
Soit E l'ensemble des sommets qui ont été dans la file et qui n'y sont plus. Au début $E = \emptyset$. A chaque passage de boucle « tant que », E gagne un élément et $card(E)$ croit strictement de 1. $card(E)$ est majoré par $card(S)$. Donc il y a $O(|S|)$ passages dans la boucle « tant que ».

La boucle « pour » coûte $card(t \mid s' \to t)$. Les s' sont tous différents à chaque fois (c'est l'élément qui entre dans E) donc tout le travail effectué dans tous les passages de la boucle « pour » coûte $\Sigma_{s' \in S} card(t \mid s' \to t) = O(|A|)$. ■

6.3.4 Correction

Théorème. *À la fin de l'algorithme, pour tout sommet t, $d[t]$ est la longueur d'un plus court chemin de s à t s'il en existe un et $+\infty$ sinon.*

Démonstration.

Notons $\delta(t)$ la longueur d'un plus court chemin de s à t s'il en existe un et $\delta(t) = +\infty$ sinon.

1) On a l'invariant suivant : $\delta \leqslant d$ (on montre l'initialisation et la conservation comme d'habitude).

2) Posons $\mathcal{P}(n)$:

> « Il y a un moment dans l'exécution tel que, pour tout $t \in S$,
>
> 1. $\delta(t) \leqslant n \Rightarrow d[t] = \delta(t)$
> 2. $\delta(t) > n \Rightarrow d[t] = +\infty$
> 3. $\delta(t) = n$ ssi t est dans la file F, notée F_n ».

Montrons $\mathcal{P}(n)$ pour tout n par récurrence sur n.

— Pour $n = 0$, le moment est juste avant de commencer l'exécution de la boucle « tant que ». D'où $\mathcal{P}(0)$.

— Supposons $\mathcal{P}(n)$ vraie. Et montrons $\mathcal{P}(n+1)$. Au moment correspondant à $\mathcal{P}(n)$, la file contient exactement les sommets à distance n. Soit k le nombre des sommets à distance n. Pour $\mathcal{P}(n+1)$, on considère le moment où l'on a exécuter k tour de boucle « tant que » supplémentaire. La file contient alors des successeurs des k sommets à distance n.

1. Soit $t \in S$ tel que $\delta(t) \leqslant n+1$.
   — Si $\delta(t) \leqslant n$, $d[t] = \delta(t)$ par $\mathcal{P}(n)$, 1. et $d[t]$ est inchangé.
   — Sinon $\delta(t) = n+1$. Notons c un plus court chemin de s à t et u le prédécesseur de t dans c. Le sous-chemin $s \to u$ de c reste un plus court chemin et $\delta(u) = n$. Par $\mathcal{P}(n)$ 3., u est F_n. Par $\mathcal{P}(n)$ 1., on a $d[u] = \delta(u)$ .Par $\mathcal{P}(n)$ 2., on avait bien $d[t] = +\infty$. Donc, la mise à jour de $d[t]$ a lieu avec $d[t] = d[u] + 1$, on a $d[t] = d[u] + 1 = \delta(u) + 1 = n + 1$.
2. Soit $t \in S$ tel que $\delta(t) > n+1$. Montrons que $t \notin F_{n+1}$. Par l'absurde, si $t \in F_{n+1}$, $d[t]$ aurait été mis à jour par $d[u] + 1$ où $u \in F_n$. Par $\mathcal{P}(n)$ 1. et 3., $d[u] = \delta(u) = n$. On aurait donc $d[t] = n + 1$ et donc $\delta(t) \leq n + 1$. Contradiction avec l'invariant $\delta \leq d$. Donc, $t \notin F_{n+1}$ Donc $d[t]$ n'a pas été mis à jour et $d[t] = +\infty$.
3. Montrons $\delta(t) = n + 1$ ssi t est dans F_{n+1}.
   $\delta(t) = n + 1$ implique par le point 1 que l'on vient de montrer (deuxième tiret) que t est mis à jour entre le moment $\mathcal{P}(n)$ et $\mathcal{P}(n+1)$. Il est donc enfilé et est donc dans F_{n+1}.
   Réciproquement, si t est mis à jour entre le moment $\mathcal{P}(n)$ et $\mathcal{P}(n+1)$, alors la mise à jour était de la forme $d[t] := d[u] + 1$ où $u \in F_n$. Mais alors, par $\mathcal{P}(n)1.$, $d[u] = \delta(u)$ et $\mathcal{P}(n)3.$, $\delta(u) = n$. Donc $d[t] = n + 1$ implique $\delta(t) = n + 1$ par le point 1.

