ANSWER TO THE QUESTION NUMBER 1

->

BFS

ADVANTAGE

i) BFS খুব সহজেই unweighted graph এ single source shortest path খুঁজে বের করতে পারে।

ii) এর প্রোগ্রাম বাস্তবায়ন করা খুব সহজ।

iii) এটি সব সময় সর্বাপেক্ষা উত্তম path খুঁজে বের করতে পারে

DISADVANTAGE

i) আমরা যখন bfs নিয়ে কাজ করি এখন কোন node এর adjacency node গুলোকে আমরা queue তে push করি। এবং node গুলো visited হওয়ার আগ মুহূর্ত পর্যন্ত queue তে stored করা থাকে। আর কোন গ্রাফের লেভেল সংখ্যা বাড়ার সাথে সাথে queue তে স্টোরকৃত adjacency node গুলোর পরিমাণও অনেক বেশি হারে বাড়তে থাকে। তাই আমরা যখন বড় কোন গ্রাফ নিয়ে কাজ করব এবং কাজটি যদি আমাদের লিমিটেড মেমোরির মধ্যে করতে হয় তাহলে bfs দিয়ে আমাদের অনেক অসুবিধা হবে।

DFS

ADVANTAGE

i) অপেক্ষাকৃত কম মেমোরি নিয়ে কাজ করে।

ii) বিশেষ কিছু ক্ষেত্রে dfs bfs অপেক্ষা অনেক দ্রুত কাজ করতে পারে। কারণ আমরা জানি bfs কে পরবর্তী লেভেলে যেতে হলে বর্তমান লেভেলের সবগুলো node কে explore করতে হয়। কিন্তু অপরদিকে dfs কোন একটি node কে পেলে এক্সপ্লোর করতে করতে একদম গভীরে চলে যায় dfs এর এই বৈশিষ্ট্যের কারণে অনেক সময় খুবই দ্রুত ফলাফল পাওয়া যায়।

iii) dfs সাহায্যে আমরা Dag এ টোপোলজিকাল সর্টিং করতে পারি।

DISADVANTAGE

i)যদি এমন হয় source node থেকে destination node এ একাধিক ভাবে যাওয়া যায়। তাহলে সব থেকে কম দূরত্বের পথটি dfs নাও দেখাতে পারে।

ii) এটি একাধিক গ্রহণযোগ্য ফলাফল প্রদর্শন করতে পারে না। অর্থাৎ এমন যদি হয় source to destination

অনেকগুলো রাস্তা থাকে তবে এটি শুধু একটি দেখাতে পারবে।

ANSWER TO THE QUESTION NUMBER 2

->

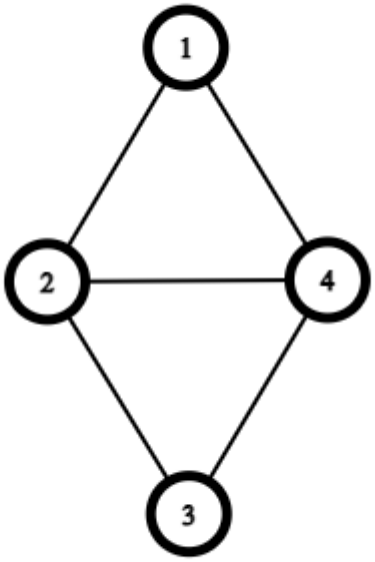
কতগুলো graph representation পদ্ধতি হল :

