**Name:Vishal Chauhan**

**Prn:2020BTECS00090**

**SUB:DAA**

**Ass no:9**

**Title of assignment:** Dynamic Programming

Q1. From a given vertex in a weighted connected graph, Implement shortest path finding

Bellman-Ford algorithm.

* Algorithm: (Pseudocode)

………………………………………………….

STEP 1 : distance[] , previous[] , V->vertex , G->Graph

STEP 2 : for each vertex V in G

distance[V] <- infinite

previous[V] <- NULL

STEP 3 : distance[S] <- 0

STEP 4 : for each vertex V in G

for each edge (U,V) in G

tempDistance <- distance[U] + edge\_weight(U, V)

if tempDistance < distance[V]

distance[V] <- tempDistance

previous[V] <- U

STEP 5 : for each edge (U,V) in G

If distance[U] + edge\_weight(U, V) < distance[V}

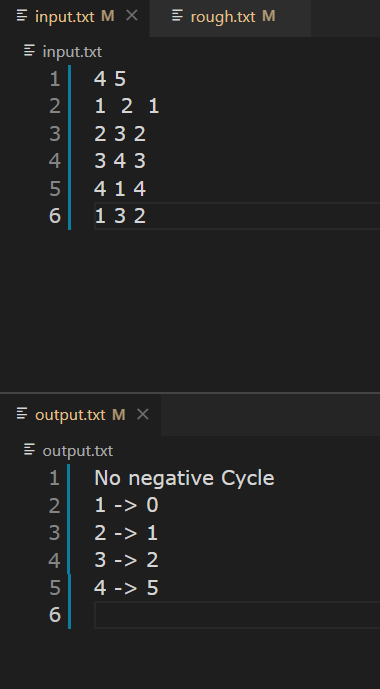
return Negative Cycle Exists

return distance[], previous[].

* **Code snapshots of implementation**
* #include <bits/stdc++.h>
* using namespace std;
* #define pii pair<int, int>
* const int N = 1000005;
* int n, m;
* struct node
* {
* int u;
* int v;
* int wt;
* node(int first, int second, int weight)
* {
* u = first;
* v = second;
* wt = weight;
* }
* };
* signed main()
* {
* ios\_base::sync\_with\_stdio(false);
* cin.tie(NULL);
* #ifndef ONLINE\_JUDGE
* freopen("C:\\Users\\Teknath\\Desktop\\code\\input.txt", "r", stdin);
* freopen("C:\\Users\\Teknath\\Desktop\\code\\output.txt", "w", stdout);
* #endif
* vector<node> edges;
* vector<int> dis(N, INT\_MAX);
* cin >> n >> m;
* for (int i = 1; i <= m; i++)
* {
* int u, v, wt;
* cin >> u >> v >> wt;
* edges.push\_back(node(u, v, wt));
* }
* for(int i=1;i<=n;i++)
* dis[i]=INT\_MAX;
* //let source is 1
* dis[1] = 0;
* //traverse for n-1 time
* for (int i = 1; i <= n - 1; i++)
* {
* for (auto it : edges)
* {
* if (dis[it.u] + it.wt < dis[it.v])
* dis[it.v] = dis[it.u] + it.wt;
* }
* }
* int fl = 0;
* for (auto it : edges)
* {
* if (dis[it.u] + it.wt < dis[it.v])
* {
* cout << "Negative Cycle is here \n";
* fl = 1;
* }
* }
* if (fl == 0)
* {
* cout << "No negative Cycle \n";
* for (int i = 1; i <= n; i++)
* {
* cout << i << " -> " << dis[i] << '\n';
* }
* }
* }

**………………………………………………….**

**OUTPUTS :**

****

**………………………………………………….**

* **Complexity of proposed algorithm (Time & Space)**

**………………………………………………….**

**Space Complexity :**

**As here I have used minimum priority queue so fetching is done in logV time and we are doing for all Edges E so time complexity is O( V)**

**Time Complexity :**

**Only distance array is extra space for all vertex So space complexity is O(VE)**

**………………………………………………….**

* **Your comment (How your solution is optimal?)**

**………………………………………………….**

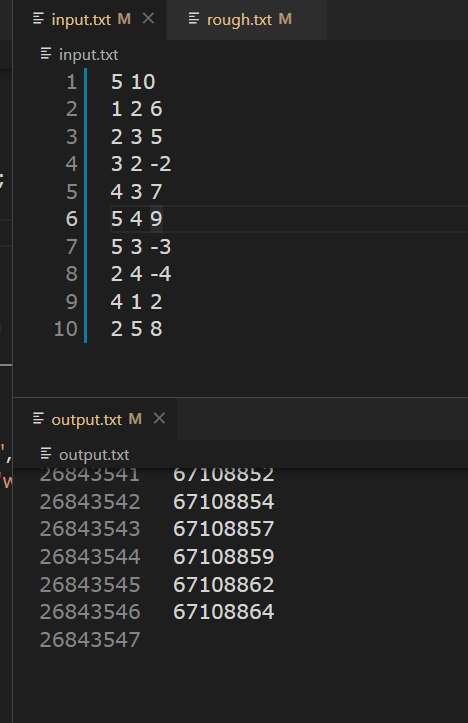
**In this algorithm as we have relax n-1 time each node so at nth relaxation we got our result , thus it optimal than Dijkistra algorithm.**

**………………………………………………….**

Q) Show that Dijkstra’s algorithm doesn’t work for above graph

Ans-> As applying Dijkistra algorithm we end up with a infinity while loop as shown in pic

Because priority queue never gets empty

****

****

ANS ->

We can simply implement this optimization of BELLMAN-FORD algorithm by remebering if v was relaxed or not.

If v is relaxed then we wait to see if v was udpated (which means being relaxed again).

If v was not updated, then we would stop

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***