Algorithme de Dijkstra

Problématique :

Un problème fréquent qui traverse les disciplines et notre quotidien consiste à chercher à déterminer le plus court chemin pour passer d'un point à un autre au sein d'un réseau ou d'un graphe en sélectionnant, au choix, des critères tels que le temps de parcours, la distance, le coût, etc...
Plusieurs solutions ont été imaginées mais l'algorithme de Dijkstra, créé en 1959 par E. W. Dijkstra, est une façon élégante et économique en calculs de résoudre ce problème.

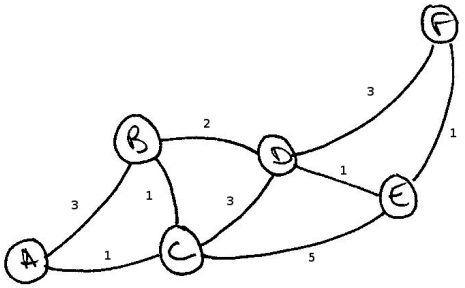
Partie I : Principe général

L'algorithme suppose la connaissance préalable d'un **graphe** défini par :

— Un ensemble de nS objets désignés sous le terme de **sommets** et indicés de 0 à $nS-1$ (ce peut être des arrêts de bus, des stations de métro, des villes au sein d'un réseau ferré, etc.). Parmi ces sommets, deux d'entre eux seront considérés comme le point de départ et d'arrivée du trajet à parcourir, peu importe lesquels. Ω désignera l'ensemble des sommets possibles.

— Une liste **d'arêtes** qui relient ces sommets entre eux en sachant qu'une arête entre les sommets i et j n'existe que si ces sommets sont adjacents au sein du graphe. Nous supposerons par la suite que toute arête peut être empruntée dans les deux sens mais ça n'a rien d'indispensable et l'algorithme décrit par la suite fonctionne aussi bien avec certains chemins à sens unique (on parlera dans ce cas d'« arc »).
A chaque arête, on associe un **poids** noté $\tau(i,j)$ qui dépend le plus souvent du problème posé : ce poids peut désigner les distances qui séparent chaque sommet ou encore les temps de parcours...

Imaginons le cas simple suivant où les distances (dans l'unité de votre choix) sont indiquées sur chaque arête :



Un trajet entre deux sommets S_i et S_j correspond à un chemin dans le graphe qui part de S_i et arrive en S_j. Le poids d'un tel trajet dont les sommets auraient pour indices : $i = i_0 \to i_1 \to \cdots \to i_n = j$ et qui utilise n arêtes est la somme des poids de chacune des arêtes. Ou encore :

$$\text{poids}(S_i = S_{i_0} \to S_{i_1} \to \cdots \to S_{i_n} = S_j) = \sum_{k=0}^{n-1} \tau(i_k, i_{k+1})$$

Le principe fondamental de l'algorithme repose sur l'idée que, si le trajet du sommet S_i au sommet S_j est optimal, alors chaque trajet intermédiaire jusqu'à l'un quelconque des sommet S_k de ce parcours est lui-même optimal (ce qui semble aller de soi car s'il existait un meilleur chemin de S_i à S_k que le sous-chemin $i_0 \to i_1 \to \cdots \to i_k$ du chemin de S_i à S_j, alors on pourrait remplacer ce sous-chemin par un autre qui ferait diminuer le poids total. Ceci contredirait l'optimalité du chemin de départ).

En partant de S_i considéré comme sommet de départ, il s'agit donc pas à pas, sommet après sommet, de construire le trajet qui minimise le « poids » associé à son parcours.

✐ Notons qu'il est possible que le trajet recherché ne soit pas unique mais en aucun cas nous ne réclamons dans ce contexte l'unicité de la solution...

Partie II : Mise en place de l'algorithme

Pour fixer les idées, supposons que nous nous intéressons aux longueurs des trajets sur le graphe. Ainsi, dans la suite, le poids de chaque arête désignera la distance nécessaire pour la parcourir. Il s'agira pour nous de déterminer le trajet de distance minimale entre un sommet initial désigné comme étant la **source** et un sommet final désigné comme le **puits**.

