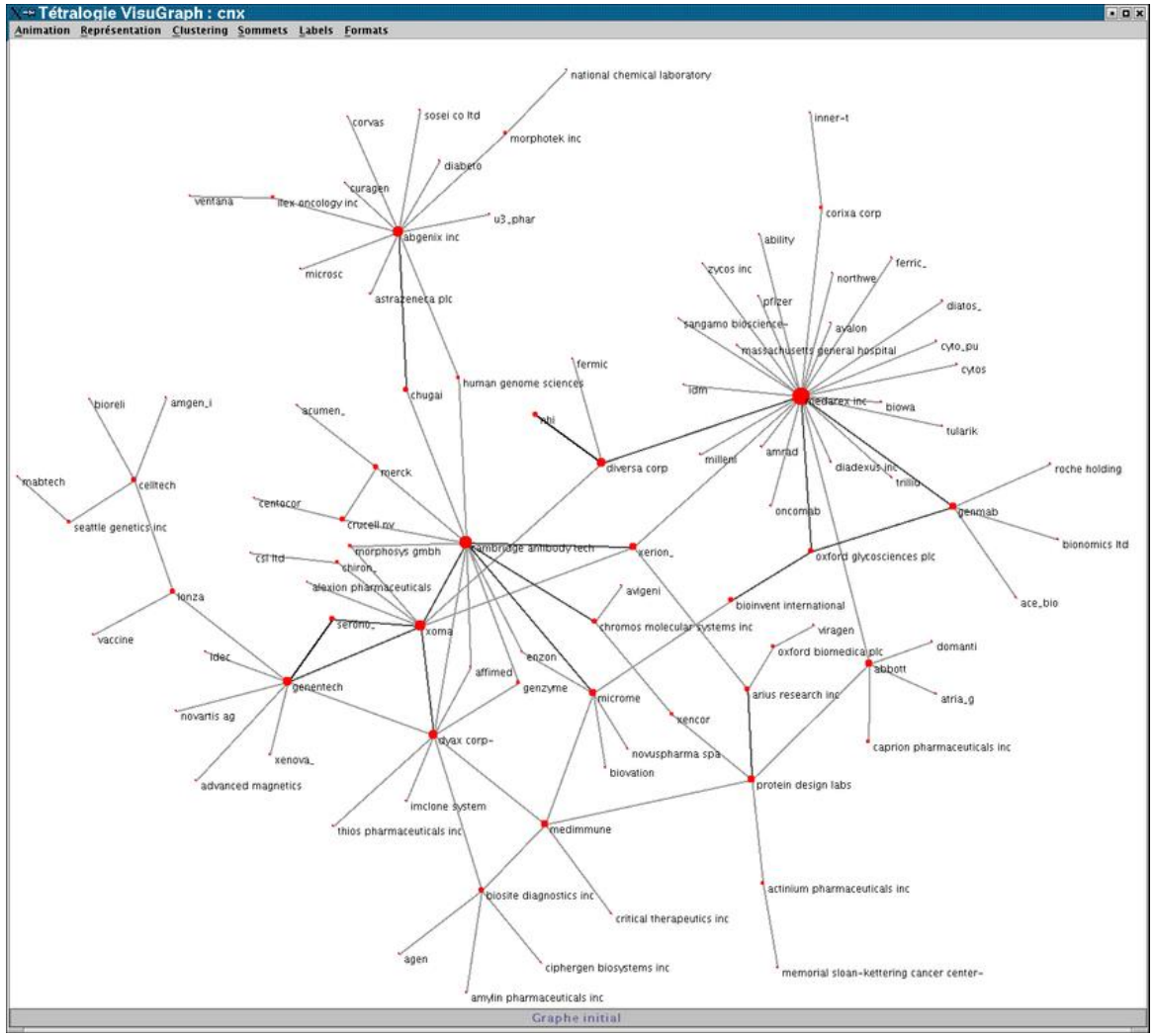
Détermination du chemin le plus court dans un graphe

