Eidgenössische
Technische Hochschule
Zürich

Ecole polytechnique fédérale de Zurich
Politecnico federale di Zurigo
Federal Institute of Technology at Zurich

Department of Computer Science
Markus Püschel
Peter Widmayer
Thomas Tschager
Tobias Pröger
Tomáš Gavenčiak

22th December 2016

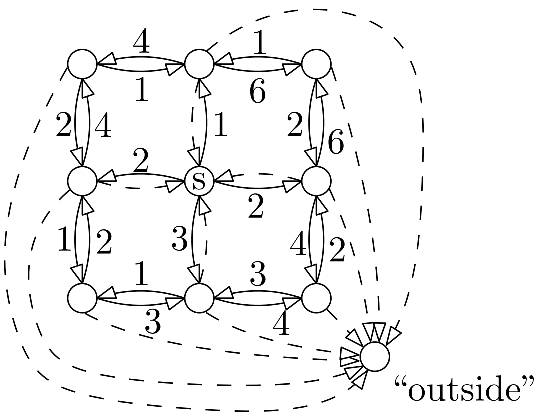
Data Structures & Algorithm   Solutions to Sheet P13   AS 16

Solution P13.1 *Jungle.*

The task can be rephrased as finding a shortest path in a weighted 2D *grid graph* from the starting position to a special vertex called "outside". The new graph has edges directed in both directions, and the cost of edge $(x, y) \to (x', y')$ has the cost of entering (x', y'). See the image below for an illustration, the dashed edges have cost 0.

4	1	6
2	0	2
1	3	4

$\longrightarrow$



Dijkstra's algorithm is an ideal solution for this problem and it is not difficult to implement it over this graph. We do not have to even "construct" the graph explicitly – we can represent every vertex by its coordinates (x, y) and when we need to visit the neighbors of vertex (x, y), we know these are $(x+1, y)$, $(x-1, y)$, $(x, y+1)$ and $(x, y-1)$ (some of these may be the "outside vertex" if (x, y) is on the jungle edge). Moreover, when we first visit the "outside" vertex, we may stop the computation, since we already found the shortest path.

Dijkstra's algorithm on this kind of graph is not difficult to implement, but we need a min-heap structure for the vertices to be visited. We can implement our own heap or use the `java.util.PriorityQueue` structure provided by the Java library. The `PriorityQueue` guarantees that the `add()` and `remove()` (applied to the minimum) operations take $\mathcal{O}(\log n)$ time, but not removing a general element, as we may not change the weights of the elements in the heap. We can resolve this by adding every visit of a vertex into the heap, possibly having multiple occurrences of a single vertex in the heap. When we pop a minimal vertex from the heap, we check if we already removed another occurence of it from the heap and if so, we ignore it.

This does not change the total algorithm complexity as the heap size will be at most m, we have $\log m = \mathcal{O}(\log n)$ and there is at most one vertex occurrence per a directed edge. We also need to implement out own class `SquareInHeap` to represent an occurrence of a vertex in the heap and implement a comparator for this class to use in the `PriorityQueue`. Implementing your own heap actually removes many of those small problems with the cost of having more code.

Solution programs On the lecture website, you can find example solution sources using the above idea with further comments on the implementation.

Data Similarly to the `test*` cases, test judge data are corner cases with the start on the edge of the jungle, large random squares and then several jungles with a high-price “wall” around the jungle. This wall ensures that the entire internal ”cheap” jungle will be explored before an expensive solution going over the fence is found.