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 **[TRUE/FALSE]**
If all capacities in a network flow are rational numbers, then the maximum flow will be a rational number, if exist.

False [TRUE/FALSE]
The Ford-Fulkerson algorithm is based on the greedy approach.

False [TRUE/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 [TRUE/FALSE]
The Ford-Fulkerson algorithm has a polynomial time complexity with respect to the input size.

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

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

True [TRUE/FALSE]
One can actually prove the correctness of the Master Theorem.

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

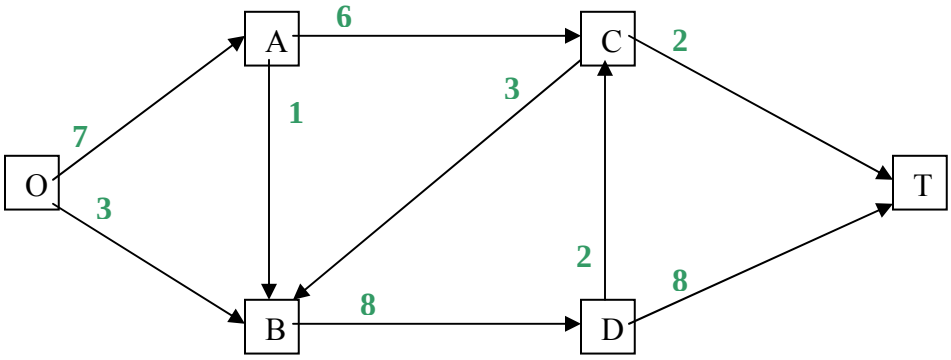
False [TRUE/FALSE]
In the Ford Fulkerson algorithm, choice of augmenting paths can affect the min cut.

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.

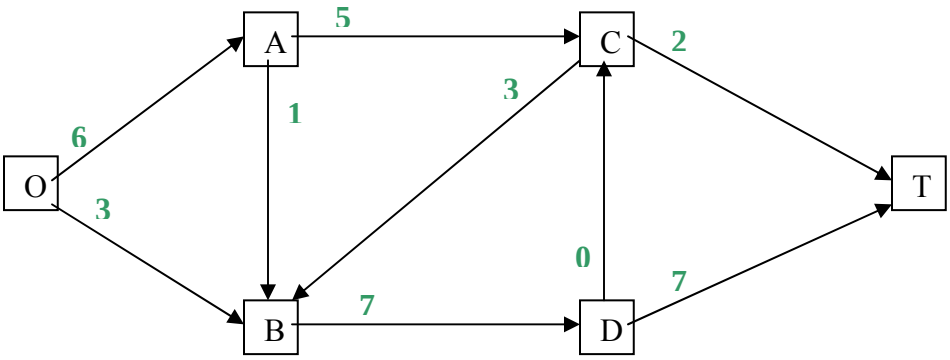
Let us assume the array is A[1…n]

```
LargestDifference(n)
{
        If n<=1, return Infinity;
        Else,    return min(LargestDifference(n-1), abs(A[n]-A[n-1]));
}
```

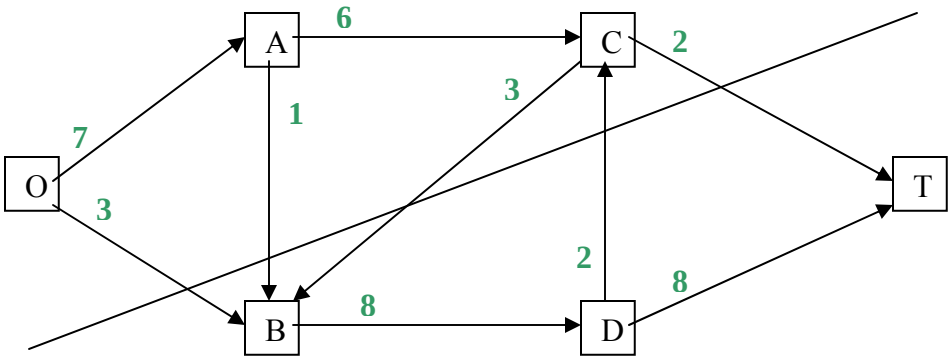
3) 15 pts
You are given the following directed network with source O and sink T.



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



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



The Capacity is 9.

4) 15 pts
Solve the following recurrences

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

By the master theorem, $T(n) = n\log^2 n$

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

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

$T(n) = 2T(n-1) – T(n-2) = 2(2T(n-2) – T(n-3)) –T(n-2) = 3T(n-2) – 2T(n-3)$
$=4T(n-3)-3T(n-4) = nT(1) – (n-1)T(0) = 3$

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.

6) 15 pts

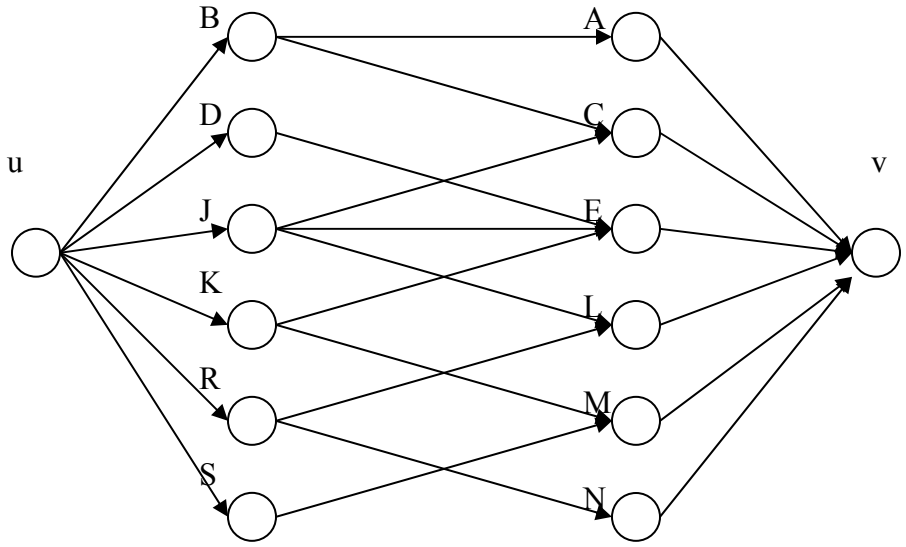
Six men and six women are at a dance. The goal of the matchmaker is to match each woman with a man in a way that maximizes the number of people who are matched with compatible mates. The table below describes the compatibility of the dancers.

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

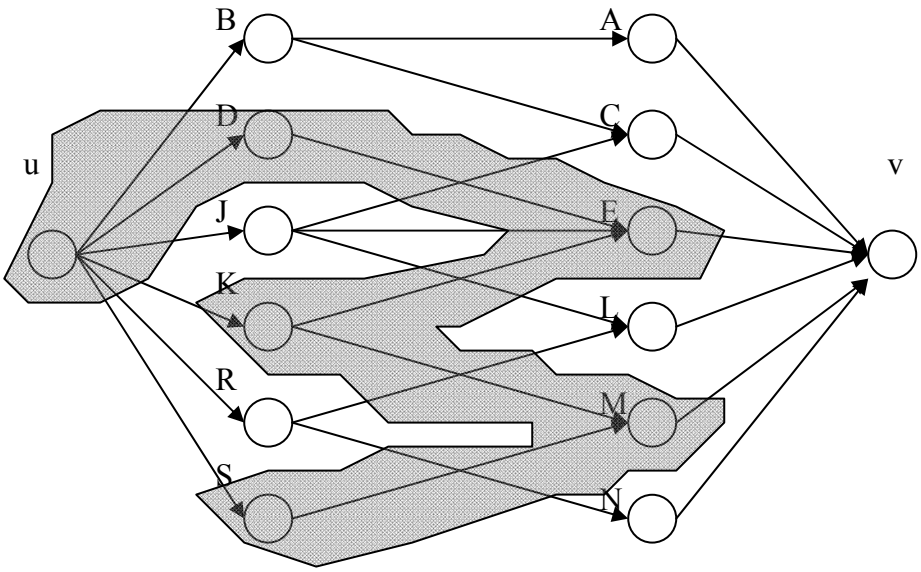
Note: C indicates compatibility.

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

All edges have capacity of one.



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



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

(B, A), (J, C), (K, E), (R, N), (S, M)

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.

This study resource was shared via CourseHero.com

https://www.coursehero.com/file/5973405/CS70-Midterm-Exam-2-Spring-2009-Solution/

[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

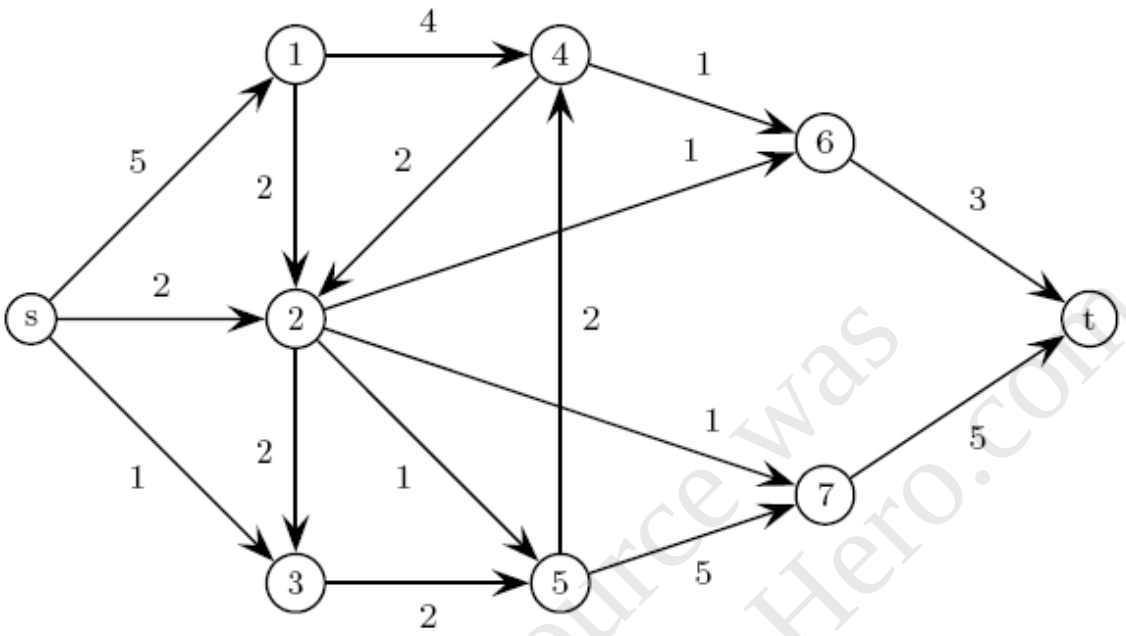
[TRUE/FALSE] FALSE
In dynamic programming you must calculate the optimal value of a subproblem twice, once during the bottom up pass and once during the top down pass.