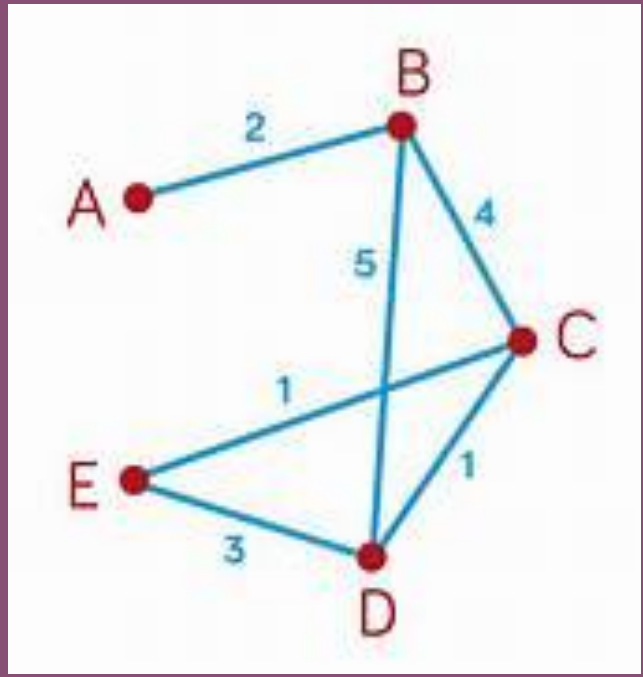


Notion de graphe

- **Définition** : schéma contenant des points (**sommets**) reliés ou non par des segments (**arêtes**).

- **Vocabulaire associé** : **connexe, orienté, pondéré, complet ...**

- **Parcours : largeur, longueur, chaîne.**

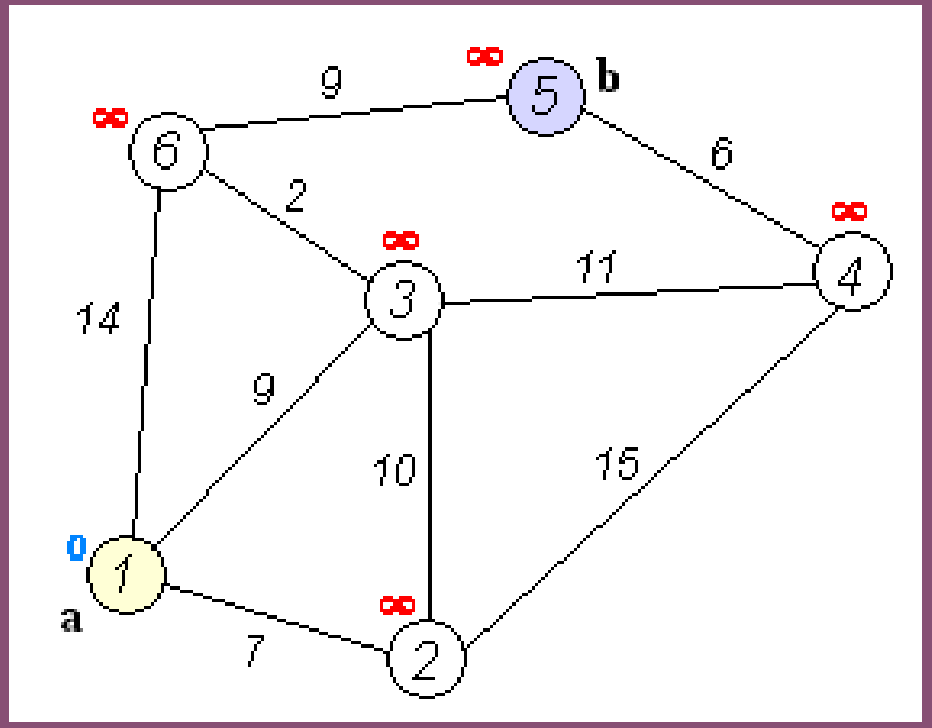


Chemin le plus court, pourquoi ?

- Economie de **temps**, **d'argent**, **d'énergie**, de **matériaux** …

- <u>Exemples</u> :
  - **Transport** : plus court, sans péages, le moins d'escales …
  - **Réseaux** : meilleur transport des paquets.
  - **Construction** : routes, voies ferrées …

Algorithme de Dijkstra



- Inventé par **Edsger Dijkstra** en 1959, lien : https://www.wikiwand.com/fr/Edsger_Dijkstra
- Algorithme **glouton**, complexité **faible** : (ordre^2) x log(sommets) au pire.
- Exemple du cours, lien : https://github.com/lmayer65/NSI_T/blob/main/Structures_Donn%C3%A9es/SDD_Graphes_Dijkstra.pdf

Algorithme de Dijkstra

Initialisation

dist[E]=0, dist[i]=infini pour tout sommet autre que l'entrée **E.**

Expl = [] (sommets explorés)

