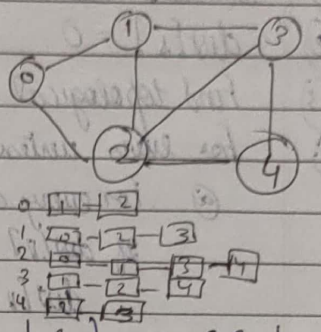
GRAPHS → Random connection among nodes

PAGE NO.
DATE: / /20

BFS →

source is given

```
void BFS (vector<int> adj[], int V, int s)
{
bool visited[V+1];
for (int i=0; i<V; i++) visited[i] = false;
queue<int> q;
visited[s] = true;
q.push(s);
while (q.empty()==false)
{
    int u = q.front(); q.pop();
    cout << u << " ";
    for (int v : adj[u]) if (visited[v]==false) visited[v]=true, q.push(v);
}
```




```
void BFS2( vector<int> adj[], int V)
{
bool visited[V+1];
for (int i=0; i<V; i++) visited[i] = false;
    for (int i=0; i<V; i++)
        if (visited[i]==false) BFS(adj, i, visited);
}
```

when source is not given / it is disconnected

★ PRIMS ALGO is used to find minimum spanning tree for given graph.
★ for general weighted graph

→ we can get single source shortest distance in O(VE) using Bellman-Ford Algorithm given no negative cycle exists.

→ for graph with no -ve weights we can get single source shortest distance in O(E + V log V) using Dijkstra

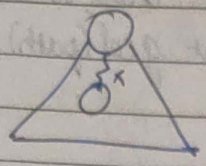
→ For DAG we can get single source distance in O(V+E)

Teacher's Signature

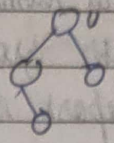
PAGE NO.
DATE: / /20

Tree DP

$N \le 10^5$
$k \le 100$



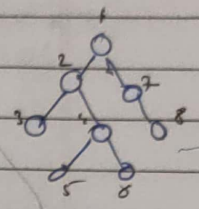
# of node pairs (a, b)
dist (a, b) = k



in dp(0, 0) = 2
in dp(a, 2) = ⊥

D) N node Tree
- each node has profit associated with it
- But no 2 neighbour can be selected together

1, 3, 4 is valid 1, 3, 5, 6 is valid
1, 4, 5 is not valid

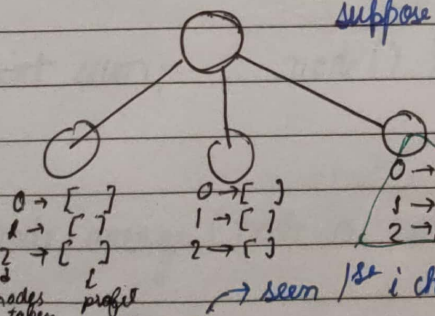


DP(node, 0) → Best profit from subtree by not taking this node
DP(node, 1) → Best profit from subtree by taking this node

$DP(node, 0) = \sum_{child} \max(DP(child, 0), DP(child, 1))$
$DP(node, 1) = \sum_{child} DP(child, 0) + Profit[node]$

D) N node Tree
- each node has profit associated with it
- But no 2 neighbours can be selected together
- Atmost k nodes can be selected

suppose you choose a node. then among child you want to chose k-1 nodes with max profit.



suppose you have to choose k-1 nodes - get max profit across all the childs.

y nodes from this child (KNAPSACK)

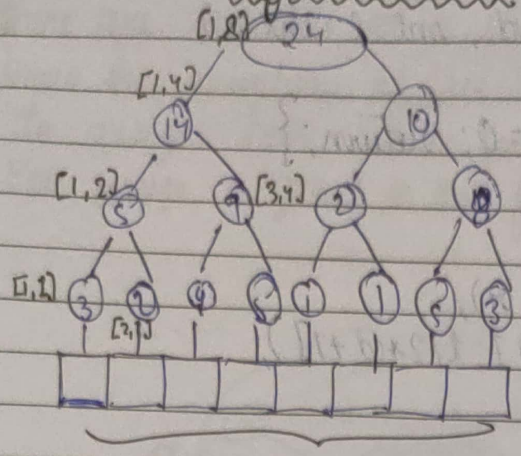
DP2(i, x) → weight seen

$= \max(DP2(i+1, x+y) + DP(child, 0/1))$

Teacher's Signature _____

PAGE NO.
DATE: / /20

Segment Trees

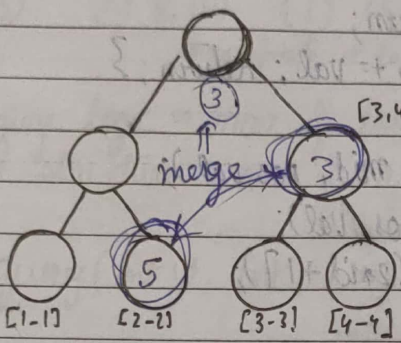


} logn levels

width of every level is 2logn.

---

① To get minimum in a range



Q :→ 2 to 4
Node structure → range ~~of~~ data
Merge logic → min of both ranges
Leaf value → element is min

---

② To get XOR of range [a,b] → Simple Prefix sums if ~~only~~ update query is not present

---

③ Q queries → ① increase each value in [a,b] by u
② get value at position k

```
struct node
{
  int sum;        node() { sum = 0; }
};

node merge (node a, node b)
{
  node ans;
  ans.sum = a.sum + b.sum;
  return ans;
}
```

Teacher's Signature _____

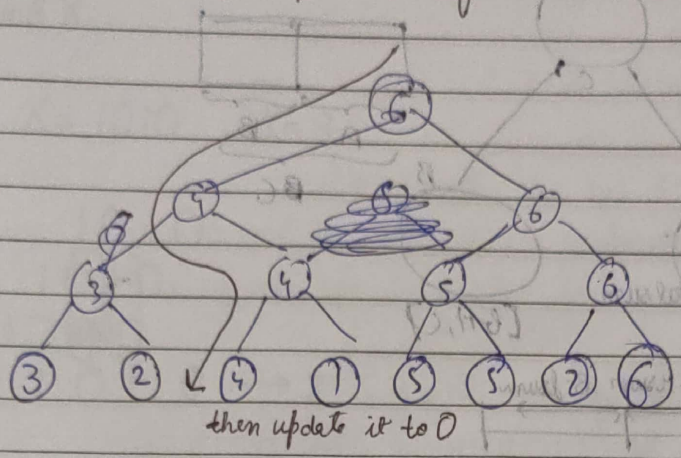
size of given array

```
node t[4 * 2*10^5]
void build (int id, int l, int r)
{
if (l==r) { t[id].sum = 0; return; }
int mid = (l+r)/2;
build (2*id, l, mid);
build (2*id+1, mid+1, r);
t[id] = merge (t[2*id], t[2*id+1]);
}

void update (int id, int l, int r, int pos, int val)
{
if (pos<l || pos>r) return;
if (l==r) { t[id].sum += val; return; }
int mid = (l+r)/2;
~~build~~ update (2*id, l, mid, pos, val);
update (2*id+1, mid+1, r, pos, val);
t[id] = merge (t[2*id], t[2*id+1]);
}

int query (int id, int l, int r, int lq, int rq)
{
if (lq>r || rq<l) return node();
int mid = (l+r)/2;
return (query (2*id, l, mid, lq, rq) +
merge query (2*id+1, mid+1, lq, rq));
}
if (lq<=l && r<=rq) return t[id];

update (1,0,n-1, a, u);   } to add u
update (1,0,n-1, b+1, -u); } to [a,b]

node ans = query (1, 0, n-1, 0, k);
cout<<ans.sum<<ans[k];
}
```

get value

PAGE NO.
DATE: / /20

Q4 There are n hotels on a street. For each hotel you know the number of free rooms. Your task is to assign hotel rooms for a group of students. You assign a group to the first hotel with enough rooms.

