CS570
Analysis of Algorithms
Summer 2005
Midterm Exam

Name: _____
Student ID: _____

Problem No.	Max. Points	Received
1	20	
2	15	
3	20	
4	15	
5	15	
6	15	
Total	100	

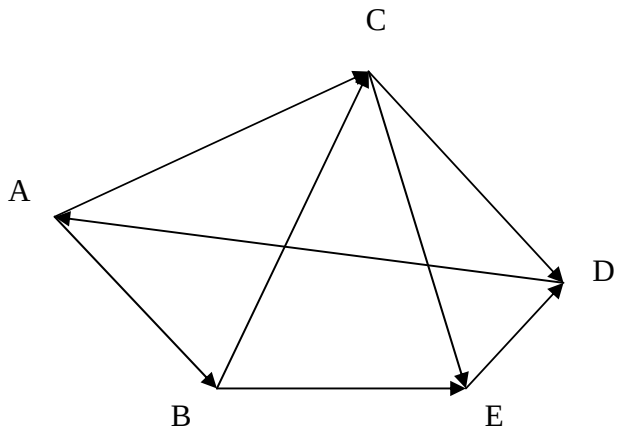
1) 20 pts
Consider the following two problems:
In P1 we are given as input a set of n squares (specified by their corner points) in the plane, and a number k. the problem is to determine whether there is any point in the plane that is covered by k or more squares.
In P2 we are given as input an n-vertex graph, and a number k; the problem is to determine whether there is a set of k mutually adjacent vertices. (E.g. for k = 3 we are looking for a triangle in the graph.)
(a) Are P1 and P2 in NP?

(b) Show a reduction between P1 and P2.

(c) If P1 is NP-complete, would your reduction in (b) (and answer to (a)) imply that P2 is NP-complete?

(d) If P2 is NP-complete, would your reduction in (b) (and answer to (a)) imply that P1 is NP-complete?

2) 15 pts
Is there a feasible circulation in G?
- If yes, show the flow on each edge.
You need to show all your work.
- If no, which edge capacity (capacities) need to be increased and by how much before a feasible circulation can be achieved?



Graph G

Directed Edges	Lower Bound	Capacity
AC	5	15
AB	10	15
DA	5	25
BC	5	15
BE	0	10
CE	5	10
CD	0	15
ED	5	20

	Demand
A	10
B	20
C	-5
D	-15
E	-10

3) 20 pts
We are given an undirected unweighted graph G = (V,E), and specified two vertices s, t Є V . The length of a path in this graph is defined to be the number of edges in the path. Recall that the shortest path between two vertices is a path connecting them that has minimum length.

a- Create a linear programming solution to the problem of finding the shortest path from s to t.

b- Prove that the optimal solution of the linear program gives the optimal value of the shortest path.

c- Describe how to construct the shortest path using the solution to the linear program.

4) 15 pts
Let G=(V,E) be a flow network with source s, sink t, and integer capacities, Suppose that we are given a maximum flow in G.

a- suppose that the capacity of a single edge (u,v) Є E is increased by 1. Give an O(V+E)-time algorithm to update the maximum flow.

b- Suppose that the capacity of a single edge (u,v) Є E is decreased by 1. Give an O(V+E)-time algorithm to update the maximum flow.

5) 15 pts

In a daring burglary, someone attempted to steal all the candy bars for the CS570 final. Luckily, he was quickly detected, and now, the course staff and students will have to keep him from escaping from campus. In order to do so, they can be deployed to monitor strategic routes.

More formally, we can think of the USC campus as a graph, in which the nodes are locations, and edges are pathways or corridors. One of the nodes (the instructor's office) is the burglar's starting point, and several nodes (the USC gates) are the escape points — if the burglar reaches any one of those, the candy bars will be gone forever. Students and staff can be placed to monitor the edges. As it is hard to hide that many candy bars, the burglar cannot pass by a monitored edge undetected.

Give an algorithm to compute the minimum number of students/staff needed to ensure that the burglar cannot reach any escape points undetected (you don't need to output the corresponding assignment for students — the number is enough). As input, the algorithm takes the graph $G = (V,E)$ representing the USC campus, the starting point s, and a set of escape points $P \subseteq V$. Prove that your algorithm is correct, and runs in polynomial time.

6) 15 pts

Let G = (V,E) be an undirected graph in which the vertices represent small towns and the edges represent roads between those towns. Each edge e has a positive integer weight d(e) associated with it, indicating the length of that road. The distance between two vertices (towns) in a graph is defined to be the length of the shortest weighted path between those two vertices.

Each vertex v also has a positive integer c(v) associated with it, indicating the cost to build a fire station in that town.

In addition, we are given two positive integer parameters D and C. Our objective is to determine whether there is a way to build fire stations such that the total cost of building the fire stations does not exceed C and the distance from any town to a fire station does not exceed D. This problem is known as the Rural Fire Station (RFS) Problem.

Your company has been hired by the American League of Rural Fire Departments to study this problem. After spending months trying unsuccessfully to find an efficient algorithm for the problem, your boss has a hunch that the problem is NP-complete. Prove that RFS is NP-complete.

CS570
Analysis of Algorithms
Summer 2006
Exam 2

Name: _____
Student ID: _____

	Maximum	Received
Problem 1	10	
Problem 2	20	
Problem 3	25	
Problem 4	25	
Problem 5	20	

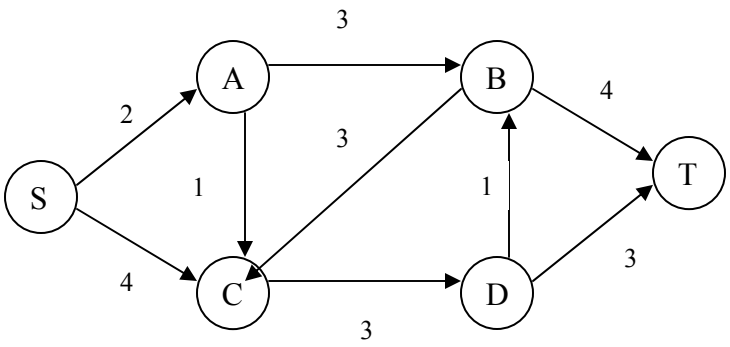
1) 10 pts
A divide and conquer algorithm is constructed the following way
Divide: Split the problem (originally of size n) into 8 equal pieces of size n/2. This takes $O(n^2)$ time.
Conquer: Solve the n subproblems recursively
Combine: Combine the subproblems. This takes O(nlgn) time.

a- What is the complexity of this algorithm?

b- What aspect of this algorithm has to improved in order to reduce the overall complexity of the algorithm.
1- time it takes to divide
2- number of subproblems at each step
3- size of the subproblems at each step
4- time it takes to combine
Mark all that apply and explain your answer.

2) 20 pts
Given an array P=(P(1),…,P(n)), use divide and conquer to find $\max_{0<i<j<n+1}$ (P(j)-P(i)) in O(n log n). (Note: this problem can be solved using other techniques but you are specifically asked to solved the problem using divide and conquer)

3) 25 pts
Question 1: 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) Draw the residual network after we have updated the flow using these two augmenting paths(in the order given)

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

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

4) 25 pts

Let G=(V,E) be a flow network with source s, sink t, and suppose each edge e have capacity c(e)=1. For convenience, assume that E=O(V)

a) Suppose we implement the Ford-Fulkerson maximum-flow algorithm by using DFS to find augmenting paths in the residual graph. What is the worst-case running time of the algorithm on G?

b) Suppose a maximum flow for G has been computed, and a new edge with unit capacity is added to E. Describe how the maximum flow can be efficiently updated. Analyze your algorithm(Note: It is not the value of the flow that must be updated, but the flow itself)

c) Suppose a maximum flow for G has been computed, but an edge is now removed from E. Describe how the maximum flow can be efficiently updated. Analyze your algorithm

5) 20 pts
Given a graph G=(V,E) and two nodes X and Y in V, present an algorithm to find the maximum number of **node** disjoint paths from X to Y. Include complexity analysis and proof of correctness.

CS570
Analysis of Algorithms
Fall 2006
Exam 2

Name: _____

Student ID: _____

	Maximum	Received
Problem 1	10	
Problem 2	20	
Problem 3	10	
Problem 4	20	
Problem 5	20	
Problem 6	20	

Note: The exam is closed book closed notes.

1) 10 pts

By using Strassen's algorithm, we can computes the product of two n x n matrices in O(n^2.81) time. This was achieved because we found a way to multiply the n/2 x n/2 matrices with only 7 multiplications rather than 8. Suppose we came up with a Strassen-like algorithm as follows. Assume that instead of splitting each matrix into four n/2 x n/2 matrices, we split each matrix into sixteen n/4 x n/4 matrices, and that the result is computed with only *m* multiplications instead of the normal 64. How small should *m* be for this new algorithm to be asymptotically faster than the original Strassen algorithm?

2) 20 pts

a- Given an array of integers A[1…n], give a divide-and-conquer algorithm to find the minimum ratio of A[i]/A[i+1] for 0<i<n. Your answer should clearly say how to divide the problem into subproblems and how to merge the results.

b- Give a recurrence relation for the time complexity of the algorithm and solve it.

3) 10 pts

Let $G = (V,E)$ be a flow network with source s, sink t, and integer capacities. Suppose that we are given a maximum flow in G. Suppose that the capacity of a single edge (u, v) is increased by 1, Give an $O(V + E)$ time algorithm to update the maximum flow.

4) 20 pts

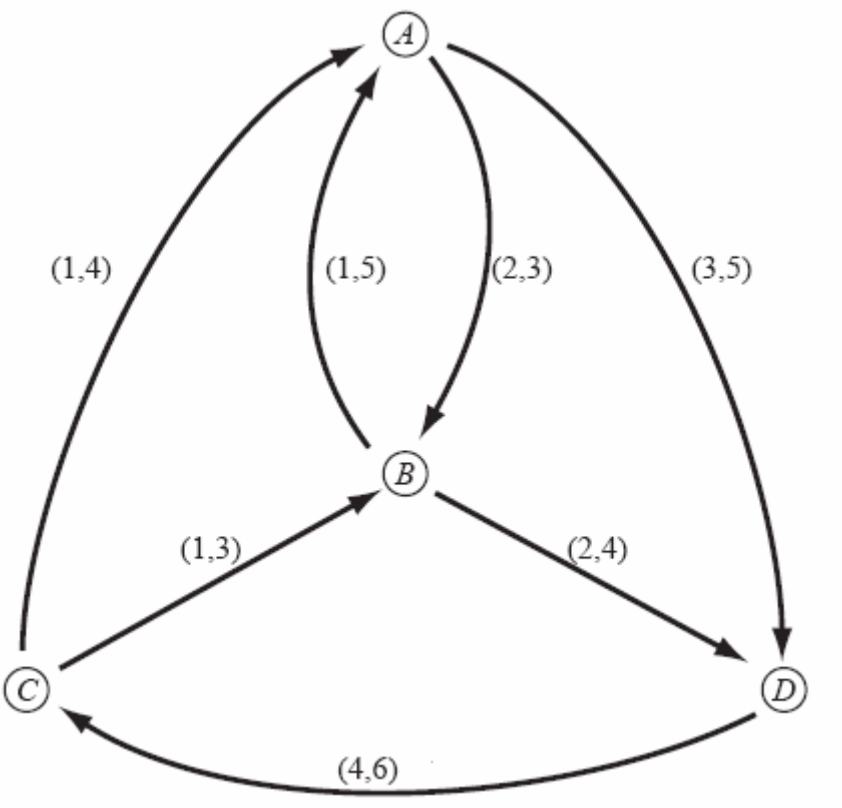
a- Prove or disprove the following:

If all edges in a flow network have distinct capacities, then there is a unique maximum flow.

b-If all of the edges have unique capacity values, the network has a unique min-cut.

5) 20 pts
Seven construction equipments are to be flown to a destination by five commercial planes. There are 4 units of each kind of construction equipment, and the 5 commercial planes can carry 8, 8, 5, 4, and 4 units, respectively. Can this construction equipment be loaded in such a way that no two units of the same kind are in the same plane?

6) 20 pts
Solve the following feasible circulation problem below. The numbers in parentheses are (*lowerbound, upperbound*) on flow.



CS570
Analysis of Algorithms
Spring 2007
Exam 2

Name: _____
Student ID: _____

	Maximum	Received
Problem 1	20	
Problem 2	10	
Problem 3	20	
Problem 4	20	
Problem 5	20	
Problem 6	5	
Problem 7	5	

Note: The exam is closed book closed notes.

1) 20 pts
Mark the following statements as **TRUE** or **FALSE**. No need to provide any justification.

