# CS570

Analysis of Algorithms

Spring 2008

Exam II

Name: ______________________

Student ID: __________________

| | Maximum | Received |
|---|---|---|
| Problem 1 | 20 | |
| Problem 2 | 15 | |
| Problem 3 | 15 | |
| Problem 4 | 15 | |
| Problem 5 | 20 | |
| Problem 6 | 15 | |
| Total | 100 | |

Note: The exam is closed book closed notes.

1) 20 pts

Mark the following statements as **TRUE**, **FALSE**. No need to provide any justification.

**[ TRUE ]**

If all capacities in a network flow are rational numbers, then the maximum flow will be a rational number, if exist.

**[TRUE]**

The Ford-Fulkerson algorithm is based on the greedy approach.

**[ FALSE ]**

The main difference between divide and conquer and dynamic programming is that divide and conquer solves problems in a top-down manner whereas dynamic-programming does this bottom-up.

**[ FALSE ]**

The Ford-Fulkerson algorithm has a polynomial time complexity with respect to the input size.

**[ TRUE ]**

Given the Recurrence, $T(n) = T(n/2) + \theta(1)$, the running time would be $O(\log(n))$

**[ FALSE ]**

If all edge capacities of a flow network are increased by k, then the maximum flow will be increased by at least k.

**[ TRUE ]**

A divide and conquer algorithm acting on an input size of n can have a lower bound less than $\Omega(n \log n)$ .

**[ TRUE ]**

One can actually prove the correctness of the Master Theorem.

**[ TRUE ]**

In the Ford Fulkerson algorithm, choice of augmenting paths can affect the number of iterations.

**[ FALSE ]**

In the Ford Fulkerson algorithm, choice of augmenting paths can affect the min cut.

2) 15 pts

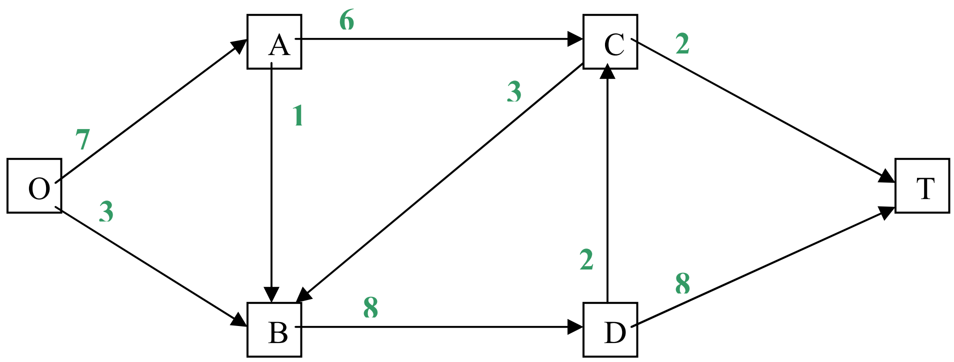
Present a divide-and-conquer algorithm that determines the minimum difference between any two elements of a sorted array of real numbers.

Key feature: The min difference can always been achieved between a pair of neighbors in the array, as the array is sorted.

```
int Min_Diff(first, last)
{
        if (last >= first)
                return inf;
        else
                return min(Min_Diff(first, (first + last)/2), Min_Diff((first + last)/2+1,
last), abs(number[(first + last)/2+1] - number[(first + last)/2]));
}
```

The complexity is liner to the array size.

3) 15 pts
You are given the following directed network with source O and sink T.

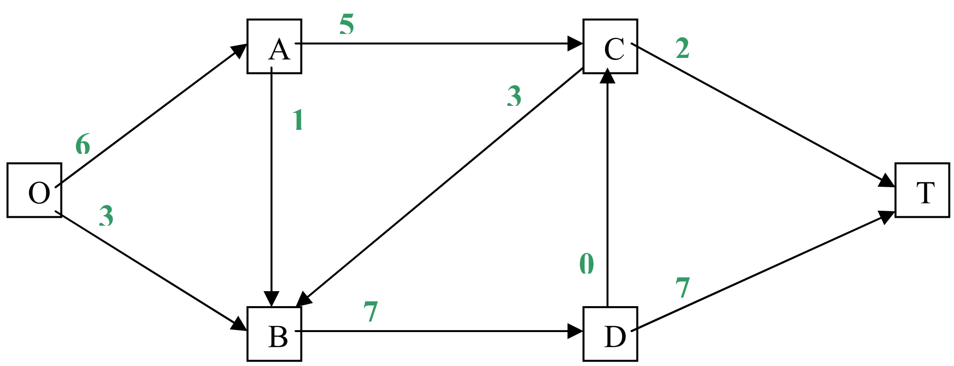



a) Find a maximum flow from O to T in the network.

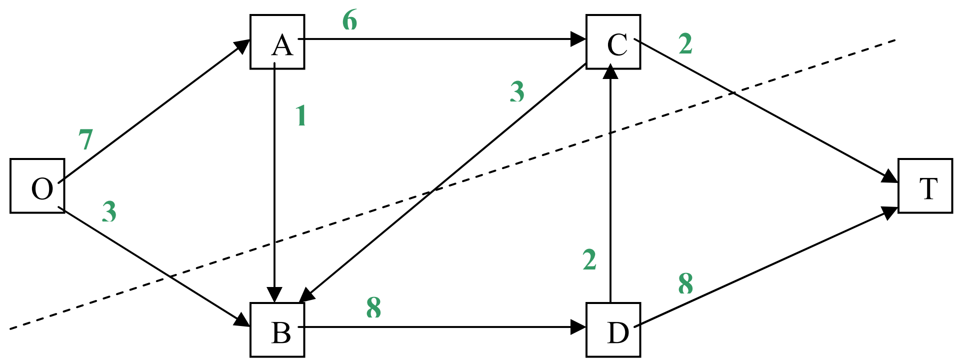
Augmenting paths and flow pushing amount:

| | |
|---|---|
| OACT | 2 |
| OBDT | 3 |
| OABDT | 1 |
| OACBDT | 3 |

And the maximum flow here is with weight 9:




b) Find a minimum cut. What is its capacity?




