Lecture Notes for Data Structures

Emulie Chhor

May 10, 2021

1 Overview of the field of Data Structures and Algorithms

One of the most important field of computer science is about Data Structures and algorithms, which is learn via 4 courses:

1. Discrete Maths
2. Data Structures
3. Algorithms
4. Theory of Computation
5. Modèle de Recherche Opérationnelle (optional)

2 Why study Data Structures

3 Introduction

The study of data structures can be divided in the following chapters:

1. Basics Data Structures: Dynamic Arrays and Linked List
2. Basics ADT: Bag, Stack and Queue
3. Sorting Algorithms
4. Heap
5. Treap (optional)
6. Binary Search Trees
7. Splay (optional)
8. Balanced Binary Trees
9. Hash Table
10. Graph Theory

4 Basic Data Structures

4.1 Overview

Le premier chapitre en structure des données introduit les notions de dynamic array et de linked list. En gros, ce sont des structures de données qui sont utilisées pour implémenter des structures abstraites.

Ces structures de données supportent certaines opérations

1. insertion
2. deletion
3. recherche

4.2 Dynamic Arrays

Le dynamic array est utilisé surtout pour storer des éléments dont on connait la taille et l'index parce qu'il requiert d'être resized

1. insertion: si la capacité est atteinte, on double la capacité et on copie les données dans le nouvel array
2. deletion: si la capacité est le quart de la taille du array, on copie les éléments dans un array qui est la moitié de la taille initiale
3. recherche: recherche linéaire si éléments non-ordonnées, recherche binaire si les éléments sont ordonnées

Complexité du resize: Preuve Crédit-Débit
On peut montrer que le resize prend un temps constant amorti par une preuve crédit-débit. Soit 1$, le cout pour push et pop. On sait qu'à chaque puissance de n éléments, on va devoir insérer n éléments et resize 1 fois. Ainsi, on va devoir:

1. ajouter les éléments une première fois (on n'a pas atteint la capacité): n fois
2. resize: coute 2n, car on doit pop n éléments et push n éléments dans le nouvel array
3. nombre d'éléments: n

Ainsi, le cout amorti est de $\frac{n+2n}{n} = 3$

4.3 Linked List

Il existe 3 types de linked list:

1. Singly Linked List: Peuvent seulement être traversée du head au tail, car on n'a pas de pointeur au précédant
2. Doubly Linked List: Can be traversed backward because we store the previous node, but takes twice the space
3. Circular Linked List: the head and tail can reach each other

Le linked list supporte les opérations suivantes:

1. insertion: On traverse la liste séquentiellement, et on ajoute le nouveau noeud entre le previous et le prochain en sauvegardant le pointeur du prochain dans une variable temporaire
2. deletion: On change le pointeur du previous et du prochain en sauvegardant le pointeur du prochain avant d'enlever le noeud
3. recherche: On doit faire de la recherche séquentielle parce qu'on n'a pas accès aux indices

4.4 Dynamic Arrays vs Linked List

5 Basic Abstract Data Types

5.1 Overview

Le bag, stack et queue sont les DS les plus basiques. On peut les implémenter avec des arrays, mais il est préférable de le faire avec des arraylist, puisqu'on ne connait pas leur taille d'avance (et on ne travaille qu'avec le head/tail)

5.2 Bag

Le bag ne supporte qu'une seule opérations: insertion. Pour la recherche, on ne peut faire qu'une recherche séquentielle, car les éléments n'ont pas d'ordre.

5.3 Stack

Le stack, communément appelé FILO (first-in, last-out), est un type abstrait qui est utilisé pour le cache et l'OS. Elle supporte les 3 opérations de base:

1. push (insertion): ajoute le nouvel élément au top
2. pop (suppression): si non vide, enleve l'élément eu top
3. recherche: pas fait pour ça, mais on peut peek

5.4 Queue

La queue, communément appelée FIFO (first-in, first-out), supporte les 3 opérations de base

1. enqueue (insertion): insère l'élément à la fin
2. dequeue (suppression): si non vide, enlève l'élément en tête de la liste
3. recherche: pas fait pour ça, mais on peut peek

6 Sorting Algorithms

6.1 Overview

Idéalement, le meilleur algorithme de tri devrait être linéarithmique et ne pas utiliser d'espace auxiliaire. Cependant, il n'existe pas de sorting algorithms satisfaisant ces 2 propriétés. Ainsi, on peut discerner quelques sorting algorithms:

1. Selection Sort
2. Insertion Sort
3. Bubble Sort
4. Mergesort
5. Quicksort
6. Heapsort
7. Raddix Sort

Notons qu'on apprend les sorting algorithms, non pas pour les implémenter, mais plutôt parce que leur structure et temps de complexité sont de bons introductions à l'analyse de la complexité. On veut donc être en mesure de connaitre le input qui va donner un temps optimal, temps moyen, temps pire temps.

3. Adjacency Matrix

11.3 Problems in Graph Theory