[Overall Information]

We had quiz#1 in today’s class.

In addition to that, we studied Depth-First-Search (DFS) algorithm. Its correctness was proved and running time was analyzed.

[During the Lecture]

1. At the beginning of today’s class, we had the first quiz of CSCI303 at Spring 2006. It is a 20 minutes quiz with four questions. The grading result will be posted on course web page a few days later.
After reviewing what we talked about the comparison of Breadth-First Search (BFS) and DFS in last lecture, the pseudocode of DFS algorithm using recursion was written onto the whiteboard. To summarize the algorithm in one word, the function DFS-explore(v) returns only when it has finished exploring all the offspring of v. In other words, BFS tries to figure out which nodes belong to the same “level” while for DFS, it figures out which nodes belong to the same “family”.
To illustrate the idea of DFS, a specific example was given. Starting from the start node “S”, we followed the algorithm step by step. Green points indicate the algorithm reaches certain node while the red points indicate the corresponding node is “explored”. Red arrows are backtracking pointers. In the case when there are multiple choices available, alphabetic order is used, but other implementations of DFS might choose different orders. Facing the final result, we can see that the path found is actually not the shortest one.

The correctness claim of DFS is that all the reachable nodes are explored and with the help of backtracking pointers, we are able to find a path back to the start node “S”.

```
DFS (G):
    Mark all vertices unexplored, unvisited
    Call DFS-explore (S)

DFS-explore (vertex v):
    Mark v as visited
    for each edge e = (v, u) out of v, do
        if u is unvisited and unexplored then
            set pred[u] = v; okp: pred[u] = v;
            DFS-explore (u);
    Mark v as explored.
```
Let $u$ be a reachable node. $l[u]$ represents the time when the algorithm first reaches $u$ and $r[u]$ represents the time when $u$ is marked to be explored. $[l[u], r[u]]$ represents the time interval between $l[u]$ and $r[u]$. Here, another interesting property we noticed is that for two reachable nodes: $u$ and $v$. The relationship between $[l[u], r[u]]$ and $[l[v], r[v]]$ is either “contained” or “disjoint”. One interval is contained by the other if and only if one node is the ancestor of the other in the DFS tree.

4. Observing the DFS algorithm we can see that there is another way to
think of DFS: it is just an implementation of BFS with a stack (LIFO: the last added element is removed first) instead of the FIFO queue.

Next, we proved the correctness of the above claim using induction.

For base case $i=0$, it is explored and it does not need $\text{pred}()$.

For the step from $i$ to $i+1$, let $v$ be any node with distance $(i+1)$, so there is a $(i+1)$ hops path from $s$ to $v$. Suppose that $u$ is the last node before $v$ on such a path. According to the induction assumption, $u$ was visited before. Consequently, $v$ is either explored at that time or it had already been explored at that time.

This proves that all reachable nodes are explored.

5. To prove that a path from the start node to the target node can be reconstructed from the $\text{pred}[]$ pointers, first we renumber the nodes into
v₁, v₂, …, vₖ according to the visiting order.

Using induction, the base case is obvious. For the i to i+1 case, when vₖ is visited, its pred pointer is pointing to vⱼ for some j less than or equal to i. According to the induction hypothesis, following the pred[] pointers from vⱼ gets us to s, and adding the pred[] pointer as the last hop shows that the pred[] pointers also give a path from vₖ to s, which completes the proof.

To analyze the running time of DFS, we can see most steps take θ(n) time. The new situation we encountered here is recursion. Since each time the function DFS-explore(v) is called, it checks all the edges connect v, and all the recursive calls can be attributed to those nodes, the
total running time can be expressed as $T(n) = \Theta(n) + \sum \Theta(\text{degree}(v)) = \Theta(n)$ + $\Theta(\sum \text{degree}(v)) = \Theta(n+m)$, where $n$ is the number of nodes and $m$ is the number of edges.