Organization of Web Information (CS 728) **Final Exam**
Computer Science and Engineering **2010-04-24 Saturday**
Indian Institute of Technology Bombay **14:30–17:30 SIC301**

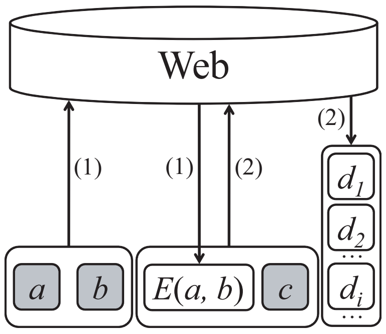
NAME ______________________ ROLL ______________________

This exam has 9 printed page/s. Write your roll number on **every sheet, because we may take apart your answer book for correction**. Write your answer clearly within the spaces provided and on any last blank page. **If you need more space than is provided, you probably made a mistake in interpreting the question.** Start with rough work elsewhere, but you need not attach rough work. Use the marks alongside each question for time management. **Illogical or incoherent answers are worse than wrong answers or even *no* answer, and may fetch negative credit.** You may not use any computing or communication device during the exam. You may use textbooks, class notes written by you, approved material downloaded **prior to the exam** from the course Web page, course news group, or the Internet, or notes made available by me for xeroxing. If you use class notes from other student/s, you must obtain them **prior to the exam** and **write down his/her/their name/s and roll number/s** here.

1. We are interested in building a query-by-analogy system that can respond to queries like "what Microsoft product is similar to Apple's iPod?" Such a query might be presented to the system in a GRE-like "$a : b :: c :$ ___", where the blank is to be filled with candidates d. Suppose we cannot afford to do Web-scale processing in-house and must depend on a public search engine.

1(a) The figure shows rounds of interaction of our system with the Web (including search engines). Describe subjectively the content sent between the blocks in the diagram, and the processing done at both ends, in temporal order. ("First, the system builds a query ... the search engine returns ... Our system collects statistics about ..." etc.) Read the whole question before answering this part.

		3



- Query the search engine with "a near b" or something close that the engine will process meaningfully. Also query with only a and only b. Let these sniipets/docs be called $\text{Doc}(a), \text{Doc}(b), \text{Doc}(a \wedge b)$.
- From the snippets (or documents) thus retrieved, locate the mentions of a and b, and, from the text surrounding them, induce patterns that appear significantly more often in $\text{Doc}(a \wedge b)$ compared to $\text{Doc}(a)$ or $\text{Doc}(b)$. We overload E to denote these patterns.
- Now ask queries of the form "E near c" or something close that the engine will process meaningfully.

- Identify candidate d_is in the returned snippets/docs. Find (samples of) snippets with d_i in them, unconditioned on c or E.
- Compare these sets to order d_is for reporting to the user.

1(b) Note that E is not known explicitly, but we must learn a (possibly probabilistic) pattern recognizer of E. (We will overload E to also denote the pattern/s and recognizer.) Let $\text{Doc}(a \wedge b)$ be the documents where a and b occur in close proximity. One proposal is to count words not equal to either a or b, that appear around them, in $\text{Doc}(a \wedge b)$, and pick some of the most frequent words. Why might this be a bad idea?

		1

We will pick up frequent and/or function words unless we contrast against other snippets containing a or b individually.

1(c) For each candidate pattern E, how can you use the query $E(a, \underset{\smile}{\ })$ to do something better than the counting proposal above? (Hint: "training loss".)

		2

One fix would be to do something like a χ-square test to retain words/patterns that appear significantly more often in $\text{Doc}(a \wedge b)$. Another way would be, given a candidate E_j, issue the query "E_j near a" (or "E_j near b") and see how often b (or a) crop up in the snippets. Then order E_j by this number, and pick some number of top E_js.

1(d) Eventually we expect to have several good patterns E_j. What kind of queries are issued in the second round? Give both a generic template and specific examples.

		2

For each qualifying E_j issue the query "E_j near c".

1(e) Propose how candidate d_is can be extracted and ranked from the responses to the second-round queries. (There is no single best/correct answer.)