This study resource was shared via CourseHero.com

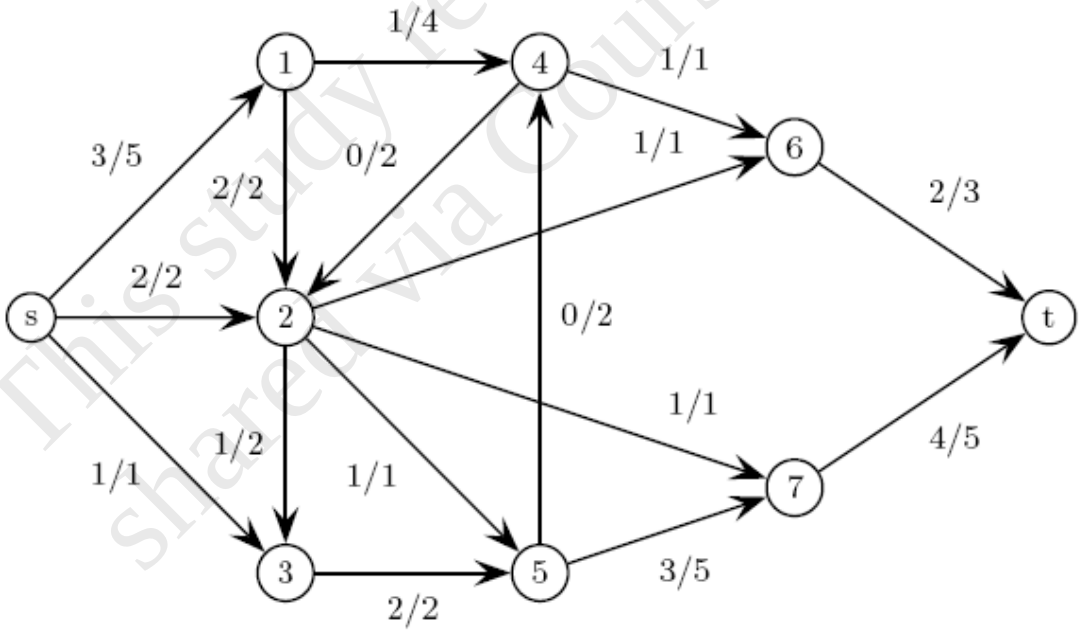
https://www.coursehero.com/file/5973405/CS70-Midterm-Exam-2-Spring-2009-Solution/

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:

This study resource was shared via CourseHero.com

https://www.coursehero.com/file/5973405/CS70-Midterm-Exam-2-Spring-2009-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).

This study resource was shared via CourseHero.com

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

https://www.coursehero.com/file/5973405/CS70-Midterm-Exam-2-Spring-2009-Solution/

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.

This study resource was shared via CourseHero.com

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*

https://www.coursehero.com/file/5973405/CS70-Midterm-Exam-2-Spring-2009-Solution/

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-u_h-v_i-t (if person i is in no constraint sets), or along s-u_h-v_i-w_j -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

https://www.coursehero.com/file/5973405/CS70-Midterm-Exam-2-Spring-2009-Solution/

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 $O(n^3)$

This study resource was shared via CourseHero.com

https://www.coursehero.com/file/5973405/CS70-Midterm-Exam-2-Spring-2009-Solution/

Additional Space

This study resource was shared via CourseHero.com

CS570
Analysis of Algorithms
Summer 2010
Exam II

Name: _____

Student ID: _____

____Check if DEN student

	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]
Given a weighted, directed graph with no negative-weight cycles, the shortest path between every pair of vertices can be determined in worst-case time $O(V^3)$.

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

[TRUE/FALSE]
Ford-Fulkerson algorithm will always terminate as long as the flow network G has edges with strictly positive capacities.

[TRUE/FALSE]
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 there is at least one path in the graph from ***s*** to ***t***)

[TRUE/FALSE]
For any network and any maximum flow on this network, there always exists an edge such that decreasing the capacity on that edge with decrease the network's max flow.

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

[TRUE/FALSE]
For edge any edge e that is part of the minimum cut in G, if we increase the capacity of that edge by any integer k>1, then that edge will no longer be part of the minimum cut.

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

[TRUE/FALSE]
One can find the value of the optimal solution AND the optimal solution to the sequence alignment problem using only O(n) memory where n is the length of the smaller sequence.

[TRUE/FALSE]
In a flow network if all edge capacities are irrational (not rational numbers) then the max flow is irrational.

2) 20 pts

You are trying to help a supermarket to distribute n types of coupons to a total of m customers. However, each type of coupon is of interest to only a subset of m customers . No coupon should be sent to a customer who is not interested in it. On the other hand, each customer can receive at most k coupons. No customer can receive duplicated coupons.

Design an efficient network flow based algorithm that given the list of customers of interest for each of the n coupons, and the upper bound k, will determine what coupons should be sent to each customer so as to maximize the total number of coupons sent. Show the running time (using parameters given above). You can assume that m>>n.

Answer:

Construct a flow graph as follows. Let s and t be the source and sink vertices respectively. For each customer, introduce a vertex. Call these b1,b2,...,bm. Likewise, for each coupon introduce a vertex. Call these c1,c2,..cn.

The edges as defined as follows. There is an edge from s to each of the coupons and it has infinite capacity. That is c(s,ci)=infty, for 1<=i<=n.

There is an edge from ci to bj if and only if the customer bj is in interested in the coupon ci. Let its capacity be 1.

There is an edge from every user to the sink t of capacity k. That is c(bj,t)=k, for 1<=j<=m.

Algorithm: Compute the(a) Max Flow of the graph. If f(ci,bj)=1, then send a ci coupon to customer bj.

The running time can be analyzed as follows:
There is a total of O(m+n) nodes and O(mn+m+n) edges in the network. To the running time is O(V^2*E)=O((m+n)^2*mn). It is reasonable to assume m>>n. Thus we can say running time is O(m^3).

Correctness Proof. We show this in two steps. First, we show that any flow translates to a valid solution to the coupon distribution problem and vice versa. Then we show that maximizing flow maximizes the number of coupons distributed.

Assume that we have an integral flow f assigned to the graph. Send f(ci,bj) number of ci coupon to customer bj.

The flow f(ci,bj) can be greater than zero only if the there is an edge between ci and bj. Thus customers are only sent coupons they are interested in. Further f(ci,bj)<=1, thus no one is sent any duplicate coupons. The flow flowing into bj corresponds to the number of coupons received by the customer bj. From the flow conservation constraint at bj, this equals the flow going out of bj which is bounded by k. Thus no customer received more than k coupons. From the conservation constraint at ci, the flow assigned to the edge (s,ci) is the total number of coupons of type ci sent. We have thus met all the

constraints of the coupon distribution problem and the flow assignments translates to a valid solution.

Similarly, we can show that any valid solution to the distribution problem can be mapped to a valid flow for the graph. The flow out of the source s is nothing but the total number of coupons distributed. Thus we indeed maximize the total number of coupons.

3) 20 pts

In the first midterm you are asked to solve the following problem:
Given a array A of n positive numbers, find indices i<j such that A[j]/A[i] is maximized.
Note that you cannot reorder the elements in the array.
Now solve the same problem using dynamic programming in O(n) time.

Answer:

Let OPT[i] be the max division with numerator A[i].
OPT[1]=1;
OPT[2]=max{OPT[1], A[2]/A[1]}
For i=3:n-1
  OPT[i+1]=max{OPT[i]*A[i+1]/a[i], 1} (*)
endFor
Find max{OPT[]} to get j.
Find min(A[1..j-1]) to get i.

Note that at stage i+1, there are 2 choices:
1. Choose A[i+1] as the numerator
2. Start a new sequence
This constructs the recurrence function (*)
Running time is O(n)+O(n)+O(n)=O(n).

Alternative solution:
Let OPT[i] be the max division up to i. Then we have
OPT[i]=max{OPT[i-1], A[i]/min[i]}
Where min[i] is the minimum element in A[1..i-1].
min[i] can be updated as min[i]=minimum{min[i-1], A[i-1]}
Use a for loop to calculate OPT[1] to OPT[n]. The max division is OPT[n].
Then trace back the array to find i and j.
The running time is O(n) because only one for loop is used and in each iteration the update of OPT[i] and min[i] takes only O(1).

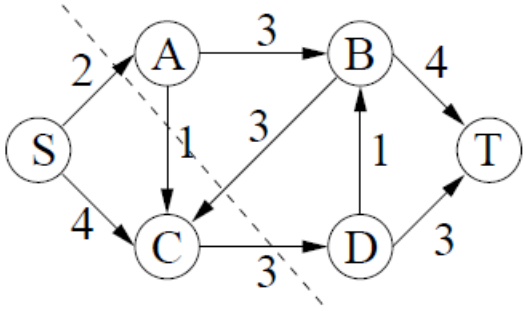
4) 20 pts

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

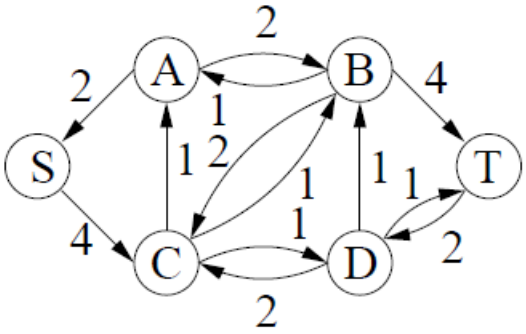
b) List all of the augmenting paths that could be chosen for the third augmentation step.

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



a.(8 points) Draw the residual network after we have updated the flow using these two augmenting paths (in the order given).

ANSWER.



b. (8 points) List all of the augmenting paths that could be chosen for the third augmentation step.

ANSWER.

S-C-A-B-T, S-C-B-T, S-C-D-B-T, and S-C-D-T.

(each 2 points, if more than 4, minus 2 points)

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

ANSWER.

5 (2 points)/ See top figure above for the minimum cut.(2 points)

5) 20 pts
In class, we saw how the Bellman-Ford algorithm finds shortest paths in a directed graph with no negative cycles. Use Bellman-Ford as your basis and provide a solution to the following problem:
Given a directed graph, determine if it contains a negative cycle. If it does, output a negative cycle.

Answer:
Let G=(V,E) be the graph and let n be the number of vertices. Let d(i,u,v) denote the cost of the shortest path from u to v using at most i edges. Let P(i,u,v) be the corresponding path.

Given a vertex u, the Bellman-Ford algorithm can be used to compute d(i,u,v) and P(i,u,v) for all vertices v and 1<=i<=n. (In the version described in class, we stopped at i=n-1, but extending it to n is straightforward). The running time is O(|V||E|).

Observe that d(n,u,u)< 0 if an only if the vertex u is a part of a negtive weight cycle(not necessarily simple).

Thus we have the following algorithm to detect and find negative cycles.

For every vertex u, compute d(n,u,u) using Bellman-Ford. If d(n,u,u) >=0 for all u, then the graph has no negative weight cycles.

If there exists a vertex u such that d(n,u,u)<0, then the graph has a negative weight cycle. Output the corresponding path P(n,u,u) as a negative cycle.

The total running time is O(|V|^2. |E|)

note: There are faster ways of doing this. Instead of computing the distances d(i,u,_) from each vertex u, we can compute the shortest paths between all pairs in one go. For instance using Floyd-Warshall algorithm (based on dynamic programming, very similar to Bellmann-ford) in O(|V|^3). There are also tricks we can use so that the Bellmann-Ford does not need to run completely every vertex.

However, these speed ups are not required and any algorithm that runs in polynomial time will get full credit.

CS570
Analysis of Algorithms
Summer 2011
Exam II

Name: _____

Student ID: _____

____Check if DEN student

	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 except for the question at the bottom of the page.

[TRUE/FALSE]
Dynamic programming is a brute force method.

