I recently appeared for Juspay OA for SDE-1. There were three questions to be solved, I was only able to solve this one. I am posting the question along with the test case and my solution  
**Round 1**: It was an online round hosted of Juspay on Talscale. It consisted of 3 coding question. The coding question goes like this:

**Problem Description :-** Largest Sum Cycle  
Given a maze with N cells. Each cell may have multiple entry points but not more than one exit(i.e entry/exit points are unidirectional doors like valves).

You are given an array Edge[] of N integers, where Edge[i] contains the cell number that can be reached from of cell i in one step. Edge[i] is -1 if the ith cell doesn’t have an exit.

**The task is to find :-** the sum of the largest sum cycle in the maze(Sum of a cycle is the sum of node number of all nodes in that cycle).

**Note:-** The cells are named with an integer value from 0 to N-1. If there is no node pointing to the ith node then weight of the ith node is zero.

**INPUT FORMAT :-**

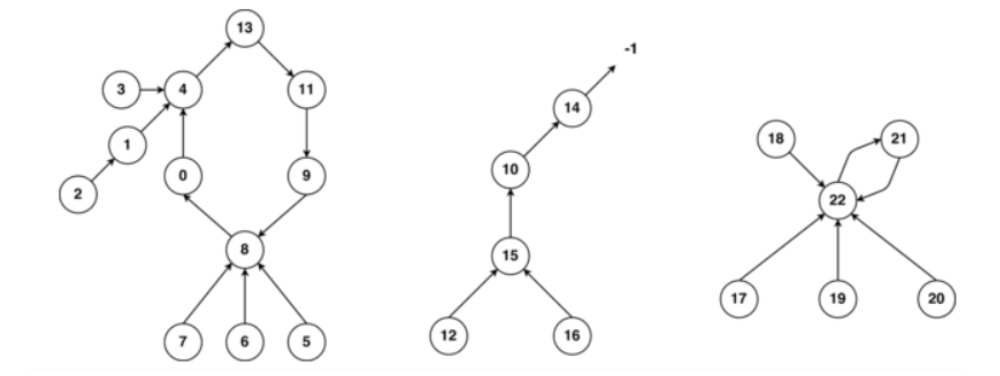
1. The first line contains the number of cells N.
2. The second line has a list of N values of the edge[ ] array, where edge[i] conatins the cell number that can be reached from cell 'i' in one step. edge[i] is -1 if the ith doesn't have ans exit.

**OUTPUT FORMAT :**

1. First line denotes length of the largest cycle..

**Sample Input :**  
23  
4 4 1 4 13 8 8 8 0 8 14 9 15 11 -1 10 15 22 22 22 22 22 21

**Sample Output :**  
6



**Working Solution :-**

int solution(vector<int> arr, int src, int dest){

// Two maps, visA for distance from src and visB for distance from dest

// They serve two purpose, if visA[x] == 0, that means we haven't reached that node yet,

// and if it holds any value, say d, that means it is d distance away from the particular node

map<int,int> visA,visB;

int start = arr[src];

int curr = 1;

set<int> s; // contains unique set of nodes to check at last

// iniitializing final nodes

for(auto &x: arr){

s.insert(x);

}

// traversing until we get to a cell where we've already reached

while(visA[start] == 0){

visA[start] = curr; // Marcking the distance

curr++;

start = arr[start];

if(start == -1){

break; // Getting out if we get to a node who is not pointing at any other node

}

}

start = arr[dest];

// Same logic as above but traversing from dest

while(visB[start] == 0){

visB[start] = curr;

curr++;

start = arr[start];

if(start == -1){

break;

}

}

// This is an array of two values, vp[i].first holds the sum of distance of vp[i].second from src and dest.

vector<pair<int,int>> vp;

for(auto &x: s){

if(visA[x] != 0 && visB[x] != 0){ // Checking if we ever got to that particular node from both src and dest or not

pair<int,int> p = {visA[x] + visB[x], x};

vp.push\_back(p);

}

}

// sorting and finding the node with list sum of visA[} + visB[]

sort(vp.begin(), vp.end());

return vp[0].second;

}

I recently appeared for Juspay OA for SDE-1. There were three questions to be solved, I was only able to solve this one. I am posting the question along with the test case and my solution

Round 1: It was an online round hosted of Juspay on Talscale. It consisted of 3 coding question. The coding question goes like this:

Problem Description :- Maximum Weight Node

Given a maze with N cells. Each cell may have multiple entry points but not more than one exit(i.e entry/exit points are unidirectional doors like valves).

