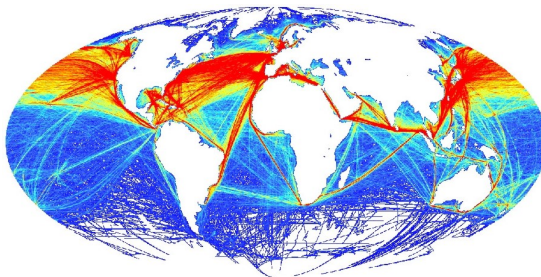


Optimisation opérationnelle des flux maritimes

Solutions algorithmiques & Application pratique



Comment minimiser le coût de l'itinéraire d'un navire marchand ?

1. Une première approche relationnelle
2. Une solution géométrique optimale
3. Développement d'une application pratique cartographique

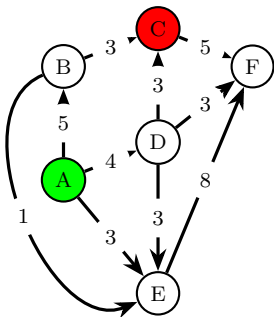
Lucas RODRIGUEZ

Vendredi 08 Novembre 2019 • 13h00

Introduction structurale

Soit $G = (S, A, \varphi, \psi)$ un graphe orienté pondéré positivement.

- ▶ S : ens des sommets
- ▶ A : ens des arrêtes orientées
- ▶ $\varphi : A \longrightarrow S^2$: fonction d'incidence
- ▶ $\psi : A \subsetneq S^2 \longrightarrow \overline{\mathbb{R}^+}$ fonction de pondération



- ▶ $\text{card}(S) = n \in \mathbb{N}^*$ et $\text{card}(A) = m \in \mathbb{N}^*$
- ▶ **Chemin** μ : séquence ordonnée de sommets :

$$\forall (x_1, \dots, x_n) \in S^n, \quad \mu = (x_1, \dots, x_n)$$

- ▶ Coût d'un chemin $d(\mu)$

$$d(\mu) = \sum_{i=1}^{n-1} \psi(x_i, x_{i+1})$$

Sommet de départ **D** et Sommet d'arrivée **A**


Approche relationnelle : Propriétés

Importance des chemins d'élémentaires : sans répétition d'arcs et de sommets

Th 1 (Lemme de König)

De tout chemin, on peut extraire un chemin élémentaire.

Th 2 (Existence des solutions)



Il existe un chemin de coût minimum de  à tout sommet si G ne possède pas de circuit à coût strictement négatif.

Unicité selon la géographie.

Th 3 (Propriété fondamentale d'un PCC)

Tout sous-chemin d'un chemin de coût minimum est un chemin de coût minimum.

Hypothèses de travail

1. Existence d'au moins un chemin, même défavorable entre  et 
2. S et A finis.

Solution par l'algorithme de DIJKSTRA

- ▶ **T** : sommets traités
- ▶ **U** : sommets à analyser
- ▶ **D** : coûts du PCC entre **D** et $x \in S$

Initialisation :

$\forall i \in S, D(i) \leftarrow +\infty; T \leftarrow []; U \leftarrow [D]; D(D) = 0;$

Traitement :

tant que $U \neq \emptyset$ faire

Choisir $i \in U$ tel que $D(i) = \min\{D(k), k \in U\}$ (Sommet jugé optimal)

Transférer i de **U** vers **T** (Changement d'état pour traitement)

pour $j \in \Gamma^+(i)$ faire

si $j \in U$ alors

si $D(j) > D(i) + \psi(i, j)$ alors

$D(j) \leftarrow D(i) + \psi(i, j);$

fin

fin

sinon

Ajouter j à **U**;

$D(j) \leftarrow D(i) + \psi(i, j);$

fin

fin

fin

Reconstruction du chemin optimal entre **D** et **A**

Amélioration possible : files de priorités

Traite efficacement des éléments dans l'ordre des priorités imposées.

