CCC Sprint Preparation Class

Geyang Liu

Agenda

Fundamental Data Structure

Basic Algorithms

Advanced Algorithms

CCC Sample Questions 3

CCC Sample Questions 4

Fundamental Data Structure

- Array
- List/LinkedList
- Stack
- Queue
- Hash Table (Map/ Set)
- Tree
- Heap/Priority Queue
- Graph

Basic Algorithms

- **Fundamentals (Analysis, Complexity Measures Big(O))**
- Sorting

  Quick Sort

  Merge Sort

  Heap Sort

  ….
- **Search**

  Hashing

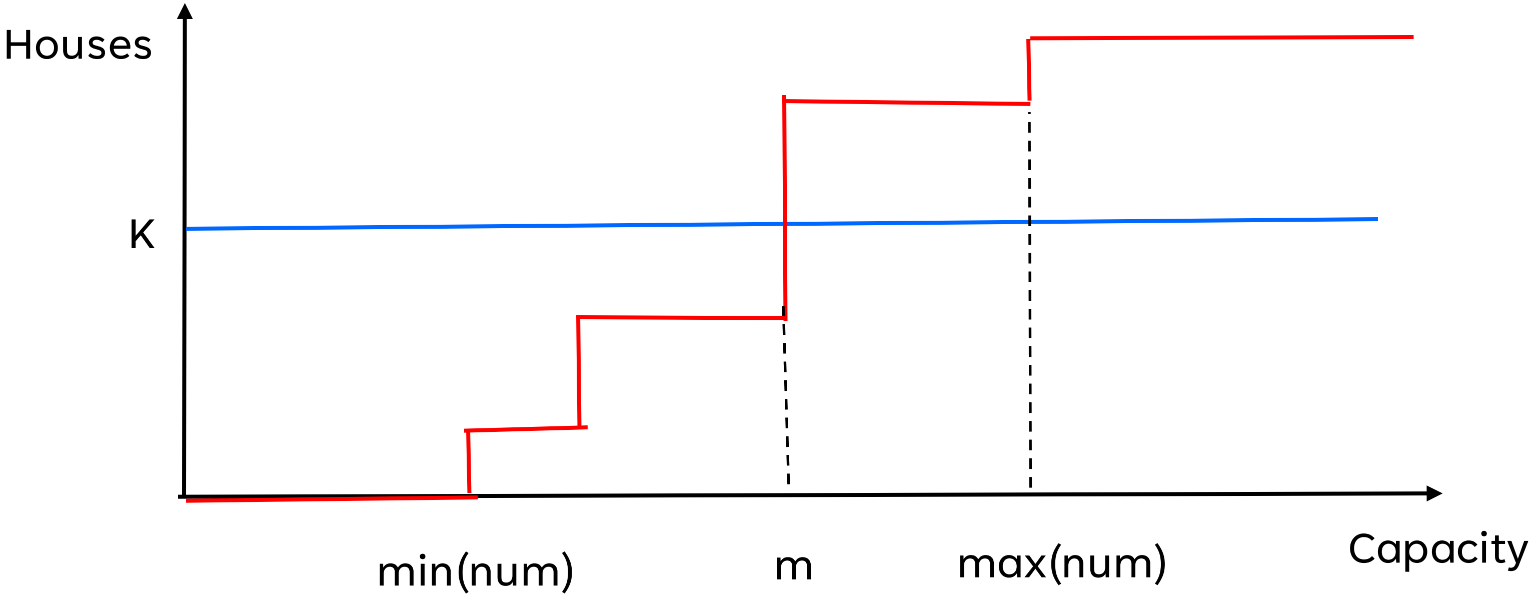
  Search trees

Binary Search

LeetCode 2560. House Robber IV https://leetcode.com/problems/house-robber-iv/

1. # of houses that can rob monotonically increases with capacity.

2. Use binary search to find the minimum m so that K <= # of houses.



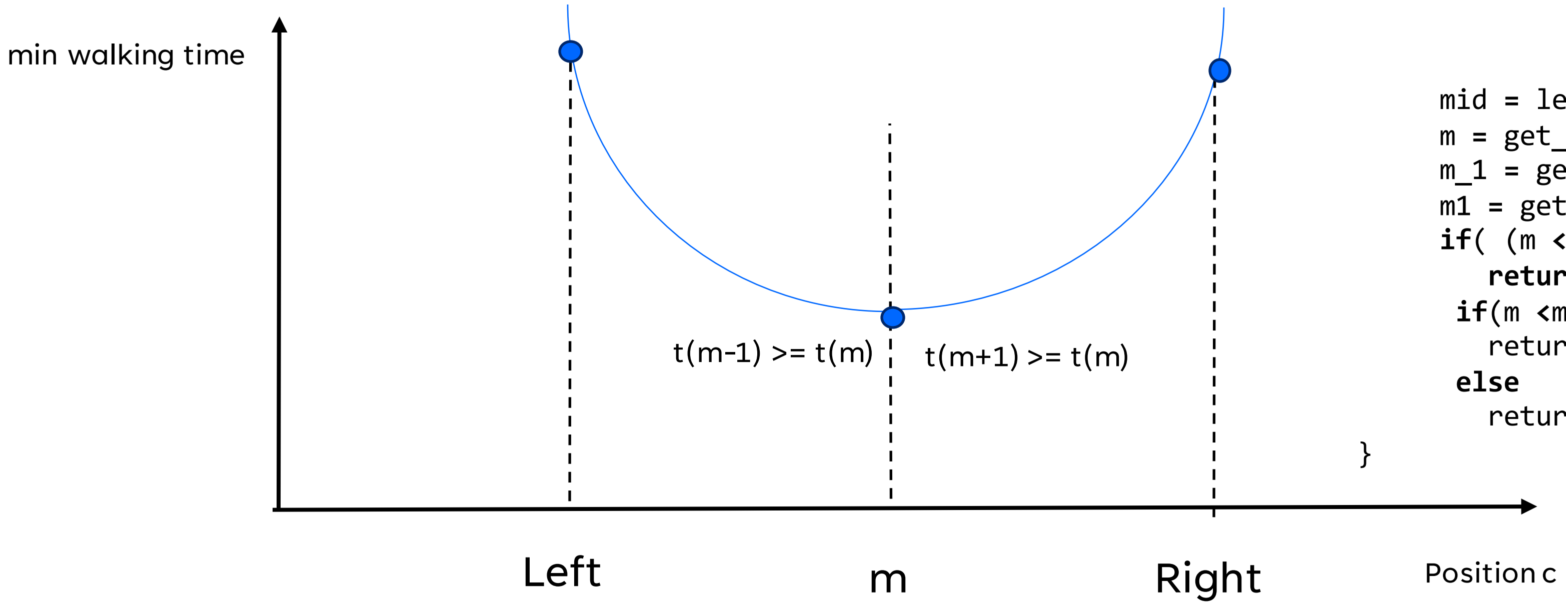
```
class Solution {
    //use binary search to enumerate the stealing ability of the thief.
   public int minCapability(int[] nums, int k) {
     int left = 0, right = (int) 1e9;
     while (left < right) {
       int mid = (left + right) >> 1;
       if (rob(nums, mid) >= k)  right = mid;
       else  left = mid + 1;
     }
     return left;
   }
   // use a greedy approach to determine whether the thief can steal at least x houses.
  private int rob(int[] nums, int x) {
        int cnt = 0, j = -2;
        for (int i = 0; i < nums.length; ++i) {
            if (nums[i] > x || i == j + 1) {
                continue;
            }
            ++cnt;
            j = i;
        }
        return cnt;
    }
}
```

Binary Search

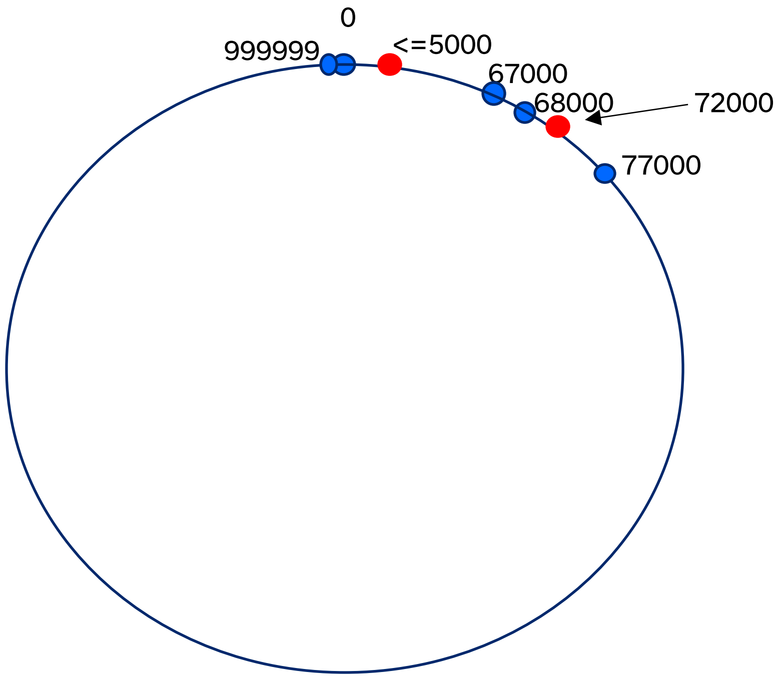
CCC '21 S3 - Lunch Concert

https://dmoj.ca/problem/ccc21s3

```
get_total_time( position, P, W D)
binary_search(left, right, P, W, D){
  if (left== right)
     return get_total_time(left, P, W,D)
  If(right-left == 1)
     return min(get_total_time(right, P, W,D), get_total_time(left, P, W,D)
  if(right -left == 2)
     return min(get_total_time(right,P,W,D),
                get_total_time(left,P,W,D));
                getSum(left + 1,P,W,D));

    mid = left + (right-left)/2;
    m = get_total_time(mid,P,W,D);
    m_1 = get_total_time(mid -1,P,W,D);  // m-1
    m1 = get_total_time(mid + 1,P,W,D);  // m+1
    if( (m <= m_1) && (m<=m1))// t(m)<= t(m-1) && t(m)<=t(m_+1)
       return m;
     if(m <m_1)
       return binary_search(mid, right, P,W,D);
     else
       return binary_search(left, mid, P,W,D);
}
```



