CORE EXAMINATION
Department of Computer Science
New York University
September 22, 2006

This is the common examination for the M.S. program in CS. It covers core computer science topics: Languages and Compilers, Operating Systems, and Algorithms. The exam has two parts. The first part lasts three hours and covers the first two topics. The second part, given this afternoon, lasts one and one-half hours, and covers algorithms.

Use the proper booklet or answer sheet for each question. Each booklet is marked with the Area and Question number, in the form PL&C1, PL&C2, PLC&C3, OS1, OS2, ALGS1 (this question has an answer sheet and not a booklet), ALGS2, ALGS3. Use the appropriate booklet for each question. DO NOT put your name on the exam booklet. Instead, your exam number must be on every booklet.

You will be graded according to your exam number, shown on the envelope containing the booklets. Remember your exam number: when grades are given out, they will be published according to this number, not by name.

Make sure your name and signature are on the envelope. This is the only place where your name appears. Please include all the booklets inside the envelope. You can keep the exam.

Good luck!

Basic Algorithms

Question 1

Instructions for short answer question: please try to write only in the boxed areas provided for each question. At our option, we may choose not to grade any answers outside the boxes.

a. [4 points]. Let $G = (V, E)$ be an undirected graph in which each vertex $v \in V$ is associated with a positive weight $W(v) > 0$. A subset $S \subseteq V$ is **independent** if for all $u, v \in S$, there is no edge between u and v, i.e., $(u, v) \notin E$. We want to compute an independent subset $S \subseteq V$ whose weight $W(S) := \sum_{u \in S} W(u)$ is maximum. For each proposed algorithm below, give a counterexample, i.e., an example on which it does not compute a maximum weight independent set. Assume that $V = \{v_1, \ldots, v_n\}$ and $W(v_1) \leq W(v_2) \leq \cdots \leq W(v_n)$.

BONUS point if you find a single example that is a counterexample to both algorithms.

PART a(i):

$S \leftarrow \emptyset$ (empty set)
For $i = n$ downto 1
    If $S \cup \{v_i\}$ is independent,
        append v_i to S.
RETURN (S)

ANSWER:

PART a(ii):

$S \leftarrow V$
For $i = 1$ to n
    If there is some $u \in S$ such that $(u, v_i) \in E$,
        remove v_i from S.
RETURN (S)

ANSWER:

b. [6 points]. We want a data structure for storing a set of keys, supporting the following operations: (a) insert a key, (b) lookup a key, (c) delete a key, (d) list all the keys in sorted order. We compare three implementations, using (respectively) a balanced binary tree, an unsorted linked list, and a sorted linked list. Do not assume any additional permanent structures such as a pointer to the node containing an item to be deleted; however, your procedures may construct and use additional temporary structures as needed. For each operation and data structure (12 pairs), write, in the corresponding entry in the table below, the asymptotic order of the worst-case time to implement the indicated operation for the given data structure. One entry has been filled in – note that time is given as a function of n, the size of the current set of keys.

	Balanced Binary Tree	Unsorted Linked List	Sorted Linked List
(a) Insert	$\log n$		
(b) Lookup			
(c) Delete			
(d) List			

PLEASE USE A NEW EXAM BOOKLET

Question 2

Two neigboring countries, Equalistan and Unfairstan, have very similar types of highway system. In each, the highway system could be viewed as a directed graph $G = (V, E)$, where nodes V represent the major cities, and edges E respresent the major highways connecting some pairs of cities. Each such edge e connecting two cities a and b has a certain *toll*, which will be defined slightly differently for the two countries, as described below. The cost of a path between two cities is the sum of the tolls which need to be paid along its edges. A driver starting with the inital wealth w and originating in the capital city s of the corresponding country wishes to compute the following two arrays for every city $u \in V[G]$:

- $d[u]$, which is the smallest cost of a path from s to u, provided this cost is at most w, and ∞ if no such path exists.
- $previous[u]$, which is $undefined$ if $u = s$ or $d[u] = \infty$, and otherwise equals the predecessor of u along a cheapest path from s to u.

For each of the following toll functions, complete the corresponding variation of the Dijkstra's algorithm to correctly compute the values $d[u]$ and $previous[u]$ for all $u \in V$. The initial common segment is given below, and you only have to continue the code.

```
function Distances(G, w, s, cost-params)
    for each vertex v in V[G]
        d[v] := ∞;
        previous[v] := undefined;
    d[s] := 0;
    Q := V[G]; {Q is a priority queue}
    while Q is not an empty set
        ...
```

a. [4 points]. In Equalistan, the toll for each edge $e = (a, b)$ is equal to $f_e = f(a, b) \geq 0$, and is the same for every driver. Thus, cost-params $= \{f_e\}_{e \in E}$. This is similar to the standard variant of Dijkstra's single source shortest path algorithm (but not exactly the same because of the parameter w) .

b. [6 points]. In Unfairstan, the toll for traversing an edge $e = (a, b)$ includes not only the fixed cost $f_e = f(a, b) \geq 0$, but in addition each driver is required to pay a certain fraction $p_e = p(a, b) \in [0, 1)$ of its current wealth x (initially equal to w). Namely, the toll is equal to $p_e * x + f_e$ (notice, Equalistan simply corresponds to a special case of Unfairstan where all $p_e = 0$). For example, if a driver has \$100 at point a, $p_e = 0.25$ and $f_e = 50$, then the toll for the edge $e = (a, b)$ for *this driver* is $0.25 * 100 + 50 = \$75$, which means that this driver will have only \$25 after traversing this edge.

