Class 98

30 May 2023 11:09

Class 98

Graphs

Multisource BFS

BFS → SSSP unweighted / Equally weighted

P1

P2

Min time

Multiple source Node

Multisource BFS

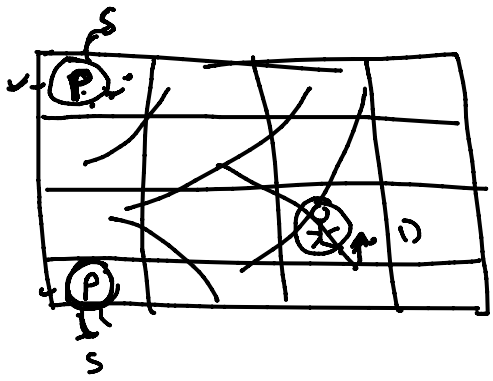
P3

grid → graph

BFS → shortest Distance

2

Shortest Distance



DP → Recursion

DFS

BFS ?

q.push({0,0}) ;

q.push({3,0})

while(q.empty())

{

int dx[4] = {-1, 0, 1, 0};

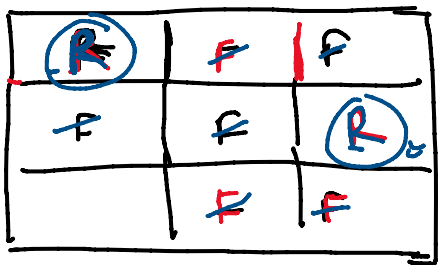
int dy[4] = {0, 1, 0, -1};

Src Node

t = 0 min

t = 1 min

Rotten oranges



2 min

Rotten

t = 2

every min

Rotten

t = 4 min

Time = 4 Hours

MinTime

Time=distance

	F	F

MinTime

Directly Connected

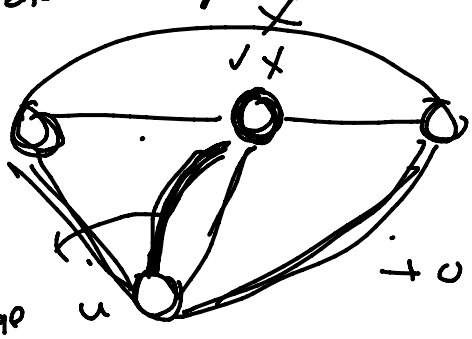
Fresh Orange Farthest from soc Rotten orange

```
queue < pair < pair < int, int >, int > > q;

For ( i=0; i<n; i++ )
{ For (j=0 ; j<m ; j++ )
  {   if ( mat[i][j] == 2)
      { q.Push ( { { i, j }, 0 } );
        mat[i][j]=0;
      }
  }
}

int dx[4] = { -1, 1, 0, 0 };

int dy[4] = { 0, 0, -1, 1 };

int ans = 0 ;
while( !q.empty() )
{   auto node= q.front ();
    int x= node.first.first;
    int y =  node.first.second;
    int dis=  node.second;
    q.pop();
    ans = max(ans, dis);
    // Travel adj list;
```

Dist matrix

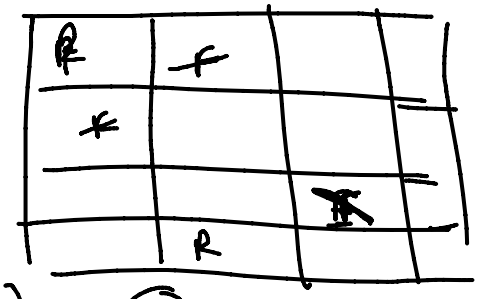
```
q.Push ( { { i, j }, 0 } );
vis[i][j] = 1;
```

```
For(k=0; k<4; k++)
{   int nu = u + du[k];
    int ny = y + dy[k];
    if( nu >= 0 and nu<n and ny >= 0 and ny<m and
                                    mat[nu][ny] == 1 );
    {  if mat[nu][ny] = 0 ;
        q.push( { nu, ny }, dis+1 );
    }
}
return ans;
```



edge u

Travel

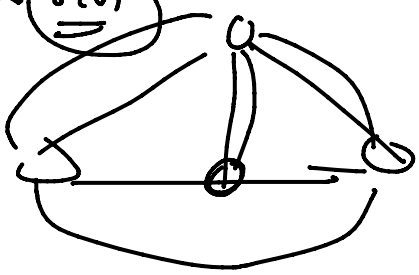


-1

+ O (V + E)