		2

For each query as above, collect a list of candidate d_is. Now we can either globally count up d_is across all patterns E_j, or keep a separate list for each E_j, and do some kind of rank aggregation of d_is over the different ranked lists. As different E_js may have diverse resolving power, the latter strategy may be better.

2. A Web search engine can benefit from named entity recognition in queries as well as documents. Given a query q, the "Q-NER" problem is to find the most likely tuple (e, t, c) where e is an entity like *Harry Potter*, t is a token context like "walkthrough" and c is a class such as *Game*. Suppose that we have named entity *Harry Potter* with classes *Movie*, *Book*, and *Game*, and find three queries containing "harry potter" in the query log: "harry potter movie", "harry potter walkthrough", and "reviews of harry potter". Then the contexts would be written as `# movie`, `# walkthrough`, and `reviews of #`.

The assumption is that some tokens in t reveal information about c over and above the mention of e itself. Thus, the mention tokens of e and the context tokens in t account for a subset of the text in the query. If t embeds mentions of other entities, we ignore that in the specific tuple (e, t, c). Standard entity and class IDs may be provided by a catalog like YAGO.

2(a) Why is c an important output of the query analysis and we cannot be satisfied with just (e, t)?

		1

Note that the class c is not instantiated in the query, but the query seeks an entity of class c. There is world knowledge embedded in c beyond what is found in e and t. E.g., if e

is the book *Black Swan* and t is "reviews", then c is *book reviews*, whereas if e is the entity *Harry Potter*, c could be either book or movie reviews. For a subsequent query module to commence work, guesses at c are required.

2(b) In the above setting, is it reasonable to assume that t is conditionally independent of e given c? If so, justify; if not, give counterexamples. (Note: the answer to this part need not be consistent with the remaining question.)

		2

That depends on the purpose of query modeling. In general, the assumption does not hold. (If you only say this much, you will get partial credit.) But, if we assume that the purpose of this analysis is to be able to respond with an attribute of entity e, where the attribute is of general category c, and the identity of the attribute is hinted by context tokens t, then the assumption is reasonable. E.g., we may be asking for the battery life of a specific laptop e, in which case c is "time duration", and t consists of tokens that indicate that *battery* life is what we are after.

2(c) Making the assumption anyway, how would you write

$$\Pr(e,t,c) = \Pr(e)\,\utilde{\hspace{6cm}}$$

(complete)?

		2

$$\Pr(e,t,c) = \Pr(e)\Pr(t,c|e) = \Pr(e)\Pr(c|e)\Pr(t|e,c) = \Pr(e)\utilde{\Pr(c|e)\Pr(t|c)},$$

where the last step makes use of the assumption.

2(d) For the moment, assume that detecting e in a query is not fraught with ambiguity, i.e., we know e for sure. (We can discard queries in the logs that have ambiguous mentions.) However, labeling c in each query would be very laborious. In the absence of knowledge about c_i, what is the overall probability of N extraction events (complete the expression below)?

$$\prod_{i=1}^{N}\Pr(e_i)\sum_{\utilde{\quad}}\Pr(\utilde{\quad}|e_i)\Pr(\utilde{\quad}|\utilde{\quad}).$$

		2

$$\prod_{i=1}^{N}\Pr(e_i)\sum_{c}\Pr(\utilde{c}|e_i)\Pr(t|\utilde{c})$$

2(e) Can you recognize something similar to a probabilistic model we have discussed? Write down the artifacts corresponding to entities, contexts and classes in that model. What is one important distinction from that model?

		2

3. Consider an undirected graph $G=(V,E)$ with $|V|=n$, $|E|=m$ and edge weights $w(u,v)>0$. Let $N(v)$ denote the neighbors of node v. Each node v has an associated non-empty set of words $P(v)$. A query is a set of words $\mathbf{P}$ of size ℓ. For a word p, $V_p \subseteq V$ are nodes that contain p. Let

$\mathbf{p}$ be any subset of $\mathbf{P}$. A subgraph T of G *hits* $\mathbf{p}$ if $V(T) \cap V_p \neq \varnothing$ for each $p \in \mathbf{p}$. The response to the query must be a connected tree T that hits $\mathbf{P}$, such that the cost of the tree, defined as $s(T) = \sum_{(u,v)\in E(T)} w(u,v)$, is minimized. Let $T(v, \mathbf{p}, h)$ be a tree with minimum cost, rooted (arbitrarily) at v, with height at most h, such that it hits $\mathbf{p}$. Let the corresponding cost be $C(v, \mathbf{p}, h)$.

