0.1 Exemple Dijkstra :

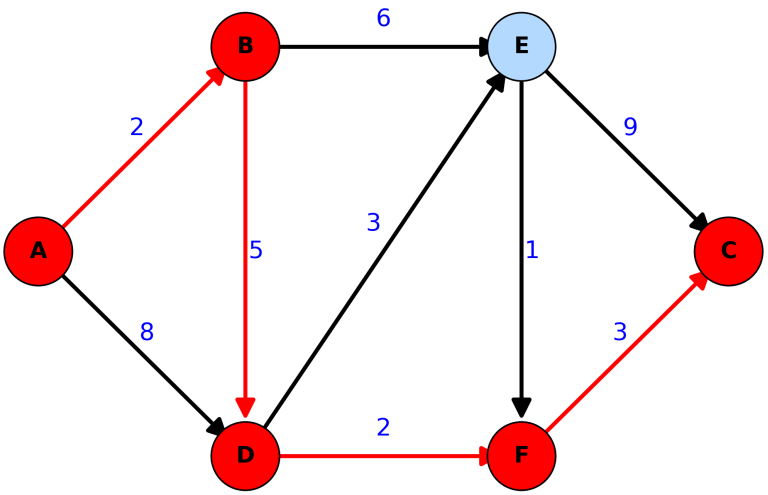

FIGURE 1 – Illustration d'un exemple d'un graphe orienté pondéré

0.1.1 Execution de l'algorithme de Dijkstra

S-visités\Noeuds	A	B	C	D	E	F
–	0	∞	∞	∞	∞	∞
A	0	2_A	∞	8_A	∞	∞
B	|	2_A	∞	7_B	8_B	∞
D	|	|	∞	7_B	8_B	9_D
E	|	|	17_E	|	8_B	9_D
F	0	2_A	12_F	7_B	8_B	9_D

TABLE 1 – Tableau des distances étiquetées avec leur prédécesseur en indice. Les valeurs représentent les distances cumulées depuis le nœud A, avec en indice le nœud précédent dans le chemin.

0.1.2 Détermination du plus court chemin

Pour trouver le plus court chemin entre A et C, on remonte les prédécesseurs de chaque nœud à partir de C jusqu'à arriver à A :

- Depuis C, prédécesseur : F
- Depuis F, prédécesseur : D
- Depuis D, prédécesseur : B
- Depuis B, prédécesseur : A

Le plus court chemin entre A et C est donc :