You are given an array Edge[] of N integers, where Edge[i] contains the cell number that can be reached from of cell i in one step. Edge[i] is -1 if the ith cell doesn’t have an exit.

The task is to find :- the node number of maximum weight node(Weight of the node is the sum of node numbers of all nodes pointing to that node).

Note:- The cells are named with an integer value from 0 to N-1. If there is no node pointing to the ith node then weight of the ith node is zero.

INPUT FORMAT :-

The first line contains the number of cells N.

The second line has a list of N values of the edge[ ] array, where edge[i] conatins the cell number that can be reached from cell 'i' in one step. edge[i] is -1 if the ith doesn't have ans exit.

OUTPUT FORMAT :

First line denotes the node number with maximum weight node.

Sample Input :

23

4 4 1 4 13 8 8 8 0 8 14 9 15 11 -1 10 15 22 22 22 22 22 21

Sample Output :

22

image

Working Solution :-

int solution(vector<int>arr){

int ans=INT\_MIN;

int result=-1;

vector<int>weight(arr.size(),0);

for(int i=0;i<arr.size();i++){

int source=i;

int dest=arr[i];

if(dest!=-1){

weight[dest]+=source;

if(ans<=weight[dest]){

ans=max(ans,weight[dest]);

result=dest;

}

}

}

if(ans!=INT\_MIN)

return result;

return -1;

}

I recently appeared for Juspay OA for SDE-1. There were three questions to be solved, I was only able to solve this one. I am posting the question along with the test case and my solution

Round 1: It was an online round hosted of Juspay on Talscale. It consisted of 3 coding question. The coding question goes like this:

Problem Description :- Nearest Meeting Cell

You are given a maze with N cells. Each cell may have multiple entry points but not more than one exit (i.e. entry/exit points are unidirectional doors like valves). The cells are named with an integer from 0 to N-1.

You are given an array Edge[] of N integers, where Edge[i] contains the cell number that can be reached from of cell i in one step. Edge[i] is -1 if the ith cell doesn’t have an exit.

The task is to find :- Nearest meeting cell : Given any two cells - C1, C2, find the closest cell Cm that can be reached from both C1 and C2.

Note:- The cells are named with an integer value from 0 to N-1. If there is no node pointing to the ith node then weight of the ith node is zero.

INPUT FORMAT :-

The first line contains the number of cells N.

The second line has a list of N values of the edge[ ] array, where edge[i] conatins the cell number that can be reached from cell 'i' in one step. edge[i] is -1 if the ith doesn't have an exit.

Third line for each testcase contains two cell numbers whose nearest meeting cell needs to be found. (return -1 if there is no meeting cell from tw.

OUTPUT FORMAT :

1.Output a single line that denotes the nearest meeting cell (NMC).

Sample Input :

23

4 4 1 4 13 8 8 8 0 8 14 9 15 11 -1 10 15 22 22 22 22 22 21

9 2

Sample Output :

4

image

Working Solution :-

int leastCommonDescendent(int nodes[], int N, int node1, int node2){

int \*visited = new int [N];

int cnt1 = 0; //used for counting length of path from node1 to node2

int cnt2 = 0; //used for counting length of path from node2 to node1

int mark = node1; //storing as a marker needed later for detecting end of search

if(node1 == node2) return node2;

for(int i = 0; i < N; i++){

visited[i] = 0;

}

while((nodes[node1] != node1) && (nodes[node1] != -1) && (visited[node1] == 0) && (node1 != node2)){

visited[node1]++;

node1 = nodes[node1];

cnt1++;

}

visited[node1]++; //so that first node in cycle has count 2

//if find a node in 2nd iteration that has count 2

//such that when node1 == node2 it means we are in the same subgraph

//elsif node1 != node2 we are in different sub graphs

while((nodes[node2] != node2) && (nodes[node2] != -1) && (visited[node2] != 2) && (node1 != node2)){

visited[node2]++;

node2 = nodes[node2];

cnt2++;

}

//In below case the nodes are in different disjoint subgraphs

//and both subgraphs have loops so node1 can never be equal to node2

//cout << visited[node1] << visited[node2] << endl;

if(node1 != node2) return -1;

//In below case both nodes are in different disjoint subgraphs

//but there is no loop in 1st one(containing node1)

//and 2nd one has a loop

if ((nodes[node1] == -1) && (visited[node2] == 1)) return -1;

//In below case both nodes are in different disjoint subgraphs

//but 1st one has a loop and second one doesn't

if(nodes[node2] == -1) return -1;

//In below case both nodes are in same subgraph so we

//need to check the length of two alternate paths

if(cnt1 > cnt2)

return node2;

else

return mark;

}

I recently appeared for Juspay OA for SDE-1. There were three questions to be solved, I was only able to solve this one. I am posting the question along with the test case and my solution

Round 1: It was an online round hosted of Juspay on Talscale. It consisted of 3 coding question. The coding question goes like this:

Problem Description :- Largest Sum Cycle

Given a maze with N cells. Each cell may have multiple entry points but not more than one exit(i.e entry/exit points are unidirectional doors like valves).

