Class Note #03
Date: 01/18/2006

[Overall Information]

In this class, after several new announcements and a brief review of Big-O notation we talked about last time, the class moved to graph related topics.

Breadth First Algorithm was given and we are able to see how it works step by step. Its running time was also analyzed in today’s class.

[During the Lecture]

1. After the class began, several new announcements were made:
(1) The TA’s office hours are Monday from 2:00pm to 4:00 pm and Wednesday from 1:00 pm to 3:00 pm. The office location is SAL 311. Feel free to contact me through email and set up a time if you have questions and can not come to my office hours.
(2) The homework NO.1 was assigned. The first question asks you to proof the correctness of the Selection Sort Algorithm. The other two questions come from textbook. They are 1-1 and 3-2 respectively. The homework grading will not count for your final grades but you can still turn it in if you want some feedback.
(3) Quiz#1 will take place next Wednesday (01/25/2006). The questions in quiz#1 will probably be similar to homework questions.
2. To review the Big-O notation we talked about last lecture, first apply it to Insertion Sort algorithm one more time to see how convenient and simple it is.

Since in Big-O notation, constant can be safely ignored, for the ith iteration, we can just express the number of comparison as O(i). As for other operations, they take constant time which can be expressed as O(1). Because O(1) can be ignored if compared with O(i), finally for the ith iteration, the running time in Big-O notation is O(i). Consequently, the running time for sorting the whole array is 
\[ \sum_{i=2}^{n} O(i) = O \sum_{i=2}^{n} i = O \left( \frac{n(n-1)}{2} - 1 \right) = O(n^2). \]

An even simpler analysis is obtained if we say that the while loop takes at most O(n) steps. This is certainly true in each iteration because each element is swapped at most n times. Then, the running time is: 
\[ \sum_{i=2}^{n} O(n) = O(n^2). \]
Hence, sometimes big-O notation lets us use even simpler and cruder bounds, but still arrive at the right result.
3. People usually use equations like $f = O(g)$ or even $f(n) = O(g(n))$ to express the Big-O notation. Such expressions, from some point of view, are quite misleading. For example, $O(g(n)) = f(n)$ is totally meaningless. The mathematically correct express should be $f(n) \in O(g(n))$. We should keep this in mind although in this course, it is allowed to use the commonly-accepted and simplified notation like $f(n) = O(g(n))$. 
4. The class moved to graph related topics (from chapter 22 in the textbook). Before starting the topic, several important definitions were given. They are: graph, path, cycle and tree. You should already be familiar with those concepts.

What are the real world application examples of graph? The applications we mentioned in the class include: map system (cities and roads), data structure of data searching, social networks, family trees, computer networks, WWW web and food chains in biology.
After that, a specific graph-related problem of finding path with fewest numbers of hops was written onto the white board.

Please notice that we don't always want the smallest number of hops. Rather, when the edges have lengths (such as for roads), we want the smallest total length. We saw that this can be reduced to the smallest number of hops by dividing each edge of length L into L edges of length 1 each (inserting new nodes). This reduction is correct, but can be very inefficient, because it generates many new nodes. In general, reductions often let us find simple solutions, but aren't always efficient.

Next, we discussed how to associate this shorted path problem to those applications we mentioned above.
6. Next, the pseudocode of Breadth-first algorithm was written in full onto the white board. We can see that the algorithm focuses on finding the “reachable” list rather than finding the specific path between two nodes.

7. To see how the algorithm fulfills the task, a specific graph example was given. We are able to see what the algorithm do step by step while variables like the list, the corresponding distance and explored markers were updated at the same time. After running the algorithm, we can tell not only the distance but also how to reach those “reachable” nodes with
the help of backtracking pointers.

8. To analyze the running time of the Breadth-first algorithm, first notice the beginning four steps take constant $O(n)$ and $O(1)$ time. Then each loop step take $O(n)$ while the number of iteration we need to do is $O(n)$. In conclusion, the running time is $O(n)+O(1)+O(n)(O(1)+O(n)+O(1)) = O(n^2)$.

A more precise analysis states that the running time is $O(n+m)$, where $n$ is the number of nodes and $m$ is the number of edges. The reason is as the following:

In the iteration for node $v$, the total work done is $O(\text{degree}(v))$. So the total work is $O(n + \text{sum of all nodes degree}) = O(n + m)$, because the sum of all node degrees is $2m$ (m is the number of edges).

We will prove the correctness of the Breadth-first algorithm in next lecture.
Algorithm BFS (Breadth-First Search)

1. Mark all nodes as unexplored, not enqueued, set \( \text{deg}(v) = 0 \) for all \( v \).
2. Set \( \text{deg}(s) = 0 \), and initialize the queue to \( \{s\} \), mark \( s \) enqueued.
3. While queue is not empty do:
   a. Let \( v \) be the head of the queue (remove it).
   b. For each edge \( e = (u,v) \) out of \( v \), do:
      i. If \( u \) is unexplored and not enqueued:
         1. Set \( \text{deg}(u) = \text{deg}(v) + 1 \).
         2. \( \text{pred}[\text{deg}][v] = u \), enqueue \( u \) and mark \( u \) enqueued.
4. \( |V| \) - \( O(1) \) + \( O(n) \) - \( O(1) + O(n) \times (O(1) + O(n) + O(1)) = O(n^2) \)