

Graduate Examination

Department of Computer Science
The University of Arizona
Spring 2004

March 5, 2004

Instructions

This examination consists of ten problems. The questions are in three areas:

1. Theory: CSc 545, 573, and 520;
2. Systems: CSc 552, 553, and 576; and
3. Applications: CSc 560, 525, 522, and 533.

You are to answer any *two* questions from each area (i.e., a total of *six* questions). If more than two questions are attempted in any area, the two highest scores will be used.

You have two hours to complete the examination. Books or notes are not permitted during the exam.

<p><i>Please write your answers directly on this exam. Use the flip side of the page if you need additional space. Keep your answers brief and to the point. The last 4 digits of your CSID number will be used as identification. You must write the CSID number on the cover page of the exam. Without the CSID number, your exam will not be graded.</i></p>

Results will be posted using these four-digit numbers.

Some problems will ask you to write an algorithm or a program. Unless directed otherwise, use an informal pseudo-code. That is, use a language with syntax and semantics similar to that of C or Pascal. You may invent informal constructs to help you present your solutions, provided that the meaning of such constructs is clear and they do not make the problem trivial. For example, when describing code that iterates over the elements of a set, you might find it helpful to use a construct such as

for $c \in S$ **do** *Stmt*

where S is a set.

SOLUTIONS

1 Theory 1 (CSc 520: Principles of Programming Languages)

Consider a Pascal-like language where we wish to introduce the notion of *persistence* to the semantics of local data declarations in procedure calls. In each scope, the programmer has the option of prefacing a data declaration with the term `persistent`, e.g.,

```
procedure Tucson(X : integer);
var
  a, b : integer;
  persistent c, d : real;
begin
  ...
```

In the above example, local variables `a` and `b` are treated as usual. Persistent variables `c` and `d` are treated exactly as local variables within the scope of `Tucson`. However, the initial values of these variables at each subsequent call must be exactly the same as were the values of the variables of those names when control most recently returned from `Tucson`. Recursive calls also obey this rule. Describe a run-time organization that will correctly implement the above persistent data mechanism. Start from the assumption that static chain pointers are used to handle scoping details. Your scheme must answer the following questions:

1. How is the correct binding between a variable name and its location in memory determined at run-time?
2. What additional actions must occur at each procedure or function invocation?
3. What additional actions must occur at each procedure or function exit?

Solution

1. This is similar to a value-result parameter mechanism. Data need to be copied on entry to the procedure and copied back on exit. Each lexical occurrence of a persistent object gets a unique location in the persistent data structure. This is then a fixed address that can be built into the generated code. Local access to each object will be in the local activation record, but this global structure will be for carrying information between invocations.
2. Copy the current value from the persistent data structure to local storage.
3. Copy the current value back to the persistent data structure.

2 Theory 2 (CSc 545: Design and Analysis of Algorithms)

Given two sequences S and S' , each consisting of n letters of some alphabet Σ . For example, assume that $\Sigma = \{a \dots z\}$, and $S = (abcdacbaedfghello)$ and $S' = (dqrworldxxx)$. The edit distance, $d(S, S')$ (as you have studied in class) is defined as the minimal number of deletions, insertions and substitutions needed to perform on S so it would be identical to S' . You have studied an algorithm whose running time is $O(n^2)$ for finding $d(S, S')$. You do not need to describe the algorithm in detail, just state which modifications to the original algorithm are required.

The input for the problem is two sequences S and S' , and a positive integer parameter k . The problem is to find an algorithm whose running time is $O(nk)$, and determines if $|d(S, S')| \leq k$. Note that the algorithm does not need to find the exact distance $d(S, S')$.

Hint — consider the $n \times n$ table used by the dynamic programming algorithm for finding $d(S_1, S_2)$, which was described in class. Show for determining if $d(S, S') \leq k$, only $O(k)$ cells need to be computed for each row of the table. Use this fact to obtain a faster algorithm. Consider first the case $k = 1$.

