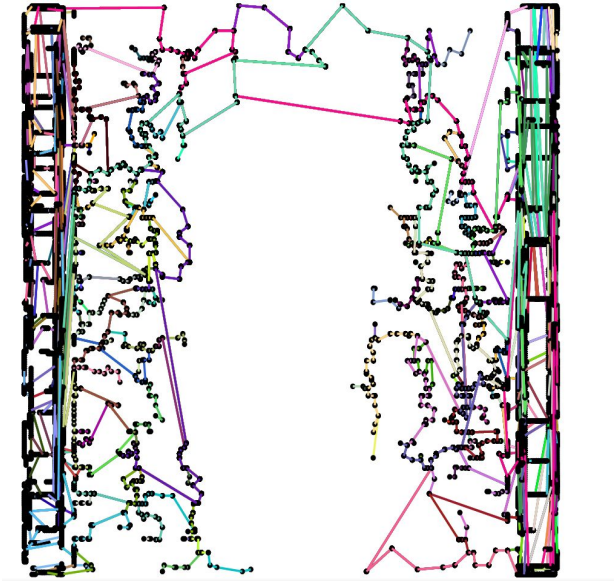
Projet - Réorganisation d'un réseau de fibres optiques

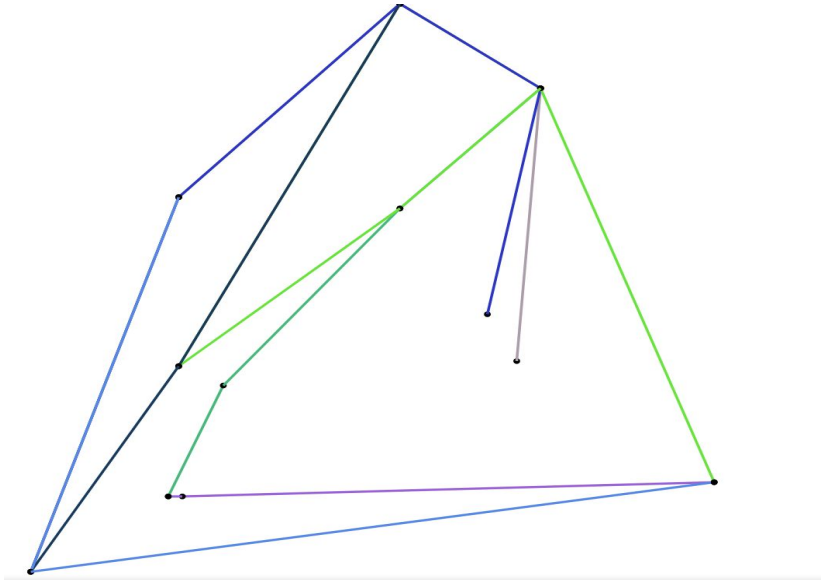


SCIENCES SORBONNE UNIVERSITÉ

Fait par SIMA Daniel et SADYKOVA Karima

Plan

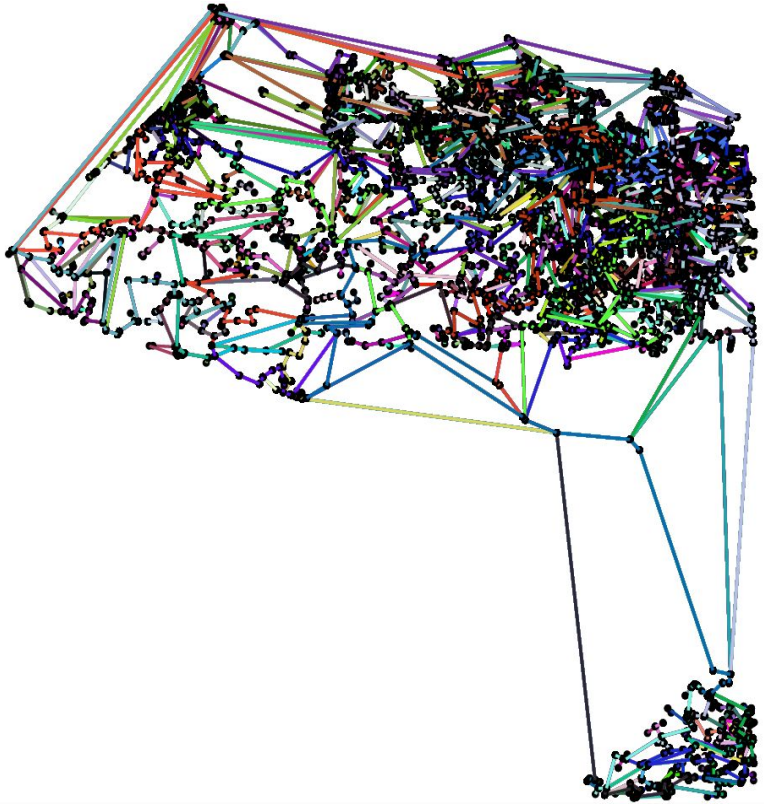
1. Introduction
2. Les structures utilisées
3. Conclusion