[TRUE/FALSE] True
Max flow problems can in general be solved using greedy techniques.

[TRUE/FALSE] False
If all edges have unique capacities, the network has a unique minimum cut.

[TRUE/FALSE] True
Flow f is maximum flow if and only if there are no augmenting paths.

[TRUE/FALSE] True
Suppose a maximum flow allocation is known. Increase the capacity of an edge by 1 unit. Then, updating a max flow can be easily done by finding an augmenting path in the residual flow graph.

[TRUE/FALSE] False
In order to apply divide & conquer algorithm, we must split the original problem into at least half the size.

[TRUE/FALSE] True
If all edge capacities in a graph are integer multiples of 5 then the maximum flow value is a multiple of 5.

[TRUE/FALSE] False
If all directed edges in a network have distinct capacities, then there is a unique maximum flow.

[TRUE/FALSE] True
Given a bipartite graph and a matching pairs, we can determine if the matching is maximum or not in O(V+E) time

[TRUE/FALSE] False
Maximum flow problem can be efficiently solved by dynamic programming

[TRUE/FALSE] True
The difference between dynamic programming and divide and conquer techniques is that in divide and conquer sub-problems are independent

2) 10pts
Give tight bound for the following recursion

a) $T(n)=T(n/2)+T(n/4)+n$

A simple method is that it is easy to easy that $T(n)=4n$ satisfy the recursive relation.
Hence $T(n)=O(n)$

b) $T(n) = 2T(n^{0.5})+\log n$

Let $n=2^k$ and $k=2^m$, we have $T(2^k)= 2T(2^{k/2})+k$ and $T(2^{k/2})= 2T(2^{k/4})+k/2$
Hence $T(2^k)=2(2T(2^{k/4})+k/2)+k=2^2\ T(2^{k/4})+2k=2^3\ T(2^{k/8})+3k=\ldots.=2^mT(1)+mk$
$=2^mT(1)+m2^m$. Note that $n=2^k$ and $k=2^m$, hence $m=\log\log n$. We have $T(n)=T(1)\log n+(\log\log n)*(\log n)$.
Hence $T(n)= (\log\log n)*(\log n)$

3) 20 pts

Let A = (a_0, a_1, . . . , a_{n-1}) be an array of n positive integers.

a- Present a divide-and-conquer algorithm which computes the sum of all the even integers a_i in A(1..n-1) where $a_i > a_{i-1}$. Solutions other than divide and conquer will receive no credit.

Algorithm: Compute-Even-Integers(l,r)
if r=l+1
    if((a_r%2=0) and ($a_r > a_l$))
        return a_r
    else
        return 0
else
    let m=$\left\lfloor \frac{l+r}{2} \right\rfloor$
    Let S= Compute-Even-Integers(l,m)+ Compute-Even-Integers(m,r)
    If a_m%2=0 and $a_m > a_{m-1}$
        S=S+ a_m

To computes the sum of all the even integers a_i in A(1..n-1) where $a_i > a_{i-1}$, invoke Compute-Even-Integers(0,n-1)

b- Show a recurrence which represents the number of additions required by your algorithm.

T(n)=2T(n/2)+c

c- Give a tight asymptotic bound for the number of additions required by your algorithm.
By master theorem, it is easy to see that $T(n)=O(n)$

d- Discuss how your approach would compare in practice to an iterative solution of the problem.
In practice, both methods are O(n) algorithm. Their complexity is the same.

4) 20 pts
Families 1…..N go out for dinner together. To increase their social interaction, no two members of the same family use the same table. Family j has a(j) members. There are M tables. Table j can seat b(j) people. Find a valid seating assignment if one exists.

We first construct a bipartite graph G whose node are the families f_i(i=1,…,N) and the tables t_j(j=1,…,M). We add edge e_{ij}=(f_i, t_j) in the graph for i=1,…,N and j=1,…,M and set e_{ij}=1. Then we add a course s and sink in G. For each family f_i, we add e=(s, f_i) in the graph and set c_e=a(i). For each stable t_j, we add e=(t_j, t) and set c_e=b(j) . After building the graph G for the original problem, we find the maximum s-t-flow value v in graph G by Fulkerson algorithm. If v is a(1)+…+a(N), then we make the seating assignment such that no two members of the same family use the same table, otherwise we can not.

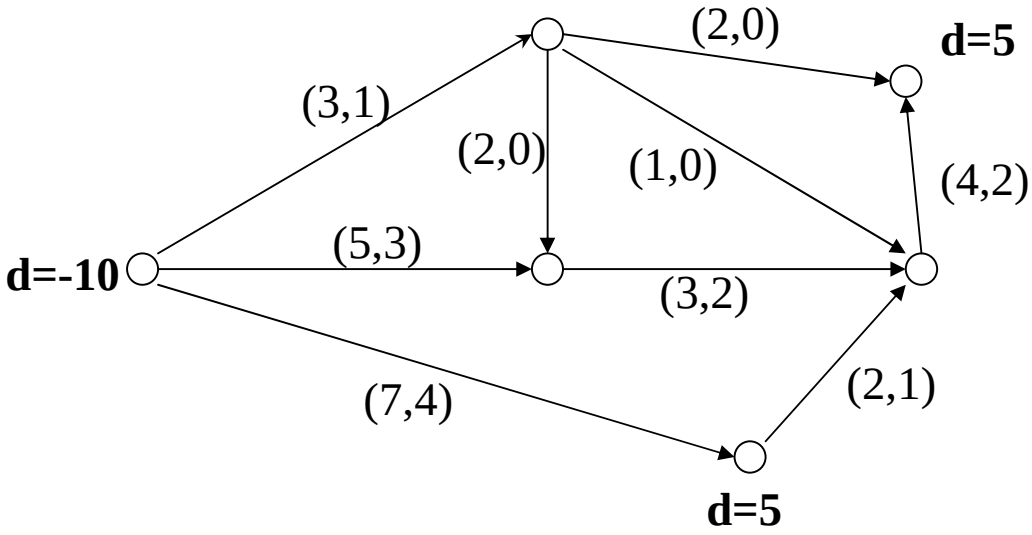
To prove the correctness of the algorithm, we prove that no two members of the same family use the same table if and only if the value of the maximum value of an s-t flow in G is a(1)+…+a(N):

First if no two members of the same family use the same table, it is easy to see that this flow meets all capacity constraints and has value a(1)+…+a(N).

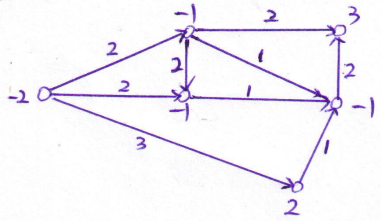
For the converse direction, assume that there is and s-t flow of value a(1)+…+a(N). The construction given in the algorithm makes sure that the there is at most one member in family j sitting in the table j as the capacity of the flow goes from f_i to c_j is 1. So we only need to make sure that all families are seated. Note that {s}, V-{s} is an s-t cut with value a(1)+…+a(N), so the flow must saturate all edges crossing the cut. Therefore, all families must be sitting in some tables. This completes the correctness proof.

Running time: The time complexity of construcing the graph is O(MN). The number of edges in the graph is MN+M+N, and v(f)= a(1)+…+a(N). Let T= a(1)+…+a(N), then the running time applying Ford-Fulkerson algorithm is O(MNT).

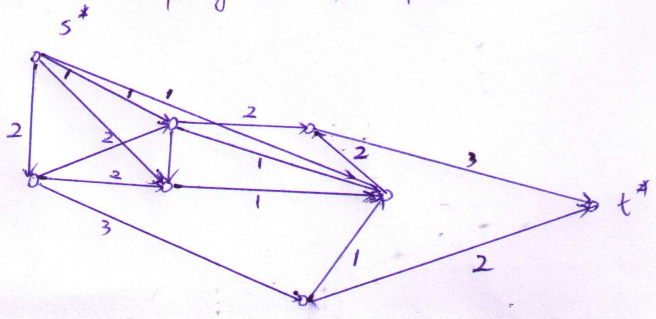
5) 20 pts
Determine if there is a feasible circulation in the below network. If so, determine the circulation, if not show where the bottlenecks are. The numbers in parentheses are (*lowerbound, upperbound*) on flow.



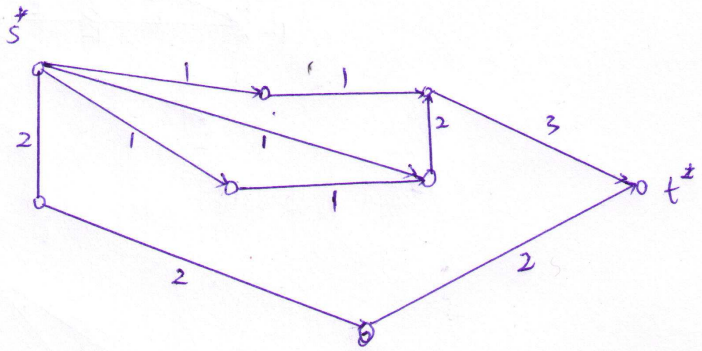
Step 1:



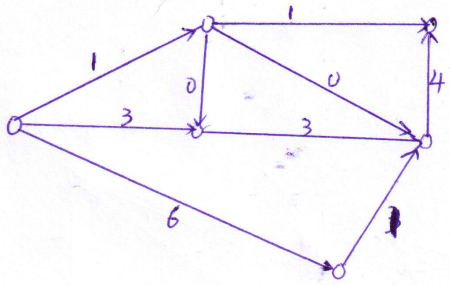
step 2: the ~~the~~ corresponding max-flow problem



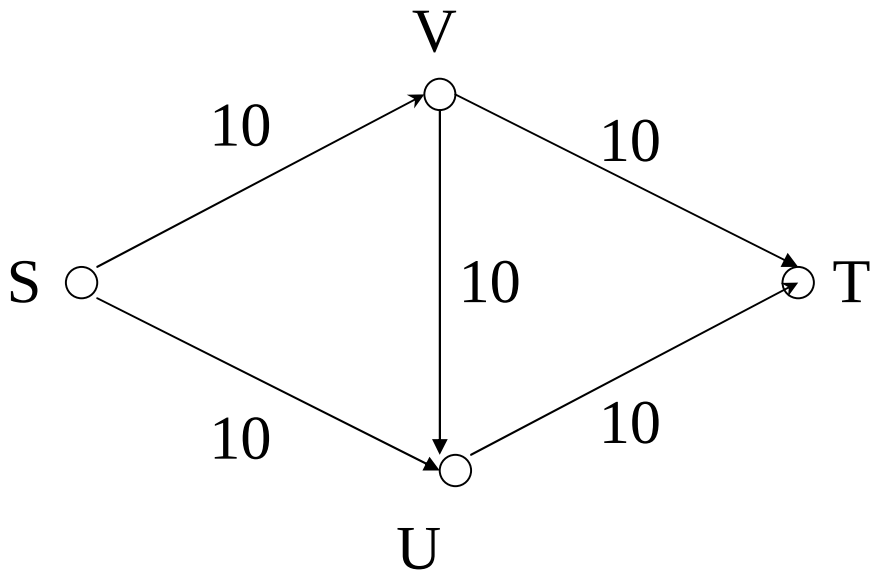
Step 3. Solving the max-flow problem and the result is as follows:



Step 4. The circulation is



6) 5 pts
List all minimum s-t cuts in the flow network pictured below. Capacity of every edge is equal to 10.



C1={{S},{V,U,T}}
C2={{S,U},{V,T}}
C3={{S,U,V},{T}}

7) 5 pts
Show an example of a graph in which poor choices of augmenting paths can lead to exactly C iterations of the Ford-Fulkerson algorithm before achieving max flow. (C is the sum of all edge capacities going out of the source and must be much greater than the number of edges in the network) Specifically, draw the network, mark up edge capacities, and list the sequence of augmenting paths.

Any example satisfying that the condition in this question is fine when you list the sequence of augmenting paths.

CS570
Analysis of Algorithms
Summer 2007
Exam 2

Name: _____

Student ID: _____

	Maximum	Received
Problem 1	10	
Problem 2	20	
Problem 3	20	
Problem 4	10	
Problem 5	20	
Problem 6	20	

Note: The exam is closed book closed notes.

1) 10 pts
Mark the following statements as **TRUE** or **FALSE**. No need to provide any justification.

