Algorithms for AI and NLP (Fall 2009)
— A Sample Exam —

General Instructions

- Please read through the complete exam once before starting to answer questions. About thirty minutes into the exam, the instructor will come around to answer questions of clarification.
- Please follow the instructions closely. Most of the questions ask for short answers. When a maximum length is given (e.g. 'in no more than two sentences'), please try to stick to those limits.
- As discussed in class, the exam is given in English, but you are free to answer in any of *Bokmål*, English, or *Nynorsk*.

1 Finite-State Technology and General Search

(a) Draw a finite-state automaton (FSA) that recognizes the language $\texttt{a}^n$, where n is a number greater or equal to zero and divisible by three or four. Thus, the automaton should recognize strings like `aaa`, `aaaa`, `aaaaaa`, `aaaaaaaa`, etc.

(b) Recall the distinction between deterministic and non-deterministic FSAs. What does it mean for an FSA to be non-deterministic? Is your solution to part (a) above deterministic or not?

(c) Recall the notions of memoization and dynamic programming: give an example of a problem that benefits from dynamic programming. In no more than two sentences, what is the general idea in memoization, and which types of computation lend themselves especially well to dynamic programming?

2 Hidden Markov Models

Assume the following part-of-speech tagged training 'corpus' of just one sentence:

still	,	time	s	move	is	being	received	well	,	once	again	.
RB	,	NNP	POS	NN	VBZ	VBG	VBN	RB	,	RB	RB	.

(a) In a few sentences, discuss the concept of smoothing and explain why it is important. Next, ignoring smoothing and making the standard simplifying assumptions for a naïve bi-gram HMM (including the assumption that the training corpus provides the full inventory of distinct tags and complete vocabulary), calculate the following:

(i) For each tag t, the probability of t following the tag RB, i.e. $P(t|\text{RB})$

(ii) The emission probabilities $P(move|\text{NNP})$, $P(move|\text{NN})$, and $P(well|\text{RB})$.

(b) Assume further that for each tag t, $P(t|\texttt{<s>}) = P(t)$; in one sentence, what does it mean to make this assumption. Construct part of the Viterbi trellis for tagging the utterance *once again , time .* Rather than calculating all values, indicate the total size of the trellis and the computations for filling in the first two columns.

(c) In a few sentences, summarize the key points of the Viterbi algorithm. What is the interpretation of each cell in the trellis? What is the complexity of the algorithm, i.e. the number of computations performed in relation to (i) the length of the input sequence and (b) the size of the tag set? Very briefly, sketch an alternative, naïve method for computing the most probable tag sequence t_1^n, given an input string w_1^n; state how the Viterbi algorithm improves over this approach.

3 Classification and Clustering

(a) Sketch the main steps of the k-Means clustering algorithm. Briefly discuss both the problems and features associated with this particular clustering method.

(b) There exists two 'families' of hierarchical clustering methods; agglomerative and divisive. Give a comparative description of these two frameworks.

(c) Consider the problem of classification in the context of vector space models. There are several ways we can choose to represent classes or collections of objects in such a model. Discuss the alternaties.

(d) Explain the concept of *decision boundaries* in relation to classification.

4 Context-Free Grammars and Parsing

Consider the language defined by the following grammar:

S → VP NP	NP → *kim*
VP → PP VP	NP → *oslo*
NP → PP NP	NP → *snow*
VP → NP V	V → *adores*
PP → NP P	P → *in*

(a) For each of the following items, identify the number of readings (distinct analyses) that the grammar of Mirror English assigns, and draw the parse trees for each of the readings.

(i) *oslo in snow adores kim.*

(ii) *kim adores snow in oslo.*

(iii) *snow adores in oslo kim.*

(b) Identify which of the following parsing strategies, if any, will run into difficulties with the grammar of Mirror English, and briefly explain why: (i) top-down parsing or (ii) bottom-up parsing. Is the grammar of Mirror English suitable for use with the CKY parser? If so, why? If not, why not?

(c) In a few sentences, discuss the concept of *local ambiguity* that is at the core of the CKY and generalized chart parsers. Do any of the examples (i) to (iii) from part (a) above contain local ambiguity? If so, where exactly, and what would happen in chart parsing.

(d) Recall very briefly the role of the chart in the CKY and generalized chart parsers. What types of information are associated with each edge in the chart? How are active edges different from passive ones, and what is the general form of the *fundamental rule* in chart parsing?

5 Common Lisp

(a) Describe the effects of the function `?()` below.

```
(defun ? (foo)
  (if (atom foo)
     foo
     (cons (? (first foo)) (? (rest foo)))))
```

(b) How many elements are contained in the list returned by the following expression? What will happen when we use the function `length()` to count them? Show the 'box notation' used to keep track of `cons()` cells.

```
(let ((foo (list 42)))
  (setf (rest foo) foo))
```

(c) At various points in the class we talked about data structures to index and efficiently retrieve information, for example the so-called transition and emission matrices in the context of HMMs. Recall that, for the emission matrix, we typically looked up probabilities for a pair of the current state (encoded as an integer) and the current word (a string). Given this scenario, briefly discuss the relative strengths and weaknesses of lists, arrays, and hash tables for associative retrieval, i.e. the look-up of values associated with keys that (conceptually) pair an integer with a string. Reflect on the number of distinct values (in a typical HMM, say one used for PoS tagging) along both dimensions, and further take into consideration whether you expect the emission matrix to be densely or sparsely populated, i.e. what proportion of combinations (state plus word) out of the set of possible combinations will typically be used.

(d) Write a two-place function `ditch()` that takes an atom as its first and a list as its second argument; `ditch()` removes *all* occurrences of the atom in the list, e.g.

```
? (ditch 'c '(a b c d e c))
→ (A B D E)
? (ditch 'f '(a b c d e c))
→ (A B C D E C)
```

Note that we are not primarily concerned with specific details of Lisp syntax here. If you find that easier, feel free to use elements of 'pseudo code' in your function definition, as long as it is clear how exactly everything will work. Give a brief informal summary of the basic principles (e.g. in terms of base case(s) vs. recursive cases) and general approach of your implementation.