Projet Algorithmique M2DS

OSMAN Ahmed

2023-01-22

Algorithme de *Bellman–Ford*

Principe de l'algorithme

L'algorithme de *Bellman-Ford* est utilisé pour trouver les plus courts chemins dans un graphe pondéré avec **des poids négatifs**.
Il s'agit d'une variante de l'algorithme de Dijkstra qui peut gérer des **poids négatifs**.

L'algorithme fonctionne en itérant sur tous les arcs du graphe un certain nombre de fois. Lors de chaque itération, les distances des sommets sont mises à jour en utilisant les distances des sommets voisins. Plus précisément, pour chaque **arc (u, v)** du graphe, **la distance de "v" est mise à jour si elle est plus courte que la distance de "u" plus le poids de l'arc (u, v).**

Après **n-1 itérations**, où **n** est le nombre de sommets dans le graphe, les distances des sommets seront correctes si le graphe ne contient pas de cycles de poids négatif. **Si l'algorithme effectue encore une itération et si une mise à jour est effectuée, cela signifie qu'il existe un cycle de poids négatif dans le graphe.**

En résumé, l'algorithme de *Bellman-Ford* est utilisé pour trouver les plus courts chemins dans un graphe pondéré avec des poids négatifs en itérant sur tous les arcs du graphe un certain nombre de fois et en mettant à jour les distances des sommets en utilisant les distances des sommets voisins.

Complexité

La complexité de l'algorithme de *Bellman-Ford* est de $O(V * E)$, où V **est le nombre de sommets** dans le graphe et **E est le nombre d'arcs**.

La raison de cette complexité est que l'algorithme effectue un certain nombre d'itérations *(généralement n-1 itérations, où n est le nombre de sommets dans le graphe)* sur tous les arcs du graphe. Pour chaque itération, l'algorithme parcourt tous les arcs du graphe pour mettre à jour les distances des sommets.

Ainsi, la complexité totale de l'algorithme est $(n-1) * E$, où n est le nombre de sommets et E est le nombre d'arcs. En utilisant la notation de complexité, cela peut être exprimé comme $O(V * E)$.

Il est important de noter que cette complexité est valable pour un graphe non-dense, c'est-à-dire lorsque le nombre d'arcs est beaucoup plus petit que V^2.

Dans des cas où le nombre d'arcs est proche de V^2, la complexité de l'algorithme est proche de O(V^3).

En effet, si le nombre d'arcs est proche de V^2, cela signifie que pour chaque sommet il y a un nombre important d'arcs connectés, c.à.d pour chaque itération, tous les arcs seront parcourus pour mettre à jour les distances des sommets.

En utilisant la notation de complexité, cela peut être exprimé comme $O(V^2 * V) = O(V^3)$. Cela est dû au fait que l'algorithme parcours tous les arcs $E = V^2$, pour V sommets, pour n-1 itérations.

Avantages et Inconvénients

Avantages

- L'algorithme de Bellman-Ford peut gérer des poids négatifs, contrairement à l'algorithme de Dijkstra qui ne peut gérer que des poids positifs.
- Il peut détecter les cycles de poids négatifs dans un graphe, ce qui n'est pas possible avec l'algorithme de Dijkstra.
- Il est relativement simple à implémenter comparé à d'autres algorithmes pour trouver les plus courts chemins.

