← Notes

Dijkstra's Algorithm in Brief

Dijkstra's Algorithm

Dijkstra's Algorithm solves the single source shortest path problem in **O(E + V)logV)** time, which can be improved to **O(E + VlogV)** when using a Fibonacci heap. This note requires that you understand basic graph theory terminology and concepts.

Single Source Shortest Path (sssp)

The **sssp** is to find the shortest distance from the source vertex to all other vertices in the graph. We can store this in a simple array.

Psuedo-code

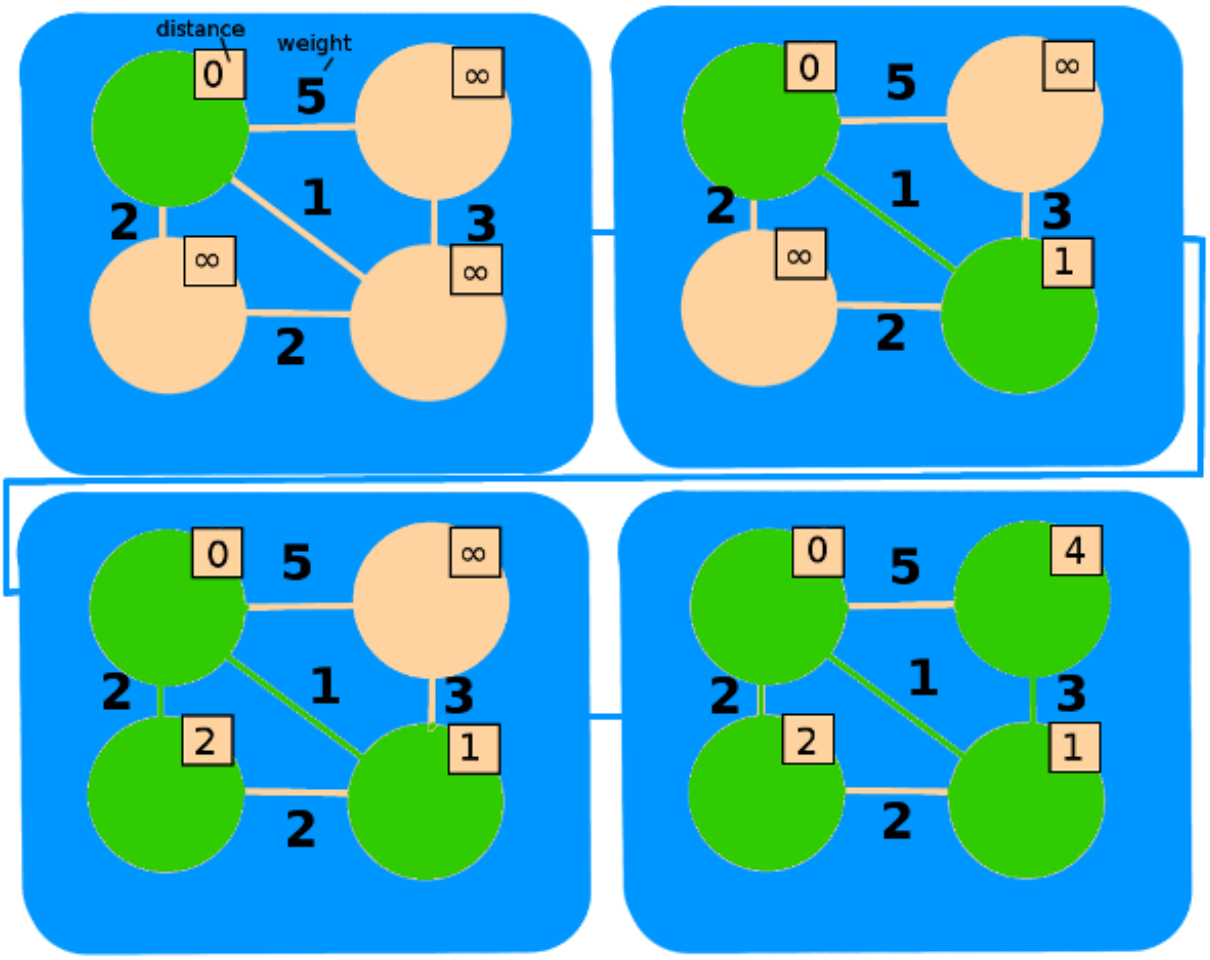
1. Set the distance to the source to 0 and the distance to the remaining vertices to infinity.
2. Set the **current** vertex to the source.
3. Flag the **current** vertex as visited.
4. For all vertices adjacent to the **current** vertex, set the distance from the source to the **adjacent** vertex equal to the minimum of its present distance and the **sum** of the **weight of the edge** from the current vertex to the adjacent vertex and the distance from the source to the **current** vertex.
5. From the set of **unvisited vertices**, arbitrarily set one as the new **current** vertex, provided that there exists an edge to it such that it is the minimum of all edges from a vertex in the set of **visited vertices** to a vertex in the set of **unvisited vertices**. To reiterate: The new current vertex must be unvisited and have a minimum weight edges from a visited vertex to it. This can be done trivially by looping through all visited vertices and all adjacent unvisited vertices to those visited vertices, keeping the vertex with the minimum weight edge connecting it.
6. Repeat steps 3-5 until all vertices are flagged as visited.

