CS170 Problem Set 2 Out: Jan 19, 2001
Due: Jan 26, 2001, 4 pm

1. Write simple programs to evaluate S_n below that run much faster than using recursion or loops. You may use assignment statements, if-then-else, and the operations $+$, $-$, $\times$, $/$, sqrt(), the power function $\text{pow}(x, y) = x^y$, and mod. For example, a good answer for the Fibonacci sequence would be a short program that involved computing `rplus = (1+sqrt(5))/2`, `rminus = (1-sqrt(5))/2`, `aplus = rplus/sqrt(5)`, `aminus = -rminus/sqrt(5)`, `F = aplus×pow(rplus,n) + aminus×pow(rminus,n)`.

   (a) $S_n = 5S_{n-1} - 6S_{n-2}$, $S_2 = -1$, $S_4 = -49$ (Yes, S_2 and S_4. Your function should work for $n \geq 2$.)

   (b) $S_n = 6S_{n-1} - 9S_{n-2}$, $S_0 = 2$, $S_1 = 9$

   (c) $S_n = 2S_{n-1} - 2S_{n-2}$, $S_0 = 0$, $S_1 = 1$

   (d) $S_n = 64S_{n-3}$, $S_0 = 1$, $S_1 = -1$, $S_2 = 0$

2. The *square* of a directed graph $G = (V, E)$ is the graph $G^2 = (V, E^2)$ such that $(u, w) \in E^2$ if and only if for some $v \in V$ both $(u, v) \in E$ and $(v, w) \in E$. In other words, there is a path of length exactly 2 from u to w. Describe efficient algorithms for computing G^2 from G for both the adjacency-list and adjacency-matrix representations of G. Analyze the running times of your algorithms in terms of $n = |V|$ and $e = |E|$. Here "efficient" means something whose running time is a low-degree polynomial in n and e, the faster the better.

   Similarly, describe an efficient algorithm for computing $G^k = (V, E^k)$, where $(u, w) \in E^k$ if and only if there is a path of length exactly k in E.

3. The *incidence matrix* of a directed graph $G = (V, E)$ without self-loops (i.e. without edges (v, v)) is a $|V|$-by-$|E|$ matrix $B = (b_{ij})$ such that

$$b_{ij} = \begin{cases} -1 & \text{if edge } j \text{ leaves vertex } i \\ +1 & \text{if edge } j \text{ enters vertex } i \\ 0 & \text{otherwise} \end{cases}$$

   Describe what the entries of the matrix product BB^T represent, where B^T is the transpose of matrix B. BB^T is called the *Laplacian* of the graph B.

4. Give linear time algorithms, based on DFS, for the following problems:

   (a) Determining whether a given directed graph has a cycle.

   (b) Determining whether a given undirected graph has a cycle. Your algorithm in this case should run in time $O(n)$, where n is the number of nodes in the graph, and independent of the number of edges.

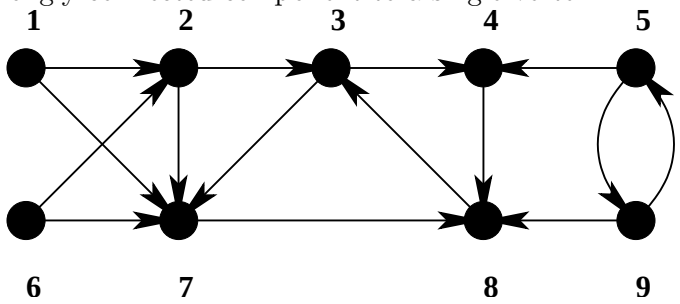
   (c) Determining whether a graph is bipartite.

5. A clique is a set of vertices such that each pair is connected by an edge. Let $G = (V, E)$ be a **connected** undirected graph, with $n = |V|$ vertices and $e = |E|$ edges. Give a tight upper bound on the size of the largest clique (in terms of n and e).

6. CLR Question 23.1-6.

7. You are given a tree T rooted at vertex r. Each vertex of the tree has an associated non-negative integer label $l(v)$. For any vertex v, we denote by $p(v)$ the parent of v in T. By convention $p(r) = r$. For $k > 1$ define the k-th ancestor $p^k(v)$ of v to be $p^{k-1}(p(v))$, and define $p^1(v) = p(v)$. Give a linear time algorithm to update the labels of all vertices in T according to the following rule: $l_{new}(v) = l(p^{l(v)}(v))$.

8. Perform a depth-first search on the directed graph below, starting at node 1. Classify each edge as a tree-edge, forward edge, cross edge, or a back-edge. Assign to each vertex its pre-order and post-order number. When considering edges out of a node, process them in increasing order of the labels of the nodes to which they lead. List the strongly connected components of the graph. Give the directed acyclic graph that results if we shrink each strongly connected component to a single vertex.



9. CLR Problem 23-3.

10. (Hard) In class we described a divide-and-conquer algorithm for computing the minimum and maximum of a list $A[1..n]$ of n numbers that did only $3n/2 - 2$ comparisons (at least for n a power of two). Prove that at least $\lceil 3n/2 \rceil - 2$ comparisons are *necessary* to compute the maximum and minimum. This means that our algorithm is optimal (at least for n a power of 2). Hint: Consider two lists m and M, each one initially containing all n numbers. List m is meant to contain all entries that might be the minimum, and list M is meant to contain all the entries that might be the maximum. After every comparison, you may be able to remove some elements from each list. For example, if the first comparison shows that $A[4] < A[10]$, then $A[4]$ can't be the max and $A[10]$ can't be the min, and so $A[4]$ can be removed from list M and $A[10]$ can be removed from list m. Other comparisons might result in removing just one item from either m or M, or perhaps none. To determine how the min and max, $2n - 2$ items must be removed from m and M, leaving two, the min and max. So try to count the number of comparisons needed *at a minimum* to remove $2n - 2$ items from m and M. For example, since each comparison can remove at most two items, $n - 1$ comparisons can remove at most $2n - 2$ items, leaving 2. This (too simple) argument shows that at least $n - 1$ comparisons are needed. A slightly fancier version of this argument will show that $\lceil 3n/2 \rceil - 2$ comparisons are needed.