|  |
| --- |
| // Implementation of Dijkstra's algorithm using adjacency lists |
|  | // and priority queue for efficiency. |
|  | // |
|  | // Running time: O(|E| log |V|) |
|  |  |
|  | #include <queue> |
|  | #include <stdio.h> |
|  |  |
|  | using namespace std; |
|  | const int INF = 2000000000; |
|  | typedef pair<int,int> PII; |
|  |  |
|  | int main(){ |
|  |  |
|  | int N, s, t; |
|  | scanf ("%d%d%d", &N, &s, &t); |
|  | vector<vector<PII> > edges(N); |
|  | for (int i = 0; i < N; i++){ |
|  | int M; |
|  | scanf ("%d", &M); |
|  | for (int j = 0; j < M; j++){ |
|  | int vertex, dist; |
|  | scanf ("%d%d", &vertex, &dist); |
|  | edges[i].push\_back (make\_pair (dist, vertex)); // note order of arguments here |
|  | } |
|  | } |
|  |  |
|  | // use priority queue in which top element has the "smallest" priority |
|  | priority\_queue<PII, vector<PII>, greater<PII> > Q; |
|  | vector<int> dist(N, INF), dad(N, -1); |
|  | Q.push (make\_pair (0, s)); |
|  | dist[s] = 0; |
|  | while (!Q.empty()){ |
|  | PII p = Q.top(); |
|  | if (p.second == t) break; |
|  | Q.pop(); |
|  |  |
|  | int here = p.second; |
|  | for (vector<PII>::iterator it=edges[here].begin(); it!=edges[here].end(); it++){ |
|  | if (dist[here] + it->first < dist[it->second]){ |
|  | dist[it->second] = dist[here] + it->first; |
|  | dad[it->second] = here; |
|  | Q.push (make\_pair (dist[it->second], it->second)); |
|  | } |
|  | } |
|  | } |
|  |  |
|  | printf ("%d\n", dist[t]); |
|  | if (dist[t] < INF) |
|  | for(int i=t;i!=-1;i=dad[i]) |
|  | printf ("%d%c", i, (i==s?'\n':' ')); |
|  |  |
|  | return 0; |
|  | } |