D'où $\mathcal{P}(n+1)$.

Conclusion On a montré par récurrence $\mathcal{P}(n)$ pour tout $n \in \mathbb{N}$. Soit $t \in S$. Si $d[t] < +\infty$, on applique $\mathcal{P}(\delta[t])$ et on $d[t] = \delta(t)$. Si $\delta[t] = +\infty$, on applique l'invariant $\delta \leq d$ et on a $d[t] = +\infty$.

■

6.4 Algorithme de Dijkstra

Dans les jeux, on s'intéresse au nombre minimum de coup à jouer. Mais il arrive que les coups aient des coûts (en euros par exemple). Ou lorsque l'on calcule un itinéraire, nous avons

les distances entre les villes. Nous allons alors modéliser le problème avec un graphe pondéré, c'est-à-dire dont les arcs sont étiquetés par des distances.

6.4.1 Adaptation du parcours en largeur

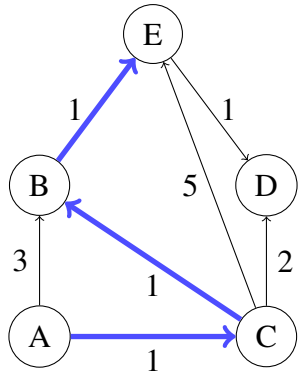
Un graphe pondéré $G = (S, A)$ est un graphe muni d'une fonction de **pondération positive** $w : A \to \mathbb{R}^+$. $w(s \to t)$ s'appelle le poids de l'arc $s \to t$. Le poids d'un chemin est le somme des poids des arcs par lesquels il passe.

Supposons un instant que les poids sont des entiers. Nous sommes ici dans une situation typique où l'algorithme du parcours en largeur fonctionne : il suffit d'ajouter des sommets intermédiaires.

Nous sommes dans une situation où beaucoup d'instants dans l'algorithme sont des instants d'attente. Rien ne sert d'attendre, il suffit.... de mettre des 'alarmes' sur les noeuds. Bref, on utilise une file de priorité.

On s'intéresse aux plus courts chemins à partir d'une origine s où le graphe G est pondéré positivement.

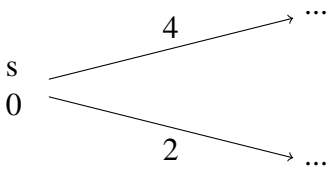
Le dessin suivant montre un plus court chemin de l'origine A vers E.



Adaptons l'algorithme du parcours en largeur. Ici, on peut mettre à jour *plusieurs fois* la valeur $d[t]$ pour un sommet t donné (on rerègle une alarme) alors que pour le parcours en largeur dès que $d[t]$ est affecté, c'est la bonne valeur.

6.4.2 Algorithme

Commençons par dire que pour aller de la source s à la source s, la distance est nulle.



Puis à chaque étape, on considère le sommet non colorié (non traité) dont la distance courante est la plus courte. On le colorie. Puis on met à jour les sommets directement accessibles à partir de celui-ci si le résultat est meilleur.

On a terminé lorsque toute les sommets sont coloriés (traités).

