CS570

Analysis of Algorithms

Summer 2008

Exam II

Name: ________________________

Student ID: ____________________

____4:00 - 5:40 Section ____6:00 – 7:40 Section

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

2 hr exam

Close book and notes

1) 15 pts

a) Suppose that we have a divide-and-conquer algorithm for a solving computational problem that on an input of size n, divides the problem into two independent subproblems of input size 2n/5 each, solves the two subproblems recursively, and combines the solutions to the subproblems. Suppose that the time for dividing and combining is O(n). What's the running time of this algorithm? Answer the question by giving a recurrence relation for the running time T(n) of the algorithm on inputs of size n, and giving a solution to the recurrence relation.

Solution:

The recurrence relation of T(n):
$T(n) = 2\,T(2n/5) + O(n)$

To solve the recurrence relation, using the substitution method:
guess that $T(n) \leq cn$
$\Rightarrow T(n) \leq 2 * 2c/5n + an \leq cn$ as long as $c \geq 5a$
Thus, $T(n) \leq cn$. $\Rightarrow T(n) = O(n)$

b) Characterize each of the following recurrence equations using the master method. You may assume that there exist constants $c > 0$ and $d \geq 1$ such that for all $n < d$, $T(n) = c$.

a. $T(n) = 2T(n/2) + \log n$
b. $T(n) = 16T(n/2) + (n \log n)^4$
c. $T(n) = 9T(n/3) + n^3 \log n$

Solution:

a. Since there is $\varepsilon > 0$ such that $\log n = O(n^{\log_2 2-\varepsilon}) = O(n^{1-\varepsilon})$,
$\Rightarrow$ case 1 of master method, $T(n) = \Theta\left(n^{\log_2 2}\right) = \Theta(n)$

b. $(n \log n)^4 = n^4 \log^4 n = \Theta\left(n^{\log_2 16} \log^4 n\right)$,
$\Rightarrow$ case 2 of master method, $T(n) = \Theta\left(n^{\log_2 16} \log^{(4+1)} n\right) = \Theta(n^4 \log^5 n)$

c. $n^3 \log n = \Omega(n^{\log_3 9 + 1})$ and $9 \times \left(\frac{n}{3}\right)^3 \log\frac{n}{3} = \frac{n^3}{3}\log\frac{n}{3} \leq \frac{1}{3} n^3 \log n$,
$\Rightarrow$ case 3 of master method, $T(n) = \Theta(n^3 \log n)$

2) 15 pts

You are given a sorted array A[1..n] of n distinct integers. Provide an algorithm that finds an index i such that A[i] = i, if such exists. Analyze the running time of your algorithm

Solution: (This one is exactly the same as 5) of sample exam 1 of exam I)

```
Function(A,n)
 {
i=floor(n/2)
if A[i]==i
 return TRUE
if (n==1)&&(A[i]!=i)
 return FALSE
if A[i]<i
  return Function(A[i+1:n], n-i)
if A[i>i]
  return Function(A[1:i], i)
}
```

Proof:

The algorithm is based on Divide and Conquer. Every time we break the array into two halves. If the middle element i satisfy A[i]<j, we can see that for all j<i, A[j]<j. This is because A is a sorted array of DISTINCT integers. To see this we note that A[j+1]-A[j]>=1 for all j. Thus in the next round of search we only need to focus on A[i+1:n]

Likewise, if A[i]>i we only need to search A[1:i] in the next round.

For complexity T(n)=T(n/2)+O(1)

Thus T(n)=O(log n)

3) 15 pts

Consider a sequence of n distinct integers. Design and analyze a dynamic programming algorithm to find the length of a longest increasing subsequence. For example, consider the sequence:

45 23 9 3 99 108 76 12 77 16 18 4

A longest increasing subsequence is 3 12 16 18, having length 4.

<u>Solution:</u>

Let X be the sequence of n distinct integers.

Denote by X(i) the ith integer in X, and by D_i the length of the longest increasing subsequence of X that ends with X(i).

