CMPT 409 Assignment 5

Heather Li, Ekjot Singh Billing, Manshant Singh Kohli

March 25, 2018

1 Firetruck

(C++11) This problem can be reduced to printing a list of all the routes between firehouse 1 and the designated firehouse n. If we represent each firehouse with a vertex, the problem input can be seen as a list of edges. We can use these edges to build our graph. Note that the restriction that no firehouse (vertex) is visited more than once makes this a path.

To make the process of generating all paths easier, we use a BFS to determine all firehouses reachable from firehouse n and store them in a bitmasked integer.

After this pre-processing, we can then use a DFS to list all the valid paths the firetrucks can take.

2 Pushing Boxes

(C++ 11) In this problem we need to find the path that minimizes box pushes. If multiple paths exists with same minimum box pushes, we return the path with the least overall moves. This can be solved with a greedy graph traversal.

We are using a nested graph traversal. For the outer traversal, we are using A-star (kind of; its mostly like BFS except uses priority queue) for box movements with the heuristic of (number of box pushes, total movements). For each of these box movements we are running an inner BFS traversal on the player to find the minimum moves required to get into the correct position to push the box. We keep track of the current best route found.

As soon as the next element in priority queue has greater or equal value than (number of box pushes, total movements) of our current best route found, we know we can't get a better solution and therefore print the path.

3 The Necklace

(C++ 11) Consider this problem as a graph. We note that if we represent each colour as a vertex, each bead with colours (a, b) can be represented by an edge from vertex a to vertex b. If two edges are adjacent (share an end vertex), then it is possible to join the corresponding beads together in the necklace.

As such, if an Eulerian circuit is present in this graph (if there is a path that uses each path in the graph exactly one), it is possible to create a necklace using all the beads. Eulerian circuits exist in a graph iff every vertex has even degree and if all non-zero degree vertices are connected.

We check these two conditions. If one of these cannot be met, then we say that 'some beads must be lost'. Otherwise, we return the Eulerian circuit that must be present in the graph.

To do so, we note that Eulerian circuits are simply collections of cycles, where each cycle joins another cycle at exactly one point. We construct cycles by following paths until a node is repeated. (By the properties of our graph, this is guaranteed). By doing so, we can construct our Eulerian circuit by joining cycles. We can print the sequence of beads by following the path of our Eulerian circuit.

4 Highways

(C++ 11) We note that the shortest valid highway system is equivalent to the minimum spanning tree on the complete graph of towns. In the graph, the vertices are towns and the weight of an edge between two towns represents the Cartesian distance between the two towns.

Using this representation, we can use Kruskal's algorithm to determine the minimum spanning tree. We put every edge of our complete graph into a priority queue sorted in ascending order of weight.

We then run Kruskal's algorithm, and use union-find to determine whether towns are connected by edges already present in our tree. The final minimum spanning tree represents the optimal highway system to be built.