CORE EXAMINATION
Department of Computer Science
New York University
February 2, 2007

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.

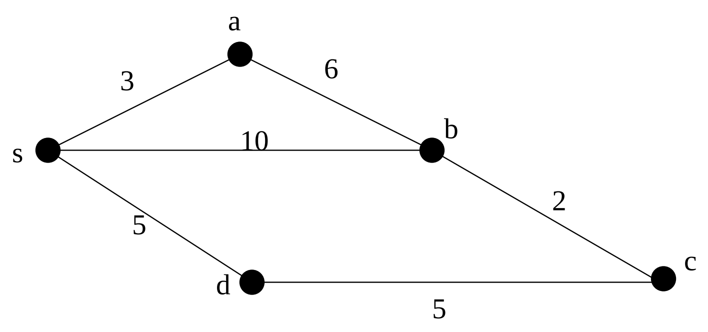
Good luck!

Algorithms

Question 1

Instructions for the short answer question: please try to write only in the boxed areas on the answer sheet provided for each question.

a. [2.5 points]. Show the operation of Prim's Minimum Spanning Tree (MST) algorithm on the following graph, starting at vertex s. Recall that Prim's algorithm grows a single tree starting from the start vertex; on completion, the resulting tree is the MST.



b. [2.5 points]. Show the operation of Dijkstra's algorithm on the above graph by showing each successive shortest path tree produced by the algorithm.

c. [3 points]. Show the operation of heap sort on the following sequence of numbers: $1, 3, 4, 2, 5$, stored in an array $A[1 : 5]$. Specifically, show the (implicit) heap used by the algorithm in tree form, at the following stages: at the start, after the heap property is established, and after each successive element is placed in sorted order.

At which array location is the root of the heap stored?

d. [2 points]. Show the operation of radix sort on the following sequence of 3 digit data, assuming each pass of the algorithm used 10 buckets, indexed by the relevant digit $(0 \cdots 9)$:

$$410, 310, 337, 417, 914, 907, 307, 414, 932, 418.$$

Show the data, in each bucket, appropriately ordered, after each of the first two passes.

PLEASE GO TO THE NEXT PAGE

Question 2 – PLEASE USE A NEW EXAM BOOKLET

Let G be a connected acyclic undirected graph. In simple English, G forms a tree, but there is no notion of a root. The diameter of such a structure is the length (in edges) of the longest path in the structure.

Use the following hint to design an efficient algorithm to compute the diameter of G.

Hint. Use an iterative process that in each iteration removes all of the current leaves of G. Repeat this process until nothing is left.

For full credit, your algorithm should run in linear time, and have enough detail to ensure that it achieves this performance. Assume that G is represented by a collection of vertices V and an array Adj of adjacency lists, so that $Adj[v]$ lists the vertices adjacent to v for each v in V.

Warning: it is better to express the algorithm in pseudocode than in the form of a written description. It is also easier to give partial credit to pseudocode because it is easier to separate what is correct from some accidental errors.

PLEASE GO TO THE NEXT PAGE

Question 3 – PLEASE USE A NEW EXAM BOOKLET

Word puzzle. A *word puzzle* consists of the following items:

- a string s of letters (each letter is from the alphabet `A`–`Z`);
- a list L of word/point pairs $((w_1, p_1), \ldots, (w_k, p_k))$, where each w_i is a non-empty string of letters and each p_i is a non-negative number.

The puzzle is played by using words from the list L to cover matching substrings in s. The same word may be played several times, and each time word w_i is played, you receive p_i points. However, in covering the string s, no two words may overlap. Subject to these constraints, the goal is to place words on the string so as to maximize the total number of points you receive.

Example. Suppose

$$s = \texttt{THERAININSPAIN}$$

and

$$L = ((\texttt{ERA}, 3), (\texttt{HER}, 2), (\texttt{PAI}, 2), (\texttt{IN}, 1))$$

One possible solution to the puzzle is to play the words `HER`, `IN`, `IN`, `IN`, as follows:

```
THERAININSPAIN
 HER ININ   IN
```

This solution is worth $2 + 1 + 1 + 1 = 5$ points. A better solution is to play the words `ERA`, `IN`, `IN`, `PAI`, as follows:

```
THERAININSPAIN
  ERAININ PAI
```

which is worth $3 + 1 + 1 + 2 = 7$ points.

Give an efficient algorithm that solves this problem. The input to the algorithm is a string s and a list $L = ((w_1, p_1), \ldots, (w_k, p_k))$ as above. It suffices if your algorithm simply outputs the point value of an optimal solution (and not the solution itself).

Your algorithm should run in time

$$O\Big(|s| \sum_{i=1}^{k} |w_i|\Big),$$

where $|x|$ denotes the length of a string x. You should also give an argument that your algorithm is correct and that its running time satisfies the above bound.

Hint: Suppose you are given s and L as above. Further, suppose you know the point value of an optimal solution for each proper prefix (initial substring) of s (with the same L); i.e. if $s = s_1 s_2 \cdots s_r$, then s_1, $s_1 s_2$, $\cdots$, $s_1 s_2 \cdots s_{r-1}$ are the proper prefixes. From this, show how to calculate the point value of an optimal solution for s. Now turn this idea into an efficient algorithm.

