Midterm Exam

CSC 173

12 October 2000

Directions

This exam has 13 questions, some of which have subparts. Each question indicates its point value. The total is 100 points. Questions 5 and 13 are optional; they are not part of the 100 points, but will count for extra credit. Please show your work here on the exam, in the space given. (Do not write on the backs or in the margins.) Blue books are available if you need scratch paper, but the proctor will collect only the exams.

I have tried to make the questions as clear and self-explanatory as possible. If you do not understand what a question is asking, make some reasonable assumption and *write that assumption down* next to your answer. The proctor has been instructed not to try to answer any questions during the exam.

You will have the entire class period to work. Good luck!

Questions 1 through 5 make use of the database in figure 1 (p. 2).

1. Suppose you are building an index for the ROUTE relation, and that flight number (FN) is the key.

   (a) (3 points) What data structure would you use to implement your index? (You don't have to show code or a diagram: just give a descriptive name.)

   Use a hash table keyed on flight number.

   (b) (3 points) Suppose you knew that flight numbers were consecutive integers from 100 to 999 (they aren't in the database shown). Would this change your answer? How?

   If the flight numbers are dense, use a characteristic array.

2. Consider the following query: "Which cities have arrivals by jet?"

   (a) (4 points) Translate this query into relational algebra.

$$\pi_{TC}(\sigma_{TP=jet}(\text{PLANE} \bowtie \text{ROUTE}))$$

ROUTE

flight number (FN)	from city (FC)	to city (TC)	distance in miles (DI)	scheduled departure time (SD)	scheduled arrival time (SA)	plane (PM)
1384	ROC	BOS	325	0650	0810	F101
1205	BOS	ROC	325	1730	1900	F101
110	BOS	LAX	3250	0900	1040	L1011
398	LAX	SFO	240	1120	1210	757
124	SFO	BOS	3180	1400	2310	L1011
448	BOS	DCA	340	0700	0830	737
540	DCA	BOS	340	1900	2030	737

FLIGHT

flight number (FN)	date (DT)	number of passengers (NP)	actual departure time (AD)	actual arrival time (AA)
1384	2000-10-09	40	0650	0800
1384	2000-10-10	12	0730	0842
1384	2000-10-11	30	0700	0815
1205	2000-10-09	42	1735	1900
1205	2000-10-10	8	1730	1855
1205	2000-10-11	20	1810	1932
110	2000-10-10	403	0940	1115
110	2000-10-11	365	0905	1035
398	2000-10-09	148	1150	1220
398	2000-10-10	95	1125	1210
124	2000-10-09	150	1500	2350
124	2000-10-10	414	1410	2310

PLANE

model (PM)	capacity (CA)	type (TP)
737	120	jet
747	460	jet
757	200	jet
L1011	440	jet
F101	45	prop

Figure 1: Simple database for questions 1 through 5. Departure and arrival times use a 24-hr (international/military) clock. Capacity (CA) in the PLANE relation indicates maximum number of passengers.

(b) (4 points) Give the result of this query when executed on the database of figure 1.

LAX
SFO
BOS
DCA

3. Consider the following relational algebra query:

$$\pi_{FN,DT}(\sigma_{NP/CA<.5 \ \wedge \ AA-SA>30}((\text{ROUTE} \bowtie \text{FLIGHT}) \bowtie \text{PLANE}))$$

(a) (4 points) Translate this query into plain English.

Which flights were less than half full, but arrived more than 30 minutes late?

(b) (4 points) Give the result of executing this query on the database of figure 1.

1384	*2000-10-10*
1205	*2000-10-11*
124	*2000-10-09*

4. (12 points) Show the result of pushing selection and projection operations as far inward as possible (i.e. moving them inside as many joins as possible) in the query of the previous question. Your answer should take the form of a modified relational algebra query.

$$\pi_{FN,DT}(\sigma_{NP/CA<.5}(\pi_{FN,DT,PM,NP}\{\sigma_{AA-SA>30}[\pi_{FN,SA,PM}(\text{ROUTE}) \bowtie \pi_{FN,DT,NP,AA}(\text{FLIGHT})]\} \bowtie \pi_{PM,CA}(\text{PLANE})))$$

5. (Extra credit; up to 15 points) Which of the projection pushes in the previous question are likely to be profitable in a database of realistic size (assuming no change in scheme)? Which are not? Why? Under what circumstances?