i) adjacency list

ii) adjacency matrix

iii) edge lis

নিচে এদের উদাহরণসহ বর্ণনা দেওয়া হল



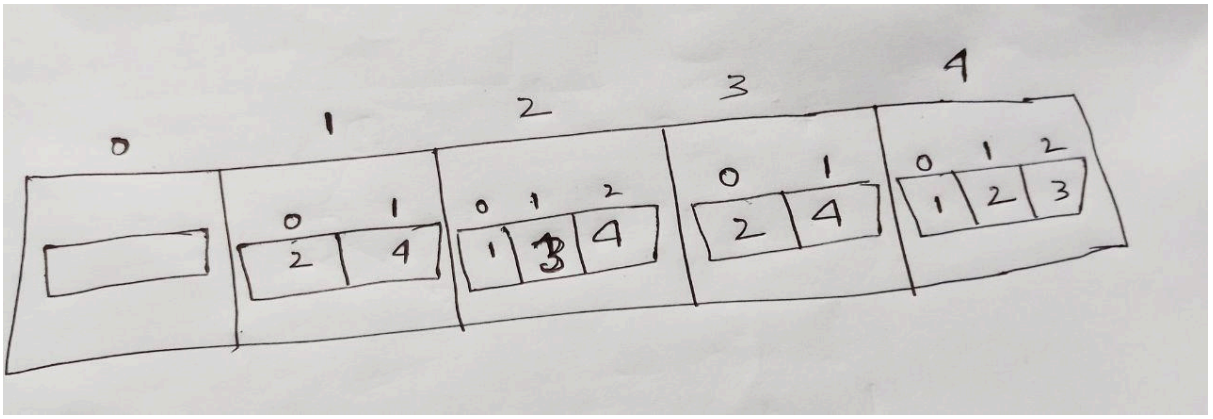
উপরের এই unweighted এবং undirected graph টি ব্যবহার করে আমরা সকল উদাহরণ এবং বর্ণনা দিব।

ADJACENCY LIST

এখানে আমরা একটি 2D vector use ইউজ করি এবং vector টির size হয় এতগুলো node আছে তত।

এবং প্রতিটি node যতগুলো node এর সাথে সংযুক্ত থাকে। তাদেরকে source node এর index এ push করা হয়।

এখন উপরের graph টির node গুলো এবং তাদের adjacency node সহকারে একটি vector এ স্টোর করলে উত্তরটি হবে অনেকটা নিচের ছবির মত।



বা

1 - > 2, 4

2 - > 1,3,4

3 - > 2,4

4 - > 1,2,3

ADJACENCY MATRIX

.

এক্ষেত্রে আমরা 2d array ব্যবহার করি। গ্রাফে যতগুলো node থাকে 2d array বা matrix এর রো এবং কলাম সংখ্যা কতগুলো হয়ে থাকে। কোন একটি node যদি অপর কোন node এর সাথে সংযুক্ত থাকে তাহলে তাদের দুজনের জন্য ম্যাট্রিক্সের যে ঘরটির নির্দিষ্ট করা সেটি এক ধারা পূর্ণ করতে হয। এবং কোন কানেকশন না থাকলে সেটাতে শূন্য হয়।