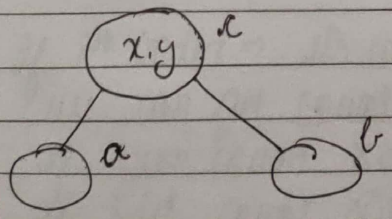


merging logic ⇒ max of both nodes
node structure ⇒ max of range

int query(int id, int l, int r, int req) ~~~ gives index of पेड़ा >=req
{
if (t[id].max_val < req) return 0;
if (l==r) { return l; } → If leaf पे पहुँचे then यहीं पहुँचे
int mid = (l+r)/2;
if (t[id*2].max_val >= req) return query(2*id, l, mid, req);
else return (2*id+1, mid+1, r, req);
}

↓ hume पेड़ा वाला चाहिए

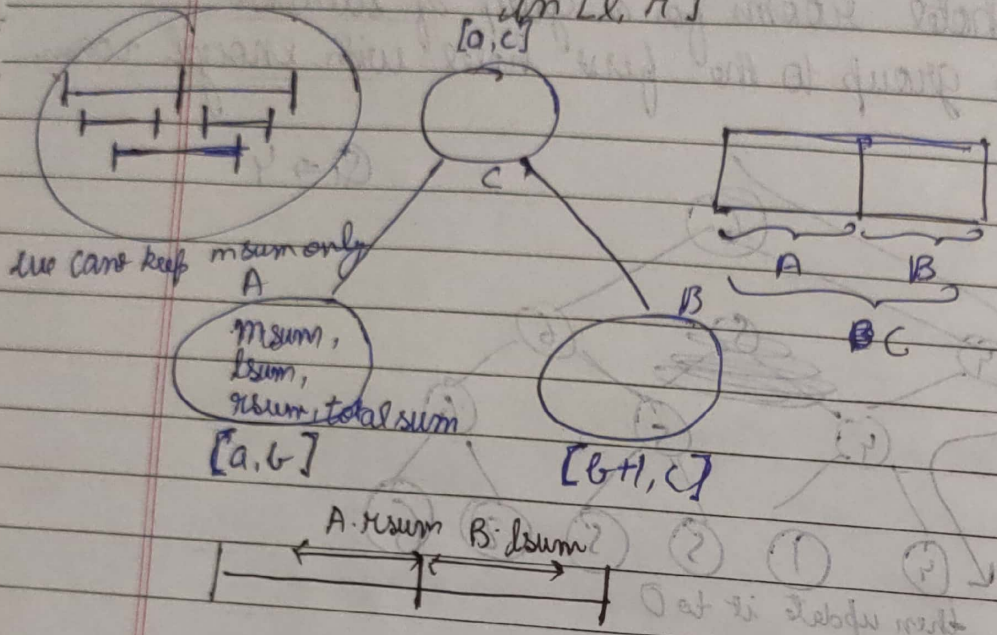
(5) (min, # of min)



if (a.x != b.x)
return (min(a.x, b.x and its one));
else
return (a.x, a.y + b.y);

Teacher's Signature

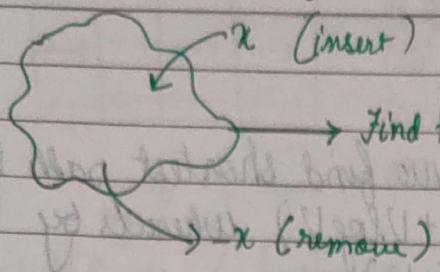
⑥ 1 x V → update arr[x] = V
2 l r → find max sum subarray in [l, r]



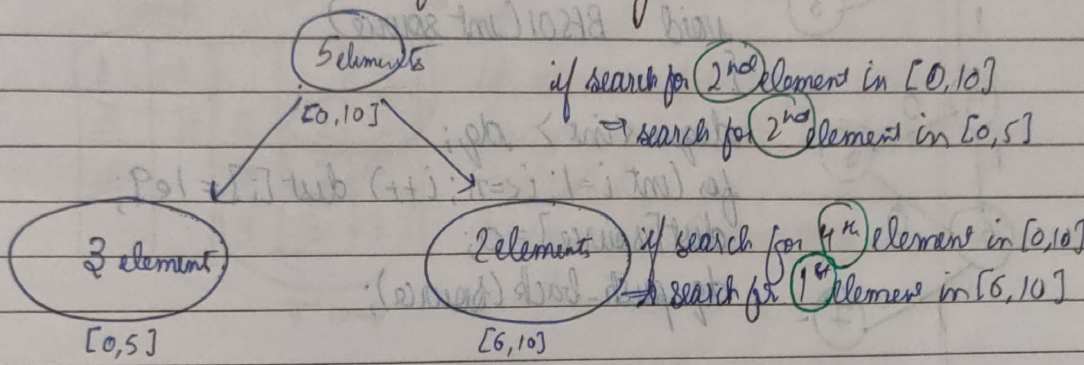
merge
C.msum = max(A.msum, B.msum, A.rsum + B.lsum);
C.lsum = max(A.totalsum + B.lsum, A.lsum);
C.rsum = max(A.rsum + B.totsum, B.rsum);
C.totalsum = A.totalsum + B.totalsum;

PAGE NO.
DATE: / /20

K^{th} segment Tree



Queries → +x → add x to set → no overlap (consider once)
-x → remove x if exist
?K → find k^{th} segment tree




```
findKth(id, l, r, k)
{
if (l==r) return l;
if (k<=t[2*id]) return findKth(2*id, l, mid, k);
else return findKth(2*id+1, mid+1, r, k - t[2*id]);
}
```

~~Inversions~~

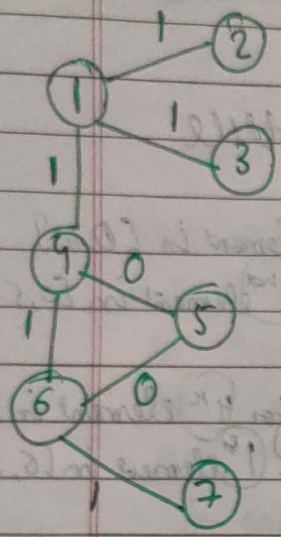
Coordinate Compression

map the coordinates after sorting

Graphs Shortest Paths

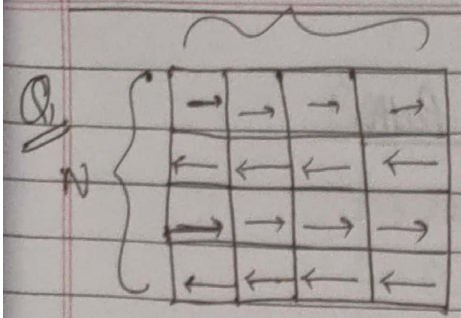
0-1 BFS

Weights are 0 or 1. If we find shortest path by Dijkstra it will be $O(E + V \log V)$ whereas by 0-1 BFS it will be $O(E+V)$

