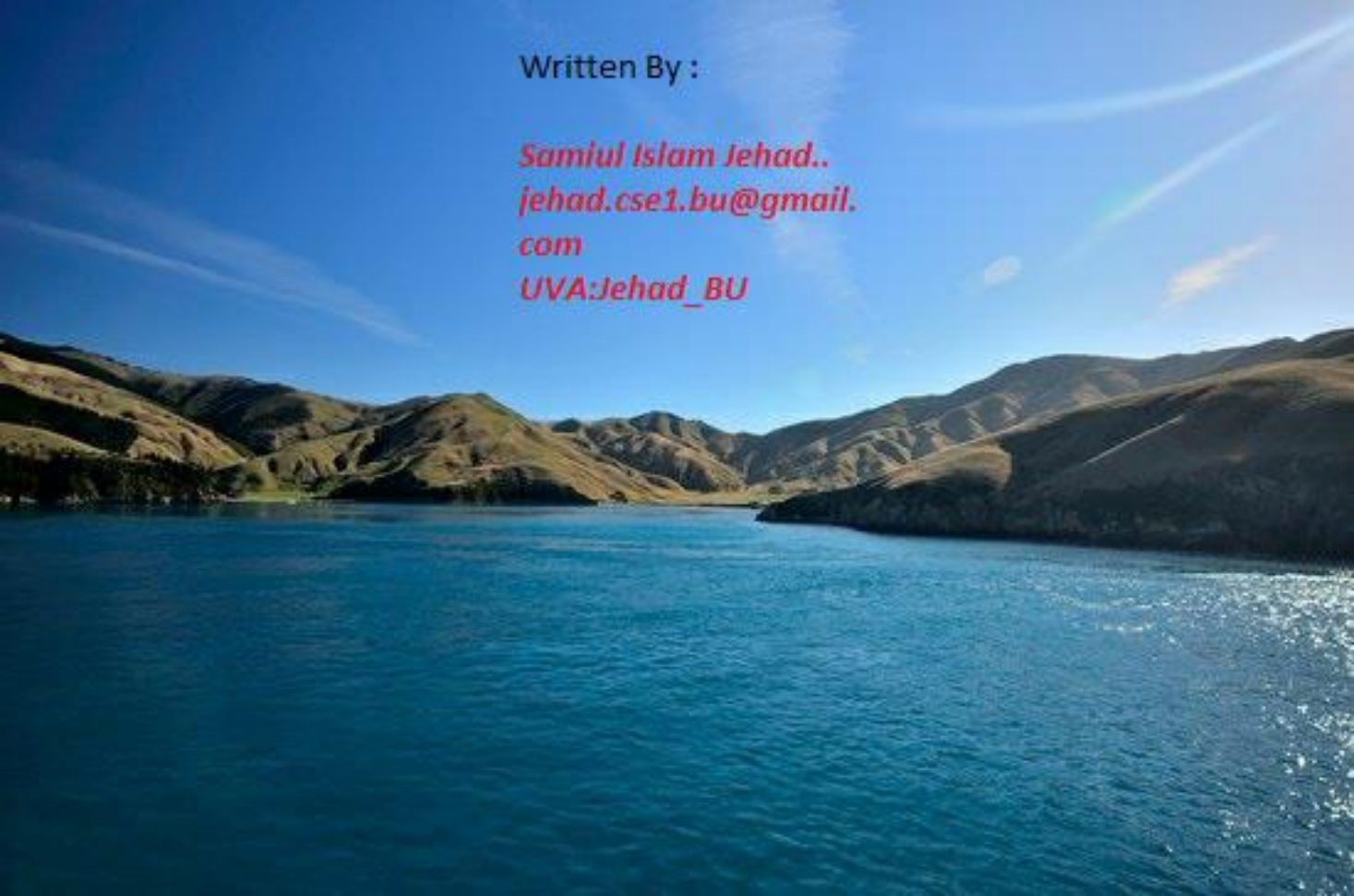
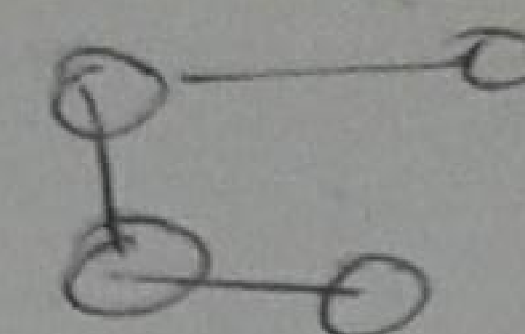


Written By :

Samiul Islam Jehad..
jehad.cse1.bu@gmail.com
UVA:Jehad_BU



① Samiul Islam Jehad.

* Algorithm & code : Minimal Spanning Tree (Prim)

```
#include<bits/stdc++.h>
using namespace std;
typedef pair<int, int> PII;
vector<PII> V[100];
struct Node{
        int u, cost;
        Node(int _u, int _cost)
        {
            u = _u;
            cost = _cost;
        }
};
bool operator < (Node A, Node B){
        return A.cost > B.cost;
}

int cost[100];
int taken[100];
priority_queue<Node> PQ;
```

vector নেয়া হয়েছে এই জন্য pair করে নেয়েছি, কারণ কোন নোডের সাথে কয়টি edge থাকবে এই অনুযায়ী cost থাকবে, আর এর জন্য code করা অনেক সুবিধা। কারণ index number দিয়ে কাজ করতে হবেনা।

* Kruskal & Prim এর মধ্যে মূল Algorithm এর মূল পার্থক্য হল Kruskal একটি graph [illegible] থেকে [illegible] Spanning tree বের করে but prim করেনা।

Samiul Islam Jehad.

```
int prim (int s, int n)
{
    for (int i=1; i<=n; i++)
    {
        cost [i] = INT_MAX;
        taken [i] = 0;
    }
    cost [s] = 0;
    PQ.push (Node (s, 0));
    int ans = 0;
    while (!PQ.empty ()) {
        Node x = PQ.top ();
        if (taken [x.u])
        { continue;
        }
        taken [x.u] = 1;
        ans += x.cost;
        for (PII v : V[x.u])
        {
            if (taken [v.first])
                continue;
            if (v.second < cost [v.first]) {
                PQ.push (Node (v.first, v.second));
```

Ranged for loop → Supported by (C++11)

V[x.u] যদি V[1] হয় তাহলে 1 এর সাথে যে যেসব নোড কানেক্টেড.

```
        const [v.first] = v.second;
      }
    }
}
printf(" cost of MIST : %d ", am);
return am;
}

int main()
{ int n;
  printf(" Enter total number of Nodes:\n");
  scanf("%d", &n);
  for(;;)
  { int x, y, cost;
    scanf("%d %d %d", &x, &y, &cost);
    if(x==-999 && y==-999 && cost==-999)
      break;
    else { V[x].push_back(make_pair(y,cost));
           V[y].push_back(make_pair(x,cost));
         }
  }
prim(1,n);
}
```

Samiul Islam Jehad.

Minimal Spanning Tree (kruskal)

```
#include<bits/stdc++.h>
using namespace std;
struct Edge{
        int u,v,w;
};
bool operator<(Edge A, Edge B)
{
        return A.w<B.w;
}
vector<Edge>E;
int p[100];
int find(int x)
{
        if(p[x]==x)
                return x;
        return p[x]=find(p[x]);
}
int kruskal()
{
        sort(E.begin(),E.end());
        int sz=E.size();
        int ans=0;
```

Jehad

```
for (int i=0; i<nz; i++)                    → Disjoint-Set with
  { if (find[E[i].u] != find(E[i].v))
      {  P[P[E[i].u]] = P[E[i].v];
         ans += E[i].w;
      }
  }
  printf ("%.d\n", ans);
}

int main()
{   for (int i=0 ; ; i++)
    {   int u, v, w;
        scanf ("%.d %.d %.d", &u, &v, &w);
        if (v == -999 && v == -999 && w == -999)
          break;
        else{
        Edge my = {u, v, w} ;
        E.push_back (my);
        P[u] = u;                  > Parent initialize করে
        P[v] = v;                    রাখো।
    }
} Krunkal (); }
```

Shortest Path Dijkstra (Version: Priority Queue)

Samiul

```
#include<bits/stdc++.h>
using namespace std;
struct Node{ int at, cost;
        Node(int _at, int _cost){ at = _at;
                                   cost = _cost;
        }
    };

bool operator <(Node A, Node B){
        return A.cost > B.cost;
}

struct Edge{ int v,w; };
vector<Edge> adj[100];
priority_queue<Node> PQ;
int dist[100];
int n;
void dijkstra(int s){     // int n, int s
        for(int i = 1; i<=n; i++){
            dist[i] = 1000000000;
        }
    dist[s] = 0;
    PQ.push(Node(s,0));
```

Adjacency but linked

* dist array তে রাখা হয় source থেকে x node পর্যন্ত দূরত্ব
main কাজ করা। dist[x] = 10 মানে হল source থেকে min দূরত্ব হল 10.

```
while (!PQ.empty()) {
    Node u = PQ.top();
    PQ.pop();
    if (u.cost != dist[u.at]) {
        continue;
    }
    for (Edge e: adj[u.at]) {
        if (dist[e.v] > u.cost + e.w) {
            dist[e.v] = u.cost + e.w;
            PQ.push (Node(e.v, dist[e.v]));
        }
    }
}
printf ("%d\n", dist[tar]);
}
```

```
int main()
{ int n, s, tar, u, v, w;
  scanf("%d", &n);
  scanf("%d", &s);
  scanf("%d", &tar);
  for(int i=0; ; i++)
    {
      scanf("%d %d %d", &u, &v, &w);
      if(u==-999 && v==-999 && w==-999)
        { break;
        }
      Edge tem = {v, w};
      Edge tem2 = {u, w};
      adj[u].push_back(tem);
      adj[v].push_back(tem2);
    }

  dijkstra(n, s, tar);

}
```

No of Nodes
Source (Starting point)
from source to target.