Binary Search



Houses H=4 : 0, 67000, 68000, 77000

Hose Count K=2

Minimum length of hose 5000

```
binary_search(H, K){
  houses = new int[H*2];
  // scan H houses to house array
  sort(houses);
  for (i = 0; i < H; i++)
     houses[i + H] = houses[i] + 1M;
  lo = 0; hi = 1M;
  while (lo < hi) {
    int mid = lo + (hi - lo) / 2;
    if (getMinHoseCount(houses,H, mid) > K) lo = mid + 1;
    else hi = mid;
  }
  print(lo);
}
// return minimum hose count needed
int getMinHoseCount(houses, H,len) {
    min = MAX_VALUE;
    for (i = 0; i < H; i++) {
      cur = i;h = 1;
      for (j = i; j < H + i; j++)
        if (houses[j] - houses[cur] > 2 * len) {cur = j;
                                                h++;}
      min = Math.min(min, h);
    }
    return min;
            }
```

Graph Problems

- Adjacency Matrix and Adjacency List
- BFS & DFS in Graph
- Cycles in Graph
- Shortest Paths in Graph
- Minimum Spanning Tree
- Topological Sorting

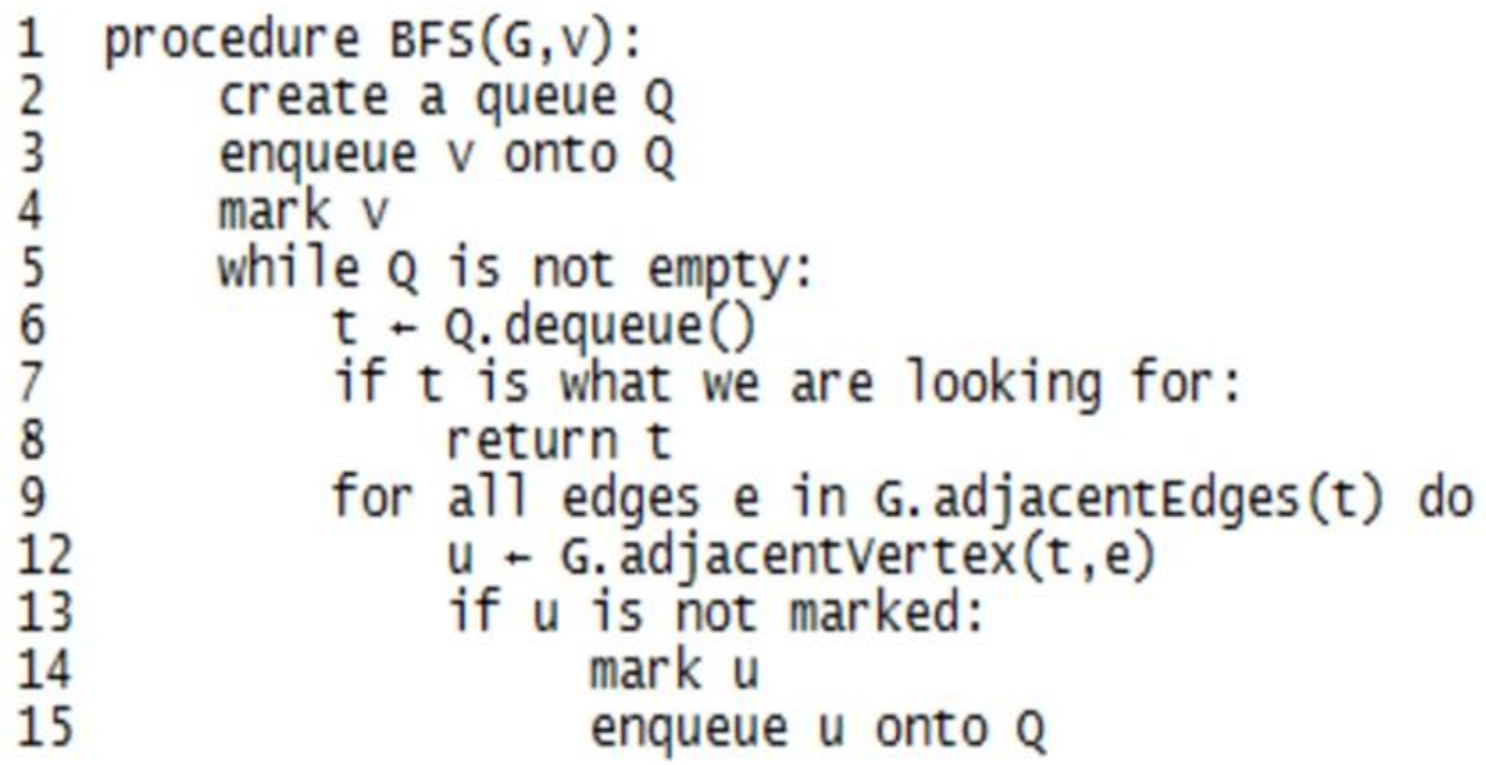
BFS Breadth-First Search

- Start at the root of the graph and visits all nodes at the current depth level before moving on to the nodes at the next depth level.
- 1. start at node
- 2. Visit all the nodes' neighbors
- 3. Visit all the neighbors' neighbors
- 4. continues in this fashion.
- https://www.cs.usfca.edu/~galles/visualization/BFS.html
- https://github.com/rayliu7717/CCC_CLASS/blob/main/CCC_GRAPH_BFS.java

BFS Problems

- Shortest Path and Minimum Spanning Tree
- Level Order traverse tree
- Cycle detection in graph
- Path Finding
- Finding all nodes within one connected component
- Connected Component
- Topological sorting
- .....

BFS pseudo code



```
procedure BFS(G,v):
    create a queue Q
    enqueue v onto Q
    mark v
    while Q is not empty:
        t ← Q.dequeue()
        if t is what we are looking for:
            return t
        for all edges e in G.adjacentEdges(t) do
            u ← G.adjacentVertex(t,e)
            if u is not marked:
                mark u
                enqueue u onto Q
```

BFS Sample Problems

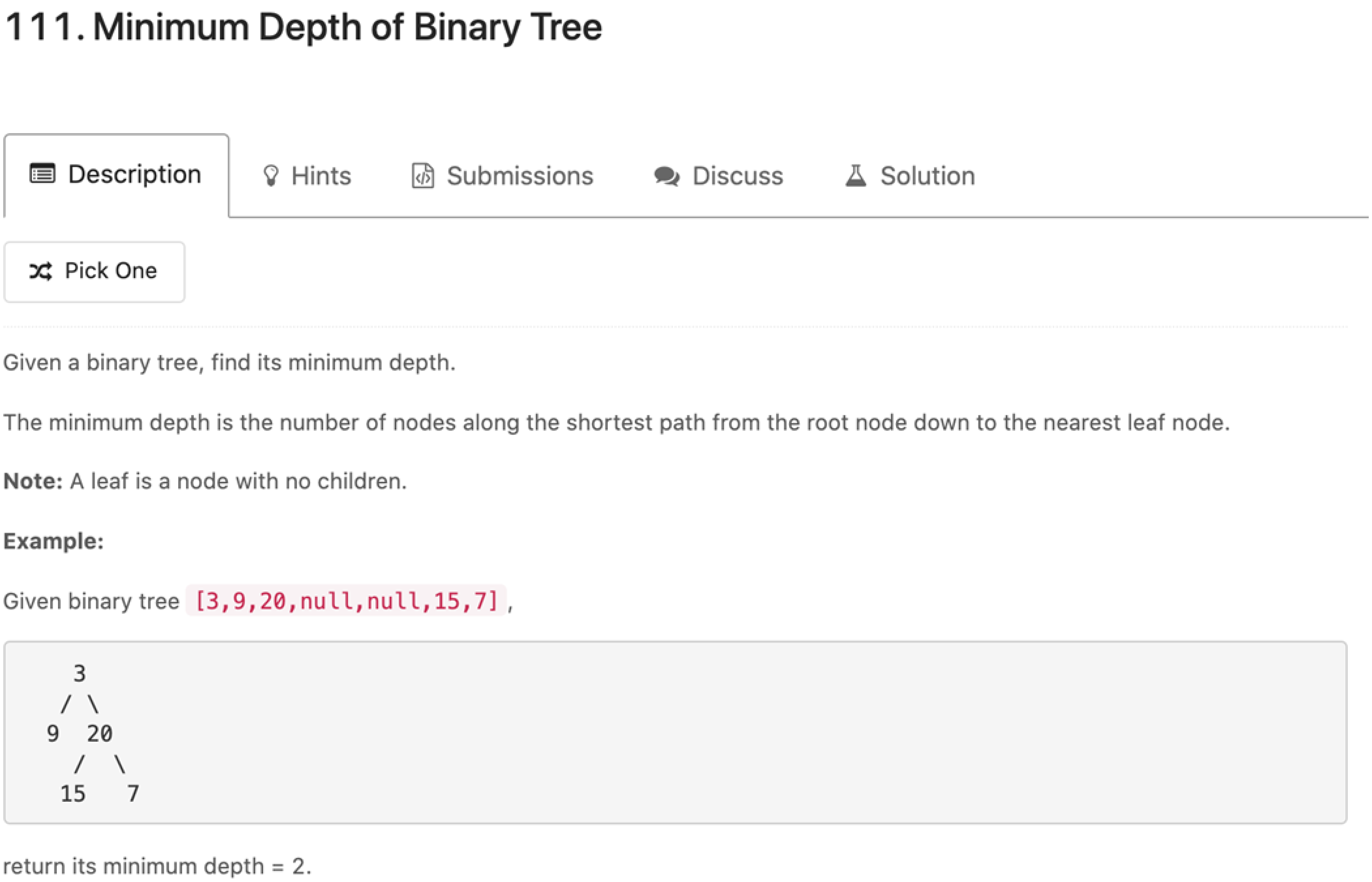
- Leetcode 111. Minimum Depth of Binary Tree
- Leetcode 102. Binary Tree Level Order Traversal
- Leetcode 127, Word Ladder

  https://leetcode.com/problems/word-ladder/submissions/

- Leetcode 207, Course Schedule

  https://leetcode.com/problems/course-schedule/

  https://www.cs.usfca.edu/~galles/visualization/TopoSortIndegree.html



111. Minimum Depth of Binary Tree

Given a binary tree, find its minimum depth.

