October 5, 2025

DRILL ON GRAPH & TREES

Ques 1. Two Currencies

https://atcoder.jp/contests/abc164/tasks/abc164_e

There are N cities numbered 1 to N, connected by M railroads.

You are now at City 1, with 10^{100} gold coins and S silver coins in your pocket.

The i-th railroad connects City U_i and City V_i bidirectionally, and a one-way trip costs A_i silver coins and takes B_i minutes. You cannot use gold coins to pay the fare.

There is an exchange counter in each city. At the exchange counter in City i, you can get C_i silver coins for 1 gold coin. The transaction takes D_i minutes for each gold coin you give. You can exchange any number of gold coins at each exchange counter.

For each $t = 2, \ldots, N$, find the minimum time needed to travel from City 1 to City t. You can ignore the time spent waiting for trains.

Less than or equal to 5000 gold coin is enough (which make 10^{100} gold coins unlimited for us)

Optimise time.

Values on edge : Time to travel (cost)

Also maintain (city, silver) — State

Time — Edge

$[(c,s) \xrightarrow{B_i} (x, S-A_i)]$

$[(c,s) \xrightarrow{D_i} (c, S+C_i)]$

↑ same city ↑ gain C_i silver coins

We will create only nodes (C, S) when C is a valid node (1-N) and S is from (1-5000) Only Nodes Possible.

How many nodes are there ?

→ N · 5000

How many edges ?

→ For each node, we will have (M+1)

So, Total (for N·5000) = (N·5000) (M+1)

Vertex #E

Now, we can apply Dijkstra.

Start from city 1 with min(S, 5000)

Starting State = (1, min(S, 5000))

Dijkstra

Ques 2. Kefa and Park

https://codeforces.com/problemset/problem/580/C

Kefa decided to celebrate his first big salary by going to the restaurant.

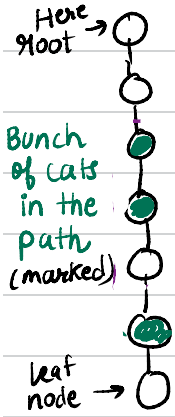
He lives by an unusual park. The park is a rooted tree consisting of n vertices with the root at vertex 1. Vertex 1 also contains Kefa's house. Unfortunaely for our hero, the park also contains cats. Kefa has already found out what are the vertices with cats in them.

The leaf vertices of the park contain restaurants. Kefa wants to choose a restaurant where he will go, but unfortunately he is very afraid of cats, so there is no way he will go to the restaurant if the path from the restaurant to his house contains more than m **consecutive** vertices with cats.

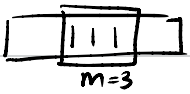
Your task is to help Kefa count the number of restaurants where he can go.

Root at vertex 1 (Given) indicates Rooted Tree.

Suppose m = 3



The question can then be boiled down similar to find (m=3), 3 consecutive 1's in an array



Is there any consecutive chunk of given m which has all cats in our tree.

Maintain consecutive cats seen in ccs

When ccs = m , Stop.

```
cnt = 0;
dfs(node, parent, ccs) {
    if (ccs > m) return;
    int child = 0;
```

```
for (auto v : g[nn]) {
    if (v != parent) {
        child++
        if (val[v] == 1)
        {
            dfs(v, node, ccs+1)
        }
        else
            dfs(v, node, 0)
    }
}
if (child == 0)  // leaf node
{
    cnt++;
}
```

Ques 3. Course Schedule IV

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

There are a total of `numCourses` courses you have to take, labeled from `0` to `numCourses - 1`. You are given an array `prerequisites` where `prerequisites[i] = [a_i, b_i]` indicates that you **must** take course `a_i` first if you want to take course `b_i`.

- For example, the pair `[0, 1]` indicates that you have to take course `0` before you can take course `1`.

Prerequisites can also be **indirect**. If course `a` is a prerequisite of course `b`, and course `b` is a prerequisite of course `c`, then course `a` is a prerequisite of course `c`.

You are also given an array `queries` where `queries[j] = [u_j, v_j]`. For the `j^th` query, you should answer whether course `u_j` is a prerequisite of course `v_j` or not.