Traitement

tant qu'il existe un sommet à explorer

choisir un sommet **A** non exploré de plus petite distance **dist[A]**

mettre **A** dans **Expl**

pour chaque sommet **V** non exploré voisin de **A**

dist[V]=min(dist[V], dist[A]+ poids(A,V))

Conclusion

Afficher dist

Algorithme de Dijkstra

- Notion de **poids négatifs**, lien :

https://fr.video.search.yahoo.com/search/video?fr=yfp-t&ei=UTF-8&p=algorithme+dijkstra+ne+fonctionne+pas+poids+negatif+exemple#id=1&vid=fc62ea67f46c37f3f568c4640db17249&action=click

  - **Graphes, poids négatifs, circuits absorbants** : jusqu'à la 4ème minute.
  - **Dijkstra** et les poids négatifs ☹ : à partir de la 4ème minute.

- Poids négatifs : système de gains / pertes par exemple ?

Algorithme de Bellman Ford

Inventé par **Bellman** (avec Ford et Moore) en 1956, republié en 1959, le père de la **programmation dynamique**.

Autorise les **poids négatifs** sauf si **circuit absorbant,** lien ici : https://www.youtube.com/watch?v=Pn7rCyqwZbQ

Exemple, lien ici : https://www.youtube.com/watch?v=iRLGzUZQxek

Algorithme de Bellman Ford

Initialisation

dist[E]=0, **dist[i]=infini** pour tous les sommets sauf l'entrée **E**

pred = {}, aucun prédécesseur connu

Traitement

 # Relâchement des arcs

pour le nombre de sauts nécessaires (ordre du graphe - 1)

    pour chaque nœud

        pour chaque voisin du nœud

            si dist[voisin] > dist[nœud] + poids(nœud, voisin)

                dist[voisin] = dist[nœud] + poids(nœud, voisin)

                pred[voisin] = nœud

Algorithme de Bellman Ford

Traitement (suite)

 # Contrôle de présence d'un cycle absorbant

```
pour chaque nœud
    pour chaque voisin du nœud
        si dist[voisin] <= dist[nœud] + poids(nœud, voisin)
            Afficher « Cycle absorbant »
            Arrêt du programme
```

Conclusion

```
Afficher dist
```

Algorithme de Bellman Ford

Utilisable même avec des **poids négatifs**.

Détection de **cycles absorbants** (ou améliorants).

Complexité au pire **cubique** (graphe complet), plus lent que celui de Dijkstra.

Quelques liens

- Exemple**, Dijkstra**, lien ici : https://github.com/lmayer65/NSI_T/blob/main/Grand%20Oral/Chemin_Plus_Court/CPC_Dijkstra.pdf

- Exemple, **Bellman-Ford**, lien ici :

https://github.com/lmayer65/NSI_T/blob/main/Grand%20Oral/Chemin_Plus_Court/CPC_Bellman_Ford.pdf

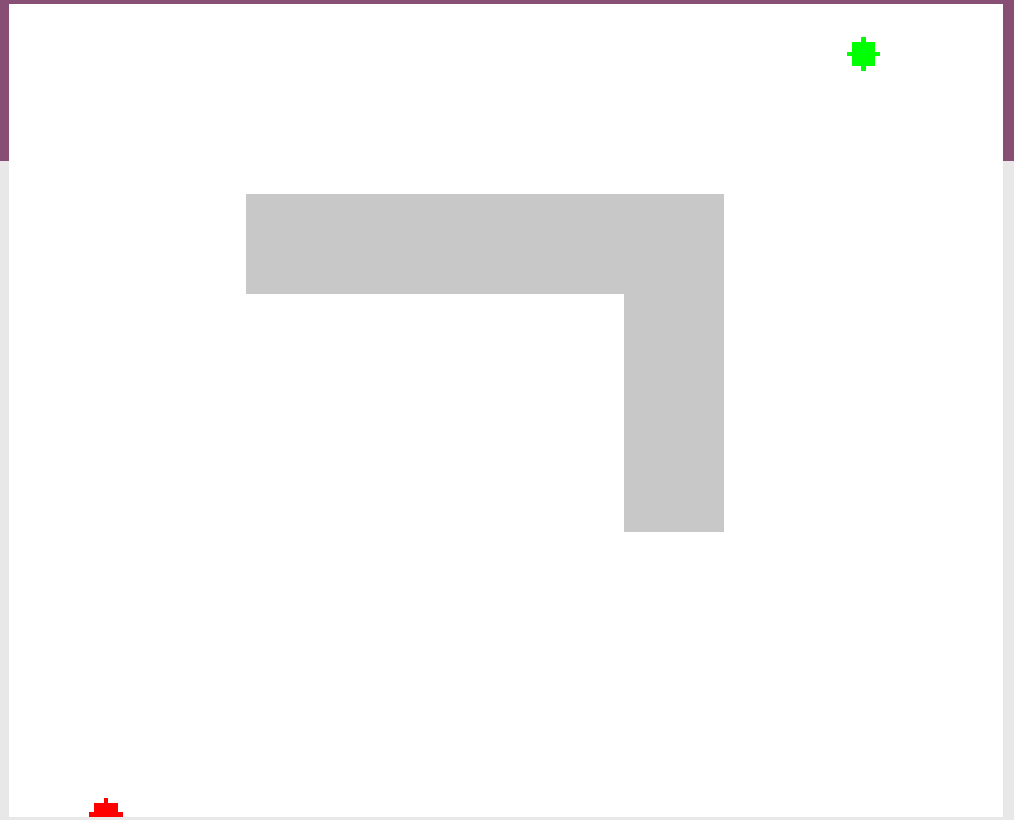
- Programmes **Dijkstra** / **Bellman-Ford**, lien ici (Jupyter) :

