CS570
Analysis of Algorithms
Summer 2009
Final Exam

Name: _____

Student ID: _____

	Maximum	Received
Problem 1	10	
Problem 2	20	
Problem 3	15	
Problem 4	20	
Problem 5		
Problem 6		
Total	100	

2hr exam, closed books and notes.

1) 10 pts

For each of the following sentences, state whether the sentence is known to be TRUE, known to be FALSE, or whether its truth value is still UNKNOWN.

(a) If a problem is in P, it must also be in NP.

TRUE.

(b) If a problem is in NP, it must also be in P.

UNKNOWN.

(c) If a problem is NP-complete, it must also be in NP.

TRUE.

(d) If a problem is NP-complete, it must not be in P.

UNKNOWN.

(e) If a problem is not in P, it must be NP-complete.

FALSE.

If a problem is NP-complete, it must also be NP-hard.

TRUE.

If a problem is in NP, it must also be NP-hard.

FALSE.

If we find an efficient algorithm to solve the Vertex Cover problem we have proven that P=NP

TRUE.

If we find an efficient algorithm to solve the Vertex Cover problem with an approximation factor $\rho \geq 1$ (a single constant) then we have proven that P=NP

FALSE.

If we find an efficient algorithm that takes as input an approximation factor $\rho \geq 1$ and solves the Vertex Cover problem with that approximation factor, we have proven that P=NP.

TRUE.

2) 20 pts

Let $G = (V, E)$ be a weighted, directed graph with weight function $w : E \to \{0, 1, \ldots, W\}$ for some nonnegative integer W. Modify Dijkstra's algorithm to compute the shortest paths from a given source vertex s in $O(WV + E)$ time.

Consider running Dijkstra's algorithm on a graph, where the weight function is $w : E \to \{1, \ldots W - 1\}$. To solve this efficiently, implement the priority queue by an array A of length $WV + 1$. Any node with shortests path estimate d is kept in a linked list at A[d]. A[$WV + 1$] contains the nodes with ∞ as estimate.

EXTRACT-MIN is implemented by searching from the previous minimun shortest path estimate until a new is found. DECREASE-KEY simply moves vertices in the array. The EXTRACT-MIN operations takes a total of $O(VW)$ and the DECREASE-KEY operations take $O(E)$ time in total. Hence the running time of the modified algorithm will be $O(VW + E)$.

3) 15 pts

At a dance, we have n men and n women. The men have height g(1),…,g(n), and women h(1),…,h(n). For the dance, we want to match up men with women of roughly the same height. Here are the precise rules:

a. Each man i is matched up with exactly one woman w_i, and each woman with exactly one man.
b. For each couple (i, w_i), the mismatch is height difference $|g(i)-h(w_i)|$.
c. Our goal is to find a matching minimizing the maximum mismatch, $MAX_i |g(i)-h(w_i)|$.

Give an algorithm that runs in O(n logn) and achieves the desired matching. Provide proof of correctness.

We use an exchange argument. Let w_i denote the optimal solution. If there is any pair i, j such that $i < j$ in our ordering, but $w_i > w_j$ (also with respect to our ordering), then we evaluate the effect of switching to $w'_i := w_j, w'_j := w_i$. The mismatch of no other couple is affected. The couple including man i now has mismatch $|g(i) - h(w'_i)| = |g(i) - h(w_j)|$. Similarly, the other switched couple now has mismatch $|g(j) - h(w_i)|$.

We first look at man i, and distinguish two cases: if $h(w_j) \leq g(i)$ (i.e., man i is at least as tall as his new partner), then the sorting implies that $|g(i)-h(w_j)| = g(i)-h(w_j) \leq g(j)-h(w_j) = |g(j)-h(w_j)|$. On the other hand, if $h(w_j) > g(i)$, then $|g(i) - h(w_j)| = h(w_j) - g(i) \leq h(w_i) - g(i) = |h(w_i) - g(i)|$. In both cases, the mismatch between man i and his new partner is at most the previous maximum mismatch.