Introduction

En quoi consiste le projet ?

Comment nous nous sommes organisés?

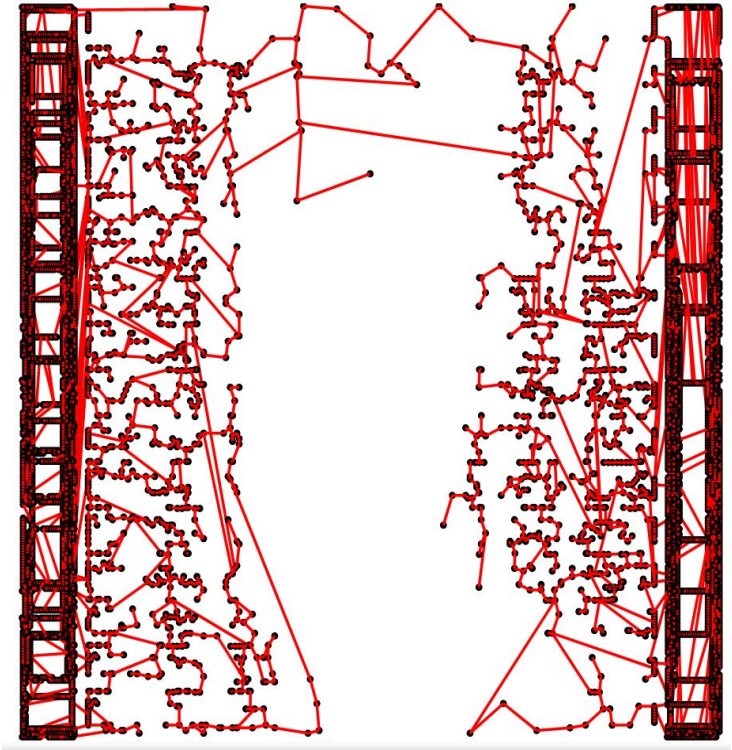


Structures de données utilisées

- ★ Listes chaînées
- ★ Table de hachage
- ★ Arbre quaternaire
- ★ Graphe
- ★ File (FIFO)

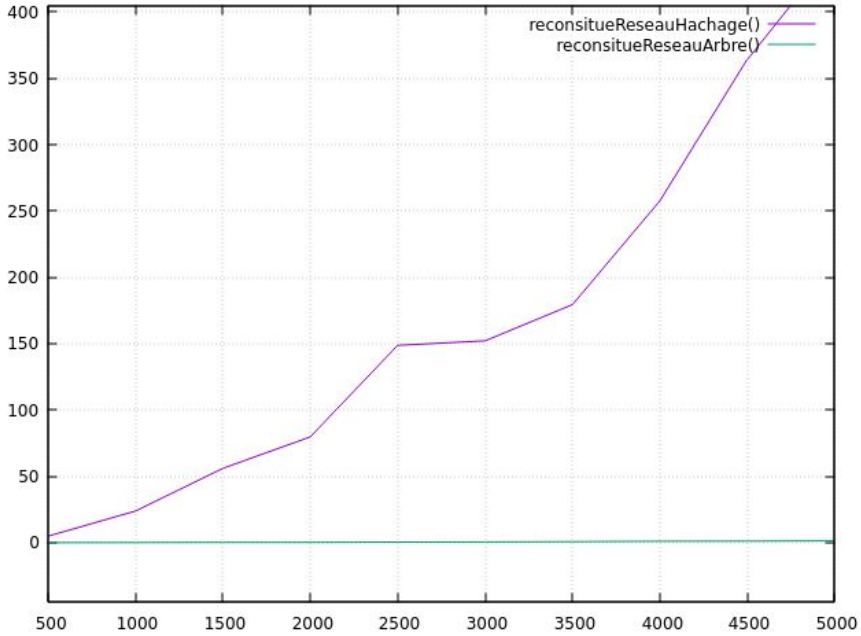
Comment avons nous implémenté ces structures ?