The minimum depth is the number of nodes along the shortest path from the root node down to the nearest leaf node.

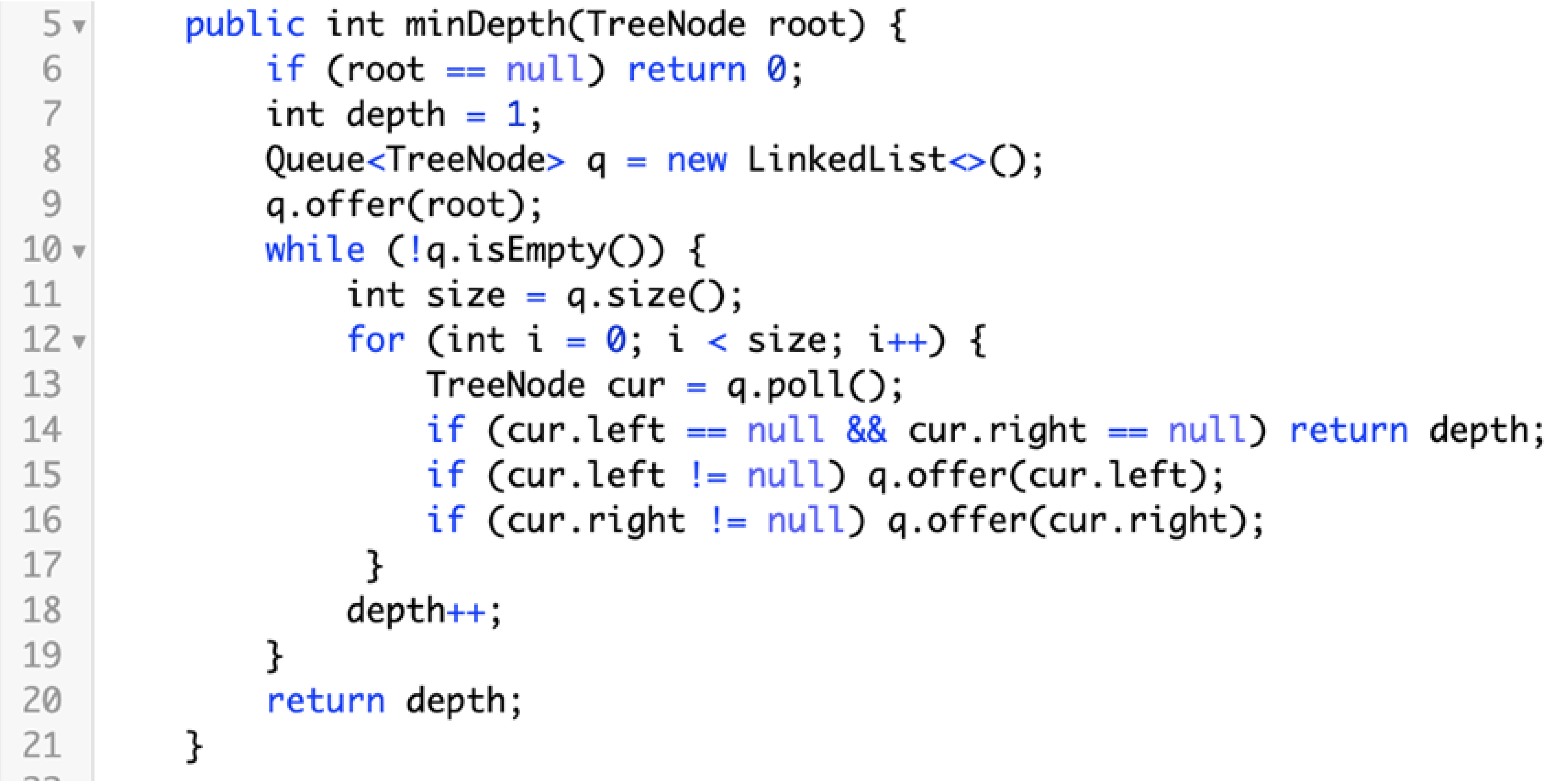
Note: A leaf is a node with no children.

Example:

Given binary tree `[3,9,20,null,null,15,7]`,

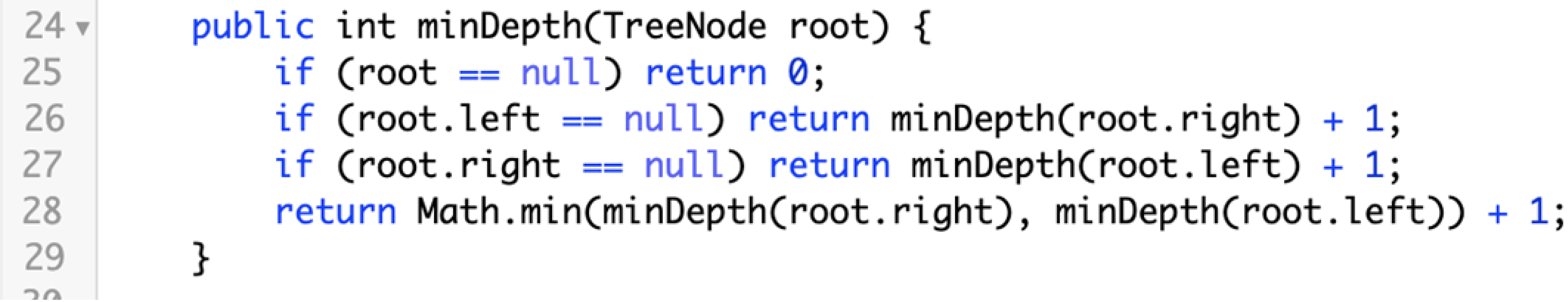
```
    3
   / \
  9  20
    /  \
   15   7
```

return its minimum depth = 2.

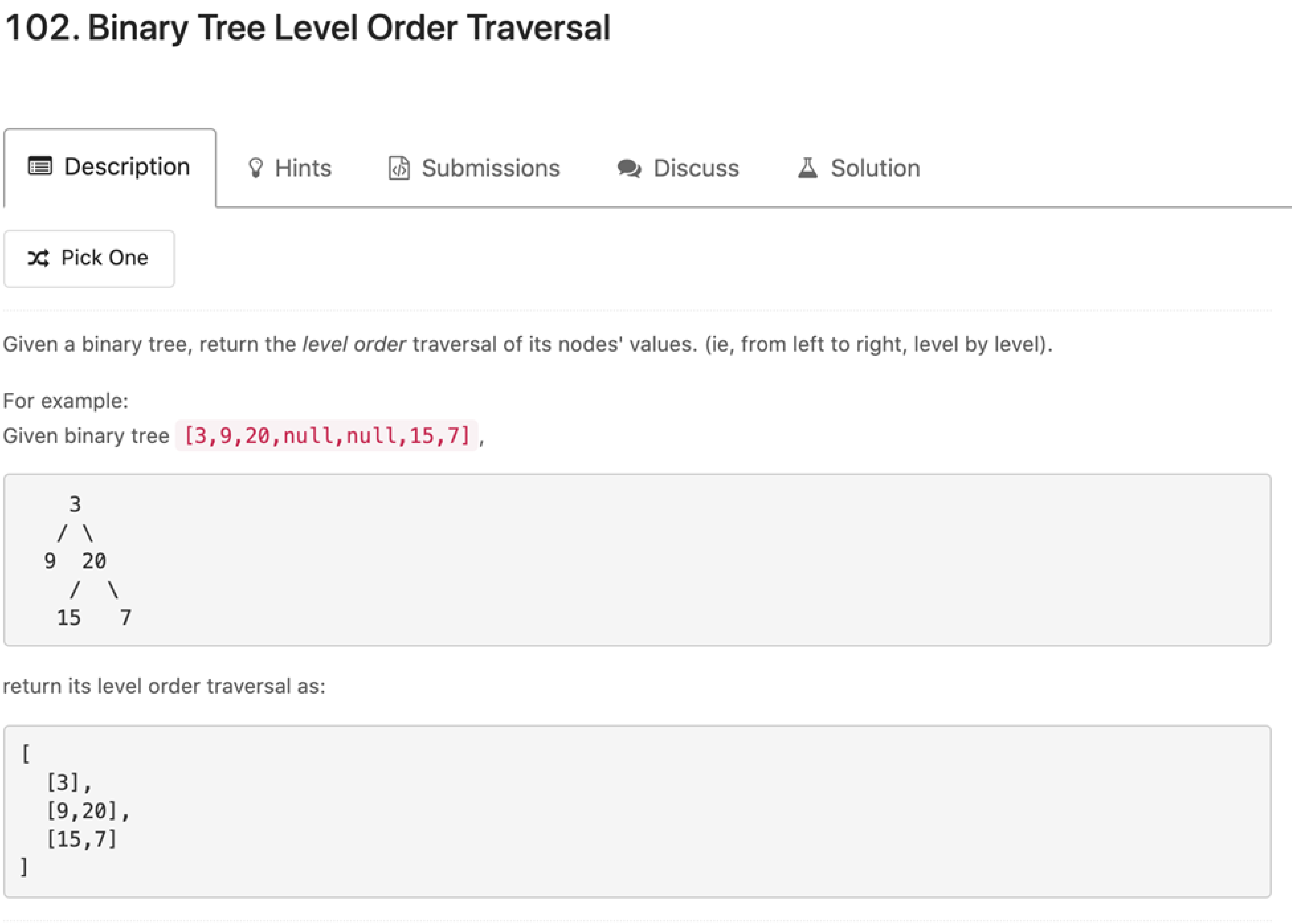


```java
    public int minDepth(TreeNode root) {
        if (root == null) return 0;
        int depth = 1;
        Queue<TreeNode> q = new LinkedList<>();
        q.offer(root);
        while (!q.isEmpty()) {
            int size = q.size();
            for (int i = 0; i < size; i++) {
                TreeNode cur = q.poll();
                if (cur.left == null && cur.right == null) return depth;
                if (cur.left != null) q.offer(cur.left);
                if (cur.right != null) q.offer(cur.right);
             }
            depth++;
        }
        return depth;
    }
```