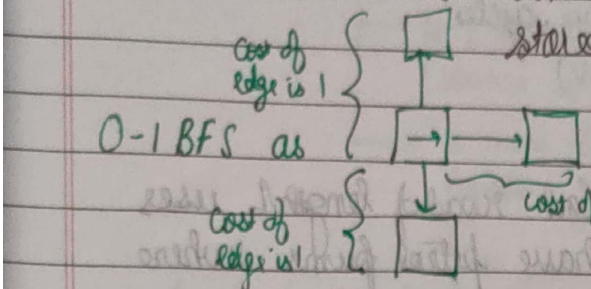


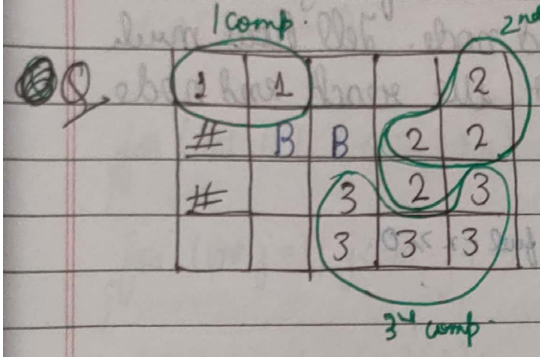
★ If encounter 0 weight, push to front of deq

```
void BFS01(int source)
{
deque<int> dq;
for(int i=1; i<=n; i++) dist[i] = 1e9;
dist[source] = 0;
dq.push_back(source);

while(!dq.empty())
{
int curr = dq.front(); dq.pop_front;
for(auto child : adj[curr]
{
if( dist[child.ff] > dist[curr] + child.ss)
{
dist[child.ff] = dist[curr] + child.ss;
if(child.ss == 0) dq.pushfront(child.ff);
else              dq.push_back(child.ss);
}
}
}
}
```



Q. from (1,1) → (N,M)
If you want to change direction of cell with a cost1, you can else you can move without cost in the direction stored. Tell min cost.





Q. We want to fill empty cells with bridges so that all components get connected.
It is guaranteed that there are 3 components.

Observation → there is always an x ~~pair~~ we can connect components in min cost.

Target → From every cell, find shortest path from that cell to 1,2,3 components. But T.C. not feasible

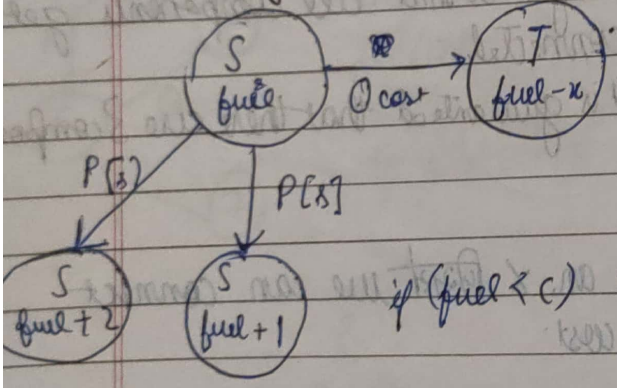
↳ 0-1 BFS as all 1's are connected with 0 weight.
↳ Start multisource 0-1 BFS from 1's to all other nodes.
dist[a][i][j] = ~~mult~~ distance of (i,j) ~~con~~ from a component where a is (1,2,3)
↳ Subtract 2 from ans ⇒ iterate and find dist[1][i][j]+dist[2][i][j]+dist[3][i][j] or minimum −2 (arr[x][y] == '.')

Single Source Shortest Paths

Dijkstra

- fails for negative cycles
- TC $O(E + V \log V)$

Q You have total fuel C. Each road of length l uses l fuel. At nodes you have petrol pumps where you can pay p[i] to ~~fill it~~ increase fuel by 1. You are given start & end node. Tell how much money to spend so that we reach end node.



Q Suppose a weighted graph. Consider node as thread slot and edges as threads. If burning is linear and you burn a source, find the time taken for full burn.

PAGE NO.
DATE: / /20

Bellman Ford Algorithm

- can handle negative cycles
- O(VE)
- N-1 iterations to relax the N-1 nodes (source is already relaxed)
- 1 more iteration to check if -ve weight cycle exists

```
vector<int> dist(n+1, 1e9);
dist[src] = 0;

for (int i=1; i<=n; i++)
{
for (int j=1; j<=m; j++)
{
int u = edges[j][0];
int v = edges[j][1];
int wt = edges[j][2];
if (dist[u] != 1e9 && (dist[u] + wt < dist[v])) //relaxing
  dist[v] = dist[u] + wt;
}
}

bool flag = 0;

for (int i=1; i<=n; i++)
    {
    }

if (dist[u] != 1e9 && (          )) flag = 1;
{ }

if(flag == 0) return dist;
else → contains neg cycle
```

if node dist != 1e9

Teacher's Signature

All pairs Shortest Paths

Floyd Warshal

- We need adj matrix
- adj(i,i) == 0

```
for (k from 1 to n)
for (i from 1 to n)
for (j from 1 to n)
dist[i][j] = min (dist[i][j], dist[i][k] + dist[k][j]);
```

When k = x

we are relaxing distance from i to j by using intermediate k^{th} node

∴ DP(i,j,k) → shortest path from i to j using (1...k)

☆ While taking input

```
for (int i=0; i<m; (i++))
{
int u, v, wt; cin >> u >> v >> wt;
dist[u][v] = min(dist[u][v], wt);
}
```

Initially inf

Q We want to check reachability, like which all nodes are reachable from some node. Dont require the distance.

```
for (i from 0 to n)
for (j from 0 to n)
if (i != j) dist[i][j] = inf;

for (i from 0 to m)
{ int a, b, c; cin >> a >> b >> c;
  dist[a][b] = 1;
}
```

Teacher's Signature

```
for (k from 0 to n)
for (i from 0 to n)
for (j from 0 to n)
dist[i][j] |= (dist[i][k] & dist[k][j]);
```

If you dont initialise dist[i][i]=0; you find the length of the smallest path from i to i (i.e. cycle).

how to check for -ve cycle

```
int ans = leq;
for (i from 0 to n)
ans = min (ans, dist[i][i]);
if (ans < 0) → -ve cycle
```

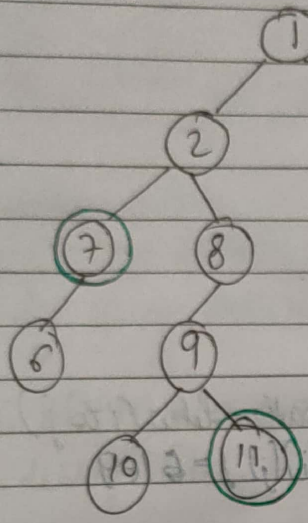
diameter of graph → longest shortest path btw (i to j)

ans = max (ans, dist[i][j]) if dist[i][j] != leq

Binary Lifting

Time Complexity → Building $O(n \log n)$
Query $O(\log n)$
Memory → $O(n \log n)$

LCAO



Step 1 → Bring them to same level
Step 2 → Jump simultaneously till not same

Building

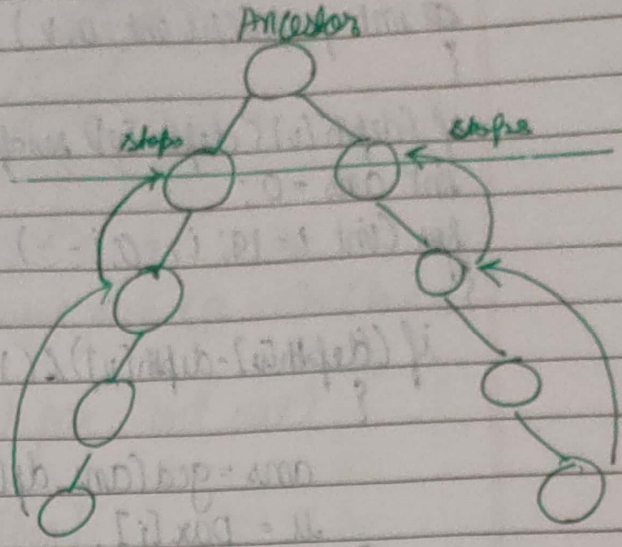
```
void dfs ( int node, int prev, int dep)
{
par[node][0]=prev;
depth[node]=dep;
for (int i=1; i<20; i++)
{
par[node][i]=par[par[node][i-1]][i-1];
}
for (auto v: adj[node]
if ( v!=prev) dfs (v, node, dep+1);
}
```