Solution

Let T be the two-dimensional $n \times n$ table used by the dynamic programming algorithm for finding $d(S_1, S_2)$. The cell $T[i, j]$ contains $d(S_i, S'_j)$ where S_i is the prefix of the first i letters of S , and S'_j is the prefix of first j letters of S' . Clearly if at some stage of the algorithm we find that $T[i, j] > k$, and $(j > i)$, then clearly $T[i, j'] > k$, for any $j' > j$. A symmetric claim holds also if $j < i$. Hence these cells need not be computed. In other words, only the cells which are either on the diagonal of T , or of distance $\leq k$ cells from the diagonal, need to be computed. Thus for a fixed i , only the cells $T[i, j]$ for $|i - j| \leq k$ need to be computed, before we can compute the cells $T[i + 1, j]$ (for $|i + 1 - j| \leq k$). There are only $O(n(1 + 2k)) = O(nk)$ such cells, and their computation requires $O(1)$ time per each cell.

3 Theory 3 (CSc 573: Theory of Computation)

Let G be a graph with weights associated with its edges. You should find for each of the following problems whether this problem is NP-hard. If it is NP-hard, you should briefly describe a reduction. If it is not, you should present a polynomial-time algorithm. The *cost* of a path in G is defined to be the sum of weights of its edges. The length of a path is the number of edges along the path. A simple path is a path along which each vertex occurs no more than once. Let $n = |V|$. For each of the following problems specify if the problem is in P , if the problem is in NP , and if it is NP-complete.

The 17-path problem. Here the input to the problem is $G(V, E)$ and a parameter M , and the problem is to find a simple path of length 17 in G whose cost is at most M .

The k -path problem . Here the input to the problem is G , and an integer $k \leq \lfloor n/2 \rfloor$. Here the input to the problem is $G(V, E)$, and the problem is to find a simple path of length k in G whose cost is at most M .

The k -path problem-revised . This is the same problem, but this time k is no larger than $\lfloor n/2 \rfloor$.

Solution

The first problem can be solved in polynomial time, by checking all subsequences of 17 vertices, and decide for each of them if it forms a simple path. The second problem is not easier than the problem of Hamiltonian path in a graph (can be seen by setting $k = n$). Finally the third problem is not easier, as can be seen by adding n vertices with no edge connecting them. Of course, all problems are in NP .

4 Systems 1 (CSc 552: Advanced Operating Systems)

Threads can be implemented entirely at the user level without kernel involvement (user-level threads) or in the kernel (kernel-level threads). These methods of implementation differ in how blocking I/O operations affect the running application.

1. Outline how a blocking I/O operation affects an application using several user-level threads.
2. Outline how a blocking I/O operation affects an application using several kernel-level threads.
3. *Scheduler activations* is a thread implementation strategy that can combine the good properties of user-level and kernel-level threads. Outline how a blocking I/O operation would be handled in an application using a thread package based on scheduler activations.

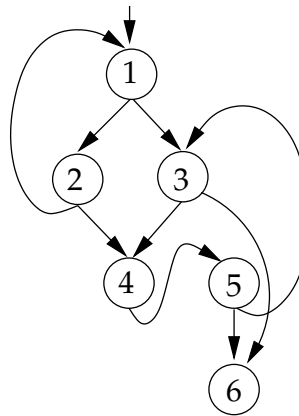
Solution

1. As the kernel is not aware of there being several user-level threads, the entire application will be blocked by the I/O operation. None of the user-level threads will run until the I/O operation completes.
2. Here the kernel is aware of the several threads, and only the one actually performing the I/O operation will be blocked. Other threads can be scheduled by the kernel when blocking occurs.
3. With scheduler activations, there is a thread runtime system at the user level, which the kernel informs about events such as start and completion of I/O operations. The user-level runtime system is responsible for scheduling threads onto a number of virtual processors provided by the kernel.

When a blocking I/O operation is started by a thread, the kernel informs the thread runtime system, which marks the blocked thread as being blocked and schedules another user-level thread onto that virtual processor. When the I/O operation is completed at some later point in time, the kernel informs the user-level runtime system about this event. The user-level runtime system will then unblock the thread, place it on the ready-queue so it can be scheduled to run, and might preempt another thread so the newly unblocked thread can be run immediately.

5 Systems 2 (CSc 553: Principles of Compilation)

(a) Draw the dominator tree for the control flow graph shown below:

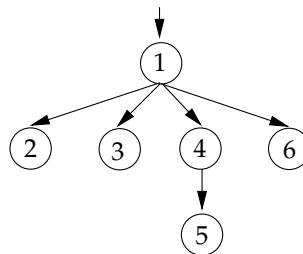


(b) [$6 \times 4 = 24$ points] Suppose we know that, in some given flow graph, vertex A dominates vertex B , and C is some other vertex in the graph. For each of the following situations, write down what we can say about the domination relationship between vertices A and C . Briefly justify your answer (a line or two of text, or a simple picture, will suffice).