Search from root level by level until get the first leaf



```java
    public int minDepth(TreeNode root) {
        if (root == null) return 0;
        if (root.left == null) return minDepth(root.right) + 1;
        if (root.right == null) return minDepth(root.left) + 1;
        return Math.min(minDepth(root.right), minDepth(root.left)) + 1;
    }
```



102. Binary Tree Level Order Traversal

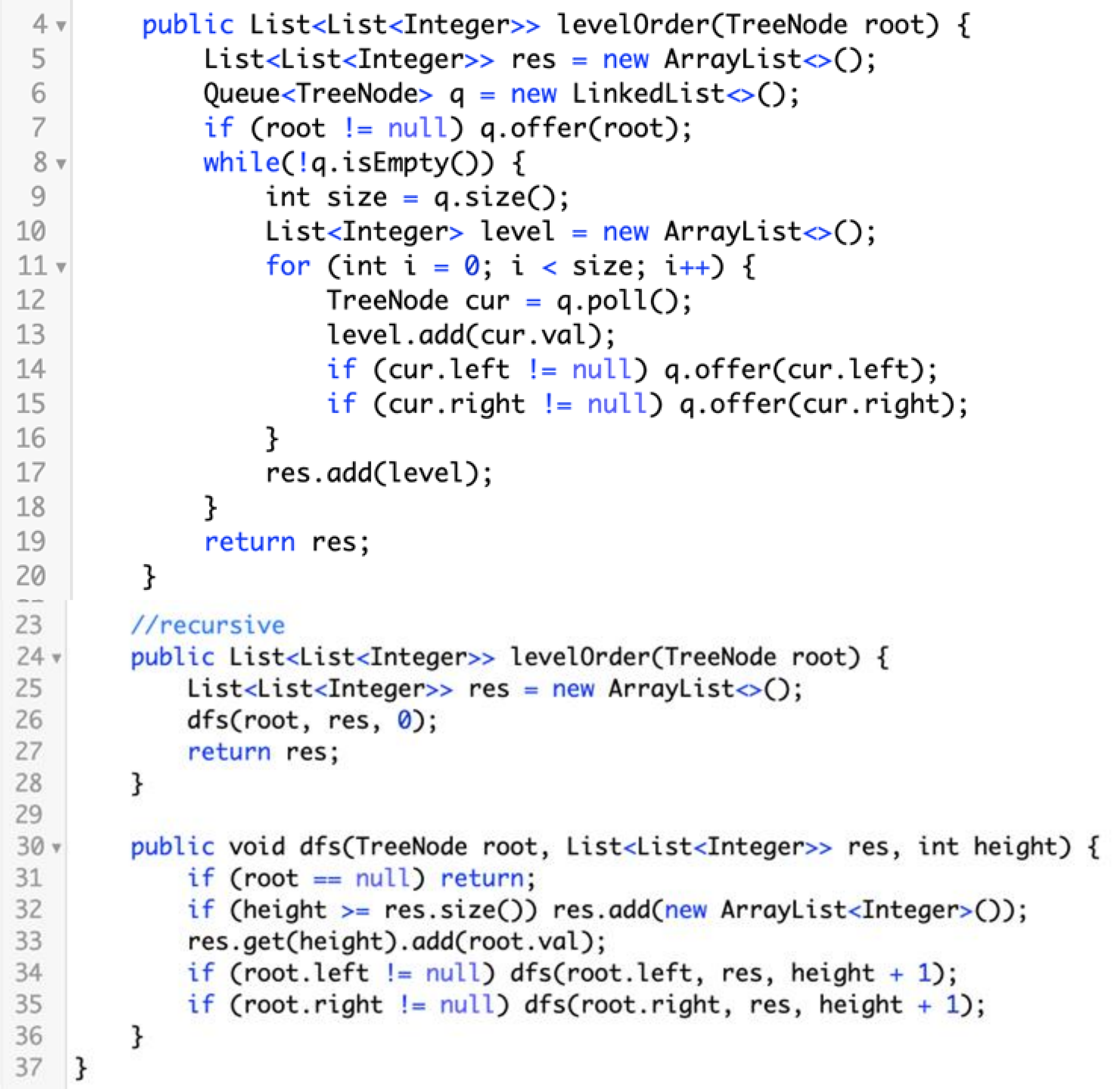
Given a binary tree, return the *level order* traversal of its nodes' values. (ie, from left to right, level by level).

For example:
Given binary tree `[3,9,20,null,null,15,7]`,

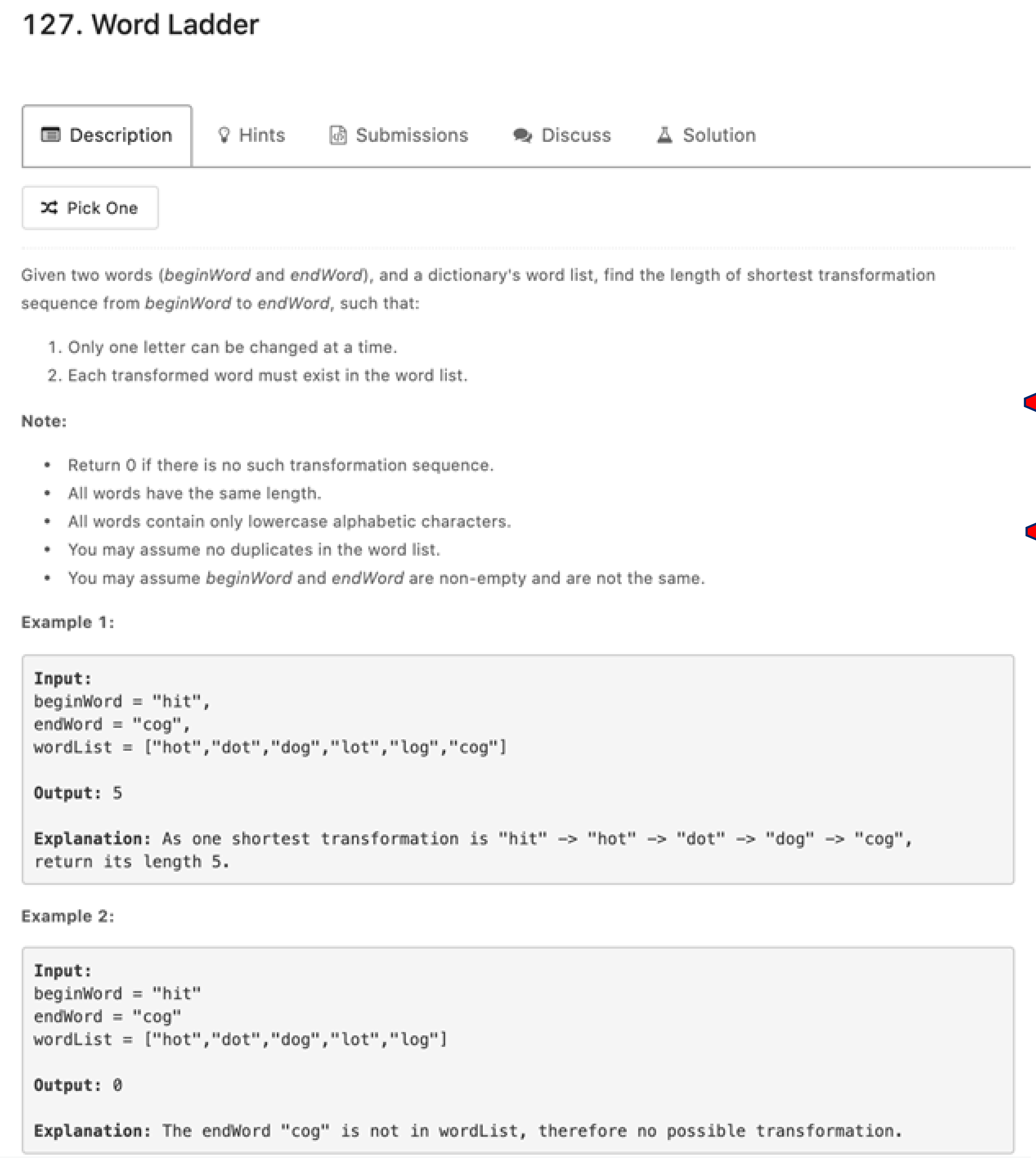
```
    3
   / \
  9  20
    /  \
   15   7
```

return its level order traversal as:

```
[
  [3],
  [9,20],
  [15,7]
]
```



```java
    public List<List<Integer>> levelOrder(TreeNode root) {
        List<List<Integer>> res = new ArrayList<>();
        Queue<TreeNode> q = new LinkedList<>();
        if (root != null) q.offer(root);
        while(!q.isEmpty()) {
            int size = q.size();
            List<Integer> level = new ArrayList<>();
            for (int i = 0; i < size; i++) {
                TreeNode cur = q.poll();
                level.add(cur.val);
                if (cur.left != null) q.offer(cur.left);
                if (cur.right != null) q.offer(cur.right);
            }
            res.add(level);
        }
        return res;
    }
    //recursive
    public List<List<Integer>> levelOrder(TreeNode root) {
        List<List<Integer>> res = new ArrayList<>();
        dfs(root, res, 0);
        return res;
    }

    public void dfs(TreeNode root, List<List<Integer>> res, int height) {
        if (root == null) return;
        if (height >= res.size()) res.add(new ArrayList<Integer>());
        res.get(height).add(root.val);
        if (root.left != null) dfs(root.left, res, height + 1);
        if (root.right != null) dfs(root.right, res, height + 1);
    }
}
```



