13. DFS, BFS, Dijkstra, Bellman-Ford

Thursday, June 2, 2022 9:04 PM

Quick Review of basic search

1) Linear search

arr = [3, 9, . . . , 15]

O(1) → O(n)

find x, x = 15

, iterate, if x, return T, else F

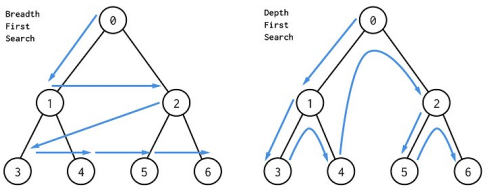
2) Binary search arr = [1, 3, . . . 15] log(n)

b/c it's a tree

- has to be sorted
- start at middle, find x
  - if x, true
  - if >x, go to middle of right
  - if <x, go to middle of left
  - repeat

Graph and Tree Traversal

- BFS and DFS
- O(n) for tree



Breadth First Search (BFS)

1) start at root

2) move left to right on first level, then left to right on second level, to n level until node match or tree ends

- ocd on close targets, finding shortest path, more memory needed
- uses queue to keep track of children you want to traverse to before jumping and a optional (for graphs) visited{} bool to keep track of visited nodes

Depth First Search (DFS)

1) start at root

2) follow one branch down to leaf, if not found then it continues at the nearest ancestor with unexplored children

- good on far targets, finding if path exists, less memory needed
- can get slow if tree is deep
- uses stack recursion

BFS Implementation

needs:

1) Declare Queue, visited[], currentNode(temp traversal)

2) push root

3) while (queue.size != 0)

{

currentNode = queue.poll() // removes or returns null if empty

add curr to visited

if (left child exists)

add to queue

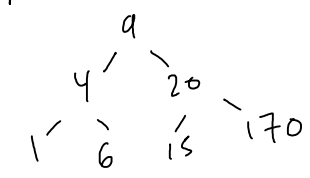
if (right child exists)

add to queue

```
public void breadthFirstSearch(){

    // Temp Node for traversal
    Node currentNode = root;

    // Optional, but mandatory for Graphs
    ArrayList<Integer> visited = new ArrayList<Integer>();


    Queue<Node> queue = new LinkedList<Node>();
    queue.add(currentNode);

    while (queue.size() != 0) {
      currentNode = queue.poll();
      visited.add(currentNode.data);

      if (currentNode.left != null) {
        queue.add(currentNode.left);
      }
      if (currentNode.right != null) {
```

```
            queue.add(currentNode.right);
          }

        }

        System.out.println("bfs: " + visited);
      }
```

DFS, 3 Types



Inorder (Left, root, right): [1, 4, 6, 9, 15, 20, 170]
- In a BST, it gives nodes in order

preorder (Root, Left, right): [9, 4, 1, 6, 20, 15, 170]
- Good for copying a tree

post order (Left, Right, root): [1, 6, 4, 15, 170, 20, 9]
- Good for deleting a tree

DFS Implemantation

Just move your operation around

tree height = deepest recursive function

memory = O(height)

```
public static void dfsInOrder(Node root) {
    if (root.left != null ){
      dfsInOrder(root.left);
    }
    System.out.print(root.data + " "); // inOrder
    if (root.right != null ){
      dfsInOrder(root.right);
    }
  }
```

```
public static void dfsPreOrder(Node root) {
    System.out.print(root.data + " "); // preOrder
    if (root.left != null ){
      dfsPreOrder(root.left);
    }
    if (root.right != null ){
      dfsPreOrder(root.right);
    }
  }
```

```
public static void dfsPostOrder(Node root) {
    if (root.left != null ){
      dfsPostOrder(root.left);
    }
    if (root.right != null ){
      dfsPostOrder(root.right);
    }
    System.out.print(root.data + " "); // postOrder
  }
```

Graph Traversal

https://visualgo.net/en/dfsbfs

- needs boolean visited[] to mark all vertices as not visited
  - default false
  - For BFS and DFS
  - O(V+E) for adjacency list

BFS : shortest path, closer nodes / cons: more memory

https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/

```
// prints BFS traversal from a given source s
  void BFS(int s)
  {
    // Mark all the vertices as not visited(By default
    // set as false)
    boolean visited[] = new boolean[V];

    // Create a queue for BFS
    LinkedList<Integer> queue = new LinkedList<Integer>();

    // Mark the current node as visited and enqueue it
    visited[s]=true;
    queue.add(s);

    while (queue.size() != 0)
    {
      // Dequeue a vertex from queue and print it
      s = queue.poll();
      System.out.print(s+" ");

      // Get all adjacent vertices of the dequeued vertex s
      // If a adjacent has not been visited, then mark it
      // visited and enqueue it
      Iterator<Integer> i = adj[s].listIterator();
      while (i.hasNext())
      {
        int n = i.next();
        if (!visited[n])
        {
          visited[n] = true;
          queue.add(n);
        }
      }
    }
  }
```

cons

Pros

- ... more backtracking after dead end | can get slow if

DFS

DFS : Solving a maze, backtracking after dead end
Less memory
Does Path Exist ?

can get slow if deep graph

```
void DFSUtil(int v, boolean visited[])
 {
    // Mark the current node as visited and print it
    visited[v] = true;
    System.out.print(v + " ");

    // Recur for all the vertices adjacent to this
    // vertex
    Iterator<Integer> i = adj[v].listIterator();
    while (i.hasNext()) {
       int n = i.next();
       if (!visited[n])
          DFSUtil(n, visited);
    }
 }

 // The function to do DFS traversal. It uses recursive
 // DFSUtil()
 void DFS()
 {
    // Mark all the vertices as not visited(set as
    // false by default in java)
    boolean visited[] = new boolean[V];

    // Call the recursive helper function to print DFS
    // traversal starting from all vertices one by one
    for (int i = 0; i < V; ++i)
       if (visited[i] == false)
          DFSUtil(i, visited);
 }
```

https://www.geeksforgeeks.org/depth-first-search-or-dfs-for-a-graph/

for adjacency matrix

this are factors in unconnected graphs

BFS can also do the same

Weighted Graphs (Shortest Path for Weighted graph)

https://www.geeksforgeeks.org/what-are-the-differences-between-bellman-fords-and-dijkstras-algorithms/

Dijkstra

https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/

- Greedy algorithm
- Can only handle positive weights
- O - (V + E . log(V)) , more efficient than Bellman

Bellman Ford

https://www.geeksforgeeks.org/bellman-ford-algorithm-dp-23/

- Dynamic Programming
- handles negative and positive weight

- handles negative and positive
- O(V.E), longer than Dijkstra