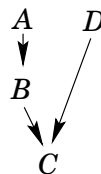
- (i) C is an immediate successor of B .
- (ii) C is an immediate predecessor of B .
- (iii) C is an immediate successor of A .
- (iv) C is an immediate predecessor of A .
- (v) B dominates C .
- (vi) C dominates B .

Solution

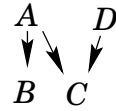
(a) The dominator tree is:



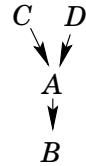
(b) (i) A may or may not dominate C , since C may have another predecessor that is not dominated by A , as shown below:



- (ii) A dominates C : for if it didn't, there would be some way to reach C without going through A , and this would imply the existence of a way to reach B without going through A , i.e., A would not dominate B .
- (iii) A may or may not dominate C , since C may have other predecessors that are not dominated by A :



- (iv) C may or may not dominate (or be dominated by) A , since A may have other predecessors that are not dominated by C :



- (v) Since the domination relation is transitive, it follows in this case that A dominates C .
- (vi) Since the domination relation can be represented as a tree, if A dominates B and C dominates B it must be the case that either A dominates C , or C dominates A .

6 Systems 3 (CSc 576: Computer Architecture)

A virtual memory system usually uses a single fixed page size. A major design decision for such a system is what size the pages should have. Discuss what advantages small and large pages have. Be sure to include, among other things, the consequences for TLB, cache size, and memory use.

Solution

Advantages with larger pages:

- Smaller page table.
- Better cache performance as cache size can grow larger. (Page can't be smaller than cache.)
- Transferring pages to/from secondary storage is more efficient.
- A TLB entry maps more memory, so more TLB hits for a given TLB size.

Advantages with smaller pages:

- Less memory wasted in last page of a segment.
- Small processes can start up quicker as less data needs to be brought into memory at startup.

NOTE: Not all of these are needed for a full score.

7 Applications 1 (CSc 522: Parallel and Distributed Programming)

Answer the following questions.

1. Atomic Broadcast. Assume one producer process and N consumer processes that all share a single buffer. The producer deposits messages into the buffer, consumers fetch them. Every message deposited by the producer has to be fetched by all N consumers before the producer can deposit another message into the buffer. Develop a solution for this problem using semaphores for synchronization.
2. Now, assume the buffer has b slots. The producer can deposit messages only into empty slots and every message has to be received by all N consumers before the slot can be reused. Furthermore, each consumer is to receive the messages in the order they were deposited. However, different consumers can receive messages at different times. For example, one consumer could receive up to b more messages than another if the second consumer is slow. Extend your answer to (a) to solve this more general problem.

Solution

1. This is a "standard" producer-consumer solution with the addition of the loop for the producer to wait for all the consumers to fetch the item, and the loop by the producer to tell all the consumers they can go.

```
sem fetch = 0, put = N;
buffer b;
```

```

Producer:                                Consumers:
  while (true) {                          while (true) {
    produce item                            P(fetch)
    for (i = 1 to N)                        get item from b
      P(put)                                V(put)
    deposit item in b
    for (i = 1 to N)
      V(fetch)
  }
}
```

2. The addition here is that the two semaphores, fetch and put, need to become arrays of semaphores. Each fetch semaphore is initially set to zero and each put semaphore is initially set to N. The producer and each consumer keeps a local mycount variable that determines which position of the arrays will be used next.

```
sem fetch[N] = ([N] 0), put[N] = ([N] N);
buffer b[N];
```

```

Producer:                                Consumers:
  int mycount = 0;                          int mycount = 0;
  while (true) {                            while (true) {
    produce item                            P(fetch[mycount]);
    for (i = 1 to N)                        get item from b[mycount];
      P(put[mycount]);                      V(put[mycount]);
    deposit item in b[mycount]
    for (i = 1 to N)
      V(fetch[mycount])
    mycount = (mycount + 1) % N;
  }
}
```

8 Applications 2 (CSc 525: Principles of Computer Networking)

The dominating end-to-end protocol in the Internet, TCP, uses a sliding window protocol. One important use of the sliding window protocol is to prevent congestion.