[TRUE/FALSE]
Max flow in a flow network with real valued capacities can be found in polynomial time.

[TRUE/FALSE]
The top down pass in dynamic programming produces the **value** of the optimal solution whereas the bottom up pass produces the actual solution.

[TRUE/FALSE]
If a dynamic programming solution is formulated correctly, then it can be solved in polynomial time.

[TRUE/FALSE]
A flow network can have many sources and sinks but only one super source and one super sink.

[TRUE/FALSE]
It is possible to have a max flow of zero in a flow network even if all edge capacities are greater than zero.

[TRUE/FALSE]
If the capacities of edges in a flow network G are changed and the total increase of all edge capacities is k, then the value of a maximum flow of G increases by at most k.

[TRUE/FALSE]
For a flow network G = (V, E) that can carry a flow of up to $k>0$ units, if the capacity of each edge in a flow network is multiplied by m, then the resulting flow network will have a max flow of value mk.

[TRUE/FALSE] with justification
Although the original Ford-Fulkerson algorithm may fail to terminate in the case where edge capacities are arbitrary real numbers, it can be slightly modified so that it is guaranteed to terminate in the case where edge capacities are rational numbers, regardless of how the augmenting paths are chosen. This is an immediate consequence of the statement in Question 3.

Justification:
For rational numbers we can always multiple edge capacities by their common denominator and convert the flow network into a integer max flow problem and solve it using algorithm given in 3.

2) 20 pts

Say you have a graph G and a collection of sources (s1, …sk) and sinks (t1, ….tk). You want to route a path from each source si to any one of the sink nodes, but you don't want the paths to share any edges, or share any sink nodes. Present a solution to this problem and describe how you determine whether a solution exits. Also provide complexity analysis.

Answer:

1) Add a super source S and a super sink T to the graph, add edge S->Si for i = 1 to k with capacity 1. Add edge ti->T for i = 1 to k with capacity 1.
2) Assign capacity 1 for all other edges.
3) Calculate max flow from S to T and see if it is equal to k
4) If yes, there exists a solution, otherwise no solution exitsts.

Proof:

Similar to problem 14 of homework 6 part (a).

Complexity: The maximum flow is no larger than k. Thus the running time should be O(k|E|).

3) 20 pts

a) Let G = (V,E) be a flow network with source s, sink t, and an integer capacity c(u,v) on each edge (u,v) in E. For a given number K, show that an augmenting path of capacity at least K can be found in O(E) time, if such a path exists.

1) Form a graph G' from G in which each edge has capacity K or greater. This takes time O(E).
If there is an augmenting path of capacity K in E, it is also the same path in E'. Find any path from s to t, since any path from s to t in E' is an augmenting path of capacity K. Finding this path takes at most |E'| edge traversals, where |E'| <= |E|. So the total time is O(E). See p. 660 of text.

(2) The capacity of an augmenting path is the minimum capacity of any edge on the path, so look for an augmentingpath whose edges all have capacity at least K. Do a BFS search to find the path, considering only edges with residual capacity of at least K. This takes O(V+E) = O(E) time, since |V| = O(E) is a flow network.

b) The following modification of Ford-Fulkerson-Method can be used to compute a maximum flow in G.

```
MAX-FLOW-BY-SCALING(G,s,t)
1 C← max (u,v) member-of E c(u,v)
2 initialize flow f to 0
3 K ← 2  ^ ( [lg C])
4 while K ≥ 1
5 do while there exists an augmenting path p of capacity at least K
6 do augment flow f along p
7 K ← K / 2
8 return f
```

Argue that MAX-FLOW-BY-SCALING returns a maximum flow.

MAX-FLOW-BY-SCALING uses the FORD-FULKERSON method. It repeatedly augments the flow along an augmenting path until there are no augmenting paths of capacity >= 1. Since all capacities are integers, and the capacity of an augmenting path is positivie, this means that there are no augmenting paths whatsoever in the residual graph Gf. Thus, by the max-flow min-cut theorem, case 2 no augmenting paths, MAX-FLOW-BY-SCALING returns a maximum flow.

4) 20 pts

Recall that the Bellman-Ford algorithm finds the shortest distance between every vertex in G to a given vertex t in O(nm) time. Provide an $O(n^3)$ algorithm to calculate **all** shortest path lengths between vertices of a graph, i.e. your solution should find the shortest distance between every two pair of vertices in the graph.

Answer:

First, let vertices of G be numbered 1 through *N*. Then let SP(*i*, *j*, *k*) denote the shortest possible path from *i* to *j* using vertices only from the set {1, 2, ..., *k*} as intermediate pointalong the way.

There are two candidates for each of these paths: either the true shortest path only uses vertices in the set {1, ..., *k*}; or there exists some path that goes from *i* to *k* + 1, then from *k* + 1 to *j* that is better. We know that the best path from *i* to *j* that only uses vertices 1 through *k* is defined by SP(*i*, *j*, *k*), and it is clear that if there were a better path from *i* to *k* + 1 to *j*, then the length of this path would be the concatenation of the shortest path from *i* to *k* + 1 (using vertices in {1, ..., *k*}) and the shortest path from *k* + 1 to *j* (also using vertices in {1, ..., *k*}).

Thus, we have SP(i, j, 0) = w(i, j) if vertex i and j are connected by an edge of weight w(i, j). Also, SP(i, j, k+1) = min{SP(i, j, k), SP(i, k+1, k) + SP(k+1, j, k)}

We can calculate SP for all i, j, k between 1 and n. This takes O(n^3).

5) 20 pts

Using dynamic programming, find the optimal order of matrix multiplication operations in the matrix multiplication chain $M_1M_2M_3M_4M_5$

Matrix dimensions are M_1 :5 × 10, M_2 : 10 × 3, M_3 :3 × 12, M_4 : 12 × 5, and M_5 :5 × 50

	1	2	3	4	5
1	0	150	330	405	1655
2		0	360	330	2430
3			0	180	930
4				0	3000
5					0

Answer: Let OPT[i, j] be the number of operations needed to compute Ai*...Aj. We have OPT[i, i]=0 and

OPT[i, j] = min{OPT[i, k]+OPT[k+1, j]+Ri*Ck*Cj} where Ri is the number of rows in Ai, Cj is the number of columns in Aj and k goes from i to j-1.

Following this, we can generate the matrix of OPT[i, j] as above.

The optimal solution should be ((A1A2)(A3A4))A5.

CS570
Analysis of Algorithms
Summer 2013
Exam II

Name: _____

Student ID: _____

____On Campus ____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.

[TRUE/FALSE]
If a network with a source s and sink t has a unique max s-t flow, then it has a unique min s-t cut.

[TRUE/FALSE]
For flow networks such that every edge has a capacity of either 0 or 1, the Ford-Fulkerson algorithm terminates in $O(n^3)$ time, where n is the number of vertices.

[TRUE/FALSE]
Fractional knapsack problem can be solved in polynomial time.

[TRUE/FALSE]
Subset-sum problem can be solved in polynomial time

[TRUE/FALSE]
0-1 knapsack problem can be solved in polynomial time

[TRUE/FALSE]
Max flow in a flow networks can be found in polynomial time

[TRUE/FALSE]
Ford-Fulkerson algorithm runs in polynomial time

[TRUE/FALSE]
One can determine if a flow is valid or not in linear time with respect to the number of edges and nodes in the graph.

[TRUE/FALSE]
Given the value of flow v(f) for a flow network, one can determine if this value is the maximum value of flow or not in linear time.

[TRUE/FALSE]
A dynamic programming algorithm always uses some type of recurrence relation.

1. False. For example consider \{(s,x),(x,t)\} where all the edges have the same capacity.

2. True. Since the source can have at most $n-1$ adjacent vertices, the capacity of the min cut is at most $n-1$. Further the number of edges is at most $n(n-1)$.

3. True. The greedy strategy of picking as much of the item with the highest value to weight ratio as possible is optimal.

4. False: The running time is in the worst case exponential in the number of bits required to write down the problem instance.

5. False: The running time is in the worst case exponential in the number of bits required to write down the problem instance.

6. True: For example the variation of Ford-Fulkerson which at each step uses BFS to compute augmenting paths runs in time polynomial in the number of vertices irrespective of the capacities.

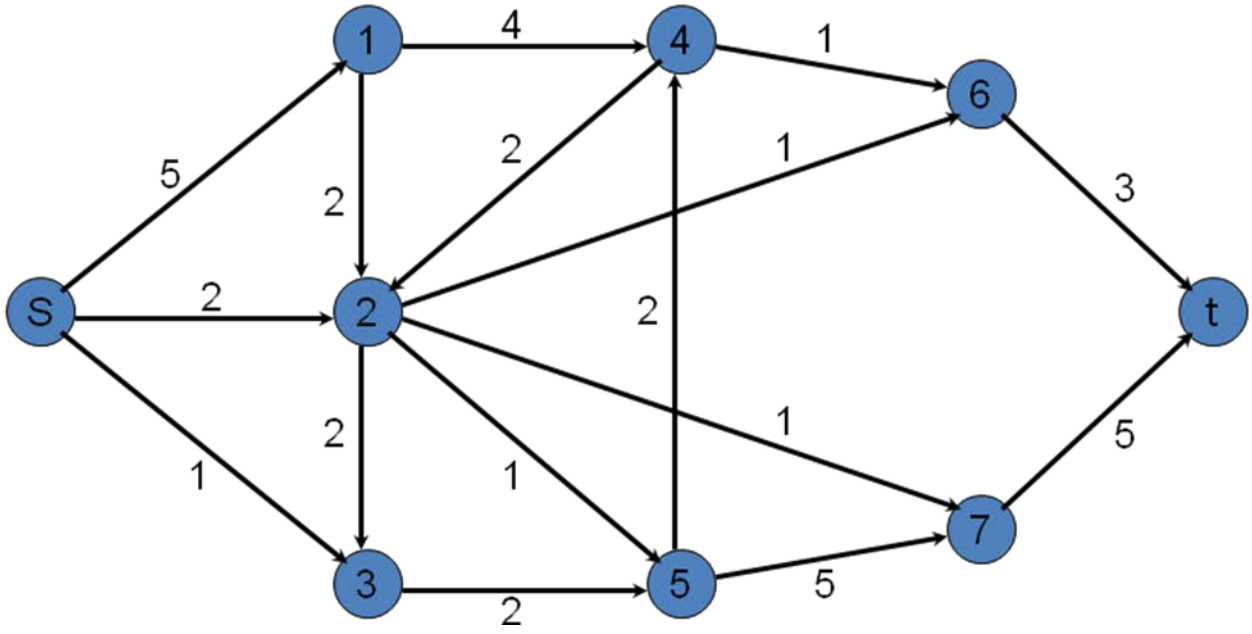
8. True: Each edge is checked once for capacity constraints and at most twice for conservation constraints.

9. True. If the flow is maximal, then for every min cut, the flow across it equals its capacity and thus every edge crossing it should be saturated.

10. True

2) 20 pts

In the familiar graph below, find the maximum s-t flow. The numbers on the edges are the capacities. Show all your steps.



Additional Space

2 One way to solve the problem is using the Ford-Fulkerson algorithm. If you choose to do so, then you have explicitly write down each step. (That is, the choice of augmenting path (and the flow sent on it) at each step and resulting residual graph). If you computed the max flow by other means, you have to prove that your flow is indeed a max flow (perhaps by demonstrating a cut whose capacity equals the value of your valid flow.)