True **[TRUE/FALSE]**
Binary search could be called a divide and conquer technique

False **[TRUE/FALSE]**
If you have non integer edge capacities, then you cannot have an integer max flow

Ture [TRUE/FALSE]
The Ford Fulkerson algorithm with real valued capacities can run forever

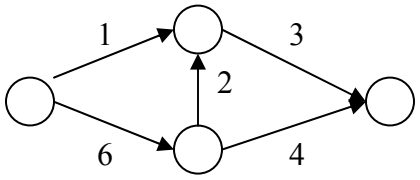
Ture [TRUE/FALSE]
If we have a 0-1 valued s-t flow in a graph of value f, then we have f edge disjoint s-t paths in the graph

Ture [TRUE/FALSE]
Merge sort works because at each level of the algorithm, the merge step assumes that the two lists are sorted

2) 20pts
Prove or disprove the following for a given graph G(V,E) with integer edge capacities Ci

a. If the capacities on each edge are unique then there is a unique min cut

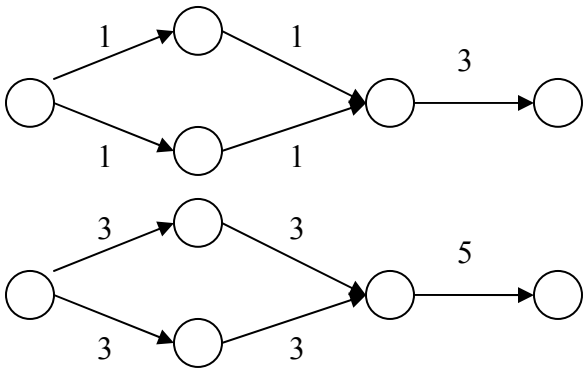
Disprove :



b. If we multiply each edge with the same multiple "f", the max flow also gets multiplied by the same factor

Prove: For each cut (A, B) of the graph G, the capacity c(A, B) will be multiplied by "f" if each edge's capacity is multiplied by "f", thus the minimal cut will be multiplied by "f" , thus the max flow also gets multiplied by "f".

c. If we add the same amount, say "a" to every edge in the graph, then the min cut stays the same (the edges in the min cut stay the same i.e.)



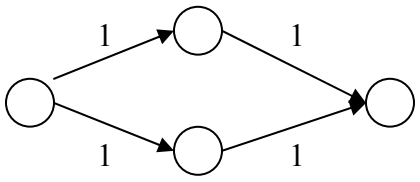
Before add 2 to every edge

After add 2 to every edge

d. If the edge costs are all even, then the max flow(min cut) is also even

The capacity of any cut (A, B) is sum of the capacities of all edges out of A, thus the capacity of any cut, including the minimal one, is also even.

e. If the edge costs are all odd, then the max flow (min cut) is also odd



Max flow: 2

3) 20 pts
Suppose you are given k sorted arrays, each with n elements, and you want to combine them into a single sorted array of kn elements.

(a) Here's one strategy: merge the first two arrays, then merge in the third, then merge in the fourth, and so on. What is the time complexity of this strategy, in terms of k and n?

The merge the first two array takes O(n), … the merge of the last array takes O(kn), totally there are k merge sorts. Thus, it takes $O(k^2n)$ times.

(b) Present a more efficient solution to this problem.
Let the final array to be A, initially A is empty.
Build up a binary heap with the first element from the k given arrays, thus this binary tree consists of k element.
Extract the minimal element from the binary tree and add it to A, delete it from its original array, and insert the next element from that array into the binary heap.

Each heap operation is O(logk), and take O(kn) operations, thus, the running time is O(kn*logk)

4) 10 pts
Derive a recurrence equation that describes the following code. Then, solve the recurrence equation.

```
COMPOSE (n)
    1. for i ← 1 to n
    2.         do for j ← 1 to sqrt(n)
    3.                 do print(i, j, n)
    4. if n > 0
    5.         then for i ← 1 to 5
    6.                 COMPOSE (⌊n/2⌋)
```

$T(n) = 5\ T(n/2) + O(n^{3/2})$

By the Master theorem, $T(n) = n^{\lg 5}$

5) 20 pts

Let G=(V,E) be a directed graph, with source $s \in V$, sink $t \in V$, and non-negative edge capacities $\{C_e\}$. Give a polynomial time algorithm to decide whether G has a unique minimum s-t cut. (i.e. an s-t cut of capacity strictly less than that of all other s-t cuts.)

Find a max flow f for this graph G, and then construct the residual graph G' based on this max flow f. Start from the source s, perform breadth-first or depth-first search to find the set A that s can reach, define B = V – A, the cut (A, B) is a minimum s-t cut. Then, for each node v in B that is connected to A in the original graph G, try the following codes: add v into set A, and then perform breadth-first or depth-first search find a new set A' that s can reach, if the sink t is included in A', then try next node v', otherwise, report a new minimum s-t cut. If all possible nodes v in B have been tried but no more minimum s-t cut can be found, then report the (A, B) is the unique minimum s-t cut.

20 pts
Consider you have three courses to study, C1, C2 and C3. For the three courses you need to study a minimum of T1, T2, and T3 hours respectively. You have three days to study for these courses, D1, D2 and D3. On each day you have a maximum of H1, H2, and H3 hours to study respectively. You have only 12 hours total to give to all of these courses, which you could distribute within the three days. On each one of the days, you could potentially study all three courses. Give an algorithm to find the distribution of hours to the three courses on the three days

If T1+T2+T3 > 12 or T1+T2+T3 > H1 + H2 + H3, then report no feasible distribution can be found.
Else, start C1 with T1 hours, and then C2 with T2 hours, and finally C3 with T3 hours.

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.

Solution:

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 $i = 1 \ldots n$

  Compute D_i

end for

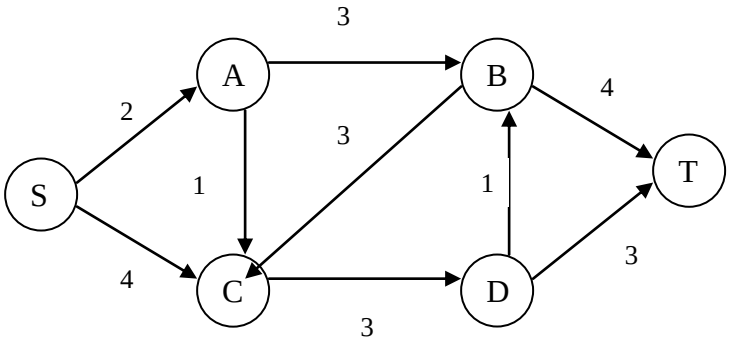
return the largest number among D_1 *to* D_n.

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.

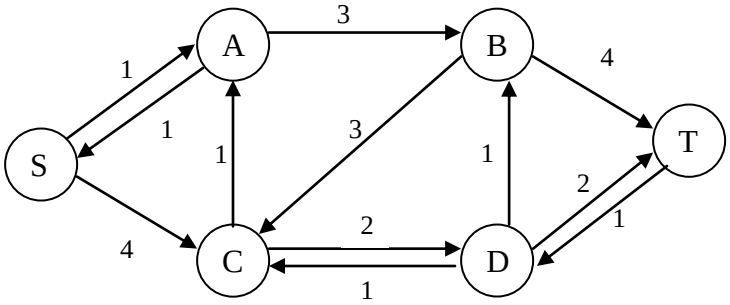


a) 10 pts

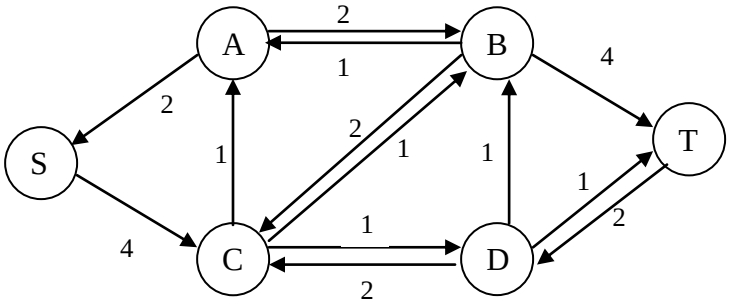
Draw the residual network after we have updated the flow using these two augmenting paths(in the order given)

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:

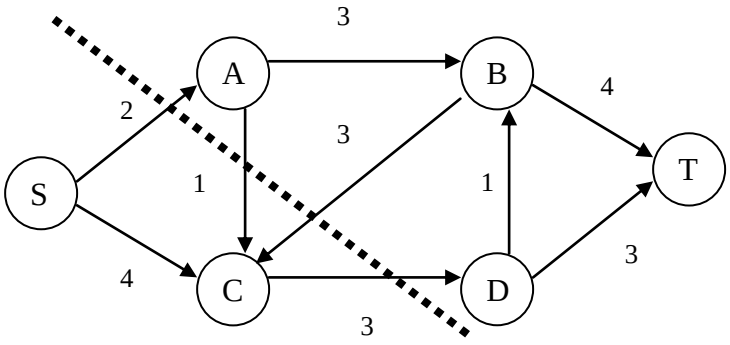
S-C-B-T
S-C-D-T
S-C-A-B-T
S-C-D-B-T

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

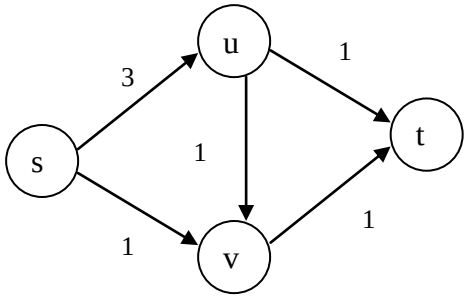
Decide whether you think the following statements are true or false. If true, give a short explanation. If false, give a counterexample.

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.

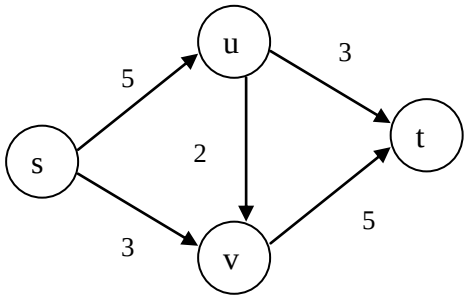


Clearly, the maximum s-t flow in the above graph is 2.
The edge (s, u) does not have $f(e) = c_e$

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)

CS570
Analysis of Algorithms
Fall 2008
Exam II

Name: _____

Student ID: _____

____Monday Section ____Wednesday Section ____Friday Section

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

2 hr exam
Close book and notes

1) 20 pts
Mark the following statements as **TRUE** or **FALSE**. No need to provide any justification.

FALSE [TRUE/FALSE]
For every node in a network, the total flow into a node equals the total flow out of a node.

TRUE [TRUE/FALSE]
Ford Fulkerson works on both directed and undirected graphs.

Because at every augmentation step you just need to find a path form s to t. This can be done in both direct and undriected graphs.

NO [YES/NO]
Suppose you have designed an algorithm which solves a problem of size n by reducing it to a max flow problem that will be solved with *Ford Fulkerson*, however the edges can have capacities which are $O(2^n)$. Is this algorithm efficient?

YES [YES/NO]
Is it possible for a valid flow to have a flow cycle (that is, a directed cycle in the graph, such that every edge has positive flow)?

positive flow cycles don't cause any problems. The flow can still be valid.

FALSE[TRUE/FALSE]
Dynamic programming and divide and conquer are similar in that in each approach the sub-problems at each step are completely independent of one another.

FALSE [TRUE/FALSE]
Ford Fulkerson has pseudo-polynomial complexity, so any problem that can be reduced to Max Flow and solved using *Ford Fulkerson* will have pseudo-polynomial complexity.

For example the edge disjoint paths problem is solved using FF in polynomial time. This is because for some problems the capacity C becomes a function of n or m.

TRUE [TRUE/FALSE]
In a flow network, the value of flow from S to T can be higher than the number of edge disjoint paths from S to T.

FALSE [TRUE/FALSE]
Complexity of a dynamic programming algorithm is equal to the number of unique sub-problems in the solution space.

TRUE [TRUE/FALSE]
In *Ford-Fulkerson*'s algorithm, when finding an augmentation path one can use either BFS or DFS.

TRUE **[TRUE/FALSE]**
When finding the value of the optimal solution in a dynamic programming algorithm one must find values of optimal solutions for all of its sub-problems.

2) 15 pts
Given a sequence of n real numbers $A_1 \ldots A_n$, give an efficient algorithm to find a subsequence (not necessarily contiguous) of maximum length, such that the values in the subsequence form a strictly increasing sequence.

Let A[i] represent the i-th element in the sequence.

```
initialize OPT[i] = 1 for all i.
best = 1
for(i = 2..n) {
  for(j = 1..i-1) {
    if(A[j] < A[i]) {
      OPT[i] = max( OPT[i], OPT[j] + 1 )
    }
    best = max( best, OPT[i] )
  }
}
return best
```

The runtime of the above algorithm is O(n^2)

3) 15 pts

Suppose you are given a table with N x M cells, each having a certain quantity of apples. You start from the upper-left corner and end at the lower right corner. At each step you can go down or right one cell. Give an efficient algorithm to find the maximum number of apples you can collect.