2-D array structure

"Longest Increasing Subsequence" ⑩

by Samiul Islam Jehad

Example → 1, 7, 12, 19, 20

Way-1 (Complexity $O(n^2)$).

input-array ৪, 1, ৭, 8, 3, 4, 6, 1, 5, 2.

calculation-array L

* input-array এর same size এর আরেকটি array নিতে হবে যা ঐ sequence এর জন্য largest বা longest sequence নির্ধারণ করবে।

প্রথমে ৪ নিয়ে কাজ শুরু,

	৪	1	৭	8	3	4	6	1	5	2
L	1	1	2	1	1	1	1	1	1	2

৪ এর থেকে কেবল ৭ বড়। তাই ৭ এর ঘরে এক increase করা হল।

Then, 1

	৪	1	৭	8	3	4	6	1	5	2
L	1	1	2	2	2	2	2	1	2	2

1 এর জন্য যেসব সংখ্যা বড় সেগুলোর L বৃদ্ধি করা হল।

৭, ৪ এর জন্য কোন change হবে না। ৩ ও কে নিয়ে

	৪	1	৭	8	3	4	6	1	5	2
L	1	1	2	2	2	3	3	1	3	2

Then, 4

	8	1	9	8	3	4	6	1	5	2
L	1	1	2	2	2	3	4	1	4	2

যেহেতু 4 এর L = 3 এর মানে হল এই 4 এর আগে নিশ্চই এর থেকে তিনটা ছোট value রয়েছে। তাই 4 এর পরবর্তী বড় সংখ্যাগুলোতে কেবল L of 4 এর সাথে 1 করে যোগ হচ্ছে। যদি এর পরবর্তী কোন সংখ্যা থাকে যার L এর মান 3 এর থেকে বেশি তাহলে ঐ সংখ্যাটির L এর value বৃদ্ধি করার দরকার নেই কেননা ঐ সংখ্যাটি 4 বাদে অন্য সংখ্যাগুলো দ্বারা গঠিত যে seq আছে সেটা বেশি বড়।

যেমন: 1, 85, 86, 4, 87

এই ক্ষেত্রে 87 তে 4 এর কোন প্রভাব নেই কারণ এই 4 বাদ দিলেই বড় seq হয়।

\# যদি List print করতে চাই সেক্ষেত্রে উল্টা দিক থেকে শুরু করব traversal. প্রথম দেখব কোনটার value max এর সাথে মিলে ঐটা নিব। 6 Then দেখব কোনটা ঐ max এর value থেকে 1 ছোট এবং ঐ সংখ্যার থেকে ছোট ঐটা নিব।

Algorithm: L → 0 to n = 1;

```
for (i → 0 to (n-1))
    for (j = i+1 to (n-1)) {
        if ( sequence [j] > sequence [i]) {
            if (L[j] < L[i]+1)
                L[j] = L[i]+1;
        }
    }
```

[illegible]

Reconstructing:

```
top = L[i] - 1        // total array size যাতে List রাখা হবে।
List [top] = sequence [i];    // এরপর থেকে শুরু করবো
top --;
for ( j ← i-1; j>=0; j--) {
    if ( sequence [j] < sequence [i] && L[j] == L[i]-1) {
        i = j;
        List [top] = sequence [i];
        top--;
    }
}
```

(13)

lower_bound এর কাজ হল যে Number টা দিব ঐ Num থেকে বেশি অথবা সমান, তার সর্বপ্রথম index বের করা। এটা ঐ same value পেলেও return করবে।

Way 2 → Complexity (n log k)

This will produce only size.

```
int main() {
        vector<int> V;
        int n, num;
        scanf("%d", &n);
        for(int i=0; i<n; i++){
            scanf("%d", &num);
            vector<int>:: iterator iV =
    lower_bound(V.begin(), V.end(), num);
    if(V.end() == iV) ①* V.push_back(num);
    else V[iV-V.begin()] = num; *②
        }
    LIS = V.size();
}
```

Binary Search এর মাধ্যমে lower bound বের করা হয়। [illegible]

যদি Last এ একটি value কে Add করতে পারি তাহলে List এ value টা array এর n তম value এর জন্য LIS বের করে save করে রাখলাম, এবং List এর ঐ value এর original index save করে রাখলাম।

① * L[s] = V.size();
indexed[v.size()-1] = n;

② * L[s] = L[indexed[iV-V.begin()]];

↳ যেহেতু Replace করছি K তম value কে তাই তার LIS value যত ছিল তাই এই n তম value এর জন্য update করলাম।

way-2 তে আসলে যা হয়,

2, 4, 8, 12, 15 → inserting 10 (↗ 12)

সবথেকে Lowest বড় সংখ্যাকে replace → 2, 4, 8, 10, 15.

যদি insert করি 18 তাহলে,

2, 4, 8, 10, 15, 18.

ব্যাখ্যা: এই Code এর মূল কাজ হল দৈর্ঘ্য বের করা। তাই যদি list এ যা আছে তার থেকে বড় সংখ্যা পাই তাহলে একদম শেষে যোগ করি। অন্যথায় Replace করি। Replace যে কারণে করি সেটা হল, যদি এমন হয় যে বিশেষ ছোট সংখ্যা ~~...~~ অনেক পাই তাহলে longest seq হতে বের করা Possible হয়। যেমন, 2, 8, 18, 20

এখন, যদি upcoming input → 9, 12, 13, 15, 16 ইনপুট হয় আর যদি Replace না করতাম তাহলে upcoming 5টি সংখ্যার seq ছাড়া সম্ভব হত না।

যদি list আকারে প্রকাশ করতে চাই সেক্ষেত্রে LIS নামক array এর মধ্যে যখন রাখব তে ঐ পর্যন্ত অর্থাৎ ঐ element পর্যন্ত কতগুলা বড় seq সম্ভব সেটা। এর জন্য index স্মরণ রাখতে হয়। index তার স্মরণ রাখব যখন নতুন value শেষে add হয়। ~~LIS~~] indexed [v.size()-1]=n; অর্থাৎ n seq এ রাখা ঐ সংখ্যাটির প্রকৃত array তে অবস্থান n। আর যখন value Replace করব তখন কি হবে? → যাকে Replace করেছি।

index use করতে হয়। L[n] = L[indexed[iV - V.begin()]];

মানে হল যেহেতু n তম value কে seq এর কোন old value দ্বারা replace করছি তাই ঐ old value যত তম seq আছে সেটাই হবে এই new ~~...~~ n এর।

"Longest Common Subsequence"

by Jehad

Longest common subsequence হল সবসময়ই দুটি String এর মধ্যে common alphabet নিয়ে সবচেয়ে বড় যে sequence পাওয়া যায়,

যেমন:

a b c d e f g
a k b c d t s > এখানে LCS = a b c d

Solving Technique: Dynamic & 2D-array

Algorithm:

```
if (str1[i] == str2[j])
{
    seq[i][j] = seq[i-1][j-1] + 1;
}
else
{
    seq[i][j] = max(seq[i-1][j],
seq[i][j-1]);
}
```

a b c d a F V_S a c b c F

i \ j	.	a	b	c	d	a	F
.	0	0	0	0	0	0	0
a	0	1	1	1	1	1	1
c	0	1	1	2	2	2	2
b	0	1	2	2	2	2	2
c	0	1	2	3	3	3	3
F	0	1	2	3	3	3	4

a, b, c, F

"Segment Tree"

Samiul Islam Jehad.

* It helps to do range query in a array.

যেমনঃ একটা array দেয়া আছে অনুরূপ বল্ল 4 to 7 এই index এর মধ্যে সর্বোচ্চ value কত, বা বড় value বা যোগফল etc.

এটা Onek fast. যদি normally ad-hoc approach এ করা হত তাহলে complexity হত — O(mn). যদি Query বেশি হয় অনেকবার। m = total query এবং n size of array.