3(a) What is the proper way to define the base cases

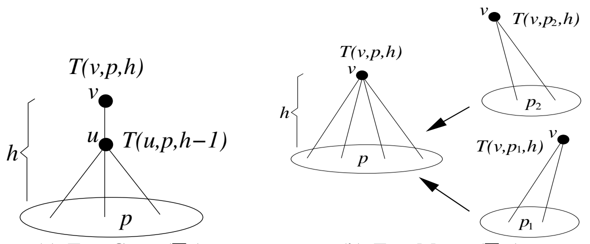
$$C(v,\mathbf{p},0) = \begin{cases} \underline{\qquad\qquad\qquad\qquad\qquad\qquad}, & \mathbf{p} \subseteq P(v) \\ \underline{\qquad\qquad\qquad\qquad\qquad\qquad}, & \text{otherwise} \end{cases}$$

for any node v? Note that this means we are free to logically ignore any subset of words in a node.

		2

$$C(v,\mathbf{p},0) = \begin{cases} \underset{\sim}{0}, & \mathbf{p} \subseteq P(v) \\ \underset{\sim}{\infty}, & \text{otherwise} \end{cases}$$

3(b) Any undirected tree can be built up through a series of *grow* and *merge* operations as shown below (read all p as boldfaced $\mathbf{p}$).



(a) Tree Grow (T_g) (b) Tree Merge (T_m)

The grow operation takes a tree $T(u, \mathbf{p}, h-1)$ and attaches a new root node v. The merge operation takes two trees with the same root node v and attaches them at v. In both cases, cycles created can be cleaned up by deleting the heaviest edge. In the grow operation, we will insist that $P(v) \cap \mathbf{p} = \varnothing$. In the merge operation, we will insist that $P(v), \mathbf{p}_1, \mathbf{p}_2$ are pairwise disjoint. Why can we afford to do this without losing any optimal tree?

		1

From the previous part, note that we have effectively made it free to regard any node as having any subset of the words that it already contains. Therefore, in any final answer tree that has nodes with overlapping words, we can remove words from nodes in such a way that the query word set is satisfied using disjoint unions, i.e., exactly one node provides each word.

3(c) Complete the following inequalities:

		3

Monotonicity: $C(v,\mathbf{p},h) \leq C(v, \underline{\qquad\qquad}, h-1)$

Grow: $C(v, \underline{\qquad\qquad}, h) \leq \underline{\qquad\qquad\qquad} + C(u,\mathbf{p},h-1)$

Merge: $C(v, \underline{\qquad\qquad\qquad}, h) \leq C(v, \underline{\qquad\qquad}, h) + C(v, \underline{\qquad\qquad}, h)$

```
initialize cells $C(v,\mathbf{p},h)$ for all $v \in V$, all $\mathbf{p} \subseteq$ ______, and all $h \in [0,$ ______$]$
for $h = 0, 1, \ldots$ up to the upper limit specified above do
  for each node $v \in V(G)$ do
    for each possible value of $\mathbf{p}$ do
      if $h = 0$ then
        set $C(v,\mathbf{p},0)$ to base case values
      else
        initialize $C(v,\mathbf{p},h) \leftarrow C(v,\mathbf{p},$ ______$)$
        for every neighbor $u \in N(v)$ do
          for each $\mathbf{p}_1$ such that $\mathbf{p}_1$ ______ $P(v)$ ______ $\mathbf{p}$ do

        for every pair of disjoint trees $T_1, T_2$ rooted at $v$ do
          for each $\mathbf{p}_1, \mathbf{p}_2$ s.t. $\mathbf{p}_1, \mathbf{p}_2, P(v)$ pairwise disjoint and ______ do

        finalize the value of $C(v,\mathbf{p},h)$
```

Figure 1: Dynamic programming pseudocode for keyword search in graphs.