Let the top-left corner be in row 1 column 1, and the bottom-right corner be in row n column m.

Let A[i,j] represent the number of apples at row i column j.

```
initialize OPT[i,j] = 0 for all i,j.
for(i = 1...n) {
  for(j = 1...m) {
    OPT[i,j] = A[i,j] + max( OPT[i-1,j], OPT[i,j-1] )
  }
}
return OPT[n,m]
```

The runtime of the above algorithm is O(nm).

4) 15 pts

Suppose that you are in charge of a large blood bank, and your job is to match donor blood with patients in need. There are *n* units of blood, and *m* patients each in need of one unit of blood. Let us assume that the only factor which matters is that the blood type be compatible according to the following rules:

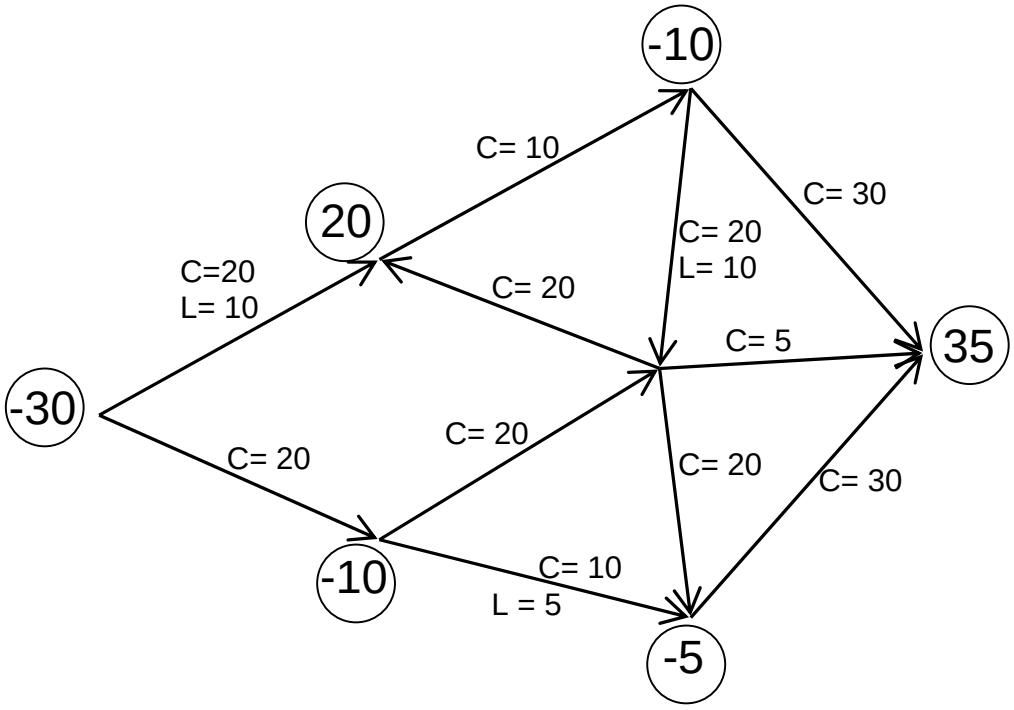
(a) Patient with type AB can receive types O, A, B, AB (universal recipient)
(b) Patient with type A can receive types O, A
(c) Patient with type B can receive types O, B
(d) Patient with type O can receive type O

Give a network flow algorithm to find the assignment such that the maximal number of patients receive blood, and prove its correctness.

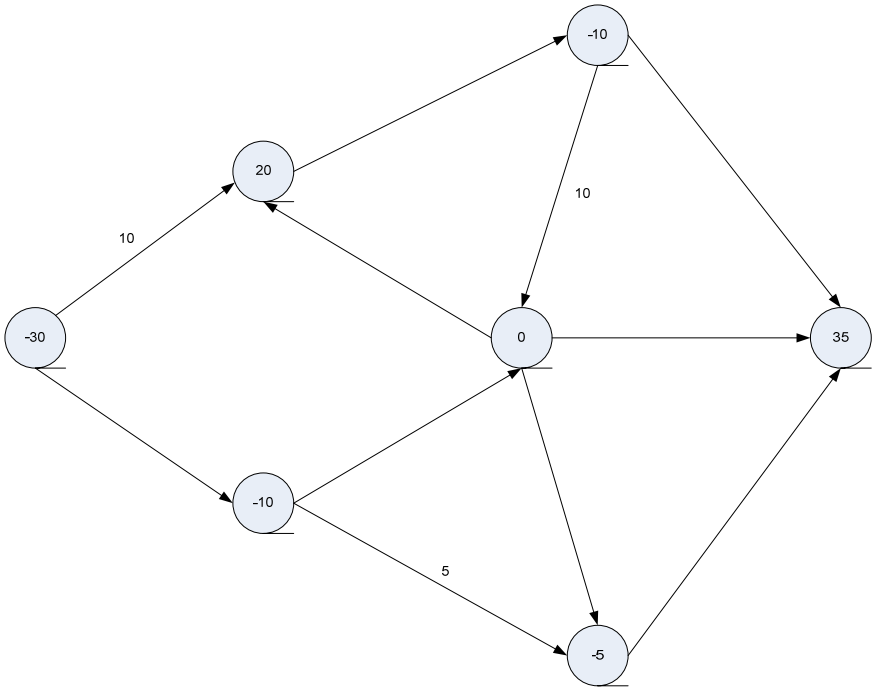
Given a set of n units of donor blood, and m patients in need of blood, each with some given blood type. We construct a network as follows. There is a source node, and a sink node, n donor nodes di, and m patient nodes pj. We connect the source to every donor node di with a capacity of 1. We connect each donor node di to every patient node pj which has blood type compatible with the donor's blood type, with a capacity of 1. We connect each patient node pj to the sink with capacity 1.

We then find the maximum flow in the network using Ford-Fulkerson. Patient j receives blood from donor i if and only if there is a flow of 1 from di to pj in the maximum flow. The total flow into each donor node di is bounded by 1, and the total flow out of each patient node pj is bounded by capacity 1, and so by conservation of flow, each patient and each donor can only give or receive 1 unit of blood. The types will be compatible by construction of the network.

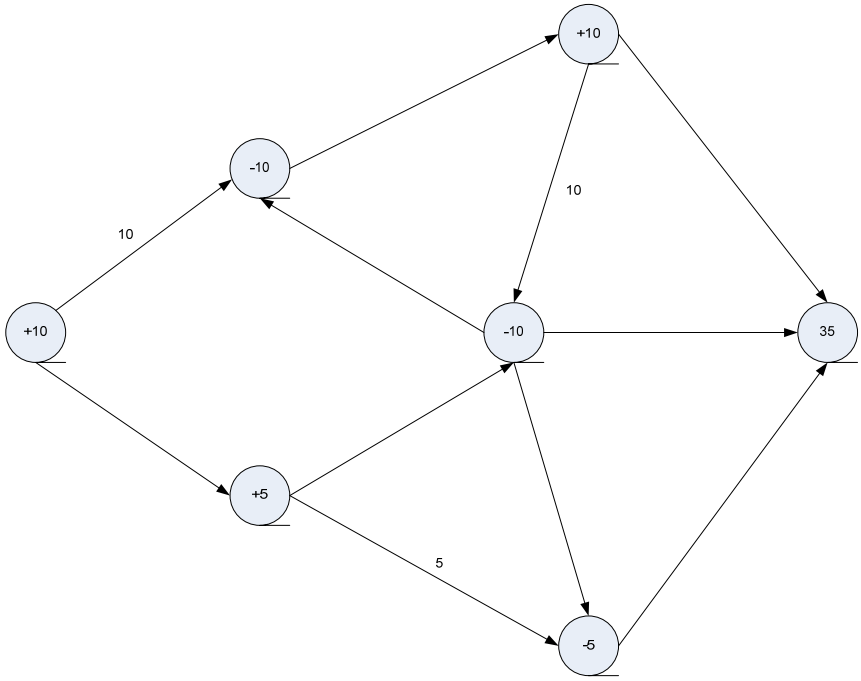
5) 20 pts
Given the graph below with demands/supplies as indicated below and edge capacities and possible lower bounds on flow marked up on each edge, find a feasible circulation. Show all your work



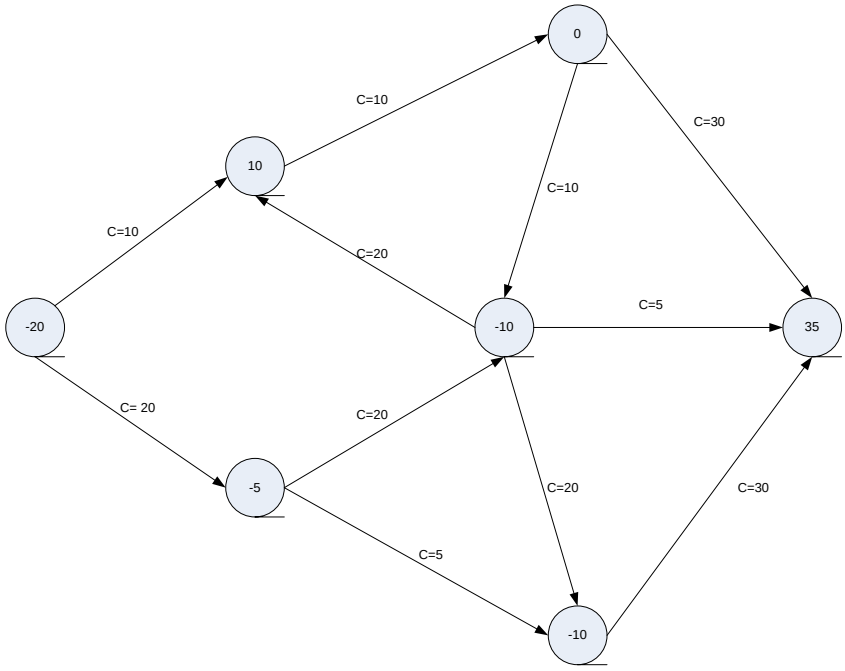
Solution to Q5
f0



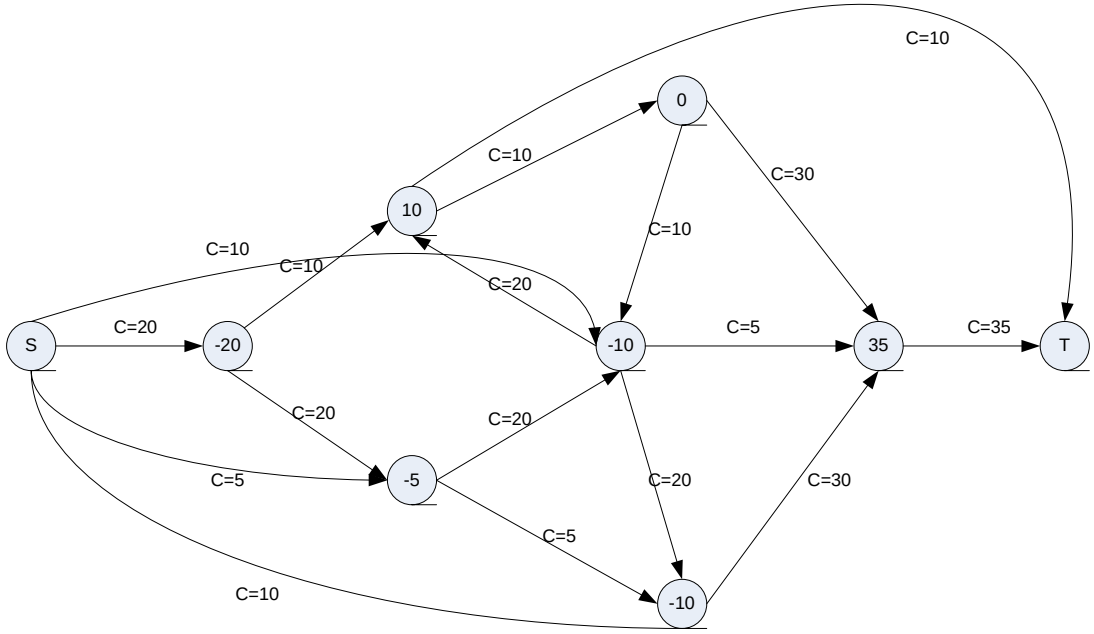
Lv



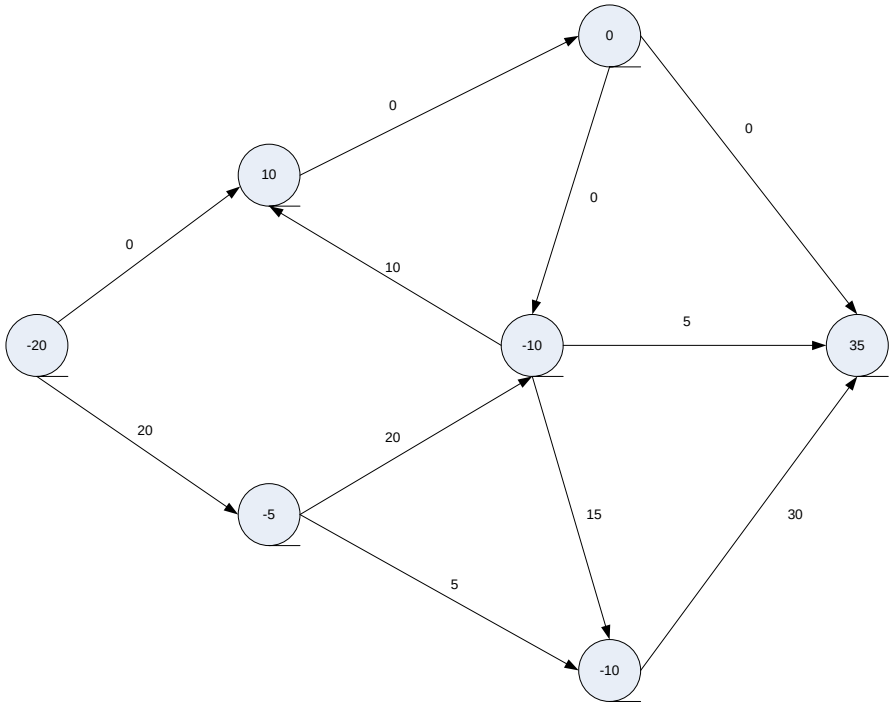
G’