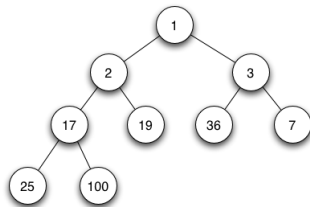


Soit h la hauteur d'un tas de n sommets :

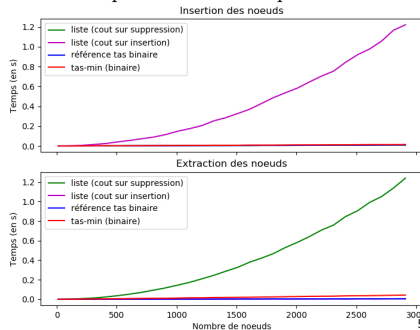
$$h \leq \lfloor \log_2 n \rfloor + 1$$

3 implémentations possibles

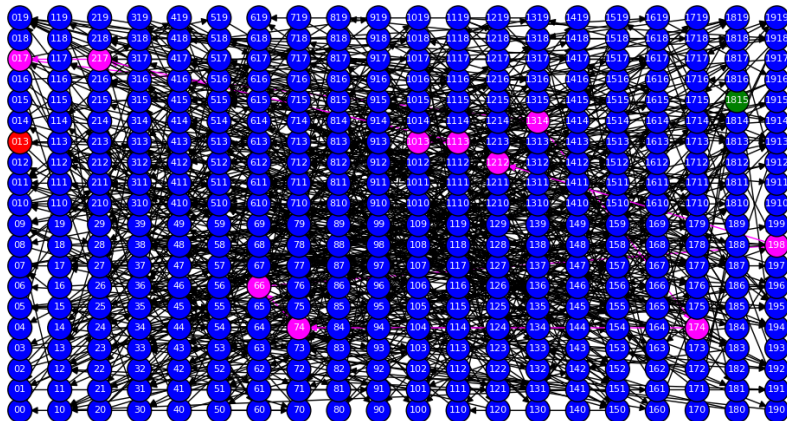
Structure	Insert.	Extract.
Liste (insert.)	$\mathcal{O}(n)$	$\mathcal{O}(1)$
Liste (extract.)	$\mathcal{O}(1)$	$\mathcal{O}(n)$
Tas binaire	$\mathcal{O}(n \log_2 n)$	$\mathcal{O}(1)$



Étude comparative des 3 implémentations

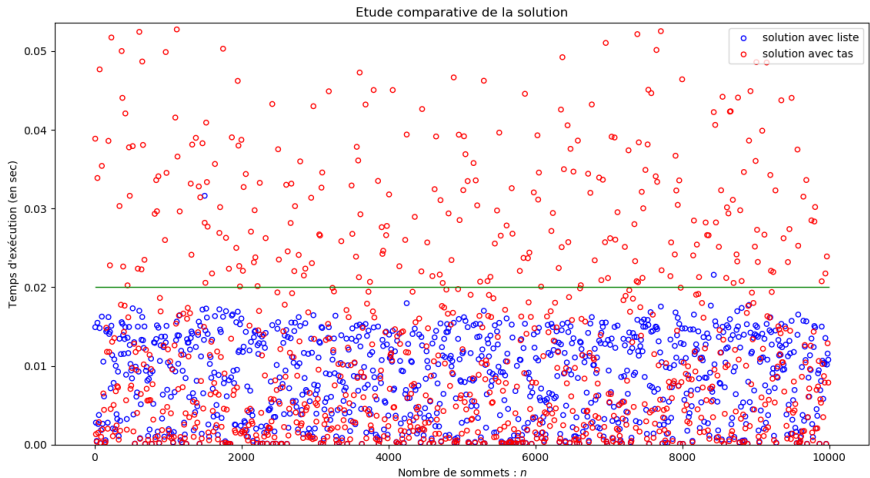


Étude expérimentale : Algorithme de DIJKSTRA



Temps d'exécution : 0.001382 sec ; $n = 400$

Étude expérimentale : Évolution temporelle pour plusieurs valeurs de n



Problème d'optimisation combinatoire où l'on doit minimiser \mathbf{D} sur $S - \textcircled{\mathbf{D}}$

Th 4 (Complexités dans le cas le plus défavorable)

Complexités spatiale en $\mathcal{O}(n + m)$ et temporelle en $\mathcal{O}(n^2)$ ou $\mathcal{O}(m + n \times \log_2(n))$.

Th 5 (Terminaison de l'algorithme)

Durée d'exécution finie.

Th 6 (Correction de l'algorithme)

En sortie de traitement $\forall x \in S - \textcircled{\mathbf{D}}$, $\mathbf{D}(x)$ est bien le coût du PCC entre $\textcircled{\mathbf{D}}$ et x .