$$\boxed{A \to B \to D \to F \to C}$$

0.2 Exemple Bellman-Ford :

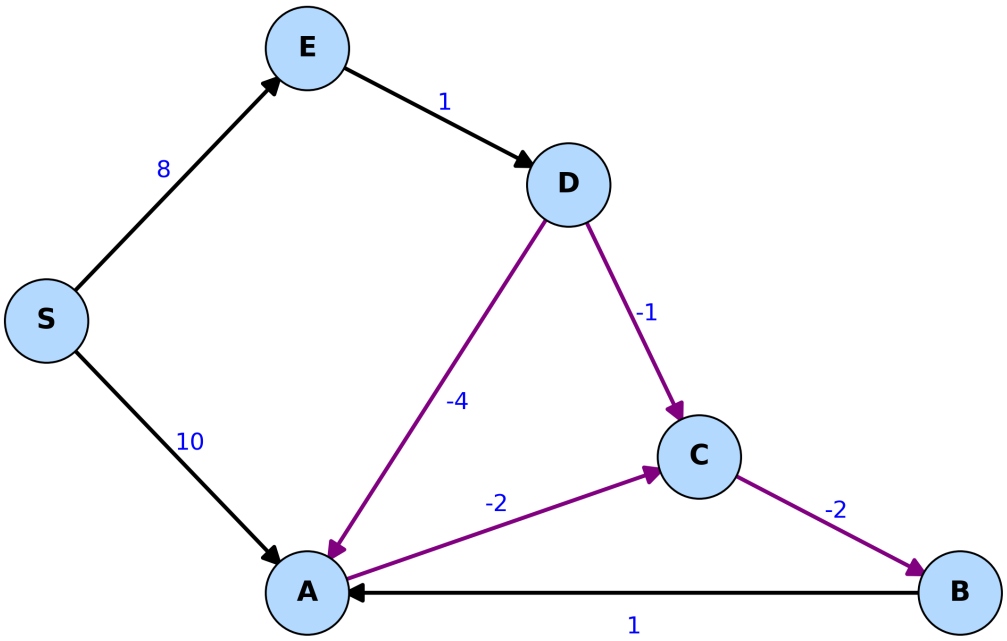

FIGURE 2 – Illustration d'un exemple d'un graphe orienté pondéré

0.2.1 Execution de l'algorithme de Bellman - Ford

itérations\Noeuds	S	A	B	C	D	E
itération 0	0	∞	∞	∞	∞	∞
itération 1	0	10	10	12	9	8
itération 2	0	5	10	8	9	8
itération 3	0	5	5	7	9	8
itération 4	0	5	5	7	9	8

TABLE 2 – Évolution des distances aux sommets à chaque itération

0.3 Algorithme A* (A Star)

L'algorithme A* est un algorithme de recherche de chemin qui combine les avantages de Dijkstra (recherche du chemin le plus court) avec une heuristique pour orienter la recherche vers la destination.

Algorithm 1 Algorithme A* pour la recherche de chemin

Entrée: Un graphe G, nœud de départ *start*, nœud d'arrivée *goal*
Sortie : Le chemin optimal ou échec

Initialisation : $openSet \leftarrow \{start\}$ `// Nœuds à évaluer`
$closedSet \leftarrow \emptyset$ `// Nœuds déjà évalués`
$cameFrom \leftarrow \{\}$ `// Dictionnaire des prédécesseurs`
$gScore \leftarrow$ tableau avec $gScore[n] = \infty$ pour tout nœud n $gScore[start] \leftarrow 0$ $fScore \leftarrow$ tableau avec $fScore[n] = \infty$ pour tout nœud n $fScore[start] \leftarrow h(start, goal)$ `// h est la fonction heuristique`

while $openSet \neq \emptyset$ **do**
  $current \leftarrow$ nœud dans $openSet$ avec le plus petit $fScore$
  if $current == goal$ **then**
    return reconstruireChemin($cameFrom$, $current$)
  end
  $openSet \leftarrow openSet \setminus \{current\}$ $closedSet \leftarrow closedSet \cup \{current\}$
  foreach *voisin* v *de* $current$ **do**
    if $v \in closedSet$ **then**
      continue
    end
    $tentative\_gScore \leftarrow gScore[current] + d(current, v)$ `// Coût actuel`
    if $v \notin openSet$ **then**
      $openSet \leftarrow openSet \cup \{v\}$
    end
    else if $tentative\_gScore \geq gScore[v]$ **then**
      continue
    end
    $cameFrom[v] \leftarrow current$ $gScore[v] \leftarrow tentative\_gScore$ $fScore[v] \leftarrow gScore[v] + h(v, goal)$ `// Coût total estimé`
  end
end
return échec `// Aucun chemin trouvé`

Fonction *reconstruireChemin(cameFrom, current)*
  $chemin \leftarrow [current]$
  while $current \in cameFrom$ **do**
    $current \leftarrow cameFrom[current]$
    $chemin \leftarrow [current] + chemin$
  end
  retourner *chemin*

Algorithm 2 Fonctions heuristiques

1: **function** $h_{\text{MANHATTAN}}(nud, destination)$
2: **retourner** $|nud.x - destination.x| + |nud.y - destination.y|$
3: **end function**
4: **function** $h_{\text{EUCLIDIENNE}}(nud, destination)$
5: $dx \leftarrow nud.x - destination.x$
6: $dy \leftarrow nud.y - destination.y$
7: **retourner** $\sqrt{dx^2 + dy^2}$
8: **end function**
$=0$