Projet d'algorithmique – Dijkstra

Théophile Bastian, Noémie Cartier

8 janvier 2016

1. Version naïve
   - Implémentation naïve
2. Tas de Fibonacci
   - Algorithme
   - Implémentation
3. Correction et performances
   - Correction
   - Performances
4. Bonus : Dijkstra dans Paris
   - Graphe de Paris : formatage
   - À une intersection donnée
   - Aux lignes RATP

Dijkstra

Idée : BFS par distance à l'origine croissante.

```
toProcess = file de priorité min vide
toProcess.insert(origine,0)
dists.fill(-1)
while(not toProcess.isEmpty()) {
    curNode,curDist = toProcess.pop()
    if(dists[curNode] >= 0)
        continue

    dists[curNode] = curDist
    foreach v in voisins[curNode] {
        toProcess.insert(v, curDist + dist[curNode, v])
    }
}
```

Complexité : $\mathcal{O}\left(|E|\left(Ins(|E|) + Pop(|E|)\right)\right)$.

Implémentation naïve

File de priorité : Tableau. Insertion en $\mathcal{O}(1)$, suppression en $\mathcal{O}(N)$
$\Longrightarrow$ Dijkstra en $\mathcal{O}(|E|^2)$.

Nécessite des tableaux à taille variable.

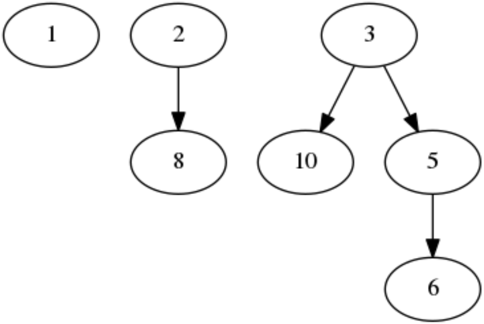
Graphes

- Nombre de sommets $|V|$ fixe
- Listes d'adjacence : tableau de $|V|$ tableaux à taille variable de voisins (et le poids de l'arête)

Tas de Fibonacci

Complexités

Insertion : $\mathcal{O}(1)$
Suppression : $\mathcal{O}(\log(N))$ (amorti)



Structures nécessaires

- Toutes les structures : fonction de création et de nettoyage mémoire
- Structure d'arbre
- Structure de liste doublement chaînée cyclique
- Structure de tas de Fibonacci

Arbre

- Structure récursive
- Pointeur `child`
- Pointeur `sibling`
- Portée des variables $\rightsquigarrow$ `malloc`

Liste doublement chaînée cyclique

- Pointeurs `prev`, `next`
- Toujours `malloc`
- Liste vide : pointeur sur `NULL`
- Gestion du passage de liste non-vide à liste vide et inversement
- Prendre un pointeur, en renvoyer un presque toujours identique
- Cycliques : permet de faciliter leur usage dans les tas de Fibonacci

Tas de Fibonacci

- `insert` : insertion d'un nœud dans la liste
- `pop` : retrait du plus petit nœud de la liste et ajout dans la liste de ses arbres fils ; fusion des arbres de taille identique
- Quelques difficultés sur la fusion qui n'était pas faite

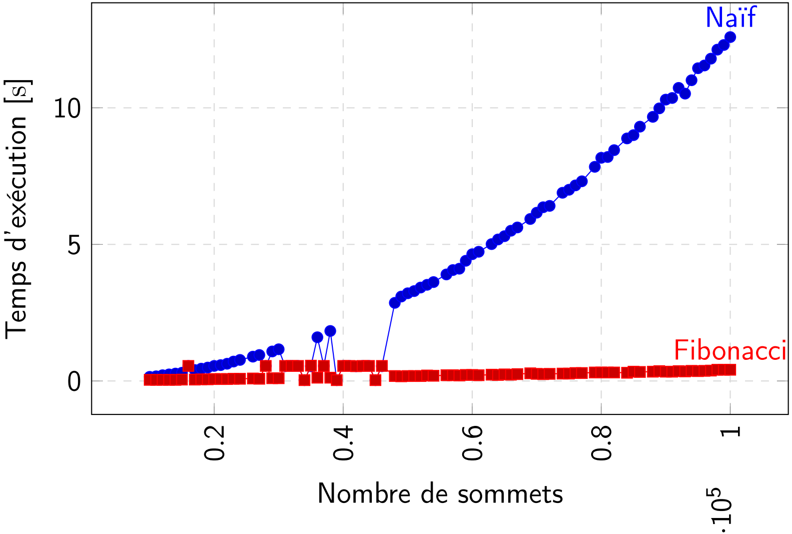
Correction

- Résultats vérifiés à la main sur de petits graphes
- Comparaison entre les deux implémentations et une implémentation de référence C++ utilisant la STL
- Tests automatisés sur 10000 graphes de grande taille
- Erreur détectée et corrigée pour des arêtes de poids 0
- Pas de fuite de mémoire : `valgrind`