1) Outils et initialisation :

Il est indispensable de se familiariser avec l'algorithme de Dijkstra, et pour ça nous proposons de construire à la main trois tableaux que nous retrouverons sous forme de liste dans notre programme informatique :

— Le premier tableau que nous désignerons par *Distances* indique les distances nécessaires pour atteindre chacun des sommets, au fur et à mesure des étapes, et en partant de la source S_0 que nous supposerons pour l'instant être A (mais qui doit pouvoir dans le cas général être n'importe lequel des sommets). A l'étape 0, la distance pour atteindre A est naturellement fixée à 0 tandis qu'on affecte l'infini à chacun des autres sommets, non encore atteints.

— Le second tableau, appelé S, contient chacun des sommets successivement sélectionnés comme minimisant à chaque étape les trajets possibles. Lors de l'initialisation, S sera réduit au seul sommet A, départ imposé de notre parcours.
Par la suite, un sommet qui a déjà été sélectionné ne sera plus étudié dans le tableau *Distance* car on ne repasse pas deux fois par le même sommet. Afin de signaler ces sommets, on les indiquera d'un 'S' (pour *Supprimé*) dans le tableau *Distance* et, dans le cas du programme Python, on fera appel à la méthode `L.remove()` pour le supprimer.

— Le troisième tableau, *Parents*, indique quant à lui le dernier sommet parcouru avant d'atteindre un nouveau sommet. Au départ, n'étant pas encore partis, nous mettrons un A pour le sommet A (on était en A avant d'être en A...) et un `None` (comme 'vide') pour chacun des autres sommets qui, n'ayant pas été parcouru, n'ont évidemment pas de sommet parent...
Ce tableau est primordial car, *in fine*, c'est lui qui nous permettra de reconstruire, « à reculons », le plus court chemin...
✐ On notera notamment que `Parents['A'] = 'A'` et que ce sera le seul sommet à vérifier cette propriété...

2) Une approche par l'exemple :

Exercice : Compléter les deux dernières lignes de *Distances*, *S* et *Parents* après avoir lu les explications.

Étape	`Distances` (depuis A)					
	A	B	C	D	E	F
0	**0**	∞	∞	∞	∞	∞
1	'S'	3	**1**	∞	∞	∞
2	'S'	**2**	'S'	4	6	∞
3	'S'	'S'	'S'	**4**	6	∞
4						
5						

S
A
C
B
D

`Parents` (Dernier sommet)					
A	B	C	D	E	F
A	`None`	`None`	`None`	`None`	`None`
A	**A**	**A**	`None`	`None`	`None`
A	**C**	A	**C**	**C**	`None`
A	**C**	A	**C**	**C**	`None`
A					
A					

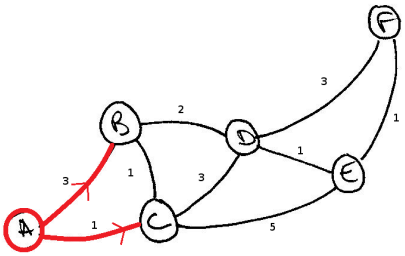


FIGURE 1 – étape 1

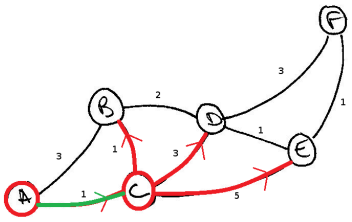


FIGURE 2 – étape 2

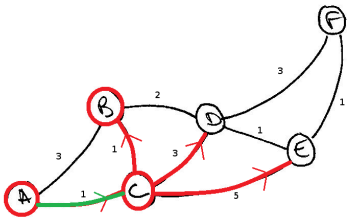


FIGURE 3 – étape 3

Explications : On parcourt chacun des sommets du graphe par distance croissante à la source.

① ***Étape 0 :*** On part de A. Dans *Distances*, la distance depuis A est nulle et tous les autres sommets sont initialisés à une distance infinie. Seul le sommet A est sélectionné (cf. tableau S).