Query

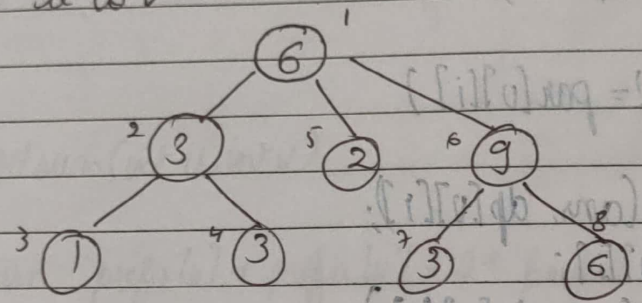
```
int lca (int u, int v)
{
if (depth[u]<depth[v]) swap (u,v);
for (int i=19; i>=0; i--)
if ((depth[u]-depth[v])&(1<<i))
u=par[u][i];
if (u==v) return u;
```

Teacher's Signature

```
for (int i = 19; i >= 0; i--)
{
if (par[v][i] != par[u][i])
{
    v = par[v][i];
    u = par[u][i];
}
}
return par[u][0];
}
```

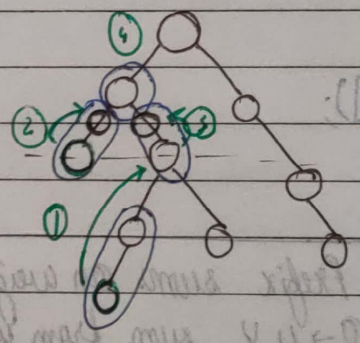


Q Given a tree and Query (u, v) find ~~GCD~~ GCD of the path from u to v.



4 7 → 3
4 5 → 1
5 1 → 2

$dp(x,n) = gcd(dp(x,n-1), dp(next(x,n-1), i-1))$

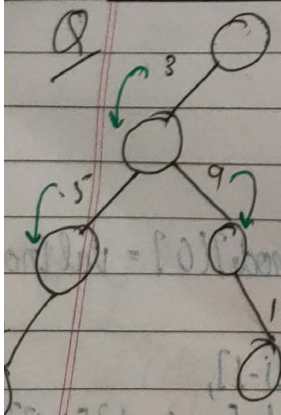


ans = gcd (①, ②, ③, val[lca])

```
void dfs(int node, int prev, int dep)
{
par[node][0] = prev;  depth[node] = dep;  dp[node][0] = val[node];
for (int i = 1; i < 20; i++)
    {   par[node][i] = par[par[node][i-1]][i-1];
        dp[node][i] = gcd(dp[node][i-1], dp[par[node][i-1]][i-1]);}
for (auto v : neigh[node])
{ if(v != prev) dfs(v, node, dep+1); }
}
```

Teacher's Signature

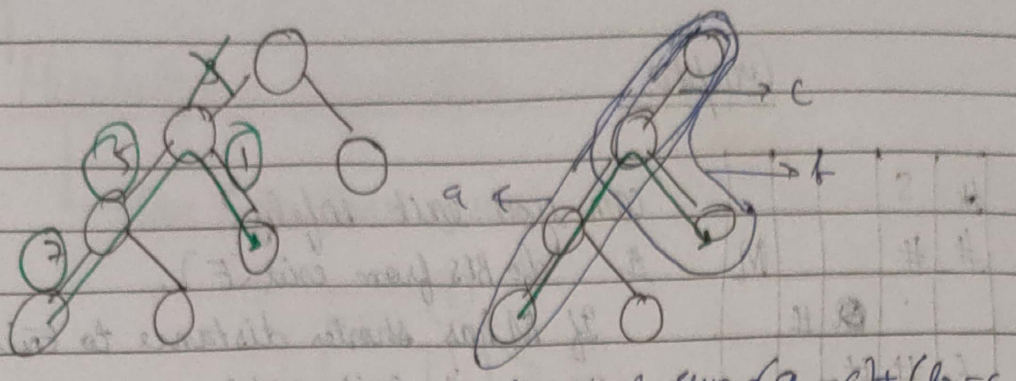
```
int pathGCD(int u, v)
{
if (depth[u] < depth[v]) swap(u,v);
int ans = 0;
for (int i = 19; i >= 0; i--)
{
  if (depth[u] - depth[v] & (1<<i))
    {
      ans = gcd(ans, dp[u][i]);
      u = par[u][i];
    }
}
if (u == v) return gcd(ans, val[u]);
for (int i = 19; i >= 0; i--)
{
  if (par[u][i] != par[v][i])
    {
      ans = gcd(ans, dp[u][i]);
      u = par[u][i];
      ans = gcd(ans, dp[v][i]);
      v = par[v][i];
    }
}
ans = gcd(ans, val[u], val[v]);
return gcd(ans, val[par[u][0]]);
}
```



Q/ Prefix sums on weights
Q → u v sum from u to v

Idea → push the edge weight to lower node

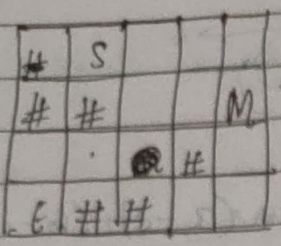
Teacher's Signature



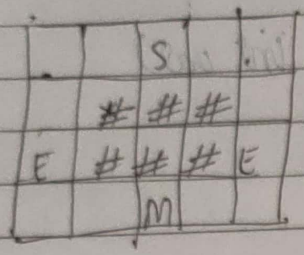
required sum = (a - c) + (b - c)

```
void dfs(int node, int dep, int par, int val)
{
    prefix[node] = val + prefix[par];
    for(auto child: neighbour[node])
    {
        if(v.ff != par) dfs(v.ff, dep+1, node, v.ss);
    }
}

int getsum(int u, int v)
{
    return prefix[u] + prefix[v] - 2 * prefix[lca(u,v)];
}
```

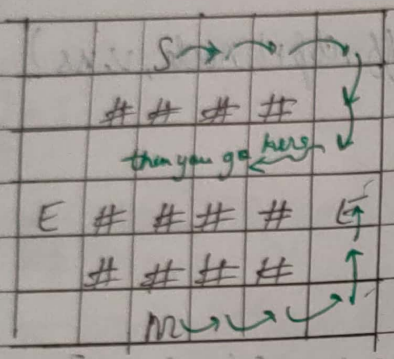
Graphs



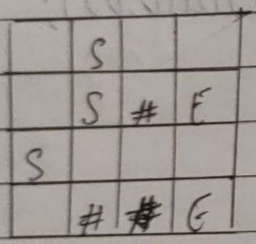
can you exit safely?
→ do BFS from exit (E).
If M has shorter distance to exit you can't exit safely.



but if multiple exits are there?
there should exist an exit where



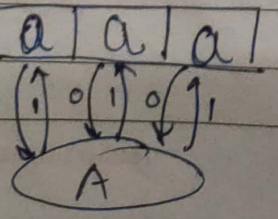
you can trick the monster



find shortest path to exit?
multi-source BFS
→ put all S in Queue then work

Q You start from index 1 and want shortest path to every node $2 <= i <= n$. From an index X in 1 sec, you can go to X+1 and X-1 or to any other index as value A[X]. Find shortest path to end

A N^2 solution would be to create a graph and run BFS on that.



