POLYTECH
PARIS-UPMC

Devoir maison d'algorithmique

Les Tas Binomiaux

Encadré par : Cécile Braunstein
Rédigé par : Fatine Bentires

Contents

Part I
Introduction

Certaines actions faciles à effectuer peuvent apparaître comme futiles à première vue. Cependant, si il est nécessaire de les effectuer à plusieurs reprises, il devient intéressant de les automatiser. Un tas est le plus souvent utilisé pour résoudre des problèmes qui doivent gérer une répétition de recherche de minimum. [1] Un exemple d'utilisation est la simulation de tâches répétitives dans les jeux vidéos. Un type particulier et très intéressant de tas est le tas binomial. De même un filtre de bloom permet de réduire considérablement le temps de recherche. Mais à quelles sont leurs particularités et comment le représenter ? C'est ce que nous allons voir dans la suite de ce rapport. Pour cela, nous allons commencer par en définir les grands principes, puis nous allons l'implémenter et enfin nous allons dégager les spécificités.

Part II
Sujet 7: Tas binomiaux

1 Explication du problème traité

1.1 Définition des éléments récurrents

On va souvent entre parler de noeud, de père, de fils et de bien d'autres termes c'est pourquoi il est important de les introduire.

Prenons un arbre, un **noeud** va correspondre à l'élément à partir duquel les branches vont bifurquer. Un noeud peut ainsi avoir au plus deux fils et donc deux branches. Ces fils seront nommés **fils gauche** et **fils droit**.

Il y aura deux types de noeuds, le premier que l'on appelle la **racine** et les autres que l'on appelle les **noeuds internes**.

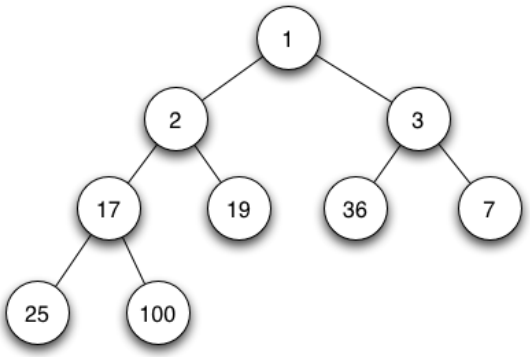
Un noeud qui n'a pas d'enfant est une **feuille** de l'arbre.

Un arbre est dit **complet** est un arbre dont tous les noeuds internes sont de degré 2 et possèdent donc deux fils.

On peut ainsi introduire deux termes importants qui sont la **profondeur** et la **hauteur** de l'arbre. La profondeur d'un noeud est le nombre d'arrêtes entre lui et le sommet. Le sommet aura une profondeur de 0 et ses enfants une profondeur de leur valeur + 1. La hauteur d'un arbre est tout simplement sa profondeur maximale.

1.2 Tas binaire

Après avoir introduit les éléments de base, continuons par définir proprement ce qu'est qu'un Tas. Lorsqu'on parle de tas, on entend généralement par là un tas binaire. Un tas binaire est une structure de données. Il s'agit également d'un arbre binaire ordonné: tous les noeuds, autre que la racine, ont une priorité plus grande que leur père. Elle permet d'accéder au minimum et au maximum d'un ensemble d'éléments.



1.3 Tas binomial

Un tas binomial est une structure de données proche du tas binaire. Il s'agit d'un ensemble d'arbres binomiaux disjoints. Le tas binomial a des caractéristiques plus étendues. Il permet notamment:

1. D'insérer un nouvel élément
2. De trouver l'élément de plus petit indice
3. De supprimer du tas un élément
4. **De fusionner deux tas en un seul**

Vous l'aurez compris, c'est la fusion de deux tas en un seul qui le rend spécial. Dans la suite de ce rapport nous allons nous intéresser aux tas binomiaux.

Pour construire un arbre binomial de rang i, il faut dans un premier temps que l'arbre B_0 ne contienne qu'un unique nœud.

Dans un second temps, pour les arbres avec i sup 0, B_i est construit à partir de 2 arbres binomiaux B_{i-1} en rajoutant l'un comme fils le plus à gauche de la racine de l'autre:

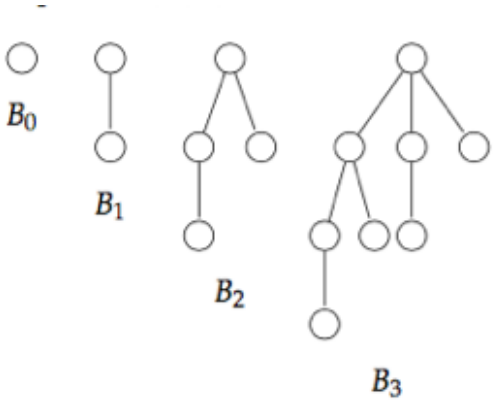


FIGURE 11 – Exemples d'arbres binomiaux

1.3.1 Propriétés du tas binomial

Comme expliqué sur l'énoncé du sujet, un tas vérifie certaines propriétés:

1. Dans chaque arbre, la valeur de chaque nœud est inférieure à la valeur de ses fils
2. Les arbres sont tous d'indices différents
3. Les arbres sont triés par indices strictement croissants

