Advanced Data Structures And Algorithm Analysis

Adithya Nair

September 20, 2024

Contents

1 DSA Notes

1.1 Graphs

1.1.1 Representations Of A Graph

1. Adjacency Matrix Adjacency matrices are matrices which can be used to represent the connections that nodes have to each other. This is done by making a matrix of size n for n nodes. And the following rules exist for generating the matrix.

   (a) If there is a link from node i to node j then the element A_{ij} for the adjacency matrix A is 1.

   (b) If there is no link, then the element is zero.

   $$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix}$$

   As a consequence of this, undirected graphs(graphs where the links represent connection rather than direction) are represented with a symmetric matrix.

2. Adjacency Lists Adjacency lists are an alternative representation of graphs. If a graph has n nodes, then the list has n elements. The list adjList is then made with the following rules.

   adjlist[i] contains all the nodes which are connected to vertex i, in the order in which they are connected.

   for an example graph,

$$\begin{tikzcd}
a \arrow[r] \arrow[d] & b \arrow[r] & c \arrow[r] & d \\
b \arrow[r] \arrow[d] & a \arrow[r] & c \\
c \arrow[r] \arrow[d] & a \arrow[r] & b \\
d \arrow[r] \arrow[d] & a \arrow[r] & e \\
e \arrow[r] & d
\end{tikzcd}$$

we have a linked list of linked lists. the problem here is that links are represented twice.

this is an optimization for when graphs have very few links.

1.1.2 traversal(visit all nodes).

there are two algorithms of traversal:

1. breadth-first search - which involves the usage of queue
2. depth-first search - which involves the usage of stack

1. breadth-first search breadth-first involves selecting a root node, and choosing a neighbour node.

   revising queues, they have a first-in, first out approach, insertion at the rear and deletion at the front. the procedure follows:

   - the queue starts by inserting any root node, say for example a.
   - now a is dequeued and stored in the result.
   - queue all neighbouring nodes of a.
   - the first node is dequeued.
   - upon dequeueing, all neighbouring nodes are checked if it's in the queue. if not, they are queued in.
     - let's say b is taken, and b is connected to c, and c is already connected to a and b.

    - a and c will not be queued.
- this iterates until there are no more nodes.

clearly, this depends on the number of edges in the graph. the maximum number of edges in a graph is ${}^{n}c_2$.

the functions required in this queue are:

- ‘insert()‘
- ‘delete()‘
- ‘traverse()‘

2. depth-first search this involves traversing the nodes until an edge node is met, using a stack.

   the procedure follows:

   - a node is selected and pushed into the stack.
   - a random node connected to a is pushed.
   - a connected node is pushed.
   - this procedure is met until no unvisited node exists
     - in which case the stack is full.
     - if the stack is still not full, it implies that there are nodes which haven’t been visited.
   - now popping occurs
   - the algorithm goes back and checks for unvisited nodes by popping the stack and checking the topmost element’s adjacent nodes for unvisited nodes.
   - this is done until the stack is empty

```
class dfs {
  dfs(int v)  {
    adj = new linkedlist[v];
    visited = new boolean[v];
    for (int i = 0; i < v; i++)
      adj[i] = new linkedlist<integer>();
  }
}
```

1.2 trees

a tree with at most two links from one node is known as a binary tree.

1.2.1 properties

- maximum number of nodes - 2^i
- height is longest distance between root to leaf nodes (count the edges)
- maximum number of nodes possible till height h is $(2^0 + 2^1 + 2^2 + \cdots 2^h) = 2^{h+1} - 1$

1.2.2 types of binary trees

1. full/proper/strict binary tree it is a tree where each node must contain 2 children except the leaf node.
   - there are either two children or no children.
   - the number of leaf nodes is equal to number of internal nodes + 1
   - the minimum number of nodes is equal to $2^h + 1$
   - maximum number of nodes is the same as number of nodes in binary tree, $2^{h+1} - 1$
   - the minimum height of the full binary tree is $log_2(n + 1) - 1$
2. complete binary tree a binary tree where all the nodes are completely filled except the last level.
3. todo degenerate binary tree exactly like a linked list.
4. balanced binary tree
   - the heights of left and right trees can have a maximum height difference of 1.

1.2.3 binary search tree

value of left node must be smaller than parent, and value of right node must be greater than the parent node.

1. binary search tree insertion

2. binary search tree deletion

    - it works the same as a linked list operation.
    - we use in order traversal to figure out the node's value, it takes the value of its in order successor.

1.2.4 avl trees(height balanced binary tree)

what this solves is the fact that we have a series of insertions that are 'skinny'. an avl tree is balanced, the height is as minimal as possible.

each node is given a balancing factor, $= h_l - h_r$ a tree is said to be imbalanced if any node is said to have a balancing factor ≥ 2. there are 4 insertions that can be performed, also known as 'rotations':

- ll - $/$, ll rotation, involve shifting a node to the right, the central node in the line becomes a root node.
- rr - $\backslash$, rr rotation, involves shifting a node to the left, the central node in the line becomes a root node.
- lr - $<$, lr rotation, the last node in the subtree becomes the root node(it forms an rr + ll rotation)
- rl - $>$, rl rotation, the last node in the subtree becomes the root node(perform an ll + rr rotation)

the first letter is the sub-tree, and the second letter is the child node.
construct an avl tree by inserting 14,17,11,7,53,4,13,12,8,16,19,60,20
all 4 rotations are capable of causing an imbalance.