Solution, Question 1

For brevity, we simply describe the form of each solution.

a. The tree grows from the single vertex s, by adding in turn edges (s, a), (s, d), (d, c), (c, b).

b. The tree grows from the single vertex s, by adding in turn edges (s, a), (s, d), (a, b), (d, c).

c. The algorithm uses a maxheap, with the root stored at array location 1. Initially, prior to any actions, $A[1]$ stores item 1, its left and right children are $A[2]$ and $A[3]$ respectively, storing items 3 and 4, and $A[2]$'s left and right children are $A[4]$ and $A[5]$, storing items 2 and 5, respectively.

After the heap property is established, the items in array order are 5,3,4,2,1. Following each successive sort step, the items in the heap in array order become: (i) 4,3,1,2; (ii) 3,2,1; (iii) 2,1; (iv) 1.

d. After pass 1 the data is stored in the buckets as follows:

bucket:	0	1	2	3	4	5	6	7	8	9
	410		932		914			337	418	
	310				414			417		
								907		
								307		

After the second pass the data is stored in the buckets as follows:

bucket:	0	1	2	3	4	5	6	7	8	9
	907	410		932						
	307	310		337						
		914								
		414								
		417								
		418								

Solution, Question 2

The code below is a bottom-up BFS that trims all of the (current) leaves from the unrooted tree G at each iteration. As a consequence, the diameter decreases by two at each iteration. To distinguish the case where the diameter is odd from the case where it is even, the algorithm terminates at the iteration that leaves a subgraph with just two vertices or fewer. If this count is two, then the diameter is odd. If it is one, then the diameter is even. As is evident from the code, a leaf in G is easily recognized because it has an adjacency list that contains just one vertex. The count $v.cnt$ stores the size of the adjacency list for v in the ever shrinking graph G as leaves are trimmed off round-by-round.

```
Create the empty sets $S$ and $T$;
$Diameter \leftarrow 0$;
Let each vertex $v$ in $V$ have a count field $v.cnt$;
$unprocessed \leftarrow$ size of $V$;
**foreach** $v$ in $V$ **do**
    $v.cnt \leftarrow$ size of $Adj[v]$;
    **if** $v.cnt = 1$ **then** insert $v$ in $S$ **endif**
**endfor** ;
**while** $unprocessed > 2$ **do**
    **foreach** $w$ in $S$ **do**
        $unprocessed \leftarrow unprocessed - 1$;
        **foreach** $v$ in $Adj[w]$ **do**
            $v.cnt \leftarrow v.cnt - 1$;
            **if** $v.cnt = 1$ **then** insert $v$ in $T$ **endif**
        **endfor**
    **endfor**
    empty $S$;
    empty $T$ into $S$;
    $Diameter \leftarrow Diameter + 2$;
**endwhile**
**if** $unprocessed = 2$ **then** $Diameter \leftarrow Diameter + 1$ **endif** ;
**print** $(Diameter)$.
```

Solution, Question 3

Let $s = s[1 \dots n]$ and $L = ((w_1, p_1), \dots, (w_k, p_k))$.

Define the predicate $\mathit{Match}(i, j)$, where $i = 1, \dots, n$ and $j = 1, \dots, k$, to be true exactly if s has a match with w_j ending at location i in s ($i \geq |w_j|$ and $w_j = s[i - |w_j| + 1 \dots i]$).

For $i = 0, \dots, n$, define $\mathit{Opt}(i)$ to be the value of an optimal solution for the prefix $s[1 \dots i]$. The following is a simple recursive formulation for $\mathit{Opt}(i)$:

$$\mathit{Opt}(i) = \begin{cases} 0 & \text{if } i = 0; \\ \max\Big(\{\mathit{Opt}(i-1)\} \cup \{\mathit{Opt}(i - |w_j|) + p_j : 1 \leq j \leq k \text{ and } \mathit{Match}(i, j)\}\Big) & \text{if } i > 0. \end{cases}$$

Intuitively, the optimal solution for $s[1 \dots i]$ is obtained by either leaving space i uncovered (whence the term $\mathit{Opt}(i-1)$), or by playing a word w_j, right justified, at position i (whence the terms $\mathit{Opt}(i - |w_j|) + p_j$, subject to the constraint $\mathit{Match}(i, j)$).

Since $\mathit{Opt}(i)$ depends only on values $\mathit{Opt}(i')$, where $i' < i$, we can easily calculate all the $\mathit{Opt}(i)$ values iteratively, as follows:

```
$Opt(0) \leftarrow 0$
for $i \leftarrow 1$ to $n$ do
    $Opt(i) \leftarrow Opt(i-1)$
    for $j \leftarrow 1$ to $k$ do
        if $Match(i, j)$ then
            $Opt(i) \leftarrow \max\{Opt(i), Opt(i - |w_j|) + p_j\}$
```

The optimal solution has value $\mathit{Opt}(n)$.

As for the running time, $\mathit{Match}(i, j)$ can obviously be implemented so as to run in time $O(|w_j|)$. Each iteration of the loop thus takes time $O(\sum_j |w_j|)$, and so the running time of the whole algorithm is $O(n \sum_j |w_j|)$.