Quels algorithmes ?

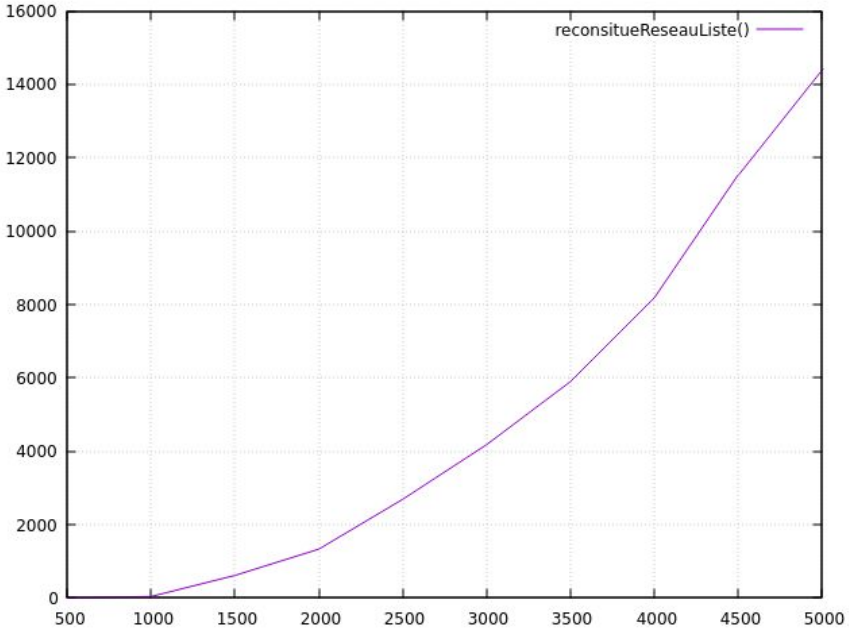


Quelle efficacité pour nos structures?

Temps mis par la Table de hachage et l'Arbre quaternaire:

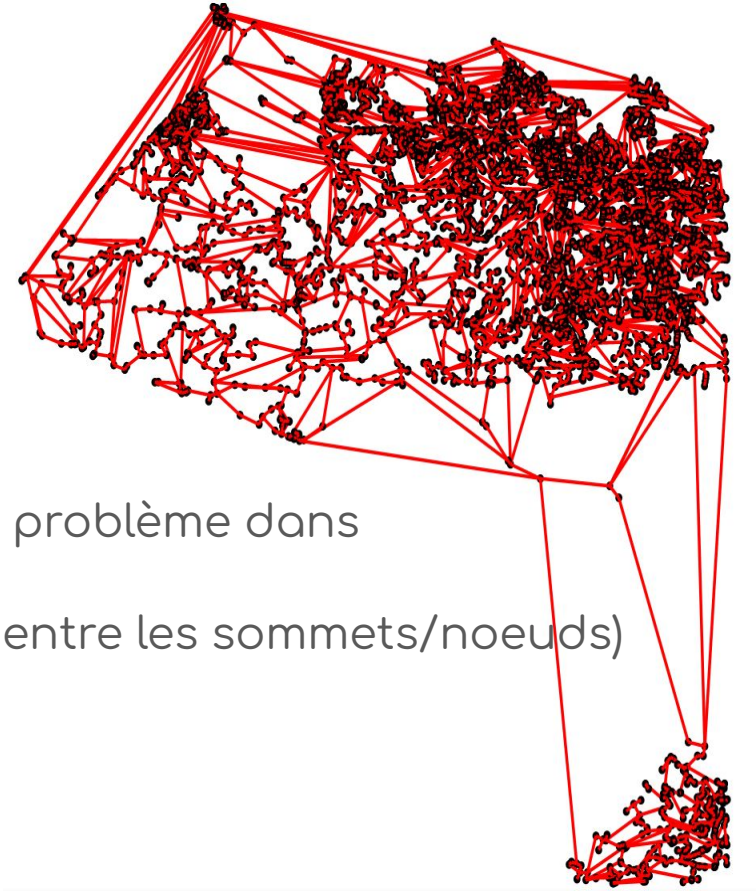


Temps mis par la Liste Chaînée:

