CSCI 5454 **Prof. Aaron Clauset**
Problem Set 4, due April 8 **Spring 2013, CU-Boulder**

1. (30 pts total) You and Gollum are having a competition to see who can compute the nth Fibonacci number more quickly. Gollum asserts the classic recursive algorithm:

```
Fib(n) {
   if n < 2
      return n
   else
      return Fib(n-1) + Fib(n-2)
   end
}
```

which he claims takes $R(n) = R(n-1) + R(n-2) + c = O(\phi^n)$ time and space.

You counter with a dynamic programming algorithm, which you claim is asymptotically faster. Dynamic programming is a kind of recursive strategy in which instead of simply dividing a problem of size n into two smaller problems whose solutions can be merged, we instead construct a solution of size n by merging smaller solutions, starting with the base cases. The difference is that in this "bottom up" approach, we can reuse solutions to smaller problems without having to recompute them.

(a) (15 pts total) You suggest that by "memoizing" (a.k.a. memorizing) the intermediate Fibonacci numbers, by storing them in an array `F[n]`, larger Fibonacci numbers can be computed more quickly. Your assert the following algorithm.[1]

```
MemFib(n) {
   if n < 2
      return n
   else
      if (F[n] == undefined)
         F[n] = MemFib(n-1) + MemFib(n-2)
      end
      return F[n]
   end
}
```

i. (7 pts) Describe the behavior of `MemFib(n)` in terms of a traversal of a computation tree. Describe how the array `F` is filled.

[1]Gollum briefly wails about your `F[n]=undefined` trick ("an unallocated array!"), but you point out that `MemFib(n)` can simply be wrapped within a second function that first allocates an array of size n, initializes each entry to `undefined`, and then calls `MemFib(n)` as given.

ii. (8 pts) Compute the asymptotic running time. Prove this result by induction.

(b) (5 pts) Golum then claims that he can beat your algorithm in both time and space by eliminating the recursion completely and building up directly to the final solution by filling the F array in order. Gollum's new algorithm[2] is

```
DynFib(n) {
   F[0] = 0
   F[1] = 1
   for i = 2 to n
      F[i] = F[i-1] + F[i-2]
   end
   return F[n]
}
```

How much time and space does `DynFib(n)` take? Justify your claim and compare your answers to those of your solution in part (a).

(c) (5 pts) With a gleam in your eye, you tell Gollum that you can do it even more efficiently because you do not, in fact, need to store all the intermediate results. Over Gollum's pathetic cries, you say

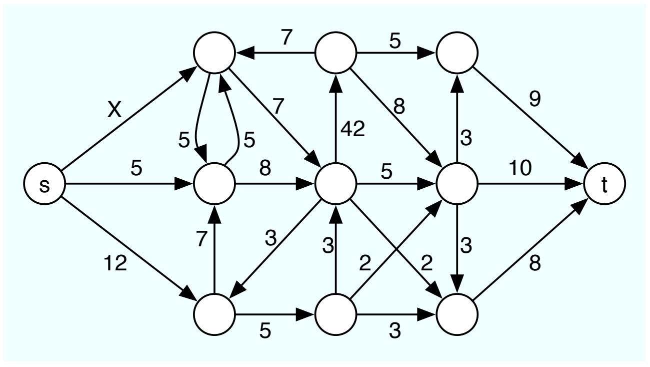
```
FasterFib(n) {
   a = 0
   b = 1
   for i = 2 to n
      c = a + b
      a = b
      b = c
   end
   return b
}
```

Derive the time and space requirements for `FasterFib(n)`. Justify your claims.

(d) (5 pts) Give a table that lists each of the four algorithms, its asymptotic time and space requirements, and the data structures each requires. Briefly compare and contrast the algorithms in these terms.

[2]Note that Gollum is now using your undefined array trick; assume he also uses your solution of wrapping this function within another that correctly allocates the array.

2. (10 pts total) The Engineers of Gondor have installed a set canals that convey water from the spring s to the town t. (They couldn't install just one big canal for technical reasons. Specifically: they're not dwarves.) Now, they are considering adding a new canal connecting the spring to their distribution network G. However, the Engineers are not sure how much additional water they will be able to push through G after adding the proposed canal; they need your help to figure it out. The diagram below shows G', the network G plus the proposed canal X.



