Graph Algorithms II (part 2 of CSC 282),

http://www.cs.rochester.edu/~stefanko/Teaching/18CS282

IMPORTANT: **Write each problem on a separate sheet of paper. Write your name on each sheet.** (Why? To speed up the grading we will use parallelization—each problem will be graded by a different TA. At the beginning of the class (on the date when the homework is due) there will be a separate pile for each problem.)

Homework 3 due: Oct. 25 (Thursday); collected before class in Wegmans 1400.

Problem sessions before Homework 3 is due:

Oct. 22 (Monday) 6:15pm - 7:15pm in Hylan 202
Oct. 23 (Tuesday) 6:15pm - 7:15pm in Goergen 109
Oct. 24 (Wednesday) 6:15pm - 7:15pm in Gavett 310
Oct. 24 (Wednesday) 7:40pm - 8:40pm in Hutchinson 473

Exam 2 is on: Nov. 1 (Thursday).

Problem sessions before Exam 2:

Oct. 29 (Monday) 6:15pm - 7:15pm in Hylan 202
Oct. 30 (Tuesday) 6:15pm - 7:15pm in Goergen 109
Oct. 31 (Wednesday) 6:15pm - 7:15pm in Gavett 310
Oct. 31 (Wednesday) 7:40pm - 8:40pm in Hutchinson 473

1 Homework (solve and turn in)

3.1 (**due Thursday, Oct 25**) Suppose we want to run Dijkstra's algorithm on a graph whose edge weights are in the range $\{0, \ldots, W\}$. Show how Dijkstra's algorithm can be made to run in time $O(W|V| + |E|)$.

3.2 (**due Thursday, Oct 25**) We are given a weighted directed graph $G = (V, E, w)$. (The weights represent widths of roads in a road network.) The width of a path P is the minimum of the weights of edges in P. (Intuitively, the width of a path is the maximum width of an object that can be transported on that path without the object ever "sticking out".) We want to compute the widest path from s to t for every pair of vertices $s, t \in V$. Modify Floyd-Warshall algorithm to solve the problem.

2 282 Bonus Homework, 482 Solve 3.4 or 3.5 (or both) (solve and turn in)

Definitions needed for problems 3.4 - 3.5:

Let $G = (V, E)$ be an undirected graph. A *matching* of G is a set of edges $E' \subseteq E$ such that every vertex of G is in *at most one* edge of E'. A *perfect matching* of G is a set of edges $E' \subseteq E$ such that every vertex of G is in *exactly one* edge of E'.

Let $G = (V, E, w)$ be an undirected graph where w are edge-weights. Let $E' \subseteq E$ be a matching of G. The weight of E' is

$$\sum_{e \in E'} w(e).$$

We will consider the following two problems:

MAX-WEIGHT MATCHING:
INPUT: weighted graph $G = (V, E, w)$
OUTPUT: a matching $E' \subseteq E$ of G with the maximum weight.

MAX-WEIGHT PERFECT MATCHING:
INPUT: weighted graph $G = (V, E, w)$
OUTPUT: a perfect matching $E' \subseteq E$ of G with the maximum weight.

You need the following information for problems 3.4 and 3.5:

FACT: there is a polynomial-time algorithm to find a maximum-weight matching.

3.3 (**warm-up problem—do not turn in**) We are given n rectangles of sizes $a_1 \times b_1, \ldots, a_n \times b_n$. We want to build the highest tower out of the rectangles. In a tower, if a rectangle of width w is on top of a rectangle of width w' then we require $w \leq w'$. We are allowed to rotate the rectangles (i. e., an $a \times b$ rectangle can be changed into a $b \times a$ rectangle). Give an $O(n)$ algorithm which finds the height of the highest possible tower.
(For example if the input is $11 \times 11, 8 \times 2, 1 \times 10$ then the solution is a tower of height $29 = 11 + 8 + 10$.)

3.4 (**due Thursday, Oct 25**) Suppose that in Problem 3.3 we change the requirement $w \leq w'$ to $w < w'$, i. e., a rectangle on top of another rectangle has to be strictly thinner. Note that now it can happen that not all rectangles get used (e. g., if we have two squares of the same size). Give a polynomial-time algorithm for the modified problem. (Hint: try to cast the problem as MAX-WEIGHT MATCHING, see definitions below.)
(For example if the input is $2 \times 11, 2 \times 10, 10 \times 10$ then the solution is a tower of height $22 = 2 + 10 + 10$.)

3.5 (**due Thursday, Oct 25**) We have n jobs and m machines. We are given $n \times m$ table T where T_{ij} is the time to complete job i on machine j. Our task is to schedule the jobs on the machines to minimize the **average completion time** (that is, for each job you take the time when it finished and then average over all jobs). Give a polynomial-time algorithm for the problem. (Hint: try to cast the problem as minimum-weight matching of given cardinality.)
(For example if we have 3 jobs and 2 machines and

$$T = \begin{pmatrix} 1 & 3 \\ 3 & 2 \\ 4 & 1 \end{pmatrix},$$

then the following schedule is optimal:

- machine 1: 1;
- machine 2: 3, 2.

The completion times of jobs are: 1, 3, 1, the average completion time is 5/3.)