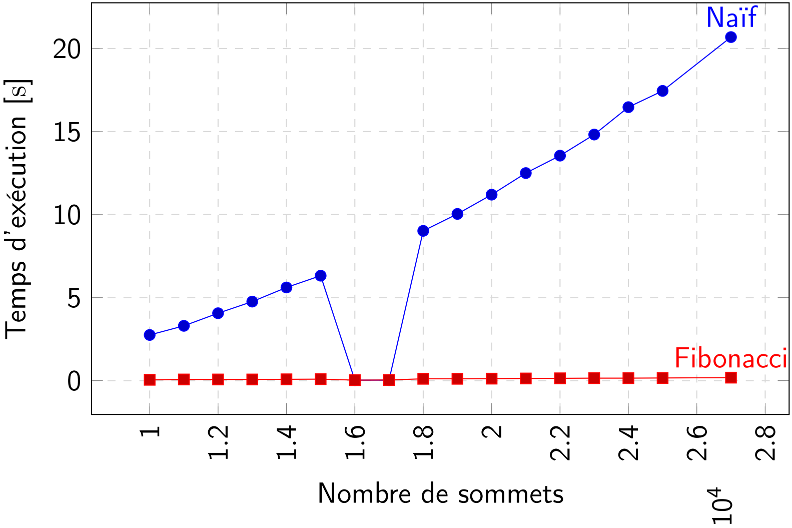
Performances

- Tests de performance effectués comparant nos deux implémentations sur de très grands graphes aléatoires de taille variable
- Fibonacci : quasi-linéaire
- Naïf : quadratique
- Quelques aberrations sur les plus petits graphes, liées à des problèmes de connexité

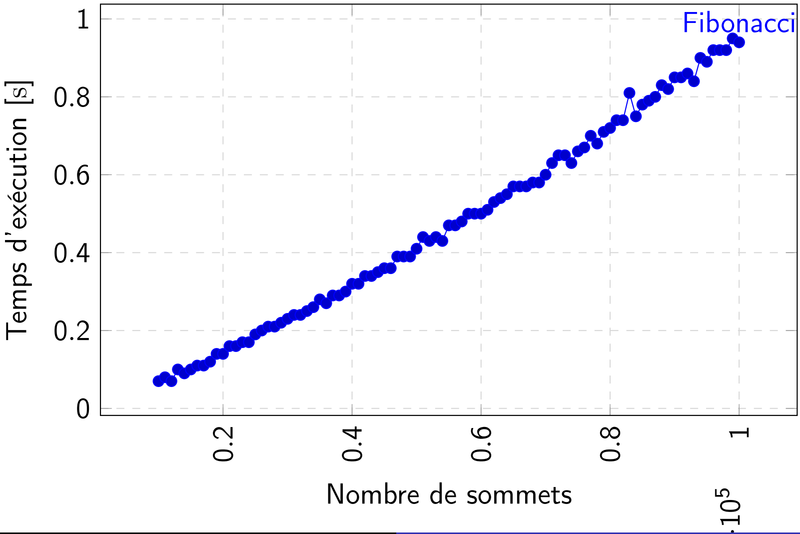
Comparatif – degrés faibles (1 – 3)



Comparatif – degrés élevés (0 – 10)



Fibonacci – degrés élevés (2 – 10)



Graphe de Paris : formatage

- Données de base : OpenStreetMaps
- Réindexer les nœuds pour avoir des IDs contigus partant de 0
- Maintenir une table de hachage pour garder la correspondance
- Arêtes : segments de `way` successifs, distance