উপরের গ্রাফটির জন্য একটি adjacency ম্যাট্রিক্স তৈরি করা হলো

	1	2	3	4
1	0	1	0	1
2	1	0	1	1
3	0	1	0	1
4	1	1	1	0

এখানে row গুলো node এর প্রতিনিধিত্ব করে।

এবং কলামগুলো adjacency node এর প্রতিনিধিত্ব করে।

যেমন 1 এর adjacency হলো 2

তাই matrix এর [1][2]=1

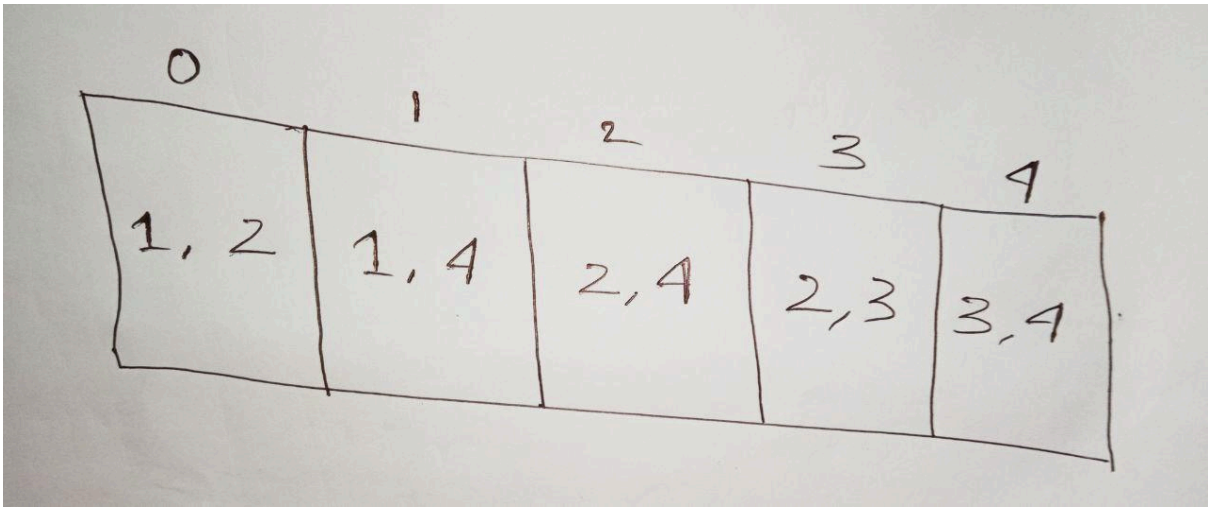
আবার 1 এর adjacency 3 নয়

তাই matrix এর [1][3]=0

EDGE LIST

Edge list এর ক্ষেত্রে কোন গ্রাফে যতগুলো node থাকে ঠিক তত size এরবআররায় বা vector declare করা হয় এবং প্রতিটি ইনডেক্সে একটি করে পেয়ার থাকবে আমরা যেহেতু গ্রাফের উপাদান গুলো নিয়েছি তাই পিয়ারের ফাস্ট এবং সেকেন্ড ইন্টেজার্ট থাকবে। এবং ভেক্টর বা এরে এর প্রতিটি ইনডেক্স একটি এজের প্রতিনিধিত্ব করবে।

উক্ত graph কে edge list এ পরিণত করলে ভিক্টর বা এরা কি হবে অনেকটা নিচের ছবির মত



ANSWER TO THE QUESTION NUMBER 3

->

```
#include<bits/stdc++.h>

using namespace std;

const int N=1000;

vector<int>adj[N];

int visited[N];
```

```
bool bfs(int src);
int main()
{
    for(int i=0; i<N; i++)visited[i]=-1;
    int n,m;
    cin>>n>>m;
    for(int i=0; i<m; i++)
    {
        int u,v;
        cin>>u>>v;
        adj[u].push_back(v);
        adj[v].push_back(u);
    }
    if( bfs(1))
    {
        cout<<"Cycle Exist";
        return 0;
    }
    cout<<"No Cycle";
    return 0;
}
bool bfs(int src)
{
    queue<int>q;
    q.push(src);
    while(!q.empty())
```

```
    {
        int head=q.front();
        visited[head]=0;
        q.pop();
        for(auto j:adj[head])
        {
            if(visited[j]==-1)
            {
                visited[j]=0;
                q.push(j);
            }
            else if(visited[j]==0)
            {
                return true;
            }
        }
        visited[head]=1;
    }
    return false;
}
```

ANSWER TO THE QUESTION NUMBER 4

->

```
#include<bits/stdc++.h>
using namespace std;
int sumCounter(vector<int>a,int sz);
int main()
{
    vector<int>a;
    int n;
    cin>>n;
    for(int i=0; i<n; i++)
    {
        int x;
        cin>>x;
        a.push_back(x);
    }
    cout<<sumCounter(a,n);
    return 0;
}
int sumCounter(vector<int>a,int sz)
{
    if(sz==1)return a[sz-1];
    return a[sz-1]+sumCounter(a, sz-1);



}
```

ANSWER TO THE QUESTION NUMBER 5

->

```
#include<bits/stdc++.h>
using namespace std;
const int N=1e5+5;
vector<int>adj[N];
bool visited[N];
void bfs(int src);
int main()
{
    int n,m;
    cin>>n>>m;
    for(int i=0; i<m; i++)
    {
        int u,v;
        cin>>u>>v;
        adj[u].push_back(v);
        adj[v].push_back(u);
    }
    bfs(1);
    if(visited[n])
```

```
    {
        cout<<"YES";
        return 0;
    }
    cout<<"NO";
    return 0;
}
void bfs(int src)
{
    visited[src]=true;
    queue<int>q;
    q.push(src);
    while(!q.empty())
    {
        int head=q.front();
        q.pop();
        for(auto j:adj[head])
        {
            if(visited[j]==false)
            {
                visited[j]=true;
                q.push(j);
            }
        }
    }
}
```

