GRAPHS

Day 3

(Shortest Path Algorithms)

Youtube link :

https://youtu.be/gASQRs0aGhI

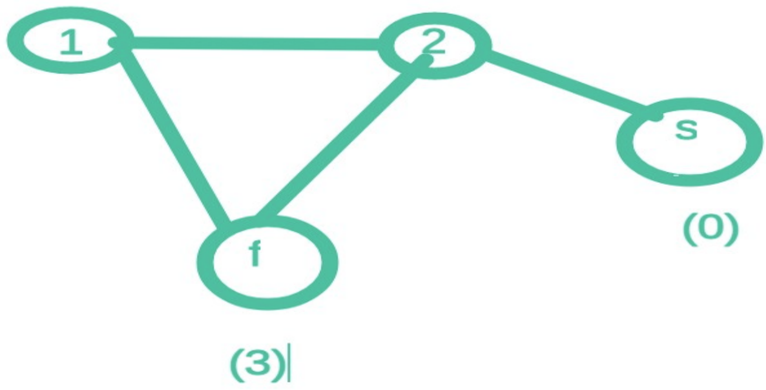
Contents:

BFS Algorithm for shortest path
(For unweighted graph)
Weight of edge will be 0 or 1

Given an undirected graph, a source vertex s, and a destination vertex f . Find the shortest distance from s to f and also the shortest path.

Link : https://cses.fi/problemset/task/1667

Example s=0, f=3



In above example, shortest distance = 2 and shortest path is 0, 2, 3

Solution

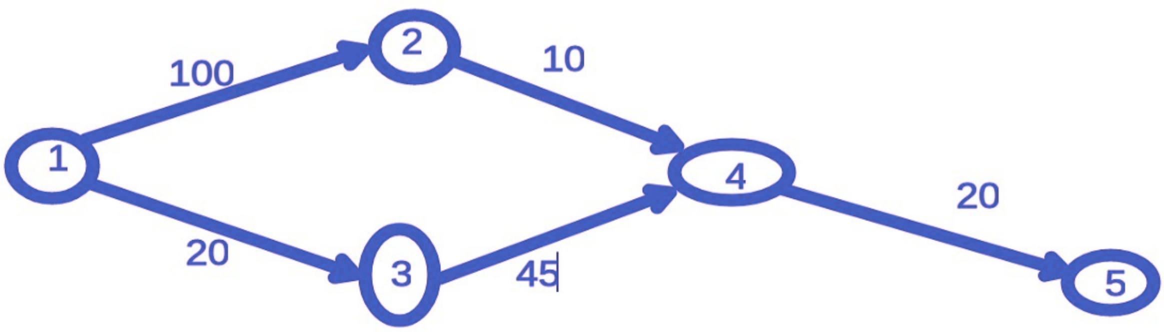
We can apply BFS to find shortest distance
For finding shortest path, store the parent of each vertex (some node from which a node was reached).

```
int parent[ ];
// parent[i] = -1 for all nodes
bool vis[ ];
queue<int> q;
int dist[]; // initialise with INFINITY
(1e18)
q.push(s);
```

```
vis[s]=true;
dist[0];
while(!q.empty())
{
  int t=q.front();
  for(auto it: adj[t])
 {
     if(!vis[it] )
     {
          vis[it] = true;
          parent[it]=t;
          q.push(it);
    dist[it]=dist[t]+1;
     }
 }
}
cout<<dist[ f ];
vector<int> path;
path.push_back(f);
trace(f);
reverse(path.begin(), path.end());
// now, print the path vector
void trace(int node) {
```

```
 if( parent[node] == -1)
     return;
 path.push(parent[node]);
 trace(parent[node]);
}
```

Shortest Path in DAG
(Weighted Graph)