127. Word Ladder

Given two words (*beginWord* and *endWord*), and a dictionary's word list, find the length of shortest transformation sequence from *beginWord* to *endWord*, such that:

1. Only one letter can be changed at a time.
2. Each transformed word must exist in the word list.

Note:

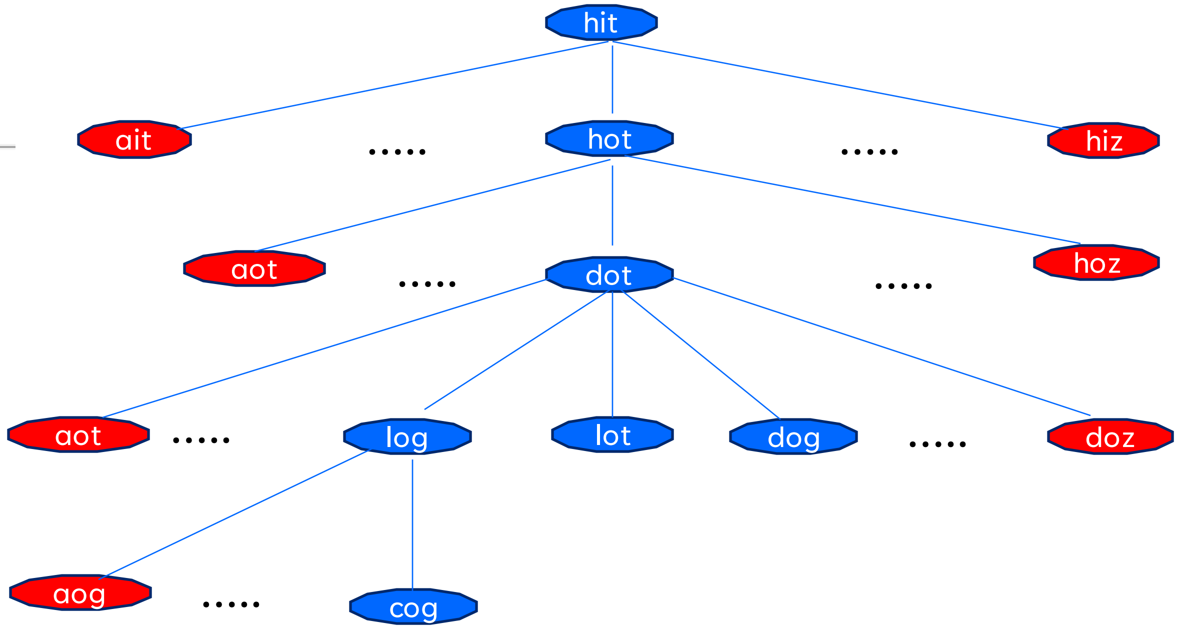
- Return 0 if there is no such transformation sequence.
- All words have the same length.
- All words contain only lowercase alphabetic characters.
- You may assume no duplicates in the word list.
- You may assume *beginWord* and *endWord* are non-empty and are not the same.

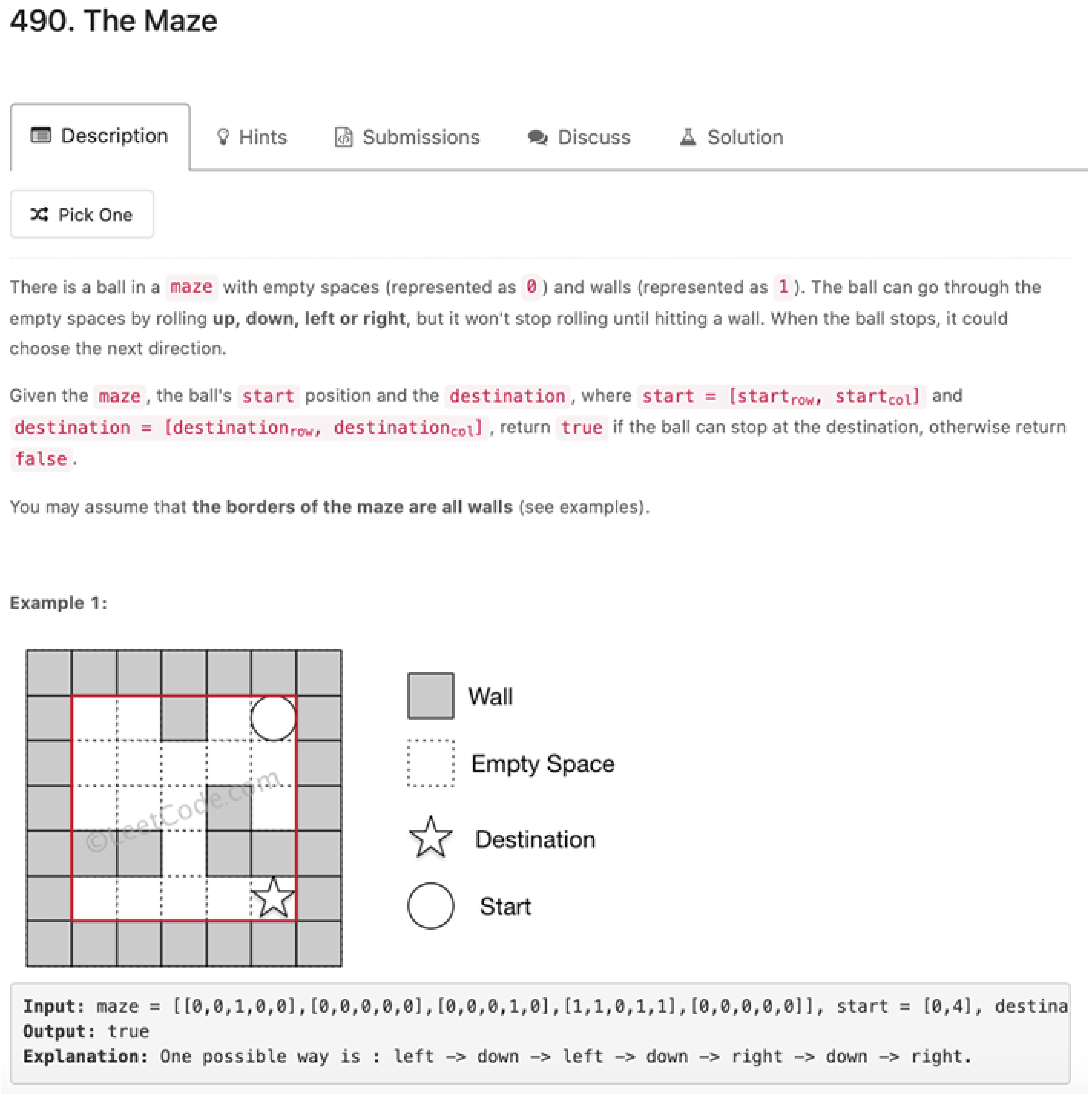
Example 1:

```
Input:
beginWord = "hit",
endWord = "cog",
wordList = ["hot","dot","dog","lot","log","cog"]

Output: 5

Explanation: As one shortest transformation is "hit" -> "hot" -> "dot" -> "dog" -> "cog",
return its length 5.
```

Example 2:

```
Input:
beginWord = "hit"
endWord = "cog"
wordList = ["hot","dot","dog","lot","log"]

Output: 0

Explanation: The endWord "cog" is not in wordList, therefore no possible transformation.
```





490. The Maze

There is a ball in a `maze` with empty spaces (represented as `0`) and walls (represented as `1`). The ball can go through the empty spaces by rolling **up, down, left or right**, but it won't stop rolling until hitting a wall. When the ball stops, it could choose the next direction.

Given the `maze`, the ball's `start` position and the `destination`, where `start = [start`$_{row}$`, start`$_{col}$`]` and `destination = [destination`$_{row}$`, destination`$_{col}$`]`, return `true` if the ball can stop at the destination, otherwise return `false`.

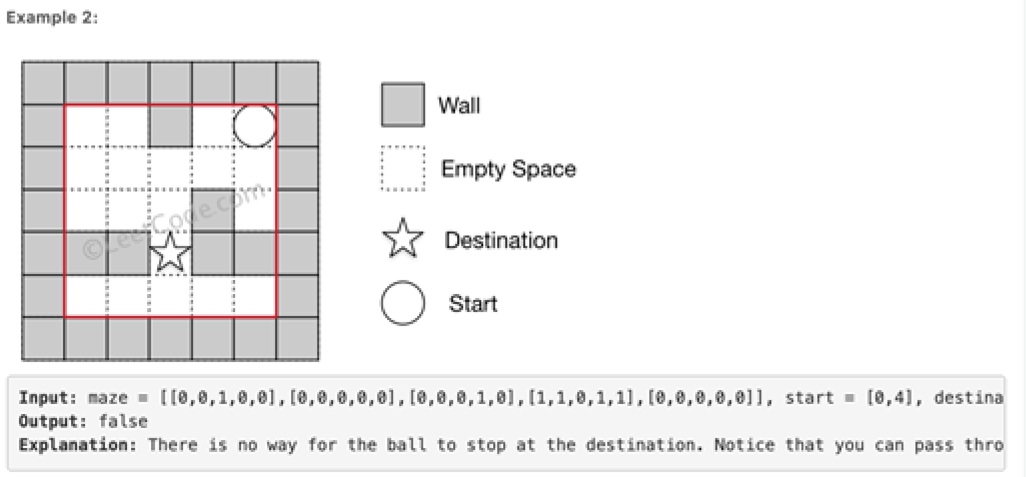
You may assume that **the borders of the maze are all walls** (see examples).

