UCLA
Computer Science Department
Fall 2006

Instructor: J. Cho

Student Name and ID: ALBERT LEE 603388255

CS246 Midterm: 1.5 Hours

*Attach extra pages as needed. Write your name and ID on the extra pages. If you need to make additional assumption to solve a problem, write it down on the sheet. A calculator and one-page double-sided cheatsheet are allowed.*

| Problem | Score | |
|---|---|---|
| 1 | 20 | 18 |
| 2 | 20 | 14 |
| 3 | 20 | 8 |
| 4 | 20 | 18 |
| 5 | 20 | 20 |
| Total | 100 | 78 |

-2
-6 -8
-12 -20
-2 -22

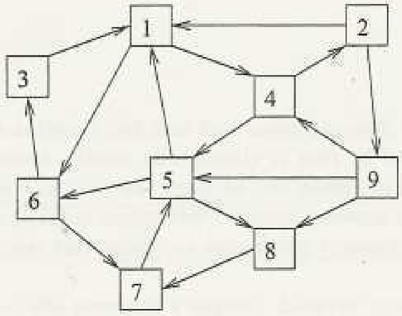
Exam Score:

AL

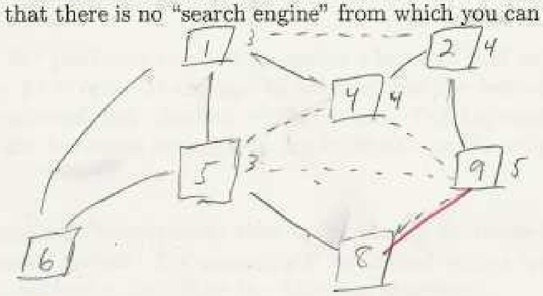
## Problem 1: 20 points

In order to take random samples from the following graph, you have run the WebWalker algorithm multiple times. In one of the runs, the algorithm visited the nodes in the sequence of 5, 1, 4, 2, 9.

d=6



Assuming that $d$, the maximum number of edges per node, is 6, draw the graph constructed by this particular run of the WebWalker algorithm. Please label each node with its node id. Also, in the space provided below, write down the number of self-loops for each of the visited nodes. Assume that there is no "search engine" from which you can obtain incoming links to a given node.



# of self loops appear to the right of each node; Graph is undirected and regular; dashed edges are not added

(-2)

Number of self-loops: gotten by subtracting degree of node from d

Node 5: 3 Node 1: 3 Node 4: 4 Node 2: 4 Node 9: 4

AL

## Problem 2: 20 points

We are building a query translation system that translates a mediator query into Amazon queries. The mediator provides the following basic predicates for its query interface:

```
LastName = $l
FirstName = $f
Title = $t
ISBN = $i
```

Here, users can specify an author's last and first name through the 'LastName = $l' and 'FirstName = $f' predicates. Users can specify a part of a book title through Title = $t predicate. Predicate 'ISBN = $i' returns the book with the given ISBN number. The mediator allows users to issue arbitrarily complex boolean queries based on the above predicates. The mediator uses the minimum subsuming translation.

Q $Q_{translated}$

Unfortunately, the Amazon site provides a slightly different query interface that supports the following predicates:

$Q_{translated}$ contains $Q_{original}$

```
Author = $l, $f
Keyword = $k
```

Here, 'Author = $l, $f' predicate specifies an author's last and first name separated by comma. The first name $f is optional and can be omitted, but the last name is not. That is, 'Author = $l,' is allowed, but 'Author = ,$f' is not. The Keyword = $k predicate returns all books with the keywords $k either in the book title or the subject field.

1. List all basic mappings (translations) that are necessary to translate any mediator queries into Amazon queries. For example, if we should map a query of the form Keyword = $k to Author = $k. Write the following mapping:

   Keyword = $k -> Author = $k

   (You may assume that amazon support the query True, which returns all books in amazon.)