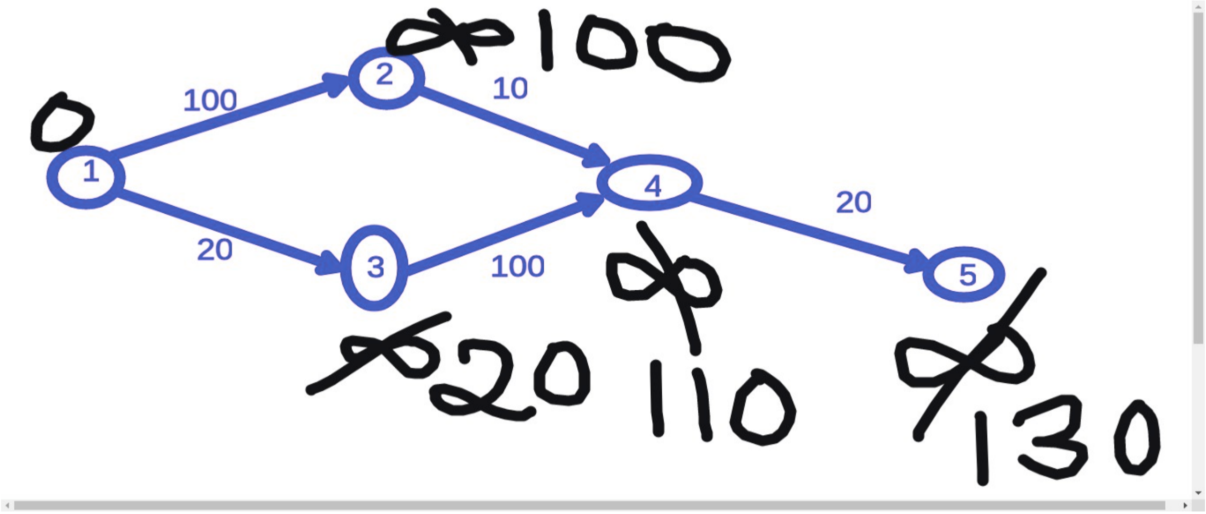


Relaxation Operation

Node u, has a neighbour v with a edge(u,v) with weight w.

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

Algorithm



1. Mark dist[i] = INFINITY (10^{18}) for all nodes and dist[s]=0;
2. Get topological sorted ordering of the graph in a vector topo.
3. Relax all neighbours of these nodes in this order.

```
for( int i=0; i<n; i++)
{
        int u=topo[i];
       for(auto it: adj[u])
      {  // Relaxing all neighbours of u
        int v=it;
        int w=weight(u,v);
         if(dist[u] + w < dist[v])
           {
```

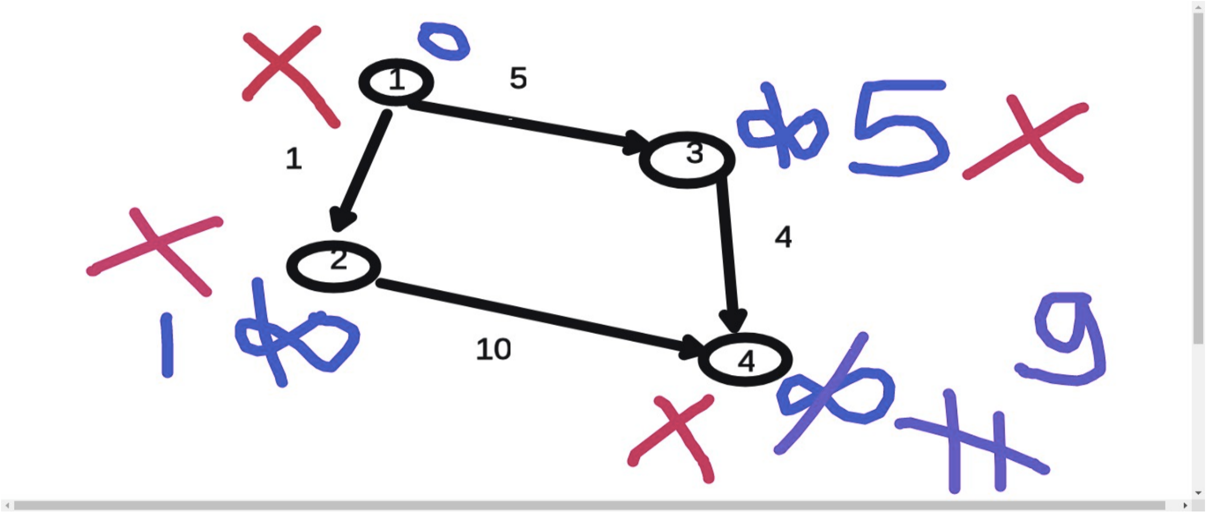
```
                dist[v] = dist[u] + w ;
                parent[v]=u; // for path
            }
        }
}
// Call trace() as we did in previous
question to trace the shortest path
```

Try this problem:
https://cses.fi/problemset/task/1680

Dijsktra's Algorithm

Given any weighted graph with **non-negative edges**, find shortest distance as well as shortest path from a source node s to destination node f.