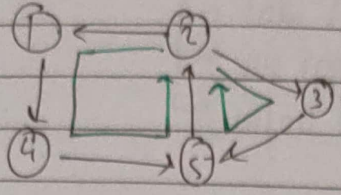
do for all colours
then use 0-1 BFS
O(N)

Teacher's Signature

PAGE NO.
DATE: / / 20

Hierholzer's algorithm

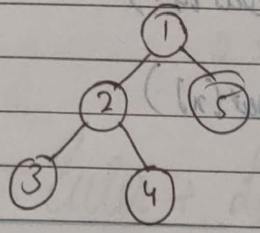
→ For Euler Tour (visit each edge exactly once)



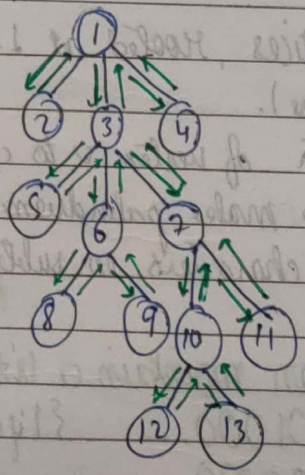
it has eulerian path from 2 to 5

Condition for euler path → ① connected graph
② all nodes have even degree OR (n-2) even & 2 odd degree

Q. Given a tree with n vertices. Rooted at 1
? x y → Print 1 if x is ancestor of y, 2 if y is ancestor of x, -1 if none.
Queries should be in O(1) and preprocessing in O(n).



? 2 3 → 1
? 2 1 → 2
? 3 2 → 2
? 5 3 → -1



Euler tour
→ 1 2 1 3 5 3 6 8 6
9 6 3 7 10 12 10 13 10
7 11 7 3 1 4 1

subtree of 7 starts and ends at 7
size of subtree starting from 7 = 5

```
void dfs (int node, int par=0)
{
euler_pb(node); for (auto child : adj[node])
                { if (child == par) continue;
                  dfs (child, node);
                  euler_pb (node); } }
```

Teacher's Signature

here we dont need the euler tour

```
int in[N], out[N];
int cnt = 0;

//Preprocess  void dfs ( )
{
in[node] = out[node] = cnt++;
for (auto child : adj[node])
{
  if (child == par) continue;
  dfs (child, node);
  out[node] = cnt++;
}
}

//Query  if (in[x] >= in[y] && out[x] <= out[y])
       //x is in subtree of y
       if (in[y] >= in[x] && out[y] <= out[x])
       //y is in subtree of x
```

Subtree Palindrome

Given a rooted tree of n vertices, rooted at 1. Each node has character c (lowercase).

1 v c → Change character of vertex v to c

2 v → Is it possible to make palindrom string from all characters in subtree of v

Note → for each node we can maintain a bitmask

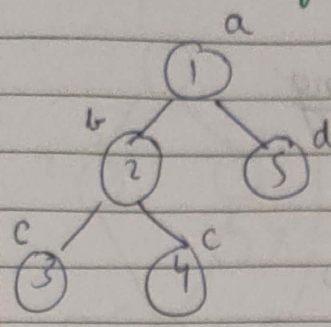
N1 subtree → aba → 01000... {1 if odd occurrence}

N2 subtree → cc → 00000...

∴ N1 ^ N2 → 01000 → only 1 bit can be set for making palindrome

Teacher's Signature

// Query 1



1 2 3 3 4 4 2 5 5 1

↳ make 2nd occurrence 0

1 2 3 0 4 0 0 5 0 0

→ take bit mask of these and XOR

→ we can do updates using segment tree

→ infact we can do Q2 with segment tree

Q Given tree of N vertices. Root of tree is 1.

1 u w → change weight of edge u to w.

2 u w → Find XOR of all path from u to v.

2 u w → XOR till u + XOR till w - 2 XOR till lca(u,w)

{after pushing down the weights}

Q Tree rooted at 1 with N nodes, initially empty.

Q queries → 1 L Y → Increase by Y coins of all nodes which are at distance of L from root.

2 X → Report sum of coins of subtree X

Euler + Lazy / Segment → LATER

Topological Sorting

Kahn's Algorithm

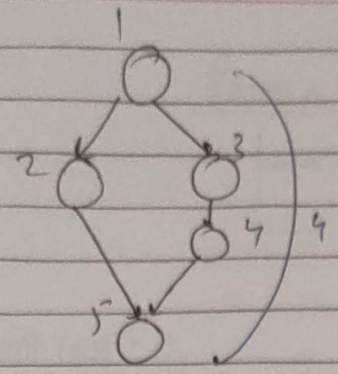
↳ push all nodes with indegree 0
↳ remove and push neighbours with indegree 0 after reducing indegree

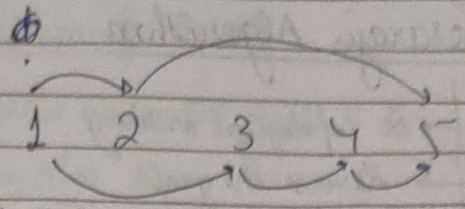
```
void kahn()
{
  queue<int> q;
  for (int i = 1; i <= n; i++)
    if (indegree[i] == 0) q.push(i);

  while(!q.empty())
  {
  int curr = q.front(); q.pop();
  topo.pb(curr);
  for (auto v : g[x])
  {
    indegree[v]--;
    if (indegree[v] <= 0) q.push(v);
  }
  }
  if (topo.size() < n) cycle exists
}
```

* after doing KAHN algo, those nodes which are part of a cycle won't appear in the topo array.

* To get lexicographically smallest T.O. use P.Q.

Q Longest path in DAG





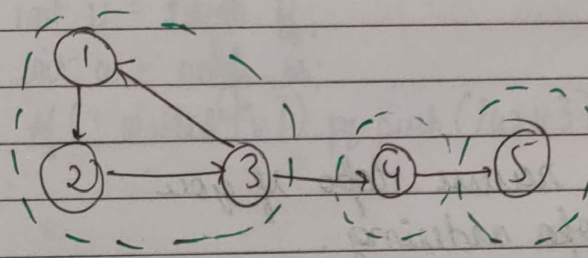
dp[5] = 1
dp[4] = 2
dp[3] = 3
dp[2] = 2
dp[1] = 4

rec(i) $\xrightarrow[x]{\text{neighbour}}$ rec(x) + 1

Strongly Connected for Directed Graphs

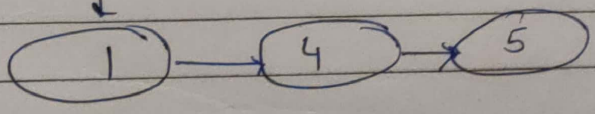
↳ if there is a directed path from any vertex to every other vertex

If there is a strongly connected comp. with atleast 2 vertices, there is a cycle in the directed graph.