One max flow is : f((s,1)) = 3, f((s,2)) = 2, f((s,3)) = 1, f((1,4)) =1, f((1,2)) =2,f((2,3)) =1,f((2,5)) =1,f((2,6)) =1,f((2,7)) =1,f((3,5)) =2,f((4,2)) =0,f((4,6)) =1, f((5,4)) =0,f((5,7)) =3,f((7,t)) =4,f((6,t)) =2.

There are multiple flows that are maximal (each of value 6). So your answer may be different and yet correct. There is however a unique min-cut ({s,1,2,3,4},{5,6,7,t}).

3) 20 pts

There are n cities {c1, c2,…, cn}. A city either has a bank or it does not have a bank and you know which cities have banks. Between every ordered pair (ci, cj) of cities, there is a one way road ri,j that goes from ci to cj . Assume that the city c1 does not have a bank and a bank robber has escaped from the prison and is now in c1. You are the police chief and you plan to install inspection stations on certain roads to prevent the bank robber from getting into cities with banks. If the bank robber attempts to travel on a road that has an inspection station, he will be caught. To install an inspection station on the road ri,j , it costs ci,j dollars. Design an algorithm to determine the set of roads on which to install the inspection stations that minimizes the total installation cost under the constraint that the bank robber cannot get to a city with a bank without being caught on one of the inspection stations. Prove that your algorithm is correct and analyze its running time.

We build a flow network with the cities as the vertices as follows. Set c_1 as the source and introduce a sink t. From c_i to c_j, add an edge of capacity r_{i,j}. From every city with a bank to t, add an edge of "infinite" capacity. Here, by "infinite" we mean something large enough that these edges won’t be in a min-cut: thus "infinite" could be the one plus the cost of placing inspections on all roads. There is now a one to one correspondence between valid placements of inspection stations on roads and c_1-t cuts. Thus the valid placement that minimizes the total cost is one that places the inspection stations on roads corresponding to a min c_1-t cut.

4) 20 pts

Solve the following knapsack problem, i.e. the combination of objects that give us the highest value. Knapsack capacity = 8

Item i	Value vi	Weight wi
1	15	1
2	10	5
3	9	3
4	5	4

a) Show the recurrence relation

Initial Settings: Set

$V[0, w] = 0$ for $0 \leq w \leq W$, no item

$V[i, w] = -\infty$ for $w < 0$, illegal

Recursive Step: Use

$$V[i, w] = \max(V[i-1, w], v_i + V[i-1, w-w_i])$$

for $1 \leq i \leq n$, $0 \leq w \leq W$.

Correctness of the Method for Computing $V[i, w]$

Lemma: For $1 \leq i \leq n$, $0 \leq w \leq W$,

$$V[i, w] = \max(V[i-1, w], v_i + V[i-1, w-w_i]).$$

Proof: To compute $V[i, w]$ we note that we have only two choices for file i:

Leave file i: The best we can do with files $\{1, 2, \ldots, i-1\}$ and storage limit w is $V[i-1, w]$.

Take file i (only possible if $w_i \leq w$): Then we gain v_i of computing time, but have spent w_i bytes of our storage. The best we can do with remaining files $\{1, 2, \ldots, i-1\}$ and storage $(w - w_i)$ is $V[i-1, w-w_i]$.
Totally, we get $v_i + V[i-1, w-w_i]$.

Note that if $w_i > w$, then $v_i + V[i-1, w-w_i] = -\infty$ so the lemma is correct in any case.

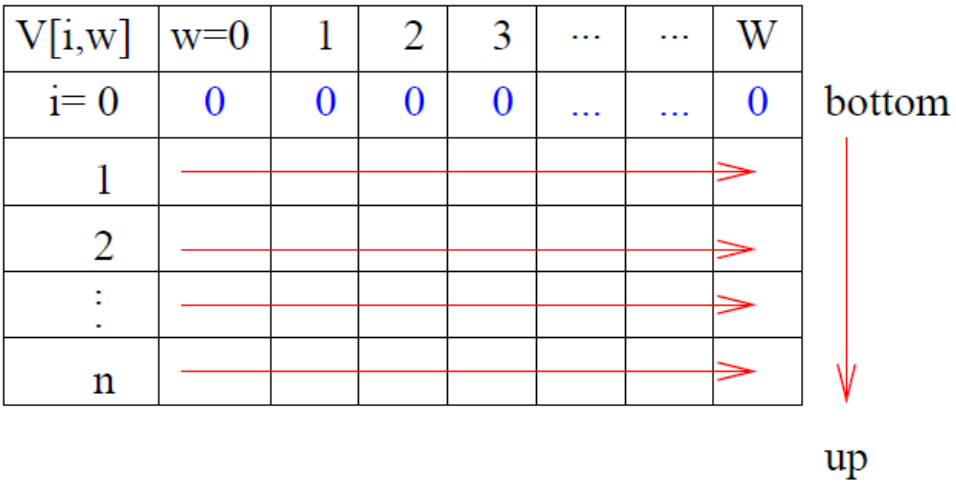
b) Numerically solve the problem.

Bottom: $V[0, w] = 0$ for all $0 \leq w \leq W$.

Bottom-up computation: Computing the table using

$$V[i, w] = \max(V[i-1, w], v_i + V[i-1, w-w_i])$$

row by row.



Example of the Bottom-up computation

Let $W = 8$ and

i	1	2	3	4
v_i	15	10	9	5
w_i	1	5	3	4

$V[i, w]$	0	1	2	3	4	5	6	7	8
$i = 0$	0	0	0	0	0	0	0	0	0
1	0	15	15	15	15	15	15	15	15
2	0	15	15	15	15	15	25	25	25
3	0	15	15	15	24	24	25	25	25
4	0	15	15	15	24	24	25	25	29

5) 20 pts

You are given integers $p_0, p_1, \ldots, p_n$ and matrices $A_1, A_2, \ldots, A_n$ where matrix A_i has dimension $p_{i-1} * p_i$

(a) Let m(i, j) denote the minimum number of scalar multiplications needed to evaluate the matrix product Ai Ai+1 … Aj. Write down a recurrence relation to compute m(i, j) for all $1 \leq i \leq j \leq n$ that runs in O(n^3) time.

This problem, like other dynamic programming problems involves determining a structure (in this case, a parenthesization). We want to break the problem into sub problems, whose solutions can be combined to solve the global problem. As is common to any DP solution, we need to find some way to break the problem into smaller sub problems, and we need to determine a recursive formulation, which represents the optimum solution to each problem in terms of solutions to the sub problems. Let us think of how we can do this.

Since matrices cannot be reordered, it makes sense to think about sequences of matrices. Let Ai..j denote the result of multiplying matrices i through j. It is easy to see that Ai..j is a pi−1 ×pj matrix. (Think about this for a second to be sure you see why.) Now, in order to determine how to perform this multiplication optimally, we need to make many decisions. What we want to do is to break the problem into problems of a similar structure.

In parenthesizing the expression, we can consider the highest level of parenthesization. At this level we are simply multiplying two matrices together. That is, for any k, $1 \leq k \leq n-1$,

$$A_{1..n} = A_{1..k} \cdot A_{k+1..n}.$$

Thus the problem of determining the optimal sequence of multiplications is broken up into two questions: how do we decide where to split the chain (what is k?) and how do we parenthesize the subchains $A_{1..k}$ and $A_{k+1..n}$?

The subchain problems can be solved recursively, by applying the same scheme.

So, let us think about the problem of determining the best value of k. At this point, you may be tempted to consider some clever ideas. For example, since we want matrices with small dimensions, pick the value of k that minimizes p_k. Although this is not a bad idea, in principle. (After all it might work. It just turns out that it doesn't in this case. This takes a bit of thinking, which you should try.) Instead, as is true in almost all dynamic programming solutions, we will do the dumbest thing of simply considering all possible choices of k,and taking the best of them. Usually trying all possible choices is bad, since it quickly leads to an exponential number of total possibilities. What saves us here is that there are only $O(n^2)$ different sequences of matrices.

(There are $C(n,2) = n(n-1)/2$ ways of choosing i and j to form $A_{i..j}$ to be precise.) Thus, we do not encounter the exponential growth.

Notice that our chain matrix multiplication problem satisfies the principle of optimality, because once we decide to break the sequence into the product $A_{1..k} \cdot A_{k+1..n}$, we should compute each subsequence optimally. That is, for the global problem to be solved optimally, the sub problems must be solved optimally as well.

Dynamic Programming Formulation: We will store the solutions to the sub problems in a table, and build the table in a bottom-up manner. For $1 \leq i \leq j \leq n$, let $m[i, j]$ denote the minimum number of multiplications needed to compute $A_{i..j}$. The optimum cost can be described by the following recursive formulation.

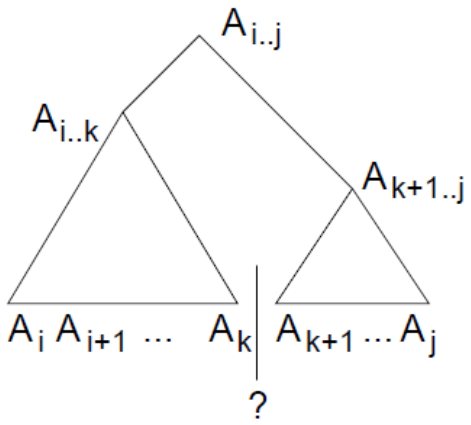
Basis: Observe that if $i = j$ then the sequence contains only one matrix, and so the cost is 0. (There is nothing to multiply.) Thus, $m[i, i] = 0$.

Step: If $i < j$, then we are asking about the product $A_{i..j}$. This can be split by considering each k, $i \leq k < j$, as $A_{i..k}$ times $A_{k+1..j}$.

The optimum times to compute $A_{i..k}$ and $A_{k+1..j}$ are, by definition, $m[i, k]$ and $m[k + 1, j]$, respectively.

We may assume that these values have been computed previously and are already stored in our array. Since $A_{i..k}$ is a $p_{i-1} \times p_k$ matrix, and $A_{k+1..j}$ is a $p_k \times p_j$ matrix, the time to multiply them is $p_{i-1}p_kp_j$. This suggests the following recursive rule for computing $m[i, j]$.

$$
\begin{aligned}
m[i,i] &= 0 \\
m[i,j] &= \min_{i \le k < j} (m[i,k] + m[k+1,j] + p_{i-1} p_k p_j) \qquad \text{for } i < j.
\end{aligned}
$$



It is not hard to convert this rule into a procedure, which is given below. The only tricky part is arranging the order in which to compute the values. In the process of computing m[i, j] we need to access values m[i, k] and m[k+1, j] for k lying between i and j. This suggests that we should organize our computation according to the number of matrices in the subsequence. Let L = j−i+1 denote the length of the subchain being multiplied. The subchains of length 1 (m[i, i]) are trivial to compute. Then we build up by computing the subchains of lengths 2, 3, . . . , n. The final answer is m[1, n]. We need to be a little careful in setting up the loops. If a subchain of length L starts at position i, then j = i + L − 1. Since we want j ≤ n, this means that i + L − 1 ≤ n, or in other words, i ≤ n−L+1. So our loop for i runs from 1 to n−L+1 (in order to keep j in bounds). The code is presented below.
The array s[i, j] is used to extract the actual sequence. The running time of the procedure is $\Theta(n^3)$. The key is that there are three nested loops, and each can iterate at most n times.

