CMPS 356 **Fall 2005-06**

Exam 2: Final

Chiraz BenAbdelkader *01/02/06*

Name: . . . .

Time: 180 MINUTES

Important Notes ...

- This exam is **closed-book** and **open-notes**.
- Make sure there are 8 pages and 5 problems for a total of 100 points.
- Use **pseudo-code** notation to write any algorithms you are asked to write.
- Write as legibly as you can, otherwise your answer will NOT be marked!

Question	Your Grade	Max. Grade
1		27
2		15
3		24
4		12
5		22
Total		100

Question 1: (27 Points) Misc. Computational Geometry

You may assume that a n-vertex polygon is given as a **counterclockwise** sequence of its vertices $p_1, p_2, \cdots, p_n$ (i.e. where $p_i p_{i+1}$ is an edge for all $1 \leq i \leq n-1$, and $p_n p_1$ is also an edge).

(a) Given four points a, b, c, and d, explain how to determine whether d lies within the interior of the triangle defined by points a, b, and c, using the **Orient** operation.

(b) Let E be a set of segments that are the edges of a **convex** polygon. (Obviously, each segment is given in terms of its two endpoints.) Describe an $O(n \log n)$ time algorithm that computes from E a list containing all the vertices of the polygon in **clockwise** order.

(c) Give an *efficient* algorithm to determine whether a point q is within a *simple* polygon. Briefly analyze the runtime complexity of your algorithm.

(d) Give an *efficient* algorithm to compute the area of a *simple* polygon. Briefly analyze the runtime complexity of your algorithm.

(e) A polygon is said to be x-monotone if: the x coordinates of its vertices always increase when walking from the leftmost vertex to the rightmost vertex. Give an efficient algorithm that determines whether a simple polygon is x-monotone.

(f) A simple polygon is said to be *rectilinear* if all its edges are either horizontal or vertical. Give an example rectilinear polygon for which $\lfloor n/4 \rfloor$ cameras are necessary to guard it.

Question 2: (15 Points) Line Segment Intersections

Recall the algorithm for the decision variant of the line segment intersection problem:

```
ANY-SEGMENTS-INTERSECT(S)
1  T ← ∅
2  S ← sort endpoints of segments in S from left to
3  right, and break ties by putting points with smaller y-coordinates first
4  for each point p ∈ S
5      if p is the left endpoint of a segment s
6        then INSERT(s,T)
7             if ABOVE(s,T) exists and intersects s or BELOW(s,T) exists and intersects s
8               then return TRUE
9      if p is the right endpoint of some segment s
10       then if ABOVE(s,T) exists and BELOW(s,T) exists and ABOVE(s,T) intersects BELOW(s,T)
11              then return TRUE
12            DELETE(s,T) return FALSE
13 return FALSE
```

(a) What happens if we replace the **if** statement on line 9 with an **else if**? What goes wrong?

(b) Suppose we modified this algorithm by **replacing** each *return* statement with one that outputs the intersection point. Does this algorithm correctly solve the non-decision variant of the line segment intersection problem; i.e. does it output all intersection points? Illustrate your answer with a small example.

(c) Recall also that the above algorithm makes the following assumptions: 1) No line segment is vertical, 2) If two segments intersect, they intersect at a single point (i.e. non overlapping segments), 3) No three (or more) line segments intersect in a common point.

Explain how we need to modify the algorithm ANY-SEGMENTS-INTERSECT to correctly handle each of the above three special cases.

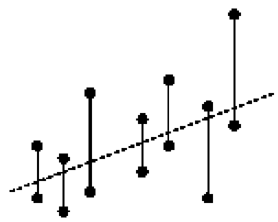
Question 3: (24 Points) Linear Programming

(a) You are the project manager at a company. Assume that you are given a list P of n employees $p_1, p_2, \cdots, p_n$, and a list of m tasks $t_1, t_2, \cdots, t_m$. Each employee p_a has a limit L_a on the number of hours that they can work. Each task t_b has a requirement R_b of the total number of hours that must be devoted to this task in order for it to be finished. For each employee p_a and each task t_b you are given an hourly salary $c_{a,b}$ that you must pay p_a to work at task t_b. Your goal is to find the cheapest way to assign employees to tasks so that all tasks are completed, and all the above constraints are satisfied. The cost of an assignment is just the total amount of money you have to pay all of the employees. For now, you need only be concerned about the total time devoted to each tasks. You need not worry about scheduling when employees work on tasks; that will be handled later by your graduate student intern.

Show that this problem can be cast as a **linear program** (and hence an efficient algorithm exists for it, since efficient algorithms exist for LPs). Your final LP must be formulated in **standard form**.

(b) You are given a set of n vertical line segments in the plane. We want to determine whether there exists a line that intersects all of these segments. An example is shown in the Figure below.

Show that this problem can be cast and solved as a linear program. *Hint:* represent the unknown line as $y = ax + b$; your $\mathbf{x}$ vector will contain other unknowns, besides a and b



(c) You are given two sets of points in the plane, the red set R contains n_r points, and the blue set B containing n_b points. The total number of points in both sets is $n = n_r + n_b$. We want to determine whether the convex hulls of the red set and the blue set intersect (i.e. they are not disjoint).

Show that this problem can be cast and solved as a linear program. *Hint:* if the CH's of R and B do **not** intersect, then there exists a line that separates the sets of points...

Question 4: (12 Points) Dynamic Programming

Given a sequence of n integers x, we say $x_{i_1}, x_{i_2}, \cdots, x_{i_k}$ is an *increasing subsequence* of x, if $i_1 < i_2 < \cdots < i_k$ and $x_{i_1} \leq x_{i_2} \leq \cdots \leq x_{i_k}$. For example, if $x = \{11, 14, -1, 2, 6, 25, 9\}$ then $\{11, 14, 25\}$ is an increasing subsequence of x.

Give a dynamic programming algorithm that computes the **longest** increasing subsequence of x. First derive the three basic DP components of the problem (optimal substructure, ...), then give the **iterative** DP algorithm and briefly analyze its **running time**.

Question 5: (22 Points) NP Completeness & Approximation Algorithms

(a) Consider the following two decision problems:

PARTITION: Given a set S of arbitrary numbers, decide whether it can be partitioned into two subsets whose sums are equal. That is, if there exists C_1 and C_2 such that $C_1 \cup C_2 = S$ and $\sum_{x \in C_1} = \sum_{x \in C_2}$.

SUBSET-SUM: Given a set S of arbitrary numbers and an arbitrary number t, decide whether there exists a subset of S whose sum is equal to t. That is, if there exists C such that $C \subseteq S$ and $\sum_{x \in C} = t$.

(i) Show that PARTITION $\leq_P$ SUBSET-SUM.
(ii) Show that SUBSET-SUM $\leq_P$ PARTITION.

(b) Consider the following *rectangle covering* problem:

Input: a collection of rectangles $I = \{R_1, R_2, \cdots, R_n\}$ in the plane such that each rectangle is aligned with the x- and y- axes (all sides are horizontal and vertical). These rectangles may overlap.

Output: a minimal set of points $P = \{p_1, p_2, \cdots, p_m\}$ such that each rectangle in I contains at least one point from P.

Give an *efficient* approximation algorithm for this problem.