   (a) (5 pts) Make a diagram showing the minimum cut corresponding to the maximum flow for G. What is the weight of that cut? If the Engineers add the canal X, what should be its capacity in order to maximize the increase in the water flow across the network? Explain.

   (b) (5 pts) Describe how the Engineers could use the min-cut/max-flow algorithm to decide what capacity X should be used for an arbitrary graph $G = (V, E)$ and arbitrary proposed edge $(u, v) \notin E$ with capacity X.

3. (30 pts total) As you and Samwise are trudging to Mordor, you begin discussing games for the party you're going to throw when you return to The Shire. Samwise suggests seating the n guests in a row, and assigning each guest a number, which may be positive, negative, or zero. Let the list of these values be given by the array $A[1 \ldots n]$.

   (a) (15 pts) In Sam's first game, guests must efficiently find the largest sum of elements in some subarray $A[i \ldots j]$, where we define an empty subarray to have a sum of zero. Describe (explain in words and give pseudocode) and analyze (prove the correctness of and give the running time for) a dynamic programming algorithm that solves this problem in $\Theta(n^2)$ time. For example, if the input is the array $A = [-6, 12, -7, 0, 14, -7, 5]$, then the largest sum of any subarray is $19 = 12 - 7 + 0 + 14 = sum(A[2 \ldots 5])$.

Gandalf's hint: How you can compute, store and reuse intermediate results? And, remember your solution must work even if A contains only negative values.

(b) (15 pts) In Sam's second game, guests must find the largest product of elements in some subarray $A[i \ldots j]$. Describe and analyze an algorithm that solves this problem in $\Theta(n^2)$ time.

Gandalf's hint: Do not assume that numbers in A are always integers, and try to avoid reinventing the wheel.

(c) (**10 pts extra credit**) Assume that A contains at least one positive value. Describe and analyze an algorithm that solves Sam's first game in $\Theta(n)$ time and $\Theta(1)$ space.

4. (15 pts) Implement an efficient simulated annealing algorithm for the *Ising model*, a classic model of magnetization from statistical physics.[3] Read the following carefully before beginning your implementation.

The Ising model is defined as a d-dimensional $n \times n$ square lattice S where each lattice element takes a value $s_i \in \{-1, +1\}$. For a particular assignment of values to lattice sites, the energy (score) is given by

$$E = -\sum_{i \neq j} J_{ij} s_i s_j$$

where s_i, s_j are lattice sites (vertices) and J_{ij} denotes the "coupling" strength between them (a weighted edge). If two sites are coupled, let $J_{ij} = 1$; otherwise, let $J_{ij} = 0$. Let $d = 2$, use the von Neumann neighborhood to define lattice site adjacencies, and assume periodic boundary conditions.[4] There are two global optimum of E, one at $\forall_i \, s_i = +1$ and the other at $\forall_i \, s_i = -1$, but many local optima.

For your implementation, choose a geometric cooling schedule with α as a free parameter and use the Metropolis `accept()` function (from Lecture 13), adapted to prefer *decreases* in energy rather than increases. For the `neighbor()` function, do the following: choose a uniformly random site $s_i \in S$ and propose to "flip" its state: if $s_i = -1$,

[3] The Ising model exhibits a rich variety of dynamical behavior, including fun things like phase transitions and criticality. Its generalization to k states is called a Potts model, but works largely just the same.

[4] For example, the corner site $S_{1,1}$ will have four neighbors $S_{1,2}$, $S_{2,1}$, $S_{n,1}$ and $S_{1,n}$. More generally, we "wrap" the left edge of the lattice around so that it neighbors the right edge and similarly for the top and bottom edges. (This can be done using the `mod` function.) Topologically, this makes the lattice into a torus.

then propose $s_i = +1$, and vice versa. Define convergence as n^2-in-a-row rejected proposals.

The inputs for your simulated annealer should be (i) the initial configuration of the lattice $S^{(0)}$, (ii) an initial temperature t_0 and (iii) the annealing parameter α. When it terminates, it should output the final state $S^{(\text{stop})}$ and its corresponding energy $E_{S^{(\text{stop})}}$.