https://github.com/lmayer65/NSI_T/blob/main/Grand%20Oral/Chemin_Plus_Court/GO_Dijkstra_Bellman.ipynb

Algorithme A*

- **L’algorithme A*** : inventé en 1968, variante de l’algorithme de Dijkstra.

- **Vitesse de calcul** plutôt qu’**exactitude du résultat** : très utilisé dans les jeux vidéo exigeants et en IA.

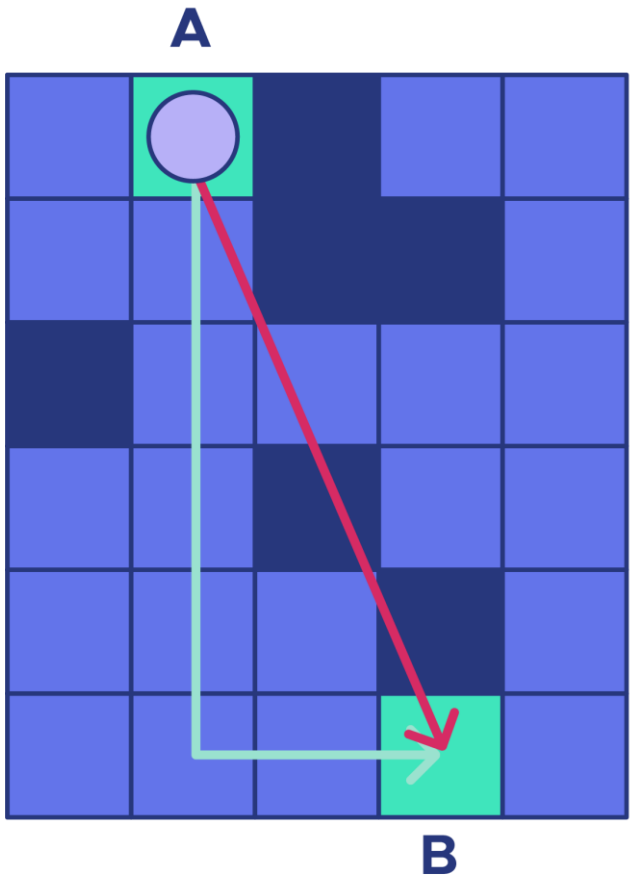


Algorithme A*

- **Alg. Déterministe** : défini précisément, solution automatisée et attendue à un problème (Dijkstra) et **démontré**.