   LastName = $l & FirstName = $f → Author = $l, $f

   LastName = $l → Author = $l

   Title = $t → Keyword = $t

   ISBN = $i → True

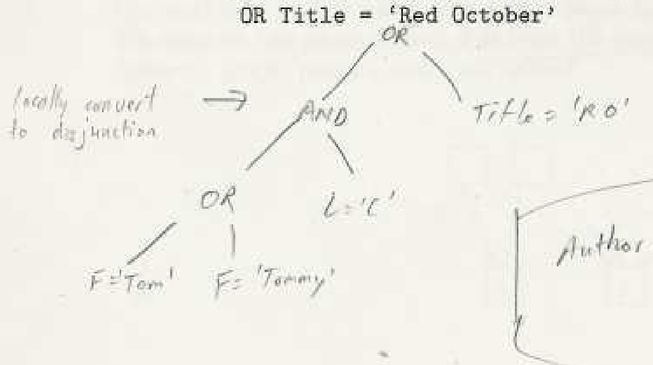
   FirstName = $f → True

   (−6)

AL

2. Using Algorithm TDQM, translate the following query:

```
((First = 'Tom' OR First = 'Tommy') AND Last = 'Clancy')
 OR Title = 'Red October'
```



Author = 'Clancy, Tom' OR Author = 'Clancy, Tommy' OR Keyword = 'Red October'

3. Again, using Algorithm TDQM, translate the following query:

```
Title = 'Red October' OR ISBN = '1234567890'
```

What will the system get from Amazon from the translated query?

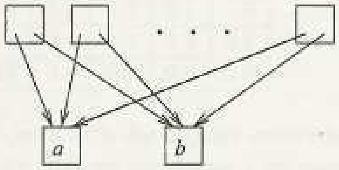
Keyword = 'Red October'

Since the translated query is minimally subsuming ($Q_{Translated}$ contains $Q_{original}$) we should get a superset of tuples from Amazon. But this seems to not be the case since ISBN search is not supported by Amazon.

A L

## Problem 3: 20 points

Our goal is to find all pairs of "related" pages from a collection of 100 million Web pages. We consider two pages related if at least 100 pages link to both pages. For example, in the following graph, pages $a$ and $b$ are related.



Describe an algorithm that can identify all pairs of related pages efficiently from the given dataset.

Build a matrix whose columns are all possible pages (one column for each page) and rows are all possible pages. There is a 1 in row r and column c if page r links to page c.

Two columns are related if they agree (both have a 1) on at least 100 rows. That is, Related $(c_i, c_j)$ iff $|c_i \cap c_j| \geq 100$.

We can now use min hash to make the matrix more succinct (while preserving similarity) and LSH to identify candidate pairs ((candidate pairs are) columns in each band that hash to the same bucket). Candidate pairs are checked to make sure they are actually related to eliminate false positives.

8

AL

## Problem 4: 20 points

We have a database of 9 objects, $a$, $b$, …, $i$. Our objects have three attributes $x_1$, $x_2$, and $x_3$. The attribute values of our objects are given in the following table:

| | $a$ | $b$ | $c$ | $d$ | $e$ | $f$ | $g$ | $h$ | $i$ |
|---|---|---|---|---|---|---|---|---|---|
| $x_1$ | 0.3 | 0.6 | 0.8 | 0.5 | 0.4 | 1.0 | 0.2 | 0.1 | 0 |
| $x_2$ | 0.2 | 0.6 | 0.8 | 0.4 | 0.3 | 0.5 | 1.0 | 0.9 | 0.7 |
| $x_3$ | 0.1 | 0.7 | 0.6 | 0.5 | 0.3 | 0.4 | 0.2 | 0.8 | 0.9 |

Our goal is to find the top-2 $\overset{k=2}{}$ objects with the highest score when we use the ranking function $f(x_1, x_2) = \min(x_1, \overset{x_3}{x_2})$. Assume that we can break a tie randomly (i.e., if two objects have the same score, it is okay to return either of the two objects).