Example 1:

```
Input: maze = [[0,0,1,0,0],[0,0,0,0,0],[0,0,0,1,0],[1,1,0,1,1],[0,0,0,0,0]], start = [0,4], destina
Output: true
Explanation: One possible way is : left -> down -> left -> down -> right -> down -> right.
```

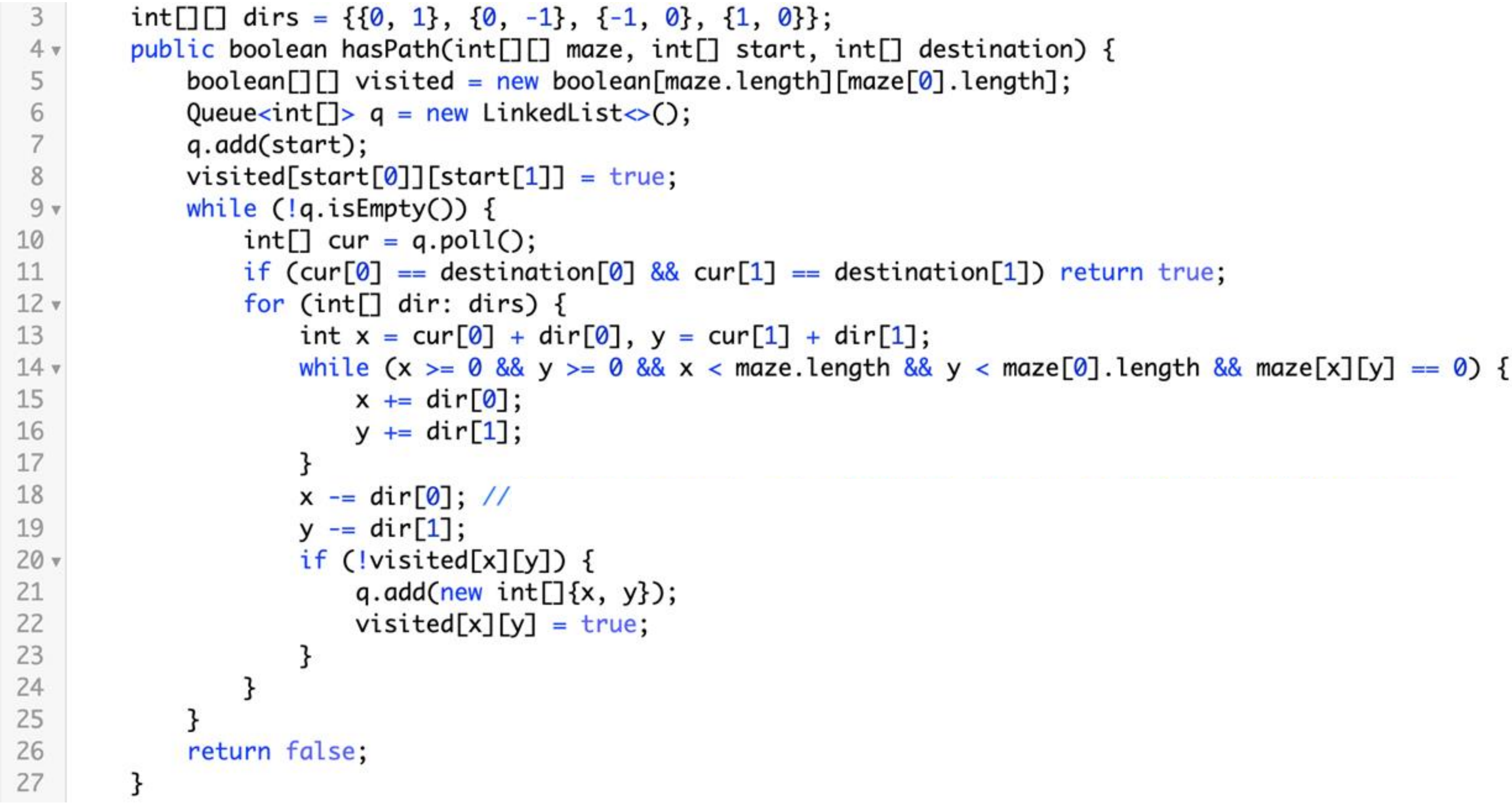
2D matrix BFS Travers Template

Use dirs array



Example 2:

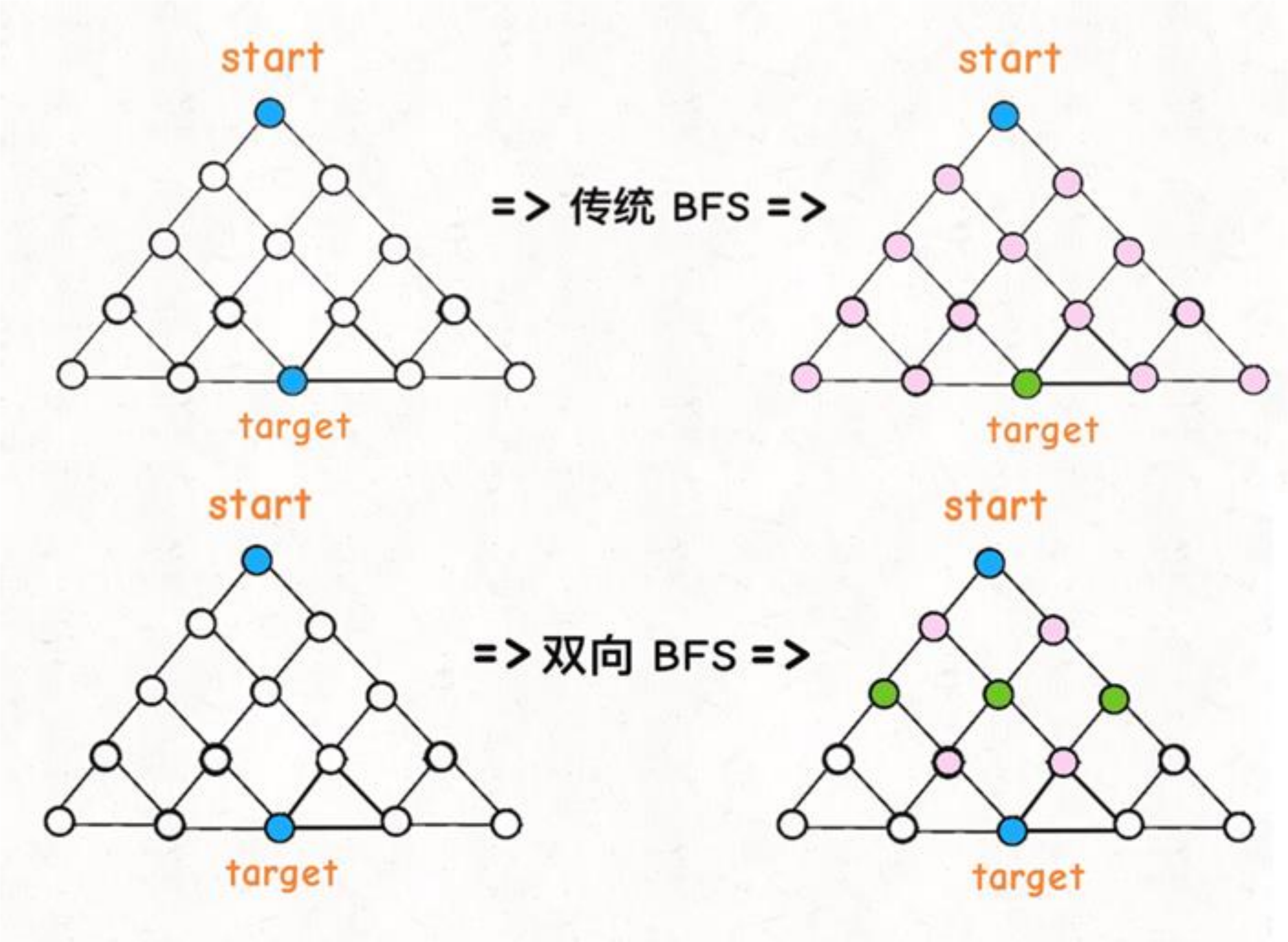
```
Input: maze = [[0,0,1,0,0],[0,0,0,0,0],[0,0,0,1,0],[1,1,0,1,1],[0,0,0,0,0]], start = [0,4], destina
Output: false
Explanation: There is no way for the ball to stop at the destination. Notice that you can pass thro
```