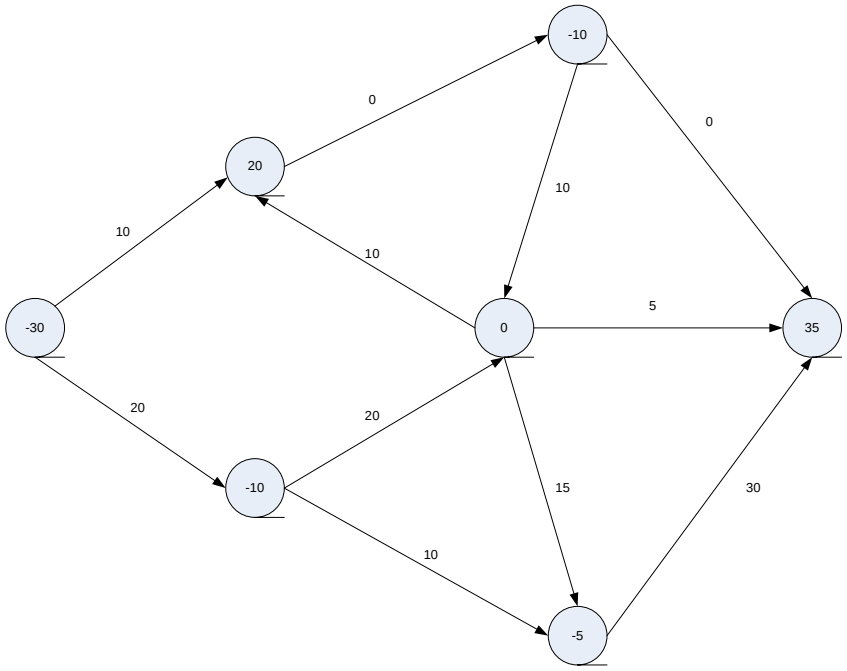
Convert G’ to a st network



f1



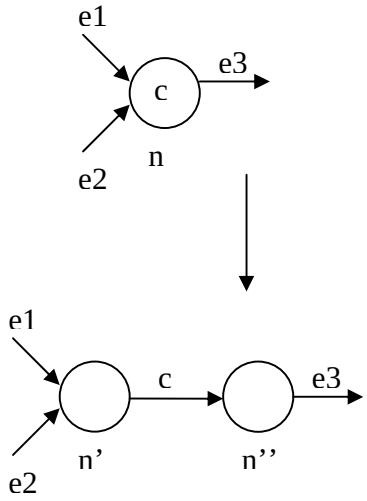
Circulation = f0+f1



6) 15 pts
Suppose that in addition to each arc having a capacity we also have a capacity on each node (thus if node i has capacity c_i then the maximum total flow which can enter or leave the node is c_i). Suppose you are given a flow network with capacities on both arcs and nodes. Describe how to find a maximum flow in such a network.

Solution:

Assume the initial flow network is G, for any node *n* with capacity *c*, decompose it into two nodes *n'* and *n''*, which is connected by the edge (*n'*, *n''*) with edge capacity *c*. Next, connect all edges into the node *n* in G to the node *n'*, and all edges out of the node n in G out of the node *n''*, as shown in the following figure.



After doing this for each node in G, we have a new flow network G'. Just run the standard network flow algorithms to find the maximal flow.

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

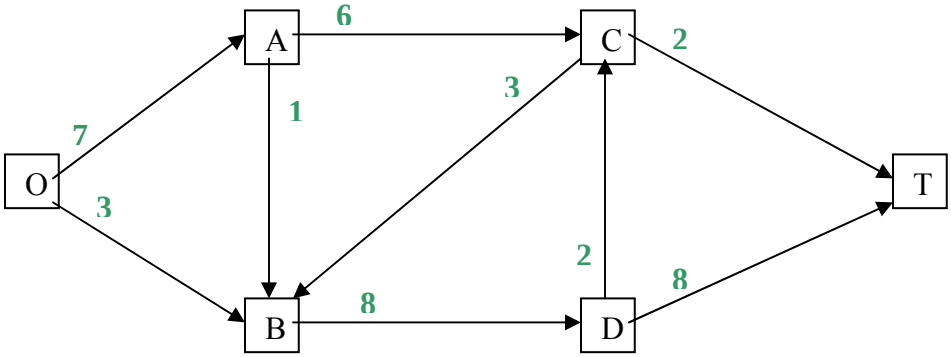
Present a divide-and-conquer algorithm that determines the minimum difference between any two elements of a sorted array of real numbers.

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.

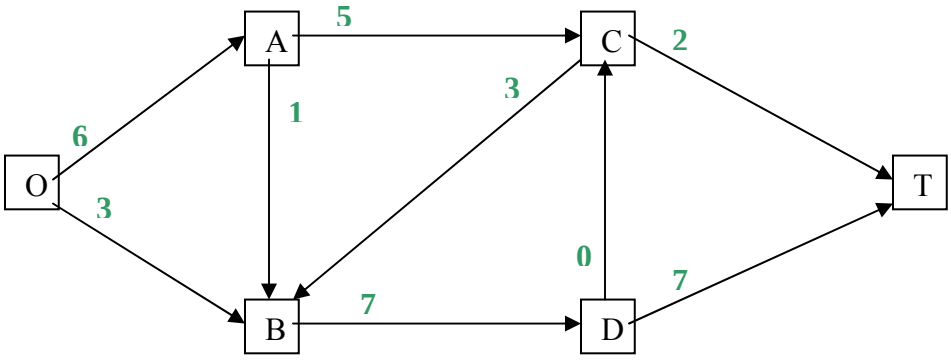
3) 15 pts
You are given the following directed network with source O and sink T.



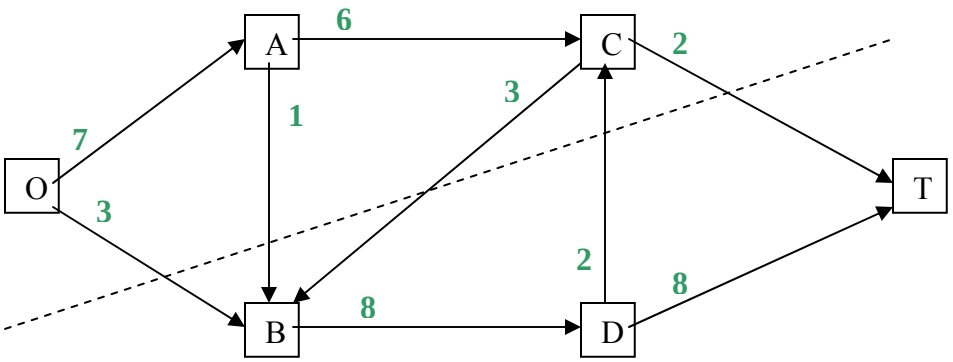
a) Find a maximum flow from O to T in the network.

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:

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

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

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

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 $n \geq 2$; $T(0) = 3$; $T(1) = 3$

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.

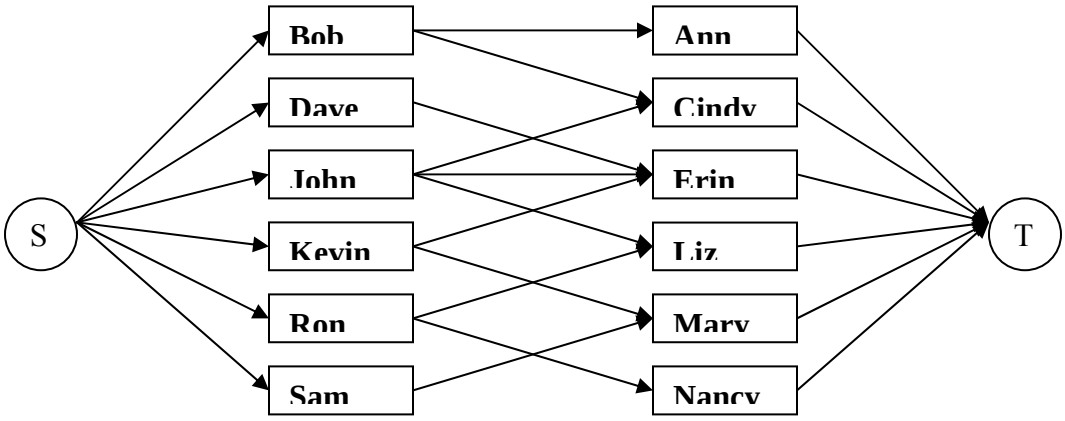
6) 15 pts

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.

CS570
Analysis of Algorithms
Spring 2009
Exam II

Name: _____

Student ID: _____

____2:00-5:00 Friday Section ____5:00-8:00 Friday Section

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

2 hr exam
Close book and notes

1) 20 pts
   Mark the following statements as **TRUE** or **FALSE**. No need to provide any justification.

[TRUE/FALSE] TRUE
The problem of deciding whether a given flow f of a given flow network G is maximum flow can be solved in linear time.

[TRUE/FALSE] TRUE
If you are given a maximum s - t flow in a graph then you can find a minimum s - t cut in time $O(m)$.

[TRUE/FALSE] TRUE
An edge that goes straight from s to t is always saturated when maximum s - t flow is reached.

[TRUE/FALSE] FALSE
In any maximum flow there are no cycles that carry positive flow.
(A cycle $<e_1, \ldots, e_k>$ carries positive flow iff $f(e_1) > 0, \ldots, f(e_k) > 0$.)

[TRUE/FALSE] TRUE
There always exists a maximum flow without cycles carrying positive flow.

[TRUE/FALSE] FALSE
In a directed graph with at most one edge between each pair of vertices, if we replace each directed edge by an undirected edge, the maximum flow value remains unchanged.

[TRUE/FALSE] FALSE
The Ford-Fulkerson algorithm finds a maximum flow of a unit-capacity flow network (all edges have unit capacity) with n vertices and m edges in $O(mn)$ time.

