Chair for Algorithms
and Data Structures
Prof. Dr. Hannah Bast
M. Brodesser/E. Sawin

Efficient Route Planning
SS 2012

http://ad-wiki.informatik.uni-freiburg.de/teaching

Exam

Monday, August 20, 2012, 2.00 pm - 3.30 pm, HS 026 in building 101

General instructions:

There are four tasks, of which you can select *three tasks of your choice*. Each task is worth 20 points. If you do all four tasks, we will only count the best three, that is, you can reach a maximum number of 60 points. You need 30 points to pass the exam. The exact correspondence between the number of points and the grade will be decided when we correct your exams. You have one and a half hours of time overall, that is, 30 minutes per task on average.

You are allowed to use any amount of paper, books, etc. You are not allowed to use any computing devices or mobile phones, in particular nothing with which you can communicate with others or connect to the Internet.

On every page of your solutions, please write your Matrikelnummer and your name IN PRINTED LETTERS, and number the pages consecutively!

Good luck!

Task 1 (Set Dijkstra, 20 points)

Consider the following grid graph (drawn in ASCII), with 25 nodes (drawn as `A`, `B`, `C`, `D`, or `o`) and 40 undirected arcs (drawn as `|` or `---`). The cost of each arc is 1.

```
A---o---o---o---B
|   |   |   |   |
o---o---o---o---o
|   |   |   |   |
o---o---o---o---o
|   |   |   |   |
o---o---o---o---o
|   |   |   |   |
C---o---o---o---D
```

1.1 (6 points)

Use the copy of the graph from the separate sheet and simulate a run of *Set* Dijkstra, with all four nodes `A`, `B`, `C`, `D` in the source set. Settle all nodes. It should be clearly visible from your drawing: (1) which nodes were settled in which order (mark the settled nodes and write the order number nearby); (2) the tentative and final distances computed. If you can, use colors!

1.2 (6 points)

From the computation of task 1.1, determine the node with the largest distance from the source set. Then generalize this, by considering an arbitrary $n \times n$ grid graph of the kind above, and determining *all* nodes with the largest distance from the four corner points for that graph. Provide an argument that your solution is correct. A formal proof is not needed. Refer to the nodes using their "coordinates" in the grid; for example, in the graph from task 1.1, the upper left node is $(1, 1)$ and the lower right node is $(5, 5)$.

1.3 (8 points)

Write a function *Array<int> computeFarthestNodes(Graph graph, int k)*, that for a given *arbitrary* graph and a given integer k, computes a set of k nodes $u_1, \ldots, u_k$ that are as far apart as possible, namely $\text{dist}(\{u_1, \ldots, u_{i-1}\}, u_i) = \max_u \text{dist}(\{u_1, \ldots, u_{i-1}\}, u)$ for all $i = 2, \ldots, k$. The choice of u_1 is up to you. Just for your understanding: recall that this function can be used to compute a given number of reasonable landmarks.

You can assume that there is a function *Array<int> setDijkstra(Graph graph, Array<int> nodes)* that for a given graph returns the distances of all nodes to the given node set. Also, you can assume, as in the exercises, that the *Graph* class has a member *Array<Array<Arc>> adjacencyLists*.

You can write your code in Java or in C++. Syntactic correctness is not important, but your code should work in principle.

Task 2 (Arc Flags and A-Star, 20 points)

2.1 (14 points)

Implement the query algorithm for arc flags. Namely, write a function *int computeShortest-Path(Graph graph, int source, int target)* that returns the cost of the shortest path in the given graph from the given source to the given target. Recall that the query algorithm is a variant of Dijkstra's algorithm. As in the exercises, you can assume that the *Graph* class has a member *Array<Array<Arc>> adjacencyLists* and that the *Arc* class has a member *bool arcFlag* that is true iff the flag for that arc is set. You can also assume that there is a standard priority queue data structure available.

2.2 (6 points)

Extend your function from task 2.1 to also consider an A-star like heuristic function, given as an additional argument *Array<int> heuristic*. Don't scribble in your function from task 2.1 (it will become a mess). Instead, mark the lines in your function from task 2.1 and for this task make a separate list, where you write down how those marked lines should be replaced.

As for task 1.3, also for 2.1 and 2.2 you can write your code in Java or in C++. And again, syntactic correctness is not important, but your code should work in principle.

Task 3 (Contraction Hierarchies, 20 points)

3.1 (4 points)

Draw the following graph with two concentric circles. The outer circle has four nodes $u_{1,1}, u_{1,2}, u_{1,3}, u_{1,4}$, connected as a ring, via four undirected edges. The inner circle has eight nodes $u_{2,1}, u_{2,2}, \ldots, u_{2,8}$, also connected as a ring, via eight undirected edges. The inner circle is connected to the outer circle as follows: for each $j = 1, \ldots, 4$, there is an undirected edge between $u_{1,j}$ and $u_{2,2j}$. All arc costs are 1.

3.2 (8 points)

Contract all nodes from the graph from task 3.1 in an order such that the number of shortcuts added is minimized. Provide an argument why the number of shortcuts is indeed minimal. If you have problems with this argument, at least argue why no contraction order with zero shortcuts exists. Clearly mark the order in which you contract the nodes, and the shortcuts added. If possible, use colors.

3.3 (8 points)

Perform the Contraction Hierarchies query from node $u_{1,1}$ to $u_{2,1}$ as follows. Perform the appropriate Dijkstra computation in the upward graph starting at $u_{1,1}$ as well as the appropriate Dijkstra computation in the upward graph starting at $u_{2,1}$. You don't have to put all details of the Dijkstra computation, it is enough if you write the correct distances at the nodes. Use the result from these Dijkstra computations to determine the cost of the optimal path.

Task 4 (Transit Networks, 20 points)

Consider the following transit network. There are five stations, named E, W, S, N (after the four directions) and C (for center).

There is an east-west line $L1$ going E, C, W, C, E. It commutes every hour, starting at 8:00, taking 10 minutes per leg (from one station to the next), 5 minutes layover time at each station (time between arrival at that station and then departure again).

There is a north-south line $L2$ going N, C, S, C, N. It also commutes every hour, starting at 8:00, taking 10 minutes per leg, 5 minutes layover time at each station.

There is a special line $L3$ going S, W, S. It only commutes once at 8:00, taking 20 minutes per leg, 10 minutes layover time.

The transfer buffer is 5 minutes for all transfers at all stations.

4.1 (6 points)

Draw all nodes of the time-expanded transit network pertaining to the center station C and with time between 8:00 and 8:30, as well as all adjacent nodes, and all arcs between all those nodes. Mark each node with the station name, the time, and its type (departure, arrival, transfer). For example, C_{arr} @ 8:10 or C_{dep} @ 8:15.

4.2 (6 points)

Determine the optimal transfer patterns from S to any other station. Provide a short justification for each station pair.

4.3 (8 points)

Now consider C as an important station. Determine the optimal *local* transfer patterns for S (pertaining to all optimal paths until a transfer at an important station or without any transfer at an important station), as well as the *global* transfer patterns for C (pertaining to all optimal paths to all other stations).

Using these (local and global) transfer patterns, build the query graph for a query from S to W. You don't have to perform a search on that graph, just drawing the graph (with a brief explanation of how you constructed it) is enough.