MATH182 DISCUSSION 6 WORKSHEET

1. Application of DFS

Exercise 1. In this exercise we develop an algorithm to find the strongly connected components of a directed graph. Given a directed graph G, the *strongly connected components* of G are the equivalence classes of vertices of G under the relation of mutual reachability. That is, two vertices u and v are in the same connected component if and only if u is reachable from v and v is reachable from u.

We also define the *transpose* G^T of G to be the graph with the same edges in the opposite directed, $G^T = (V, E^T)$, where $E^T = \{(v,u) \ : \ (u,v) \in E\}$.

The idea of the algorithm is that if v is a descendant of u in a depth-first tree for G, then v is reachable from u, and if v is a descendant of u in a depth-first tree for G^T, then u is reachable from v in G. Unfortunately, not every vertex reachable from u is a descendant of u. However, the ones which are not descendants of u must either be in the same depth-first tree as u with a later finishing time, or have finishing time earlier than u's start time, in which case u is not reachable from them. In order to avoid the possibility of reachable vertices in the same tree, we will try vertices with later start times first.

(a) Use Theorems 8.3.1-3 of the notes to show that
   (i) If v is reachable from u in G, then either v is a descendant of u, or u and v have a common ancestor which is reachable from u, or $v.d < u.d$.
   (ii) If $v.d < u.d$, then either u is a descendant of v, or $v.f < u.d$.
   (iii) If $v.f < u.d$, and u and v do not have a common ancestor reachable from v, then u is not reachable from v.

(b) Show that if v is a descendant of u and u is reachable from v, then all vertices on the tree path from u to v are in the same strongly connected component.

(c) Suppose that we now do a depth-first search of G^T, where the (second) **for** loop in DFS loops over the vertices v of G^T in order of decreasing $v.f$. In this depth-first search, we will not update the discovery or finishing times, so $v.d$ and $v.f$ will only refer to discovery and finishing times in the DFS of G. Show that, if DFS-Visit(u) is called by the main algorithm DFS, then for every vertex v reachable from u in G^T (i.e. u is reachable from v in G), either
   (i) $v.f > u.f$ and hence $v.\mathit{color} = \text{BLACK}$, or
   (ii) u and v have a common ancestor reachable from v in G, or
   (iii) v is a descendant of u.

(d) In the scenario above, show that all ancestors of u are black, and any vertex reachable in G^T from a black vertex is also black. Deduce that in cases (i) and (ii), $v.\mathit{color} = \text{BLACK}$ and v cannot be reachable from u in G.

(e) Show that the vertex sets of the depth-first trees generated by the DFS on G^T are the strongly connected components of G.

2. Dynamic programming & greedy algorithms

Exercise 2. In HW5 Exercise 2, we find the longest palindromic subsequence of a given string. Here we find the longest palindromic sub*string*, i.e. we insist that the subsequence be contiguous.

(a) Describe briefly a brute-force algorithm to find the longest palindromic substring of a given string, and estimate its running time.

(b) For any $i \geq 0$, relate the longest palindromic substring of s which ends at the $i+1$st character to the longest palindromic substring of s which ends at the ith character. Prove carefully that your characterization is correct.

(c) Use the above to write an algorithm to find the longest palindromic substring of a given string, with running time linear in the length of the string.

Exercise 3. You are working on a project which requires the use of several different tools. Each time you need a tool, you must retrieve it from where it is stored. Tools are stored in two locations: a bench right next to you, which can store k tools at once, and shelves which are some distance away. Knowing in advance the sequence of tools that you will need, you wish to plot which tools to leave on the bench at any given time and which to put away on the shelves, in such a way as to minimize the amount of walking you need to do. (*Note: This problem is actually about caching.*)

(a) Note that it is always preferable to retrieve the next tool from the shelves which you will need after returning a tool. That is, you should always have a tool in hand when walking to your work area from the shelves, so any trip to return a tool is also a trip to retrieve a tool. Minimizing walking is therefore equivalent to minimizing the number of times that a tool must be retrieved from the shelves.

(b) We use a furthest-in-future strategy: whenever we need to retrieve a new tool and the bench is already full, we return the tool on the bench whose next use is furthest in the future. Show that there is always an optimal solution which makes this choice. (*Hint:* Given an optimal solution which returns a different tool, that tool will need to be retrieved before the furthest-in-future choice is used. Change this to storing and retrieving the FIF tool.)

(c) Show that this problem exhibits optimal substructure.

(d) Write pseudocode to solve this problem. What is the running time? The relevant variables are the bench size k, the number of distinct tools m, and the number n of times you need to change which tool you're using.