CSE 542 – Advanced Data Structures and Algorithms *Jon Turner*

Lab 5

Due 5/2/2013

General notes for labs. Review and follow the general notes from lab 1.

In this lab, you'll be implementing Edmond's algorithm for weighted matching in bipartite graphs, and using it to implement Karp's algorithm for the asymmetric traveling salesman problem. In order to implement Edmond's algorithm, you'll need to extend the *d*-heap to implement the *addtokeys* operation. The parts of the lab are outlined below. You will find more details in the provided code and the lab report template.

- ***Part A.*** In this part, you will extend the *Dheap* data structure to include a constant time *addtokeys* operation. The repository includes a partial implementation of a derived class *Ddheap* that you will complete and test using a provided unit test.
- ***Part B***. In this part, you will implement two versions of Edmonds algorithm for weighted matchings in bipartite graphs. The first version, called *edmondsBW,* finds a max weight matching, using the algorithm described in the lecture notes. Your implementation should use the *Ddheap* data structure developed in part A.

  The second version of the algorithm, called *edmondsBWmin*, finds a *minimum weight* matching of maximum size. It differs from the first version in two ways. First, it starts by negating all the edge weights. This effectively causes the algorithm to find a min weight matching, instead of a max weight matching. Second, it does not halt when all the free vertices have zero labels. Instead, it continues so long as there are unexamined edges with either two even endpoints, or an even and an unreached endpoint.
- ***Part C***. In this part, you will be implementing Karp's algorithm for the asymmetric traveling salesman problem that was described in the notes. This algorithm starts by constructing a bipartite graph and finding a min weight matching in that graph. You will use *edmondsBWmin* for this. The matching is used to define a collection of cycles in the original graph. The algorithm then patches these cycles together, always combining a shorter cycle with the longest cycle in the collection. Your implementation should be designed to handle graphs that are not complete. For such graphs, it's possible for the algorithm to fail because there may not be any way to patch two cycles, using the available edges. If this case comes up, your program should print an error message and halt.
- ***Parts D, E.*** In this part, you will experimentally evaluate the performance of Edmond's matching algorithm and Karp's TSP algorithm.