Monotonicity: $C(v,\mathbf{p},h) \le C(v,\mathbf{p},h-1)$

Grow: $C(v,\underline{P(v)\cup\mathbf{p}},h) \le \underline{w(v,u)} + C(u,\mathbf{p},h-1)$

Merge: $C(v,\underline{P(v)\cup\mathbf{p}_1\cup\mathbf{p}_2},h) \le C(v,\mathbf{p}_1,h) + C(v,\mathbf{p}_2,h) + \underline{C(v,P(v),0)}$

3(d) Complete the pseudocode in Figure 1.

		4

3(e) How much space will be needed to store table C? Why is this not necessarily a disaster?

		2

There are n choices for v, 2^ℓ choices for $\mathbf{p}$, and at most n choices for h, so the overall space is $O(2^\ell n^2)$. In practice ℓ is very small, typically under 4.

3(f) Give a reasonable (asymptotic) upper bound on the time taken to fill the table. What is the nature of dependence of the time on ℓ, n, m and any other graph properties?

		2

```
initialize cells $C(v, \mathbf{p}, h)$ for all $v \in V$, all $\mathbf{p} \subseteq \underline{\mathbf{P}}$, and all $h \in [0, \underline{\text{diameter of } G}]$
for $h = 0, 1, \ldots$ up to the upper limit specified above do
  for each node $v \in V(G)$ do
    for each possible value of $\mathbf{p}$ do
      if $h = 0$ then
        set $C(v, \mathbf{p}, 0)$ to base case values
      else
        initialize $C(v, \mathbf{p}, h) \leftarrow C(v, \mathbf{p}, \underline{h-1})$
        for every neighbor $u \in N(v)$ do
          for each $\mathbf{p}_1$ such that $\mathbf{p}_1 \underline{\cup} P(v) \underline{\supseteq} \mathbf{p}$ do
            $\underline{C(v, \mathbf{p}, h) \leftarrow \min\{C(v, \mathbf{p}, h), w(v, u) + C(u, \mathbf{p}_1, h-1)\}}$
        for every pair of disjoint trees $T_1, T_2$ rooted at $v$ do
          for each $\mathbf{p}_1, \mathbf{p}_2$ s.t. $\mathbf{p}_1, \mathbf{p}_2, P(v)$ pairwise disjoint and $\underline{\mathbf{p}_1 \cup \mathbf{p}_2 \cup P(v) \supseteq \mathbf{p}}$ do
            $\underline{C(v, \mathbf{p}, h) \leftarrow \min\{C(v, \mathbf{p}, h), C(v, \mathbf{p}_1, h) + C(v, \mathbf{p}_2, h) + C(v, P(v), 0)\}}$
        finalize the value of $C(v, \mathbf{p}, h)$
```

Figure 2: Completed dynamic programming pseudocode for keyword search in graphs.

h ranges over $O(n)$ values. Inside the h loop, n nodes are iterated through. The iteration through $\mathbf{p}$ leads to another factor of 2^ℓ. This accounts for the innermost two loops being executed $O(n^2 2^\ell)$ times. The iteration over $N(v)$ adds up to $O(m)$, so the innermost loop is executed $O(n^2 2^\ell m)$ times. The innermost loop is executed at most 2^ℓ times. Then we need time to check for disjointedness etc., which can be done in $O(n \log n)$ time. So overall, $O(4^\ell m n^3 \log n)$ is an upper bound, but it is probably loose. But what matters is that it is polynomial in m, n even if it exponential in ℓ.

4. We are given an entity and type catalog that is a tree, with internal nodes representing types and leaf nodes representing entities. Each entity e is connected by one $\in$ link to the most specific type T it belongs to, and each type T_1 (except for "entity", the root type) is connected to one immediate supertype T_2 using a $\subset$ link. We are also given a corpus, which is a set of documents. Each document is a sequence of tokens. (We will ignore punctuation and sentence boundaries.) Each token is linked to zero or one entity node. The simplest kind of query is a pair (T, w) where T is a type and w is a token. The desired response is a set of token sequences, each at most L tokens long (for some specified window width parameter L), containing at least one occurrence of token w and at least one occurrence of some token linked to an entity $e \in^+ T$, which is shorthand for $e \in T_1 \subset T_2 \subset \cdots \subset T$.