Inconvénients

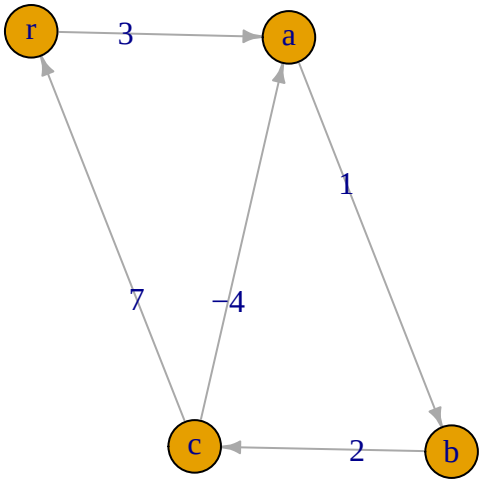
- Il est plus lent que l'algorithme de *Dijkstra*, lorsque les poids sont tous positifs, l'algorithme de *Dijkstra* est généralement plus rapide que *Bellman-Ford*, car il ne parcourt pas tous les arcs du graphe un certain nombre de fois.
- Il peut être inefficace pour les graphes denses *(grand nombre d'arcs)*: lorsque le nombre d'arcs est proche de V^2 (où V est le nombre de sommets), la complexité de l'algorithme devient $O(V^3)$, ce qui peut rendre l'algorithme inefficace pour les graphes denses.
- Il n'est pas adapté pour les graphes de grande taille *(grand nombre de sommets)*: l'algorithme de *Bellman-Ford* peut être inefficace pour les graphes de grande taille en raison de sa complexité $O(V * E)$, donc il peut être nécessaire d'utiliser des algorithmes plus efficaces comme *Johnson's algorithm* ou *Floyd-Warshall algorithm* pour résoudre des problèmes sur des graphes de grande taille.

Implémentation de l'algorithme

```
BellmanFord <- function(vertices, edges, start) {

  # intialisation
  n <- length(vertices)
  distance <- rep(Inf, n)
  parents <- rep(NA, n)

  names(distance) <- vertices
  names(parents) <- vertices
  distance[start] <- 0

  # relaxation des arcs
  for (i in 1:(n-1)) {
    for (j in 1:(nrow(edges))) {

      u <- edges$from[j]
      v <- edges$to[j]
      w <- edges$weight[j]

      if (distance[v] > distance[u] + w) {
        distance[v] <- distance[u] + w
        parents[v] <- u
      }
```

```
    }
  }

  # vérifications des cercles négatifs
  for (k in 1:(nrow(edges))) {

    u <- edges$from[k]
    v <- edges$to[k]
    w <- edges$weight[k]

    if (distance[v] > distance[u] + w) {
      # trouver le cercle négatif
      negativeLoop <- c(v, u)
      for (l in 1:(n-1)) {
        u <- negativeLoop[1]
        for (j in 1:(nrow(edges))) {

          v <- edges$to[j]
          w <- edges$weight[j]
          if (v != u) {
            if (distance[v] > distance[u] + w) {
              negativeLoop <- c(v, negativeLoop)
            }
          }
        }
      }
      cat("\nWarning! - Graph contains a negative-weight cycle!\nThe shortest path will not be efficien
    }
  }
  return(list(distance = distance, parents = parents))
}
```

```
vertices <- c("r", "a", "b", "c")
edges <- data.frame(from   = c("r", "a", "b", "c", "c"),
                    to     = c("a", "b", "c", "r", "a"),
                    weight = c(3, 1, 2, 7, -4))

# Create the graph
g <- graph_from_data_frame(edges,
                           directed = TRUE)

plot(g,
     edge.label=E(g)$weight,
     edge.label.cex=1,
     vertex.size = 25,
     edge.arrow.size = 0.5)
```