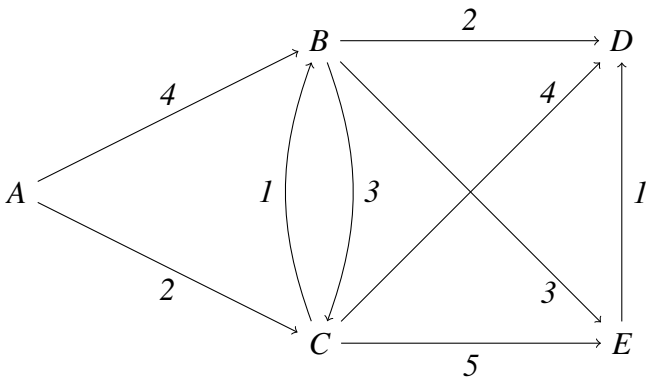
Algorithme 24 : Dijkstra

Entrées : Un graphe G et un sommet s
Sorties : Les longueurs $d[t]$ des plus courts chemins de s à t

```
1  pour t ∈ S faire
2      d[t] := +∞
3  d[s] := 0
4  pred[s] := []
5  F :=créer_file_priorité([s]) (triée avec d)
6  tant que F ≠ ∅ faire
7      u := F.défiler_min
8      pour t tel que u -p-> t faire
9          si d[t] > d[u] + p alors
10             d[t] := d[u] + p
11             pred[t] := u
12             F.mettreAJourDiminution(t)
```

On a besoin d'adapter la file de priorité traditionnelle avec une nouvelle opération : la mise à jour en cas de diminution.

Exemple 32. *Appliquer l'algorithme de Dijkstra sur le graphe suivant :*



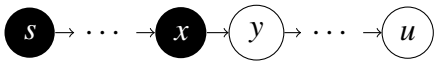
6.4.3 Correction

Théorème.
L'algorithme est correct.

Démonstration.
Comme d'habitude, on appelle $\delta(t)$ la distance minimale de s à t et $+\infty$ s'il n'y a pas de chemin. Comme pour le parcours en largeur, on a le même invariant qui dit que d surestime δ : $\delta \leqslant d$.

On a aussi l'invariant suivant : toutes les distances calculées pour les noeuds déjà traités sont exactement les distances minimales. Formellement, $\forall t \notin F$, $d[t] = \delta(t)$. Montrons le.

- initialisation : Après que s est défilé, on a toujours $d[s] = 0 = \delta(s)$.
- Conservation : Soit u le sommet qui va être défilé ($u \neq s$). S'il n'y a pas de chemin de s à u, alors $+\infty = \delta(u) \leqslant d[u] = +\infty$.
  Sinon, on a un plus court chemin c de s à u. Soit y le premier sommet de c qui est dans F (il existe, au pire il s'agit de u lui-même) et $x \notin F$ son prédécesseur (il a bien un précécesseur car $s \notin F$).



Voici les faits :

— Montrons $d[y] = \delta(y)$. D'après l'invariant, on a $d[x] = \delta(x)$. C'était le cas dès que x a été retiré de la file et lorsque $d[y]$ a été mis à jour. A ce moment là, l'arrête $x \to y$ a été considéré dans l'algorithme. Le sous-chemin de s à y est un plus court chemin, donc $d[y] = \delta(y)$;

— On a $\delta(y) \leq \delta(u)$ car il n'y a que des **poids positifs**.

— On a $\delta(u) \leq d[u]$ car d surestime δ.

— De plus, comme u est sur le point d'être défilé, c'est le minimum de la file donc $d[u] \leqslant d[y]$.

En résumé :

$$d[y] = \delta(y) \leqslant \delta(u) \leqslant d[u] \leqslant d[y]$$

Donc $d[u] = \delta(u)$.

■

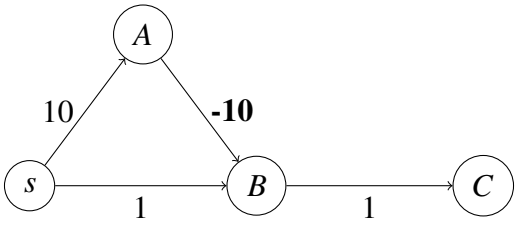
6.4.4 Complexité

Théorème.
L'algorithme effectue $O(|S| + |S||défiler\_min| + |file\_créer| + |A||MàJ|)$.

Remarque.
Avec une implémentation sous forme de tableau, on a une complexité en $O(|S|^2)$.
Sous forme d'un tas, c'est $O((|S| + |A|)\log(|S|))$.
Sous forme d'un tas de Fibonacci, c'est $O(|S|\log(|S|) + |A|)$.

6.4.5 Contre-exemple avec des poids négatifs

Considérons le graphe suivant :



Si on exécute l'algorithme de Djikstra à partir du sommet s, alors la valeur $d[C]$ calculé est 2 alors que le chemin $s \to A \to B \to C$ est de longueur 1.