* Square matrix করেও করা যেতে পারে সেক্ষেত্রেও $O(n^2)$ এবং Space ও n^2 পরিমাণ। যা বড় value এর জন্য অগ্রহণযোগ্য।

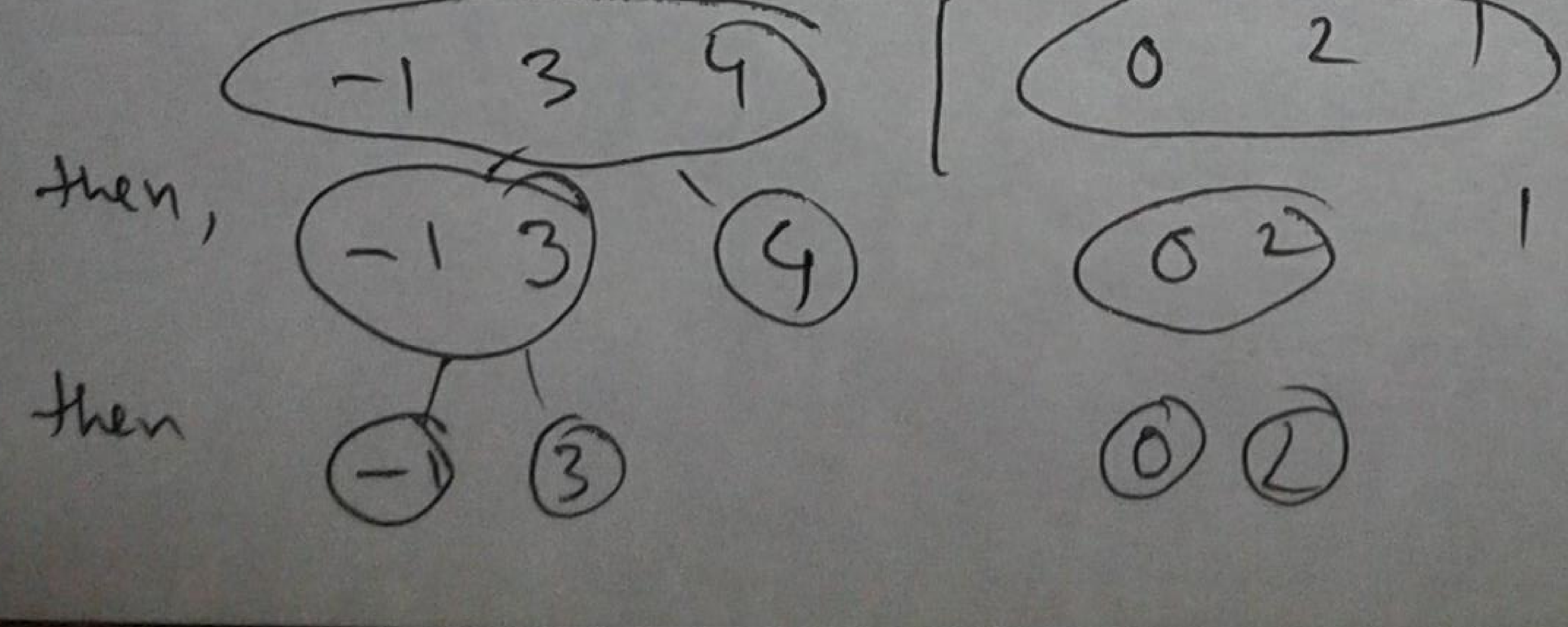
Segment tree is binary tree & element are leaf.

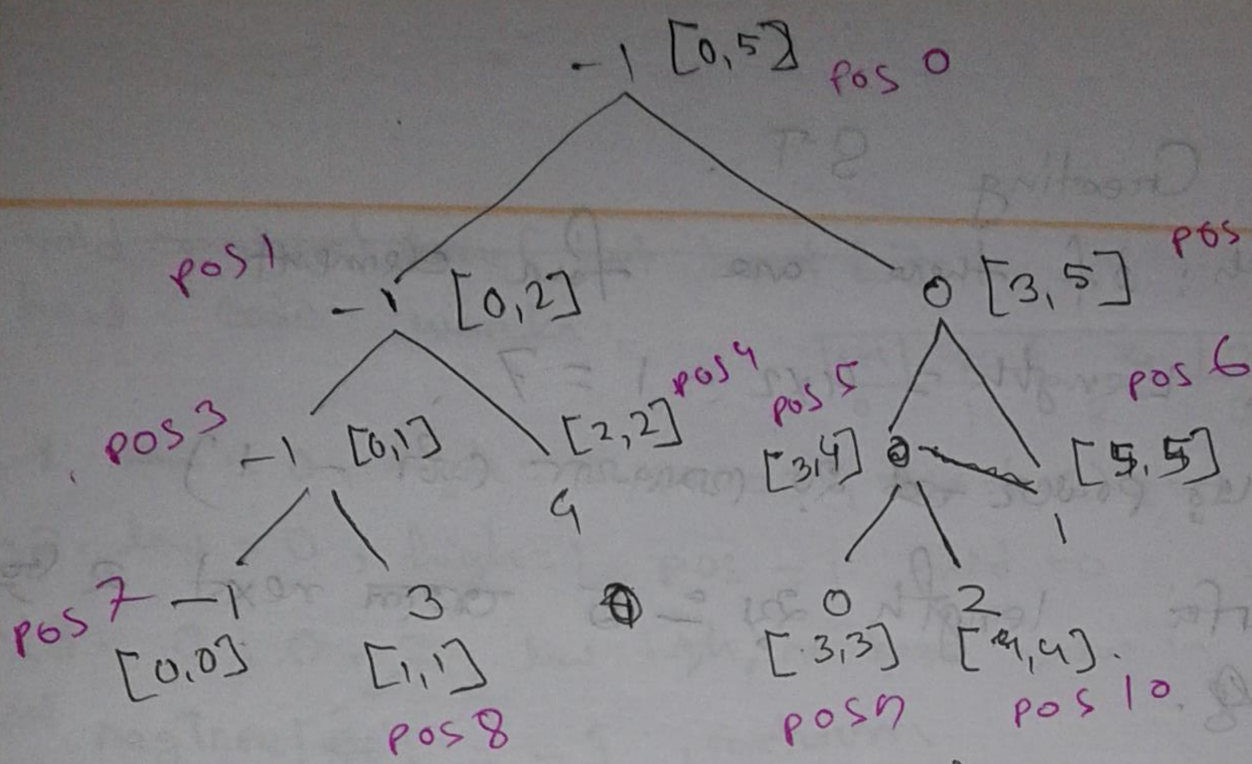
For example:

-1	3	4	0	2	1

First এ leaf গুলো বসিয়ে উপরের দিকে আসব,

ধরি, rang এর ভেতরের ~~max~~ min value বের করব — প্রথমে দুই ভাগ করি,

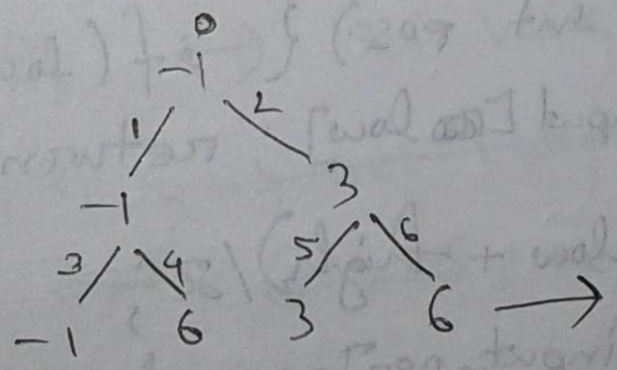




* How range query works? ঃ? Rang [2,4]

প্রথমে root থেকে [0,5] তে যাবে partial over lap
So, left এ যাবে [0,2] তে গেলে PO So আবার left এ NO এর back to previous [0,2] now right [2,2] totally overlap So, back এস 4.
Now go to right right, [3,5] PO এর left ~~totally~~ TO so return value 0. Now [5,5] NO
So, answer is 4,0 and min = 0.

* Constructing a tree of value: -1 0 3 6



Creating ST.

* length: if there are four element

∴ Length = 4×2 − 1 = 7.

(2 এর power এর 2 গুণ থেকে − 1)

যেমন: যদি length হয় = 5 তাহলে next 2 এর power 8.

∴ length = 8*2 − 1 = 15.

For child: left → 2i+1 (Index from 0)

Right → 2i+2

For Parent: (i−1)/2.

So, total size for worst case : 4n.

How to construct:

```
* void construct_tree(int input[], int seg_tree[], int
low, int high, int pos) { if(low == high) {
segTree[pos] = input[low]; return; }
int mid = (low + high)/2;
construct_tree(input, segTree, low, mid, 2*pos+1);      ← line-1   (margin: Right Child)
construct_tree(input, segTree, mid+1, high, 2*pos+2);   ← line-2   (margin: Left child)
segTree[pos] = min(segTree[2*pos+1], segTree[2*pos+2]); (margin: Parent)
}
}
```

~~void construct_tree(int input[]~~.

how code works:

-1	2	4	0
0	1	2	3

1. low = 0, high = 3, pos = 0, mid = (0+3)/2 = 1, line 1
2. low = 0, high = 1, pos = 1, mid = 0, line = 1

X3. 0, 0, 3, low = high, so keep value in array of SegTree
completed segTree[3] = −1, return,

3. 0, 1, 1, 0, 2

X4. 1 1, 4 So, set 2 into segtree at pos = 4 → segTree[4] = 2;

এর and also set segTree[pos] = min of righ and left child. এখানকার pos হবে right step 2 এর pos এর value. কারণ ঐ pos এর value দিয়েই এখানে কাজের right ও left child এ ঐ pos এর উপর depend করবে? value বসানো হয়েছে segtree[1] = −1.