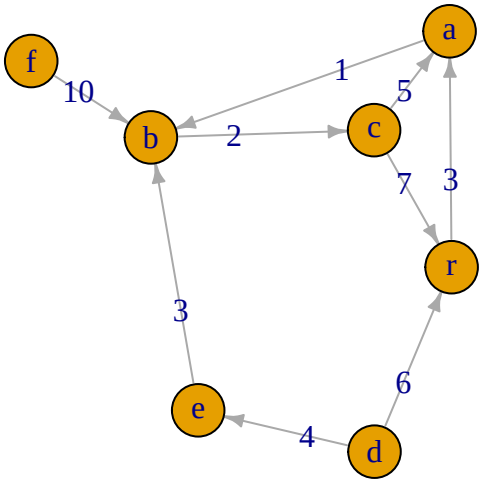



```
BellmanFord(vertices, edges, "r")
```

```
##
## Warning! - Graph contains a negative-weight cycle!
## The shortest path will not be efficient!
```

```
## $distance
## r a b c
## 0 0 2 4
##
## $parents
##   r   a   b   c
##  NA "c" "a" "b"
```

```
vertices <- c("r", "a", "b", "c", "d", "e", "f")
edges <- data.frame(from   = c("r", "a", "b", "c", "c", "d", "e", "f", "d"),
                    to     = c("a", "b", "c", "r", "a", "r", "b", "b", "e"),
                    weight = c(3, 1, 2, 7, 5, 6, 3, 10, 4))

# Create the graph
g <- graph_from_data_frame(edges,
                           directed = TRUE)

plot(g,
     edge.label=E(g)$weight,
     edge.label.cex=1,
     vertex.size = 25,
     edge.arrow.size = 0.5)
```

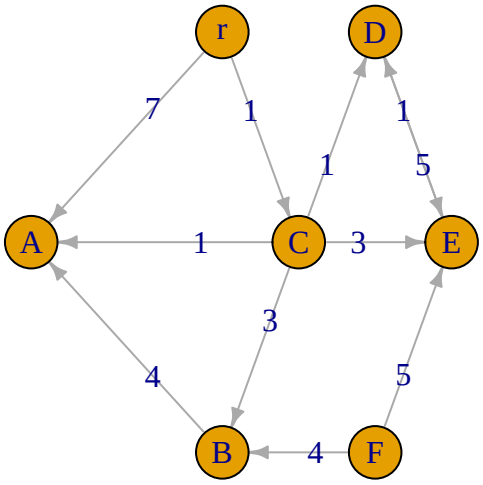



```
BellmanFord(vertices, edges, "r")
```

```
## $distance
##   r   a   b   c   d   e   f
##   0   3   4   6 Inf Inf Inf
##
## $parents
##    r    a    b    c    d    e    f
##   NA  "r"  "a"  "b"   NA   NA   NA
```

```
vertices <- c("r", "A", "B", "C", "D", "E", "F")
edges <- data.frame(from   = c("r", "r", "C", "C", "C", "C", "D", "E", "F", "F", "B"),
                    to     = c("A", "C", "A", "B", "D", "E", "E", "D", "E", "B", "A"),
                    weight = c(7, 1, 1, 3, 1, 3, 1, 5, 5, 4, 4))

coords <- matrix(c(2, 2, # r
                   4, 0, # C
                   6, 2, # D
                   8, 0, # E
                   6, -2,
                   2, -2,
                   0, 0), byrow = TRUE, ncol = 2)

rescale_coords <- function(coords, scale_n) {
  scale_matrix <- matrix(0, nrow = nrow(coords), ncol = ncol(coords))

  for (row in 1:nrow(coords)) {
    for (col in 1:ncol(coords)) {
      if (coords[row, col] < 0)
        scale_matrix[row, col] <- coords[row, col] - scale_n
      if (coords[row, col] > 0)
        scale_matrix[row, col] <- coords[row, col] + scale_n
    }
  }
  return(scale_matrix)
}
```

```
new_coords <- rescale_coords(coords, scale_n = 3)
g <- graph_from_data_frame(edges,
                           directed = TRUE)

plot(g,
     edge.label=E(g)$weight,
     edge.label.cex=1,
     vertex.size = 25,
     edge.arrow.size = 0.5,
     layout=new_coords)
```




```
BellmanFord(vertices, edges, "r")
```

```
## $distance
##   r   A   B   C   D   E   F
##   0   2   4   1   2   3 Inf
##
## $parents
##    r    A    B    C    D    E    F
##   NA "C" "C" "r" "C" "D"  NA
```