ANSWER TO THE QUESTION NUMBER 6

->

```
#include<bits/stdc++.h>
using namespace std;
int n,m;
const int N=1000;
int grid[N][N];
bool visited[N][N];
int dx[] = {0, 0, -1, 1};
int dy[] = {1, -1, 0, 0};
int level[N][N];
bool insidegrd(pair<int,int>src);
bool not_block(pair<int,int>src);
bool bfs(pair<int,int>src,pair<int,int>dst);
void path_printer();
pair<int,int> srcc;
pair<int,int>parent[N][N];
queue<pair<int,int>>DST;
void border_finder(int n,int m);
void path_printer(pair<int,int>dst);
int main()
```

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

            grid[i][j]=1;
        }
    }
    cin>>n>>m;
    for(int i=0; i<n; i++)
    {
        string a;
        cin>>a;
        for(int j=0; j<m; j++)
        {
            if(a[j]=='.')grid[i][j]=0;
            else if(a[j]=='A')
            {
                grid[i][j]=0;
                srcc= {i,j};
            }
        }
    }
    bool ok=false;
    int x,y;
```

```
border_finder(n,m);
while(!DST.empty())
{
    x=DST.front().first;
    y=DST.front().second;
    DST.pop();
    ok=bfs(srcc, {x,y});
    if(ok==true)
    {
        cout<<"YES"<<"\n";
        cout<<level[x][y]<<"\n";
        path_printer({x,y});
        return 0;
    }
    for(int i=0; i<N; i++)
    {
        for(int j=0; j<N; j++)
        {
            visited[i][j]=false;
            level[i][j]=0;
            parent[i][j]= {0,0};
        }
    }

}
cout<<"NO";
```

```
    return 0;
}
bool bfs(pair<int,int>src,pair<int,int>dst)
{
    visited[src.first][src.second]=true;
    queue<pair<int,int>>q;
    q.push(src);
    level [src.first][src.second]=0;
    while(!q.empty())
    {
        pair<int,int>head;
        head=q.front();
        q.pop();
        int x=head.first;
        int y=head.second;
        for(int i=0; i<4; i++)
        {
            pair<int,int> newnode= {x+dx[i],y+dy[i]};
            if(insidegrd(newnode)&&not_block(newnode)&&
                    visited[newnode.first][newnode.second]==false)
            {
                level[newnode.first][newnode.second]=level[x][y]+1;
                visited[newnode.first][newnode.second]=true;
                parent[newnode.first][newnode.second]= {x,y};
                q.push(newnode);
                if(newnode==dst)
```

```
            {
                return true;
            }
        }
    }
    return false;
}
bool insidegrd(pair<int,int>src)
{
    if(src.first<n&&src.first>=0&&src.second<m&&src.second>=0)return true;
    return false;
}
bool not_block(pair<int,int>src)
{
    if(grid[src.first][src.second]!=1)return true;
    return false;
}
void path_printer(pair<int,int>dst)
{
    queue<pair<int,int>>q;
    stack<char>st;
    q.push(dst);
    while(q.front()!=srcc)
    {
        pair<int,int>head=q.front();
```

```
        q.pop();
        if(head.first-parent[head.first][head.second].first==-1
                &&head.second-parent[head.first][head.second].second==0)
            st.push('U');
        else if(head.first-parent[head.first][head.second].first==1
                &&head.second-parent[head.first][head.second].second==0)
            st.push('D');
        else if(head.first-parent[head.first][head.second].first==0
                &&head.second-parent[head.first][head.second].second==1)
            st.push('R');
        else if(head.first-parent[head.first][head.second].first==0
                &&head.second-parent[head.first][head.second].second==-1)
            st.push('L');
        q.push(parent[head.first][head.second]);
    }
    while(!st.empty())
    {
        cout<<st.top();
        st.pop();
    }
}
void border_finder(int n,int m)
{
    for(int i=0; i<n; i++)
    {
        for(int j=0; j<m; j++)
```

