```cpp
#include <cstdio>
#include <vector>
#include <queue>
using namespace std;

typedef pair<int, int> ii;
typedef vector<int> vi;
typedef vector<ii> vii;
#define INF 1000000000

int main() {
  int V, E, s, u, v, w;
  vector<vii> AdjList;

  /*
  // Graph in Figure 4.17
  5 7 2
  2 1 2
  2 3 7
  2 0 6
  1 3 3
  1 4 6
  3 4 5
  0 4 1
  */

  freopen("in_05.txt", "r", stdin);

  scanf("%d %d %d", &V, &E, &s);

  AdjList.assign(V, vii()); // assign blank vectors of pair<int, int>s to AdjList
  for (int i = 0; i < E; i++) {
    scanf("%d %d %d", &u, &v, &w);
    AdjList[u].push_back(ii(v, w));                              // directed graph
  }

  // Dijkstra routine
  vi dist(V, INF); dist[s] = 0;                        // INF = 1B to avoid overflow
  priority_queue< ii, vector<ii>, greater<ii> > pq; pq.push(ii(0, s));
                             // ^to sort the pairs by increasing distance from s
  while (!pq.empty()) {                                               // main loop
    ii front = pq.top(); pq.pop();      // greedy: pick shortest unvisited vertex
    int d = front.first, u = front.second;
    if (d > dist[u]) continue;    // this check is important, see the explanation
    for (int j = 0; j < (int)AdjList[u].size(); j++) {
      ii v = AdjList[u][j];                           // all outgoing edges from u
      if (dist[u] + v.second < dist[v.first]) {
        dist[v.first] = dist[u] + v.second;                    // relax operation
        pq.push(ii(dist[v.first], v.first));
  } } }  // note: this variant can cause duplicate items in the priority queue

  for (int i = 0; i < V; i++) // index + 1 for final answer
    printf("SSSP(%d, %d) = %d\n", s, i, dist[i]);

  return 0;
}
```