2. 0, 3, 0, 1, 2

3. 2, 3, 2, 2, 1

X4. 2, 2, 5, → set value → seg_tree[5] = 4;

4. 2, 3, 2, 2, 2 ~~→ set value → seg_tree[6] = 0~~;

5. 3, 3, ~~2, 2~~ 6 → seg_tree[6] = 0;

seg_tree[2] = 0;

seg_tree[0] = −1;

So, SegTree → −1, −1, 0, −1, 2, 4, 0

At first কেবল Recursion এ সম্পূর্ণ tree তৈরি হয়, last এ value [illegible] হবে।

Overlap করবে
হবে যদি index
তোমার Query
এর ভিতর থাকে।

* Range-Query Search:

Total overlap ↑

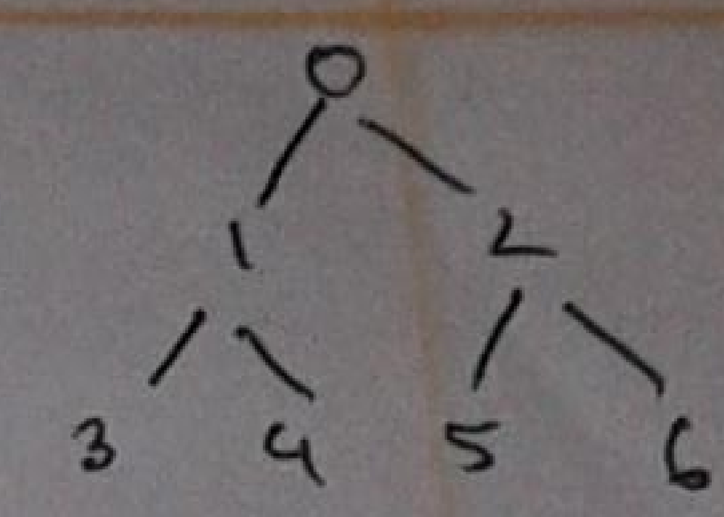
```
int rangeMinQuery (int segTree[], int ~~query~~ qlow,
int q high , int low, int high , int pos) {
if ( qlow <=low && qhigh >= high) {
        return segTree [pos]; }
if (qlow > high || qhigh < low)
        return Max. value;
int mid = (low + high)/2;
return min( rangeMinQuery ( segTree, qlow, qhigh,
low, mid , 2*pos+1 ), rangeMinQuery ( segTree, qlow,
qhigh , mid+1 , high , 2 *pos +2));
}
```

No Overlap

11235
11297
11402

Binary-tree - -1, 2, 4, 0

Segment-tree → -1 -1 0 -1 2 4 0
0 1 2 3 4 5 6



query 1-3

qlow	qhigh	low	high	pos	mid	lineNumber
1	3	0	3	0	1	1, (recursion)
1	3	1	1	1	0	1
1	3	0	0	→ max to calling funct.		
1	3	1	1	4 → return value - 4		

No Overlap

Otherside

qlow	qhigh	low	high	pos	
1	3	2	3	2	return value → 0.

* প্রথমে Query এর input 1-3. আর মূল array তে 0-3 তাই এটা partial overlap. তাই এর child এর দিকে যাবে হবে। left এ → 1,3 send করব mid = (0+3)/2 = 1 হবে. এর মানে হল left child এ index 0,1 (মূল array এর)

Floyd Warshel (All pair shortest Path).

by Samiul Islam Jehad

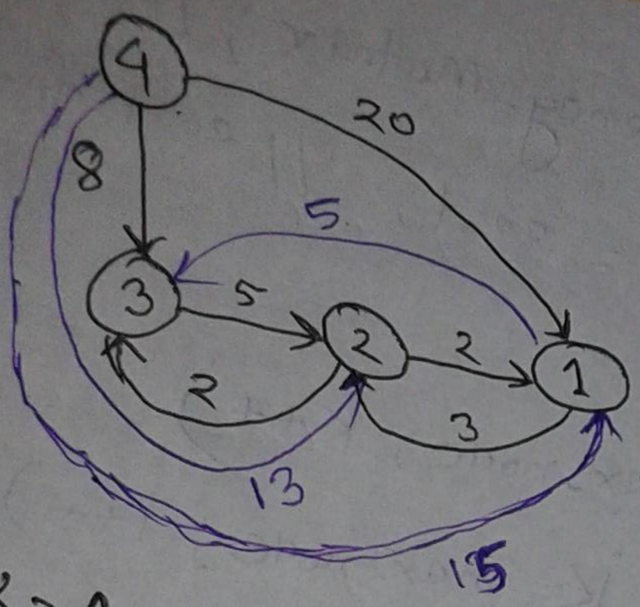
Algorithm:

```
create_adjacency_Matrix; // with cost.
matrix[u][v] = cost;  // cost from u to v.

for(int K=1; K<=max_node; K++)
    for(int i=1; i<=max_node; i++)
        for(int j=1; j<=max_node; j++)
            { if(matrix[i][j]>matrix[i][k]
+ matrix[k][j])
            {
                matrix[i][j] = matrix[i][k] + matrix[k][j];
            }
}
```

If current cost is greater than the cost form $i \to k$ then $k \to j$ the replace the current cost. we are going from i to j via k.

* matrix[i][i] = 0; out initially all are infinity except diagonal.



Node	1	2	3	4
1	0	3	inf	inf
2	2	0	2	inf
3	inf	5	0	inf
4	20	inf	8	0

K = 1 ,

i = 1, j = 1 No change.
i = 1 , j = 2 No change.
i = 1 , j = 3 No change.
i = 1 , j = 4 No change.

---

i = 2 , j = 1 No Chang.
i = 2 , j = 2 N/A.
i = 2, j = 3 N/A.
i = 2 , j = 4 N/A.

---

i = 3 , j = 1 N/A.
i = 3, j = 2 N/A.
i = 3 , j = 3 N/A.
i = 3 , j = 4 N/A.

---

i = 4 , j = 1 N/A.
i = 4 , j = 2 , Matrix [4] [2] = 20+3=23.
i = 4 , j = 3 , N/A

∴ K=2, i=1, j=3 → Matrix [1][3] = 3+2=5.

i=3, j=1 → Matrix [3][1] = 7

∴ K=3, i=4, j=1 → Matrix [4][1] = 8+5+2=15.

i=4, j=2 → Matrix [4][2] = 13.

∴ K=4 No changes.

Node	1	2	3	4
1	0	3	5	inf
2	2	0	2	inf
3	7	5	0	inf
4	15	13	8	0

* 2টি total minimum cost অর্থাৎ এক Node থেকে অন্য কোন থেকে ১ টি cost বের করতে বলে যদি বলে যে কোন x নোড থেকে Y নোডে গেল অথবা সেই পথ এ edge গুলোর মধ্যে অবথেকে ৬ বেশি cost কত; তাহলে ৭ গুলোর মধ্যে,

matrix[i][j] = min(matrix[i][j], max(matrix[i][K], matrix[K][j]);

আর যদি অবথেকে ছোট value কত তাহলে,

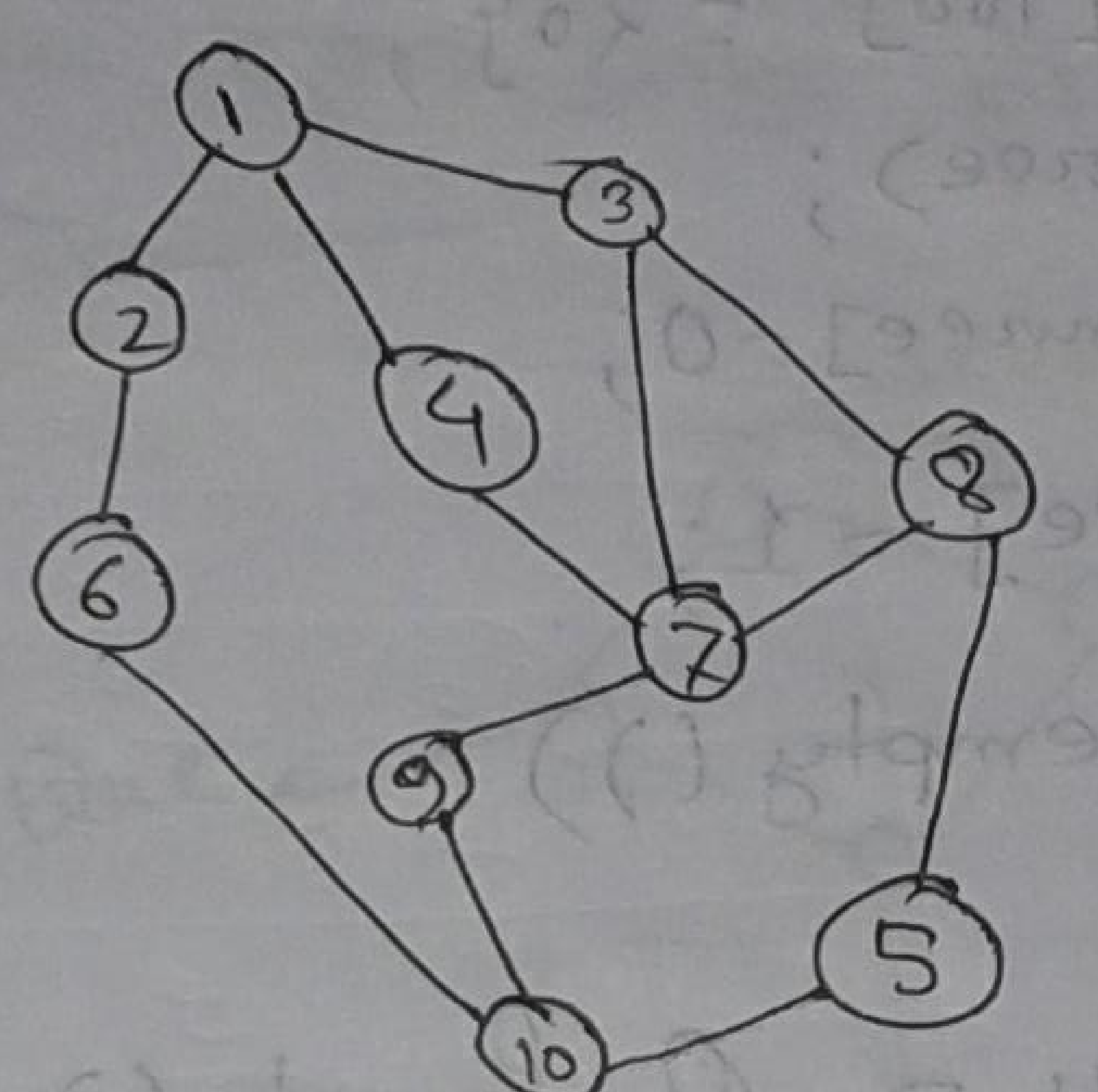
* যদি বলে কোন কোন cost এর পথটাই যাচ্ছে হবে. তাহলে matrix[i][j] = -1 দ্বারা initialized করতে হবে। আর যদি বলে Path এর মধ্যে যে edge টার cost সবচেয়ে কম সেই পথটার value বের করতে হবে, then,