4(a) Write informal pseudocode to associate intervals $I_T = (\ell_T, r_T) \in \mathbb{Z}_+^2$ with each type T and number $c_e \in \mathbb{Z}_+$ with each entity e such that the query "$e \in^+ T$" translates to "$c_e \geq \ell_T \wedge c_e \leq r_T$", and similarly the query "$T_1 \subset^+ T_2$" translates to "$\ell_{T_1} \geq \ell_{T_2} \wedge r_{T_1} \leq r_{T_2}$". A single integer c_e can be considered as a degenerate interval (c_e, c_e). We will write interval containment as a subset relation: if interval I_1 is contained in interval I_2, we write $I_1 \subseteq I_2$.

		2

```
embed the catalog tree in the plane without any crossings
number the $n$ leaves (entities) arbitrarily from 1 to $n$; let these be $c_e$ for each node $e$
for each internal (type node) $T$ do
  set $\ell_T \leftarrow \min_{e \in^+ T} c_e$
  set $r_T \leftarrow \max_{e \in^+ T} c_e$
```

4(b) What is the smallest range of integers within which ℓ, c, r can lie and still achieve the above function?

		1

$[1, n]$ where n is the number of leaves.

4(c) We have discussed in class that, to answer the given kind of queries, we can index any token w that is connected to e together with the path of types from e to the root type as being at that same token position. But this can consume much index space. Instead, consider the following scheme:

- Carefully choose a set $\mathcal{I}$ of intervals, perhaps based on query workload from the past.
- At indexing time, when token t linked to entity e is encountered in the text stream, instead of simulating the occurrence of all T such that $e \in^+ T$ at that offset, $\underset{\sim\sim\sim\sim}{}$.

Complete the scheme including data structures and representations used.

		2

In the exhaustive scheme, every internal type node became a synthetic word in the vocabulary. In the proposed scheme, for each interval $I \in \mathcal{I}$, we will create a synthetic word. When we encounter t linked to e in the token stream, we will enumerate all $I \in \mathcal{I}$ such that $c_e \in I$, and only inject the corresponding interval tokens into the stream indexer.

4(d) During query time, in the query (T, w), T maps to an interval I_T. Given our reduced index, we need to find some subset $\mathcal{I}(T)$ of intervals from $\mathcal{I}$ such that (complete)

		1

$$I_T \underset{\sim\sim\sim\sim\sim\sim\sim\sim}{} \bigcup_{I' \in \underset{\sim\sim\sim\sim}{}} I'.$$

We require that $I_T \underset{\approx}{\subseteq} \bigcup_{I' \in \underset{\sim\sim}{\mathcal{I}(T)}} I'$. In words, we have to find an $\mathcal{I}(T) \subseteq \mathcal{I}$ that *covers* I_T.

4(e) Write informal pseudocode describing what you need to do with the posting lists corresponding to each such I' and the token w to be able to respond to the query.

		1

Once we find the cover $\mathcal{I}(T)$, we open cursors on the posting lists for all $I \in \mathcal{I}(T)$ and also a cursor on the posting list of w. We scan these postings, looking for document IDs (and offsets) where at least one of the interval pseudo-words appear within L tokens of an occurrence of w.

4(f) The time cost of the above code has two components:

- The number of intervals, $|\mathcal{I}(T)|$.
- The total posting length over all intervals in $|\mathcal{I}(T)|$.

If the posting list corresponding to interval I has length P_I, the cost can be modeled as $|\mathcal{I}(T)| + a \sum_{I' \in \mathcal{I}(T)} P_{I'}$ where a is a suitable constant. Justify these assumptions and model. On what system performance constants does a depend?

		2

During the merge scan, we have to inspect all elements in all postings in question, this will take time proportional to $P_w + \sum_{I \in \mathcal{I}(T)} P_I$. In one implementation involving a heap, the time per merge step will be $O(\log |\mathcal{I}(T)|)$, but we expect the number of intervals in the cover to be small compared to the total posting length of intervals. Initialization time will be proportional to $|\mathcal{I}(T)| + 1$. However, it might be more accurate to estimate the cost as

$$|1 + \mathcal{I}(T)| + a \log |1 + \mathcal{I}(T)| \left(P_w + \sum_{I' \in \mathcal{I}(T)} P_{I'} \right) \qquad \text{(CostModel)}$$

Since our goal is to optimize over choices of intervals and P_w is fixed, it can be removed from the cost model for our purposes. The constant a depends on the relative fixed cost of initializing a posting merge as compared to the unit cost of scanning through a posting entry.