- Distance Minimisation Problem
- Greedy algorithm
- Pick node with shortest distance till now and relax all its neighbours



STL Priority Queue

- Contains the largest element at top, by default.

```
#include<bits/stdc++.h>
using namespace std;

int main()
{
        priority_queue<int> pq;
        pq.push(2);
        pq.push(5);
        pq.push(1);
        pq.push(4);
        cout<< pq.top(); // 5
```

```
        return 0;
}
```

3 functions in priority_queue:

1. pq.push() ; O(log N)
2. pq.pop(); O(log N)
3. pq.top(); O(1)

You can also pass a comparator like this, so that it contains the smallest element at top:

```
int main()
{
        priority_queue<pair<int,int>,
vector<pair<int,int>> ,
greater<pair<int,int>> > pq;
        pq.push({3,1});
        pq.push({8,2});
        pq.push({6,0});
        pq.push({8,1});
        cout<< pq.top().first<<' ' <<
pq.top().second; // 3 1
        return 0;
}
```

Priority Queue of structure (Optional)

```
#include<bits/stdc++.h>
using namespace std;
struct student
{
   int marks;
   int rollno;
   bool operator < (const student & s2)
const
   {
      return (marks < s2.marks) ||
(marks==s2.marks && rollno > s2.rollno);
   }
// Top element would have highest marks and
if marks are equal then lowest roll number
};


int main()
{
      priority_queue<student> pq;
      pq.push({3,1});
```

```
    pq.push({6,0});
    pq.push({8,2});
    pq.push({8,1});
    cout<< pq.top().marks<<"
"<<pq.top().rollno; // 8 1
    return 0;
}
```

Code for Dijkstra Algorithm

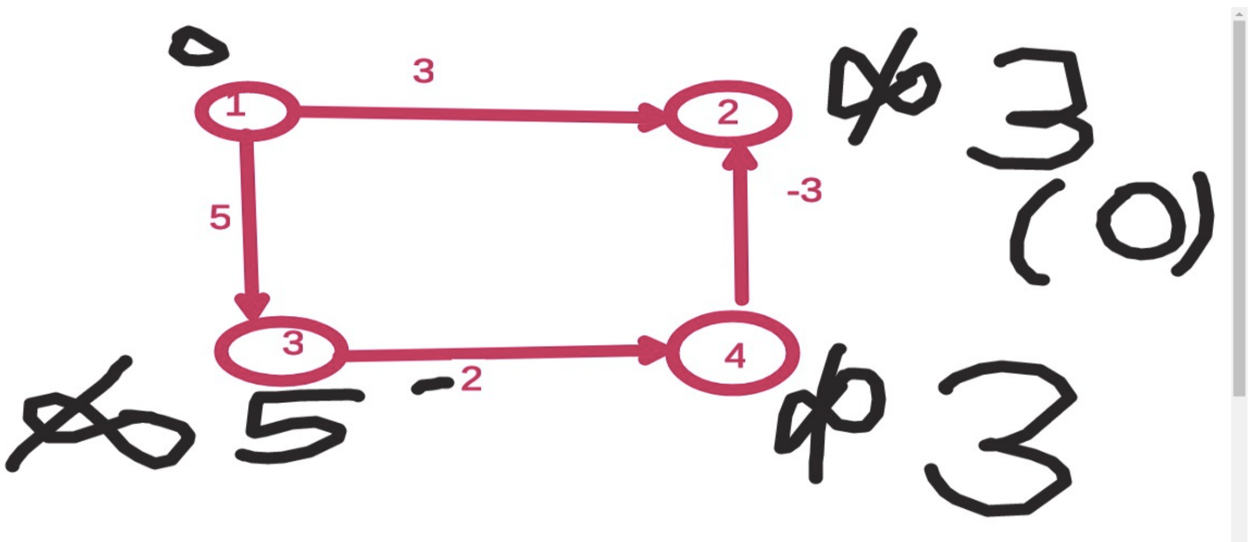
```
#define pii pair<int,int>
// n: number of nodes
int dist[n];
void dijkstra(int s)
{
priority_queue<pii, vector<pii> ,
greater<int,int> > pq;
for(int i=0; i<n; i++)
{
  dist[i]=INFINITY;
}
vector<bool> vis(n,false);
```

```
dist[s]=0;

pq.push({dist[s], s});

while (!pq.empty())
{
int u=pq.top().second;
pq.pop();
if (vis[u]) continue;
vis[u]=true;

for(auto p: adj[u])
{
    int v=p.first;
    int w=p.second;
    if(!vis[v] && dist[u] + w < dist[v])
   {
        dist[v] =dist[u] + w;
        parent[v] = u;
       pq.push({dis[v], v});
   }
}
}
```

```
}
```