matrix[i][j] = max(matrix[i][j], min(matrix[i][k], matrix[k][j]));

BFS (Breadth First Search)

Samiul Islam (Jehad)

* BFS হল Source node থেকে শুরু করে এর neighbouring সব নোডে ছড়িয়ে পড়ে। এভাবে আগাতে থাকবে।



Source : 1

1 থেকে যাওয়া যায় 2, 3, 4 এ.

So, প্রথমে 2 এ গেলাম। 2 এ দূরত্ব 1.

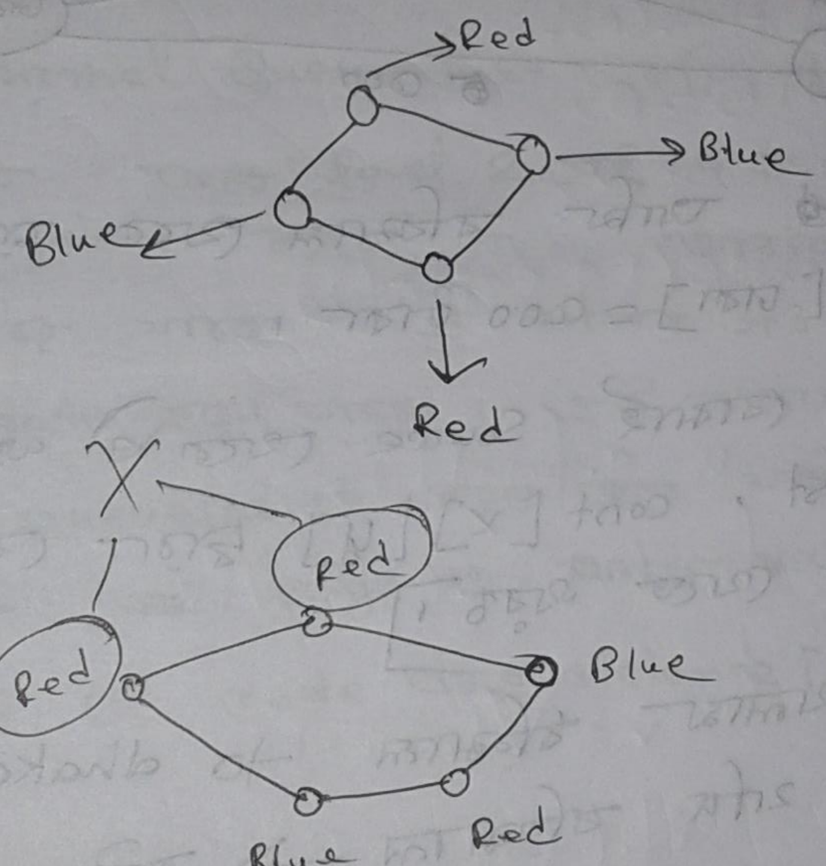
distance[2] = distance[u] + 1. (যেখান থেকে এই নোডে আসল।)

এবং node এ যে visit হল তা Specify করা হবে আর কখন কোথায় যাচ্ছে তা একটা Queue এর মাধ্যমে Save করে রাখব। Queue হল First in First out. তাই 1 থেকে 2, 3, 4 এ যাওয়া যাবে তাই এগুলো Q তে রেখে দিব। অতঃপর 2, 3, 4 এ Q থেকে নিয়ে visit করব এবং pop করব।

```
void bfs (int source)
{
    queue <int> Q;
    int distance [100] = {0};   → max_number of node (node ka identity)
    int taken [100] = {0};
    Q. push (source);
    distance [source] = 0;
    taken [source] = 1;
    while (! Q. empty ())
    {
        int u = Q. front ();
        for (int i = 0; i < G[u]. size (); i++)
        {
            int v = G[u][i];
            if (! taken [v])
            {
                taken [v] = 1;
                Q. push (v);
                distance [v] = distance [u] + 1;
            }
        }
        Q. pop ();
    }
}
```

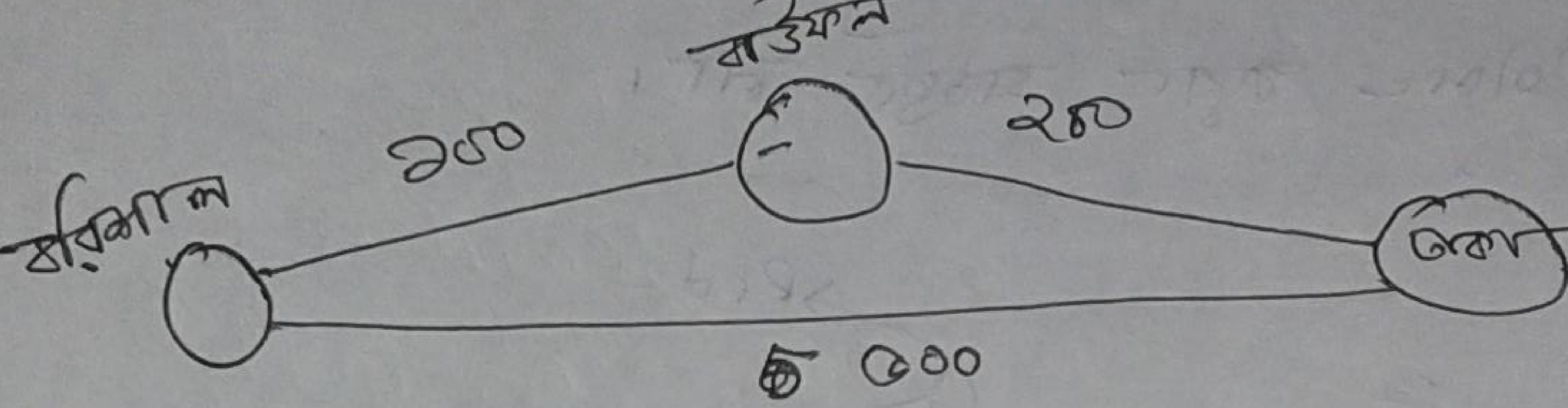
* B.F.S দ্বারা Check করা যায় কোন Graph কে দুটো Color দ্বারা ✗ Colored করা যাবে কিনা।

~~Negative cycle~~ বিজোড় সংখ্যক কোনো cycle থাকলে সেটাকে অর্থাৎ ঐ Graph কে দুটো-Color দ্বারা Color করা সম্ভব না।



Dijkstra Algorithm (Part - 2)

* BFS সাধারণত weighted graph এ কাজ করেনা, তাই কোন Node থেকে অন্য কোন নোডে যাওয়ার জন্য এই Algorithm টা use করা হয়।



So, আমি বরিশাল থেকে ঢাকা যেতে চাই। d[ঢাকা] = 500 টাকা।

[d দ্বারা বোঝাই Source থেকে ঐ খানে যেতে lowest cost. cost[x][y] দ্বারা বোঝায় x থেকে y তে যেতে খরচ।]

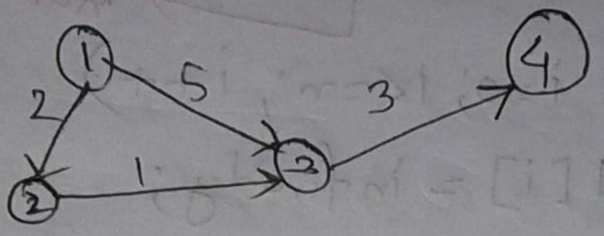
এখন দেখলাম বরিশাল to dhaka সরাসরি না গিয়ে যদি বরিশাল → বাউফল → ঢাকা যাই তাহলে খরচ কমবে (700 টাকা)।

```
if (d[u] + cost[u][v] < d[v])
{
    d[v] = d[u] + cost[u][v];
}
```