Avantages

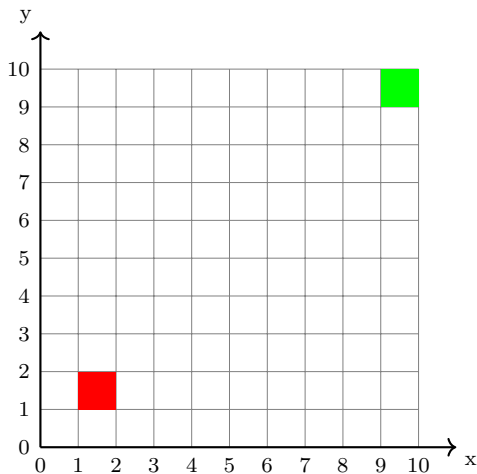
- ▶ Détermination assez efficace sur des simulations de moyenne taille
- ▶ Complexité spatiale tolérable
- ▶ Dispositif rapide & facile à mettre en place

Inconvénients

- ▶ Complexité quadratique peu satisfaisante pour les graphes de grande taille
- ▶ Perte d'information en se restreignant aux arcs
- ▶ Critère de minimisation peut être amélioré
- ▶ Ne permet pas de résoudre directement le problème

Approche géométrique : Présentation

Idée Quadrillage à coordonnées entières : $\llbracket 0, N - 1 \rrbracket^2 \subset \mathbb{N} \times \mathbb{N}$



$$\psi : S = \llbracket 0, N - 1 \rrbracket^2 \longrightarrow [\psi_{\min}, \psi_{\max}] \subset \mathbb{R}^+ \cup \{-1\}$$

Approche géométrique : Principe de fonctionnement d'A*

Remarque

La grille est également un GOP avec $n = N^2$ et $m = 2 \times 2(\sqrt{n} - 1)\sqrt{n} \sim 4n$.

Idée : Estimer également la distance à la cible (principe de l'algorithme A*, dérivé de DIJKSTRA)



Le sommet courant doit maintenant minimiser :

$$\begin{array}{lll} \Lambda & : & S \longrightarrow \mathbb{R}^+ \\ & & n \longmapsto \Lambda(n) = g(n) + h(n) \end{array}$$

- ▶ $g(n)$: coût entre **D** et n (identique à celle de DIJKSTRA)
- ▶ $h(n)$: distance **estimée** entre n et **A**

Approche géométrique : Détermination de h

3 systèmes de navigation envisagés :

NSEO
4 directions

N NE E SE S SO O NO
8 directions

360
Toute direction

$A = (x, y)$ et $B = (x', y')$ deux nœuds distincts.

► **4 directions** \rightsquigarrow Distance de Manhattan

$$d_1(A, B) = |x - x'| + |y - y'|$$

► **8 directions** \rightsquigarrow Distance de Tchebychev

$$d_\infty(A, B) = \max\{|x - x'|, |y - y'|\}$$

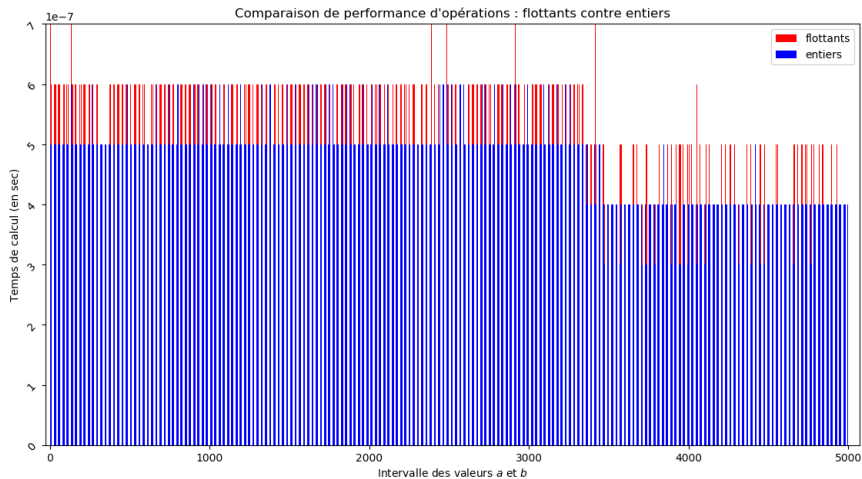
► **Toutes directions** \rightsquigarrow Distance euclidienne

