CS 344 Assignment 4

Craig Perkins, Alex Tang, Steve Grzenda

Due April 20, 2014

Problem 1

A. Another way to think of this problem is to present the problem in terms of a graph G, where each vertex is an intersection and each edge is a street. This way, a dead end is viewed as a vertex with degree 1 and a n-way intersection, where n is odd, is a vertex with odd degree. Then, the proof that an Eulerian circuit C exists if and only if every vertex has even degree goes as follows:
$\Rightarrow$ If an Eulerian circuit C exists, then every vertex has even degree.
Pick any vertex v. Then C contains all the edges adjacent to v. As we traverse C, we must enter and leave v the same number of times, so $\deg(v)$ must be even.
$\Leftarrow$ If every vertex has even degree, then an Eulerian circuit C exists.
We prove this by induction on the number of edges in G.
Base case: The smallest possible graph where every vertex has even degree is a graph with two vertices with two edges between them. In the case of simple graphs, the smallest possible graph is a triangle, i.e., a complete graph with 3 vertices.
Let the induction hypothesis be: Let H be a connected graph with $\leq k$ edges.
If every vertex of H has even degree, then H contains an Eulerian circuit.
Consider G, a graph with $k+1$ edges, where every vertex has even degree. Since G has no vertices with odd degree, G is not a tree so it must contain a cycle W. Remove the edges of W from G to create a graph G' which may be disconnected consisting of connected components $G'_1, G'_2, \ldots, G'_k$. Observe that removing the edges of W causes each vertex in W to lose an even number of edges so the parity of each vertex is unchanged in G'. By the induction hypothesis, each of $G'_1, G'_2, \ldots, G'_k$ must contain its own Eulerian circuit $W'_1, W'_2, \ldots, W'_k$. We can now to construct an

Eulerian circuit for G. Pick any vertex v from W. Traverse along W until we reach a vertex v_i belonging to a connected component G'_i. Then traverse along the Eulerian circuit W'_i until we return to v_i. Repeat this along W until we return to v. This completes an Eulerian circuit in G.

B. To find an Eulerian circuit, use Hierholzer's algorithm which goes as follows:

1. Make sure every vertex in the graph G has even degree.
2. Pick any vertex v.
3. Starting at v, follow any cycle of edges until you return to v, marking each edge as visited. If this cycle covers all the edges in G, then you are done.
4. If not, then as long as there exists a vertex u that belongs to the current circuit but has adjacent edges that are not visited, start another cycle from u and joining this cycle to the previous circuit.
5. Repeat until all edges are visited.

Hierholzer's algorithm will run in $O(|E|)$ time.

Problem 2

A. To solve the problem of starting with an amount of substance A and producing the max amount of substance B given a conversion matrix and n substances. We can construct a complete directed graph where the nodes of the graph are the substances and the edge from chemical A to chemical B has a weight equal to that of how much of chemical A it takes to produce 1 of chemical B. To find the optimal sequence of conversions to produce the most of chemical B we want to find the max product along a path on the graph from A to B. This problem can be solved as a modification on a typical Dijkstra's algorithm by computing product along paths instead of the sum along paths. Once the algorithm terminates after each node has been labeled with the optimal way of producing chemical B from chemical A. The algorithm differs from Dijkstra's in that the initiation starts by labeling the source chemical as 1 and the rest as 0. Also Node B's label gets updated if $w_i \cdot R_{iB} > w_b$, which means Chemical B's label is updated if we can find a sequence to produce more of B from A. After the algorithm terminates the label of B will be the maximum way to convert A to B and you can follow the predecessors of B to A to find the optimal sequence of conversions.

Since the graph is dense, $|E| = |V|^2$, it is best to use the fibonacci heap implementation of Dijkstra's algorithm. The overall runtime will be on the order of $O(|V| \log |V| + |E|) = O(|V|^2) = O(n^2)$

B. In order to find a positive product cycle, we can run a modified Bellman-Ford algorithm. The algorithm will loop n times instead of n - 1 times. The modification on Bellman-Ford will be the same as the modification on Dijkstra's as described above. If the products do not converge after the n-1-th iteration, then a positive product cycle exists.