বাকিটা B.F.S এর মতন।

এই Process কে আরও fast করা যাবে।

যেমন:



যেমন: যদি আগে আসলে আগে পাবে ভিত্তিতে অর্থাৎ normal Queue দিয়ে করি then 1 থেকে 3 কে যাবে তার level 2 হবে 4 কে গেলে cost 5+3 এ update করবে. then আবার যখন 1→2 node ধরে যাবে তখন আবার 3+3=6 update. দুইবার update হয়ে যাচ্ছে। এই সমস্যা সমাধানের জন্য priority-queue(ডেক) use করতে হবে যাতে যত থেকে ছোট weight এর edge এ আগে visit করে। আর P.Q তে node এর distance [u] ও মুখ্য করতে হবে।

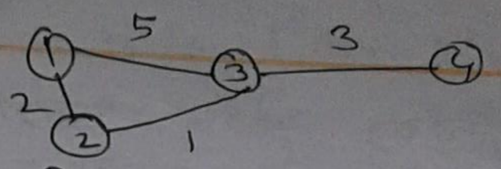
Code: Dijkstra / with Ordinary Queue

Source

```
void dijkstra(int s, int n, int des)
{
    for (int i=0; i<=n; i++)
        d[i] = infinity;
    queue<int> q;
    q.push(s);
    d[s] = 0;
    int u, v;
    while(!q.empty())
    {
        u = q.front();
        for(int i=0; i<V[u].size(); i++)
        {
            v = V[u][i];
            if((d[u]+Cost[u][v]) < d[v])
            {
                d[v] = d[u] + Cost[u][v];
                q.push(v);
            }
        }
        q.pop();
    }
printf("Shortest distance is: %d", d[des]);
}
```

des → where to get distance
n → number of Nodes (max node)

কিন্তু কেন হয়?



চিত্র:- ১

কারণ, যদি উপরের গ্রাফটি 1 থেকে first ৫ 3 ৫ গেল, then ও ৩ 3 রাখবে, আবার 2 রাখবে, 1 বাদ দিবে। 1 থেকে 3 ৫ যাওয়ার main মান হল d[3] = ~~∞~~ infinity যা d[4] + cost[4][3] এর থেকে বড়। এজন্য হয়। আবার এজন্য 1-3-4 থেকে- d[4] = 8 বসিয়ে রাখে।

একবার d[4] ও update করার পরেও আবার d[4] ৫ ৪ যাবে কারণ যদি এবং কেবল যদি 3 পর্যন্ত আগের ৫ cost ৫ আসল তার থেকে কম cost এর কোন way নাই। যেহেতু 2→3 ৫ cost total 3 আর আগে ছিল 5 তাই পরবার আবার Queue ৫ 3 রাখবে যেন বড় 3 থেকে আবার নতুন করে কিছু পাওয়া যায়।

Priority_Queue কেন?

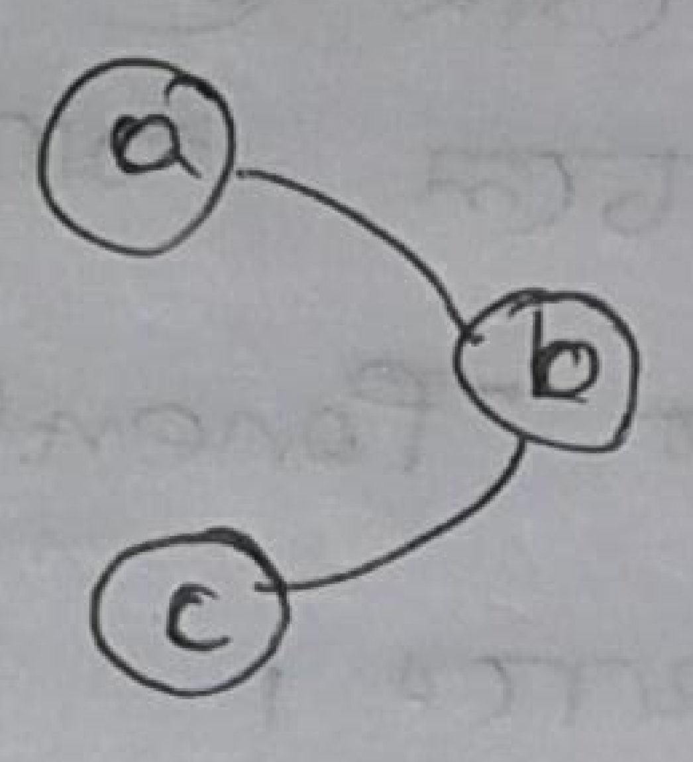
Priority_Queue ৫ কম খরচের মানগুলো আগে visit হয় যায়। তাই ৫ 3 পর্যন্ত আসার ৫ কম খরচের path গুলো already connect হয়ে যাবে। আর Priority_queue এর main সুফল হল- বড় গ্রাফগুলোর ক্ষেত্রে, একটা path বারবার visit করে লাগবেনা।

Disjoint Set

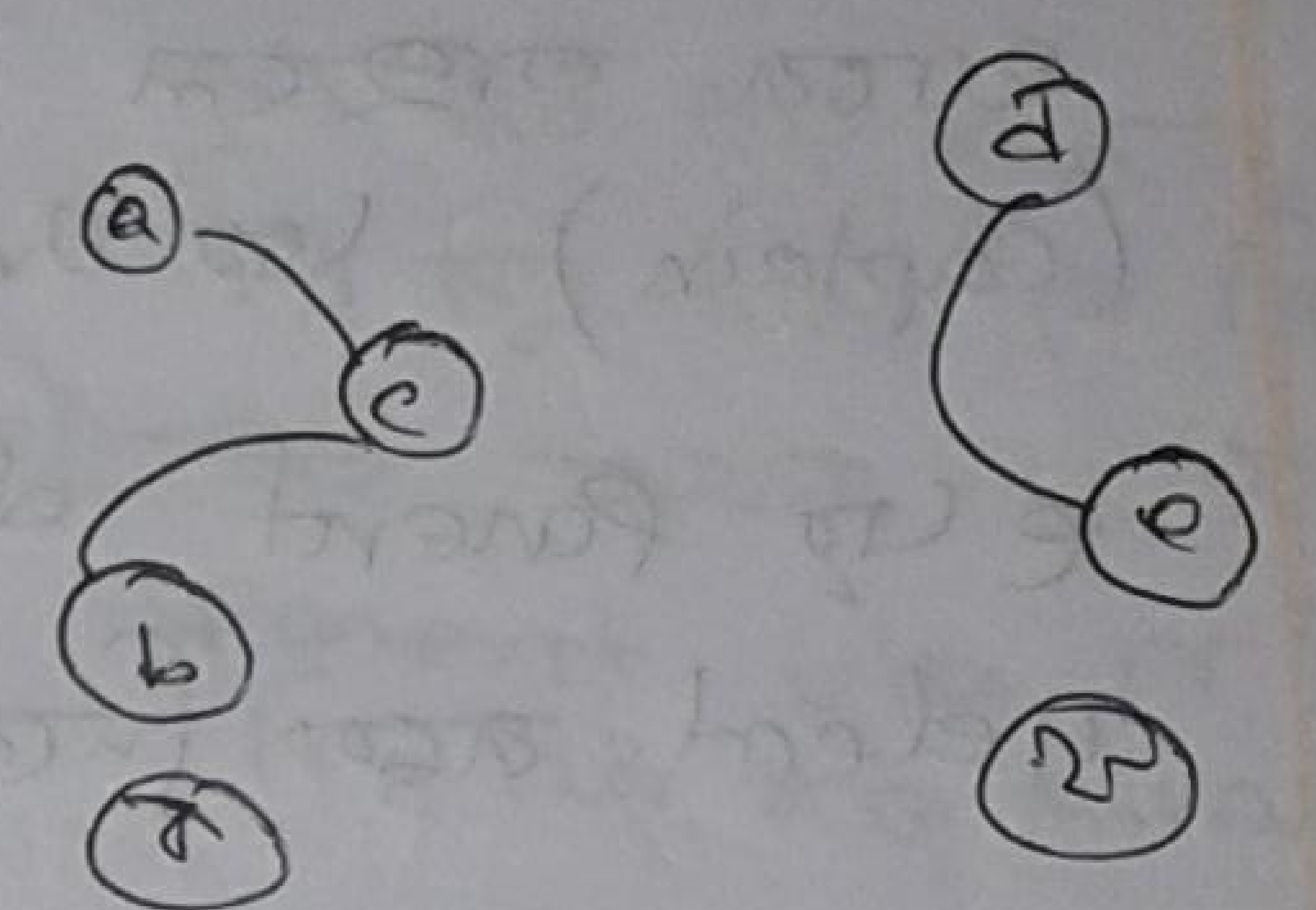
by Samiul Islam Jehad

*** একাধিক সেট যাদের মধ্যে কোন কমন উপাদান নাই তাকে Disjoint Set বলে। A = {1}, B = {2, 5, 7}, C = {3, 6}

যেমন: A, B এর বন্ধু B, C এর বন্ধু অর্থাৎ A ও C এর বন্ধু।



বা.