It is reasonable to assume that there is an index for ROUTE keyed on FN, an index for FLIGHT keyed by (FN, DT) pairs, and an index for PLANE keyed on PM.

The ROUTE $\bowtie$ FLIGHT join is likely to iterate over all elements of FLIGHT and use the index to look things up in ROUTE. It therefore does not make sense to push a projection of ROUTE inside this join. It does make sense, however, to push the projection of FLIGHT inside, since we'll be iterating over all tuples in that relation anyway.

Similarly, it probably does not make sense to push the projection of PLANE inside the second join, since we would use the index to avoid looking up all plane models, but it does make sense to do the projection of the results of the first join, since we're iterating over all tuples.

These observations suggest implementing

$$\pi_{FN,DT}(\sigma_{NP/CA<.5}(\pi_{FN,DT,PM,NP}\{\sigma_{AA-SA>30}[\text{ROUTE} \bowtie \pi_{FN,DT,NP,AA}(\text{FLIGHT})]\} \bowtie \text{PLANE}))$$

6. Suppose you were using a relation to represent the transition function of a finite automaton.

   (a) (3 points) What would be the scheme of the relation?

   state, input_char, new_state

   (b) (3 points) If the automaton were deterministic, which field(s) (attribute(s)) would constitute a key for the relation?

   the pair (state, input_char)

   (c) (4 points) If the automaton were non-deterministic, which field(s) would constitute a key?

   the entire scheme (nothing less)

7. (6 points) Why is it hard to decide which indices to build for a given relation? Why not create one for every combination of fields on which the user might want to perform a lookup operation?

   Insert and delete operations are slower when we have to maintain a bunch of indices. Extra indices also take space. We have to strike a balance.

8. (8 points) Briefly state the distinction between an abstract data type and a data structure.

   An abstract data type describes a collection of data in terms of what operations it supports and how the results of the operations are related to one another.

   A data structure determines how the data is organized in memory and, consequently, how fast the various operations can be performed.

9. (8 points) The following function contains an error that may cause its program to terminate abruptly with a run-time error. What is the bug? How would you fix it?

```
typedef struct node {
    char * s;
    struct node * next;
} node;

void delete_list(node *l) {
    while (l) {
        free(l);
        free(l->s);
        l = l->next;
    }
}
```

   Function `delete_list` *uses* `*l` *after freeing it. One solution is to replace the body of the loop with the following:*

```
node * t = l->next;
free(l->s);
free(l);
l = t;
```

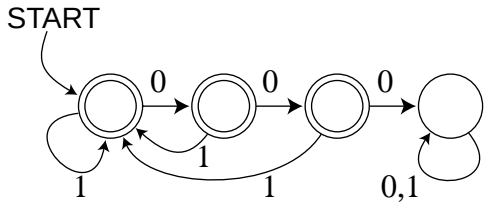
10. (6 points) Briefly explain why it is easier to create a computer program corresponding to a DFA than it is to create one corresponding to an NFA.

    Because the DFA has only one choice in any given situation: you don't have to keep track of multiple possible "guesses".

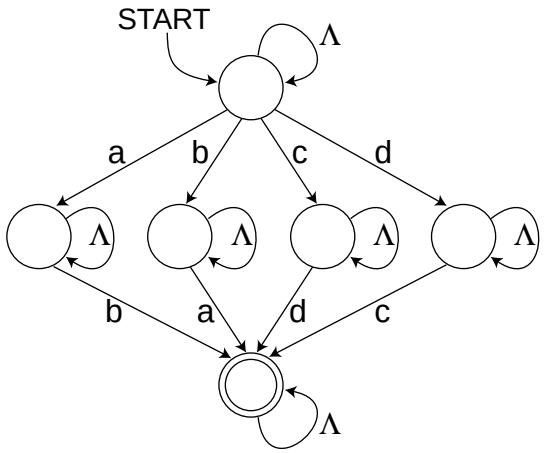
11. Consider the following regular expression: $(1 \mid 01 \mid 001)^* \ (\epsilon \mid 0 \mid 00)$

    (a) (5 points) Describe in English the language generated by this expression.
    The set of all binary strings with no more than two consecutive zeros.

