

A decorative graphic on the left side of the slide consisting of two overlapping parallelograms. The front one is blue and the back one is light green. Both are tilted at an angle.

Shortest Path Algorithms (Single Source)

- Harisam Sharma




Single Source Shortest Path Problem

- We are given a node called source, and we are interested in finding the length of the shortest path from source to all other nodes of the graph.
- In case the graph has negative cycles, the notion of shortest paths does not make any sense, since we can repeatedly loop around it to make distances $-\infty$.
- We have 2 standard algorithms for this, Dijkstra's Algorithm, and Bellman Ford Algorithm.



Dijkstra's Algorithm

- This is a Greedy Algorithm
- We will maintain a set of those elements whose shortest distances from the source have not been finalized, along with the length of their shortest paths from the source, using only those nodes whose distances have been finalized, in an array $\text{dist}[]$.
- Naturally, at each moment, We can pick the node 'u', with the smallest value of $\text{dist}[u]$ and claim that this is the final shortest distance of u from the source, remove u from the set, and change $\text{dist}[v]$ for all v whose distances are affected by u. (adjacent nodes of u!)



```
vector<ll>dijkstra(ll source)
{
    vll dist(n+1,inf);
    dist[source] = 0;
    set<pair<ll,ll>>st;
    st.insert({0,source});

    while(!st.empty())
    {
        pair<ll,ll>pr = *st.begin();
        st.erase(st.begin());
        ll curr_node = pr.second;
        ll curr_dist = pr.first;
        for(auto x : adj[curr_node])
        {
            ll node_here = x.first;
            ll edge_wt = x.second;
            if(curr_dist + edge_wt < dist[node_here])
            {
                st.erase({dist[node_here],node_here});
                dist[node_here] = curr_dist + edge_wt;
                st.insert({dist[node_here],node_here});
            }
        }
    }
    return dist;
}
```



Proof of correctness(Informal)

- According to our algorithm, all the nodes that we erase from the set (and finalize their shortest distances) have distances smaller than the ones that are still in the set.
- Now, obviously, the shortest path from the source to a node u will only contain those nodes v such that $\text{dist}[v] < \text{dist}[u]$. So, we can Confidently say that out all the nodes u in the set, the one with the smallest value of $\text{dist}[u]$ can be removed from it, and its shortest Distance finalized.
- Time Complexity: $O((N + M)\log N)$.
- Note: Dijkstra does **NOT** work if there are negative edges in the graph. (The argument in point 2 fails in this case)




Bellman Ford Algorithm

- To solve the SSSP Problem in case of negative weights (but not negative cycles), we can use this Algorithm.
- This algorithm is based on Dynamic Programming.
- It can also be used to detect whether the graph has any negative cycles or not.



Bellman Ford Algorithm

- The state of our DP is quite simple:
 - $Dp[i][j]$ = Length of the shortest path from the source to node i , such that we have used $\leq j$ edges in total.
- Catch: We will never use more than $n-1$ edges in any shortest path.



```
vector<ll>bellman_ford(ll source)
{
    vll dist(n+1,inf);
    dist[source] = 0;

    fo(i,1,n-1)
    {
        for(auto e : edges)
        {
            if(dist[e.first_node] + e.weight < dist[e.second_node])
            {
                dist[e.second_node] = dist[e.first_node] + e.weight;
            }
        }
    }
    return dist;
}
```

Time Complexity: $O(N*M)$



Detecting Negative Cycles

- We can continue the algorithm for 1 more iteration. I.e. N iterations instead of $N-1$ iterations.
- If any node's distance decreases in the N 'th iteration, we can conclude that the graph possesses negative weight cycles.
- This is because, if $\text{dist}[u][N] < \text{dist}[u][N-1]$, is only possible in case of a negative cycle.