2 Algorithmique

2.1 Solution algorithmique

2.1.1 Les structures crées

2.1.2 Les principales fonctions d'un tas

2.2 Fonctions pour créer un tas

2.2.1 Initialisation

2.2.2 Création de noeud

2.2.3 Création d'arbre

2.2.4 Création de tas

2.3 Fonctions pour l'interaction d'un tas

2.3.1 Trouver le minimum

2.3.2 Insérer un élément

2.3.3 Supprimer un élément

3 Etude de complexité

3.1 La complexité qu'est-ce que c'est?

3.1.1 Principe

3.1.2 Ce que ça nous apprend

3.2 Tas binaire vs tas binomial

3.2.1 Tas binaire

3.2.2 Tas binomial

3.3 Avantages et limites d'un tas binomial

Part III
Sujet 6: Le filtre de Bloom

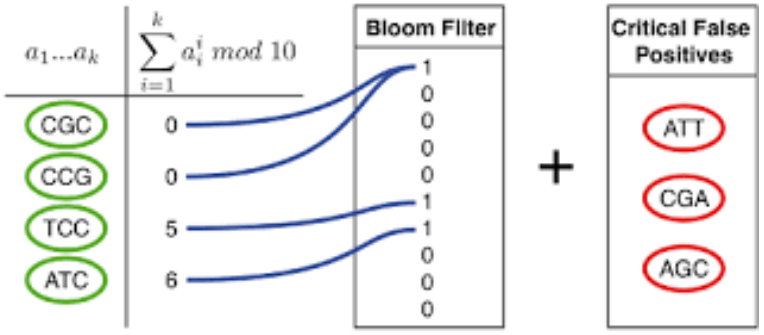
4 Explication du problème

4.1 Explication du problème traité

Le filtre de bloom est une structure de donnée permettant de voir rapidement si un élément appartient à un ensemble. C'est une structure probabiliste qui nous apporte deux informations:

- Elle permet de savoir avec certitude qu'un élément n'appartient pas
- Elle donne avec une certaine probabilité la présence d'un élément dans l'ensemble

On comprend ainsi qu'il existe de faux négatifs mais pas de faux positifs. Cette structure permet de faire gagner beaucoup de temps. En effet, elles permettent d'éviter des appels inutiles à de très grandes bases de données. Un exemple suivant est la recherche pour le séquençage génomique:



Comme vous vous en doutez, les séquences d'ADN sont très longues. Ainsi, le filtre de Bloom permet de voir si un codon, triplet de nucléotides, apparaît dans la séquence ou non.

5 Solution algorithmique

5.1 Structure

5.2 Fonctions crées

6 Étude de la complexité

Part IV
Makefile

7 Principe de fonctionnement

Les Makefile sont des fichiers utilisés par le programme make pour exécuter un ensemble d'actions.[3] Ils permettent entre autre la compilation de plusieurs programmes et leur mise à jour. Si j'ai crée un Makefile c'est dans un but précis. Dans un premier temps car lancer la commande **make** est facile et rapide et dans un second temps car je veux l'utiliser dans l'interface graphique.

```
CC=gcc
CCFLAGS= -Wall -g -std=c99
LIBFLAGS=
SRC=  heap.c
OBJ= $(SRC:.c=.o)
EXEC= heap

all: $(EXEC)

$(EXEC): $(OBJ)
        $(CC)  $^ -o $@  $(LIBFLAGS)

%.o: %.c
        $(CC) $(CCFLAGS) -o $@ -c $<

.depends:
        $(CC) -MM $(SRC) > .depends
-include .depends

clean:
        rm -f $(OBJ) $(EXEC) *.*~
```

8 Comment je l'utilise

L'interface et le Makefile laissent ainsi deux choix: soit on veut compiler les programmes et n'en lancer qu'un et dans ce cas là il faudra taper make puis ./nomdufichier, soit on veut lancer l'interface qui fait appel au make puis ne taper que sur des boutons et dans ce cas là on lance l'interface.

Part V
Interface graphique en Python

9 Principe d'automatisation

Etant donné que plusieurs programmes sont à lancer, j'ai décidé de créer une interface permettant de donner le choix à l'utilisateur.

Dans un premier temps, tous les fichiers sont compilés à l'aide de l'appel de fonction **os.system('make')**. Cette fonction permet d'exécuter la commande se trouvant entre les parenthèses dans le terminal.

Dans un second temps, l'utilisateur à plusieurs choix. Soit il peut choisir d'ouvrir ce rapport pour en savoir plus sur la documentation puis lancer un programme ou il peut décider directement quel programme il veut lancer. Dans le second cas, l'interface exécute directement le programme à lancer, plus besoin de taper de commande dans le terminal.

Pour pouvoir lancer l'interface, il faut avoir certaines bibliothèques python notamment os, numpy, sympy.

10 Algorithmique

Fonctions etc

Part VI
Organisation du travail

github

Part VII
Conclusion

Bibliographie

1. **Images**

   Logo Polytech UPMC. Available: http://www.polytech.upmc.fr

   Computer. Available:http://www.iconsdb.com/black-icons/notebook-icon.html

   Tas binomial exemple. Available:https://upload.wikimedia.org/wikipedia/commons/6/69/Min-heap.png

2. **Site Web**

   [2] **Algo7.** Available: http://www.lita.univ-lorraine.fr/ kratsch/teaching/algo7.pdf

   [3] Makefile. Available: http://gl.developpez.com/tutoriel/outil/makefile/

3. **Livres**

   [1] **Algorithmes et Structures de Données**. Tas et Arbre binaire de recherche. Cours 4. Available:http://australe.upmc.fr/access/content/group/a1b19d16-d5f8-4568-9044-afc69b61033d/Le%C3%A7on%204/heap_bst.pdf