    (b) (5 points) Give a DFA that accepts the same language. Note that your machine will need to consume its entire input before accepting or rejecting. Hint: start from scratch; don't try to apply the algorithms to create an NFA and turn it into a DFA.



12. Consider the following NFA:



    As usual, Λ represents all characters in the input alphabet.

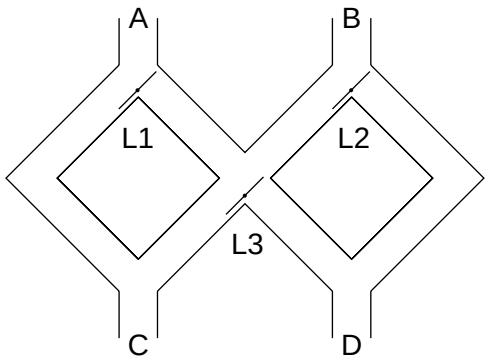
    (a) (5 points) Give a regular expression that generates the language accepted by this NFA. $\Lambda^*\ (\ a\Lambda^*b \mid b\Lambda^*a \mid c\Lambda^*d \mid d\Lambda^*c\ )\ \Lambda^*$

    (b) (5 points) Describe this language in English.
    The set of all strings containing an `a` *and a* `b`*, or a* `c` *and a* `d`.

(c) (8 points) Describe a DFA that accepts this language. What is the minimum number of states such a DFA must have? What do the states represent? Hint: start from scratch; don't try to apply the algorithms to create and minimize a DFA, starting from an NFA. Note that I am *not* asking you to draw a picture of the machine or to enumerate all the transitions.

The minimum DFA has 10 states: a start state, a single final state, and eight other states. Four of these indicate that the DFA has seen an `a`*, a* `b`*, a* `c`*, or a* `d`*, but none of the other interesting letters. The remaining four indicate that the DFA has seen an* `a` *and a* `c`*, a* `b` *and a* `c`*, an* `a` *and a* `d`*, or a* `b` *and a* `d`*, but none of the matching characters.*

13. (Extra credit, up to 20 points) Consider the following toy:



Marbles are dropped, one at a time, into opening A or B. Each marble is deflected by one or two of the levers shown, falling to the left or right depending on the direction in which the lever tilts. As a side effect, a lever that deflects a marble switches direction immediately afterward. The question arises: if we drop marbles into openings A and B in some order, which opening will the last marble come out of?

To answer the question, we can model the toy as a finite state machine. Let us say that a marble dropped into A corresponds to an "input character" of A, and a marble dropped into B corresponds to an "input character" of B. The toy "accepts" its input if the last marble comes out of D; otherwise it rejects.

Describe a DFA that correctly captures this model (you probably *don't* want to draw a picture).

We can define 16 states, each of which is labeled by a 4-tuple $< L_1 L_2 L_3 X >$*, where the first three elements indicate whether levers 1, 2, and 3, respectively, tilt to the left or the right, and the fourth element indicates whether the last marble came out of C or D. We can chart the transition function as follows. The first column indicates the current state. The second and third columns indicate where we go on an A or a B, respectively. All states ending in D are final. LLLC is the start state.*

		A	B		A	B	
start	$LLLC$	$RLLC$	$LRRC$	$LLLD$	$RLLC$	$LRRC$	*final*
	$LLRC$	$RLRC$	$LRLD$	$LLRD$	$RLRC$	$LRLD$	*final*
	$LRLC$	$RRLC$	$LLLD$	$LRLD$	$RRLC$	$LLLD$	*final*
	$LRRC$	$RRRC$	$LLRD$	$LRRD$	$RRRC$	$LLRD$	*final*
	$RLLC$	$LLRC$	$RRRC$	$RLLD$	$LLRC$	$RRRC$	*final*
	$RLRC$	$LLLD$	$RRLD$	$RLRD$	$LLLD$	$RRLD$	*final*
	$RRLC$	$LRRC$	$RLLD$	$RRLD$	$LRRC$	$RLLD$	*final*
	$RRRC$	$LRLD$	$RLRD$	$RRRD$	$LRLD$	$RLRD$	*final*

Three states—LRLC, LRRD, and RRRD—are unreachable. The minimal machine has 13 states.