② ***Étape 1 :*** Depuis A, le sommet B est à 3 km et le sommet C est à 1 km. On indique donc respectivement 3 et 1 dans les colonnes correspondant à ces sommets dans le tableau *Distances* des longueurs de trajet depuis A.
Le plus court trajet est d'un **1** km et aboutit en $\boxed{C}$: On sélectionne donc le sommet C dans le tableau 2 et dans le tableau *Parents*, on indique pour B et C qu'on vient de passer par A, marqué d'un « S » pour « Sélectionné » dans `Distances`.

③ ***Étape 2 :*** Partant de A $\underline{\text{via } C}$ (tous deux désormais sélectionnés), on va :
- en B avec un trajet total de longueur : 1 km $+d(\underline{C}, B) = 1 + 1 = 2$ km. C'est mieux que 3 km. Aussi on remplace 3 par 2 et on indique dans *Parents* que pour aller en B on passe désormais par C.
- en D avec un trajet total de longueur : 1 km $+d(\underline{C}, D) = 1 + 3 = 4$ km. C'est mieux qu'une distance infinie. On remplace ∞ par 4 et on met à jour le tableau *Parents* en indiquant qu'on passe par C.
- en E avec un trajet total de longueur : 1 km $+d(\underline{C}, E) = 1 + 5 = 6$ km. C'est également mieux qu'une distance infinie. DU et DS sont mis à jour pour le sommet E.

A l'étape 2, le plus court trajet est de longueur 2 ; c'est celui qui mène en B : On sélectionne le sommet B dans le tableau 2.

④ ***Étape 3 :*** Partant de A via C puis B (distance 2 km), on va :
— en D avec un trajet total de longueur 2 km+$d(B, D) = 2 + 2 = 4$ km. On ne gagne rien par rapport au précédent trajet via C et donc ni *Distances* ni *Parents* ne sont modifiés pour le sommet D.

A cette étape, le plus court trajet est celui qui mène en D (4 km) : On sélectionne le sommet D dans le tableau 2.

⑤ ***Etape 4 :*** Partant de A via C, B puis D (4 km), on va :
A détailler en complétant les tableaux précédents.

⑥ ***Etape 5 :*** **A détailler...**
C'est fini !

A l'issue de l'algorithme, le tableau des *Dernier sommet* doit être :

`Parents=[A,C,A,C,D,E]`

En partant de F, il suffit de remonter la chaîne des derniers sommets dans le tableau `Parents` jusqu'à rejoindre A : En effet, partant de la dernière colonne de `Parents`, on obtient que le dernier sommet parcouru pour rejoindre F est E, puis grâce à la colonne du sommet E, on obtient que son prédécesseur est D dont le prédécesseur est C (colonne du sommet D), lui-même précédé du sommet A (colonne du C) avec `Parents[A] = A`. Nous sommes arrivés à la source. On s'arrête...

Conclusion : Le plus court chemin est $A - C - D - E - F$ et il fait 6 km

3) Description d'une itération :

Chaque itération se fait en deux temps :

① *Premier pas de temps :*
Soit S_{selec} le dernier sommet sélectionné et ajouté au tableau S et marquée d'un 'S' dans le tableau `Distances`. On connaît donc le trajet optimal $D_{opt} = Distances(S_{selec})$ pour aller de A à S_{selec}.
On regarde alors le poids de toutes les arêtes (S_{selec}, S_x) qui joignent S_{selec} aux autres sommets S_x possibles, c'est-à-dire **extérieurs** aux sommets de S qui ont déjà été traités (et donc plus concrètement dans $\Omega \setminus S$)...
Le trajet total pour aller de A à chacun de ces sommets, via le sommet S_{selec}, a donc un poids (mettons une « distance »ou une« durée » selon qu'on cherche le parcours le plus court ou le plus rapide...) :

$$D_{totale} = D_{opt} + \tau(S_{selec}, S_x) \text{ où } S_x \in \Omega \setminus S \text{ et } \tau(S_1, S_2) = \text{poids de l'arête } (S_1, S_2)$$

Si $D_{totale} < Distances(S_x)$ (en notant que $Distances(S_x)$ peut être infinie), on affecte D_{totale} à $Distances(S_x)$ et S_{selec} à $Parents(S_x)$ puisque, à cette étape, le meilleur trajet pour aller de A à S_x dure D_{totale} et passe par S_{selec}.