1. Draw a graph that shows the relationship between (sending) window size and throughput for a representative TCP connection. Let the X axis be window size, and the Y axis be throughput (bits/s). The curve will have three components. Describe what happens in the network in each of the three components.
2. Explain what point in the graph classic TCP congestion control detects. Outline how classic TCP congestion control would move along the curve in the diagram.
3. Explain which point in the graph TCP Vegas congestion control detects. Outline how TCP Vegas would move along the curve in the diagram.

Solution

1. The first component starts at $(0,0)$ and increases linearly to the point $(\text{delay} \times \text{bandwidth}, \text{bandwidth})$, where *delay* is the round-trip time for the TCP connection, and *bandwidth* is the bandwidth available to the TCP connection. Any increase in window size will increase throughput correspondingly as more of the available bandwidth is being used. This goes on until the window has size equal to the delay x bandwidth product, at which point the throughput is maximal.

The second component is constant throughput with increasing window size. Increased window size increases buffering in the network, but there is no increase in throughput. This goes on until buffering becomes large enough that some packets are dropped due to router buffer overflow.

The third component is decreasing throughput with increasing window size. This happens when increasing congestion results in increasing network packet drops and retransmissions.

2. Classic TCP congestion control detects the transition between components two and three. It initially increases the window size quickly (exponentially over time) until congestion occurs (i.e., until the end of the second component), after which the window size is halved. The window size then increases linearly over time until congestion occurs again, the window is halved, and the process repeats.

Classic TCP will often substantially overshoot and increase the window size far into component three during the initial quick increase. Recovery from the resulting losses can initially slow down data transfer substantially.

3. TCP Vegas detects the transition point between components one and two. As long as no drops are provoked by other competing TCPs, TCP Vegas will keep the window size 2-3 packet sizes more than the delay x bandwidth product, i.e., a little bit to the right of the transition between components one and two. Vegas will typically reach this point without provoking any packet losses, i.e., will not overshoot into component three.

9 Applications 3 (CSc 533: Computer Graphics)

Discuss pro and cons of radiosity methods, compared to ray tracing, as methods for rendering a single image, from a single viewpoint, once the BRDF function and illumination conditions are given for a 3D scene. Would your answer be different if many viewpoints are used (e.g., when making animation of the viewer moving inside the scene.)?

Solution

In the presence of extremely shiny surfaces, finding the paths of light might require very large depth of the path tracing tree. This might be very costly to compute. This problem does not occur when using radiosity methods. However, using radiosity methods requires much more information, so in general the cost of this process is significant. However, once the radiosity from each patch of the surfaces has been computed, computing many pictures is much easier, so this computational effort might pay off.

10 Applications 4 (CSc 560: Database Systems)

Answer the following questions about optimizing a sample SQL query below.

```
SELECT Name, Title, Sal, Dname
FROM Emp, Dept, Job
WHERE Emp.Dno = Dept.Dno AND Emp.Job = Job.Job
```

Table Dept has <Dno, Dname, Loc> attributes, table Emp has <Name, Dno, Job, Sal> attributes, and table Job has <Job, Title> attributes.

1. What are the interesting orders in the sample query? In the first pass, the System R Optimizer examines the cheapest single-relation access paths. Which access paths will be examined by the System R Optimizer for the Emp table?
2. A multi-way join query is generally processed as a series of two-way joins. Therefore, there may be quite a few plans with different join orders that should be considered for the query. List all the distinct join orders to be considered by the System R Optimizer. Note that join operations are commutative and associative but the cost of $A \bowtie B$ may be different from that of $B \bowtie A$, and the cost of $(A \bowtie B) \bowtie C$ may be different from that of $A \bowtie (B \bowtie C)$.

Solution

1. The interesting orders are Dno and Job attributes. Since both the attributes are part of the Emp table, the System R Optimizer examines three access paths, namely (1) one that will produce Emp tuples in sorted order by the Dno attribute values, (2) one that will produce Emp tuples in sorted order by the Job attribute values, and (3) one that scans the Emp table in no particular order.
2. The System R Optimizer considers only left-deep join plans:

```
(Dept  $\bowtie$  Emp)  $\bowtie$  Job
(Emp  $\bowtie$  Dept)  $\bowtie$  Job
(Job  $\bowtie$  Emp)  $\bowtie$  Dept
(Emp  $\bowtie$  Job)  $\bowtie$  Dept
```