Graph with Negative Weight

1 The Shortcoming of Dijkstra

Dijkstra is a **search algorithm**, which means it starts from one vertex, and explores the graph by walking along edges and marking vertices. The algorithm maintains the invariant that when we process remove the vertex u from the set (or whatever structure you are using to keep track of the next vertex to process) we have already found the shortest path from the source to u. Furthermore, we will not find any path from the source to u with less cost in the future. In a graph with negative weight edge, this invariant can no longer be maintained. This is because after processing vertex u, you might discover a negative weight edge to u that might yield a path with less cost.

2 Bellman Ford Algorithm

Bellman Ford is an algorithm to solve the single source shortest path problem on graphs with negative weight edge. The core of the algorithm is the following function:

```
void relax(int u, int v) {
  if (dist[v] >= dist[u] + weight[u][v])
    dist[v] = dist[u] + weight[u][v];
}
```

The main idea is that if the edge (u, v) is the last edge of the shortest path to v, the the cost of the shortest path to v is the cost of the shortest path to u plus the weight of (u, v). Thus, if we iterate through every edge (u, v) in the graph and call `relax(u, v)`, then we would have found the shortest path for every vertex whose shortest path consists of only one edge. If we build upon this solution and call `relax(u, v)` again for every edge, then we would have found the shortest path for every vertex whose shortest path consists of two edges or less. Continuing in this fashion, after the k^{th} iteration, we would have found the shortest path for every vertex whose shortest path consist of k edges or less. Since the shortest path in a graph with V vertices and E edges has at most $V - 1$ edges, we only need to repeat the process at most $V - 1$ time.

```
int dist[128];
vector<int> graph[128]; // adjacency list
vector<int> cost[128];  // cost[i][j] = cost of edge from i to graph[i][j]

void bellman_ford(int s) {
  memset(dist, 0x3f, sizeof(dist));
  dist[s] = 0;

  for (int i = 0; i < V - 1; ++i)
    for (int j = 0; j < V; ++j)
      for (int k = 0; k < graph[j].size(); ++k)
        dist[graph[j][k]] = min(dist[graph[j][k]], dist[j] + cost[j][k]);
}
```

The Bellman Ford algorithm iterate through each edge V times, and its time complexity is $O(VE)$. Note that after the k^{th} iteration in the algorithm, we may have found some shortest paths with more than k edges. However, we are only guaranteed to find shortest paths with less than or equal to k edges.

3 Negative Weight Cycle

A subtlety of computing shortest path in a graph with negative weight is the existence of negative weight cycle. In this case, we may get as little cost as possible by simply traversing the negative weight cycle indefinitely. Thus, it's a good idea to be able to detect negative weight cycles. The Bellman Ford algorithm can be easily extended to do so. Note that if the graph does not contain negative weight cycle, then after $V - 1$ iteration, we would not be able to relax any edges any further. Thus, if we iterate through every edge again and call `relax(u, v)` and the shortest path to one of the vertex decreases, then a negative weight cycle exists.

4 Floyd Warshall and Negative Edges

It is very fortunate that the Floyd Warshall algorithm actually handles the negative weight edges correctly. Thus, we can still use this algorithm for all pair shortest path on negative weight graph. It is also very easy to detect negative weight cycle with Floyd Warshall. Essentially, we iterate through every vertex v and check whether `dist[v][v] < 0`. If such a vertex exists, then a negative weight cycle exists.