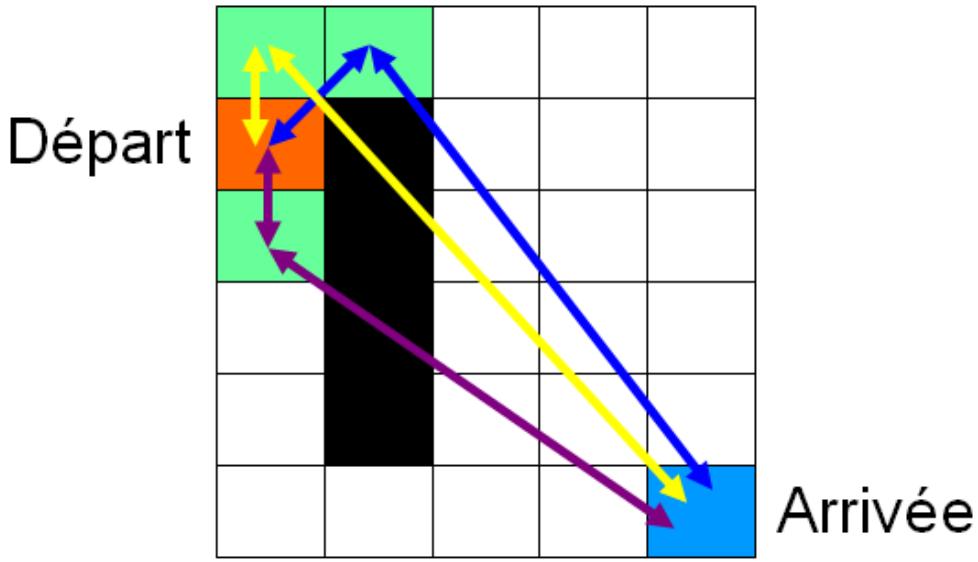
- **Heuristique** : « sorte d'algorithme » avec application d'un critère « arbitraire mais éclairé », pour trouver une solution acceptable rapidement mais **non démontré**.

Ici : **distance** (euclidienne, Manhattan)



Algorithme A*

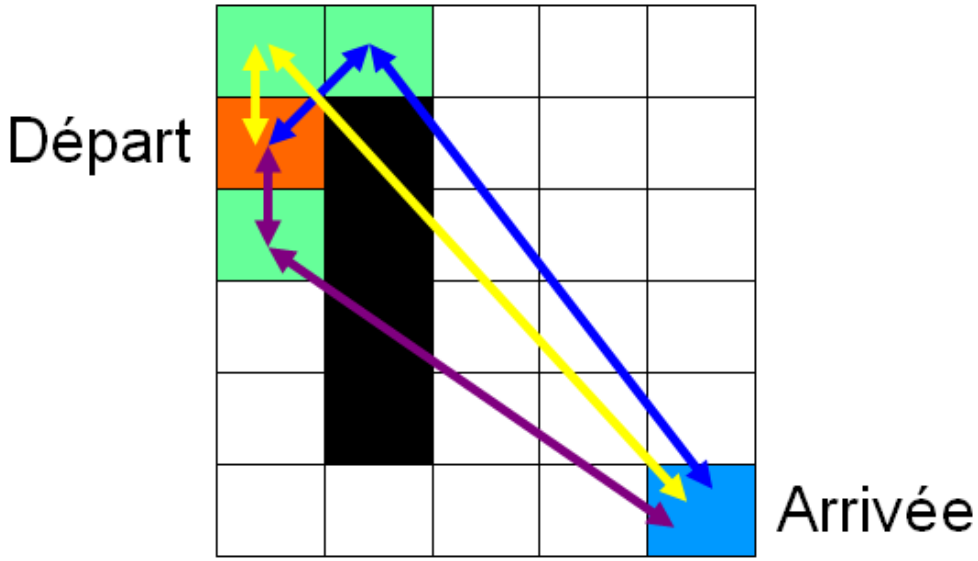
- **Principe** :
  - Une liste **ouverte** : contient les nœuds à étudier, à la frontière de la partie explorée du graphe (voisins).
  - Une liste **fermée** : contiendra les nœuds **déjà traités**, dans la frontière délimitée précédemment.
  - **Critère heuristique** : tri en fonction de *dist_réelle*(départ, nœud) + *dist_estimée*(nœud, arrivée), le plus petit possible.
  - Pour **chaque nœud choisi** : étude des **voisins** et accessibilité (liste ouverte si oui) + calcul de la qualité.
  - Le meilleur : dans la liste **fermée**.



Algorithme A*

Attention au choix du critère heuristique :

- **Ne pas surestimer** la distance restante : choix dit **non admissible** car pas de solution.
- Ne pas trop **sous-estimer** la distance restante : **augmentation** des temps de calcul.
- Critère heuristique nul : algorithme de Dijkstra.



Algorithme A*

- Un exemple concret: https://www.youtube.com/watch?v=lSzElQ2Belk

- Un lien ici : http://gdac.uqam.ca/inf4230/diapos/04-recherche-heuristique.pdf

(à partir de la page 16).