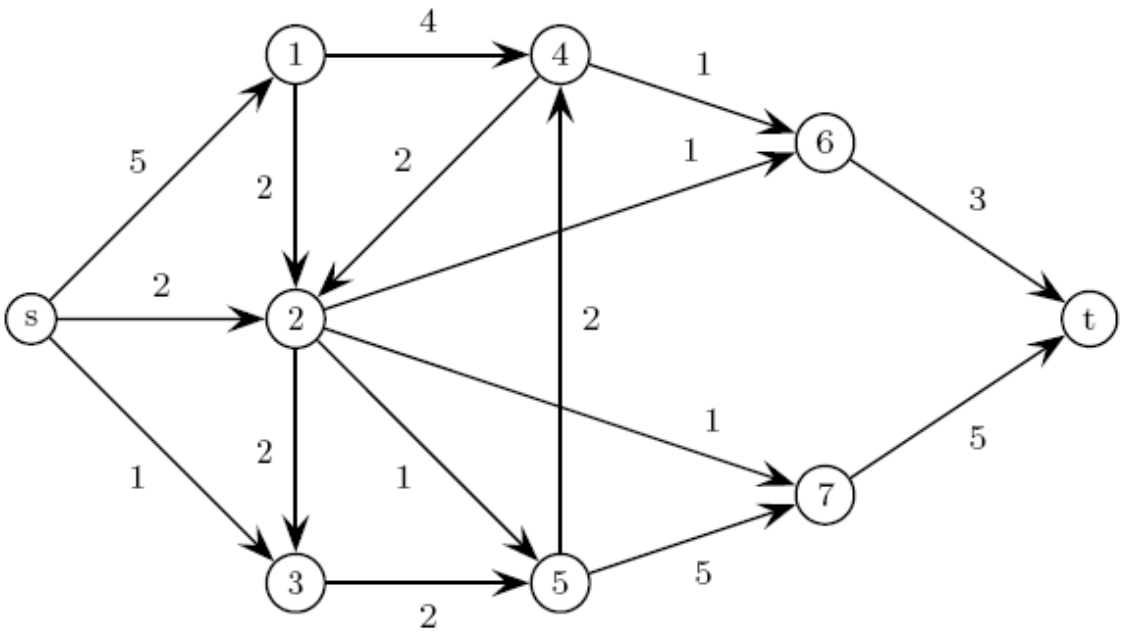
[TRUE/FALSE] FALSE
Any Dynamic Programming algorithm with n unique subproblems will run in $O(n)$ time.

[TRUE/FALSE] FALSE
The running time of a pseudo polynomial time algorithm depends polynomially on the size of of the input

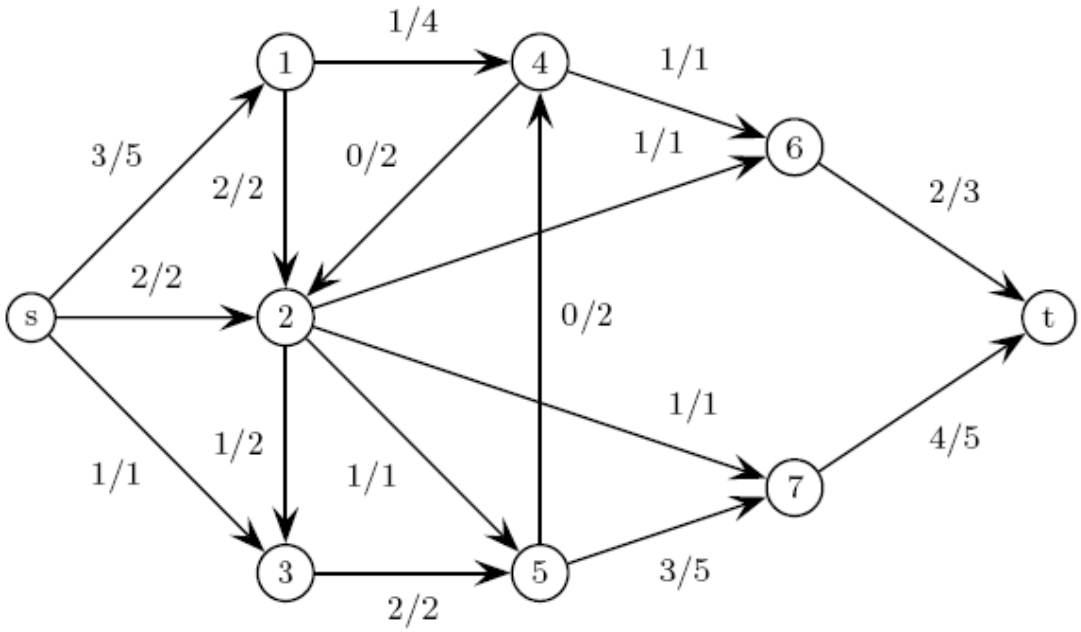
[TRUE/FALSE] FALSE
In dynamic programming you must calculate the optimal value of a subproblem twice, once during the bottom up pass and once during the top down pass.

2) 20 pts

a) Give a maximum *s*-*t* flow for the following graph, by writing the flow f_e above each edge e. The printed numbers are the capacities. You may write on this exam sheet.



Solution:



(b) Prove that your given flow is indeed a max-flow.

Solution:

The cut ({s: 1, 2, 3, 4}, {5, 6, 7}) has capacity 6. The flow given above has value 6. No flow can have value exceeding the capacity of any cut, so this proves that the flow is a max-flow (and also that the cut is a min-cut).

3) 20 pts
On a table lie *n* coins in a row, where the i*th* coin from the left has value $x_i >= 0$. You get to pick up any set of coins, so long as you never pick up two adjacent coins. Give

a polynomial-time algorithm that picks a (legal) set of coins of maximum total value. Prove that your algorithm is correct, and runs in polynomial time.

We use Dynamic Programming to solve this problem. Let OPT(i) denote the maximum value that can be picked up among coins 1, . . . , *i*.
Base case: OPT(0) = 0, and OPT(1) = x1.

Considering coin *i*, there are two options for the optimal solution: either it includes coin *i*, or it does not. If coin *i* is not included, then the optimal solution for coins 1, . . . , *i* is the same as the one for coins 1, . . . , *i*-1. If coin *i* is included in the optimal solution, then coin *i*-1 cannot be included, but among coins 1, . . . , *i* − 2, the optimum subset is included, as a non-optimum one could be replaced with a better one in the solution for i. Hence, the recursion is

OPT(0) = 0
OPT(1) = x_1
OPT(i) = max(OPT(i-1), x_i + OPT(i − 2)) for i > 1.

Hence, we get the algorithm Coin Selection below: The correctness of the computation follows because it just implements the recurrence which we proved correct above. The output part just traces back the array for the solution constructed previously, and outputs the coins which have to be picked to make the recurrence work out.

The running time is O(n), as for each coin, we only compare two values.

4) 20 pts
We assume that there are n tasks, with time requirements $r_1, r_2, \ldots, r_n$ hours. On the project team, there are *k* people with time availabilities $a_1, a_2, \ldots, a_k$. For each task *i*

and person *j*, you are told if person *j* has the skills to do task *i*. You are to decide if the tasks can be split up among the people so that all tasks get done, people only execute tasks they are qualified for, and no one exceeds his time availability. Remember that you can split up one task between multiple qualified people. Now, in addition, there are group constraints. For instance, even if each of you and your two roommates can in principle spend 4 hours on the project, you may have decided that between the three of you, you only want to spend 10 hours. Formally, we assume that there are *m* sets $S_j \subseteq \{1, \ldots, k\}$ of people, with set constraints t_j . Then, any valid solution must ensure, in addition to the previous constraints, that the combined work of all people in S_j does not exceed t_j, for all *j*.

Give an algorithm with running time polynomial in *n, m, k* for this problem, under the assumption that all the S_j are disjoint, and sketch a proof that your algorithm is correct.

Solution:

We will have one node u_h for each task h, one node v_i for each person i, and one node w_j for each constraint set S_j. In addition, there is a source s and a sink t. As before, the source connects to each node u_h with capacity r_h. Each u_h connects to each node v_i such that person i is able to do task h, with infinite capacity. If a person i is in no constraint set, node v_i connects to the sink t with capacity a_i. Otherwise, it connects to the node w_j for the constraint set S_j with $i \in S_j$, with capacity a_i. (Notice that because the constraint sets are disjoint, each person only connects to one set.) Finally, each node wj connects to the sink t with capacity t_j.

We claim that this network has an s-t flow of value at least $\sum_h r_h$ if and only if the tasks can be divided between people. For the forward direction, assume that there is such a flow. For each person i, assign him to do as many units of work on task h as the flow from u_h to v_i. First, because the flow saturates all the edges out of the source (the total capacity out of the source is only $\sum_h r_h$), and by flow conservation, each job is fully assigned to people. Because the capacity on the (unique) edge out of v_i is a_i, no person does more than a_i units of work. And because the only way to send on the flow into v_i is to the node w_j for nodes $i \in S_j$, by the capacity constraint on the edge (w_j, t), the total work done by people in S_j is at most t_j.

Conversely, if we have an assignment that has person i doing x_{ih} units of work on task h, meeting all constraints, then we send x_{ih} units of flow along the path $s\text{-}u_h\text{-}v_i\text{-}t$ (if person i is in no constraint sets), or along $s\text{-}u_h\text{-}v_i\text{-}w_j\text{-}t$, if person i is in constraint set S_j. This clearly satisfies conservation and non-negativity. The flow along each edge (s, u_h) is exactly r_h, because that is the total amount of work assigned on job h. The flow along the edge (v_i, t) or (v_i, s_j) is at most a_i, because that is the maximum amount of work assigned to person i. And the total flow along (w_j, t) is at most t_j, because each constraint was satisfied by the assignment. So we have exhibited an s-t flow of total value at least $\sum_h r_h$.

5) 20 pts

There are n trading posts along a river numbered *n, n – 1… 3, 2, 1*. At any of the posts you can rent a canoe to be returned at any other post downstream. (It is

impossible to paddle against the river since the water is moving too quickly). For each possible departure point *i* and each possible arrival point *j(< i)*, the cost of a rental from *i* to *j* is known. It is *C[i, j]*. However, it can happen that the cost of renting from *i* to *j* is higher than the total costs of a series of shorter rentals. In this case you can return the first canoe at some post *k* between *i* and *j* and continue your journey in a second (and, maybe, third, fourth . . .) canoe. There is no extra charge for changing canoes in this way. Give a dynamic programming algorithm to determine the minimum cost of a trip by canoe from each possible departure point *i* to each possible arrival point *j*. Analyze the running time of your algorithm in terms of *n*. For your dynamic programming solution, focus on computing the minimum cost of a trip from trading post *n* to trading post *1*, using up to each intermediate trading post.

Solution

Let OPT(i,j) = The optimal cost of reaching from departure point "i" to departure point "j".

Now, lets look at OPT(i,j). assume that we are at post "i". If we are not making any intermediate stops, we will directly be at "j". We can potentially make the first stop, starting at "i" to any post between "i" and "j" ("j" included, "i" not included)

This gets us to the following recurrence

OPT(i,j) = min(over j<= k < i)(C[i,k] + OPT(k,j))

The base case is OPT(i,i) = C[i,i] = 0 for all "i" from 1 to n

The iterative program will look as follows

Let OPT[i,j] be the array where you will store the optimal costs. Initialize the 2D array with the base cases

```
for (i=1;i<=n;i++)
{
        for (j=1;j<=i-i;j++)
        {
                calculate OPT(i,j) with the recurrence
        }
}
```

Now, to output the cost from the last post to the first post, will be given by OPT[n,1]

As for the running time, we are trying to fill up all OPT[i,j] for i<j. Thus, there are $O(n^2)$ entries to fill (which corresponds to the outer loops for "i" and "j") In each loop, we could potentially be doing at most k comparisons for the min operation . This is O(n) work. Therefore, the total running time is $O(n^3)$

CS570
Analysis of Algorithms
Spring 2010
Exam II

Name: _____

Student ID: _____

DEN Student YES / NO

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

2 hr exam
Close book and notes

If a description to an algorithm is required please limit your description to within 200 words, anything beyond 200 words will not be considered.

1) 20 pts
Mark the following statements as **TRUE** or **FALSE**. No need to provide any justification.

In the final residual graph constructed during the execution of the Ford–Fulkerson Algorithm, there's no path from source to sink.
TRUE

In a flow network whose edges all have capacity 1, the maximum flow value equals the maximum degree of a vertex in the flow network.
FALSE.

Memoization is the basis for a top-down alternative to the usual bottom-up approach to dynamic programming solutions.
TRUE

The time complexity of a dynamic programming solution is always lower than that of an exhaustive search for the same problem.
FALSE

If we multiply all edge capacities in a graph by 5, then the new maximum flow value is the original one multiplied by 5.
TRUE.

For any graph G with edge capacities and vertices s and t, there always exists an edge such that increasing the capacity on that edge will increase the maximum flow from s to t. (Assume that there is at least one path in the graph from $\boldsymbol{s}$ to $\boldsymbol{t}$.)
FALSE.
There might be more than one min-cut, by increasing the edge capacity of one min-cut doesn't have to impact the other one.

Let G be a weighted directed graph with exactly one source s and exactly one sink t. Let (A, B) be a maximum cut in G, that is, A and B are disjoint sets whose union is V, $s \in A, t \in B$, and the sum of the weights of all edges from A to B is the maximum for any two such sets. Now let H be the weighted directed graph obtained by adding 1 to the weight of each edge in G. Then (A, B) must still be a maximum cut in H.
FALSE
There could exist other edge which has many more cross-edges, which was not max-cut previously, to become the new max-cut.