```
        {
            if(i==0||i==n-1||j==0||j==m-1)
            {
                if(grid[i][j]==0)
                {
                    DST.push({i,j});
                }
            }
        }
    }
}
```

ANSWER TO THE QUESTION NUMBER 7

->

ঊক্ত GRAPH টির

VERTEX=9

EDGES= 15

একে adjacency list এ প্রকাশ করা হলো

A -> (B,2) (C,1) (I,18) (J,5)

B -> (A,2) (C,7) (E,20) (F,13) (H,15) (J,6)

C -> (A,1) (B,7) (E,9)

E -> (B,20) (E,9) (G,5)

F -> (B,13) (G,1) (H,21)

G -> (E,5) (F,1)

H -> (B,15) (F,21) (I,7)

I -> (A,18) (H,7) (J,6)

J -> (A,5) (B,6) (I,6)

গ্রাফটিতে E কে source node ধরে Dijkstra চালালে যা হবে তা একটি table এর মাধ্যমে represent করা হলো

SLECTED NODE	A	B	C	E	F	G	H	I	J
-	X	X	X	0	X	X	X	X	X
E	X	20	9	0	X	5	X	X	X
G	X	20	9	0	6	5	X	X	X
F	X	19	9	0	6	5	27	X	X
C	10	16	9	0	6	5	27	X	X
A	10	12	9	0	6	5	27	28	15
B	10	12	9	0	6	5	27	28	15
J	10	12	9	0	6	5	27	21	15
I	10	12	9	0	6	5	27	21	15
H	10	12	9	0	6	5	27	21	15

X ধারা infinity distance বুঝানো হয়েছে

টেবিলটি থেকে আমরা বলতে পারি

Source node থেকে সবার সর্বনিম্ন দূরত্ব

A =10

B=12

C=9

E=0

F=6

G=5

H=27

I=21

J=15

গ্রাফটিতে E কে source node ধরে Dijkstra এর optimized version চালালে যা ঘটবে তা লেখা হলো

1) যেহেতু আমরা E কে source node ধরেছি তাই মেইন ফাংশন থেকে এর দূরত্ব

শূন্য করে দেওয়া হবে। বাকিগুলোর দূরত্ব অসীম করে দেওয়া হবে।

2) Dijkstra function টিতে E সাথে নিজের দূরত্ব শূন্য কেউ নিয়ে আসবে। এবং একটি priority_queue তে আমরা এদেরকে পেয়ার আকারে স্টোর করব। অবশ্যই খেয়াল রাখতে হবে যাতে করে এদের দূরত্বের আগে মাইনাস চিহ্ন দেওয়া হয়। প্রায়োরিটি কিউতে বড় থেকে ছোট আকারে থাকে। আমাদের এখানে ছোটটিকে আগে এক্সেস করতে হবে। ছোটকে সবার আগে অ্যাক্সেস করার জন্য স্টোর করার সময় মাইনাস চিহ্ন দিতে হবে আবার যখন কিউ থেকে গ্রহণ করব আবার তার সাথে মাইনাস 1 গুণ করতে হবে ।

Priority_queue তে E কে push করব

3) কিউটি যতক্ষণ পর্যন্ত খালি না হবে এতক্ষণ পর্যন্ত একটি loop চলবে {

head= Priority_queue এর front বা E

Priority_queue .pop

E কে এরকম কখনো ভিজিট করা হয়নি

visited [E]= 1 করে

এর adjacency node গুলো আমরা চেক করব

কনো adjacency node এর দূরত্ব যদি E এর দূরত্ব এবং E থেকে ওই node যেতে যে দূরত্ব লাগে তার যোগফলের বড় হয় তাহলে

adjacency node এর দূরত্ব=E এর দূরত্ব এবং E থেকে ওই node যেতে যে দূরত্বকরে দিব এবং

adjacency node কে Priority_queue পুষ করে দেব।

E এর adjacency node গুলো হলো B,C,G এবং এদের বর্তমান দূরত্ব ইনফিনিটি এদের দূরত্ব আপডেট হবে
B=0+20=20

B=0+9=9 G=0+5=5

এবং এরা সবাই Priority_queue তে পুশ হবে।

আবার

Priority_queue তে B C G আছে এবং এদের মধ্যে সবথেকে কম দূরত্ব হলো G এর এবং এটি ভিজিটেড নয়

head= Priority_queue এর front বা G

Priority_queue .pop

visited [G]= 1 করে

এর adjacency node গুলো আমরা চেক করব

কনো adjacency node এর দূরত্ব যদি G এর দূরত্ব এবং E থেকে ওই node যেতে যে দূরত্ব লাগে তার যোগফলের বড় হয় তাহলে