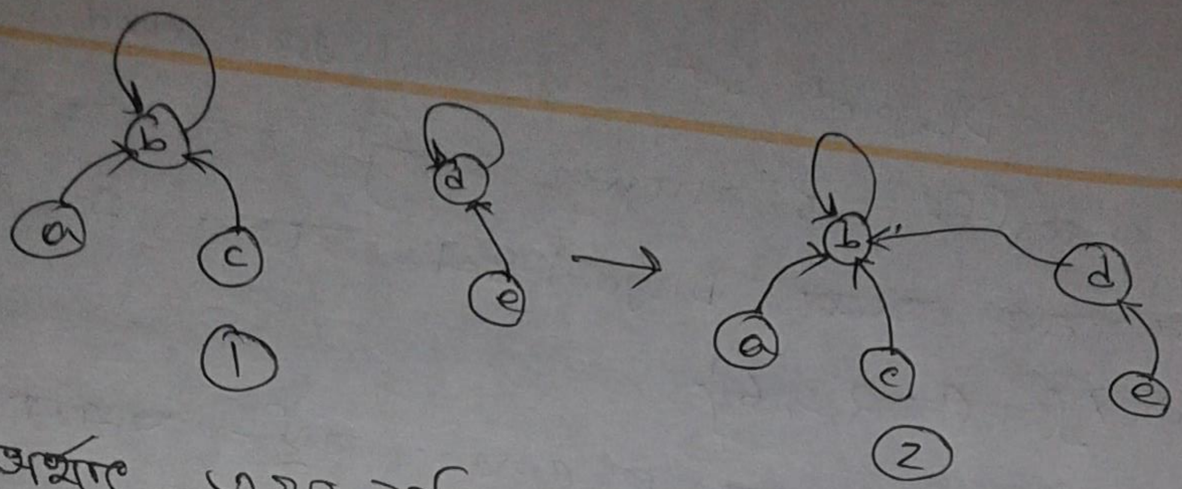
আবার, d ও e এর বন্ধু।

~~এখন যদি~~

আর অনেক গুলো সেটের মধ্যে connection create এর জন্য Parent অ্যারে ও Representative function এর দরকার পড়ে।

a → b → c এর মধ্যে যদি এমন হয় c এর বন্ধু e তাহলে (ক) ও (খ) নং চিত্র মিলে একটি Tree তৈরী হবে, যখন e কে b এর সাথে Connect করা হবে আর যদি- parent হয় a তাহলে a কে করতে হবে যেটি হল e এর parent এর সাথে Connect করে দেয়।।

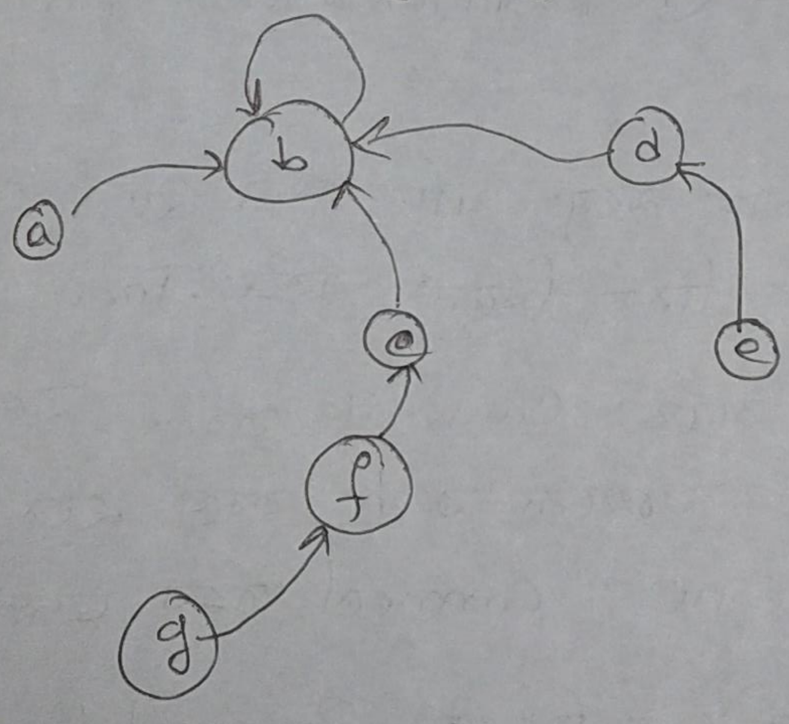
আর মূল যে তিনি যে নিজেই Representative + নিজেরই Parent.



অর্থাৎ এখন যদি e এর সাথে c এর Relation থাকে তাহলে e ও সেই b এর Representative (Captain) এর Under এ চলে আসবে। সেজন্য e এর Parent d হবে এবং Parent হিসাবে b কে select করে দিলেই হয়ে যাবে।

সেই Representative যে নিজেই নিজের Parent.

```
parent[r] = r;
```



এখন g এর representative কত বের করি।

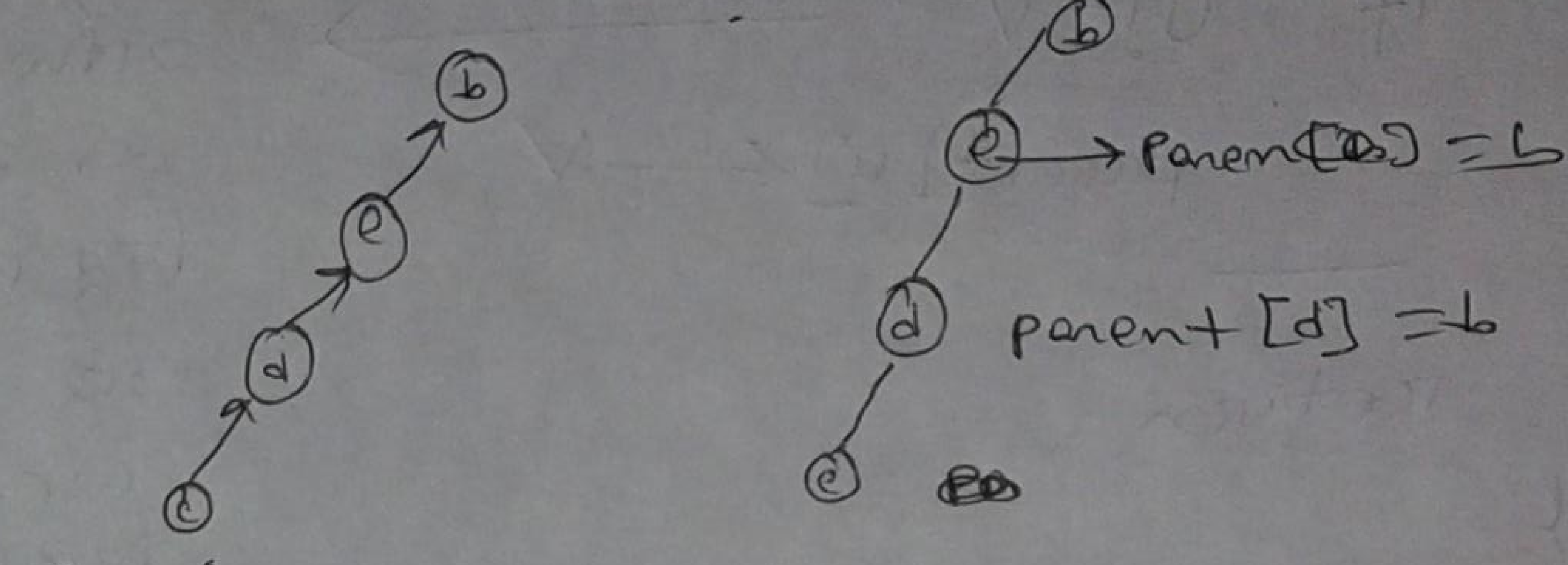
g এর Parent f, f এর parent c, c এর Parent b এবং b এর parent b.