   The algorithm described above has a total runtime of $O(|V||E|) = O(|V|^3) = O(n^3)$.

Problem 3

```
Input:  Graph G = (V,E)
Output: For all vertices u reachable from s,
        dist(u) is set to the distance from s to u

for all u in V:
  dist(u) = INFINITY
  prev(u) = nil

dist(s) = 0
H = makequeue(V)

while H is not empty:
  u = deletemin(H)
  for all edges (u,v) in E:
    if v != destination:
      if dist(v) > dist(u) + getTime(u,v) + 1:
        dist(v) = dist(u) + getTime(u,v) + 1
        prev(v) = u
        decreaseKey(H,v)
    else:
      if dist(v) > dist(u) + getTime(u,v):
        dist(v) = dist(u) + getTime(u,v)
        prev(v) = u
        decreaseKey(H,v)
```

Where `getTime()` finds the next available truck leaving on the edge and

finds the difference between departure and arrival. We are assuming that lookup of the times and adding them will take constant time. The reason for the if/else statement is that if the vertex v is the destination then we will not be taking the extra hour to transfer between trucks. This algorithm is an implantation of Dijkstra's algorithm and will run in $O(|V| \log |V| + |E|)$ assuming we use a fibonacci heap as we do not know anything about the density of the trucks. The proof of correctness is the proof of Dijkstra's algorithm. By relaxing the edges we guarantee that each time we update edges it will be the optimal distance at that point.

Problem 4

A. Ye Olde Maester will suggest a modified version version of the mighty Floyd-Warshall algorithm. In the modified version the Maester does not care about the shortest paths, however, wants to find the max of the strengths of the preference sequences. As such he suggests that the subproblems for the optimal solution will be as follows: $t^*[i, j, k] = \max\{\min\{t^*[i, j, k-1], t^*[k, j, k-1]\}, t^*[i, j, k-1]\}$. The runtime of this algorithm will be on the order of $O(|V|^3)$ and the underlying data structure needed will be 2 matrices of size $|V| \times |V|$.

B. i.

```
d(BS,SC) = 17
d(BS,LT) = 19
d(BS,BT) = 24
d(BS,JL) = 29

d(SC,BS) = 28
d(SC,LT) = 15
d(SC,BT) = 14
d(SC,JL) = 12

d(LT,BS) = 26
d(LT,SC) = 30
d(LT,BT) = 22
d(LT,JL) = 20

d(BT,BS) = 21
d(BT,SC) = 26
d(BT,LT) = 23
d(BT,JL) = 27
```

```
d(JL,BS) = 16
d(JL,SC) = 33
d(JL,LT) = 25
d(JL,BT) = 13
```

We created a graph of 5 vertices labeled `BS`, `SC`, `LT`, `BT`, and `JL`. There was a directed edge with weight `d(A,B)` from `A` to `B` if `d(A,B) > d(B,A)`. Using this graph, we identified several cycles, indicating there is no undisputed winner.

ii.

```
$t^*$(BS,SC) = 29
$t^*$(BS,LT) = 25
$t^*$(BS,BT) = 24
$t^*$(BS,JL) = 29

$t^*$(SC,BS) = 28
$t^*$(SC,LT) = 25
$t^*$(SC,BT) = 24
$t^*$(SC,JL) = 28

$t^*$(LT,BS) = 28
$t^*$(LT,SC) = 30
$t^*$(LT,BT) = 24
$t^*$(LT,JL) = 28

$t^*$(BT,BS) = 26
$t^*$(BT,SC) = 27
$t^*$(BT,LT) = 25
$t^*$(BT,JL) = 27

$t^*$(JL,BS) = 28
$t^*$(JL,SC) = 33
$t^*$(JL,LT) = 25
$t^*$(JL,BT) = 24
```

iii. Brienne of Tarth comes out as the overall winner because she has a row of 1's in the T matrix. After discovering that she is the winner, her node gets plucked from the graph and all edges pointing to or away from her get removed from the graph and the algorithm

is repeated with the remaining nodes. Loras Tyrell comes in a respectable second, followed by the honorable Barristan Selmy. In fourth we have Jaime Lannister and Sandor Clegane comes in last.