$$d_2(A, B) = \sqrt{(x - x')^2 + (y - y')^2}$$

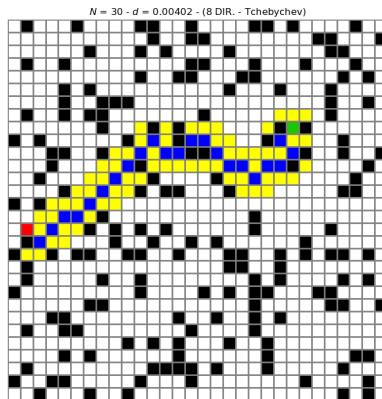
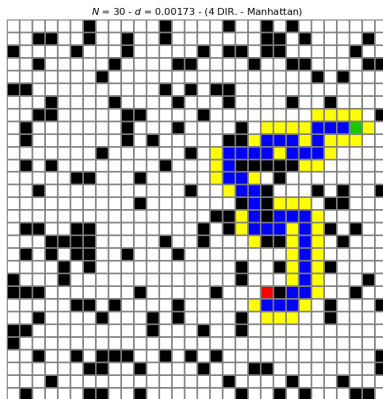
Inconvénient

Manipulation de nombres flottants avec la distance euclidienne. On préfère h^2 .

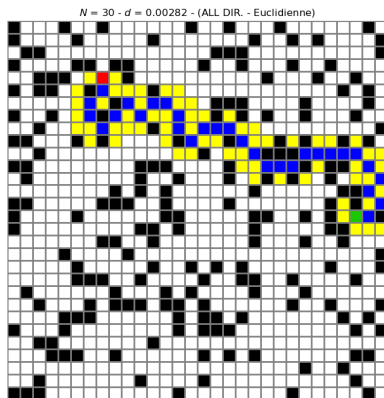
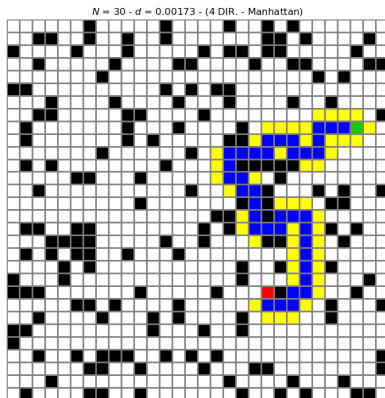
Comparaison des performances entre h et h^2



Comparaison expérimentale (1/2) : 4 vs 8 directions



Comparaison expérimentale (2/2) : 4 vs toutes les directions



Raffinement par « depixellisation récursive »

Définition d'un secteur :

$$\forall (i, j) \in \llbracket 1, a \rrbracket^2, S_{(i,j)}^a = \left\{ (x, y) \in \llbracket 0, N-1 \rrbracket^2, \left\lfloor x \times \frac{a}{N} \right\rfloor + 1 = i \text{ et } \left\lfloor y \times \frac{a}{N} \right\rfloor + 1 = j \right\}$$

Protocole

1. On exécute A^* sur les secteurs de la maille
2. Le secteur extrait minimise une somme globale
3. On segmente encore la grille (en gardant uniquement les secteurs pertinents)

Invariant

$$\forall a \in \mathbb{N}, \bigcup_{\substack{i \in \llbracket 1, a \rrbracket \\ j \in \llbracket 1, a \rrbracket}} S_{(i,j)}^a = \llbracket 0, N-1 \rrbracket^2$$

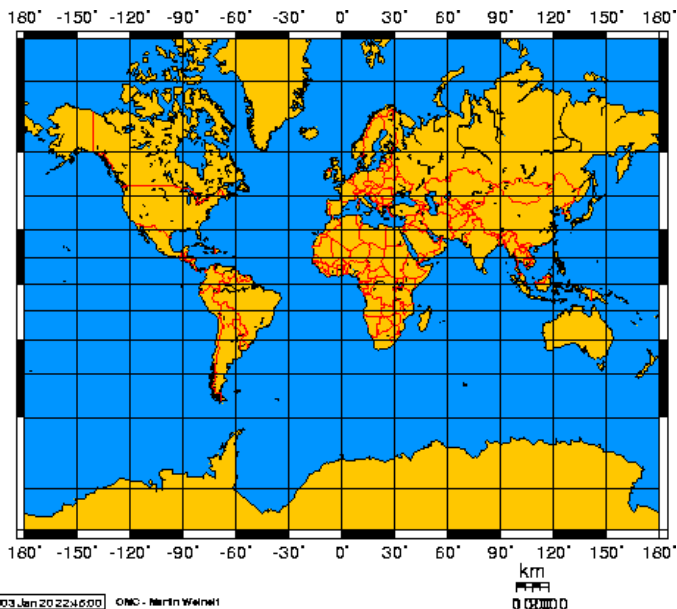
Cas terminal $a \geq N$

On obtient un nombre réduit de nœuds à traiter

Réduit le temps de calcul¹ et le nombre de nœuds à analyser

1. en calculant plusieurs fois Λ sur des ensembles de définitions plus restreints