1.3 heap

heap is a data structure with a complete binary tree. this tree is mainly used for creating an efficient sorting algorithm.

1.3.1 Types Of Heap

there are two types of heap:

1. Min Heap the value of the root node is less than or equal to its children
2. Max Heap the value of the root node is more than or equal to its children

1.3.2 Definition

a heap is a special form of complete binary tree where the key value is lesser than or greater than its children. a heap is typically represented as an array, the array representation for a tree, is for each node a parent node is represented by $\frac{i}{2} - 1$, the left child node is $2 \times i + 1$ and the right child node is $2 \times i + 2$

heaps can be used for sorting, by deleting the root node until none are left, we get a sorted array.

1.3.3 Insertion

the process of insertion involves:

- adding a node to the leftmost child node available.
- comparing the node's value with the parent node
  - if current node's value is i, we have $\frac{(i-1)}{2}$
- if the comparison yields the result that the definition of heap does not hold, then the values of the parent node and the child node are swapped.
- repeat until the comparison holds.

```
// i is the variable holding the last position
// k is the value we are trying to add to the heap.
void insert(a,i,k){
i = i + 1;
a[i] = k;
while(i > 0){
    if(a[(i-1)/2] > a[i]){
        t = a[i];
        a[i] = a[(i-1)/2];
        a[(i-1)/2] = t;
    }
    else
        return;
    }
```

1.3.4 Deletion

- Deletion can only happen at the root node.
- This deletion takes place when you're trying to perform 'heap sort'.

For an array, [52,24,30,12,16,5]
We have 52 as the root node.

```
int k = a[0];
a[0] = a[n];
a[n] = k;
n = n-1;
```

1.4 Tries

How do you go about storing a dictionary? We construct a root node with 26 child nodes, one for each letter. Then each letter, sequentially forms the subsequent child node.

Trie is a sorted tree based data structure that stores a set of strings, it has n pointers, where n is the number of characters in the alphabet, in each node. It can search a word in the dictionary with the help of the character nodes preceding the end of the word. It searches incrementally by character.

1.4.1 Properties Of Trie

- the root node of the trie is empty with n pointers, it represents the 'full' node
- each child node is sorted alphabetically.
- each node can have a maximum of n children.

1.4.2 Applications

- Dictionary
- Address Book
- Phone Book
- Spell checker
- Browser history

1.5 Hashing

1.5.1 Hash Table

Hash tables are a 2-dimensional data structure of a fixed size. They allow you to have one probe, the data can be returned immediately

1.5.2 Hash Functions

1. Division method Given a set of data k, for a one-dimensional hash table of a size n, We store the elements, where u is the element in the hash and the hash function is $h(k)$ $i = h_k \% n$ $u_i = k$, we find the index through the modulus function and store it in the corresponding element within the array.

QUESTION: Use division method, to store the following data into a hash table of size $m = 10$, the has function is $h_k = (2k_i + 5)$

1. Folding method
2. Mid-square method
3. Modulo-multiplication method

1.5.3 Collision Avoidance Methods

For open hashing, chaining method is used for collision handling The chaining method involves creating a pointer to another node, and the conflict element is added to that node instead.

For closed hashing, Three methods to handle collision:

1. Linear probing When a collision occurs, insert the hash into the subsequent node $(u + i)\%n)$.
2. Quadratic probing When a collision occurs, insert the hash into the node $(u + i^2)\%10)$
3. Double hashing We use two hashing functions
   - $h_1(k) = u$
   - $h_2(k) = v$

When there is a collision, the second hashing function is used, in every other case the first hashing function is used.

Then the new index is calculated,

$$w = (u + v * i)\%m$$

Let's take an example, we define $h_1(k) = (2k + 1)\%10$, $h_2(k) = (3k + 2)\%10$

Key	U	V	W	Probe
6	3	-	-	1
12	5	-	-	1
18	7	-	-	1
10	1	-	-	1
2	5	-	9	4
18	-	-	-	-
22	-	-	-	-

Now we write the hash table,

index	**key**
0	-
1	10
2	-
3	6
4	-
5	12
6	-
7	8
8	-
9	2

The data array A is equal to 3,2,9,6,11,13,7,12, $n = 10$, $h_k = 2k+3$, $h2(k) = 3k+1$, plot the order of storage of the data in a hash table and total probing, and average probing, with respect to chaining, linear probing, quadratic probing, double hashing

Key	U	V	W	Probe
3	9	0		
2	7	7		
6	5	9		
9	1	8		
11	5	4		
13	9	0		
7	7	2		
12	7	7		

1.6 Merkel Trees

A hash tree is a data structure used for data verification and synchronization. It is a tree where each non-leaf node is a hash of its child node.

- The first layer is the list of all transacttion IDs in a block
- The data itself is not part the merkel tree, the first hashed values are the child nodes.
- These IDs are concatenated and hashed using SHA-256
- The second layer will be half the length of the first layer

1.6.1 Usage

The hash nodes are used to check if data is corrupted. If the root node of the hashing function differs from the merkel tree generated from the data, then we have an issue.