② *Second pas de temps :* On cherche parmi les sommets de $\Omega \setminus S$ ceux qui minimisent la durée de trajet depuis A. Le sommet trouvé est ajouté à S.

L'algorithme s'achève lorsqu'on a épuisé tous les sommets sélectionnables possibles (ou bien lorsqu'on a atteint le « puits »). Il suffit alors de lire *Parents* pour remonter les derniers sommets jusqu'au sommet A...
✍ On notera que si on poursuit l'algorithme jusqu'à épuiser tous les sommets possibles, alors il retourne un plus court chemin de la « source » à **chaque** autre sommet du graphe.

Partie III : Programmation

Exercice 1 : Mise en place de l'algorithme de `dijkstra` :

Reportez-vous au notebook intitulé `devoir_maison_Dijkstra` et complétez chacune des fonctions afin de pouvoir mettre en pratique l'algorithme de Dijkstra sur le parcours de A à F proposé dans ce polycopié.
Vérifiez à cette occasion que vous pouvez choisir n'importe lequel des sommets comme étant la « source » et le « puits » et vérifiez les solutions proposées.

Exercice 2 : Un trajet en bus :

On souhaite, en empruntant le réseau de transport de la TAN reproduit ci-dessous, se rendre du Chêne des Anglais à l'arrêt Mondésir, autrement dit du point A au point N dans une version simplifiée réduite aux lignes les plus rapides du réseau comme les lignes de tramway L_2 et L_3 et les lignes C_2 et C_6.
Les temps entre les stations sont indiqués en rouge. Ce sont ceux fournis par la TAN et sont donnés en minutes. Les temps de changement de lignes, quant à eux, sont le fruit de pures spéculations...

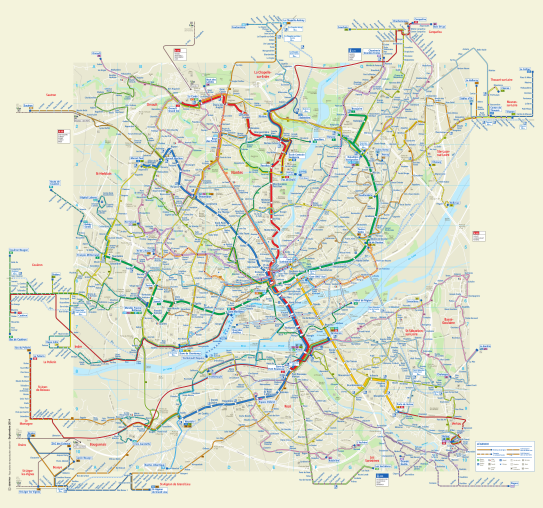


FIGURE 4 – Plan de la TAN

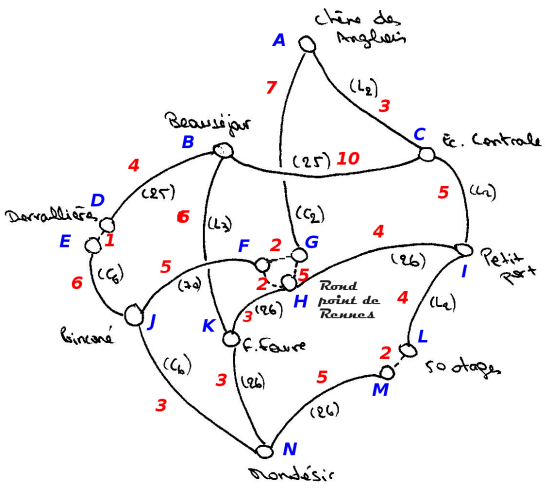


FIGURE 5 – Graphe simplifié (temps en minutes)

① Le graphe étant à la fois défini par l'ensemble des stations marquées d'une lettre de A à N et par l'ensemble des arêtes qui les relient, vérifier que '`Graphe2` fourni en fin de Notebook est bien sa liste d'adjacence.

② Appliquer vos fonctions `dijkstra()` et `meilleur_trajet()` et fournir le meilleur trajet pour arriver en moins de 20 mn au lycée en partant du Chêne des anglais.