Gandalf's hint: Implement the `accept()` function so that it takes $\Theta(d)$ time to compute the change in energy from flipping the proposed site, rather than $\Theta(n^d)$, which would not be efficient at all.

5. (25 pts total) Use your implementation from 4 to solve the following questions.

   (a) (10 pts) Describe the conditions on the structure of S that define every local optima of E. Justify your claim.

   (b) (15 pts) Using your implementation, let $n = 50$, $t_0 = 1000$, $\alpha = 1 - 10^{-4}$ and the initial state $S^{(0)}$ be one in which each site s_i is chosen uniformly at random from $\{-1, 1\}$. Run your annealer until it converges on a *local* optimum.

   Produce and submit (i) three snapshots of the system S, one of the initial configuration, one when 80% of the sites have the same state and one showing the final annealed state, and (ii) a paragraph describing your results and what they mean with respect to local/global optima of E.

   Since each site has a binary value, you can render these snapshots as black-and-white bitmap images. To get the second image, you'll need to implement some measurement code to track the number of sites in either state and to take a snapshot when the system hits the target configuration. For each figure, give the temperature of the annealer at that moment, the energy of the configuration and the fraction of sites with the majority state.

   Gandalf's hint: Make your bitmap images large enough to be examined visually, i.e., at least a few inches across.

6. (**10 pts extra credit**) Currency arbitrage is a form of financial trading that uses discrepancies in foreign currency exchange rates to transform one unit of some currency into more than one unit of the same currency. For instance, suppose 1 U.S. dollar bought 0.82 Euro, 1 Euro bought 129.7 Japanese Yen, 1 Japanese Yen bought 12 Turkish Lira and one Turkish Lira bought 0.0008 U.S. dollars. Then, by converting currencies, a trader could start with 1 U.S. dollar and buy $0.82 \times 129.7 \times 12 \times 0.0008 \approx$

1.02 U.S. dollars, thus turning a 2% profit. Of course, this is not how real currency markets work because each transaction must pay a commission to a middle-man for making the deal.

Suppose that we are given n currencies $c_1, c_2, \ldots, c_n$ and an $n \times n$ table R of exchange rates, such that one unit of currency c_i buys $R[i,j]$ units of currency c_j. A traditional arbitrage opportunity is thus a cycle in the induced graph such that the product of the edge weights is greater than unity. That is, a sequence of currencies $\langle c_{i_1}, c_{i_2}, \ldots, c_{i_k} \rangle$ such that $R[i_1,i_2] \times R[i_2,i_3] \times \cdots \times R[i_{k-1},i_k] \times R[i_k,i_1] > 1$. Each transaction, however, must pay a commission, which is typically some α fraction of the transaction value, e.g., $\alpha = 0.01$ for a 1% rate.

(a) (**5 pts extra credit**) Give an efficient algorithm to determine whether or not there exists such an arbitrage opportunity, given a commission rate α. Analyze the running time of your algorithm.

Gandalf's hint: It's possible to solve this problem in $O(n^3)$. Recall that Bellman-Ford can be used to detect negative-weight cycles in a graph.

(b) (**5 pts extra credit**) Explain what effect varying α has on the structure of the set of possible arbitrage opportunities your algorithm might identify.

7. (**25 pts total extra credit**) Suppose we want to maintain a dynamic set of values, subject to the following operations:

- `insert(x)`: Add x to the set (if it isn't already there)
- `printAndDeleteBetween(a,b)`: Print every element x in the range $a \leq x \leq b$, in increasing order, and delete those elements from the set.

For example, if our current set is $\{1, 5, 3, 4, 8\}$, then invoking `printAndDeleteBetween(4,6)` will print the numbers 4 and 5 and changes the set to $\{1, 3, 8\}$. For any underlying set, `printAndDeleteBetween(`$-\infty$`,`∞`)` prints its contents in increasing order and deletes everything.

(a) (**15 pts extra credit**) Describe and analyze a data structure that supports these operations, each with amortized cost $O(\log n)$, where n is the maximum number of elements in the set.

(b) (**5 pts extra credit**) What is the running time of your `insert` algorithm in the worst case?

(c) (**5 pts extra credit**) What is the running time of your `printAndDeleteBetween` algorithm in the worst case?