2E, 2V

~ O(V)

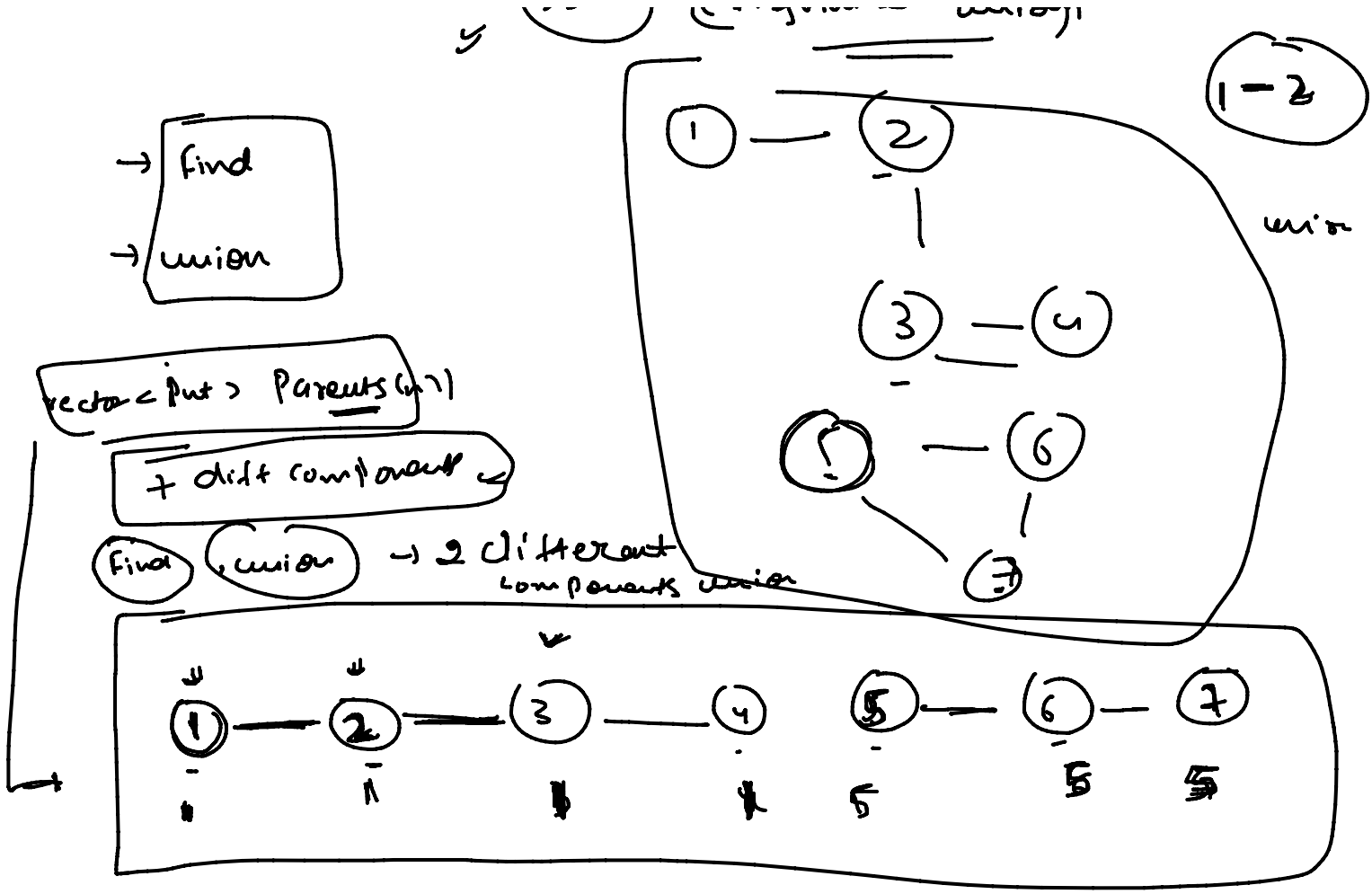


adjacency list → Edge

```
For( n )
{
  n
  For( m )
  {
    n
  }
}
```

(n*m)

DSU ((Disjoint set union))