Projection de Mercator



Passage du sphérique³ au planaire

Théorème 1 (Fonction de projection ψ)

$$\begin{aligned} \Pi :]-\pi, \pi[\times]-\frac{\pi}{2}, \frac{\pi}{2}[&\longrightarrow [-\pi, \pi] \times]-\infty, +\infty[\\ (\lambda, \varphi) &\longmapsto (x, y) = (\lambda, \ln |\tan(\frac{\pi}{4} + \frac{\varphi}{2})|) = (\lambda, \ln |\tan \varphi + \sec \varphi|) \end{aligned}$$

Quadrillage pour A^* :

$$\mathcal{A}_{\text{terre}} = 4\pi R_T^2 \simeq 510\,064\,471,9 \text{ km}^2$$

Nombre de secteurs² :

$$n = \left\lceil \frac{\mathcal{A}_{\text{terre}}}{\mathcal{A}_{\text{secteur}}} \right\rceil \sim 10^6$$

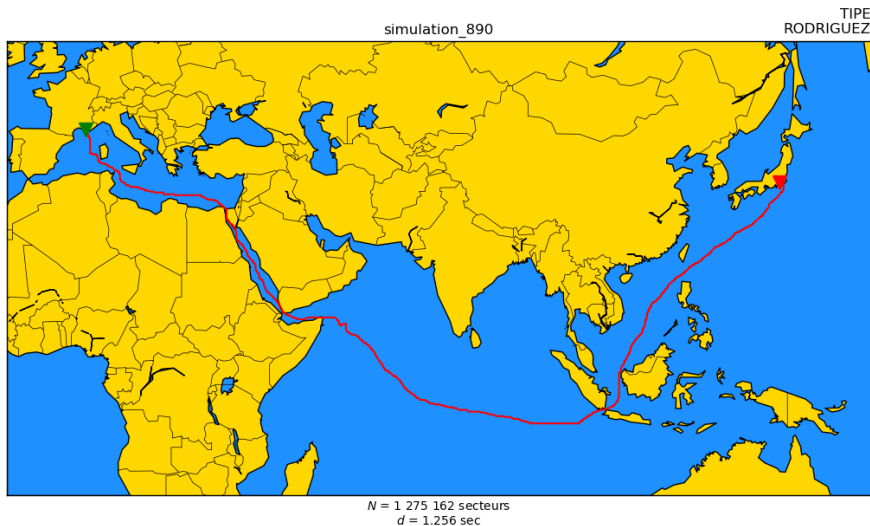
$$\psi : S = \llbracket 1, n \rrbracket \longrightarrow [\psi_{\min}, \psi_{\max}] \subset \mathbb{R}^+ \cup \{-1\}$$

- ▶ -1 réservé pour les continents/îles
- ▶ Aucun fleuve, ni rivière, pas de navigation côtière

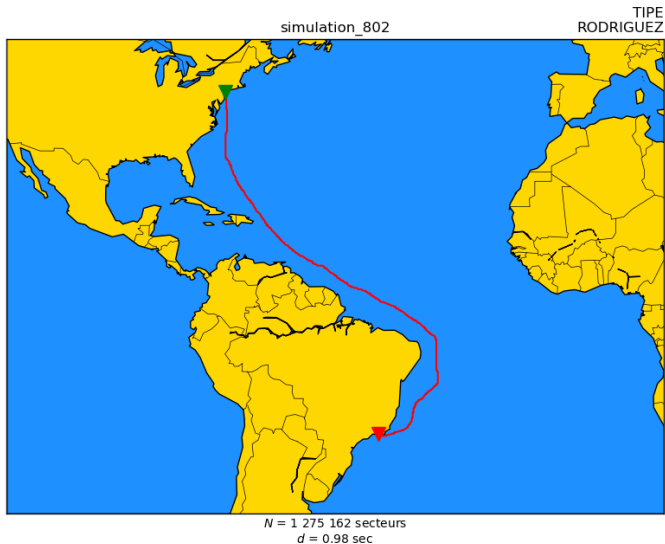
2. $\mathcal{A}_{\text{secteur}} = 400\text{km}^2$

3. λ : longitude et φ : latitude

Application 1 : Marseille ▷ Tokyo



Application 2 : New York ▷ Rio De Janeiro



Application 3 : Helsinki ▷ San Francisco



Entretien

Th 1

Lemme de König

De tout chemin, on peut extraire un chemin élémentaire.

