Programmation & Algorithmique
TP 7
Le plus court chemin

On représente souvent une carte par un graphe, chaque sommet étant un lieu et chaque arête étant une route entre deux lieux. On peut pondérer le graphe pour représenter le temps mis pour aller d'un lieu à un autre.

1 Algorithme de Bellman–Ford

Prenons l'exemple suivant avec monsieur Ford (le constructeur automobile) dans sa belle voiture qui souhaite se rendre à sa boutique B. Il se retrouve devant un carrefour C et se demande s'il n'y a pas un moyen de se rendre à son usine que le chemin direct. Il décide de dérouler l'algorithme de Bellman-Ford (Les mathématiciens) et ce malgré les conducteurs chevronnés qui attendent derrière lui. Cet algorithme consiste à vérifier les chemins entre C et chaque lieu à proximité qu'il connait.
Il connait notamment l'école E de ses enfants, l'appartement A d'un ami et une décharge D.
Posons le graphe suivant :

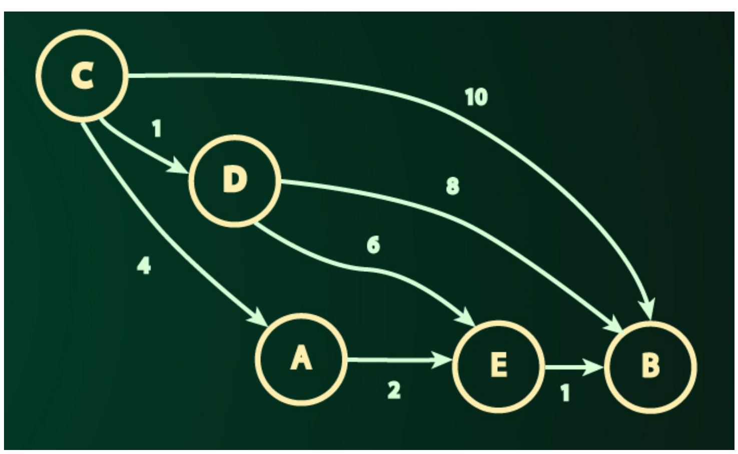

Figure 1 – Graphe des lieux à proximités

1.1 Algorithme

L'algorithme consiste à déterminer, pour chaque sommet s, le prédécesseur qui permet d'accéder le plus rapidement à s.
Pour cela on va regarder tous les chemins possibles en partant de C.

Algorithm 1 Belmann-Ford

to_visit $\Leftarrow$ pile de sommets à visiter
s0 $\Leftarrow$ sommet de depart
distances $\Leftarrow$ tableau des distances minimums en partant de *s0*
previous $\Leftarrow$ tableaux des prédécesseurs de chaque sommet

to_visit.push(s0)
distances $\Leftarrow$ initialisation à ∞
previous $\Leftarrow$ initialisation à $\emptyset$
distances[s0] $\Leftarrow$ 0
Tant que not *to_visit*.isEmpty() **faire**
    s = *to_visit*.pop()
    Pour chaque Edge *e* dans s.edges **faire**
        Si *distances*[*s*] + *e*.distance < *distances*[*e*.distination] **Alors**
            distances[*e*.distination] $\Leftarrow$ *distances*[*s*] + *e*.distance
            previous[*e*.distination] $\Leftarrow$ *s*
            Si *e*.distination $\notin$ *to_visit* **Alors**
                to_visit.push(*e*.distination)
            fin Si
        fin Si
    fin Pour
fin Tant que

1.2 Déroulement

Déroulement de l'algorithme en partant du carrefour C.

iter.	d(A)	d(B)	d(C)	d(D)	d(E)	P(A)	P(B)	P(C)	P(D)	P(E)
0	∞	∞	0	∞	∞	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
1 (C)	4	10	0	1	∞	C	C	$\emptyset$	C	$\emptyset$
2 (A)	4	10	0	1	6	C	C	$\emptyset$	C	A
~~3 (B)~~	4	10	0	1	6	C	C	$\emptyset$	C	A
4 (D)	4	9	0	1	6	C	D	$\emptyset$	C	A
5 (E)	4	7	0	1	6	C	E	$\emptyset$	C	A

On obtient ainsi un tableau de successeurs permettant de trouver rapidement les chemins les plus courts vers chaque sommet, notamment celui qui nous intéresse la boutique B.

$$P(B) = E \Rightarrow P(E) = A \Rightarrow P(A) = CPath = \{C, A, E, B\}$$

Dommage pour ce cher Ford, il se retrouvera non pas à sa boutique mais à l'hôpital suite aux baffes qu'il a reçu des conducteurs qui attendaient derrière lui.

