University of Waterloo

University of Waterloo Final Examination

Winter 2002

Student Name	________________________________
Student ID Number	________________________________

Course Abbreviation & Number	CS 341
Course Title	Algorithms
Section(s)	01 and 02
Held With Course(s)	none
Section(s) of Held With Course(s)	none
Instructor	T. Biedl and D. Brown

Date of Exam	April 9, 2002	
Time Period: evening	Start Time: 7:00pm	End Time: 10:00pm
Duration of Exam:	3 hours	
Number of Exam Pages (including this cover sheet)	12 pages	
Exam Type	Special Material	
Additional Materials Allowed	1 letter-sized sheet of paper with anything written or typed on both sides.	

1. Complete all answers in the spaces provided.
2. Write neatly so you do not lose marks unnecessarily.
3. Proctors will only confirm or deny the existence of errors on the exam.
4. In the case of perceived ambiguity, state a clear assumption and proceed to answer the question.
5. Cheating is an academic offense. Your signature on this exam indicates that you understand and agree to the university's policies regarding cheating on exams.

#	Pts.	Actual	Initial
1	10		
2	8		
3	10		
4	15		
5	20		
6	15		
7	5		
8	17		
$\sum$	100		

Signature:________________________________

1. (10 pts): **True or False?**
   For each of the following statements, say whether it is true or false. Assume that $\mathcal{P} \neq \mathcal{NP}$ unless explicitly said otherwise.

   You must write the word "True" or "False." Please don't write "T" or "F", for the danger is too big that we cannot distinguish them!

   (a) The Master Theorem can be used to compute asymptotic upper and lower bounds for the recursion $T(n) = 2T(n/2) + n \log n, T(1) = 0$.

   (b) The recursion $T(n) = T(n/5) + T(7n/10) + 10n, T(1) = 0$ satisfies $T(n) \in \theta(n)$.

   (c) Computing the Convex Hull takes $\Omega(n \log n)$ time, because there is a linear reduction from Sorting to Convex Hull.

   (d) For every satisfiable instance of SAT, there is a certificate of polynomial size to prove that it is satisfiable.

   (e) Any problem in $\mathcal{NP}$ has a polynomial time reduction to Integer Programming.

   (f) If $\mathcal{P} = \mathcal{NP}$, then all $\mathcal{NP}$-hard problems are solvable in polynomial time.

   (g) To prove that Subset-Sum is $\mathcal{NP}$-hard, it suffices to give a polynomial reduction from Subset-Sum to SAT.

   (h) There exists a polynomial reduction from Single-Source Shortest Path (phrased as a decision problem) to TSP (also phrased as a decision problem).

   (i) Branch & Bound will always find an optimal solution if run to completion.

   (j) The existence of a 2-approximation algorithm for VertexCover implies that any minimization problem in $\mathcal{NP}$ has a 2-approximation algorithm.

2. (8 pts): **$\mathcal{P}/\mathcal{NP}/\mathcal{NP}$-hard**
Enter "True" or "False" in each entry of the table. Justify your answer briefly where requested to do so. Assume $\mathcal{P} \neq \mathcal{NP}$.

For all problems below, use the corresponding decision problems. If you are unsure about the exact definition of a problem, state your assumptions clearly.

	$\in \mathcal{P}$	$\in \mathcal{NP}$	$\mathcal{NP}$-hard
Knapsack		$(*_1)$	
Minimum Spanning Tree			$(*_2)$
Hamiltonian Cycle	$(*_3)$		
Longest Common Subsequence		$(*_4)$	

Please provide brief explanations (1-2 sentences) for the following entries:

$(*_1)$

$(*_2)$

$(*_3)$

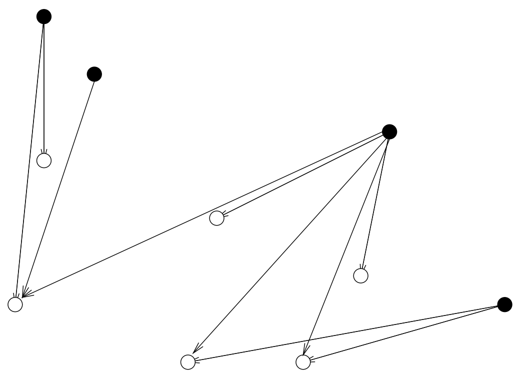
$(*_4)$

3. (10 pts): **Branch and Bound**
Here is pseudo-code for computing the size of a maximum independent set with backtracking (derived from the solution to the midterm question.)

```
(1) Input: Graph $G$
(2) Call RecursiveIndependentSet($G$)
(3)
(4) RecursiveIndependentSet($G$) {
(5)         if $G$ has no vertices, return 0
(6)         else {
(7)                 let $v$ be a vertex of $G$, let $w_1, \ldots, w_k$ be its neighbours.
(8)                 delete $v$ from $G$.
(9)                 let value1 = RecursiveIndependentSet($G$).
(10)                delete $w_1, \ldots, w_k$ from $G$.
(11)                let value2 = 1 + RecursiveIndependentSet($G$).
(12)                return max{value1, value2}
(13)        }
(14) }
```

Describe two different ways to improve the running time of this algorithm by pruning useless subproblems, i.e., subproblems that will not lead to an improvement in the solution. Your pruning rules should be such that they can be tested in polynomial time.

4. (15 pts): **Extreme points**
Let $p_i = (x_i, y_i)$, $i = 1, \ldots, n$, be n distinct points. Point p_i *dominates* point p_j if $x_i \geq x_j$ and $y_i \geq y_j$. A point is said to be *extreme* if no other point dominates it. The figure below shows a set of points where arrows indicate some (not all) of the dominations, and extreme points are black.



Describe an $O(n \log n)$ time algorithm that gets as input n distinct points, and outputs the extreme points. Justify the correctness and running time of your algorithm. $O(n^2)$ algorithms will receive partial credit.

5. (20 pts): **GeneralizedSubsetSum**
The GeneralizedSubsetSum problem is the following problem. You are given n numbers $a_1, \ldots, a_n$ and a target-value T, where $a_1, \ldots, a_n, T$ are non-negative integers. You want to find a set $I \subseteq \{1, \ldots, n\}$ such that $\sum_{i \in I} a_i \leq T$, and $\sum_{i \in I} a_i$ is maximal. In other words, you want to get as close as possible to the target value without exceeding it.

   (a) Phrase GeneralizedSubsetSum as a decision problem.

   (b) Show that the GeneralizedSubsetSum decision problem is in $\mathcal{NP}$.

   (c) Show that there exists a polynomial reduction from GeneralizedSubsetSum to Knapsack, i.e., GeneralizedSubsetSum $\leq_p$ Knapsack. (You can use the associated decision problems instead, if you prefer.)

(cont'd on next page)

(d) There exists a Dynamic Programming algorithm that computes the optimum value for GeneralizedSubsetSum and works in $O(nT)$ time. To show this, give the definition a recursive function whose value is the optimum value of the GeneralizedSubsetSum instance. Justify your answer.

Your function must be such that a standard bottom-up implementation of it would yield an $O(nT)$ time algorithm, but you do *not* have to give pseudo-code for it.

(e) Show that the decision version of GeneralizedSubsetSum is $\mathcal{NP}$-hard.

6. (15 pts): **Not-all-equal 3-SAT**
An instance of NAE-3SAT is defined as follows. You are given n boolean variables $x_1, \ldots, x_n$ and m clauses $c_1, \ldots, c_m$. Each clause consists of exactly three different variables. We say that a clause is satisfied (in a given TRUE/FALSE assignment to the variables) if its variables do not all have the same value. Put differently, $c_j = \{x_{j_1}, x_{j_2}, x_{j_3}\}$ is satisfied if and only if

   - at least one of $x_{j_1}, x_{j_2}, x_{j_3}$ is TRUE, and
   - at least one of $x_{j_1}, x_{j_2}, x_{j_3}$ is FALSE.

   (a) Given an instance of NAE-3SAT, we want to know whether there there an assignment of TRUE/FALSE to the variables such that all clauses are satisfied. Show that this problem has a polynomial reduction to 3-SAT, i.e., show that NAE-3SAT $\leq_p$ 3-SAT. Explicitly give the reduction, i.e., do not just argue that one must exist.

(cont'd on next page)

(b) Give a randomized algorithm that finds a TRUE/FALSE assignment to the variables such that the expected number of satisfied clauses is at least $\frac{3}{4}m$. Your algorithm should have polynomial running time. Justify why the bound on the expected number holds.