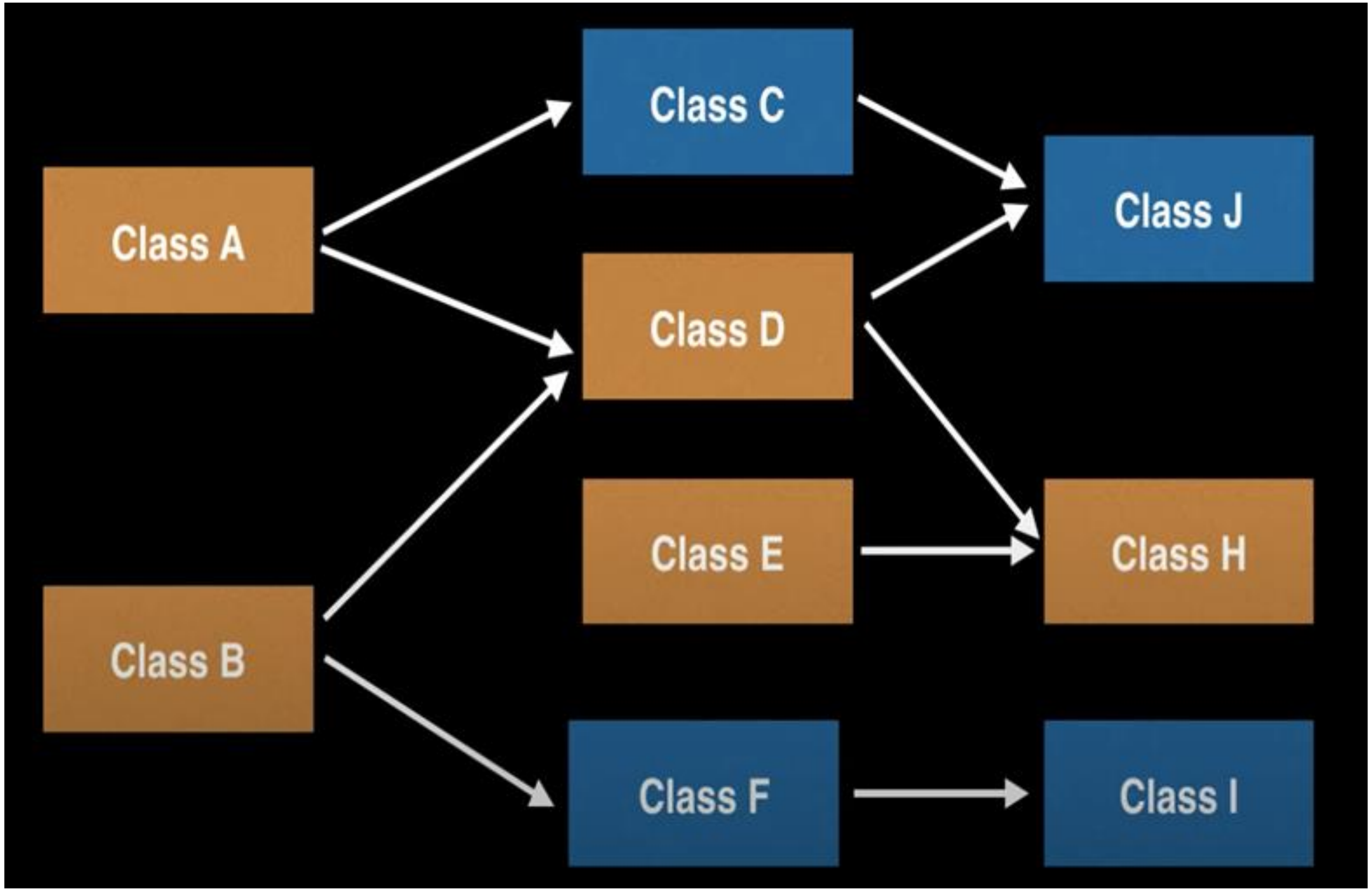
```java
    int[][] dirs = {{0, 1}, {0, -1}, {-1, 0}, {1, 0}};
    public boolean hasPath(int[][] maze, int[] start, int[] destination) {
        boolean[][] visited = new boolean[maze.length][maze[0].length];
        Queue<int[]> q = new LinkedList<>();
        q.add(start);
        visited[start[0]][start[1]] = true;
        while (!q.isEmpty()) {
            int[] cur = q.poll();
            if (cur[0] == destination[0] && cur[1] == destination[1]) return true;
            for (int[] dir: dirs) {
                int x = cur[0] + dir[0], y = cur[1] + dir[1];
                while (x >= 0 && y >= 0 && x < maze.length && y < maze[0].length && maze[x][y] == 0) {
                    x += dir[0];
                    y += dir[1];
                }
                x -= dir[0]; //
                y -= dir[1];
                if (!visited[x][y]) {
                    q.add(new int[]{x, y});
                    visited[x][y] = true;
                }
            }
        }
        return false;
    }
```

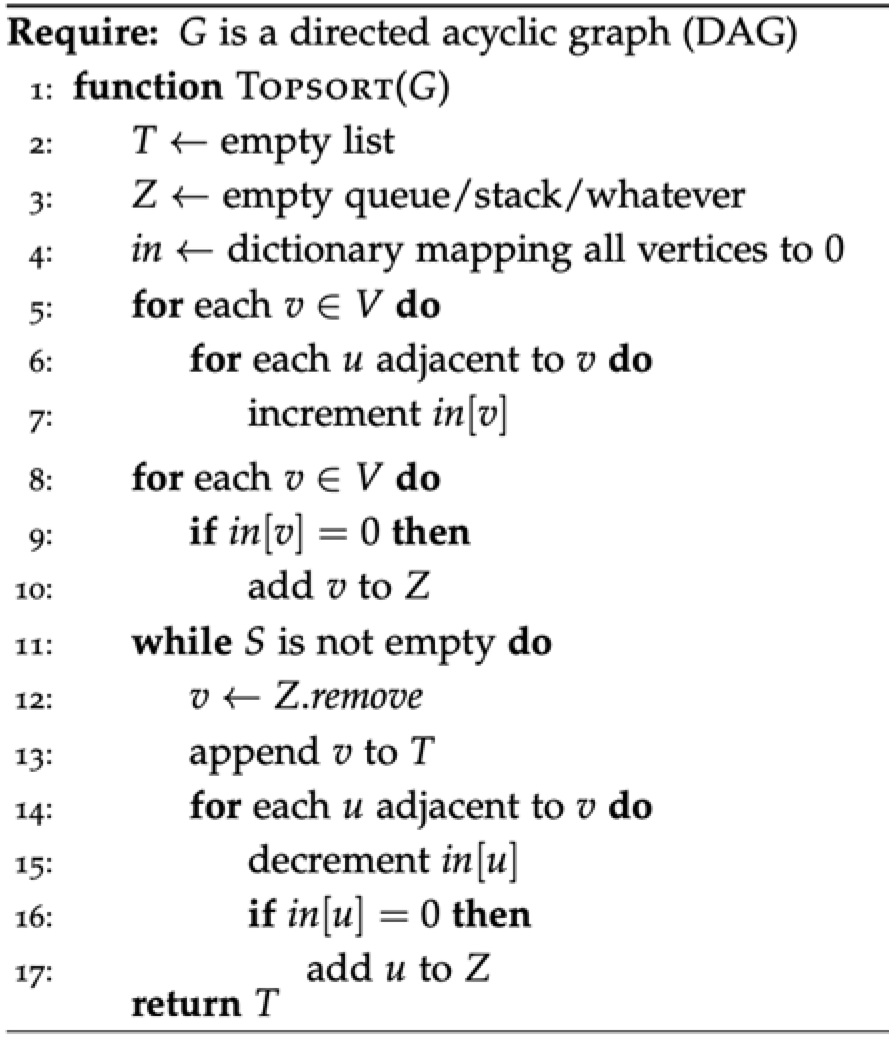
Optimize BFS
(Leetcode 127, Word Ladder , Leetcode 752 Open the Lock)

One way direction BFS searches from the start node expanding down until gets to the end node. While bi way direction BFS searches from both start node and end nodes and expand from both sides, until two search node sets have the intersections 。



BFS in Topological Sort (Leetcode 207, Course Schedule)





```
Require: $G$ is a directed acyclic graph (DAG)
function TopSort($G$)
    $T \leftarrow$ empty list
    $Z \leftarrow$ empty queue/stack/whatever
    $in \leftarrow$ dictionary mapping all vertices to 0
    for each $v \in V$ do
        for each $u$ adjacent to $v$ do
            increment $in[v]$
    for each $v \in V$ do
        if $in[v] = 0$ then
            add $v$ to $Z$
    while $S$ is not empty do
        $v \leftarrow Z.remove$
        append $v$ to $T$
        for each $u$ adjacent to $v$ do
            decrement $in[u]$
            if $in[u] = 0$ then
                add $u$ to $Z$
    return $T$
```

CCC '21 S4 - Daily Commute(BFS)

1. Train runs once daily, get off train only once.

   Only Get off train if it is faster to walk for that station.
   [1 2 3 4 5 6]   w: 2->5
   take subway: 1->2->3->4->5->6   5 minutes
   subway and walk   1->2 walk to 5 2 minutes, wait 2 minutes for subway to 5.

2. Walkways never change
   Using BFS to set the walk distance to station N from every station.
   Reverse search start node N

3. Subway Station Array index is the minutes to get that station.
   `[1 4 3 2]-> get to station 3, index(from 0) is 2, 2 minutes`

4. `Each day only 2 subway station indexes changed`

5. Final Commute Time = Subway minutes + walk Minutes

   Calculate every station commute time and put into an sorted set.
   For each day removed the 2 swapped stations
   swap the stations
   re-calculate these 2 station commute time and put back to the set.
   Print out the first minimum value from the set.

   The sorted Set key should be commute time + station number.

```
bfs(int start, graph,  walkDistances) {
       Queue<Integer> queue = new LinkedList<>();
       boolean [] vis = new boolean[start+1];  // start is station N
       queue.add(start);
       walkDistances[start]  = 0;
       vis[start] = true;
       while (!queue.isEmpty()) {
           int u = queue.poll();
           for (int v : graph[u]) {
               if(!vis[v]){
                   queue.add(v);
                   walkDistances[v]  = walkDistances[u] + 1;
                   vis[v]  = true;
 }}}}
KeyPair (dist, stationId)
CommuteTime(N, W[], S[], D[]){
    Set<KeyPair>  commuteTimeSet;
     graph =  scan W[] to generate graph.
    bfs(N, graph,  walkDistances);
    For (each station index of  S[]){
        StationNo = s[index]
        trainDistances[StationNo] = index; // train minutes
    }
    For ( each station id) {
         totalCommuteTime  = walkDistances[id]  + trainDistances[id];
         CommuteTimeSet.add(  KeyPair(totalCommuteTime,id);
    }
    For (each d in D[]){
        get swapped index, i1,i2  and station no s1, s2;
        CommuteTimeSet.remove( s1 commute time);
        CommuteTimeSet.remove( s2 commute time);
        Swap  s1, s2  trainDistances
        Swap  i1, i2  S[] // current train route array
         CommuteTimeSet.add(  s1 commute  time);
         CommuteTimeSet.add(  s2 commute  time);
    }
     Print(CommuteTimeSet[0].dist);
}
```

BFS Dijkstra's Shortest Path Algorithm using priority_queue CCC '15 S4 - Convex Hull https://dmoj.ca/problem/ccc15s4

```
1.PriortyQueue  Edge as Key

Edge { dest, time, hull, compareTo(time)}

1.BFS to traverse the Edges and update the output Dist array

  Dist[i][j]  ->  the travel time from start point to ith Island and remaining j Hull
  Update and add to queue only for 2 conditions

  Rest Hull >0

  The total travel to this island < saved island time in Dist array.

1. Print the minimum time from Dist[] .
```

