Data Structures & Algorithms 2

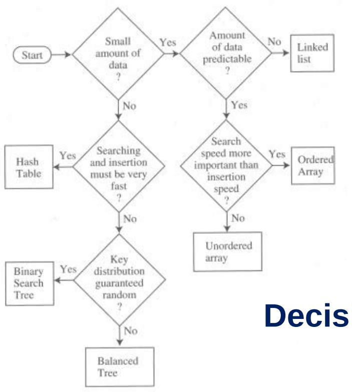
Topic 10 – Review

Data Structure & Algorithms

- We have come to the end of the data structures and algorithms course
- Now that we have learned about many different types it is important to identify which one to use
  - General-purpose data structures: arrays, linked lists, trees (balanced), hash tables
  - Specialized data structures: stacks, queues, priority queues, graphs (adjacency matrix or list)
  - Sorting: bubble, selection, insertion, mergesort, quicksort

General-purpose data structures

- These data structures are needed for storing real-world data such as personnel records, inventories, contact lists or sales data
- They allow you to access *any* of the information in them by searching for some key values (unlike stacks, queues)
- In order to select the correct structure we need to take into account the speed of operations such as inserting and searching as well as the nature of the data


Start
Small amount of data ?
Yes
Amount of data predictable ?
No
Linked list
No
Yes
Hash Table
Yes
Searching and insertion must be very fast ?
Search speed more important than insertion speed ?
Yes
Ordered Array
No
No
Unordered array
Binary Search Tree
Yes
Key distribution guaranteed random ?
No
Balanced Tree

Decision Chart

Speed and Complexity

- Arrays and linked lists are slow, trees are fairly fast and hash tables are very fast
- There is a penalty for using faster algorithms - they are more complex to program
- Hash tables require you to know in advance how much data can be stored and they don't use memory very efficiently
- Binary trees become unbalanced with non-random data and keeping them balanced is quite complicated

Processor Speed

- Every year there's an increase in the CPU and memory access speed of computers
- Moore's Law (1965) specifies that CPU performance doubles every 18 months
- This means that slow structures like arrays become more attractive
- If you can get away with using an array, why bother to create a Red-Black tree?
- Of course, Java comes with built in structures such as *Vector*, *Stack*, *PriorityQueue*, *RedBlackTree* and *Hashtable* classes if you want to avoid the hassle

Arrays

- Useful when
  - The amount of data is reasonably small
  - The amount of data is predictable in advance
  - Lots of insertion, not so much searching or deleting
- The *Vector* class supplied with Java uses arrays that expand themselves when they become too full
- Programs may periodically grind to a halt as the vector expands

Linked Lists

- Linked lists are useful when the amount of data to be stored cannot be predicted in advance
- Linked lists avoid wasting any memory and there are no holes to fill during deletion
- Insertion is fast, searching and deletion is slow (though faster than an array)
- Relatively simple to program

Binary Search Trees

- A binary tree is the first structure to consider when arrays and linked lists prove too slow
- Insertion, searching and deletion are all O(logn)
- Trees also have the bonus of being traversable
- The data inserted must be random

Balanced Trees

- Guarantees O(logn) time whether data is random or not
- Tricky to program (as you know and we didn't even attempt deletion!)
- If the size of the data is known then a hash table may be more appropriate

Hash Tables

- Hash Tables are the fastest data structure
- Typically used in spelling checkers and as symbol tables in computer language compilers when a program must check thousands of words in a fraction of a second
- They require additional memory, especially for open addressing
- The amount of data to be stored must be known in advance because an array is used
- Separate chaining is more robust if the exact amount of data is not known
- Traversal of values is impossible or access to the minimum of maximum values – use binary trees for this

Comparison of Data Structures

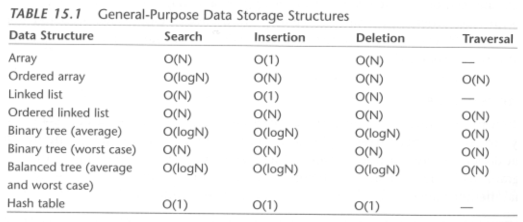


TABLE 15.1 General-Purpose Data Storage Structures

Data Structure	Search	Insertion	Deletion	Traversal
Array	O(N)	O(1)	O(N)	—
Ordered array	O(logN)	O(N)	O(N)	O(N)
Linked list	O(N)	O(1)	O(N)	—
Ordered linked list	O(N)	O(N)	O(N)	O(N)
Binary tree (average)	O(logN)	O(logN)	O(logN)	O(N)
Binary tree (worst case)	O(N)	O(N)	O(N)	O(N)
Balanced tree (average and worst case)	O(logN)	O(logN)	O(logN)	O(N)
Hash table	O(1)	O(1)	O(1)	—

Special Purpose Data Structures

- These include stacks, queues and priority queues
- These are usually used by a computer program to aid in carrying out some algorithm (e.g. depth first search)
- These are abstract data types (ADTs) that are implemented using a more fundamental structure such as an array or linked list
- These ADTs present a simple interface to the user, typically allowing the ability to access only one data item

