Graphes

Algorithme des graphes: algorithme de Dijkstra

Anicet E. T. Ebou, ediman.ebou@inphb.ci

01

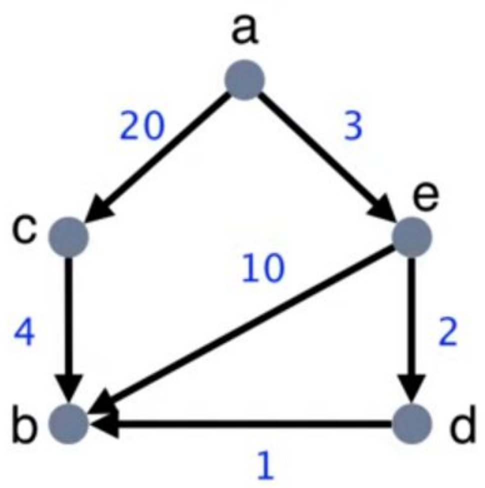
Présentation et distance pondérée

Objectif

L'objectif de l'algorithme de Djikstra est de construire les plus courts chemins à partir pondérés à partir d'un sommet d'un graphe G.

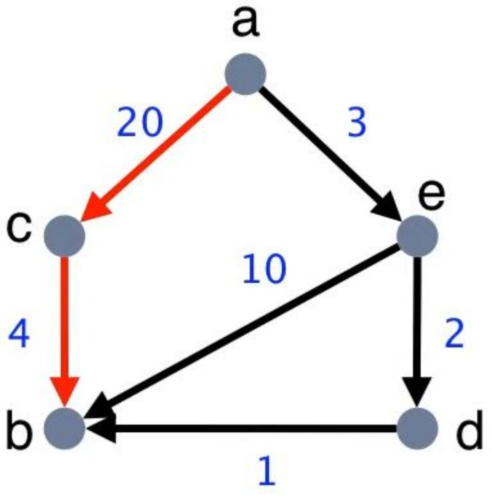
Distance pondérée

Prenons un graphe orienté pour exemple.



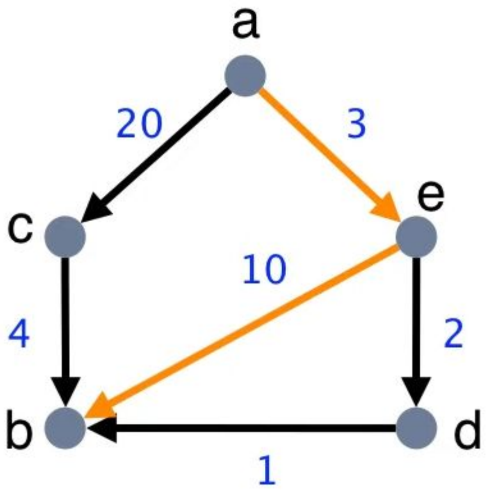
Distance pondérée

Cherchons le chemin le plus courts possible entre les sommets a et b. Une première option est le chemin colorié ici en rouge et de longueur 24.



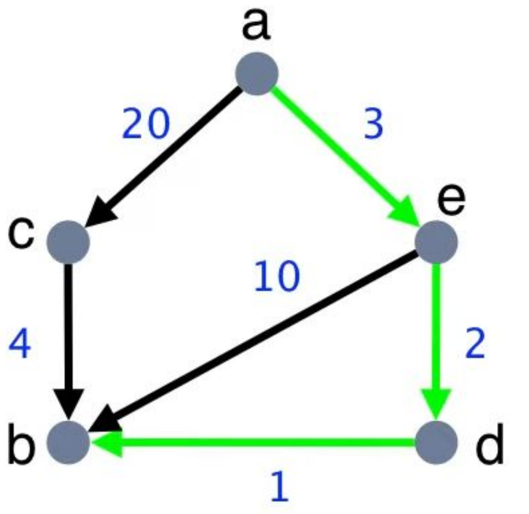
Distance pondérée

On peut aussi trouver un chemin (ici en orange) de longueur 13.



Distance pondérée

Finalement on peut trouver un chemin plus court (ici en vert) de longueur 6.



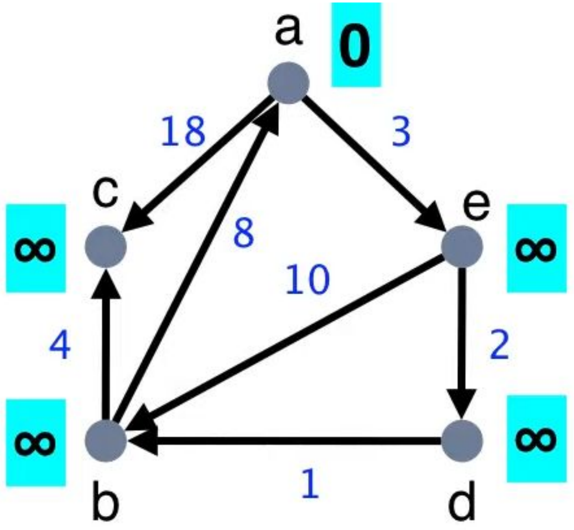
02

Algorithme de Dijkstra

Initialisation

En partant du graphe ci-contre, on se donne de chercher le chemin le plus court entre les sommets a et b.