On the other hand, if a driver had only \$60 at point a, then the toll would be $0.25 * 60 + 50 = \$65$, which is greater than \$60, implying that the driver does not have enough wealth to traverse this edge.

Thus, cost-params $= \{p_e, f_e\}_{e \in E}$. Please change a few lines in your solution to part A to correctly fill in the values $d[u]$ and $previous[u]$. Briefly argue why your algorithm is correct.

PLEASE USE A NEW EXAM BOOKLET

Question 3

The webservice WhyWhatWhere is selling advertizing slots associated with a collection of keywords. In this problem we will consider only a single keyword, the word *exam*. The word exam appears repeatedly over time, and for each occurrence there is an associated advertizing slot. WhyWhatWhere sells each of these slots to the highest bidder as they appear. A buyer B can enter a bid at any time; her bid consists of a pair (price, total), price being the amount B will pay for each slot it receives, and total being the total amount B will pay; e.g. if B bids (\$2, \$100), it means that B will pay \$2 each time she is given the slot for keyword *exam*, and B wants to pay for this up to 50 times (100/2).

When a slot is available it is sold to the highest bidder who has already bid and who still has enough funds, and the bidder's available budget is decreased accordingly.

Bidders can join at any time. Suppose that there are n bids altogether and m slots for the word *exam*.

For example we might have the sequence of actions: B_1 bids $(\$4, \$4)$, B_2 bids $(\$3, \$6)$, fill-slot, fill-slot, B_3 bids $(\$5, \$10)$, fill-slot. Then the first slot is given to B_1 at a price of \$4, the second slot to B_2 at a price of \$3, and the third slot to B_3 at a price of \$5.

a. [6 points]. Give a data structure that allows the webservice to allocate slots and include new bidders in $O(\log n)$ time per operation.

b. [4 points]. Show how to obtain a total time of $O(n \log n + m)$.
Hint. How many times can the buyer of the word *exam* change?

Solution, Question 1

a. The following example works for both parts. v_1 has weight 2, all the other nodes have weight 1. v_1 is connected to every other vertex, and there are no other edges in the graph. Both algorithms end up with an independent set consisting of just the vertex v_1 of weight 2; the complementary set of weight $n-1$ is the maximum weight independent set here.

b.

	Balanced Binary Tree	Unsorted Linked List	Sorted Linked List
(a) Insert	$\log n$	1	n
(b) Lookup	$\log n$	n	n
(c) Delete	$\log n$	n	n
(d) List	n	$n \log n$	n

Solution, Question 2

A. Full code is below:

```
function Equalistan(G, w, s, {f_e})
    for each vertex v in V[G]
        d[v] := ∞;
        previous[v] := undefined;
    d[s] := 0;
    Q := V[G];
    while Q is not an empty set
        u := Extract_Min(Q);
        if d[u] > w
            d[u] := ∞;
            previous[u] = undefined;
            continue to the next loop interation;
        for each edge (u, v) outgoing from u
            if d[u] + f(u, v) < d[v]
                d[v] := d[u] + f(u, v);
                previous[v] := u;
```

B. Full code is below:

```
function Unfairstan(G, w, s, {p_e, f_e})
    for each vertex v in V[G]
        d[v] := ∞;
```

```
        previous[v] := undefined;
d[s] := 0;
Q := V[G];
while Q is not an empty set
        u := Extract_Min(Q);
        if d[u] > w
                d[u] := ∞;
                previous[u] = undefined;
                continue to the next loop interation;
        for each edge (u, v) outgoing from u
                if d[u] + p(u, v) * (w − d[u]) + f(u, v) < d[v]
                        d[v] := d[u] + p(u, v) * (w − d[u]) + f(u, v);
                        previous[v] := u;
```

Solution, Question 3

a. We use a maxheap to store the bidders, using the bid value as the key. For each bidder, its record also stores its remaining amount of unspent money. A simple implementation, such as the standard Floyd heap, suffices. To add a new bidder, simply insert it into the heap; this takes $O(\log n)$ time.

To fill a slot, do a findmax; the returned bidder receives the slot and is charged her bid value; this charge is deducted from the bidder's remaining unspent money. If the new unspent sum is smaller than the bidder's bid, then this bidder is deleted from the heap. This all takes $O(\log n)$ time.

b. By keeping a pointer to the current maximum value item in the heap, a fill slot operation can be done in time $O(1)$ plus the $O(\log n)$ time for a deletemax if one is performed.

There is one insertion per bidder and at most one deletemax. These take time $O(n \log n)$ overall. The remaining time used by the m fill slot operations is $O(m)$, giving an overall runtime of $O(m + n \log n)$.