3 strongly connected components

Condensed Component Graph - representing a scc as a node



Source node - Node with no incoming edges
Sink node - Node with no outgoing edges

Teacher's Signature

Kosaraju Algorithm

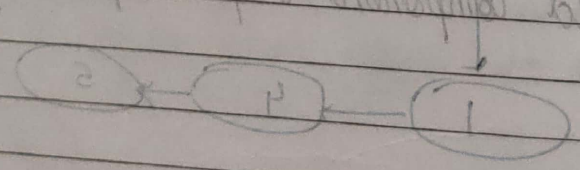
- Do topological ordering
- Reverse the graph
- Do DFS

// topo sort
{ do using ~~Kahn Algo~~ DFS }
// reverse graph and call DFS for all nodes

// call dfs using above adj. list and ordering
Let say you have them in stack S

```
int cnt = 0;
while(!S.empty())
{
  int curr = s.top(); s.pop();
  if(!vis[curr]) dfs(curr), cnt++;
}

return cnt;
```

dont forget to reverse topo if you use DFS for topo ordering.



PAGE NO.
DATE: / /20

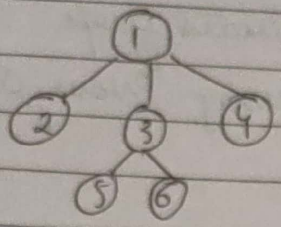
Prim's Algorithm for MST

- Weighted undirected graph
- Greedy Algo using Priority Queue

```
int cost = 0;
minheap pq; pq.push({0,1});

while (!pq.empty())
{
 auto curr = pq.top(); pq.pop();
 int u = curr.ss;
 int weight = curr.ff;
 if (visited[u]) continue;
 visited[u] = true;
 cost += weight;

 for (auto neigh: adj[u])
 {
   int v = neigh.ff;
   int w = neigh.ss;
   if (!visited[v]) pq.push({w,v});
 }
}
```

Euler Tour

Type 1



1 2 1 3 5 3 6 3 1 4 1
(flear rep. of tree)

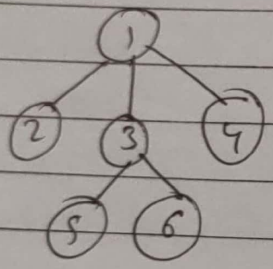
lca(5,4) = map their indexes and find the one having min. level in b/w their path

Tour	1	2	1	3	5	3	6	3	1	4	1
Level	0	1	0	1	2	1	2	1	0	1	0

LCA

We can answer every query in log n using segment tree.

Type 2



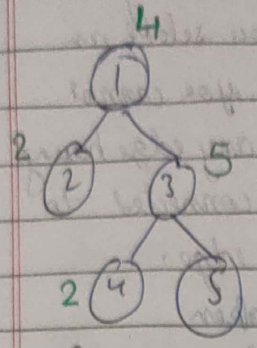
~~1 2 3 5~~
1 2 2 3 5 5 6 6 3 4 4 1
(write once more if you are leaving it)

```
void dfs (int node, int par)
{
mp[node].first = index;
order.pb(node);
index++;
for (auto child: adj[node])
mp[node].second = index;
order.pb(node);
index++;
}
```

At first occurrence we are entering the node
at second occurrence we are leaving the node

PAGE NO.
DATE : / / 20

Q Given tree rooted at 1
1 s x → change value of node s to x
2 s → calculate value of sum in subtree of s



Queries = 2 3
1 5 3
2 3

Sum = 8

Flat tree → 1 2 2 3 4 4 5 5 3 1
Segment → 4 2 0 5 2 0 1 0 0 0

Idea → at first occurrence place its value
at second occurrence place 0, or keep value other/2

For update query → get first index of node then update

Q Given tree rooted at 1
1 s x → change value of node s to x
2 s → sum of values from root to node s

Queries → 2 4
1 3 2
2 1 4

sum = 9

Flat tree → 1 2 2 3 4 4 5 5 3 1
Segment → 4 0 -2 5 2 -2 1 -1 -5 1

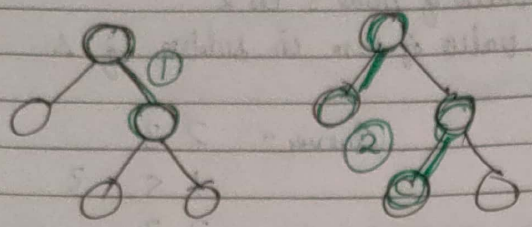
Idea → at first occurrence nothing changes
at second occurrence → - of ~~fir~~ val

For update query → get first index → +Val
get second index → -Val

Teacher's Signature

PAGE NO.
DATE :

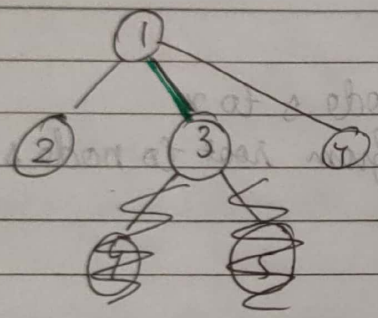
DP on trees

Q 

Tree Matching
→ If you select an edge, you cannot select any edge having a node connected to that edge.

dp[node][flag]
dp[node][0] → that node has not been taken
dp[node][1] → that node has been taken

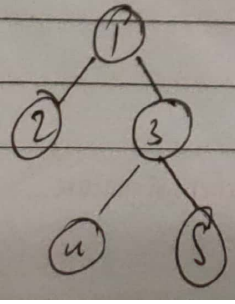
dp[node][0] = max(dp[child][0], dp[child][1])
dp[node][1] = 1 + max(dp[child][0])
dp[node][1] = max(dp[node][1], 1 + dp[child][0] - dp[node][0]
+ max(dp[i+1][0], dp[child][0])



dp[1][0] = max(dp[2][0], dp[2][1])
+ max(dp[3][0], dp[3][1])
+ max(dp[4][0], dp[4][1]);

dp[1][1] = max(dp[1][1], 1 + dp[3][0] + (taking this edge)
they may or may not be taken { max(dp[2][0], dp[2][1])
+ max(dp[4][0], dp[4][1]));

Q Given a tree, find the maximum distance for each node to all other nodes.



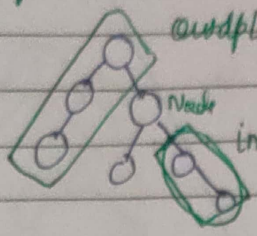
Ans → 2 3 2 3 3

Concept of Rerooting tree

Teacher's Signature

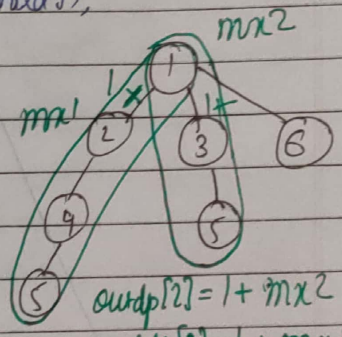
Concept

indp[node] → max depth from that node
outdp[node] → max depth towards parent



longest path may be outdp[node] OR indp[node]

```
int in[N] = {}, out[N] = {};

void dfs1 (int node, int par)
{
for (auto it : adj[node])
{
   if (child == par) continue;
   dfs(child, node);
   in[node] = max (in[node], 1 + in[child]);
}}
```



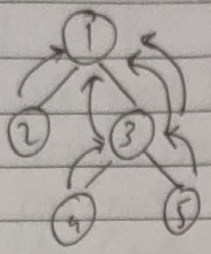
outdp[2] = 1 + mx2
outdp[3] = 1 + mx1

```
void dfs2 (int node, int par)
{
int mx1 = -1, mx2 = -1, temp = 0;
for (auto child : adj[node])
{
 if (child == par) continue;
   [ update mx1, mx2, and don't dfs!
              temp ]
}
for (auto child : adj[node])
{
 if (child == par) continue;
 if (child == temp) outdp[child] = 1 + max (out[node] mx2);
 else   out[child] = 1 + max (out[node], mx1);
 dfs2 (child, node);
}}
```