The recurrence that relates D_i to D_i's with $j < i$ is as follows:

$$D_i = \max_{j<i, X(j)<X(i)} (D_j + 1)$$

The algorithm is as follows:

for i =1…n

Compute D_i

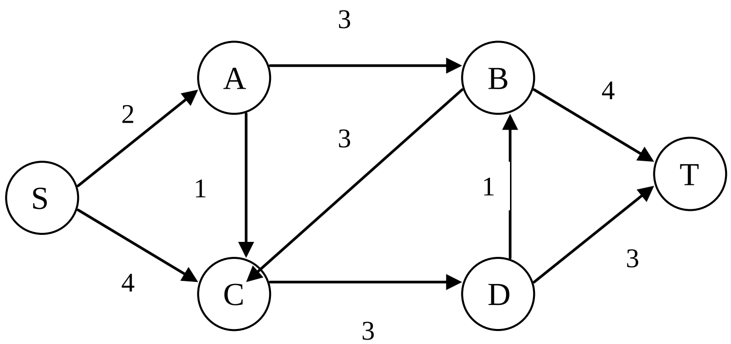
end for

return the largest number among D_1 to D_n.

When computing each D_i, the recurrence finds the largest D_j such that $j < i, X(j) < X(i)$. Thus, each D_i is maximized. The length of the longest increasing subsequence is obviously among D_1 to D_n.

Since computing each D_i costs O(n) and the loop runs for n times, the complexity of the algorithm is $O(n^2)$.

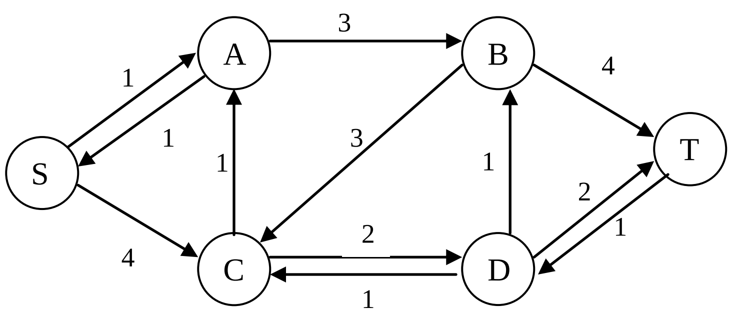
4) 20 pts
In the flow network illustrated below, each directed edge is labeled with its capacity. We are using the Ford-Fulkerson algorithm to find the maximum flow. The first augmenting path is S-A-C-D-T, and the second augmenting path is S-A-B-C-D-T.



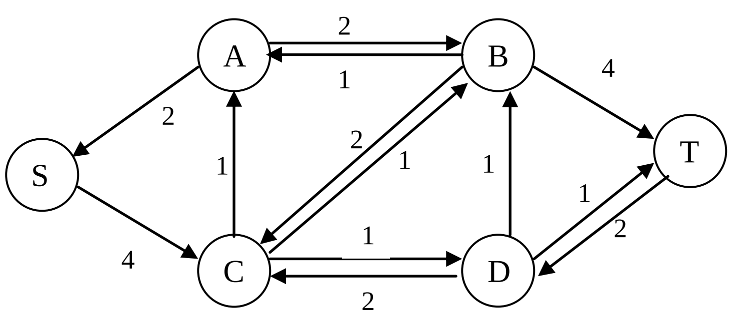
a) 10 pts
Draw the residual network after we have updated the flow using these two augmenting paths(in the order given)

Solution:

Residual network after S-A-C-D-T:



Residual network after S-A-B-C-D-T:



b) 6pts
List all of the augmenting paths that could be chosen for the third augmentation step.

Solution:

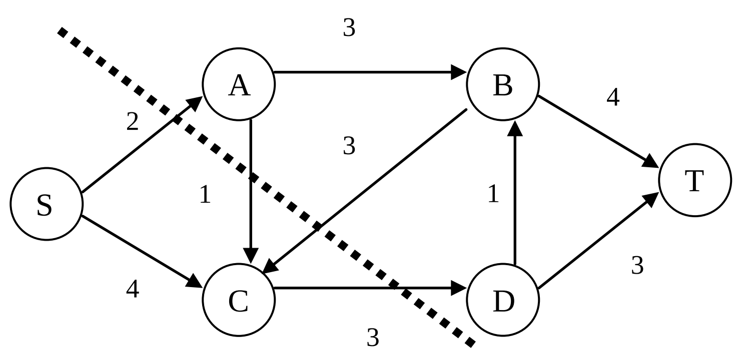
S-C-B-T
S-C-D-T
S-C-A-B-T
S-C-D-B-T

c) 4pts
What is the numerical value of the maximum flow? Draw a dotted line through the original graph to represent a minimum cut.