Capacity of this min cut is 9.

4) 15 pts
Solve the following recurrences

a) T (n) = 2T (n/2)+ n log n

According to the master theorem, $T(n) = \Theta(n\log^2 n)$.

Or we can solve it like this:

$$T(n) = 2T(\frac{n}{2}) + n\log n = 2(2T(\frac{n}{4}) + \frac{n}{2}\log n - \frac{n}{2}\log 2) + n\log n$$

$$= 4T(\frac{n}{4}) + 2n\log n - n\log 2 = ... = 2^k T(\frac{n}{2^k}) + kn\log n - \frac{k(k-1)}{2} n\log 2$$

$$= ... =_{(k=\log n)} nT(1) + \Theta(n\log^2 n) = \Theta(n\log^2 n)$$

b) T (n) = 2T (n/2) + log n

Similar to a), the result is $T(n) = \Theta(n)$.

c) T(n) = 2T(n-1) - T(n-2) for n ≥ 2; T(0) = 3; T(1) = 3

It is very easy to find out that for the initial values T(0)=T(1), we always have T(i)=T(0), i >0. Thus T(n) = 3.

5) 20 pts

You are given a flow network with integer capacity edges. It consists of a directed graph G = (V, E), a source s and a destination t, both belong to V. You are also given a parameter k. The goal is to delete k edges so as to reduce the maximum flow in G as much as possible. Give a efficient algorithm to find the edges to be deleted. Prove the correctness of your algorithm and show the running time.

We here introduce a straightforward algorithm (assuming k<=|E|, otherwise just return failure):

```
Delete_k_edges()
{
    E' = E;
    for i=1 to k
    {
            curr_Max_Flow = inf;
            for j in E'
                    if Max_Flow(V, E'-j) < curr_Max_Flow
                    {
                            curr_Max_Flow = Max_Flow(V, E'-j);
                            index[i] = j;
                    }
            E' = E' – index[i];
    }
}
```

Then the final E' is a required edge set, and indices of all k deleted edges are stored in the array index[].

Running time is $O(k|E| \cdot T(\max\_flow))$, depending on the max_flow algorithm used here, the time complexity varies: if Edmonds_Karp is used here the time would be $O(k|V||E|^3)$; if Dinic or other more advanced algorithm is used here the time complexity can be reduced.

Proof hint:
By induction.
k = 1, the algorithm is correct.
Assume k = i the algorithm is correct. Then we prove for k = i+1, it is also correct. Here, it is better to divide this i+1 into the first step and the folloing i steps, not vice versa.

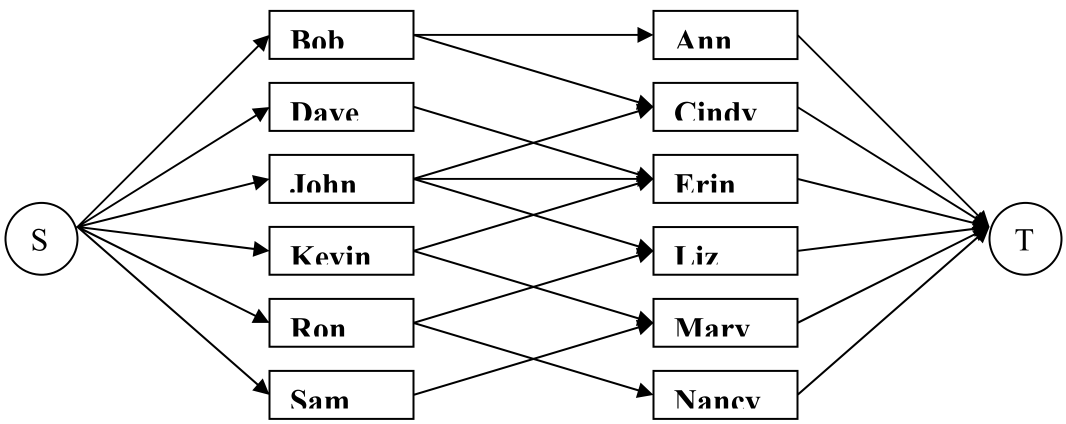
6) 15 pts

Six men and six women are at a dance. The goal of the matchmaker is to match each woman with a man in a way that maximizes the number of people who are matched with compatible mates. The table below describes the compatibility of the dancers.

| | **Ann** | **Cindy** | **Erin** | **Liz** | **Mary** | **Nancy** |
|---|---|---|---|---|---|---|
| **Bob** | C | C | - | - | - | - |
| **Dave** | - | - | C | - | - | - |
| **John** | - | C | C | C | - | - |
| **Kevin** | - | - | C | - | C | - |
| **Ron** | - | - | - | C | - | C |
| **Sam** | - | - | - | - | C | - |

*Note*: C indicates compatibility.

a) Determine the maximum number of compatible pairs by reducing the problem to a max flow problem.




All edges are with capacity 1.
Run some maximum flow algorithm like Edmonds-Karp, it would guarantee to return a 0-1 solution within polynomial time, with represents the required match.

b) Find a minimum cut for the network of part (a).

A={S, Dave, Kevin, Sam, Erin, Mary} and A' = V-A constitute a minimum cut, with capacity 5.

c) Give the list of pairs in the maximum pairs set.

Maximum 5 pairs. One solution:
Bob-Ann, Dave-Erin, John-Cindy, Ron-Nancy, Sam-Mary.