Démonstration.

Raisonnons par récurrence sur le nombre de sommets l de μ , un chemin.

- ▶ si $l(\mu) = 0$: μ est élémentaire.
- ▶ soit $\mu = (x_1, x_2, \dots, x_k)$ avec $l(\mu) = k > 0$.
Si μ n'est pas élémentaire, $\exists(a, b) \in \llbracket 1, l(\mu) \rrbracket, a \neq b \implies x_a = x_b$, c-à-d :

$$\mu = (x_1, \dots, x_{a-1}, x_a, \dots, x_a, x_{b+1}, \dots, x_k)$$

Mais alors, en supprimant le cycle entre x_a et x_b , on obtient :

$$\mu' = (x_1, \dots, x_a, x_{b+1}, \dots, x_k)$$

μ' est strictement plus petit que μ . On peut donc extraire de μ' donc de μ un chemin élémentaire



Th 2

Existence d'un PCC (opérationnel pour DJK/A*)

Il existe un chemin de coût minimum de s à tout sommet si G ne possède pas de circuit à coût strictement négatif.

Démonstration.

Prendre un circuit possédant un arc à coût strictement négatif.

Puis conclure sur le caractère infini de la minimisation, qui doit être tout de même minorée par une borne inférieure fixée par l'utilisateur. ■

Th 3

Règle fondamentale de la programmation dynamique

Soit $\mu = (x_0, \dots, x_q)$ un chemin de coût minimum. Tout chemin $\gamma = (x_k, \dots, x_l)$ avec $0 \leq k \leq l \leq q$ est un chemin de coût minimum.

Tout sous-chemin d'un chemin de coût minimum est un chemin de coût minimum

Démonstration.

Soit λ un chemin de x_k à x_l différent de γ tel que $d(\lambda) < d(\gamma)$

Soit μ' , le chemin obtenu à partir de μ en remplaçant γ par λ . Alors $d(\mu') < d(\mu)$.

Contradiction avec le choix initial de μ . ■

Th 4

Majoration de la hauteur d'un tas binaire

La hauteur h d'un tas binaire \mathcal{H} peut être majorée :

$$h(\mathcal{H}) \leq \lfloor \log_2 n \rfloor + 1$$

Démonstration.

Si \mathcal{H} est un tas binaire de hauteur h et comportant n nœuds, alors :

$$\forall m \in \mathbb{N}, h \leq m - 1 \implies n < 2^{m-1}$$

Par contraposée, on a $n \geq 2^{m-1} \implies h \geq m$. D'où le plus grand entier m vérifiant la condition $n \geq 2^{m-1}$ est $m = \lfloor \log_2 n \rfloor + 1$. Donc $h \leq \lfloor \log_2 n \rfloor + 1$. ■

Th 5

Complexités dans le cas le plus défavorable

Complexités spatiale en $\mathcal{O}(n + m)$ et temporelle en $\mathcal{O}(n^2)$ ou $\mathcal{O}(m + n \times \log_2(n))$.

Démonstration.

Espace : 2 dictionnaires, 2 listes et un dictionnaire d'adjacence : $\mathcal{O}(n + m)$

- ▶ Sélection/Extraction du nœud :
 - ▶ avec liste : parcours linéaire pour chaque sommet de \mathcal{O} donc $\mathcal{O}(n)$.
 - ▶ avec tas H : $\mathcal{O}(\log_2(n))$ (car coût de l'insertion majorée par $h(H)$)
- ▶ Répété au plus n fois
- ▶ Relâchement des arcs : unique pour chaque arc d'où $\mathcal{O}(m)$
- ▶ avec liste : $\boxed{\mathcal{O}(m + n^2)}$
- ▶ avec tas : $\boxed{\mathcal{O}(m + n \times \log_2(n))}$



Th 6

Terminaison pour Dijkstra et A*

Durée d'exécution finie.

Démonstration.

Cdts d'arrêts : O vide **ou** extraction de c depuis O .