Solution:

The numerical value of the maximum flow is 5.

A minimum cut is shown in the following figure:



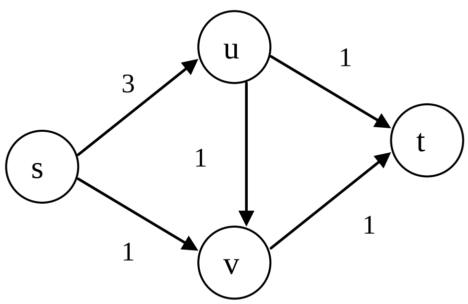
5) 20 pts
Decide whether you think the following statements are true or false. If true, give a short explanation. If false, give a counterexample.

Let G be an arbitrary flow network, with a source s, a sink t, and a positive integer capacity c_e on every edge e.

a) If f is a maximum s-t flow in G, then for all edges e out of s, we have $f(e) = c_e$.

Solution:

False.

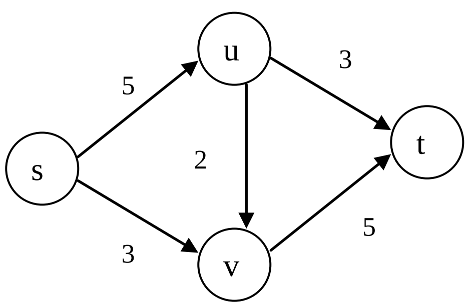


Clearly, the maximum s-t flow in the above graph is 2.
The edge (s, u) does not have $f(e) = c_e$

b) Let (A,B) be a minimum s-t cut with respect to the capacities $\{ c_e : e \in E\}$. Now suppose we add 1 to every capacity. Then (A,B) is still a minimum s-t cut with respect to these new capacities $\{1 + c_e : e \in E\}$.

Solution:

False.



Clearly, one of the minimum cuts is A={s,u} and B={v,t}. After adding 1 to every capacity, the maximum s-t flow becomes 10 and the cut between A and B is 11.

6) 15 pts

Suppose that in county X, there are only 3 kinds of coins: the first kind are 1-unit coins, the second kind are 4-unit coins, and the third kind are 6-unit coins. You want to determine how to return an amount of K units, where K≥0 is an integer, using as few coins as possible. For example, if K=7, you can use 2 coins (a 1-unit coin and a 6-unit coin), which is better than using three 1-unit coins and a 4-unit coins since in this case, the total number of coins used is 4.

For $0 \leq k \leq K$ and $1 \leq i \leq 3$, let c(i,k) be the smallest number of coins required to return an amount of k using only the first i kinds of coins. So for example, c(2,5) would be the smallest number of coins required to return 5 units if we can only use 1-unit coins and 4-unit coins. In what follows, you can assume that the denomination of the coins is stored in an array d, i.e., d[1]=1,d[2]=4,d[3]=6.

Give a dynamic programming algorithm that returns c(i,k) for $0 \leq k \leq K$ and $1 \leq i \leq 3$

Solution:

Let the coin denominations be $d_1, d_2, \ldots, d_i$.
Because of the optimal substructure, if we knew that an optimal solution for the problem of making change for k cents used a coin of denomination d_j, we would have
$c(i, k) = 1 + c(i, k - d_j)$.

As base cases, we have that $c(i, k) = 0$ for all $k \leq 0$.

To develop a recursive formulation, we have to check all denominations, giving

$$c(i, k) = \begin{cases} 0, & \text{if } k \leq 0 \\ 1 + \min_{1 \leq j \leq i} \{c(i, k - d_j)\}, & \text{if } k > 1 \end{cases}$$

The algorithm is as follows:

for i = 1…3
    for k =0…K
        Compute c(i, k)
    end for
end for

When $i = j$, to compute each $c(j, k)$ costs $O(j)$. Thus, the complexity is O(K+2K+3K) = O(6K)