Time complexity: O(V log(V) + E log(V))

Limitation:
Dijkstra algorithm fails when edges have negative weights. See example below



Bellman Ford Algorithm

- Will also work for negative edges

n = number of nodes
e = edges

```
dp[i] = shortest distance to node i
```

```
dp[i] = INFINITY for all
dp[s]=0;
for(int i=1; i<=n-1; i++)
{
      for(auto e: edges)
      {
        int u=e.first;
        int v=e.second.first;
        int w=e.second.second;
        if(dp[u]+w <dp[v])
         {    // relaxation
              dp[v]=dp[u]+w;
         }
      }
}
```

Time complexity: O(V. E)

Floyd Warshall Algorithm
(All Pair Shortest Path Algorithm)

- Based on DP

dist[i][j] = shortest distance from node i to node j
// initially all dist[i][j]= INFINITY if there is no edge of i to j
// dist[i][j] = weight(i,j) if there is edge from it to j
// dist[i][i]=0;

```
for(int k=0; k<n; k++)
{
    for(int i=0; i<n; i++)
   {
      for(int j=0; j<n; j++)
            {
            dp[i][j]  = min(dp[i][j],
dp[i][k] + dp[k][j]);
            }
   }
}
```

Time complexity: O (V^3)

Try:

1. https://cses.fi/problemset/task/1671
2. https://cses.fi/problemset/task/1672
3. https://www.spoj.com/problems/ARBITRAG/

Detecting negative cycle in a graph using Bellman Ford

1. Run normal bellman ford algorithm
2. Run an extra iteration and check if any vertex can be relaxed, we can say there is a “Negative cycle”.

(In negative cycle, there is no finite shortest path possible)

Graph Modelling

Try to solve this problem:
(Similar problem was asked in Goldman Sachs and Sprinklr coding rounds this year)

You want to travel from city A to city B
There are N cities and M bidirectional roads connecting these cities

Your car can store only upto C litres of fuel and the tank is initially full.
Each road(i,j) has a value Wij = amount of fuel needed to go from i to j
And in every city, you can buy fuel at a rate C[i] dollar / litre
Find the minimum amount of dollars to travel from A to B.

Solution Approach :
Imagining a new graph:
```
struct node {
  int city;
  int remainingFuel;
};
```
dist[node] = min. dollars spent to reach this state (node)
Total states (nodes) = N * (C+1)

Suppose u are at a node (u, f) // city u and fuel remaining is f
Suppose there is edge from u to v

You can move to :
(i) node (v, f - W(u,v)) by paying 0 dollar
(ii) node (u, f +1) by paying C[u] dollars
(These will cover all the cases)

Now, It becomes a simple shortest path problem
So, you can apply Dijkstra Algorithm here.
Such problems are also called **State Dijkstra.**
You can also refer this blog for graph modelling :
https://codeforces.com/blog/entry/45897

Now, think if NxC <= 10^10, array of C[i] was of 120 size only, what change will you make ?

For distance array instead of using dist[N][C+1], take an unordered map and when updating distance, check whether node is present in unordered map or not.

Try this problem: (Exactly similar problem was asked in Sprinklr internship coding round this year)

https://www.hackerearth.com/practice/algorithms/graphs/shortest-path-algorithms/practice-problems/algorithm/shortest-path-revisited-9e1091ea/

If you get stuck, have a look at my submission:
https://csacademy.com/code/6lzpE1DT/

Also try this problem:
(CSES Flight Discount)
https://cses.fi/problemset/task/1195

Also try this problem (asked in GS intern test '20):
https://www.hackerrank.com/challenges/synchronous-shopping/problem

[Hint: Use bitmasks for a state/node]
In case you need it, link to my submission:

https://csacademy.com/code/abQnzrEa/