A recursive implementation of a dynamic programming solution is often less efficient in practice than its equivalent iterative implementation.
We give points for both TRUE and FALSE answers due to the ambiguity of "often".

Ford-Fulkerson algorithm will always terminate as long as the flow network G has edges with strictly positive capacities.
FALSE
If some edges are irrational numbers, Ford-Fulkerson may never terminate.

Any problem that can be solved using dynamic programming has a polynomial time worst case time complexity with respect to its input size.
FALSE

2) 20 pts

Judge the following statement is true or false. If true, prove it. If false, give a counter-example.

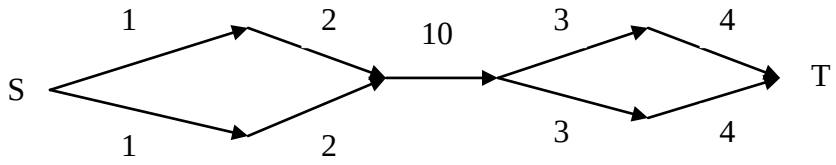
Given a directed graph, if deleting edge *e* reduces the original maximum flow more than deleting any other edge does, then edge *e* must be part of a minimum s-t cut in the original graph.

Solution:

False.

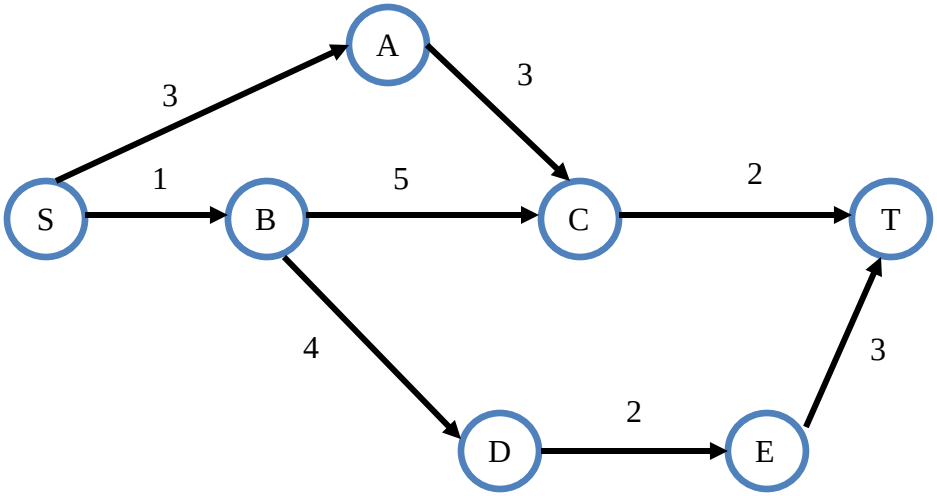
Counter-example:

Apparently deleting the edge with capacity 10 will reduce the flow by the most, while it is not part of minimum s-t cut.



Comment: some of you misunderstood the problem in different ways. Some counter-examples have multiple min s-t cuts and the edge "e" is actually in one of them. And some others failed to satisfy the precondition "more than… any other", and have some equal alternatives, which are also incorrect.

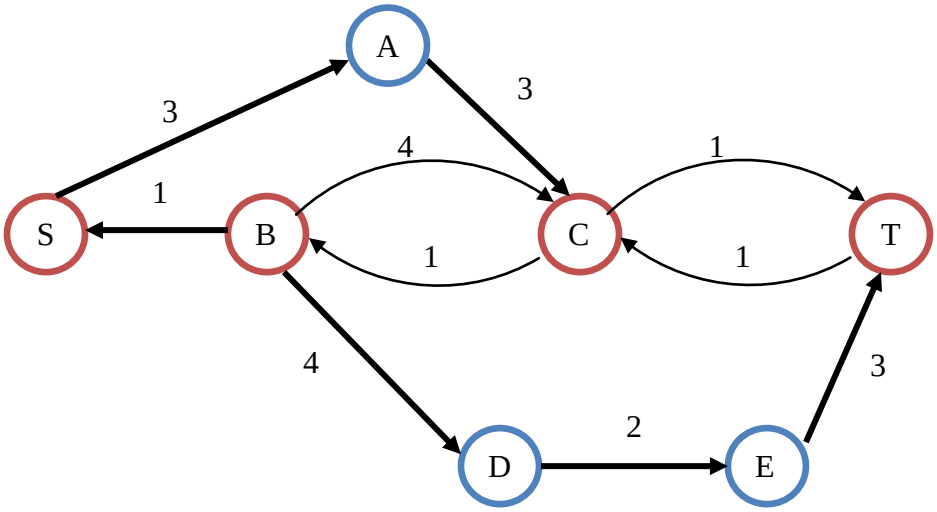
3) 20 pts



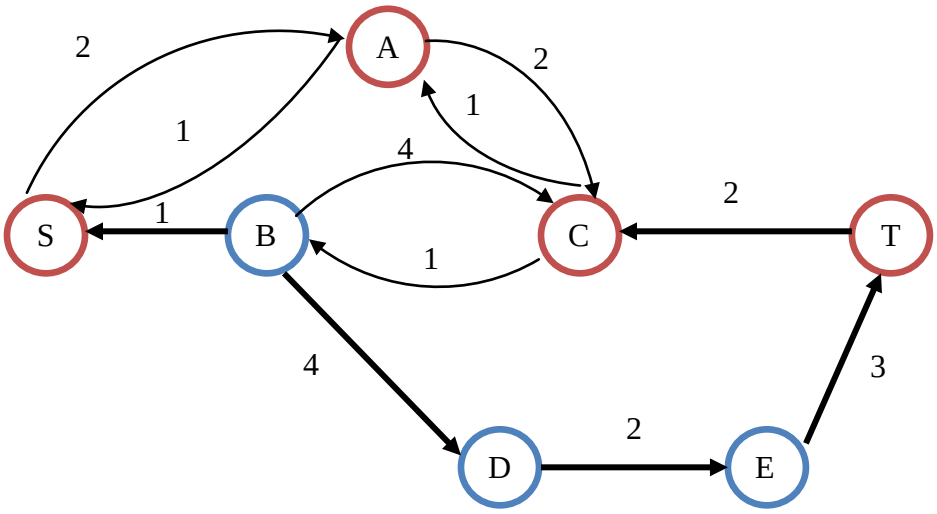
a- Show all steps involved in finding maximum flow from S to T in above flow network using the Ford-Fulkerson algorithm.
b- What is the value of the maximum flow?
c- Identify the minimum cut.

a.

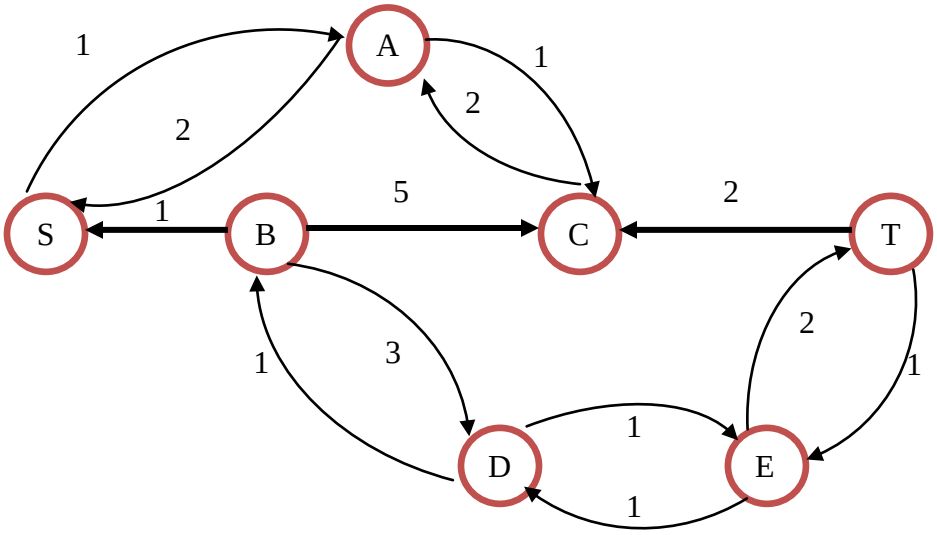
1. Augment path: [S, B, C, T], bottleneck is 1, residual graph:



2. Augment path: [S, A, C, T], bottleneck is 1, residual graph:



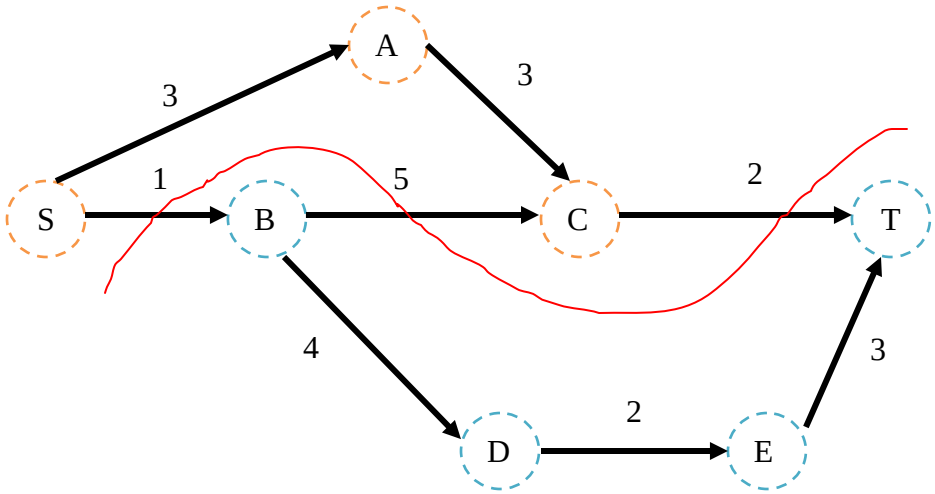
3. Augment path: [S, A, C, B, D, E, T], bottleneck is 1, residual graph:



4. No forward path from S to T, terminate.

b. The maximum flow is 3.

c. The Min-Cut is: [S, A, C] and [B, D, E, T].



4) 20 pts

The words `computer' and `commuter' are very similar, and a change of just one letter, p→m will change the first word into the second. The word `sport' can be changed into `sort' by the deletion of the `p', or equivalently, `sort' can be changed into `sport' by the insertion of `p'.

The edit distance of two strings, s1 and s2, is defined as the minimum number of point mutations required to change s1 into s2, where a point mutation is one of:

1. change a letter,
2. insert a letter or
3. delete a letter

a) Design an algorithm using dynamic programming techniques to calculate the edit distance between two strings of size at most *n*. The algorithm has to take less than or equal to $O(n^2)$ for both time and space complexity.
b) Print the edits.

The following recurrence relations define the edit distance, $d\ (s1, s2)$, of two strings s1 and s2:

1. $d\ (', ') = 0$ (for empty strings)
2. $d\ (s, ') = d\ (', s) = |s|$ (length of s)
3. $d\ (s1 + ch1, s2 + ch2) = \boldsymbol{min}(d(s1, s2) + \begin{cases} 0, if\ ch1 = ch2 \\ 1, else \end{cases}, d(s1 + ch1, s2) + 1, d(s1, s2 + ch2) + 1\ )$

The first two rules above are obviously true, so it is only necessary consider the last one. Here, neither string is the empty string, so each has a last character, $ch1$ and $ch2$ respectively. Somehow, $ch1$ and $ch2$ have to be explained in an edit of $s1 + ch1$ into $s2 + ch2$.

- If $ch1$ equals $ch2$, they can be matched for no penalty, which is 0, and the overall edit distance is $d(s1, s2)$.
- If $ch1$ differs from $ch2$, then $ch1$ could be changed into $ch2$, costing 1, giving an overall cost $d(s1, s2) + 1$.
- Another possibility is to delete $ch1$ and edit $s1$ into $s2 + ch2$, $d(s1, s2 + ch2) + 1$.
- The last possibility is to edit $s1 + ch1$ into $s2$ and then insert $ch2$, $d(s1 + ch1, s2) + 1$.

There are no other alternatives. We take the least expensive, ***min***, of these alternatives.

Examination of the relations reveals that $d(s1, s2)$ depends only on $d(s1', s2')$ where $s1'$ is shorter than $s1$, or $s2'$ is shorter than $s2$, or both. This allows the *dynamic programming* technique to be used.