অর্থাৎ parent[b] = b; Representative কোড চলমান। :D

```
procedure Union (a, b):{        // a ও b কে যুক্ত হলে
    u = FindRepresentative(a)   → a এর Rep বের করলাম
    v = FindRepresentative(b)   → b এর Rep বের করলাম
    if   u != v                 → যদি u এবং v সমান না হয় তার মানে তারা
                                   ভিন্ন ভিন্ন tree তে ছিল। তাই এখন u এর
                                   parent v এর যে parent ছিল সেটা করে দিলাম।
        parent[u] ← v
    ___
    return
}

procedure FindRepresentative (r):
{ if   parent[r] = r            → যদি parent = নিজেই হয় তার মানে
                                   Representative পেয়ে গেলাম।
        return r
  end if
  return FindRepresentative (parent [r])   → অন্যথায় parent এর parent
                                              recursion এর মাধ্যমে বের করি।
}
```

* Previous findRepresentative function এর কমপ্লেক্সিটি ছিল $O(n)$. n = depth of the tree. যেখানে দেখা যাচ্ছে child থেকে একদম Root পর্যন্ত সব কয়টি node কে পরিদর্শন করতে হত।

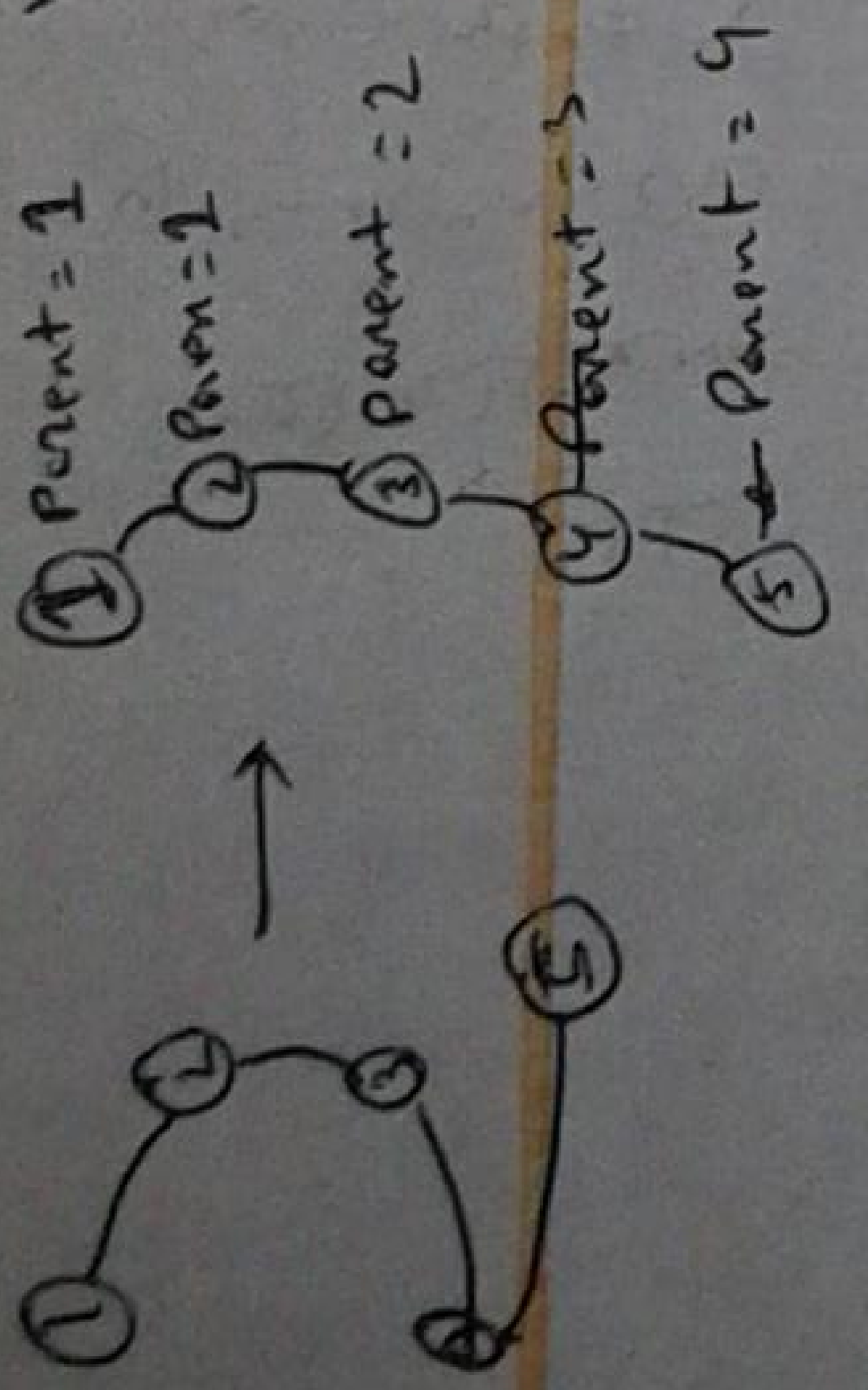


এখন যদি ঐ function টিকে একবার call করে যদি প্রতিটি child এর parent একদম Root কে করে দেয়া যায় তাহলে complexity কমে।

```
procedure findRepresentative(r) {
    if (parent[r] = r)
        return r
    end if
    parent[r] ← findRepresentative(parent[r])
    return parent[r]
}
```

Recursion শেষ হলে উল্টা দিকে আসার সময় প্রত্যেক Root node কে assign হতে থাকে পুরো পথ জুড়ে।

Final Rec: 4 → 3 → 2 → 1 Stoped then → parent[2] = 1 → parent[3] = 1 → parent[4] = 1 → parent[5] = 1.



উপরের সহজ ছিল কঠিন হলো (Path Compression).
Full code:

```
#include <bits/stdc++.h>
using namespace std;
int parent[1000];
int findRepresentative(int a)
{   if(parent[a]==a)
        return a;
    else
        findRepresentative(parent[a]);
}

void Union(int a, int b)
{   int u = findRepresentative(a);
    int v = findRepresentative(b);
    if(u!=v)
        parent[a]=v;
    return;
}
void makeset(int u)
{   parent[u]=u;
    return;
}
void init_disjoint_set(int n)
{   for(int i=0; i<n; i++)
    {   makeset(i);
    }
    return;
}
int main()
{   init_disjoint_set(999);
    cout<<"Enter numer of pair of Friend"<<endl;
    int n;
    cin>>n;
    int a,b;
```

```
for (int i=0; i<n; i++)
    {   cin>>a>>b;
        Union(a,b);
    }
for(int i=0; i<n; i++)
    {   cout<<parent[i]<<endl;
    }
}
```

Topological Sort.

Samiul Islam Jehad.

** কোন একটা কাজ একাধিক কাজের উপর Depent করে, তখন ঐ কাজ অনুসারে Sort করা হয় তাকে topological Sort বলে.

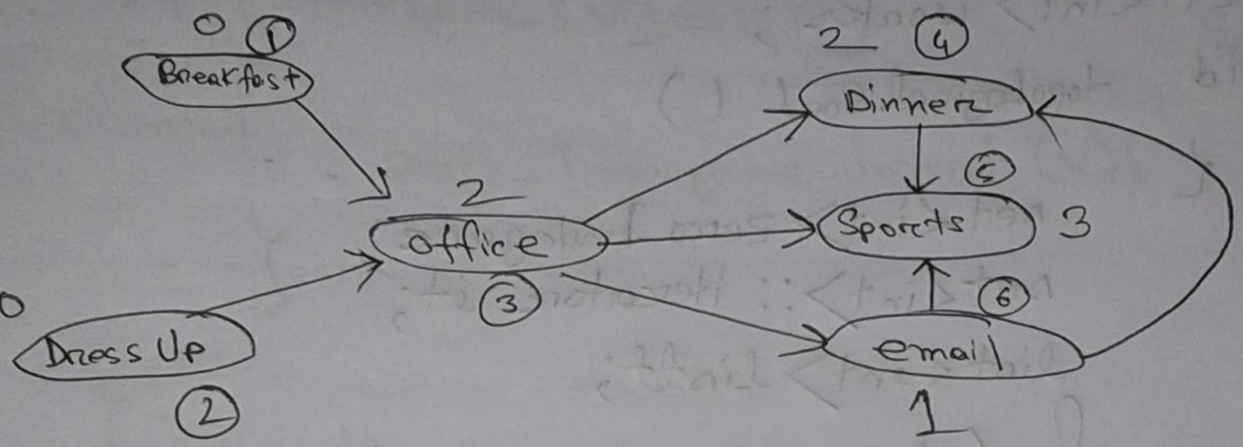


fig: 1 ① → ② → ③ → ⑥ → ④ → ⑤

এ চিত্র থেকে বোঝা যায় office এ যাওয়ার 2টা কাজের উপর depend করে ① breakfast ② Dress up. তাহলে, তাই office এর indegree = 2. Indegree হল কতগুলো edge প্রবেশ করল.

এই Sort টা computer এর জন্য খুবই গুরুত্বপূর্ণ.

```
#include <bits/stdc++.h>
map<int, int> indegee;
vector<int> G[100];
set <int> works;
void topological_Sort ()
{
    set<int> zero_Indegree;
    set<int>:: iterator it;
    list<int> listt;
    for (it = works.begin(); it!=works.end(); it++)
    {   if(indegree[*it] == 0)
        {
            zero_Indegree.insert(*it);
        }
    }
int u,v;
while (!zero_Indegee.empty())
{
    it = zero_Indegree.begin();
    u = *it;
    listt.push_back(u);
    zero_Indegree.erase(it);
    for(int i=0; i<G[u].size(); i++)
    {   v=G[u][i];
```

```
        if (indegree[v] != 0)
            indegree[v]--;
        if (indegree[v] == 0)
        {
            zero.insert(v);
        }
    }
}
list<int>::iterator itt;
for(itt = listt.begin(); itt != listt.end(); itt++)
    {   cout << "-->" << *it;
    }
}
int main()
{   cout << "First Input independent & Sec dependent"
    << endl;
    int independent, dependent;
    while(cin>> independent >> dependent)
    {   G[independent].push_back(dependent);
        indegree[dependent]++;
```

```
        works.insert(independent);
        works.insert(dependent);
    }

    topologicalSort();
}
```