

Informatique 08

Files de priorité

Heapsort

Graphes

François Bourdoncle

`Francois.Bourdoncle@ensmp.fr`

`http://www.ensmp.fr/~bourdonc/`

Files de priorité

- Ensemble de valeurs totalement ordonnées (int, String, float, ...)
- Opérations abstraites

```
void ajouterValeur(Valeur v);
Valeur enleverMaximum();
```
- Utilisations
 - Gestion des files d'attente (clients)
 - Tri (heapsort)

Implémentation : arbre binaire complet

```
class FilePriorite {
    int taille;
    Valeur[] val;

    FilePriorite(int tailleMax) {
        taille = 0;
        val = new Valeur[tailleMax];
    }

    static int pere(int i) {
        return (i - 1) / 2;
    }

    static int filsG(int i) {
        return 2 * i + 1;
    }

    static int filsD(int i) {
        return 2 * i + 2;
    }

    static boolean existeFilsD(int i) {
        return filsD(i) < taille;
    }

    static boolean existeFilsG(int i) {
        return filsG(i) < taille;
    }

    void ajouterValeur(Valeur v) { ... }
    Valeur enleverMaximum() { ... }
}
```

Insertion

```
void ajouterValeur(Valeur v) {
    if (taille == val.length) {
        erreur("File pleine");
    }
    int i = ++taille;
    while (i > 0 && val[pere(i)] < v) {
        val[i] = val[pere(i)];
        i = pere(i);
    }
    val[i] = v;
}
```

Suppression

```
Valeur enleverMaximum()
{
    if (taille == 0) {
        erreur("File vide");
    }
    Valeur max = val[0];
    val[0] = val[--taille];
    int i = 0;
    Valeur v = val[0];
    while (existeFilsG(i)) {
        int j;
        if (val[filsG(i)] > v) {
            j = filsG(i);
        } else {
            j = i;
        }
        if (existeFilsD(i) && val[filsD(i)] > val[j]) {
            j = filsD(i);
        }
        if (i == j) {
            break;
        } else {
            val[i] = val[j];
            i = j;
        }
    }
    val[i] = v;
    return max;
}
```

Heapsort

```
static void heapsort(Valeur[] t)
{
    // Allouer une file de la taille du tableau
    FilePriorite file = new FilePriorite(t.length);

    // Ajouter tous les éléments du tableau
    for (int i = 0 ; i < t.length ; ++i) {
        file.ajouterValeur(t[i]);
    }

    // Extraire les éléments de la file un à un
    for (int i = t.length - 1 ; i >= 0 ; --i) {
        t[i] = file.enleverMaximum();
    }
}
```

- Remarque : tri en place possible
- Complexité : $O(n \log(n))$

Graphes

– Exemples

- Carte des transports aériens
- Circuit électrique
- Succession de tâches
- ...

– Questions

- Comment payer le moins cher pour aller de Lyon, France à Paris, Texas ?
- Capacité totale d'un réseau de transport ?
Combien suffit-il de couper de liaisons pour qu'il soit déconnecté ?
- Simulation d'un circuit intégré
- Dans quel ordre effectuer ces tâches ?
- ...

Graphes : représentations

- Sommets + arêtes
- Matrice d'adjacence

```
class Graphe {
    boolean adj[][];
}
```
- Tableau de fils

```
class Graph {
    List fils[];
}
```
- Liste de noeuds avec fils

```
class Graph {
    int num;
    List fils;
    Graph next;
}
```

Graphes particuliers

- Graphes dirigés
- Graphes avec coûts
- Graphes dirigés acycliques (DAG)
- Arbres (DAG sans partage)
- Forêts = collections d'arbres
- etc.

Parcours en profondeur

- **Définition**
 - Parcours récursif
 - Marquage des sommets déjà visités
- **Complexité** : $O(|\text{Sommet}| + |\text{Arêtes}|)$
- **Applications**
 - Existe-t-il un chemin de A à B ?
 - Enumération des sommets
 - Le graphe a-t-il un cycle ?
 - Composantes connexes du graphe
 - Tri topologique
 - “Arborescence de Trémaux”
 - *Forêt recouvrante*
 - Arcs de descente, de retour, transverses

Composantes connexes d'un graphe non dirigé

```
class Graph {
    List fils[];

    Graph(...) { ... }

    void visiter(boolean[] vu, int i) {
        if (!vu[i]) {
            vu[i] = true;
            writeln("-> " + i);
            List a = fils[i];
            while (!List.isEmpty(a)) {
                visiter(vu, a.head);
                a = a .tail;
            }
        }
    }

    void composantes() {
        boolean vu = new boolean[fils.length];
        int n = 0;
        for (int i = 0 ; i < vu.length ; ++i) {
            if (!vu[i]) {
                ++n;
                writeln("Composante numéro " + n);
                visiter(vu, i);
            }
        }
    }
}
```

Tri topologique d'un DAG

- Problème : énumérer les sommets du graphe, de façon à ce que chaque nœud apparaisse après tous ses prédécesseurs dans le graphe.
- Solution
 1. Faire un parcours en profondeur
 2. Imprimer chaque noeud i en sortie de `visiter(i)`
 3. On obtient un tri topologique inverse
- Complexité : $O(|Sommet| + |Arêtes|)$
- Applications
 - Ordonnancement de tâches
 - Simulation de circuits
 - Compilation
 - etc.

Tri topologique

```
class Graph {
    List fils[];

    List tri(boolean[] vu, int i, List l) {
        if (vu[i]) {
            return l;
        } else {
            vu[i] = true;
            List a = fils[i];
            while (!List.isEmpty(a)) {
                l = tri(vu, a.head, l);
                a = a .tail;
            }
            return List.cons(i, l);
        }
    }
}

List tri() {
    boolean vu = new boolean[fils.length];
    List l = List.empty;
    for (int i = 0 ; i < vu.length ; ++i) {
        if (!vu[i]) {
            l = tri(vu, i, l);
        }
    }
    return l;
}
}
```

Parcours en largeur d'abord

- Utilisation d'une file FIFO (First In First Out)
- Algorithme itératif

```
class FIFO { ... }
class Graph {
    List[] fils;

    void largeur() {
        boolean vu = new boolean[fils.length];
        FIFO file = new FIFO();
        for (int i = 0 ; i < vu.length ; ++i) {
            if (!vu[i]) {
                // Ajouter une racine
                vu[i] = true;
                file.ajouterDernier(i);
                // Boucle principale
                while (!file.estVide()) {
                    int j = file.enleverPremier();
                    writeln(j);
                    // Ajouter les fils
                    List a = fils[j];
                    while (!List.isEmpty(a)) {
                        if (!vu[a.head]) {
                            vu[a.head] = true;
                            file.ajouterDernier(a.head);
                        }
                        a = a.tail;
                    }
                }
            }
        }
    }
}
```