Comme V est fini, O et F également. D'après hypothèse 2, le nombre d'opérations est fini, d'où terminaison garantie. ■

Th 7 (1/2)

Echelle Nord - Sud (en φ) = Echelle Est - Ouest (en λ)

$$\forall (\lambda, \varphi) \in [-\pi, \pi] \times \left] -\frac{\pi}{2}, \frac{\pi}{2} \right[, \quad \frac{\frac{\partial x}{\partial \lambda}}{\frac{\partial y}{\partial \varphi}} = \frac{2\pi R_T \cos \varphi}{2\pi R_T}$$

Projection de Mercator : $\frac{\partial x}{\partial \lambda} = 1 \Rightarrow \boxed{x = \lambda}$

$$\frac{\partial y}{\partial \varphi} = \frac{1}{\cos \varphi} = \sec \varphi \quad (\star)$$

$$\frac{1}{\cos \varphi} = \frac{1}{\sin(\frac{\pi}{2} + \varphi)} = \frac{1}{2 \sin(\frac{\pi}{4} + \frac{\varphi}{2}) \cos(\frac{\pi}{4} + \frac{\varphi}{2})} = \frac{\frac{1}{2} \frac{1}{\cos(\frac{\pi}{4} + \frac{\varphi}{2})^2}}{\tan(\frac{\pi}{4} + \frac{\varphi}{2})} = \frac{\frac{\partial(\tan(\frac{\pi}{4} + \frac{\varphi}{2}))}{\partial \varphi}}{\tan(\frac{\pi}{4} + \frac{\varphi}{2})}$$

$$\Rightarrow \boxed{y = \ln |\tan(\frac{\pi}{4} + \frac{\varphi}{2})|}$$

Th 7 (2/2)

D'après (★) :

$$\begin{aligned}
 \int \sec \varphi \, d\varphi &= \int \frac{d\varphi}{\cos \varphi} \\
 &= \int \frac{\cos \varphi}{\cos^2 \varphi} \, d\varphi \\
 &= \int \frac{\cos \varphi}{1 - \sin^2 \varphi} \, d\varphi \\
 &= \int \frac{\cos \varphi}{(1 - \sin \varphi)(1 + \sin \varphi)} \, d\varphi \\
 &= \frac{1}{2} \int \frac{\cos \varphi}{1 - \sin \varphi} + \frac{\cos \varphi}{1 + \sin \varphi} \, d\varphi \\
 &= \frac{1}{2} (-\ln |1 - \sin \varphi| + \ln |1 + \sin \varphi|) + c, c \in \mathbb{K} \\
 &= \frac{1}{2} \ln \left| \frac{1 + \sin \varphi}{1 - \sin \varphi} \right| + c, c \in \mathbb{K} \\
 &= \frac{1}{2} \ln \left| \frac{1 + \sin \varphi}{1 - \sin \varphi} \times \frac{1 + \sin \varphi}{1 + \sin \varphi} \right| + c, c \in \mathbb{K} \\
 &= \frac{1}{2} \ln \left| \frac{(1 + \sin \varphi)^2}{1 - \sin^2 \varphi} \right| + c, c \in \mathbb{K} \\
 &= \frac{1}{2} \ln \left| \frac{(1 + \sin \varphi)^2}{\cos^2 \varphi} \right| + c, c \in \mathbb{K} \\
 &= \ln \left| \frac{1}{\cos \varphi} + \frac{\sin \varphi}{\cos \varphi} \right| + c, c \in \mathbb{K} \\
 &= \ln |\sec \varphi + \tan \varphi| + c \blacksquare
 \end{aligned}$$

Annexe : Fonctions d'initialisation, d'extraction/sélection et de sélection du voisinage

```
def initialisation(G, source):  
    """ Fonction d'initialisation """  
    INF = float("inf")  
    liste_ouverte = [source]  
    liste_fermee = []  
  
    distances = {sommet: INF for sommet in G}  
    parents = {sommet: sommet for sommet in G}  
    chemin = []  
    distances[source] = 0  
  
    return liste_ouverte, liste_fermee, distances, parents, chemin  
  
def selection_sommet_courant(liste_ouverte, distances):  
    """ Fonction de sélection du sommet courant """  
    sommet_courant = liste_ouverte[0]  
    for k in range(0, len(liste_ouverte)):  
        if distances[list_ouverte[k]] < distances[sommet_courant]:  
            sommet_courant = liste_ouverte[k]  
    return sommet_courant  
  
def successeurs_sommet_courant(G, sommet_courant):  
    """ Fonction retournant les successeurs du sommet courant dans G """  
    return G[sommet_courant]
```