We use a similar argument for man j. If $h(w_i) \leq g(j)$, then $|g(j)-h(w_i)| = g(j)-h(w_i) \leq g(j)-h(w_j) = |g(j) - h(w_j)|$. On the other hand, if $h(w_i) > g(j)$, then $|g(j) - h(w_i)| = h(w_i) - g(j) \leq h(w_i) - g(i) = |h(w_i) - g(i)|$. Hence, the mismatch between j and his partner is also no larger than the previous maximum mismatch.

Hence, in all cases, we have that both of the new mismatches are bounded by the larger of the original mismatches. In particular, the maximum mismatch did not increase by the swap. By making such swaps while there are inversions, we gradually transform the optimum solution into ours. This proves that the solution found by the greedy algorithm is in fact optimal.

4) 20 pts

You are given integers p0, p1, …, pn and matrices A1, A2, …, An where matrix Ai has dimension (pi-1) * pi

(a) Let m(i, j) denote the minimum number of scalar multiplications needed to evaluate the matrix product Ai Ai+1 … Aj. Write down a recursive algorithm to compute m(i, j), 1 <= i <= j <= n, that runs in O(n^3) time.
Hint: You need to consider the order in which you multiply the matrices together and find the optimal order of operations.

```
Initialize m[i, j] = -1  1 <= i <= j <= n

Output mcr(1, n)

mcr(i, j) {
   if m[i, j] >= 0 return m[i, j]
   else if I == j m[i, j] = 0
   else m[i, j] = min (i <= k < j) { mcr(i, k) + mcr(k+1, j) +Pi-1*Pk*Pk}

   return m[i, j]
}
```

(b) Use this algorithm to compute m(1,4) for p0=2, p1=5, p2=3, p3=6, p4=4

m[i, j]

i\j	2	3	4
1	30	66	114
2		90	132
3			72

m[1, 4] = 114

5) 20 pts

In a certain town, there are many clubs, and every adult belongs to at least one club. The townspeople would like to simplify their social life by disbanding as many clubs as possible, but they want to make sure that afterwards everyone will still belong to at least one club.
Prove that the Redundant Clubs problem is NP-complete.

First, we must show that Redundant Clubs is in NP, but this is easy: if we are given a set of K clubs, it is straightforward to check in polynomial time whether each person is a member of another
club outside this set.
Next, we reduce from a known NP-complete problem, Set Cover. We translate inputs of Set Cover to inputs of Redundant Clubs, so we need to specify how each Redundant Clubs input element
is formed from the Set Cover instance. We use the Set Cover's elements as our translated list of people,
and make a list of clubs, one for each member of the Set Cover family. The members of each club are just the elements of the corresponding family. To finish specifying the Redundant Clubs input,
we need to say what K is: we let $K = F - K_{SC}$ where F is the number of families in the Set Cover instance and K_{SC} is the value K from the set cover instance. This translation can clearly be done in polynomial time (it just involves copying some lists and a single subtraction).
Finally, we need to show that the translation preserves truth values. If we have a yes-instance of Set Cover, that is, an instance with a cover consisting of K_{SC} subsets, the other K subsets form a solution to the translated Redundant Clubs problem, because each person belongs to a club in the
cover. Conversely, if we have K redundant clubs, the remaining K_{SC} clubs form a cover. So the answer
to the Set Cover instance is yes if and only if the answer to the translated Redundant Clubs instance
is yes.

6) 15 pts

A company makes two products (X and Y) using two machines (A and B). Each unit of X that is produced requires 50 minutes processing time on machine A and 30 minutes processing time on machine B. Each unit of Y that is produced requires 24 minutes processing time on machine A and 33 minutes processing time on machine B.

At the start of the current week there are 30 units of X and 90 units of Y in stock. Available processing time on machine A is forecast to be 40 hours and on machine B is forecast to be 35 hours.

The demand for X in the current week is forecast to be 75 units and for Y is forecast to be 95 units—these demands must be met. In addition, company policy is to maximize the combined sum of the units of X and the units of Y in stock at the end of the week.

a. Formulate the problem of deciding how much of each product to make in the current week as a linear program.

