

“Mankind's progress is measured by the number of things we can do without thinking.”

- Whitehead

Data types & structures

There are numerous options for data structures for many commonly used abstract data types:

Containers
Dictionaries
Priority Queues

Changing data structures should not change the *correctness* of a program, but it can have a dramatic effect on the efficiency.

Choosing a Data Structure

It is important to choose the proper data structure when you first design an algorithm.

There are many data structures that can handle common operations: *insertion, deletion, sorting, searching, finding the maximum or minimum, predecessor or successor, etc.*

Different data structures will each take their own time for the different operations.

Guidelines...

- Building an algorithm around a properly chosen data structure leads to both a clean algorithm and good performance
- Using an incorrect data structure can be disastrous, but you don't always need the best structure.
- Sorting is at the heart of many good algorithms.
- Common algorithm design paradigms include *divide-and-conquer, randomization, incremental construction, and dynamic programming.*

Fundamental Data Types

An abstract data type is a collection of well-defined operations that can be performed on a particular structure.

Different data structures make different tradeoffs that make certain operations (say, insertion) faster at the cost of others (say, searching.) Often there will be other considerations that will make one structure more desirable over others.

Containers

Containers are abstract data types that hold stuff for later retrieval and not much more. There are three common interfaces to containers:

Stack: Supports last-in, first-out (LIFO).

Queue: Supports retrieval in first-in, first-out (FIFO) order.

Tables: Supports retrieval by position.

These are typically implemented using either arrays or linked lists.

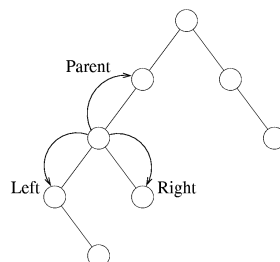
Dictionaries

Dictionaries are a form of container that permits access to data items by content. In addition to *insert* and *delete* operations, there is also a *search* operation to look up specific data within the container.

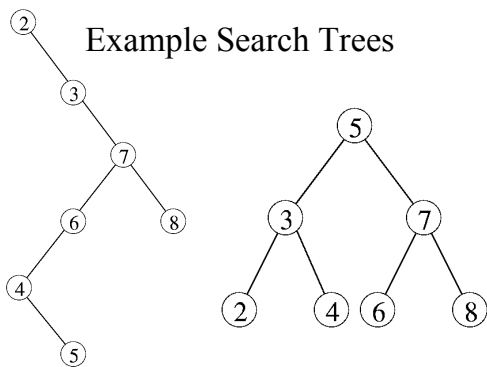
A linked list is a natural data structure for a dictionary, but has an $O(n)$ search time.

What are our other options?

Binary Trees



Example Search Trees



Successor and Predecessor

Successor: Find the minimal entry in the right sub-tree, if there is a right sub-tree. Otherwise find the first parent that the entry is in its left sub-tree.

Predecessor: Find the maximal entry in the left sub-tree, if there is a left sub-tree. Otherwise find the first parent that the entry is in its right sub-tree.

In either test, if the root node is reached, no predecessor/successor exists.

Simple Insertion and Deletion

Insertion: Traverse the tree as you would when searching. When the required branch does not exist, attach the new entry at that location.

Deletion: Three possible cases exist:

- a) *Entry is a leaf*: Just delete it.
- b) *Entry has one child*: Remove entry replacing it with child.
- c) *Entry had two children*: Replace entry with successor. Successor has at most one child; use step a or b on it.

Hash Tables

Hash tables are a *very practical* way to maintain a dictionary.

Looking up a function in an array is a $\Theta(1)$ operation once you have the index *if you used a good hash function*. A hash function maps dictionary search terms to array indices.

What would a good hash functions look like? A bad hash function?

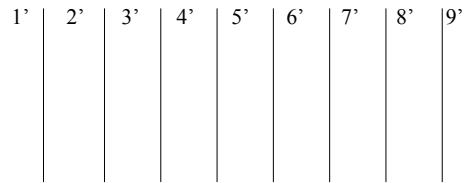
Hash Functions

A good hash function:

- Is cheap to evaluate.
- Tends to use all position 1..M with equal frequency.
- Puts similar keys into different parts of the table.

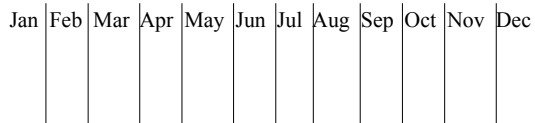
So, what functions should we avoid?

Hashing by Height



The Birthday Paradox

Even if we have a good function, we will still have collisions:



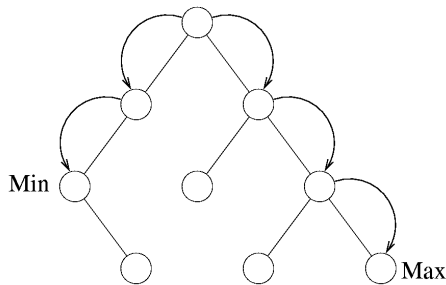
Priority Queues

Insert(x) : Given an item x , insert it into the priority Queue.

Find-Maximum() : Return the item with the maximal priority.

Delete-Maximum() : Remove the item from the queue whose key is maximum.

A Priority Queue Implementation



Specialized Data Structures

- Strings
- Geometric shapes
- Graphs
- Sets
- Schedules

Example Problems

a. You are given a pile of thousands of telephone bills and thousands of checks sent in to pay the bills. Find out who did not pay.

b. You are given a list containing the title, author, call number and publisher of all the books in a school library and another list of 30 publishers. Find out how many of the books in the library were published by each of those 30 companies.

c. You are given all the book checkout cards used in the campus library during the past year, each of which contains the name of the person who took out the book. Determine how many distinct people checked out at least one book.