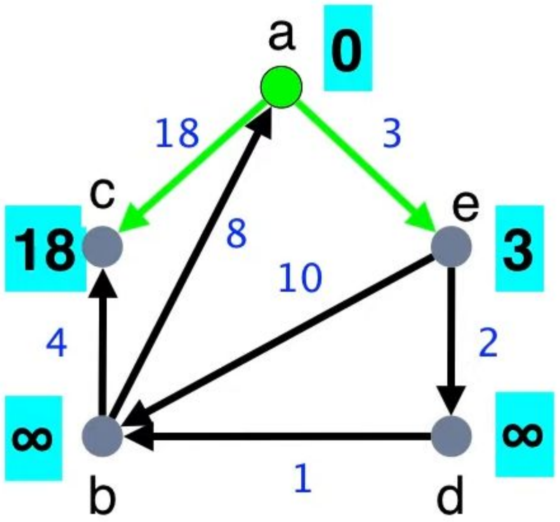
L'algorithme commence par une initialisation (assez pessimiste), ou la distance entre a et b sont supposés infinis. La distance entre a et a est elle égale à 0.



Traitement du sommet a

Commençons en partant du sommet a et relâchons les arcs sortant du sommet a.

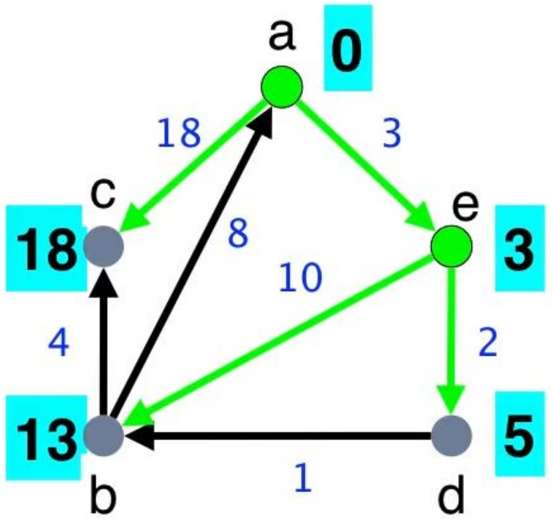
- On découvre alors une route de a vers c de longueur 18. 18 étant strictement inférieur à l’infini, on remplace la valeur initiale.
- On fait de même pour le second arc sortant qui révèle une route de a vers e avec une longueur 3.



Traitement du sommet e

L’algorithme de Dijkstra dit que le sommet suivant qui doit être traité est **un sommet non encore traité qui a la plus petite étiquette**. Ce sommet est ici le sommet e.

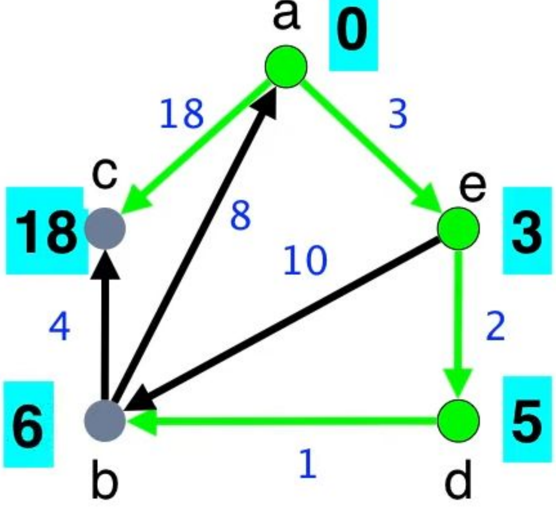
- L’arc de e vers d est de longueur 2. On met à jour l’étiquette du sommet d qui devient 5.
- On fait de même pour l’arc de e vers b. L’ étiquette de b est alors mis à jour vers 13.



Traitement du sommet d

En utilisant le même principe, le prochain sommet est le sommet d.

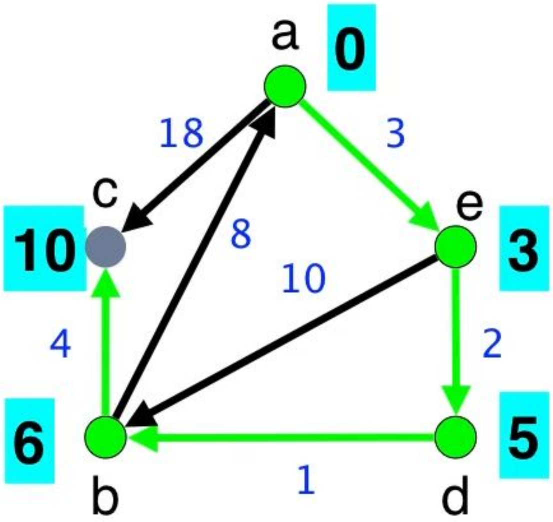
- En relâchant l’unique arc sortant de d, on a obtient une longueur de a à b de 6. 6 étant strictement meilleur que 13 on met à jour l’ étiquette du sommet b.



