CSE 5500 Algorithms. Fall 2018

Solutions to Model Exam 2.

1. Evaluate the given polynomial at each integer in the range $[1, cn]$. If the polynomial evaluates to zero at any v, then v is a root. We can evaluate the polynomial at cn points in $O(n \log^2 n)$ time as has been mentioned in class.

2. We first get an upper bound on d (within a factor 2) using the doubling trick. Followed by this, we use binary search to get the value of d. Once we know d, we interpolate the first d pairs in X to get and output the right polynomial.

   $k := 1$;
   repeat
   1. Interpolate the first k pairs of X to get a polynomial $f(x)$;
   2. Check if $f(r_i) = a_i$ for each $i, 1 \le i \le n$;
   3. If yes, **then quit else** $k := 2k$;
   forever

   The k we get from the above code is an upper bound on d such that $k \le 2d$. Now we perform a binary search in the range $\left[\frac{k}{2}, k\right]$ to get the actual value of d in a similar manner. Once we know the value of d we do an interpolation on the first d pairs of X to get the correct polynomial $f(x)$.

   Note that one execution of step 2 above can be done in $O(n \log^2 k) = O(n \log^2 d)$ time. In the entire algorithm we perform step 2 a total of $O(\log d)$ times. Thus the total time for step 2 is $O(n \log^3 d)$. We also perform step 1 a total of $O(\log d)$ times and each execution of step 1 takes $O(k \log^3 k) = O(d \log^3 d)$ time. Thus the total time for step 1 is $O(d \log^4 d)$. Step 3 takes a total of $O(\log d)$ time.

   Thus, the run time of the entire algorithm is $O(n \log^3 d + d \log^4 d)$. If n is much larger than d, then this run time is $O(n \log^3 d)$.

3. Recall that Prim's algorithm starts with a tree that only has the minimum edge (u, v). It then computes the *near* value for each node (other than u and v). This takes $O(|V|)$ time. After this, each node $w \in V - \{u, v\}$ is inserted into a priority queue. If we use a Fibonacci heap, this will take $O(|V| \log |V|)$ time. Followed by this we have a **for** loop that is done $|V| - 2$ times. In each run of the **for** loop, we identify the minimum element from the Fibonacci heap. This will cost a total of $O(|V| \log |V|)$ time in the entire **for** loop. If w is the node with the minimum key, for each neighbor x of w, we check if x changes its *near* value and if so we change its key. In the worst case we have to change keys $O(|E|)$ times. If we use a Fibonacci heap, this will cost a total of $O(|E|)$ time. As a result, the total run time of the algorithm will be $O(|V| \log |V| + |E|)$.

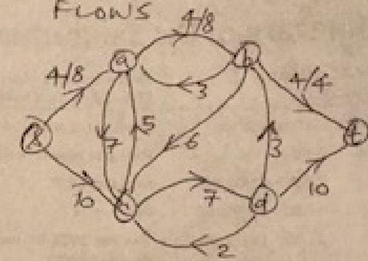
4. Use BFT to find the connected components of the given graph. Fill the transitive closure matrix A^* as follows: $A^*(i, j) = 1$ iff either $i = j$ or $i \neq j$ and i and j are in the same connected component.
   Total complexity = complexity of BFT + complexity of filling the A^* matrix = $O(|V| + |E|) + O(|V|^2) = O(|V|^2)$.

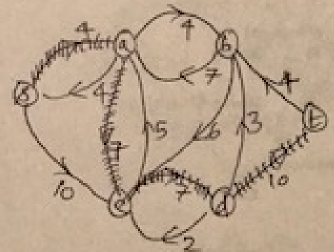
5. Define a function $shuffle$ as below:
   $shuffle(i, j) = 1$ if $z_1, \ldots, z_{i+j}$ is a shuffle of $x_1, \ldots, x_i$ and $y_1, \ldots, y_j$.
   $shuffle(i, j)$ can be calculated from $shuffle(i-1, j)$ and $shuffle(i, j-1)$ as below:
   $shuffle(i, j) = 1$ if $shuffle(i-1, j) = 1$ and $z_{i+j} = x_i$ OR
   $shuffle(i, j-1) = 1$ and $z_{i+j} = y_j$.
   $shuffle(m, n)$ tells whether z is a shuffle of x and y.
   We have to compute $shuffle(i, j)$ for $1 \le i \le m$ and $1 \le j \le n$ and hence the total time taken is $O(mn)$.

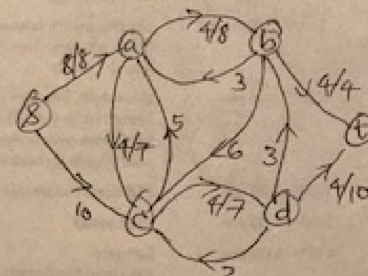
6. Using the Ford-Fulkerson algorithm we realize that the max flow is 11.

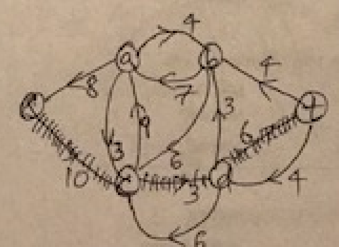
RESIDUAL GRAPHS

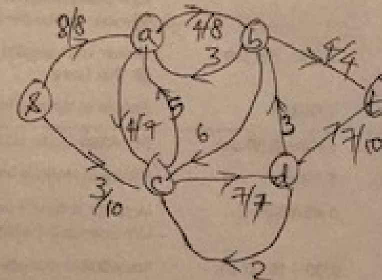
FLOWS

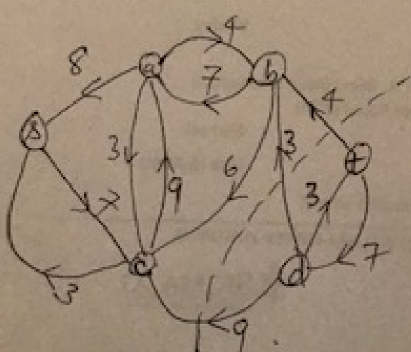












MAXFLOW = 11.

NO AUGMENTING PATHS