Annexe : Fonctions de relâchement et de reconstruction

```
def relachement_sommet(sommet_courant, s, liste_ouverte, liste_fermee, distances,
    parents):
    """ Procédure de relâchement du sommet """
    nouveau_cout = distances[sommet_courant] + s[1]
    if s[0] in liste_ouverte:
        if nouveau_cout < distances[s[0]]:
            distances[s[0]] = nouveau_cout
            parents[s[0]] = sommet_courant
    else:
        liste_ouverte.append(s[0])
        distances[s[0]] = nouveau_cout
        parents[s[0]] = sommet_courant

def construction_chemin(source, cible, liste_ouverte, parents, chemin):
    """ Fonction de construction du chemin trouvé """
    if len(liste_ouverte) == 0:
        return []
    else:
        n = cible
        while n != source:
            chemin.append(n)
            n = parents[n]
        chemin.append(source)
        chemin.reverse()
        return chemin
```

Annexe : Corps principal de l'algorithme

```
def dijkstra(G, source, cible):  
    """ Fonction implémentant l'algorithme de Dijkstra """  
    assert source != cible  
  
    # Initialisation  
    liste_ouverte, liste_fermee, distances, parents, chemin = initialisation(G,  
    source)  
    while len(liste_ouverte) > 0:  
        # Sélection du sommet  
        sommet_courant = selection_sommet_courant(liste_ouverte, distances)  
        if sommet_courant == cible:  
            break  
        # Transfert du sommet  
        liste_ouverte.remove(sommet_courant)  
        liste_fermee.append(sommet_courant)  
  
        # Etape de relâchement de chaque successeur du sommet courant  
        for s in successeurs_sommet_courant(G, sommet_courant):  
            if s[0] in liste_fermee:  
                continue # passer au suivant  
  
            relachement_sommet(sommet_courant, s, liste_ouverte, liste_fermee,  
            distances, parents)  
  
    # renvoyer le chemin trouvé  
    return construction_chemin(source, cible, liste_ouverte, parents, chemin),  
    distances[cible]
```


Annexe : Corps principal de l'algorithme

```
class Noeud:
    def __init__(self, x, y, w):
        """ Méthode de construction d'une instance Noeud """
        self.x = x                # abscisse de la case
        self.y = y                # ordonnée de la case
        self.w = w                # pondération de la case
        self.empruntable = 0      # booléen indiquant s'il s'agit d'un obsta

        if self.w != -1:
            self.empruntable = 1

        self.g = 0                # coût g (distance source - noeud)
        self.h = 0                # coût h (distance noeud - cible)
        self.f = 0                # coût f = g + h
        self.parent = self        # parent du noeud (initialisé à lui-même)

    def __str__(self):
        """ Méthode de représentation du Noeud """
        return "Noeud ({}, {})".format(self.x, self.y)
```

Annexe : Norme IEEE-754 (rev 2008)



$$\forall x \in \mathbb{R}, x = (-1)^s \times 1.m \times 2^e$$

- ▶ s : signe (0 si $x > 0$ et 1 sinon)
- ▶ $m \in \{0, 1\}^{52}$: mantisse
- ▶ e : exposant $e = E - 1023$ avec $E \in [1, 2^{11} - 2]$

Annexe : numérisation d'un GOP

Matrice d'adjacence de G :

$$M = \begin{pmatrix} 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

Liste d'adjacence de G :

0 : [1, 4]
 1 : [6]
 2 : []
 3 : [1, 6]
 4 : []
 5 : [0, 1, 2]
 6 : []
 7 : [6]

Critères	Matrice d'adjacence	Liste d'adjacence
Complexité spatiale	$O(n^2)$	$O(n + m)$
Accès à un sommet	$O(1)$	$O(1)$
Parcours des sommets	$O(n)$	$O(n)$
Parcours des arcs	$O(n^2)$	$O(n + m)$
Existence d'un arc	$O(1)$	$O(d^+(x))$

TABLE – Comparaison détaillée des complexités spatiales et temporelles

Densité : $\Delta = \frac{|E|}{|V|^2} = \frac{m}{n^2}$ (faible si $\Delta < 0.3$) (page 4 pour ex. de GOP)