adjacency node এর দূরত্ব=G এর দূরত্ব এবং G থেকে ওই node যেতে যে দূরত্বকরে দিব এবং

adjacency node কে Priority_queue পুষ করে দেব।

G এর adjacency node গুলো হলো E,F এবং এদের মাঝে Eদূরত্ব আমরা কমাতে পারবো না। কিন্তু F কে কমাতে পারবো F এর আপডেট হবে F=5+1=6

F কে Priority_queue পুষ করে দেব।

আবার

Priority_queue তে B C F আছে এবং এদের মধ্যে সবথেকে কম দূরত্ব হলো F এর এবং এটি ভিজিটেড নয়

head= Priority_queue এর front বা F

Priority_queue .pop

visited [F]= 1 করে

এর adjacency node গুলো আমরা চেক করব

কনো adjacency node এর দূরত্ব যদি F এর দূরত্ব এবং Fথেকে ওই node যেতে যে দূরত্ব লাগে তার যোগফলের বড় হয় তাহলে

adjacency node এর দূরত্ব=F এর দূরত্ব এবং F থেকে ওই node যেতে যে দূরত্বকরে দিব এবং

adjacency node কে Priority_queue পুষ করে দেব।

F এর adjacency node গুলো হলো B,G,G এবং এদের মাঝে G দূরত্ব আমরা কমাতে পারবো না। কিন্তু B and H কে কমাতে পারবো B এর আপডেট হবে B=6+13=19

H এর আপডেট হবে B=6+21=27

B and H কে Priority_queue পুষ করে দেব।

আবার

Priority_queue তে B C H আছে এবং এদের মধ্যে সবথেকে কম দূরত্ব হলো C এর এবং এটি ভিজিটেড নয়

head= Priority_queue এর front বা C

Priority_queue .pop

visited [C]= 1 করে

এর adjacency node গুলো আমরা চেক করব

কনো adjacency node এর দূরত্ব যদি C এর দূরত্ব এবং C থেকে ওই node যেতে যে দূরত্ব লাগে তার যোগফলের বড় হয় তাহলে

adjacency node এর দূরত্ব=C এর দূরত্ব এবং C থেকে ওই node যেতে যে দূরত্বকরে দিব এবং

adjacency node কে Priority_queue পুষ করে দেব।

C এর adjacency node গুলো হলো A, B,E এবং এদের মাঝে E দূরত্ব আমরা কমাতে পারবো না। কিন্তু A and B কে কমাতে পারবো B এর আপডেট হবে B=9+7=16

A এর আপডেট হবে B=9+1=10

B and A কে Priority_queue পুষ করে দেব।

আবার

Priority_queue তে A B H আছে এবং এদের মধ্যে সবথেকে কম দূরত্ব হলো A এর এবং এটি ভিজিটেড নয়

head= Priority_queue এর front বা A

Priority_queue .pop

visited [A]= 1 করে

এর adjacency node গুলো আমরা চেক করব

কনো adjacency node এর দূরত্ব যদি A এর দূরত্ব এবং A থেকে ওই node যেতে যে দূরত্ব লাগে তার যোগফলের বড় হয় তাহলে

adjacency node এর দূরত্ব=A এর দূরত্ব এবং A থেকে ওই node যেতে যে দূরত্বকরে দিব এবং

adjacency node কে Priority_queue পুষ করে দেব।

A এর adjacency node গুলো হলো C, B, I,J এবং এদের মাঝে C দূরত্ব আমরা কমাতে পারবো না। কিন্তু I and B and j কে কমাতে পারবো B এর আপডেট হবে B=10+2=12

I এর আপডেট হবে I=10+18=28

J এর আপডেট হবে I=10+5=15

B ,J and I কে Priority_queue পুষ করে দেব।

আবার

Priority_queue তে B H I J আছে এবং এদের মধ্যে সবথেকে কম দূরত্ব হলো B এর এবং এটি ভিজিটেড নয়

head= Priority_queue এর front বা B

Priority_queue .pop

visited [B]= 1 করে

এর adjacency node গুলো আমরা চেক করব

কনো adjacency node এর দূরত্ব যদি B এর দূরত্ব এবং B থেকে ওই node যেতে যে দূরত্ব লাগে তার যোগফলের বড় হয় তাহলে

adjacency node এর দূরত্ব=B এর দূরত্ব এবং B থেকে ওই node যেতে যে দূরত্বকরে দিব এবং