Extracting the final Sequence: Extracting the actual multiplication sequence is a fairly easy extension. The basic idea is to leave a split marker indicating what the best split is, that is, the value of k that leads to the minimum value of m[i, j].

Chain Matrix Multiplication

```
Matrix-Chain(array p[1..n]) {
    array s[1..n-1,2..n]
    for i = 1 to n do m[i,i] = 0;              // initialize
    for L = 2 to n do {                        // L = length of subchain
        for i = 1 to n-L+1 do {
            j = i + L - 1;
            m[i,j] = INFINITY;
            for k = i to j-1 do {              // check all splits
                q = m[i, k] + m[k+1, j] + p[i-1]*p[k]*p[j]
                if (q < m[i, j]) {
                    m[i,j] = q;
                    s[i,j] = k;
                }
            }
        }
    }
    return m[1,n] (final cost) and s (splitting markers);
}
```

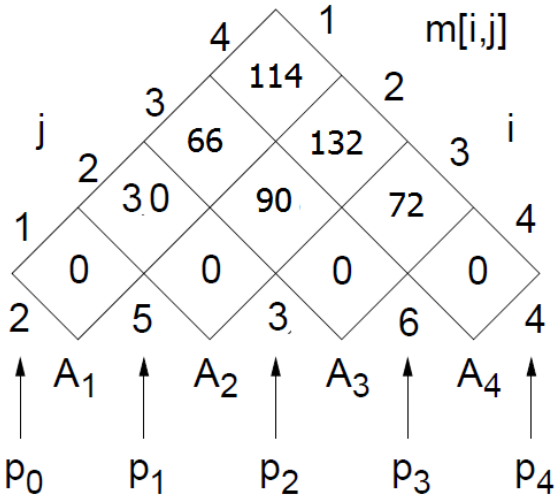
We can maintain a parallel array s[i, j] in which we will store the value of k providing the optimal split. For example, suppose that s[i, j] = k. This tells us that the best way to multiply the subchain Ai..j is to first multiply the subchain Ai..k and then multiply the subchain Ak+1..j , and finally multiply these together. Intuitively, s[i, j] tells us what multiplication to perform last. Note that we only need to store s[i, j] when we have at least two matrices, that is, if j > i.
The actual multiplication algorithm uses the s[i, j] value to determine how to split the current sequence. Assume that the matrices are stored in an array of matrices A[1..n], and that s[i, j] is global to this recursive procedure.
The recursive procedure Mult does this computation and below returns a matrix.

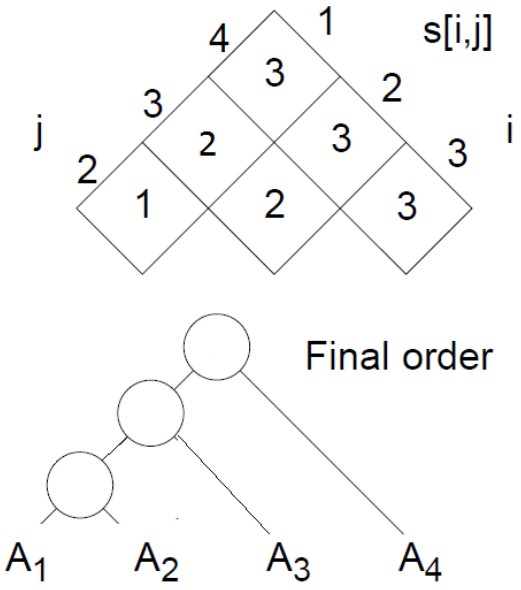
Extracting Optimum Sequence

```
Mult(i, j) {
    if (i == j)                                // basis case
        return A[i];
    else {
        k = s[i,j]
        X = Mult(i, k)                         // X = A[i]...A[k]
        Y = Mult(k+1, j)                       // Y = A[k+1]...A[j]
        return X*Y;                            // multiply matrices X and Y
    }
}
```

(b) Use this algorithm to compute m(1,4) for p0=2, p1=5, p2=3, p3=6, p4=4

The initial set of dimensions are <2, 5, 3, 6, 4> meaning that we are multiplying A1 (2 × 5) times A2 (5 × 3) times A3 (3 × 6) times A4 (6 × 4). The optimal sequence is (((A1A2)A3)A4) which results in 114 multiplications.


m[i,j]
j
i
1
2
3
4
114
66
132
30
90
72
0
0
0
0
2
5
3
6
4
A1
A2
A3
A4
p0
p1
p2
p3
p4


s[i,j]
j
i
1
2
3
4
3
2
3
2
3
1
2
3
Final order
A1
A2
A3
A4

CS570
Analysis of Algorithms
Fall 2014
Exam II

Name: _____

Student ID: _____

Email: _____

Wednesday Evening Section

	Maximum	Received
Problem 1	20	
Problem 2	16	
Problem 3	16	
Problem 4	20	
Problem 5	16	
Problem 6	12	
Total	100	

Instructions:

1. This is a 2-hr exam. Closed book and notes
2. If a description to an algorithm or a proof is required please limit your description or proof to within 150 words, preferably not exceeding the space allotted for that question.
3. No space other than the pages in the exam booklet will be scanned for grading.

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

[TRUE/FALSE]
If an iteration of the Ford-Fulkerson algorithm on a network places flow 1 through an edge (u, v), then in every later iteration, the flow through (u, v) is at least 1.

[TRUE/FALSE]
For the recursion T(n) = 4T(n/3) + n, the size of each subproblem at depth k of the recursion tree is $n/3^{k-1}$.

[TRUE/FALSE]
For any flow network *G* and any maximum flow on *G*, there is always an edge *e* such that increasing the capacity of *e* increases the maximum flow of the network.

[TRUE/FALSE]
The asymptotic bound for the recurrence T(n) = 3T(n/9) + n is given by $\Theta(n^{1/2} \log n)$.

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

[TRUE/FALSE]
A pseudo-polynomial time algorithm is always slower than a polynomial time algorithm.

[TRUE/FALSE]
The sequence alignment algorithm can be used to find the longest common subsequence between two given sequences.

[TRUE/FALSE]
If a dynamic programming solution is set up correctly, i.e. the recurrence equation is correct and each unique sub-problem is solved only once (memoization), then the resulting algorithm will always find the optimal solution in polynomial time.

[TRUE/FALSE]
For a divide and conquer algorithm, it is possible that the divide step takes longer to do than the combine step.

[TRUE/FALSE]
Maximum value of an $s-t$ flow could be less than the capacity of a given $s-t$ cut in a flow network.

2) 16 pts

Recall the Bellman-Ford algorithm described in class where we computed the shortest distance from all points in the graph to t. And recall that we were able to find all shortest distance to t with only O(n) memory.

How would you extend the algorithm to compute both the shortest distance and to find the actual shortest paths from all points to t with only O(n) memory?

3) 16 pts

During their studies, 7 friends (Alice, Bob, Carl, Dan, Emily, Frank, and Geoffrey) live together in a house. They agree that each of them has to cook dinner on exactly one day of the week. However, assigning the days turns out to be a bit tricky because each of the 7 students is unavailable on some of the days. Specifically, they are unavailable on the following days (1 = Monday, 2 = Tuesday, …, 7 = Sunday):

- Alice: 2, 3, 4, 5
- Bob: 1, 2, 4, 7
- Carl: 3, 4, 6, 7
- Dan: 1, 2, 3, 5, 6
- Emily: 1, 3, 4, 5, 7
- Frank: 1, 2, 3, 5, 6
- Geoffrey: 1, 2, 5, 6

Transform the above problem into a maximum flow problem and draw the resulting flow network. If a solution exists, the flow network should indicate who will cook on each day; otherwise it must show that a feasible solution does not exist

4) 20 pts

Suppose that there are n asteroids that are headed for earth. Asteroid i will hit the earth in time t_i and cause damage d_i unless it is shattered before hitting the earth, by a laser beam of energy e_i. Engineers at NASA have designed a powerful laser weapon for this purpose. However, the laser weapon needs to charge for a duration ce before firing a beam of energy e. Can you design a dynamic programming based pseudo-polynomial time algorithm to decide on a firing schedule for the laser beam to minimize the damage to earth? Assume that the laser is initially uncharged and the quantities c, t_i, d_i, e_i are all positive integers. Analyze the running time of your algorithm. You should include a brief description/derivation of your recurrence relation. Description of recurrence relation = 8pts, Algorithm = 6pts, Run Time = 6pts

5) 16 pts

Consider a two-dimensional array A[1:n,1:n] of integers. In the array each row is sorted in ascending order and each column is also sorted in ascending order. Our goal is to determine if a given value x exists in the array.

a. One way to do this is to call binary search on each row (alternately, on each column). What is the running time of this approach? [2 pts]

b. Design another divide-and-conquer algorithm to solve this problem, and state the runtime of your algorithm. Your algorithm should take strictly less than $O(n^2)$ time to run, and should make use of the fact that each row and each column is in sorted order (i.e., don't just call binary search on each row or column). State the run-time complexity of your solution.

6) 12 pts

Consider a divide-and-conquer algorithm that splits the problem of size n into 4 sub-problems of size n/2. Assume that the divide step takes O(n^2) to run and the combine step takes O(n^2 log n) to run on problem of size n. Use any method that you know of to come up with an upper bound (as tight as possible) on the cost of this algorithm.

Additional Space

CS570
Analysis of Algorithms
Fall 2015
Exam II

Name: _____

Student ID: _____

Email Address:_____

_____Check if DEN Student

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

Instructions:

1. This is a 2-hr exam. Closed book and notes
2. If a description to an algorithm or a proof is required please limit your description or proof to within 150 words, preferably not exceeding the space allotted for that question.
3. No space other than the pages in the exam booklet will be scanned for grading.
4. If you require an additional page for a question, you can use the extra page provided within this booklet. However please indicate clearly that you are continuing the solution on the additional page.

1) 20 pts

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

[TRUE]

The Ford-Fulkerson Algorithm finds a maximum flow of a unit-capacity flow network with *n* vertices and *m* edges in time *O(mn).*

[/FALSE]

In a flow network, if maximum flow is unique then min cut must also be unique.

[/FALSE]

In a flow network, if min cut is unique then maximum flow must also be unique.

[/FALSE]

In dynamic programming you must calculate the optimal value of a sub-problem twice, once during the bottom up pass and once during the top down pass.

[TRUE/]

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

[TRUE/]

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

[/FALSE]

An optimal solution to a 0/1 knapsack problem will always contain the object *i* with the greatest value-to-cost ratio V_i/C_i

[TRUE/]

The Ford-Fulkerson algorithm is based on greedy.

[TRUE/]

A flow network with unique edge capacities may have several min cuts.