Return *a boolean array* `answer`, *where* `answer[j]` *is the answer to the* `j^th` *query.*

Example 1:

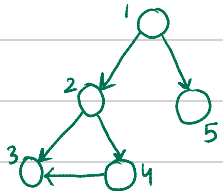
Input: numCourses = 2, prerequisites = [[1,0]], queries = [[0,1],[1,0]]
Output: [false,true]
Explanation: The pair [1, 0] indicates that you have to take course 1 before you can take course 0.
Course 0 is not a prerequisite of course 1, but the opposite is true.

Since we are dealing with prerequisites Topological sort can be one choice.

Here, if x comes before y

Can we say that x is a pre req of y?
No, it doesn't mean that. Topological ordering says if x is a pre-req of y then x should come before y in the ordering but if they are not related, still x can come before y and in such a case x won't be a pre-req of y.

Example:



Here, one of the possible topological ordering that can be done is:

- 1 5 2 4 3 (valid)

→ Does this mean 5 is a pre req of 3?
NO. So just because something occurs before doesn't make it a pre-requisite.
Topological ordering is not unique.
So, we now know that Topological ordering cannot directly help us in this problem.
One solution can be: For every node, do a DFS. We

will get which all nodes can be reached and then we can answer acc. to that.

$$T.C = O(N+M) \cdot N^2 \cdot N$$

Since no. of edges can be fairly large, it will be N^2 only. (propogating for n times)

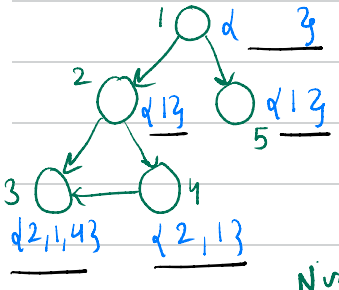
Ideal Solution :

If there are 100 courses, how many diff. unique queries can possibly come? $\Rightarrow {}^{N}C_2 = {}^{100}C_2$ queries

If this ${}^{N}C_2$ is less or equal to the query length then we can easily pre process every answer. Every query will take $O(1)$

Pre compute everything

- Maintain a set for every node
- Build Topological ordering of the graph
- Go front → Back, assign pre-req to childs by parent value.



Set of pre-requisites

$M \cdot N$

N val pushing to next node

$$T.C = O(N \cdot N + M \cdot N)$$

for every node

Total push

```cpp
int n,m;
vector<vector<int>> g;

vector<int> indeg;
queue<int> zeroindeg;
```

```cpp
vector<int> topo;
void build_topo(){ // Kahn's Algorithm
    for(int i=1;i<=n;i++){
        if(indeg[i]==0)zeroindeg.push(i);
    }
    while(!zeroindeg.empty()){
        int cur = zeroindeg.front();
        zeroindeg.pop();
        topo.push(cur);
        for(auto v:g[cur]){
            indeg[v]--;
            if(indeg[v]==0)zeroindeg.push(v);
        }
    }
}
```

push_back

```cpp
void solve(){
    cin>>n>>m;
    g.resize(n+1);
    indeg.assign(n+1,0);
    for(int i=0;i<m;i++){
        int a,b;cin>>a>>b;
        g[a].push_back(b);
        indeg[b]++;
    }
    build_topo();
    int prereq[n+1][n+1];
    for(auto x:topo){
        for(int i=1;i<=n;i++){
            if(prereq[i][x]){
                for(auto v:g[x]){
                    prereq[i][v]=1;
                }
            }
        }
        for(auto v:g[x]){
            prereq[x][v]=1;
        }
    }
    for(auto x:topo){
        cout<<x<<" : ";
        for(int i=1;i<=n;i++){
            if(prereq[i][x]){
                cout<<i<<", ";
            }
        }
        cout<<endl;
    }
```

memset (prereq, 0, sizeof(0))

```
5 5
1 2
1 5
2 3
2 4
4 3
Output
1 :
2 : 1,
5 : 1,
4 : 1, 2,
3 : 1, 2, 4,
```