For your convenience, the following table shows the sorted list of object values for each attribute: t=.9 .8 .6

threshold

| | |
|---|---|
| $x_1$ | ($f$:1.0), ($c$:0.8), ($b$:0.6), ($d$:0.5), ($e$:0.4), ($a$:0.3), ($g$:0.2), ($h$:0.1), ($i$:0) |
| $x_2$ | ($g$:1.0), ($h$:0.9), ($c$:0.8), ($i$:0.7), ($b$:0.6), ($f$:0.5), ($d$:0.4), ($e$:0.3), ($a$:0.2) |
| $x_3$ | ($i$:0.9), ($h$:0.8), ($b$:0.7), ($c$:0.6), ($d$:0.5), ($f$:0.4), ($e$:0.3), ($g$:0.2), ($a$:0.1) |

3 sorted accesses after each step

TA

1. When we use the Fagin's threshold algorithm, how many sorted accesses and random accesses do we perform? ✓ Stops after k obj grades are higher than max grade of unseen objects

| | $x_1$ | $x_2$ | $x_3$ | $\min(x_1,x_2,x_3)$ |
|---|---|---|---|---|
| f | 1 | .5 | .4 | .4 |
| g | .2 | 1 | .2 | .2 |
| i | 0 | .7 | .9 | 0 |
| c | .8 | .8 | .6 | .6 |
| h | .1 | .9 | .8 | .1 |
| b | .6 | .6 | .7 | .6 |

# of sorted accesses = (3 sorted access / step)(3 steps) = 9
assuming that we still do sorted access on c at step 3 even though we probed it earlier

# of rand accesses = 10 (gotten by the circled items)

b and c are returned since they have a score of .6 and threshold is .6

2. Assume that we use the algorithm MPro, where we use $x_1$ for sorted access and do random probing for the remaining attributes in the sequence of $x_2$ and $x_3$. How many random probing do we perform? threshold is grade of kth highest object

t=.4

f: (1, .5, .4) no need (−2)

c: (.8, .8, .6)

b: (.6, .6, .7)

t=.6

d: (.5, ~~.4~~, ~~.5~~) ← 2nd and 3rd attribute probes here aren't needed since threshold is currently .6 and we know that the largest score possible for d is .5 based on min scoring function

Since 1st attributes of each item was gotten by sorted access, there are a total of (2+2+2) = 6 random probes.

AL

## Problem 5: 20 points

We are running a company that caches two pages, $P_1$ and $P_2$. On average, $P_1$ changes once an hour and $P_2$ changes once a day. Because of our resource constraints, we can refresh only *one page per month*.

1. If we want to maximize the time averaged *Freshness*, how should we refresh $P_1$ and $P_2$? You don't have to give exact numbers. Roughly describe how often you would visit each page and explain your reasoning. (*Freshness* is the fraction of the pages that are up-to-date within our cache.)

Algorithm : Visit $P_2$ every month.

This maximizes our expected freshness. We expect that $P_2$ will remain fresh for about half a day if we refresh P2. On the other hand we expect P1 will remain fresh for about half an hour if we refresh P1. Since $\frac{1}{2}$ a day of freshness is better than $\frac{1}{2}$ an hour of freshness, we should always refresh P2.

2. If we want to minimize the time averaged *Age*, how should we refresh $P_1$ and $P_2$? Again, roughly describe how often you would visit each page and explain your reasoning. (*Age* of a cached page is 0 if the page is up-to-date, and *Age* is the time from modification if the page is obsolete.)

Alternate between refreshing $P_1$ and $P_2$ (each month) so that the age of each page is at most 2 months. Unlike the above algorithm, we can't afford to not refresh P1 or else its age will grow infinitely large. Also, we know that uniformly refreshing is better than proportionally refreshing to minimize age. Thus, we should alternately refresh P1 and $P_2$.