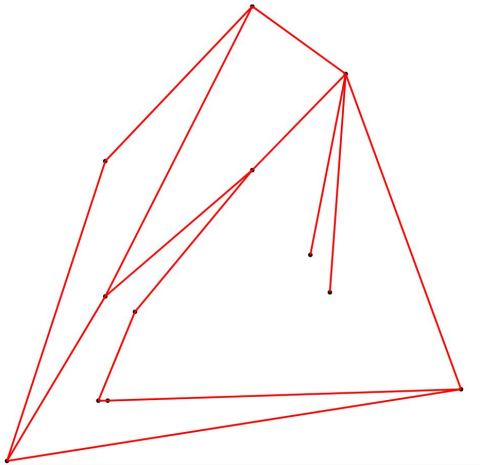


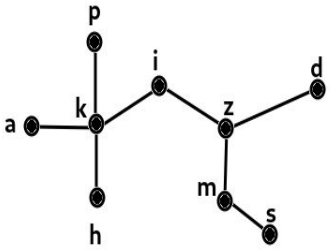
Conclusion

- Problèmes rencontrés
  - ➔ ReconstitutionReseauListe
  - ➔ InsererNoeudArbre
  - ➔ reorganiseReseau
- Ameliorations a proposer (?)
  - ➔ Récupération des arêtes qui posent problème dans reorganiseReseau()
  - ➔ Pondération des Graphes (distance entre les sommets/noeuds)
  - ➔ Orientation des Graphes
- Pourquoi utiliser un Arbre Quaternaire?



Merci pour votre attention !





/* Initialisaion */

arete[k]=k
arete[p]=p
arete[a]=a
arete[h]=h
arete[i]=i
arete[z]=z
arete[m]=m
arete[s]=s
arete[d]=d

arborescence[k]=k
arborescence[p]=p
arborescence[a]=a
arborescence[h]=h
arborescence[i]=i
arborescence[z]=z
arborescence[m]=m
arborescence[s]=s
arborescence[d]=d

/* Quand la S_file est vide*/

arete[k]=k
arete[i]=k->i,
arete[p]=k->p
arete[a]=k->a
arete[h]=k->h
arete[z]=i->z
arete[m]=z->m
arete[s]=m->s
arete[d]=z->d

arborescence[k]=k
arborescence[p]=p
arborescence[a]=a
arborescence[h]=h
arborescence[i]=i
arborescence[z]=z
arborescence[m]=m
arborescence[s]=s
arborescence[d]=d

/* A l'entrée du parcorus de tous les Sommets du Graphe*/

arborescence[k]=k, deja ok
arborescence[p]=p, traitement =>arborescence[p]=k->p ok
arborescence[a]=a, traitement =>arborescence[a]=k->a ok
arborescence[h]=h, traitement =>arborescence[h]=k->h ok
arborescence[i]=i, traitement =>arborescence[i]=i->p ok
arborescence[z]=z, traitement =>arborescence[z]=i->p pas encore ok
arborescence[m]=m, traitement =>arborescence[m]=z->m pas encore ok
arborescence[s]=s, traitement =>arborescence[s]=m->p pas encore ok
arborescence[d]=d, traitement =>arborescence[d]=z->p pas encore ok

↓ Traitement des cas pas ok

arborescence[z]=i->p, traitement => arborescence[z]=k->i->p ok
arborescence[m]=z->m, traitement => arborescence[m]=i->z->m pas encore ok
arborescence[s]=m->p, traitement => arborescence[s]=z->m->p pas encore ok
arborescence[d]=z->p, traitement => arborescence[d]=i->z->p pas encore ok

→

Traitement des cas pas ok

arborescence[m]=i->z->m, traitement => arborescence[m]=k->i->z->m ok
arborescence[s]=z->m->p, traitement => arborescence[s]=i->z->m->p pas encore ok
arborescence[d]=i->z->p, traitement => arborescence[d]=k->i->z->p ok

↑ Traitement des cas pas ok

arborescence[s]=i->z->m->p, traitement => arborescence[s]=k->i->z->m->p ok

Donc arborescence[i] donne le chemin allant de la racine (ici k) à i.