Implementation

```
#include <iostream>
#include <algorithm>
using namespace std;
const int INF = 1<<29;
```

```
int N, M, adj[1002][1002], dist[1002]; bool flag[1002];
void dijkstra(int s){
    fill(dist, dist+1002, INF);
    dist[s] = 0;
    for(int i=1; i<=N; i++){
        int d=INF, u=0;
        for(int j=1; j<=N; j++)
            if(!flag[j] && dist[j]< d){
                d=dist[j]; u=j;
            }
        flag[u] = 1;
        for(int j=1; j<=N; j++)
            if(!flag[j])
                dist[j]=min(dist[j], dist[u]+adj[u][j]);
    }
}
int main(){
    cin >> N >> M;
    fill_n(&adj[0][0], 1002*1002, INF);
    for(int i=0, u, v, w; i<M; i++){
        cin >> u >> v >> w;
        adj[u][v] = adj[v][u] = min(adj[u][v], w);
    }
    dijkstra(1);
    for(int i=1; i<=N; i++)
        if(flag[i]) cout << dist[i] << endl;
        else cout << -1 << endl;
}
```

Diagram



Proof of Correctness

CLRS explains it best, but the Wikipedia page is good as well.

Priority Queue

Dijkstra's algorithm can be easily sped up using a priority queue, pushing in all unvisited vertices during step 4 and popping the top in step 5 to yield the new current vertex.

Visualizations

Visit VisuAlgo.

AUTHOR

Bob Hob
Not Applicable at Not Appl...
Bangalore, Karnataka, India
1 note

Write Note
My Notes
Drafts

TRENDING NOTES

Strings And String Functions
written by Vinay Singh

Segment Tree and Lazy Propagation
written by Akash Sharma

Number Theory - II
written by Tanmay Chaudhari

Matrix exponentiation
written by Mike Koltsov

Graph Theory - Part II
written by Pawel Kacprzak

more ...

RECRUIT

Developer Sourcing

Lateral Hiring

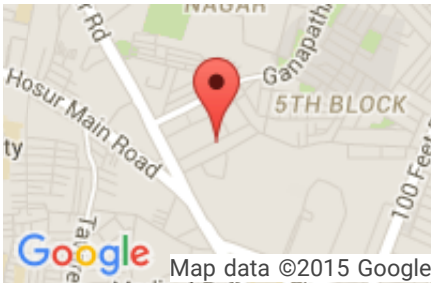
Campus Hiring

FAQs

Customers

Annual Report

REACH US



IIIrd Floor, Salarpuria Business Center,
4th B Cross Road, 5th A Block,
Koramangala Industrial Layout,
Bangalore, Karnataka 560095, India.

contact@hackerearth.com

+91-80-4155-4695

+1-650-461-4192