You are given an array Edge[] of N integers, where Edge[i] contains the cell number that can be reached from of cell i in one step. Edge[i] is -1 if the ith cell doesn’t have an exit.

The task is to find :- the sum of the largest sum cycle in the maze(Sum of a cycle is the sum of node number of all nodes in that cycle).

Note:- The cells are named with an integer value from 0 to N-1. If there is no node pointing to the ith node then weight of the ith node is zero.

INPUT FORMAT :-

The first line contains the number of cells N.

The second line has a list of N values of the edge[ ] array, where edge[i] conatins the cell number that can be reached from cell 'i' in one step. edge[i] is -1 if the ith doesn't have ans exit.

OUTPUT FORMAT :

First line denotes length of the largest cycle..

Sample Input :

23

4 4 1 4 13 8 8 8 0 8 14 9 15 11 -1 10 15 22 22 22 22 22 21

Sample Output :

6

image

Working Solution :-

int solution(vector<int> arr, int src, int dest){

// Two maps, visA for distance from src and visB for distance from dest

// They serve two purpose, if visA[x] == 0, that means we haven't reached that node yet,

// and if it holds any value, say d, that means it is d distance away from the particular node

map<int,int> visA,visB;

int start = arr[src];

int curr = 1;

set<int> s; // contains unique set of nodes to check at last

// iniitializing final nodes

for(auto &x: arr){

s.insert(x);

}

// traversing until we get to a cell where we've already reached

while(visA[start] == 0){

visA[start] = curr; // Marcking the distance

curr++;

start = arr[start];

if(start == -1){

break; // Getting out if we get to a node who is not pointing at any other node

}

}

start = arr[dest];

// Same logic as above but traversing from dest

while(visB[start] == 0){

visB[start] = curr;

curr++;

start = arr[start];

if(start == -1){

break;

}

}

// This is an array of two values, vp[i].first holds the sum of distance of vp[i].second from src and dest.

vector<pair<int,int>> vp;

for(auto &x: s){

if(visA[x] != 0 && visB[x] != 0){ // Checking if we ever got to that particular node from both src and dest or not

pair<int,int> p = {visA[x] + visB[x], x};

vp.push\_back(p);

}

}

// sorting and finding the node with list sum of visA[} + visB[]

sort(vp.begin(), vp.end());

return vp[0].second;

}

I recently appeared for Juspay OA for SDE-1. There were three questions to be solved, I was only able to solve this one. I am posting the question along with the test case and my solution

Round 1: It was an online round hosted of Juspay on Talscale. It consisted of 3 coding question. The coding question goes like this:

Problem Description :- Maximum Weight Node

Given a maze with N cells. Each cell may have multiple entry points but not more than one exit(i.e entry/exit points are unidirectional doors like valves).

You are given an array Edge[] of N integers, where Edge[i] contains the cell number that can be reached from of cell i in one step. Edge[i] is -1 if the ith cell doesn’t have an exit.

The task is to find :- the node number of maximum weight node(Weight of the node is the sum of node numbers of all nodes pointing to that node).

Note:- The cells are named with an integer value from 0 to N-1. If there is no node pointing to the ith node then weight of the ith node is zero.

INPUT FORMAT :-

The first line contains the number of cells N.

The second line has a list of N values of the edge[ ] array, where edge[i] conatins the cell number that can be reached from cell 'i' in one step. edge[i] is -1 if the ith doesn't have ans exit.

OUTPUT FORMAT :

First line denotes the node number with maximum weight node.

Sample Input :

23

4 4 1 4 13 8 8 8 0 8 14 9 15 11 -1 10 15 22 22 22 22 22 21

Sample Output :

22

image

Working Solution :-

int solution(vector<int>arr){

int ans=INT\_MIN;

int result=-1;

vector<int>weight(arr.size(),0);

for(int i=0;i<arr.size();i++){

int source=i;

int dest=arr[i];

if(dest!=-1){

weight[dest]+=source;

if(ans<=weight[dest]){

ans=max(ans,weight[dest]);

result=dest;

}

}

}

if(ans!=INT\_MIN)

return result;

return -1;

}