import java.util.\*;

class OSPF {

private int vertices;

private LinkedList<Edge>[] adjList;

static class Edge {

int target, weight;

Edge(int target, int weight) {

this.target = target;

this.weight = weight;

}

}

OSPF(int vertices) {

this.vertices = vertices;

adjList = new LinkedList[vertices];

for (int i = 0; i < vertices; i++) {

adjList[i] = new LinkedList<>();

}

}

void addEdge(int source, int target, int weight) {

adjList[source].add(new Edge(target, weight));

adjList[target].add(new Edge(source, weight));

}

void dijkstra(int source, int target) {

PriorityQueue<Edge> priorityQueue = new PriorityQueue<>(vertices, Comparator.comparingInt(e -> e.weight));

int[] distances = new int[vertices];

Arrays.fill(distances, Integer.MAX\_VALUE);

distances[source] = 0;

priorityQueue.add(new Edge(source, 0));

while (!priorityQueue.isEmpty()) {

int u = priorityQueue.poll().target;

for (Edge edge : adjList[u]) {

int v = edge.target;

int weight = edge.weight;

if (distances[u] + weight < distances[v]) {

distances[v] = distances[u] + weight;

priorityQueue.add(new Edge(v, distances[v]));

}

}

}

System.out.println("Shortest path from vertex " + source + " to vertex " + target + " is " + distances[target]);

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

OSPF graph = new OSPF(6);

graph.addEdge(0, 1, 4);

graph.addEdge(0, 2, 3);

graph.addEdge(1, 2, 1);

graph.addEdge(1, 3, 2);

graph.addEdge(2, 3, 4);

graph.addEdge(3, 4, 2);

graph.addEdge(4, 5, 6);

System.out.print("Enter the source vertex: ");

int source = scanner.nextInt();

System.out.print("Enter the target vertex: ");

int target = scanner.nextInt();

graph.dijkstra(source, target);

}

}