A two-dimensional matrix, m[0..|s1|,0..|s2|] is used to hold the edit distance values:

```
m[i,j] = d(s1[1..i], s2[1..j])

m[0,0] = 0
m[i,0] = i,  i=1..|s1|
m[0,j] = j,  j=1..|s2|

m[i,j] = min(m[i-1,j-1] + if s1[i]=s2[j] then 0 else 1,
             m[i-1, j] + 1,
             m[i, j-1] + 1 ),  i=1..|s1|, j=1..|s2|
```

$m[,]$ can be computed *row by row*. Row $m[i,]$ depends only on row $m[i-1,]$. The time complexity of this algorithm is $O(|s1| * |s2|)$. If $s1$ and $s2$ have a similar length n, this complexity is $O(n^2)$.

b)
Once the algorithm terminates, we have generated the edit distance matrix m, we can do a trace-back for all the edits, e.g. Figure 1:

		S	a	t	u	r	d	a	y
	0	1	2	3	4	5	6	7	8
S	1	0	1	2	3	4	5	6	7
u	2	1	1	2	2	3	4	5	6
n	3	2	2	2	3	3	4	5	6
d	4	3	3	3	3	4	3	4	5
a	5	4	3	4	4	4	4	3	4
y	6	5	4	4	5	5	5	4	3

Figure 1 Edit distance matrix (from Wikipedia)

Here we are using a head recursion so that it prints the path from beginning to the end.

Print_Edits($n1$, $n2$)
if $n = 0$ then
    Output nothing
else
    Find $(i,j) \in \{(n1-1,n2),(n1,n2-1),(n1-1,n2-1)\}$ that has the minimum value from $m[i,j]$.
    Print_Edits(i,j)
    if $m[i,j] = m(n1,n2)$ then

Do nothing
else
if $(i, j) = (n1 - 1, n2 - 1)$ then
Print "$change\ s1's\ n1^{th}\ letter\ into\ s2's\ n2^{th}\ letter$"
else if $(i, j) = (n1, n2 - 1)$ then
Print "$insert\ s2's\ n2^{th}\ letter\ after\ s1's\ n1^{th}\ position$"
else
Print "$delete\ s1's\ n1^{th}\ letter$"

The trace-back process has the complexity of $O(n)$.

5) 20 pts

Given a 2-dimensional array of size $m \times n$ with only "0" or "1" in each position, starting from position [1, 1] (top left corner), every time you can only move either one cell to the right or one cell down (of course you must remain in the array). Give an *O(mn)* algorithm to calculate the number of different paths without reaching "0" in any step, starting from [1, 1] and ending in [*m*, *n*].

Solution:

We can solve this problem using dynamic programming. We denote a[i][j] as the array, num[i][j] as the number of different paths without reaching "0" in any step starting from [1, 1] and ending in [i, j]. Then the desired solution is num[m][n].
We have

$$num[i][j] = \begin{cases} 0 & \text{i=0 or j=0 or a[i][j]=0} \\ a[1][1] & \text{i=1 and j=1} \\ num[i-1][j] + num[i][j-1] & other \end{cases}$$

This is simply because we can reach cell [i, j] by coming from either [i-1, j] or [i, j-1]. When a[i][j]=1, num[i][j] is just the sum of the two num's from the two different directions.
By calculating num row by row, we can get num[m][n] at last. This is an O(mn) solution since we use O(1) time to calculate each num[i][j] and the size of array num is O(mn).

Comment: this is different from the "number of disjoint paths" problem which can be solved by being reduced to max flow problem. And some used max function instead of plus. Some other used recursion, but without memorization which would course the complexity become exponential. And some other tried to find paths and count, which is also an approach with exponential complexity.

CS570
Analysis of Algorithms
Fall 2010
Exam II

Name: _____

Student ID: _____

____Monday ____Friday ____DEN

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

2 hr exam
Close book and notes

If a description to an algorithm is required please limit your description to within 150 words, anything beyond 150 words will not be considered.

1) 20 pts
Mark the following statements as **TRUE** or **FALSE**. No need to provide any justification.

[FALSE]
If you have non integer edge capacities, then you cannot have an integer max flow.

[TRUE]
The maximum value of an s-t flow is equal to the minimum capacity of an s-t cut in the network.

[FALSE]
For any graph there always exists an edge such that increasing the capacity on that edge will increase the maximum flow from source s to sink t. (Assume that there is at least one path in the graph from source s to sink t.)

[FALSE]
If a problem can be solved using both the greedy method and dynamic programming, greedy will always give you a lower time complexity.

[FALSE]

There is a feasible circulation with demands $\{d_v\}$ if $\sum_v d_v = 0$.

[FALSE]
The best time complexity to solve the max flow problem is O(Cm) where C is the total capacity of the edges leaving the source and m is the number of edges in the network.

[TRUE]
In the Ford–Fulkerson algorithm, choice of augmenting paths can affect the number of iterations.

[FALSE]
0-1 knapsack problem can be solved using dynamic programming in polynomial time.

[TRUE]
Bellman-Ford algorithm solves the shortest path problem in graphs with negative cost edges in polynomial time.

[FALSE]
If a dynamic programming solution is set up correctly, i.e. the recurrence equation is correct and the subproblems are always smaller than the original problem, then the resulting algorithm will always find the optimal solution in polynomial time.

2) 20 pts

You are given an n-by-n grid, where each square (i, j) contains $c(i, j)$ gold coins. Assume that $c(i, j) \geq 0$ for all squares. You must start in the upper-left corner and end in the lower-right corner, and at each step you can only travel one square down or right. When you visit any square, including your starting or ending square, you may collect all of the coins on that square. Give an algorithm to find the maximum number of coins you can collect if you follow the optimal path. Analyze the complexity of your solution.

We will solve the following subproblems: let $dp[i, j]$ be the maximum number of coins that it is possible to collect while ending at (i, j).

We have the following recurrence: $dp[i, j] = c(i, j) + max(dp[i - 1, j], dp[i, j - 1])$

We also have the base case that when either $i = 0$ or $j = 0$, $dp[i, j] = c(i, j)$.

There are n^2 subproblems, and each takes $O(1)$ time to solve (because there are only two subproblems to recurse on). Thus, the running time is $O(n^2)$.

3) 20 pts

King Arthur's court had n knights. He ruled over m counties. Each knight i had a quota q_i of the number of counties he could oversee. Each county j, in turn, produced a set S_j of the knights that it would be willing to be overseen by. The King sets up Merlin the task of computing an assignment of counties to the knights so that no knight would exceed his quota, while every county j is overseen by a knight from its set S_j. Show how Merlin can employ the Max-Flow algorithm to compute the assignments. Provide proof of correctness and describe the running time of your algorithm. (You may express your running time using function $F(v, e)$, where $F(v, e)$ denotes the running time of the Max-Flow algorithm on a network with v vertices and e edges.)

We make a graph with $n+m+2$ vertices, n vertices $k_1,\ldots,k_n$ corresponding to the knights, m vertices $c_1,\ldots,c_m$ corresponding to the counties, and two special vertices s and t.

We put an edge from s to k_i with capacity q_i. We put an edge from k_i to c_j with capacity 1 if county j is willing to be ruled by knight i. We put an edge of capacity 1 from c_j to t.

We now find a maximum flow in this graph. If the flow has value m, then there is a way to assign knight to all counties. Since this flow is integral, it will pick one incoming edge for each county c_j to have flow of 1. If this edge comes from knight k_i, then county j is ruled by knight i.

The running time of this algorithm is $F(n+m+2, \sum_j |S_j|+n+m)$.

4) 20 pts

You are going on a cross country trip starting from Santa Monica, west end of I-10 freeway, ending at Jacksonville, Florida, east end of I-10 freeway. As a challenge, you restrict yourself to only drive on I-10 and only on one direction, in other words towards Jacksonville. For your trip you will be using rental cars, which you can rent and drop off at certain cities along I-10. Let's say you have n such cities on I-10, and assume cities are numbered as increasing integers from 1 to n, Santa Monica being 1 and Jacksonville being n. The cost of rental from a city i to city j (>i) is known, C[i,j]. But, it can happen that cost of renting from city i to j is higher than the total costs of a series of shorter rentals.

(A) Give a dynamic programming algorithm to determine the minimum cost of a trip from city 1 to n.

(B) Analyze running time in terms of n.

Denote OPT[i] to be the rental cost for the optimal solution to go from city i to city n for $1 \le i \le n$. Then the solution we are looking for is OPT[1] since objective is to go from city 1 to n. Therefore OPT[i] can be recursively defined as follows.

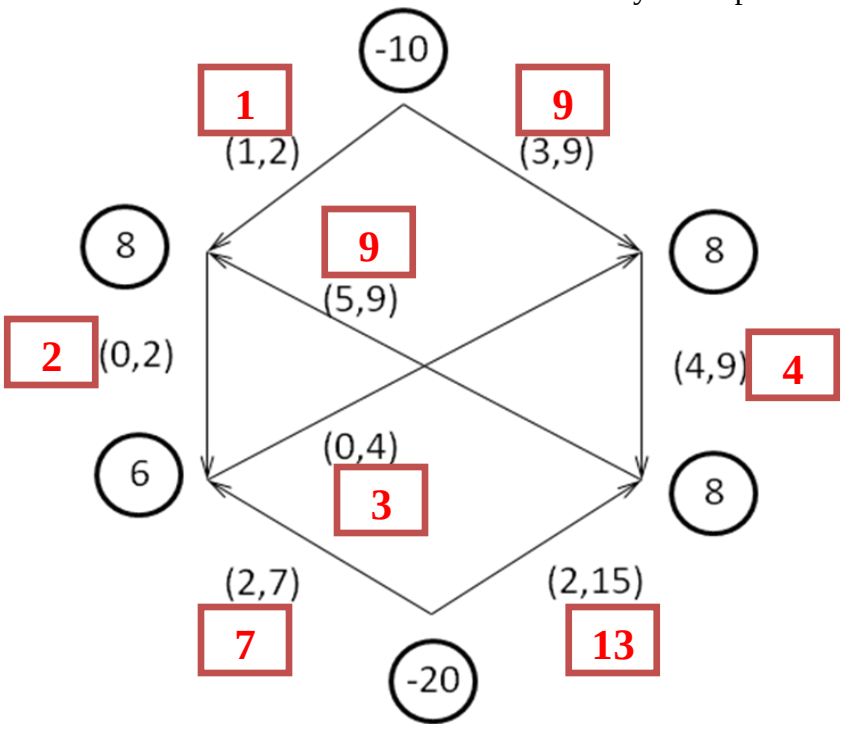
$$OPT[i] = \begin{cases} 0 & if \quad i = n \\ \min_{i \le j \le n} \left(C[i,j] + OPT[j] \right) & otherwise \end{cases}$$

Proof: The car must be rented at the starting city i and then dropped off at another city among i+1,…,n. In the recurrence we try all possibilities with j being the city where the car is next returned. Furthermore, since C[i,j] is independent from how the subproblem of going from city j,…,n is solved, we have the optimal substructure property.

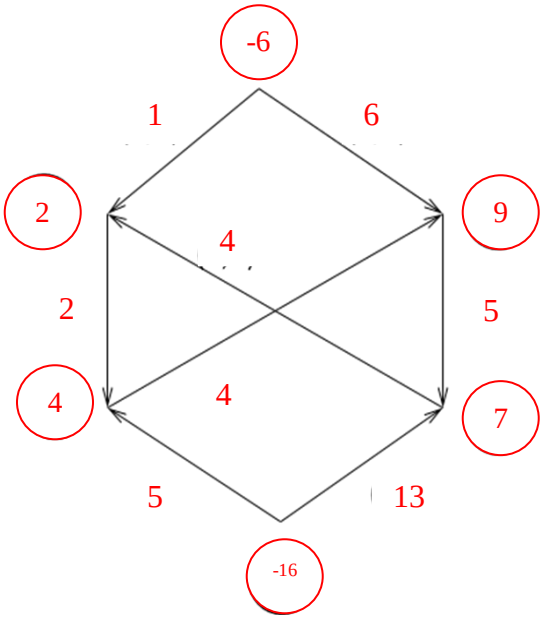
For the time complexity, there are n subproblems to be solved each of which takes linear time, O(n). These subproblems can be computed in the order OPT[n], OPT[n-1],…,OPT[1], in a linear fashion. Hence the overall time complexity is $O(n^2)$.

5) 20 pts

Solve the following feasible circulation problem. Determine if a feasible circulation exists or not. If it does, show the feasible circulation. If it does not, show the cut in the network that is the bottleneck. Show all your steps.



First we eliminate the lower bound from each edge:



Then, we attach a super-source s^* to each node with negative demand, and a super-sink t^* to each node with positive demand. The capacities of the edges attached accordingly correspond to the demand of the nodes.

We then seek a maximum s^*-t^* flow. The value of the maximum flow we can get is exactly the summation of positive demands. That is, we have a feasible circulation with the flow values inside boxes shown as above.