Algorithmique 1

L3 RI

Table des matières

1 Algorithmes

1.1 Tris

- ▷ Tri par insertion : $O(n^2)$
  Considérer chaque élément un à un pour l'insérer à sa bonne place (penser à un jeu de cartes)
- ▷ Tri fusion : $O(n \log n)$
  Paradigme diviser pour régner, diviser en deux sous-problèmes
- ▷ Tri Shell : $O(n^2)$
  Suite de tris par insertion sur chaque constituant d'une partition du tableau
- ▷ Tri par tas : $O(n \log n)$
  Utiliser une structure de file de priorité, ici un tas
- ▷ Tri sélection : $\mathcal{O}(n^2)$
  Mettre le plus grand à sa place, puis le suivant... Peu d'écritures. Utilisé pour le tri fusion en place.
- ▷ Optimalité : $\Omega(n \log n)$ nécessaire, arbres de décision

1.2 Arbres binaires

- Arbre binaire : $1 + h \leq n \leq 2^{h+1} - 1$
- Arbre binaire presque complet : $2^h \leq n \leq 2^{h+1} - 1$
- Tas
- Arbre binaire de recherche (ABR)
  La recherche d'un élément ne suit qu'une branche, problème si arbre non équilibré
- Arbre AVL
  Rééquilibrage d'un arbre par des rotations : $\log_2(n+1) \leq h \leq 1.44 \log_2 n$

1.3 Graphes

- Graphes orientés, pondérés
- Implémentations par liste d'adjacence ou matrice d'adjacence
- Parcours en profondeur
  Valeurs de `pre` et `post` traitement, types d'arc, détection de cycles, tri topologique
  Composantes fortement connexes, Algorithme de Kosaraju (un premier PP, puis un second PP dans l'ordre décroissant des temps de `post` sur le graphe transposé), graphe quotient
- Parcours en largeur
  Recherche d'un plus court chemin
  Algorithme de Dijkstra (mise à jour de distances, et considérer le sommet qui minimise)
  Algorithme A* (même principe, mais le choix se base en plus sur une heuristique)
- Arbre couvrant de poids minimal
  Algorithme de Kruskal (utiliser une structure Union-Find, trier les arêtes par poids croissants, et les considérer toute une à une, si pas dans la même classe, on fusionne)
  Algorithme de Prim (similaire à Dijkstra, tant qu'il reste des sommets non traités, on prend l'arête qui minimise à partir d'un sommet traité)

1.4 Algorithmes gloutons

- Prendre un choix localement meilleur
- Algorithmes de Kruskal, de Prim
- Rendu de monnaie
- Couverture d'ensemble (exemple où le choix glouton ne donne pas forcément la solution optimale)

1.5 Programmation dynamique

- Paradigme de conception d'algorithmes
- Définir les sous-problèmes, en revoyant à la baisse l'objectif si nécessaire
- Trouver une relation de récurrence
- Écrire l'algorithme (mémoïzation)
- Exemples
  Recherche plus court chemin dans un graphe

— Algorithme de Floyd-Warshall
  $d_{i,j,k}$ = distance minimale d'un chemin allant de i à j passant par les k premiers sommets
— Algorithme de Bellman-Ford
  $d_{i,j,k}$ = distance minimale d'un chemin allant de i à j contenant au plus k arcs

Recherche plus longue sous-suite croissante
Problème du sac à dos

1.6 Flots

- ▷ Problème du flot maximal
- ▷ Algorithme de Ford-Fulkerson (Tant qu'il existe un chemin de la source à la cible dans le graphe résiduel, maximiser les flux sur ce chemin)
- ▷ L'algorithme se termine si les poids sont entiers (ou rationnels), sinon ne termine pas forcément
- ▷ Réduction du problème de couplage maximal au problème de flot maximal

1.7 Programmation linéaire

- ▷ Forme canonique
- ▷ Algorithme du simplexe (Tant qu'on peut maximiser la solution, échanger deux variables en utilisant l'expression la plus contraignante)

2 Structures de données

2.1 Files de priorité

Implémentées par exemple avec un tas.
Méthodes :

- ▷ Enfiler
- ▷ Défiler un élément maximal
- ▷ Est vide ?
- ▷ Construire file vide

2.2 Tables de hachage

Méthodes :

- ▷ Ajout d'un élément
- ▷ Suppression d'un élément
- ▷ Contient x ?

Risque de collisions, n'est pas rare (idem paradoxe des anniversaires)

2.3 Structure Union-Find

Méthodes :

- ▷ Créer partition
- ▷ Fusionner deux classes (union)

- ▷ Obtenir un représentant (find)

Implémentation par une forêt d'arbres. Complexité améliorée en utilisant la compression de chemin.

3 Complexité

Master Theorem Soit $a \geq 1,\ b \geq 0$ et $d \geq 2$. Si $T \in \mathbb{R}^{\mathbb{N}}$ vérifie

$$T(n) = a \cdot T\left(\frac{n}{d}\right) + O\left(n^b\right)$$

alors

$$T(n) = \begin{cases} O\left(n^b\right) & \text{si } b > \log_d(a) \\ O\left(n^b log_d(n)\right) & \text{si } b = \log_d(a) \\ O\left(n^{log_d(a)}\right) & \text{si } b < \log_d(a) \end{cases}$$

4 Autres

- ▷ Encodage de Huffman
- ▷ Formules de Horn
- ▷ FFT
- ▷ Classes P, NP, EXPTIME
- ▷ Classe NP : Réduction à SAT, Branch&Bound, Local Search