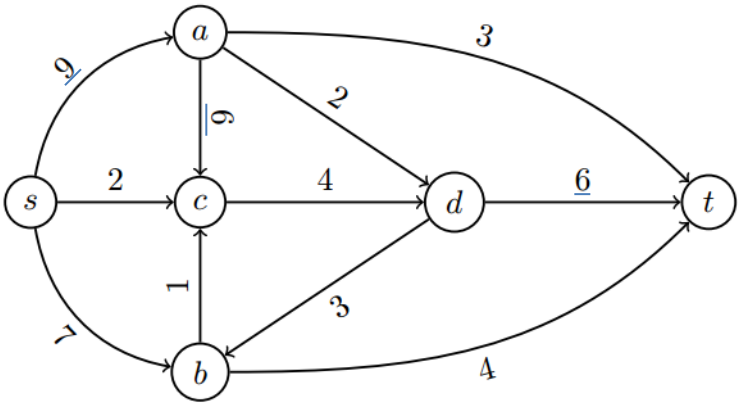
[/FALSE]

Complexity of a dynamic programming algorithm is equal to the number of unique sub-problems in the solution space.

2) 20 pts

Consider the flow network G below with source s and sink t. The edge capacities are the numbers given near each edge.

(a) Find a maximum flow in this network using the Ford Fulkerson algorithm. Show all steps of augmentation. Once you have found a maximum flow, draw a copy of the original network G and clearly indicate the flow on each edge of G in your maximum flow. (12 pts)



(b) Find a minimum s-t cut in the network, i.e. name the two (nonempty) sets of vertices that define a minimum cut. (8 pts)

Solution:

Starting from 0 flow (i.e. flow of 0 on each edge), we find augmenting paths.

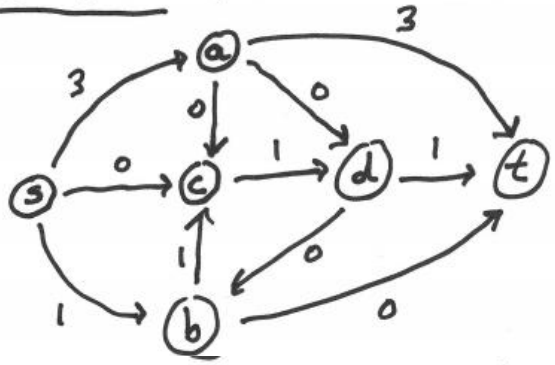
We can find some that are non-overlapping and augment along all simultaneously.

For example, we can take

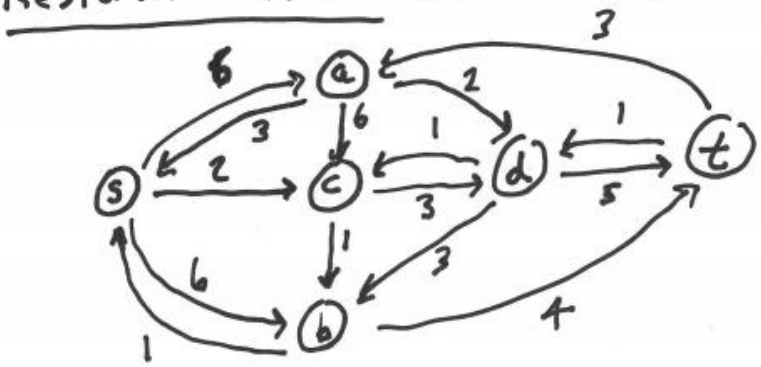
s, a, t augment 3 units

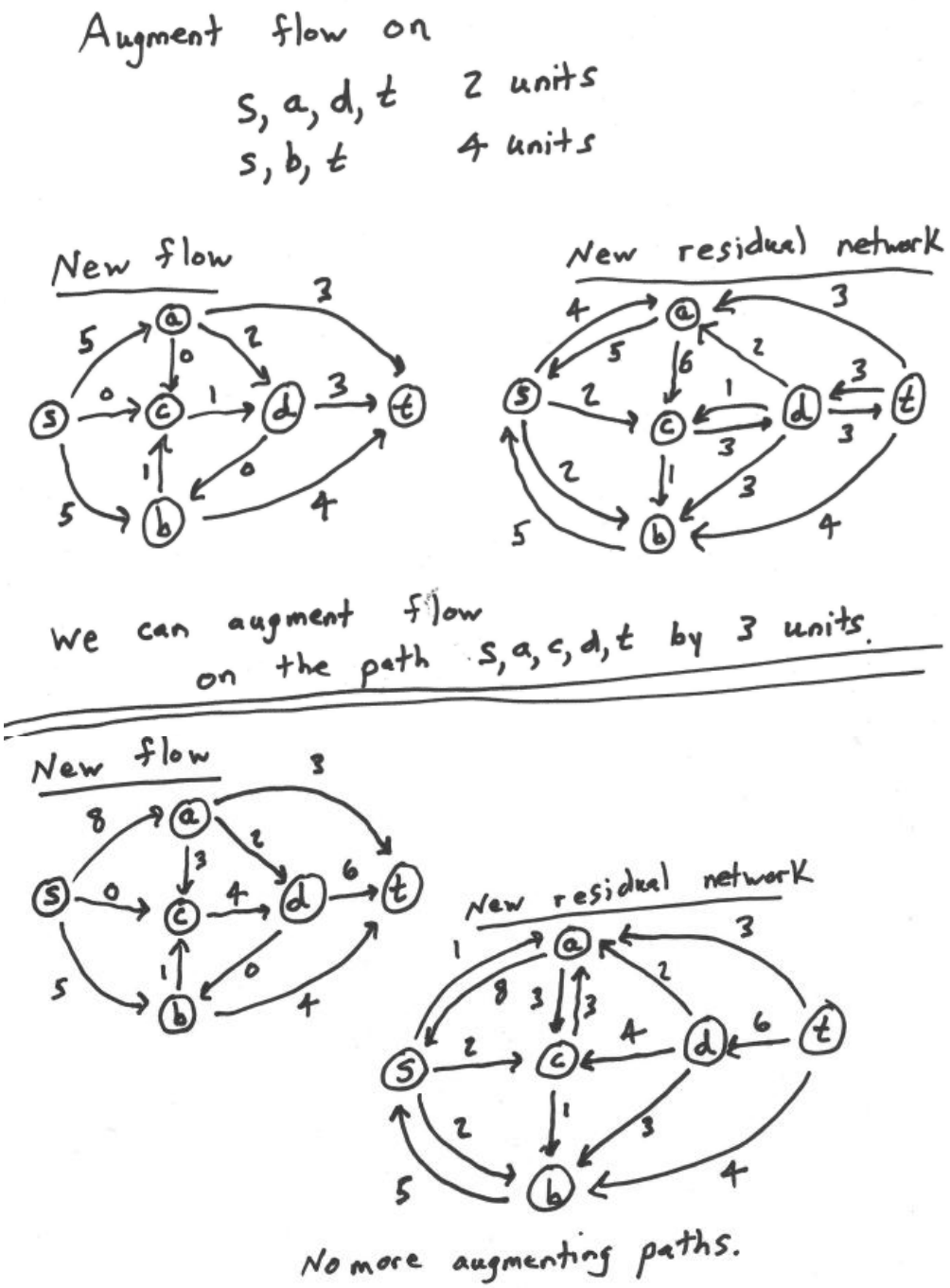
s, b, c, d, t augment 1 unit

New flow (flows on edges)



Residual network (residual capacities on edges)





Thus, the max flow has value 13. Note that the flow shown is not unique, i.e. there are other possible maximum flows in the network, but they will all have the same value of 13.

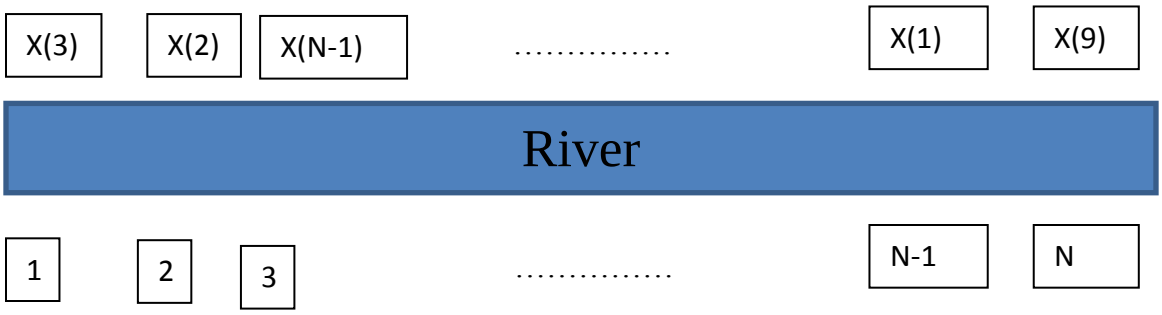
(b)

As in problem (1), start from *s* and find which vertices can be reached in the final residual network for one set of the cut, and the remaining vertices form the other set. So one minimum cut is *{s, a, b, c} {d, t}*. The (forwards) edges across this cut are (*a*, *d*), (*a*, *t*), (*c*,

d), and (*b*, *t*). The capacities of those edges are, respectively, 2, 3, 4, and 4, the sum of which is 13 (which it should be according to the Max Flow/Min Cut Theorem). Note, also in this case that the minimum cut isn't unique. For example, the cut consisting of the two sets *{s, a, b, c, d}* and *{t}* also has capacity 13.

3) 20 pts

There is a river going from east to west, on the south bank of the river there are a set of cities 1…N as shown in the figure below. On the northern bank there are also N cities. Each city on the south bank has a unique sister city on the north bank of the river. We use X(i) to denote the sister city corresponding to city i. Suppose you are assigned a task to build bridges to connect the southern cities with their sister cities such that as many bridges as possible are built without any bridges crossing each other. Solve this problem using dynamic programming.



a) Recurrence formula (7 pts)

Solution 1:

If we build a bridge between city i and X(i), then for all cities j<i, we try to most as many bridges as possible such that X(j)<X(i) to avoid crossing.

Define OPT(i) as maximum number of bridges built for southern cities from 1 to i, while guaranteeing that there is a bridge is built between city i and X(i).

Base case: OPT(1) = 1

Recursive relation (for i>1):
$OPT(i) = \max_{\{k:\ 1<=k<i,\ X(k)<X(i)\}} \{OPT(k)\} + 1$

$Max\_bridge\_num = \max_{\{i\}} \{OPT(i)\}$

Solution 2:
Define OPT(n,m) as the maximum number of no crossing bridges that can be built between southern cities 1,,,,n and northern cities 1,…, m.

Base case: OPT(1,m) = 0, if m<X(1), OPT(1,m)=1, if m>=X(1);
OPT(n,1) = 0, if n<X^{-1}(1), OPT(n,1) = 0, if n>=X^{-1}(1)

Recursive relation:

OPT(n,m) = OPT(n-1, m-1)+1, if m = X(n)
OPT(n,m) = max{OPT(n-1,m), OPT(n,m-1)}, if m != X(n)

a) Provide the algorithm to find the value of the optimal solution using the above recurrence formula (6 pts)

Solution 1:

```
For i = 1  to N
        Compute OPT(i) according to a)
End

Find the maximum among OPT(1), OPT(2), …..OPT(N)
```

Solution 2:

```
OPT(1) = 1;
For i = 2 to N
        L(i) = 0;
        For k = 1:i-1
                If( (X(k)<X(i)))
                        L(i) = k;
                End
        End
        OPT(i) = max{ OPT(L(i)) + 1, OPT(i-1) }
End
```

Solution 2:

```
Compute Base case;
For n = 2:N
        For m = 2: N
                Compute OPT according to recursive relation.
        End
End
Returen OPT(N,N)
```

b) Provide the algorithm to list the actual bridges corresponding to the value found in part b (7 pts)

Solution 1:

Method 1:
Denote i* = argmax_i {OPT(i)}

```
Bridge_list = i*;

If(i*==1)
        Return Bridge_list;
End

For j = i* to 1

        For k = 1:j-1
                If( (X(k)<X(j)) &&(OPT(j)==OPT(k)+1) )
                        Bridge_list = [k, Bridge_list];
                End
        end
 End
 Return Bridge_list;
```

Method 2:
If you keep recording the corresponding k which achieves the maximum of OPT(i) in part b), denote it as p(i); if no k achieves the maximum, i.e., OPT(k)=0, then p(i) = i.

