Sample Midterm Exam

Open Book and Notes

(45) 1. Suppose that you have a bipartite graph \( G = (L, R, E) \) and a current matching \( M \) which is not maximum. Let \( n_L \) be the number of vertices in \( L \), \( n_R \) the number of nodes in \( R \), and \( m \) the number of edges in \( E \). Let \( G_L \) be the layered graph in \( G \) with respect to \( M \), where the layer 0 of \( G_L \) has the unmatched vertices in \( L \), the last layer has only unmatched vertices in \( R \), and in general layer \( i \) has those vertices which are distance \( i \) from an unmatched vertex in \( L \) on a shortest augmenting path, and only the last layer \( k \) has unmatched vertices in \( R \) (thus \( k \) is the length of a shortest A-path with respect to \( M \)).

For each of the following statements say whether it is true or false and briefly justify your answer:

b) All augmenting paths can be found in \( G_L \) in \( O(n_L + n_R) \) time.
False. The layered graph may have many more than this many edges all of which must be examined.

c) An adjacency list is a good way to represent the edges leaving a layer in \( G_L \) which contains nodes from \( L \).
True. The number of edges from different nodes may vary greatly, so don't want a fixed size structure.

d) An adjacency list is a good way to represent the edges leaving a layer in \( G_L \) which contains nodes from \( R \).
False. Each node in \( R \) has at most one outgoing edge (its matched edge, or its edge to \( t \)).

e) If \( G_L \) has at most 3 vertices in each layer, then we will find at most 3 A-paths in \( G_L \).
True. Each augmenting path kills off a node as useful, so after three such augs they are all dead.

g) There are at most \( 2n_L \) layers in \( G_L \).
False. Consider a graph with 4 nodes (1 each in left and right plus \( s/t \)) this can have four layers.

(20) 2. After students graduate from Medical school they are assigned to hospitals. Each graduating student can choose those hospitals he/she would like to be assigned to, and each hospital can choose which graduating students they are willing to hire.

Your goal is to assign students to hospitals so that each student is assigned to a hospital which they want, and which wants to hire them (or is such an assignment is impossible, assign as many students as possible). You may assume you have all the preference data for both students and hospitals. Assume there are \( u \) students and \( h \) Hospitals.

a) Describe how to find an assignment of students to hospitals when each hospital wants to hire ONE student.

Create a bipartite graph \( B \) with \( u \) nodes on the left, for students, and \( h \) nodes on the right for hospitals. Connect nodes \( i \) and \( j \) exactly when student \( i \) is interested in hospital \( j \) and the hospital wants student \( i \).
Now find a maximum matching in $B$.

b) Describe how to find an assignment of students to hospitals when the $ith$ hospital wants to hire up to $w_i$ students.

Now use network flow. Create a network which has the graph $B$ in the middle, a source which has an arc to each left (student) node of capacity one, and a sink which gets an arc from each right (hospital) node $i$ of capacity $w_i$.

c) What is the running time of your solution technique in case a) and in case b)?

If one plugs in the best bounds for matching/net-flow we get $O(m \text{ root}(n))$ for a) and $O(n^3)$ for b) where $m$ is the number of edges in $B$ and $n = u + h$ the number of nodes. However, using the ideas from problem 1 you can improve the run times to $O(m \text{ root}(h))$ and $O(n^3h)$ respectively (though this is more than I would have expected on the exam).

(20) 3. In the select program for assignment 1 the main bottleneck of the code was the loop which partitioned the array into those less than $t$ and those greater than $t$

```c
m = L;
for (i=L+1; i <= u; i++) /* this loop rearranges the elements in
the range L+1..u so that all those < t are at the front */
  if (x[i] < t)
    swap(++m,i,x);
```

This approach works in-place (it rearranges the data within the existing array). An alternative is to use auxiliary array(s). This has the advantages of not changing the input array, and simplifies the partition problem in some ways:

If we use a new array $y$ which is of size $u-L+1$ we can use the following code to put the small elements at the front of $y$ and the large ones at the end (with $t$ in the middle):

```c
small = 0; /* next element smaller than $t$ goes here */
big = u-L; /* next element larger than $t$ goes here */
for (i=L+1; i <= u; i++)
  if (x[i] < t)
    y[small++] = x[i];
  else y[big--] = x[i];
y[small] = t;
```

What are the potential advantages and disadvantages of this change in approach? Would you expect it to speed up or slow down the overall run time? (justify your answer).

Advantages: simpler code (fewer bugs),
Disadvantages: More space, slower if have to copy $y$ back to $x$ (though about the same if alternate copying $x$ to $y$ and $y$ to $x$), may have worse cache performance due to more space.
(25) 1. Suppose you have a directed graph $G$. For two specified vertices $a$ and $b$ you want to find the smallest set of vertices whose removal disconnects all directed paths between $a$ and $b$. Describe how to do this and give the running time of our solution when $G$ has $n$ vertices and $m$ edges.

Create a new graph $G^*$ where each node $i$ is split into two nodes $i'$ and $i''$. If $(i,j)$ was an arc in $G$, then $(i'',j')$ will be an arc in $G^*$. In addition we add an arc $(i',i'')$ of capacity $1$ for each node $i$ in $G$. All other arcs have capacity infinity, and we treat $a''$ as the source and $b'$ as the sink. Now find a min-cut in $G^*$. Clearly this min-cut will not use any have any infinite arcs across it if possible. Similarly, note that if removing nodes $i, j$, and $k$ blocks all directed paths from $a$ to $b$, then we can create a cut of value $3$ by putting $i'j'$ and $k'$ on the $s$ ($=a''$) side of the cut and $i''j''$, $k''$ are on the $T$ ($=b'$) side of the cut. The $S$ side of the cut also gets nodes $v'$ and $v''$ for every node $v$ which $a$ can reach in the graph $G$ with $i,j$ and $k$ removed. The rest of the nodes go on the $T$ side.

Thus for set of $r$ nodes whose removal separates $a$ from $b$, we can find a cut of values $r$, and for any cut, we can remove the nodes whose arcs cross the cut and separate $a$ from $b$. Thus the min cut in $G^*$ will be the minimum separating set of vertices.

The run time is the time for one max flow in $G^*$, or $O(n^3)$ (you can actually exploit the properties of $G^*$ to get a better bound.)

(25) 3. Consider the following variant of the select program for assignment 1. In each loop iteration to process the elements in $A[L..u]$, instead of just picking $t$ (the split element) at random we instead do the following:

$$
\text{let } s = \text{ the square root of } u-L; \\
\text{Choose a random sample } S \text{ of size } s \text{ from the elements in } A[L..u]; \\
(*) \text{ let } t = \text{ the median element in } S;
$$

Now proceed as in the original algorithm using $t$ to partition into small and large, and so on.

In each of the following questions assume we start with a large list (say $n > 1,000,000$).

a) In step (*) above we could find the median either by sorting the random sample or by a recursive call to our select algorithm.

In you compared the running time of two implementations of the above select algorithm, one of which did step (*) by a good sorting algorithm, and one using the recursive call would you expect the sorting version to be: much faster, much slower, or about the same as using the select algorithm? Justify your answer.

Since the sample is much smaller than the list (size at most 1,000) the time to process it will be negligible compared to the time to partition. Thus both implementations will run in about the same time (with the recursive one presumably slightly faster).

b) Suppose you solve (*) using a good sorting algorithm. Would you expect this version of select to be faster or slower than the original select algorithm you were given for PS1? Justify your answer.

Faster. As discussed in a) the time to find the median of the sample is small compared to partition, thus preventing unbalanced splits should more than makeup for the time to find the sample median.