4(g) Suppose $\mathcal{I}$ has N intervals in general positions (no coincident begin or end point). Then there are $2N$ distinct interval endpoints. If these are all projected to a number line, they induce two half-open intervals and $2N-1$ closed intervals, for a total of $2N+1$ projected intervals. If two queries $(T_1, \cdot)$ and $(T_2, \cdot)$ with $I_{T_1} = (\ell_1, r_1)$ and $I_{T_2} = (\ell_2, r_2)$ are such that ℓ_1, ℓ_2 fall in the same projected interval and so do r_1, r_2, is there any reason to choose $\mathcal{I}(T_1) \neq \mathcal{I}(T_2)$, assuming the cost model above (and ignoring other tokens in queries)? Either show an example that justifies the choice, or prove that $\mathcal{I}(T_1) = \mathcal{I}(T_2)$ will suffice.

		2

Strictly speaking, the exact form of (CostModel) is such that a variation in P_w from query to query may change the optimal cover even if ℓ_1, ℓ_2 and r_1, r_2 fall in the same projected intervals. E.g., if $P_{w_1} \gg P_{w_2}$, aggressively reducing $\mathcal{I}(T_1)$ as compared to $\mathcal{I}(T_2)$ may prove worthwhile. In practice, however, interval pseudo-words will tend to be much more frequent in the corpus than ordinary words, because they are effectively unions of many words, and so the sum in (CostModel) will tend to be dominated by the posting lists for interval pseudo-words. If that assumption is valid, then there will be no benefit to choosing a different cover plan for the two queries.

4(h) How does the above result help us in designing an algorithm to find $\mathcal{I}(T)$, given $\mathcal{I}$ and T?

		1

If we can assume that P_w does not affect the cover plan, then we can precompute and store the best cover for each query interval, and just look it up at query time. Otherwise, the time for cover computation will be added to overall query processing time.

4(i) Give a dynamic programming algorithm to find $\mathcal{I}(T)$, given $\mathcal{I}$ and T. Number the projected endpoints $-\infty = x_0 < x_1 < \cdots < x_{2N} < x_{2N+1} = +\infty$. The dynamic programming array A will have three indices (i, j, k). Cell $A(i, j, k)$ will record the cost of the best subset of intervals, $\mathcal{I}_{i,j} \subseteq \mathcal{I}$ for query interval (x_i, x_j), such that the rightmost point of $\mathcal{I}_{i,j}$ ends at x_k. Give precise pseudocode for filling in the cells of A, starting from the base cases. (If you can give a more efficient algorithm you will get extra credit.)

		3

- If $i > j$, the interval to be covered is empty, so there is no cost and we set $A(i, j, k) = 0$ for all k.
- For all i, j, k such that $i \leq j$, if $x_k < x_j$, then there can be no cover, and we set the cost $A(i, j, k) = \infty$.
- Otherwise, given there are no coincident begin or end points, exactly one interval ends at x_k. Suppose this interval begins at x_ℓ. The interval left to be covered now is $[x_i, x_\ell]$, with the understanding that this is empty if $\ell < i$. Therefore, in this case,

$$A(i, j, k) = \underbrace{1 + a|P_{[\ell,k]}|}_{\text{interval cost}} + \min_{k' \geq \ell} A(i, \ell, k')$$

We should fill up the table in increasing order of $j - i$, so that values we need at each step are computed and available.

4(j) What is the time taken to fill the table?

		1

i, j, k each cycle through N values. In filling each cell we spend $O(N)$ time. So the total time is $O(N^4)$. This is why precomputation would be desirable.

4(k) How do you retain and read off the final answer/s?

		1

In each cell we would have to remember the identity of the interval $[\ell, k]$ (or these two numbers). Then we can easily trace back and build the cover.

Total: 50