AlgoZenith

Ques 4. Count ways to Build Rooms in an Ant Colony.

https://leetcode.com/problems/count-ways-to-build-rooms-in-an-ant-colony/

You are an ant tasked with adding `n` new rooms numbered `0` to `n-1` to your colony. You are given the expansion plan as a **0-indexed** integer array of length `n`, `prevRoom`, where `prevRoom[i]` indicates that you must build room `prevRoom[i]` before building room `i`, and these two rooms must be connected **directly**. Room `0` is already built, so `prevRoom[0] = -1`. The expansion plan is given such that once all the rooms are built, every room will be reachable from room `0`.

You can only build **one room** at a time, and you can travel freely between rooms you have **already built** only if they are **connected**. You can choose to build **any room** as long as its **previous room** is already built.

Return *the **number of different orders** you can build all the rooms in*. Since the answer may be large, return it **modulo** $10^9 + 7$.

Solution blog:

https://codeforces.com/blog/entry/75627

```
void solve(){
    cin>>n>>m;
    g.resize(n+1);
    indeg.assign(n+1,0);
    for(int i=0;i<m;i++){
        int a,b;cin>>a>>b;
        g[a].push_back(b);
        indeg[b]++;
    }
    build_topo();

    vector<int> quickest(n+1,0);
    for(auto x:topo){
        for(auto v:g[x]){
            quickest[v] = max(quickest[v], quickest[x] + timetaken[v])
        }
    }
    // answer queries in O(1).
}
```

Ques 5. Find Edges in Shortest Paths

https://leetcode.com/problems/find-edges-in-shortest-paths/description/

You are given an undirected weighted graph of `n` nodes numbered from 0 to `n - 1`. The graph consists of `m` edges represented by a 2D array `edges`, where `edges[i] = [a_i, b_i, w_i]` indicates that there is an edge between nodes a_i and b_i with weight w_i.

Consider all the shortest paths from node 0 to node `n - 1` in the graph. You need to find a **boolean** array `answer` where `answer[i]` is `true` if the edge `edges[i]` is part of **at least** one shortest path. Otherwise, `answer[i]` is `false`.

Return the array `answer`.

Note that the graph may not be connected.

Ques 5. Parallel Courses III

https://leetcode.com/problems/parallel-courses-iii/

You are given an integer `n`, which indicates that there are `n` courses labeled from `1` to `n`. You are also given a 2D integer array `relations` where `relations[j] = [prevCourse_j, nextCourse_j]` denotes that course `prevCourse_j` has to be completed **before** course `nextCourse_j` (prerequisite relationship). Furthermore, you are given a **0-indexed** integer array `time` where `time[i]` denotes how many **months** it takes to complete the `(i+1)^th` course.

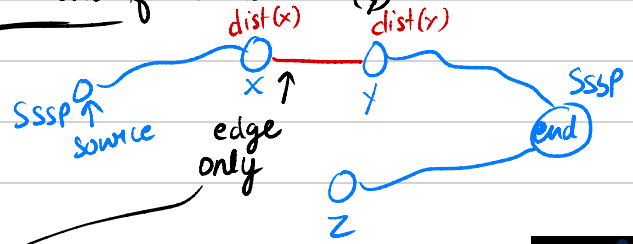
You must find the **minimum** number of months needed to complete all the courses following these rules:

- You may start taking a course at **any time** if the prerequisites are met.
- **Any number of courses** can be taken at the **same time**.

Return *the **minimum** number of months needed to complete all the courses*.

Note: The test cases are generated such that it is possible to complete every course (i.e., the graph is a directed acyclic graph).

From the starting node, we can find SSSP to any node.
We can also run SSSP from the ending node to every node.
So, we can get the distance from source $dist_{src}(x)$ & we can get the distance from end $dist_{end}(y)$



Here, pre-req of every node is available. So, changes in Q3 code:

$dist_{sc}(x) + wt(e) + dist_{end}(x) == dist_{sc}(end)$

either

When:

$dist_{sc}(x) + dist_{end}(y) + wt(e) == dist_{sc}(end)$

AlgoZenith