Solution

Let

- x be the number of units of X produced in the current week
- y be the number of units of Y produced in the current week

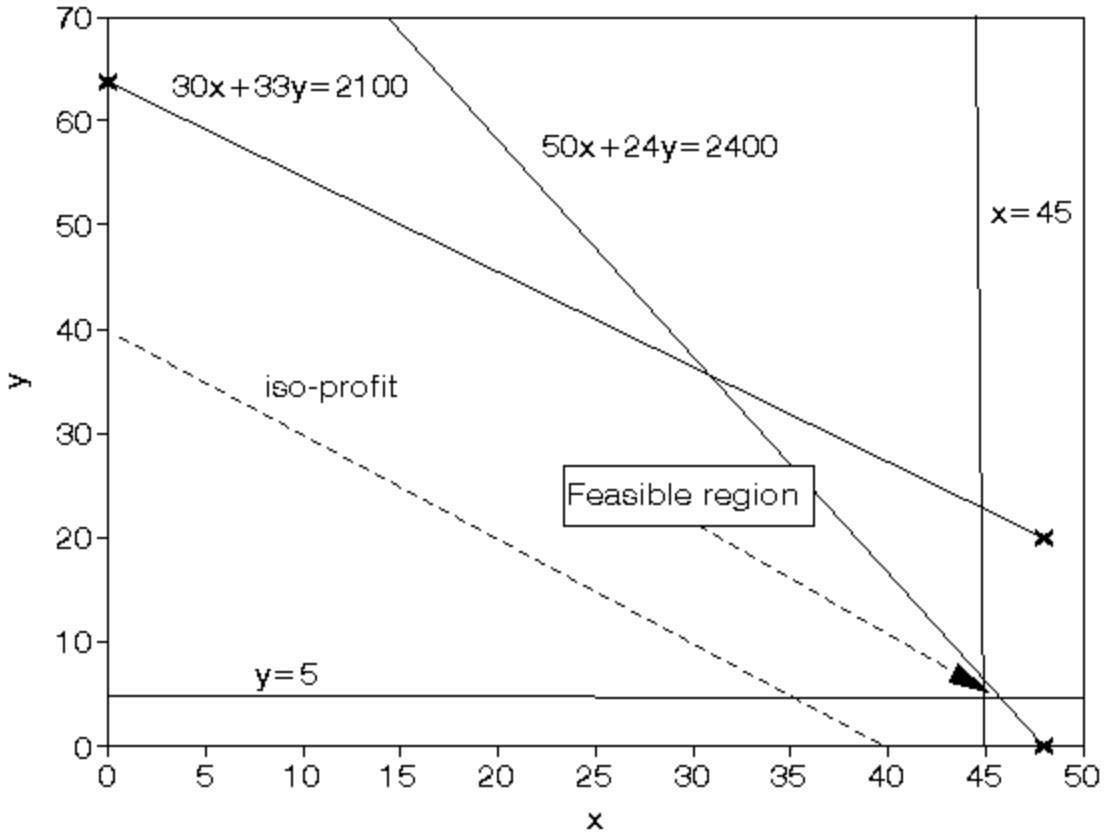
then the constraints are:

- 50x + 24y <= 40(60) machine A time
- 30x + 33y <= 35(60) machine B time
- x >= 75 - 30
- i.e. x >= 45 so production of X >= demand (75) - initial stock (30), which ensures we meet demand
- y >= 95 - 90
- i.e. y >= 5 so production of Y >= demand (95) - initial stock (90), which ensures we meet demand

The objective is: maximise (x+30-75) + (y+90-95) = (x+y-50)
i.e. to maximise the number of units left in stock at the end of the week

b. Solve this linear program graphically.

It can be seen in diagram below that the maximum occurs at the intersection of x=45 and 50x + 24y = 2400



Solving simultaneously, rather than by reading values off the graph, we have that x=45 and y=6.25 with the value of the objective function being 1.25