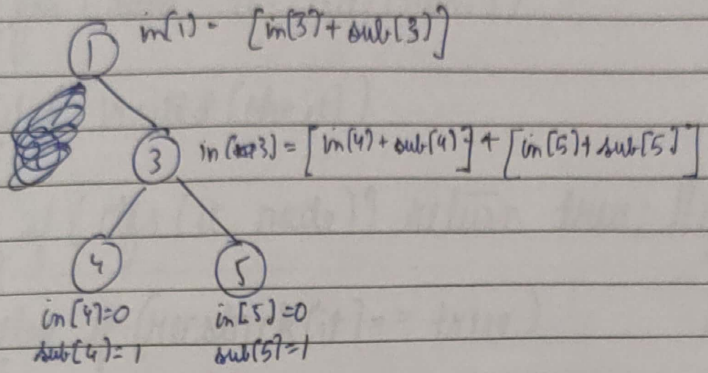
Teacher's Signature

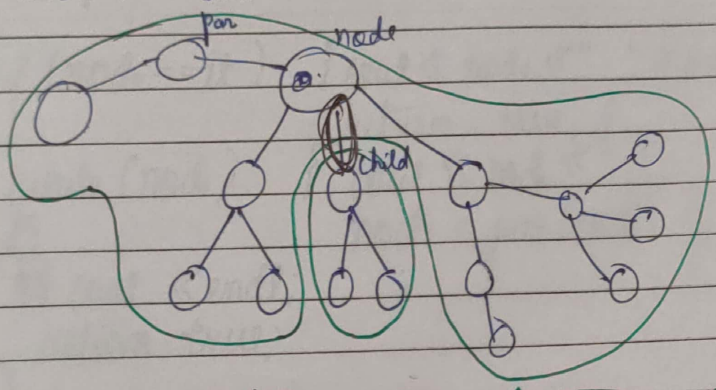
PAGE NO.
DATE: / / 20

Q. Given a tree, determine for each node the sum of distances to all other nodes.



eg. 1 → 6





$dp[child] = in[it] + dp[node] - in[it] - subtree[it]$ (that 1 node being used subtree[i] times)
$+ n - subtree[it]$ (no. of times it is now being used)

$dp[1] = in[1]$ base case

{in dfs2 → first calculate then do dfs}

```
void dfs2 (int node, int par=0)
{
for (auto child: adj[node])
{
  if (child == par) continue;
  dp[child] = dp[node] - 2ll * subtree[child] + n;
  dfs2 (child, node);
}
}
```

Teacher's Signature

PAGE NO.
DATE: / / 20

Detect and print cycle in directed graph

```
bool dfs(int node, int parent)
{
vis[node] = true;
recstack[node] = true;
par[node] = parent;
for(auto it: adj[node])
{
if(!visited[~~ch~~ it])
{
  if(dfs(it, node)) return true; // if cycle found in future
}
else if(recstack[it] == true)
{
if(node == it) { cout << node << " " << node << endl;
                 return true; }
while(node)    { cout << node << " ";
                 node = par[node]; }
  cout << endl;
  return true;
}
}

recstack[node] = false;
return false;
}
```

sub loop

in main ; if not visited send with parent 0

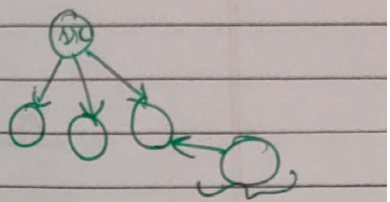
Teacher's Signature

PAGE NO.
DATE: / /20

Single Source Shortest Path on DAG

- Use topological ordering to relax the adj nodes
- Can be used to get longest path by multiplying edges by -1
- Can be used to get longest path in Unweighted DAG

```
vector<int> topo;      //get using dfs & reverse (just put w = -1 for all edges)
vector<int> dist(N, INT_MAX);
dist[src] = 0;
for (int i=0; i<n; i++)
{
  int node = topo[i];
  if (dist[node] == INT_MAX) continue;
  for (auto it : adj[node])
  {
    if (dist[it.ff] > dist[node] + it.ss) { update;
                                            pred[it.ff] = node; }
  }
}
```



Continue done cause of these nodes

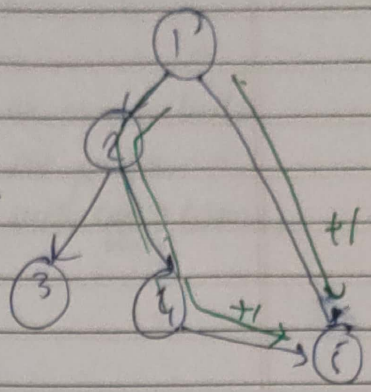
```
// Printing paths
for (int i=1; i<=n; i++)
{
  if (dist[i] == INT_MAX) //Not reachable
  else
  {
    cout << "path to node " << i << ":";
    int node = i;
    while (node != -1) { cout << node << " ";
                         node = pred[node]; }
    cout << endl;
  }
}
```

Teacher's Signature

PAGE NO.
DATE: / /20

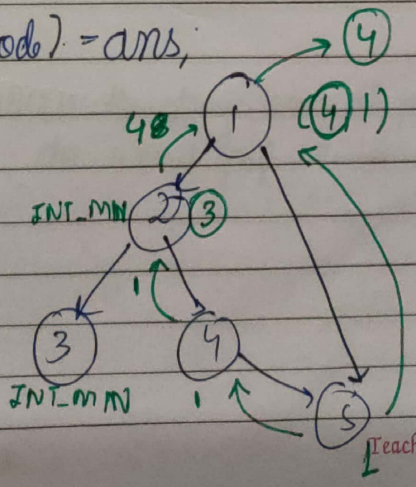
Number of paths from 1 to n in DAG

```
int rec(int node)
{
if (node==n) return 1;
if (dp[node]!=-1) return dp[node];
int ans=0;
for(auto it: adj[node])
{ans += rec(it)%mod;
return dp[node]=ans;
}
```



Longest path in DAG from 1 to n

```
int rec(int node)
{
if (node==n) return 1;
if (adj[node].size()==0) return INT_MIN; //if leaf node!=n
if (dp[node]!=1) return dp[node];

int ans=INT_MIN;
for(auto it: adj[node])
{
 if (rec(it)== INT_MIN) continue;
 if (1+rec(it)>ans) { ans=1+rec(it);
                      succ[node]=it;}
}
return dp[node]=ans;
}
```



Teacher's Signature

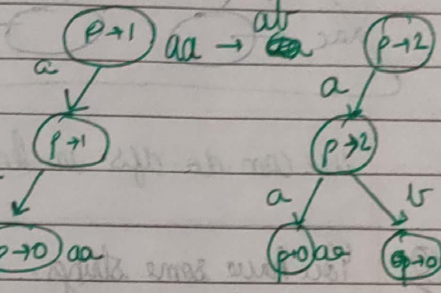
Tries (String)

```
struct node
{
    node *child[26];
    node() { for(int i=0; i<26; i++) child[i] = NULL; prefix = 0; }
};

struct trie
{
    node *t;
    trie() { root = new node; }
};
```

we can include
`int prefix;`
`vector<string> wend;`

Insertion / deletion

```
void insert(string s)
{
    node *cur = root;
    for(int i=0; i<s.length(); i++) {
        int x = s[i] - 'a';
        cur->prefix++;
        if(cur->child[x] == NULL)
            cur->child[x] = new node;
        cur = cur->child[x];
    }
    cur->wend.push_back(s);
}
```