Note : On peut faire abstraction de l'itération 3 (B) si on ne cherche que les chemins menant vers B (inutile de regarder où on peut aller en partant de B si on veut aller à B).

2 Algorithme de Dijkstra

L'algorithme de Dijkstra est très semblable à celui de Bellman-Ford mais cet algorithme évite les redondances que peuvent causer les circuits d'un graphe en se focalisant à chaque fois sur le sommet accessible le plus rapidement. On marque chaque sommet visité pour éviter d'y retourner plus tard.

2.1 Algorithme

Algorithm 2 Dijkstra

```
M ⇐ sommets marqués
S ⇐ listes des sommets
s0 ⇐ sommet de depart
distances ⇐ tableau des distances minimums en partant de s0
previous ⇐ tableaux des prédécesseurs de chaque sommet

distances ⇐ initialisation à ∞
previous ⇐ initialisation à ∅
distances[s0] ⇐ 0
s ⇐ s0
Tant que s n'est pas nul faire
    M.push(s)
    Pour chaque Edge e dans s.edges faire
        Si distances[s] + e.distance < distances[e.distination] Alors
            distances[e.distination] ⇐ distances[s] + e.distance
            previous[e.distination] ⇐ s
        fin Si
    fin Pour
    s ⇐ sommet ayant la distance la plus petite parmi les sommets qui ne sont pas dans M
fin Tant que
```

Note : On peut utiliser un tableau *visited* pour savoir si un sommet est déjà dans M ou pas.

2.1.1 Déroulement

Déroulement de l'algorithme en partant du carrefour C.

Chemin	d(A)	d(B)	d(C)	d(D)	d(E)	P(A)	P(B)	P(C)	P(D)	P(E)
0 M=$\emptyset$	∞	∞	0	∞	∞	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
1 M={C}	4	10		1	∞	C	C	$\emptyset$	C	$\emptyset$
2 M={C, D}	4	9			7	C	D	$\emptyset$	C	D
3 M={C, D, A}		9			6	C	D	$\emptyset$	C	A
4 M={C, D, A, E}		7				C	E	$\emptyset$	C	A
5 M={C, D, A, E, B}						C	E	$\emptyset$	C	A

3 TP

Soit le graphe G suivant.

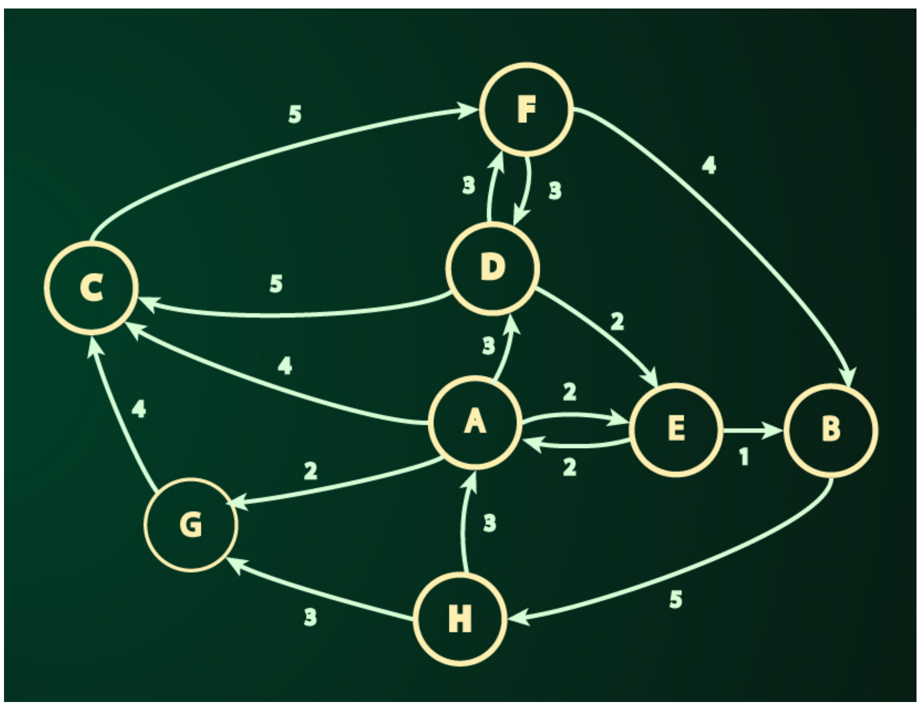

Figure 2 – Graphe G

Dérouler l'algorithme de Bellman-Ford et de Dijkstra une première fois en partant de A et une deuxième fois en partant de F.