Denote i* = argmax_i {OPT(i)}

```
Bridge_list = i*;

If (i*==1)
        Return Bridge_list
End

j= i*;
While(p(j)!=j)
        Bridge_list = [p(j), Briedge_list];
End
Return Bridge_list.
```

Complexity: O(n^2)

Solution 2:

Compute Build_bridge (N,N)

Bridge = [];

```
Build_bridge (n,m)
{
        If n ==1
                If m>=X(1)
                        Return [Bridge, (1,X(1))];
                Else
                        Return Bridge;
        End

        If m = 1
                If m == X(n)
                        Return [Bridge, (n,1)]
                Else
                        Return Bridge
        End

        If OPT(n,m) == OPT(n-1,m-1) + 1
                Bridge = [Bridge,(n,m)];
                Buildbridge(n-1,m-1)
        Elseif OPT(n,m) == OPT(n-1,m)
                Buildbridge(n-1,m);
        Else
                Brildbridge(n,m-1);
        End

}
```

4) 20 pts

There are n reservoirs and m cities. Due to the drought you have to bring water from the reservoirs to the cities through a network of pipes. Each city *i* has a request for *Ci* gallons of water every day. To make matters worse, engineers have detected a leaking pipe in the water network. If the leaking pipe is used, they will lose L gallons of water per day through that pipe regardless of how much water flows through it (i.e., as long as the pipe is used, it leaks L gallon, we cannot control the leak by trying to push less water through that pipe). The other option is to shut down the leaking pipe at both ends, but that might reduce the capacity of their network. The water network is represented by a graph. Each edge represents the capacity of the pipe in gallons per day. Provide algorithms to determine if it is possible to:

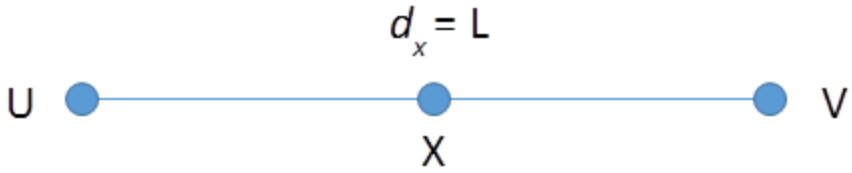
a) Meet all demands for water at all cities without using the leaking pipe (7 pts)

Assume the original graph is G=(V,E). Build a new graph with a supersource s and a supersink t.
Connect the supersource to each reservoir with an edge of infinity capacity. Connect city i to supersink with capacity C_i
Remove the edge corresponding to the leaking pipe. Run a max flow from s to t. If the flow over the edge from city i to supersink t equals to C_i for all city i. Then the demand of all city can be met.

b) Meet all demands for water at all cities by using the leaking pipe (6 pts)

The easiest way to model the second subproblem would be to:

- split the edge representing that segment of the pipe with a leak into two edges
- The point connecting these two edges will represent the location of the leak
- Each of the two segments will have the same capacity as the original leaking pipe
- We place a demand of L units at the location of the leak
- The rest is similar to how we dealt with demand for flow in part, i.e. connet all nodes with demand including X to supersink, etc. and solve for max flow, etc.



c) Meet all demands for water at all cities after fixing the leaking pipe (6 pts)

Pretty similar with solution of a), this time we don't have to remove the leaking pipe. Just use it as a usual edge. Then use the same algorithm in solution a)

5) 20 pts

Assume you want to ski down the mountain. You want the total length of your run to be as long as possible, but you can only go down, i.e. you can only ski from a higher position to a lower position. The height of the mountain is represented by an *n* X *n* matrix A. A[i][j] is the height of the mountain at position (i,j). At position (i,j), you can potentially ski to four adjacent positions (i-1,j) (i,j-1), (i,j+1), and (i+1,j) (only if the adjacent position is lower than current position). Movements in any of the four directions will add 1 unit to the length of your run. Provide a dynamic programming solution to find the longest possible downhill ski path starting at any location within the given *n* by *n* grid.

Define OPT(i,j) be the longest path start from position (i,j).

The recurrence is

If there are positions (i',j') adjacent position of (i,j) with height less than A[i][j]
OPT(i,j) = max_{(i',j') is adjacent to (i,j), A[i'][j'] < A[i][j]} OPT(i',j') + 1,
If not, OPT(i,j) = 0 (boundary condition).

==========

We solve it by memorization:
pseudocode:

```
search(i,j):
if OPT(i,j) >= 0
  return OPT(i,j)
else
  max = 0
  if A[i-1][j] < A[i][j] and search(i-1, j) +1> max
    max =search(i-1, j) +1
  if A[i+1][j] < A[i][j] and search(i+1, j) +1> max
    max =search(i+1, j) +1
  if A[i][j-1] < A[i][j] and search(i, j-1) +1> max
    max =search(i, j-1) +1
  if A[i][j+1] < A[i][j] and search(i, j+1) +1> max
    max =search(i, j+1) +1
  OPT(i,j) = max
  return OPT(i,j)

find_path(i,j):
if OPT(i,j) == 0:
  return [(i,j)]
else
  if OPT(i,j) = OPT(i-1,j) + 1
     return [(i,j), find_path(i-1,j)]
```

```
  if OPT(i,j) = OPT(i+1,j) + 1
     return [(i,j), find_path(i+1,j)]
  if OPT(i,j) = OPT(i,j-1) + 1
     return [(i,j), find_path(i,j-1)]
  if OPT(i,j) = OPT(i,j+1) + 1
     return [(i,j), find_path(i,j+1)]

OPT(i,j) = -1 for all  1<= i , j <=n
result = 0
for i = 1 to n
 for j = 1 to n
  if search(i,j) > max
    max = search(i,j)
    start = (i,j)

return find_path(start)
```

==========

The time complexity is O(n^2) because we have n^2 subproblems and solve each of them will require O(1) time.

==========

You can also solve the problem iteratively as follows:

```
update(i,j):

  max = 0

  if A[i-1][j] < A[i][j] and OPT(i-1, j) +1> max

    max =OPT(i-1, j) +1

  if A[i+1][j] < A[i][j] and OPT(i+1, j) +1> max

    max =OPT(i+1, j) +1

  if A[i][j-1] < A[i][j] and OPT(i, j-1) +1> max

    max =OPT(i, j-1) +1

  if A[i][j+1] < A[i][j] and OPT(i, j+1) +1> max

    max =OPT(i, j+1) +1
```

return max

for (i,j) in increasing order of A[i][j]
OPT(i,j) = update(i,j)

The path finding procedure is the same as the recursive one.

The time complexity is O(n^2log n) because of sorting.

CS570
Analysis of Algorithms
Spring 2015
Exam II

Name: _____

Student ID: _____

Email Address:_____

_____Check if DEN Student

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

Instructions:

1. This is a 2-hr exam. Closed book and notes
2. If a description to an algorithm or a proof is required please limit your description or proof to within 150 words, preferably not exceeding the space allotted for that question.
3. No space other than the pages in the exam booklet will be scanned for grading.
4. If you require an additional page for a question, you can use the extra page provided within this booklet. However please indicate clearly that you are continuing the solution on the additional page.

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

[TRUE/FALSE]
A flow network with unique edge capacities has a unique min cut.

[TRUE/FALSE]
If a problem can be solved by dynamic programming, then it can always be solved by exhaustive search (Brute Force).

[TRUE/FALSE]
A divide and conquer algorithm acting on an input size of n can have a lower bound less than Θ(n log n).

[TRUE/FALSE]
If a flow in a network has a cycle, this flow is not a valid flow.

[TRUE/FALSE]
In the divide and conquer algorithm to compute the closest pair among a given set of points on the plane, if the sorted order of the points on both X and Y axis are given as an added input, then the running time of the algorithm improves to O(n).

[TRUE/FALSE]
In a flow network, an edge that goes straight from *s* to *t* is always saturated when maximum *s* - *t* flow is reached.

[TRUE/FALSE]
The Bellman-Ford algorithm always fails to find the shortest path between two nodes in a graph if there is a negative cycle present in the graph.

[TRUE/FALSE]
If f is a max s-t flow of a flow network G with source s and sink t, then the capacity of the min s-t cut in the residual graph G_f is 0.

[TRUE/FALSE]
In a dynamic programming solution, the space requirement is always at least as big as the number of unique sub problems.

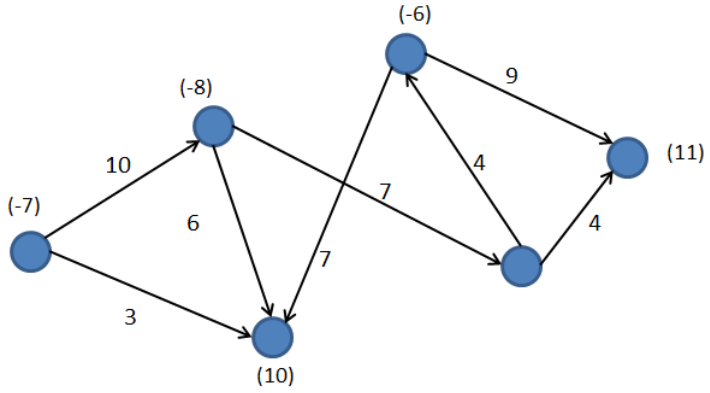
[TRUE/FALSE]
Decreasing the capacity of an edge that belongs to a min cut in a flow network may not result in decreasing the maximum flow.

2) 20 pts

A city is located at one node x in an undirected network $G = (V, E)$ of channels. There is a big river beside the network. In the rainy season, the flood from the river flows into the network through a set of nodes Y. Assume that the flood can only flow along the edges of the network. Let c_{uv} (integer value) represent the minimum effort (counted in certain effort unit) of building a dam to stop the flood flowing through edge (u, v). The goal is to determine the minimum total effort of building dams to prevent the flood from reaching the city. Give a pseudo-polynomial time algorithm to solve this problem. Justify your algorithm.

3) 20 pts

The following graph G is an instance of a circulation problem with demands. The edge weights represent capacities and the node weights (in parantheses) represent demands. A negative demand implies source.



(i) Transform this graph into an instance of max-flow problem.

(ii) Now, assume that each edge of G has a constraint of lower bound of 1 unit, i.e., one unit must flow along all edges. Find the new instance of max-flow problem that includes the lower bound constarint.

4) 20 pts

There is a series of activities lined up one after the other, $J_1, J_2, \ldots, J_n$. The i^{th} activity takes T_i units of time, and you are given M_i amount of money for it. Also for the i^{th} activity, you are given N_i, which is the number of immediately following activities that you cannot take if you perform that i^{th} activity. Give a dynamic programming solution to maximize the amount of money one can make in T units of time. Note that an activity has to be completed in order to make any money on it. State the runtime of your algorithm.