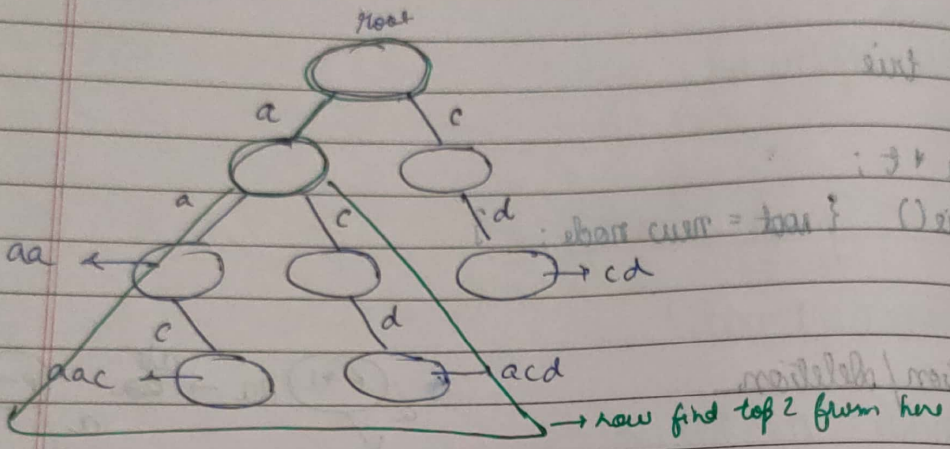
☆ For deletion, traverse the string and do prefix--. When reach the destination, do wend.pop_back()

Teacher's Signature

Q. You have some strings. Answer Q queries which are ?'S' k → find lexicographically smallest top k results for prefix match of 'S'.

eg: aa, aac, acd, cd

S = "a" k = 2 ans aa, aac



we can do dfs in lex first order

Q You have some strings. Answer Q queries ?SK : find all strings with atleast k mismatch and same length.

eg: ab, acd, abd, bcd

S = "acd" ans → acd, abd, bcd

k = 1

we can allow k change from original path

```
void search(node *cur, int pos, int mismatch)
{ if(mismatch > k) return;
  if(pos == s.length()) { for(auto v: cur->vend)
                            result.append(v);
                          return; }
for(int i=0; i<26; i++)
{ if(cur->ch[i] != NULL) search(cur->ch[i], pos+1,
                                mismatch + (s[pos] != (i+'a')));
}
```

Teacher's Signature

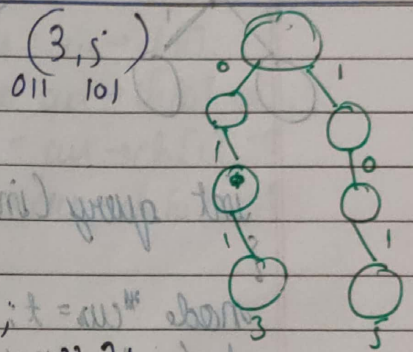
PAGE NO.
DATE: / /20

Bit Tries

→ XOR related maximisation minimisation problems

```
struct node
{
 node *child[2]; int cnt;
 node() { child[0] = NULL;
          child[1] = NULL; }
          cnt = 0;
};

struct trie
{
 node *root;
 trie() { root = new node; }
};
```

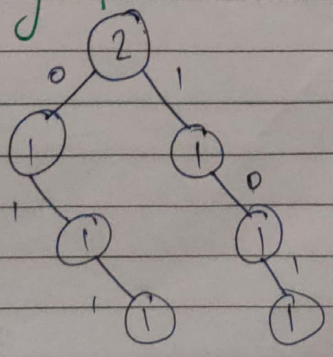
Insertion / Deletion




```
void insert(int n)
{
node *cur = root;
for(int i=LN-1; i>=0; i--) { cur->cnt++;
                             int x = (n&(1<<i))?1:0;
                             if(cur->child[x]==NULL)
                             cur->child[x] = new node;
                             cur = cur->child[x]; }
cur->cnt++;
}
```

☆ In deletion do cur--

While Traversing if come across 0 count node, end.



Teacher's Signature

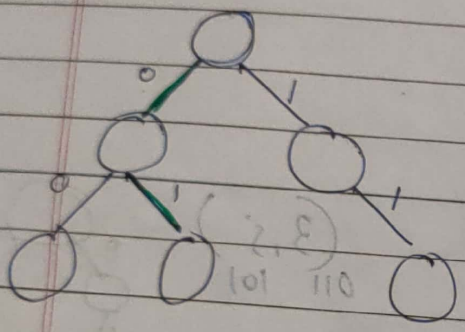
Q. Given an array of N nums, find pairs of indices (i, j) such that i != j and A[i] ^ A[j] is maximum as possible.

Idea → a1 a2 a3 ... ax | ax+1 ... an

a1 a2 a3 ... ax: these elements have been inserted in trie

ax+1 ... an: we get query and maximise

→ Given set of elements in trie, for query x, find y in set s.t. x ^ y is max.

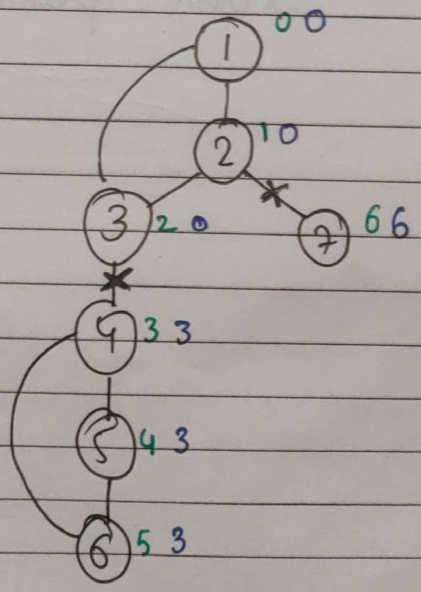


suppose x is 10
y will be 01 for max x ^ y

```
int query(int n)
{
node *cur = t; int ans=0;
for(int i=31;i>=0;i--)
{
int x=(n & (1<<i))?1:0;
if(cur->child[!x]!=NULL){ ans^=(1<<i);
                          cur = cur->child[!x];}
else cur=cur->child[x];
}
return ans;
}
```

Bridges in Graph

```
int in[N], low[N];
int timer = 0;

void dfs (int cur, int par)
{
  if(visited)
  in[cur] = low[cur] = timer++;
  for (auto child : adj[cur])
  {
    if (child == par) continue;
    if (vis[child]) low[cur] = min (low[cur], in[child]);
    else
    {  Here child is bridge
      dfs (child, cur);
      if (low[child] > in[cur])
      {
        // cur → child is bridge
      }
      low[cur] = min (low[cur], low[child]);
    }
  }
}
```



Teacher's Signature

DSU

```
struct DSU
{
  vector<int> rank, par;
  int n;
  DSU(int _n)
  {
     n = _n;
     par.assign(n+1, 0); rank.assign(n+1, 0);
     for(int i=1; i<=n; i++) par[i]=i, rank[i]=0;
  }
  int find(int x)
  {
     if (x == par[x]) return x;
     else return par[x]= find(par[x]);
  }
  int unite(int x, int y)
  {
     int rootx = find(x);
     int rooty = find(y);
     if (rank[rootx] < rank[rooty]) swap(x,y);
     rank[rootx] += rank[rooty];
     par[rooty] = rootx;
  }
};
```

Teacher's Signature