Sorting

- As with the choice of data structures, it's worthwhile initially to try a slow but simple sort such as insertion sort (for fewer than 1,000 items)
- Insertion sort is best for almost-sorted files
- If it proves too slow then try the more complex sorts
- Mergesort requires extra memory
- Quicksort requires no extra memory and is slightly quicker so is the usual choice
- Be aware that small coding mistakes in quicksort may slow the algorithm and be very difficult to spot

Comparison of Sorting Algorithms

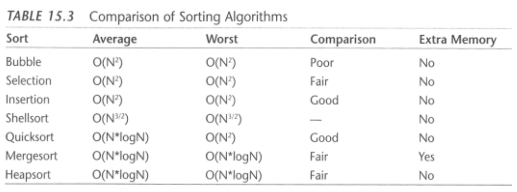


TABLE 15.3 Comparison of Sorting Algorithms

Sort	Average	Worst	Comparison	Extra Memory
Bubble	$O(N^2)$	$O(N^2)$	Poor	No
Selection	$O(N^2)$	$O(N^2)$	Fair	No
Insertion	$O(N^2)$	$O(N^2)$	Good	No
Shellsort	$O(N^{3/2})$	$O(N^{3/2})$	—	No
Quicksort	O(N*logN)	$O(N^2)$	Good	No
Mergesort	O(N*logN)	O(N*logN)	Fair	Yes
Heapsort	O(N*logN)	O(N*logN)	Fair	No

Graphs

- Graphs are a unique kind of data storage structure
- They don't store general-purpose data and they don't act as programmer's tools for other algorithms
- Instead, they directly model real-world situations
- When you need a graph, nothing else will do
- Your primary choice is whether to use an adjacency matrix or list to represent the graph
- If the graph is full use an adjacency matrix
- If the graph is sparse use an adjacency list

Real World Scenarios

- A program needs to be able to look-up records as fast as possible. The data in the database might grow slightly but will never get much bigger than it is now.
- Use hashing with separate chaining

- A program needs to look up records as fast as possible but the records are going to be updated constantly. Records are constantly being inserted, deleted and searched. They are often entered in a non-random order. The size of the database is unpredictable.
- Use a balanced binary tree –Red-Black is more efficient if there are going to be frequent deletions

Real World Scenarios

- A large number of records are backed up each night into a data structure which will only be consulted in the unlikely event that the building burns down. There are many insertions but no deleting or searching.
- Use a simple linked list, search time doesn't matter

- A company stores a record of all employees and each is indexed by a unique number which is incremented each time a new employee joins
- Use array if amount of data is small and unlikely to grow much

- You need to sort a large number of records which are partially sorted but have a few random ones thrown in
- Use insertion sort, if completely unsorted use quicksort

The Exam

- The exam is 2 hours long
- It follows the pen and paper parts of the labs, as well as asking you to explain the algorithms from the 9 topics
- There are 3 questions with no choice
- This means 40 minutes per question
- Each questions is worth 25 marks in total
- Exam is worth 70% and the CA is worth 30%

Practical questions

1. Add / delete / traverse nodes in a binary tree
2. Create Huffman codes for a short phrase
3. Quicksort a few numbers step by step
4. Add values into an AVL tree
5. Add values into a Red-Black tree
6. Obtain / send a message from an ElGamal ciphertext
7. Add values to a hash table using a particular strategy
8. Solve an unweighted / weighted graph problem
9. Follow a string searching algorithm step by step

Question 1

- Add values into a binary tree and show how the tree would be traversed
- Show how values are deleted and then delete nodes from a specific example
- Write a piece of code for traversal or manipulation of the binary tree
- Compare the efficiency of binary trees to other data structures
- Describe the concept of data compression
- Describe the Huffman algorithm and generate codes for a specific example

Question 2

- Define AVL-balanced and Red-Black balanced
- Describe the balancing algorithms
- Insert some values into an AVL / Red-Black tree
- Describe or code up the quicksort algorithm
- Show how a group of numbers are sorted by quicksort
- Describe how public key cryptography and the ElGamal system works
- Extract or send a message using a particular ciphertext
- Explain the discrete log problem

Question 3

- Explain the concept of hashing
- Insert some values into a hash table using different hashing strategies
- Demonstrate depth-first search and breadth-first search
- Draw a graph from an adjacency matrix or list representation
- Solve a weighted graph-based problem using the appropriate algorithm (i.e. Prim's algorithm, Dijskstra's algorithm)
- Explain the concepts P, EXP, NP, P=?NP, TSP, Knapsack
- Demonstrate Rabin-Karp or Knuth-Morris-Pratt string searching algorithms

Time-Keeping

- All questions have 4 parts
- Adapt the detail of your answer to the number of marks awarded for the question
- Guidelines
  - 10 marks = 15 minutes
  - 8 marks = 12 minutes
  - 6 marks = 10 minutes
  - 5 marks = 8 minutes
  - 4 marks = 6 minutes

- If you are asked for a diagram provide one, asked for examples, provide them
- All code must be written in Java