7. (5 pts): **Coin splitting as Integer Program**
   Recall the coin splitting problem: You are given k denominations of coins $c_1, \ldots, c_k$, and a target value T, where $c_1, \ldots, c_k, T$ are positive integers and $c_1 = 1$. You want to find the fewest number of coins that sum up to exactly T.

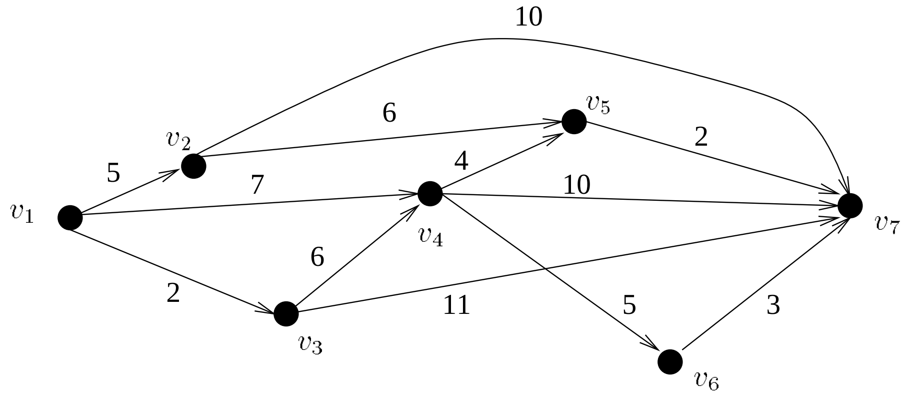
   Show how to phrase the coin splitting problem as an Integer Program. Briefly explain the meaning of the variables and the constraints in your Integer Program. You do not need to give a formal proof of correctness of your reduction.

8. (17 pts): **Longest paths**
Recall the Longest Path problem: You are given a weighted directed graph G and two vertices s and t. You want to find a simple path from s to t that has the maximum possible total weight. This problem is $\mathcal{NP}$-complete, but becomes polynomial in the special case described below.

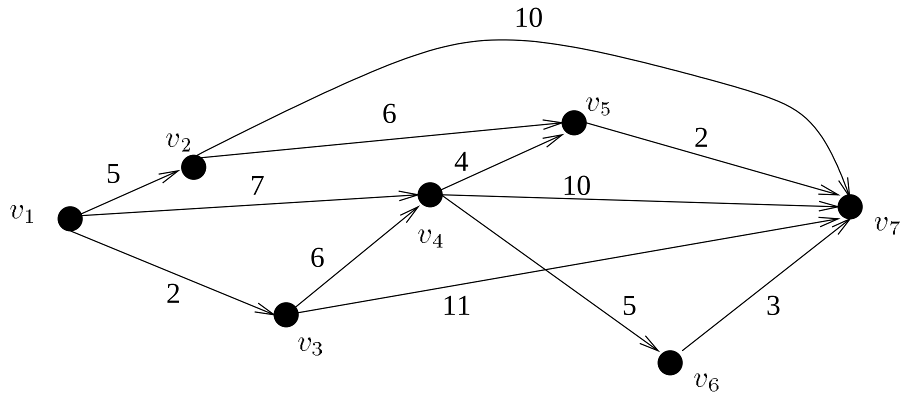
   (a) In the graph below, the vertices have been numbered $v_1, \ldots, v_7$ such that all edges go from a lower-numbered to a higher-numbered vertex, i.e., all edges have the form $v_i \to v_j$ with $i < j$. Indicate a longest path from v_1 to v_7 in this graph with thick lines. Break ties arbitrarily.

   You can mark more edges than needed, as long as your set of marked edges contains a unique path from v_1 to v_7.

   You need not explain how you obtained your answer.



Here is a duplicate of the graph, just in case you need to start over. If you use both graphs, please indicate clearly which one contains the correct solution.



(b) Assume that you are given a weighted directed graph G with the vertices numbered as $v_1, \ldots, v_n$ such that all edges have the form $v_i \to v_j$ with $i < j$. Give an algorithm to compute a longest path from v_1 to v_n. You may assume that all vertices except v_1 have at least one incoming edge.

Analyze the running time of your algorithm. Your algorithm should take $O(m)$ or $O(n^2)$ time, but slower algorithms will receive partial credit.

You need not give a formal proof of correctness of your algorithm, but explain briefly the underlying idea.