adjacency node কে Priority_queue পুষ করে দেব।

এর adjacency node গুলো হলোA, C, E, F,H,J এবং এদের মাঝে কারো দূরত্ব আমরা কমাতে পারবো না।

আবার

Priority_queue তে H I J আছে এবং এদের মধ্যে সবথেকে কম দূরত্ব হলো J এর এবং এটি ভিজিটেড নয়

head= Priority_queue এর front বা J

Priority_queue .pop

visited [J]= 1 করে

এর adjacency node গুলো আমরা চেক করব

কনো adjacency node এর দূরত্ব যদি Jএর দূরত্ব এবং J থেকে ওই node যেতে যে দূরত্ব লাগে তার যোগফলের বড় হয় তাহলে

adjacency node এর দূরত্ব=J এর দূরত্ব এবং J থেকে ওই node যেতে যে দূরত্বকরে দিব এবং

adjacency node কে Priority_queue পুষ করে দেব।

J এর adjacency node গুলো হলো A, B, I এবং এদের মাঝে A,B দূরত্ব আমরা কমাতে পারবো না। কিন্তু I কে কমাতে পারবো I

এর আপডেট হবে I=15+6=21

I কে Priority_queue পুষ করে দেব।

আবার

Priority_queue তে H I আছে এবং এদের মধ্যে সবথেকে কম দূরত্ব হলো I এর এবং এটি ভিজিটেড নয়

head= Priority_queue এর front বা I

Priority_queue .pop

visited [I]= 1 করে

এর adjacency node গুলো আমরা চেক করব

কনো adjacency node এর দূরত্ব যদি I এর দূরত্ব এবং I থেকে ওই node যেতে যে দূরত্ব লাগে তার যোগফলের বড় হয় তাহলে

adjacency node এর দূরত্ব=I এর দূরত্ব এবং I থেকে ওই node যেতে যে দূরত্বকরে দিব এবং

adjacency node কে Priority_queue পুষ করে দেব।

এর adjacency node গুলো হলো A,H,J এবং এদের মাঝে কারো দূরত্ব আমরা কমাতে পারবো না।

আবার

Priority_queue তে H আছে এবং এদের মধ্যে সবথেকে কম দূরত্ব হলো H এর এবং এটি ভিজিটেড নয়

head= Priority_queue এর front বা H

Priority_queue .pop

visited [H]= 1 করে

এর adjacency node গুলো আমরা চেক করব

কনো adjacency node এর দূরত্ব যদি H এর দূরত্ব এবং H থেকে ওই node যেতে যে দূরত্ব লাগে তার যোগফলের বড় হয় তাহলে

adjacency node এর দূরত্ব=H এর দূরত্ব এবং H থেকে ওই node যেতে যে দূরত্বকরে দিব এবং

adjacency node কে Priority_queue পুষ করে দেব।

এর adjacency node গুলো হলো B,F,I এবং এদের মাঝে কারো দূরত্ব আমরা কমাতে পারবো না।

সবশেষে কিউ টি এমটি হয়ে যাবে এবং আমাদের loop টি break হয়ে প্রোগ্রামটি শেষ হবে

}

ANSWER TO THE QUESTION NUMBER 8

->

কোন recursive function এর ক্ষেত্রে base case এবং recursive case অতি গুরুত্বপূর্ণ।

নিচে এদের উদাহরণসহ বর্ণনা দেওয়া হল।

BASE CASE

এটি হলো কোন সমস্যার সব থেকে ক্ষুদ্রতম অংশ এবং এর উত্তর আমাদের আগে থেকে জানা থাকে তাই এক্ষেত্রে আমরা আর ফাংশনটি কে কল করি না।

অর্থাৎ যে নির্দিষ্ট একটি পর্যায়ে চলে গেলে recursive function নিজেকে আর কল করে না তাকেই বলা হয় base case

RECURSIVE CASE

এগুলোও recursive function এর ক্ষুদ্র ক্ষুদ্র অংশ।

> কিন্তু এরা হলো অরজিনাল বড় সমস্যাটির এবং base case এর মাঝামাঝি ধাপগুলো। অর্থাৎ মেইন ফাংশন থেকে কোন recursive ফাংশন কে কল করার পর ফাংশনটি base case এ পৌঁছানোর আগ মুহূর্ত পর্যন্ত এ ধাপ গুলোতে ফাংশন নিজেই নিজেকে কল করতে থাকে সেই ধাপ গুলোকেই আলাদা আলাদা ভাবে recursive case বলা যায়।

নিচে একটি রিকার্সি ফাংশন ব্যবহার করে আরো স্পষ্টভাবে বর্ণনা করা হলো