Traitement du sommet b

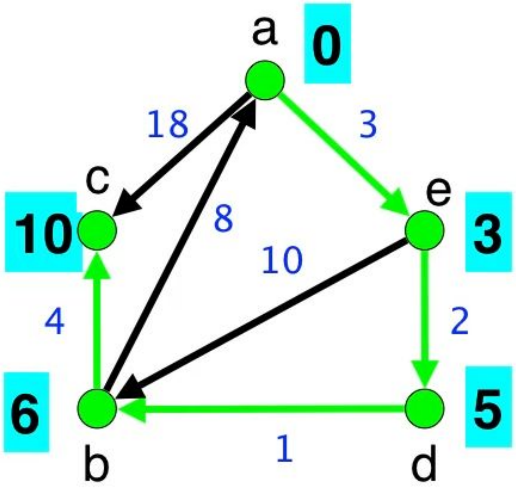
Le prochain sommet à visiter est le sommet b.

- On relâche l'arc de b vers a. On obtient alors un chemin de a vers a de longueur 14 (6 + 8). 14 est strictement plus grand que 0, donc cet arc n'est pas mémorisé ni pris en compte.
- On relâche ensuite l'arc de b vers c et on découvre un chemin de longueur 10 qui est meilleur que le précédent.

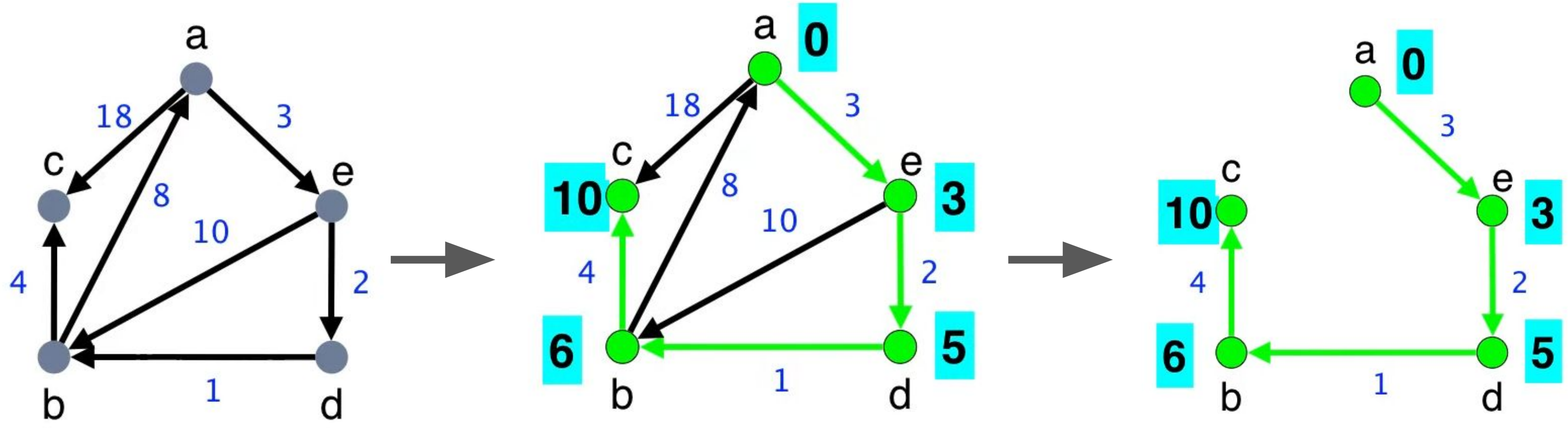


Traitement du sommet c

Le prochain sommet à visiter est le sommet c. Le sommet c n’a pas de sommet sortant. A ce moment l’algorithme s’arrête.



Recapitulatif



En partant du premier graphe on arrive au second graphe. Le troisième est celui que l'on obtient après avoir retiré les graphes non visités et on obtient un arbre simple (ce qui est une pur coïncidence dû à la topologie du graphe).

```
def dijkstra(adj_matrix, start):
      num_nodes = len(adj_matrix)
      distances = [float('inf')] * num_nodes # Initialise les distances et les prédécesseurs
      predecessors = [None] * num_nodes
      distances[start] = 0
      unvisited = list(range(num_nodes)) # Liste pour conserver les sommets non visites
      while unvisited:
      current_node = min(unvisited, key=lambda node: distances[node]) # Trouve le sommet avec la min distance
      unvisited.remove(current_node)

      for neighbor in range(num_nodes): # Explore les voisins du sommet actuel
            if adj_matrix[current_node][neighbor] > 0:
            distance = distances[current_node] + adj_matrix[current_node][neighbor]

            # If a shorter path is found, update the distance and predecessor
            if distance < distances[neighbor]:
                  distances[neighbor] = distance
                  predecessors[neighbor] = current_node
      return distances, predecessors
```