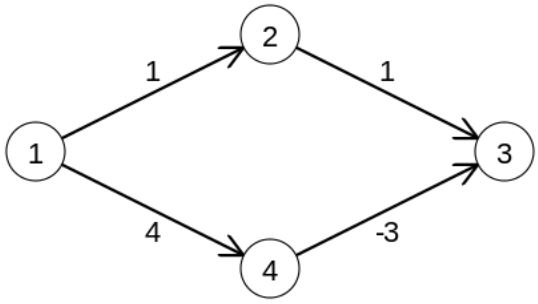
[A13] DijkstraLand



Was ist der Dijkstra Algorithmus?
Wie schon in A11 beschrieben:
Der Dijkstra Algorithmus sucht innerhalb eines gewichteten Graphen den kostengünstigsten Weg zwischen zwei Knoten. Von einem Startknoten ausgehend werden die, im nächsten Schritt erreichbaren, Knoten mit dem geringsten Gewicht, ausgehend von den bereits erreichten Knoten, abgearbeitet. In jedem Durchlauf wird aus der Menge der erreichbaren Knoten, der Knoten mit den geringsten Kosten aus der Menge der bereits erreichten Knoten hinzugefügt. Dadurch entsteht ein neuer Graph mit den minimalsten Entfernungen zwischen allen Knoten.

In unserem DijkstraLand haben wir mit einem Heap gearbeitet. Somit müssen alle Knoten abgearbeitet werden und jeder Knoten muss aus dem Heap geholt werden. Es muss für jede entdeckte Kante das Gewicht im Heap aktualisiert werden.

Laufzeit: O((V+E)log(V))

Wichtige Elemente:

```
public static List<Integer> dijkstra(Graph graph, int from, int to) {

   int[] predecessor = new int[graph.numVertices()];
   int[] distance = new int[graph.numVertices()];

   VertexHeap heap = new VertexHeap();

   for(int i=0; i < graph.numVertices(); i++) {    // loop to add all
vertices to heap and set initial values in arrays
      distance[i] = Integer.MAX_VALUE;          // set for all vertices the
distance to max integer value
      predecessor[i] = -1;                    // set predecessor to -1 (=no
predecessor set)
      heap.insert(new Vertex(i, distance[i]));    // add new Vertex / Node
to heap and set initial distance to max integer value (from distance array)
   }

   distance[from] = 0;              // set distance for starting node to 0
   heap.setCost(from, 0);     // set cost/priority for starting node to 0 in
heap
```

```
   //heap.printHeap();

   while (!heap.isEmpty()) {     // loop while heap is not empty
      Vertex v = heap.remove();  // get top element (smallest
priority/cost) and remove it from heap

      for (WeightedEdge edge : graph.getEdges(v.getVertex())) {  // iterate
through list of edges from current element (removed from top of heap)

         int borderCost = 0;                                              //
reset border cost to 0

if(!graph.getLand(v.getVertex()).equals(graph.getLand(edge.vertex)))    //
check if target is in another country
            borderCost = BORDER_CROSSING_COST;                            //
set cost for border if border is crossed due different countries

         if((edge.weight + v.getCost() + borderCost) <
distance[edge.vertex]){  // check if way is better (cost) than existing one
            predecessor[edge.vertex] = v.getVertex();                      //
set predecessor node
            distance[edge.vertex] = edge.weight + v.getCost() + borderCost;
// distance to target node calculated from edge weight and distance and
border cost to current node
            heap.setCost(edge.vertex, distance[edge.vertex]);
// set new priority / cost in heap
         }
      }
   }
```