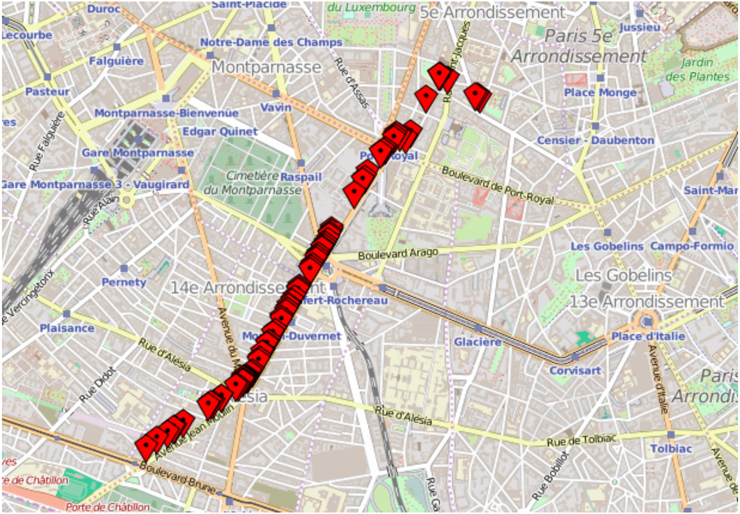
$$R_T \sqrt{(\Delta\Phi)^2 + (\cos(\Phi_m)\,\Delta\lambda)^2}$$

- Relations (OSM) ignorées
- Génère deux fichiers : un graphe *non-orienté* (option adaptée) et un fichier de correspondance des IDs.

Vers un lieu donné

- Se rendre à l'intersection du boulevard Brune et de l'avenue Jean Moulin (aka avenue de la Porte de Châtillon)
- Google maps : 2.9Km
- Notre Dijkstra : 2849m (cohérent)
- Premier essai sur graphe orienté : 3653m

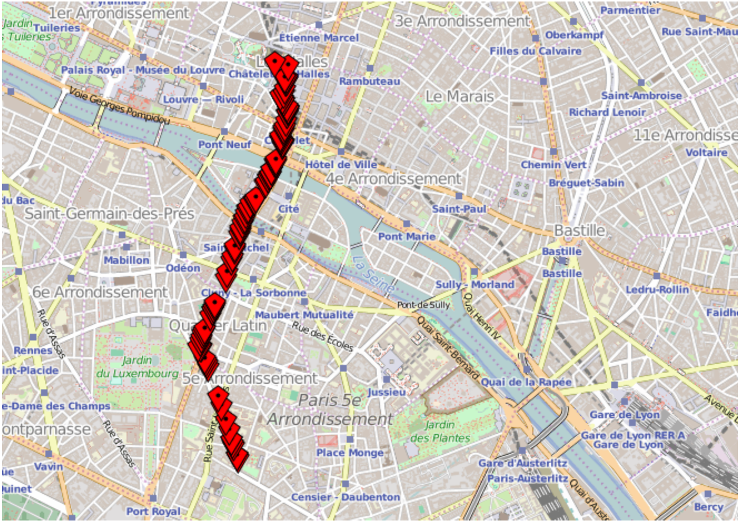
Carte associée



Aux lignes RATP

- Récupération des noms, latitude, longitude des stations par ligne
- Pour chaque ligne, chaque station, trouver tous les nœuds à moins de 50m
- Lier les station à ces nœuds par leur distance à vol d'oiseau
- Lier les stations à un nœud « ligne » avec poids 0
- Dijkstra
- Relier les stations : naïvement, $\mathcal{O}(\text{nbStat} \times |V|)$. Raffinage du graphe pour supprimer les zones non-connexes.

RER A


1er Arrondissement
Tuileries
Étienne Marcel
3e Arrondissement
Parmentier
Oberkampf
Rue Saint-Mau
Filles du Calvaire
Palais Royal - Musée du Louvre
Rambuteau
Le Marais
Saint-Ambroise
Voie Georges Pompidou
Louvre — Rivoli
Richard Lenoir
11e Arrondisse
Pont Neuf
Voltaire
Hôtel de Ville
Chemin Vert
4e Arrondissement
Bréguet-Sabin
Saint-Germain-des-Prés
Cité
Saint-Paul
Bastille
Pont Marie
Mabillon
Odéon
La Seine
Sully - Morland
Ledru-Rollin
6e Arrondissement
Cluny - La Sorbonne
Pont de Sully
Maubert Mutualité
Rue des Écoles
Quai Henri IV
Rue d'Assas
Quai Saint-Bernard
Rennes
Jardin du Luxembourg
5e Arrondissement
Quai de la Rapée
Jussieu
Paris 5e Arrondissement
Gare de Lyon
Gare de Lyon RER A
Jardin des Plantes
Place Monge
Gare d'Austerlitz
Paris-Austerlitz
Vavin
Censier - Daubenton
Bercy
Port Royal

Conclusion

- Notre Dijkstra semble marcher sur tous les exemples
- Si vous voulez prendre la ligne de bus 321, n'y allez pas à pied. 22,848Km, c'est long.