CS 473G: Combinatorial Algorithms, Fall 2005
Homework 5

Due Thursday, November 17, 2005, at midnight

(because you *really* don't want homework due over Thanksgiving break)

Name:	
Net ID:	Alias:
Name:	
Net ID:	Alias:
Name:	
Net ID:	Alias:
Name:	
Net ID:	Alias:

Homeworks may be done in teams of up to three people. Each team turns in just one solution; every member of a team gets the same grade.

Neatly print your name(s), NetID(s), and the alias(es) you used for Homework 0 in the boxes above. Attach this sheet (or the equivalent information) to the top of your solution to problem 1.

If you are an I2CS student, print "(I2CS)" next to your name. Teams that include both on-campus and I2CS students can have up to four members. Any team containing both on-campus and I2CS students automatically receives 3 points of extra credit.

Problems labeled ϒ are likely to require techniques from next week's lectures on cuts, flows, and matchings. See also Chapter 7 in Kleinberg and Tardos, or Chapter 26 in CLRS.

ϒ 1. Suppose you are asked to construct the minimum spanning tree of a graph G, but you are not completely sure of the edge weights. Specifically, you have a *conjectured* weight $\tilde{w}(e)$ for every edge e in the graph, but you also know that up to k of these conjectured weights are wrong. With the exception of one edge e whose true weight you know exactly, you don't know which edges are wrong, or even how they're wrong; the true weights of those edges could be larger or smaller than the conjectured weights. Given this unreliable information, it is of course impossible to reliably construct the true minimum spanning tree of G, but it is still possible to say something about your special edge.

Describe and analyze an efficient algorithm to determine whether a specific edge e, whose *actual* weight is known, is *definitely not* in the minimum spanning tree of G under the stated conditions. The input consists of the graph G, the conjectured weight function $\tilde{w} : E(G) \to \mathbb{R}$, the positive integer k, and the edge e.

2. Most classical minimum-spanning-tree algorithms use the notions of 'safe' and 'useless' edges described in the lecture notes, but there is an alternate formulation. Let G be a weighted undirected graph, where the edge weights are distinct. We say that an edge e is *dangerous* if it is the longest edge in some cycle in G, and *useful* if it does not lie in any cycle in G.

   (a) Prove that the minimum spanning tree of G contains every useful edge.

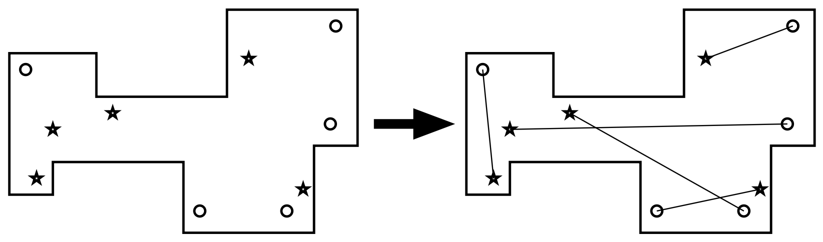
   (b) Prove that the minimum spanning tree of G does not contain any dangerous edge.

   (c) Describe and analyze an efficient implementation of the "anti-Kruskal" MST algorithm: Examine the edges of G in *decreasing* order; if an edge is dangerous, remove it from G. *[Hint: It won't be as fast as the algorithms you saw in class.]*

ϒ 3. The UIUC Computer Science department has decided to build a mini-golf course in the basement of the Siebel Center! The playing field is a closed polygon bounded by m horizontal and vertical line segments, meeting at right angles. The course has n *starting points* and n *holes*, in one-to-one correspondence. It is always possible hit the ball along a straight line directly from each starting point to the corresponding hole, without touching the boundary of the playing field. (Players are not allowed to bounce golf balls off the walls; too much glass.) The n starting points and n holes are all at distinct locations.

   Sadly, the architect's computer crashed just as construction was about to begin. Thanks to the herculean efforts of their sysadmins, they were able to recover the *locations* of the starting points and the holes, but all information about which starting points correspond to which holes was lost!

   Describe and analyze an algorithm to compute a one-to-one correspondence between the starting points and the holes that meets the straight-line requirement, or to report that no such correspondence exists. The input consists of the x- and y-coordinates of the m corners of the playing field, the n starting points, and the n holes. Assume you can determine in constant time whether two line segments intersect, given the x- and y-coordinates of their endpoints.



A minigolf course with five starting points ($\star$) and five holes ($\circ$), and a legal correspondence between them.

ϒ 4. Let $G = (V, E)$ be a directed graph where the in-degree of each vertex is equal to its out-degree. Prove or disprove the following claim: For any two vertices u and v in G, the number of mutually edge-disjoint paths from u to v is equal to the number of mutually edge-disjoint paths from v to u.

5. You are given a set of n boxes, each specified by its height, width, and depth. The order of the dimensions is unimportant; for example, a $1 \times 2 \times 3$ box is exactly the same as a $3 \times 1 \times 2$ box of a $2 \times 1 \times 3$ box. You can nest box A inside box B if and only if A can be rotated so that it has strictly smaller height, strictly smaller width, and strictly smaller depth than B.

   (a) Design and analyze an efficient algorithm to determine the largest sequence of boxes that can be nested inside one another. *[Hint: Model the nesting relationship as a graph.]*

   $\curlyvee$ (b) Describe and analyze an efficient algorithm to nest all n boxes into as few groups as possible, where each group consists of a nested sequence. You are not allowed to put two boxes side-by-side inside a third box, even if they are small enough to fit.[1] *[Hint: Model the nesting relationship as a **different** graph.]*

6. *[Extra credit]* Prove that Ford's generic shortest-path algorithm (described in the lecture notes) can take exponential time in the worst case when implemented with a stack instead of a heap (like Dijkstra) or a queue (like Bellman-Ford). Specifically, construct for every positive integer n a weighted directed n-vertex graph G_n, such that the stack-based shortest-path algorithm call RELAX $\Omega(2^n)$ times when G_n is the input graph. *[Hint: Towers of Hanoi.]*

[1]Without this restriction, the problem is NP-hard, even for one-dimensional "boxes".