```
#include<bits/stdc++.h>
using namespace std;
int fac(int n);
int main()
{
   int n;
   cin>>n;
   cout<< fac(n);
   return 0;
}
int fac(int n)
{
   if(n==1)return 1;
   return n*fac(n-1);
}
```

এখানে রিচার্শিভ ফাংশনের মাধ্যমে কোন সংখ্যার ফ্যাক্টরিয়াল বের করার চেষ্টা করা হয়েছে।

ধরি আমাদের নাম্বারটি 5

i) এটি হলো আমাদের অরজিনাল বড় সমস্যা। আমরা জানি না 5 এত ফ্যাক্টরিয়াল কত কিন্তু আমরা জানি 5 এর সাথে 5-1 বা 4 এর ফ্যাক্টরিয়াল গুণ করলে আমরা 5 এর ফ্যাক্টরিয়াল পাব। আবার 4 এর ক্ষেত্রে একই নিয়ম প্রযোজ্য। অর্থাৎ আমরা সমস্যাটিকে ছোট ছোট অংশে ভাগ করে নিব। কিন্তু আমরা জানি 1 এর ফ্যাক্টরিয়াল 1 অর্থাৎ এটি হলো আমাদের প্রোগ্রামের ক্ষুদ্রতম অংশ যারভমান আমরা আগে থেকে জানি এবং এই ধাপে চলে গেলে আমরা আর প্রোগ্রাম কে কল করব না শুধুমাত্র রিটার্ন করে চলে আসব।

তাই আমাদের বেস কেস হলো যদি n == 1 ওয়ান হয় তাহলে 1 রিটার্ন করব।

ii) 5 কে call করব main function থেকে।

iii) মেইন ফাংশনে এখন এন এর মান 5। তার মানে base case স্পর্শ করেনি এটি একটি recursive case।

অর্থাৎ ফাংশনটি নিজেকে এখন 5-1 কল করবে।

iv) মেইন ফাংশনে এখন এন এর মান 4। তার মানে base case স্পর্শ করেনি এটি একটি recursive case।

অর্থাৎ ফাংশনটি নিজেকে এখন 3-1 কল করবে।

v) মেইন ফাংশনে এখন এন এর মান 2 । তার মানে base case স্পর্শ করেনি এটি একটি recursive case।

অর্থাৎ ফাংশনটি নিজেকে এখন 2-1 কল করবে।

v) মেইন ফাংশনে এখন এন এর মান 1 । তার মানে base case স্পর্শ করেছে অর্থাৎ ফাংশনটি নিজেকে এখন আর কল করবে না এবং এটি এর নির্ধারণ করে দেওয়া মান রিটার্ন করে ধাপে ধাপে একদম মেইন ফাংশন এর নিকট পৌঁছাবে।