Low Parent → super Parent → represent that component

No of components → DSU

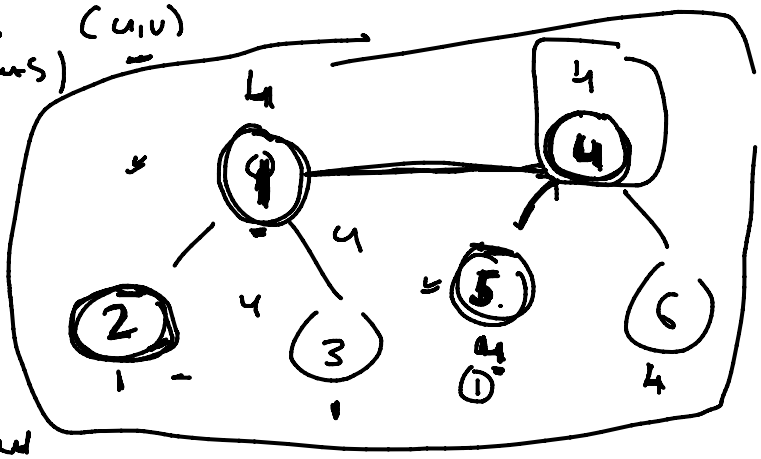
Cycle detect

```
Vector<int> Parent(n);
For( i=0; i<n; i++)
{ Parent[i] = i;
}
// using the edge[] union of different components
For(auto it : edge[]
```

```
{   int u = it[0];
    int v = it[1];
    union(u, v, parents);
}
}
```

```
void union(int u, int v, vector<int> parents)
{   int para = find_parent(u);
    int parb = find_parent(v);
    if(para == parb)
    {  // already same component
        return;
    }
    else
    {  parent[parb] = para;
    }
}
```

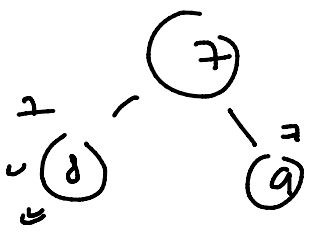
(u,v)



2 → comp / super parent

5 - 3

u == parent[u]



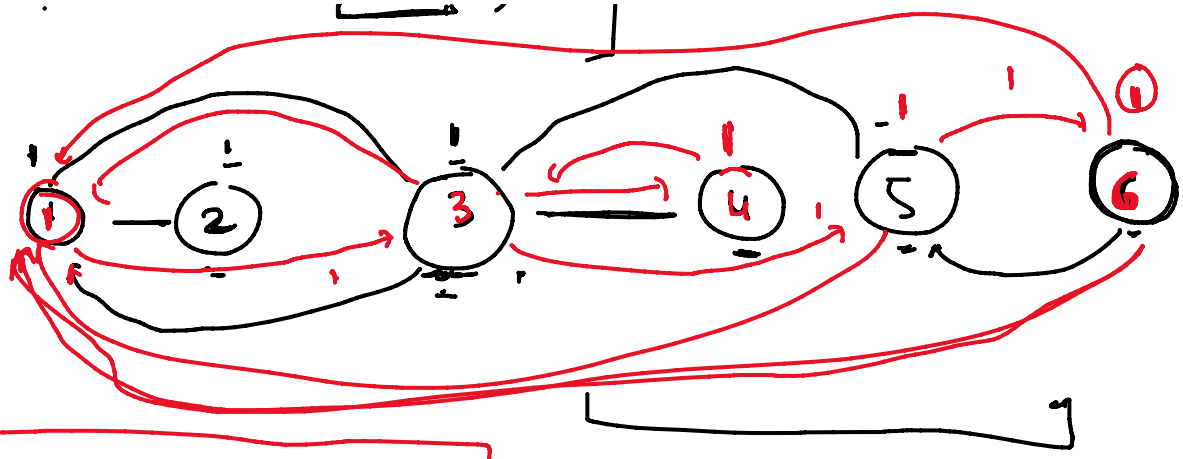

```
int find_parent(int u)
{   if(u == par[u])
    {  return u;
    }
    return find_parent(par[u]);
}
```

Recursion

find_parent(u)

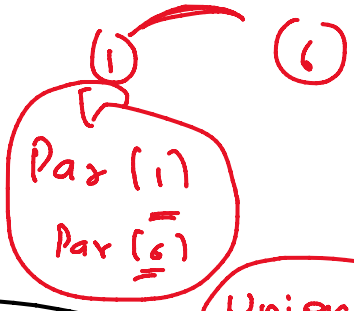
find_parent

Find-Parent



Path compression technique

5 : 4



Union by Rank

```
int find-Parent (int u)
{   if (u == par(u)) return u;
    return Par[u] = find - parent (Par[u]);
```

Par [x] = 1

Attansion → DSU

Union

Cycle