```
#include <iostream>
#include <vector>
#include <queue>
using namespace std;

struct Node {
    int val;  // value of the node
    int dist;  // distance from the starting point
    int heuristic;  // estimated distance to the destination
    bool operator<(const Node& other) const {
        return dist + heuristic > other.dist + other.heuristic;
    }
};

vector<vector<pair<int, int>>> adj_list;
vector<int> dist;
vector<int> heuristic;
vector<bool> visited;
int start, dest;
```

```
void a_star() {
    priority_queue<Node> q;
    q.push({start, 0, heuristic[start]});

    while (!q.empty()) {
        Node node = q.top();
        q.pop();

        if (node.val == dest) {
            cout << "Shortest path: " << dist[dest] <<
endl;
            return;
        }

        if (visited[node.val]) continue;
        visited[node.val] = true;

        for (auto it : adj_list[node.val]) {
            int neighbor = it.first;
            int weight = it.second;
            if (visited[neighbor]) continue;

            int new_dist = dist[node.val] + weight;
            if (new_dist < dist[neighbor]) {
                dist[neighbor] = new_dist;
                q.push({neighbor, new_dist,
heuristic[neighbor]});
            }
        }
    }

    cout << "No path found!" << endl;
}

int main() {
    int n, m;
    cin >> n >> m;

    adj_list.resize(n);
    dist.resize(n, INF);
    heuristic.resize(n);
    visited.resize(n);
    // input the starting point and destination
    cin >> start >> dest;

    // input the edges
    for (int i = 0; i < m; i++) {
        int a, b, w;
        cin >> a >> b >> w;
        adj_list[a].push_back({b, w});
    }

    // input the heuristics
    for (int i = 0; i < n; i++) {
        cin >> heuristic[i];
    }

    dist[start] = 0;
    a_star();
    return 0;
}
```