```cpp
// Implementação confiavel de Dijkstra,
// usada no problema URI 1931 - Mania de Par

#include<stdio.h>  // entrada e saida padrao
#include<string.h> // memset()
#include<queue>    // priority_queue
#include<utility>  // pair<int, int>
#include<vector>   // container usado pela priority_queue
#define MAX 11234  // LEMBRE-SE DE SETAR O MAX PARA CADA PROBLEMA!!!!
#define INFTO 1123456789 // 10^9 (1 bilhao), para nao ficar contando zeros

using namespace std;

typedef pair<int, int> ii;

int N, E;          // N = vertices, E = arestas
int LG[MAX][MAX]; // Lista de adjacencia: LG[u][i] eh o i-esimo vizinho de u
int d[MAX];        // Qtd. de arestas partindo de u
int AG[MAX][MAX]; // Pesos das arestas: AG[u][w] eh o peso da aresta de u a w
int dist[MAX];     // lista de distancias a partir de s

// Computa todas as distancias a partir de um vertice s
// em um grafo com pesos nao-negativos nas arestas
void dijkstra(int s) {
    int u, w, i;
    priority_queue<ii, vector<ii>, greater<ii> > pq;
    for (u = 1; u <= N; u++) {
        dist[u] = INFTO;
    }
    dist[s] = 0;               // distancia da origem ate a origem = zero
    pq.push(ii(0, s));         // enfileiramos a origem, com distancia zero
    while(!pq.empty()){        // enquanto a fila de prioridade nao estiver vazia
        u = pq.top().second; pq.pop();   // retiramos o vertice mais proximo
        for(i = 0; i < d[u]; i++){       // para cada vizinho w de u
            w = LG[u][i];    // se a distancia ate u + o peso da aresta de u a w
            if(dist[u] + AG[u][w] < dist[w]){ // for menor que a distancia atual
                dist[w] = dist[u] + AG[u][w]; // relaxamos a aresta de u a w
                pq.push(ii(dist[w], w));      // e enfileiramos w e sua dist.
            }
        }
    }
}

int main(void){
    int i, u, w, peso;
    while(scanf("%d %d", &N, &E) != EOF){
        memset(d, 0, sizeof(d));
        for(i = 1; i <= E; i++){
            scanf("%d %d %d", &u, &w, &peso);
            LG[u][d[u]++] = w;
            LG[w][d[w]++] = u;
            AG[u][w] = AG[w][u] = peso;
        }
        // computa todas as distancias a partir do vertice 1
        dijkstra(1);
        printf("%d\n", dist[N] != INFTO ? dist[N] : -1);
    }
    return 0;
}
```