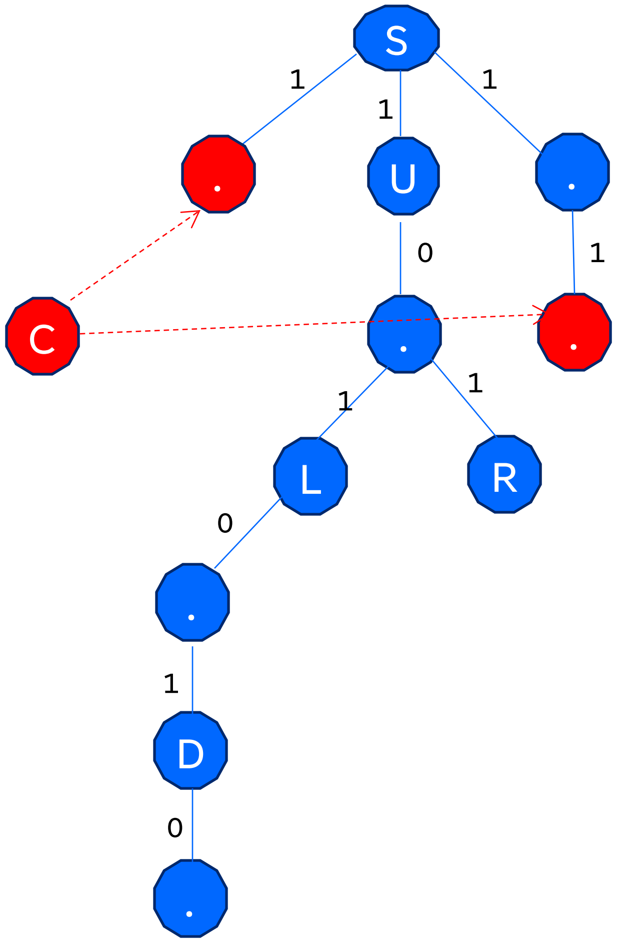
Dijkstra’s Shortest Path Algorithm using BFS PQ

```
1.Initialize distances of all vertices as infinite.

2.Create an empty priority queue pq. Every item of pq is a pair (weight, vertex). Weight (or
  distance) is used as comparator

3.Insert source vertex into pq and make its distance as 0.

   While either pq doesn't become empty
    a) Extract minimum distance vertex from pq u.
    b) Loop through all adjacent of u and do following for every vertex v.
           // If there is a shorter path to v through u.
          If dist[v] > dist[u] + weight(u, v)
               (i) Update distance of v, i.e., do
                     dist[v] = dist[u] + weight(u, v)
               (ii) Insert v into the pq (Even if v is already there)

4. Print distance array dist[] to print all shortest paths.
```

BFS Dijkstra’s Shortest Path Algorithm using priority_queue

CCC '18 S3 - RoboThieves https://dmoj.ca/problem/ccc18s3

```
WWWWWWW
WD.L.RW
W.WCU.W
WWW.S.W
WWWWWWW
```



```
BoboThieves( G,   ANS ){
  G: char[N][M]
  ANS: int[N][M]   // will save the steps from start for each cell.
  CC: boolean[N][M]  // will mark if the cell is under camera.
  MarkCC(G, CC)
  bfs(G, CC, ANS)
  print ANS cell steps marked as '.'
}
MarkCC( G, CC) {
   // mark all the cells can be caught by Camera
    find all  'C' x, y from G
    for ecah x, y marked as 'C'
       scan all 4 directions to mark the cells in CC to true if the camera can catch.
}
Bfs( start, G, CC, ANS){
   Queue <Pair<x,y>>  // in java we can use LInkedList with 2 values as pair.
   Queue.add(start)
   While(!Queue.isEmpty(){
       currentCell = Queue.poll()
       if currentCell is '.'   moveStep = 1, moveDir = 4 directions
       if currentCell is Conveyor, MoveStep = 0 moveDir = one direction.
       for(nextCell : moveDir){
           if(canMoveToNextCell){  // inMoveRange and not 'W' and not in CC'
             If(not visited[extCell] | ANS[nextCell] > ANS[currentCell] + moveStep){
                visited[nextCel] = true;
                ANS[nextCell] = ANS[currentCell] + moveStep;
          Queue.add(nextCell);
           }
       }
   }
}
```

Kruskal Minimum Cost Spanning Tree Algorithm

1. Sort all the edges in non-decreasing order of their weight.
2. Pick the smallest edge. Check if it forms a cycle with the spanning tree formed so far. If the cycle is not formed, include this edge, else, discard it.
3. Repeat step#2 until there are (V-1) edges in the spanning tree.
4. https://www.cs.usfca.edu/~galles/visualization/Kruskal.html
5. https://github.com/rayliu7717/CCC_CLASS/blob/main/CCC_KruskalsMST.java

Kruskal Minimum Cost Spanning Tree Algorithm
CCC '10 S4 - Animal Farm https://dmoj.ca/problem/ccc10s4

1. Convert Corner Vertex Graph to Pen Vertex Graph
2. Pen Vertex Graph may have multiple edges to connect to 2 Pens with different edge length.
3. Use **Kruskal Minimum Cost Spanning Tree Algorithm to find minimal cost**

DFS   Depth-First Search

- Start at the root node and explore as far as possible along each branch before backtracking.
- Graphs may contain cycles (a node may be visited twice). To avoid processing a node more than once, use a boolean visited array. A graph can have more than one DFS traversal.
- https://www.cs.usfca.edu/~galles/visualization/DFS.html
- https://github.com/rayliu7717/CCC_CLASS/blob/main/CCC_GRAPH_DFS.java

DFS Template

```
Result = []
void DFS ( path, list)
    if(match exist condition)
          result.add(path)
    For (item : list)
        select this item
        DFS ( path, list) // back track
        cancel the selection
```

DFS is a Brute Force to enumerate all combinations.

Enumerate recursively,

Cannot use "for loop" to implement since we don't know how many loop level yet.

DFS Classic Problems

- Leetcode 78 Subset

  https://leetcode.com/problems/subsets/

- Leetcode 46 Permutations
- https://leetcode.com/problems/permutations/submissions/
- Leetcode 77 Combinations
- Leetcode 37 Sudoku Solver
- Leetcode 51 N-Queens

Travel Plan (DFS)

There are n cities, and the adjacency matrix arr represents the distance between any two cities.arr[i][j] represents the distance from city i to city j .Alice made a travel plan on the weekend. She started from city 0, then she traveled other cities 1 ~ n-1, and finally returned to city 0. Alice wants to know the minimum distance she needs to walk to complete the travel plan. Return this minimum distance. Except for city 0, every city can only pass once, and city 0 can only be the starting point and destination. Alice can't pass city 0 during travel.

Input:

[[0,1,2],[1,0,2],[2,1,0]]

Output:

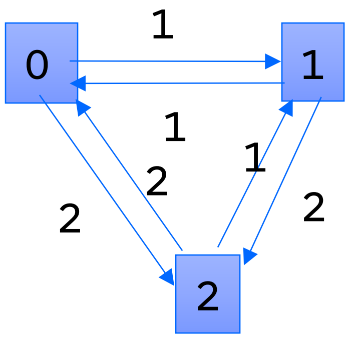
4

Explanation:

There are two possible plans.

The first, city 0-> city 1-> city 2-> city 0, cost = 5.

The second, city 0-> city 2-> city 1-> city 0, cost = 4.



```
public class Solution {
    /**
     * @param arr: the distance between any two cities
     * @return: the minimum distance Alice needs to walk to
complete the travel plan
     */
    void dfs(int [][] arr, int nowpos, int n, boolean[] vis, int
sum, int cnt, int[] ans )
    {
        // exit
        if(cnt == n -1){
            ans[0] = Math.min(ans[0], sum+ arr[nowpos][0]);
            return;
        }
        for(int i = 1;i<n; ++i){
            if(!vis[i]){
                vis[i] = true;
                dfs(arr,i, n, vis,sum+ arr[nowpos][i], cnt + 1,
ans);
                vis[i] = false;  //backtrak
            }
        }
    }
    public int travelPlan(int[][] arr) {
        // Write your code here.
        int n = arr.length;
        boolean [] vis = new boolean[n];
        int[] ans = new int[1];
        ans[0] = Integer.MAX_VALUE;
        dfs(arr, 0, n, vis, 0, 0,ans);
        return ans[0];
    }
}
```

CCC '04 S3 – Spreadsheet (DFS)

1. Each Cell is a Graph Node (id is r,c)
2. Letter with number is graph edge
3. Traverse each Node to check circle
4. All the nodes on the circle will mark with "*"
5. If no circle, calculate the sum number.

```
int dfs(int r, int c, grid[][], vis[][]) {
   if (isNumeric(grid[r][c])) return grid[r][c].toInt();
   if (vis[r][c] || grid[r][c] == "*") return -1;
   vis[r][c] = true;
   String [] depend = grid[r][c].split("+");
    int sum = 0;
    for (int i = 0; i < depend.length; i++) {
        int ret = dfs(depend[i].charAt(0)-'A', depend[i].charAt(1)-'0' - 1);
        if (ret==-1) { grid[r][c] = "*"; return -1;}
        else sum+=ret;
    }
    grid[r][c] = sum.toString();
    return sum;
}

void spreadsheet(grid, rows, cols)
{
     for (int i = 0; i < rows; i++) {
         for (int j = 0; j < cols; j++) {
             vis = new boolean[rows][cols];
              dfs(i, j, grid, vis );
         }
     }
}
```

Greedy Algorithm

- Build up a solution piece by piece, always choosing the next piece that offers the most obvious and immediate benefit.
- The problems where choosing locally optimal also leads to global solution are the best fit for Greedy.

Thank you

Gerry Liu

Liu.geyang@gmail.com