5) 20 pts

A polygon is called convex if all of its internal angles are less than 180°and none of the edges cross each other. We represent a convex polygon as an array V with n elements, where each element represents a vertex of the polygon in the form of a coordinate pair (x, y). We are told that $V[1]$ is the vertex with the least x coordinate and that the vertices $V[1], V[2], \ldots, V[n]$ are ordered counter-clockwise. Assuming that the x coordinates (and the y coordinates) of the vertices are all distinct, do the following.

Give a divide and conquer algorithm to find the vertex with the largest x coordinate in $O(\log n)$ time.

CS570
Analysis of Algorithms
Fall 2016
Exam II

Name: _____

Student ID: _____

Email Address:_____

_____Check if DEN Student

	Maximum	Received
Problem 1	20	
Problem 2	16	
Problem 3	16	
Problem 4	16	
Problem 5	16	
Problem 6	16	
Total	100	

Instructions:

1. This is a 2-hr exam. Closed book and notes
2. If a description to an algorithm or a proof is required please limit your description or proof to within 150 words, preferably not exceeding the space allotted for that question.
3. No space other than the pages in the exam booklet will be scanned for grading.
4. If you require an additional page for a question, you can use the extra page provided within this booklet. However please indicate clearly that you are continuing the solution on the additional page.

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

[FALSE] The run-time is O(nW) so it's pseudopolynomial
0-1 knapsack problem can be solved using dynamic programming in polynomial time

[FALSE] If neither of the two nodes are reachable by the negative cycle, the distance between them is calculated accurately.
The Bellman-Ford algorithm always fails to find the shortest path between two nodes in a graph if there is a negative cycle present in the graph.

[TRUE] Value of max flow = capacity of min-cut (iterate over all edges of the min-cut)
Given the min-cut, we can find the value of max flow in O(|E|).

[TRUE] Set the penalties to 0 and reward of 1 for every match
The sequence alignment algorithm can be used to find the longest common subsequence between two given sequences.

[FALSE] You only have to do it once.
In dynamic programming you must calculate the optimal value of a sub-problem twice, once during the bottom up pass and once during the top down pass.

[TRUE] For any cut which is not a min-cut this is True.
Maximum value of an s-t flow could be less than the capacity of a given s-t cut in a flow network.

[FALSE] Consider a graph with 4 nodes s, a, b and t and E = { (s,a), (a, t), (s, b), (b, t)}. The increase in the flow would be f + 2.
Suppose the maximum (s, t)-flow of some graph has value f. Now we increase the capacity of every edge by 1. Then the maximum (s, t)-flow in this modified graph will have value at most f +1.

[FALSE] The Orlin algorithm runs in O(mn)
There are no known polynomial-time algorithms to solve maximum flow.

[FALSE] Edges having capacity 1 does not mean |f| = 1 so the Ford-Fulkerson is still going to be pseudo-polynomial.
If all edges in a graph have capacity 1, then Ford-Fulkerson runs in linear time.

[FALSE] Choice of augmenting path can still affect the number of iterations of the inner loop.
In the scaled version of the Ford Fulkerson algorithm, choice of augmenting paths cannot affect the number of iterations.

2) 16 pts

Given $N(G(V,E),s,t,c)$, a flow network with source s, sink t, and positive edge capacities c, we are asked to reduce the max flow by removing at most k edges. Prove or disprove the following statement:

Maximum reduction in flow can be achieved by finding a min cut in N, creating a sorted list of the edges going out of this cut in decreasing order and then removing the top k edges in this list.

The Statement is False. (8 points)

Proof (8 points). This requires a counter example where deleting the edges from the min-cut would not result in maximum reduction in max flow.

Many students tried to prove this statement. The statement is true only if all the edges have equal capacities. If the proof was correct based on that assumption, you were given 8 points.

3) 16 pts

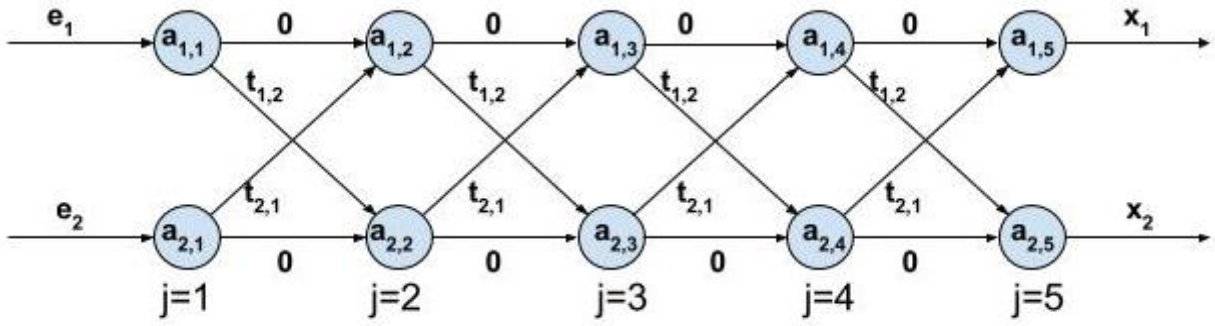
A car factory has k assembly lines, each with n stations. A station is denoted by $S_{i,j}$ where i indicates that the station is on the i-th assembly line (0<i<=k), and j indicates the station is the j-th station along the assembly line (0<j<=n). Each station is dedicated to some sort of work like engine fitting, body fitting, painting and so on. So, a car chassis must pass through each of the n stations in that order before exiting the factory. The time taken per station is denoted by $a_{i,j}$. Parallel stations of the k assembly lines perform the same task. After the car passes through station $S_{i,j}$, it will continue to station $S_{i,j+1}$ unless we decide to transfer it to another line. Continuing on the same line incurs no extra cost, but transferring from line x at station j-1 to line y at station j takes time $t_{x,y}$. Each assembly line takes an entry time e_i and exit time x_i which may be different for each of these k lines. Give an algorithm for computing the minimum time it will take to build a car chassis.

Below figure provides one example to explain the problem. The example shows k=2 assembly lines and 5 stations per line. To illustrate how to evaluate the cost, let's consider two cases: If we always use assembly line 1, the time cost will be:

$$e_1 + a_{1,1} + a_{1,2} + a_{1,3} + a_{1,4} + a_{1,5} + x_1.$$

If we use stations $s_{1,1} \rightarrow s_{2,2} \rightarrow s_{1,3} \rightarrow s_{2,4} \rightarrow s_{2,5}$, the time cost will be

$$e_1 + a_{1,1} + t_{1,2} + a_{2,2} + t_{2,1} + a_{1,3} + t_{1,2} + a_{2,4} + a_{2,5} + x_2.$$



a) Recursively define the value of an optimal solution (recurrence formula) (6 pts)

Dynamic programming Solution

Subproblem: minCost(i,j), if we use assembly line-i at station-j, the minimal cost so far.

Initialization: minCost(1,1) = e1+a11, minCost(2,1) = e2+a21, … minCost(k,1) = ek + ak1 (1 point)

Recurrence relation: *minCost*(i,j)= $\min_{(1\le l\le k)}$ *minCost*(*l*,j-1) + $t_{l,i}$ + $a_{i,j}$ (4 points)
($t_{i,i} = 0$ in this formulation)

Final cost: minCost(i,exit) = minCost(i,n) + x_i (1 point)

Attention: Some students may assume that the car can only pass to the station (i,j) from adjacent lines i-1 and i+1. In this case, you will have 3 points off.

b) Describe (using pseudocode) how the value of an optimal solution is obtained using iteration. Make sure you include any initialization required. (6 pts)

```
// initialization, boundary condition
For i=1:k
   minCost[i][1] = e[i]+a[i][1]          (1 point)

// recurrence
For t=2:n
   For i=1:k
      minCost[i][t] = minCost[1][t-1] + t[1][i]+a[i][t]
      for j=2:k
         minCost[i][t] = min(minCost[j][t-1] + t[j][i]+a[i][t], minCost[i][t])   (4 points)

for i=1:k
   minCost[i][n] = minCost[i][n]+x[i]

// final solution
return min( minCost[1][n], minCost[2][n], … minCost[k][n] )          (1 point)
```

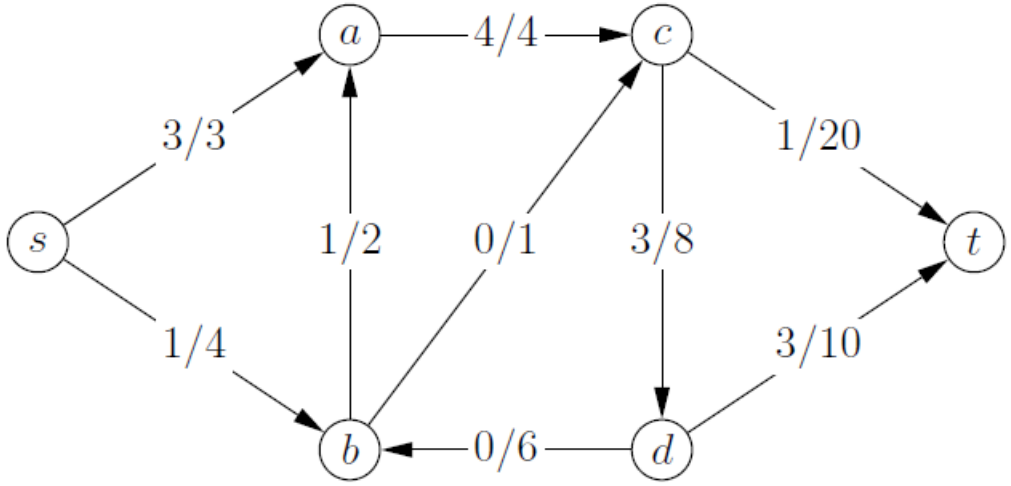
Attention: some students may ignore the initialization.

c) What is the complexity of your solution in part b? (4 pts)

From the recurrence loops, we can tell the complexity is $O(n\times(k^2))$

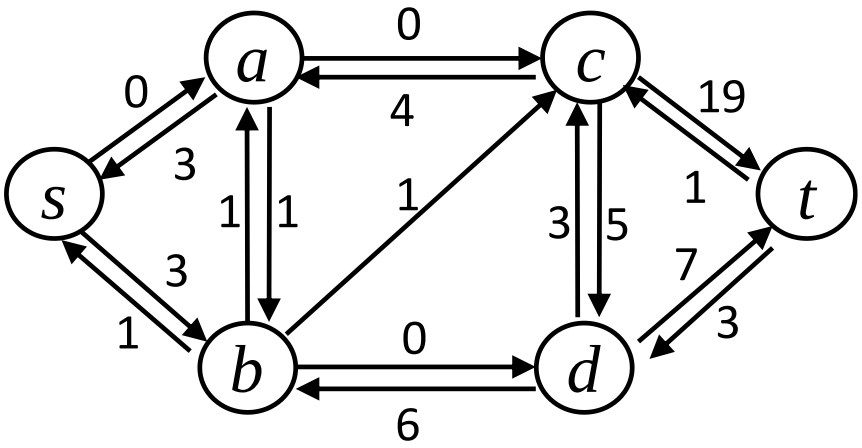
Attention: If you don't show your work process and your final solution is incorrect, you will have 4 points off.

4) 16 pts
You are given a directed graph which after a few iterations of Ford-Fulkerson has the following flow. The labeling of edges indicate flow/capacity:



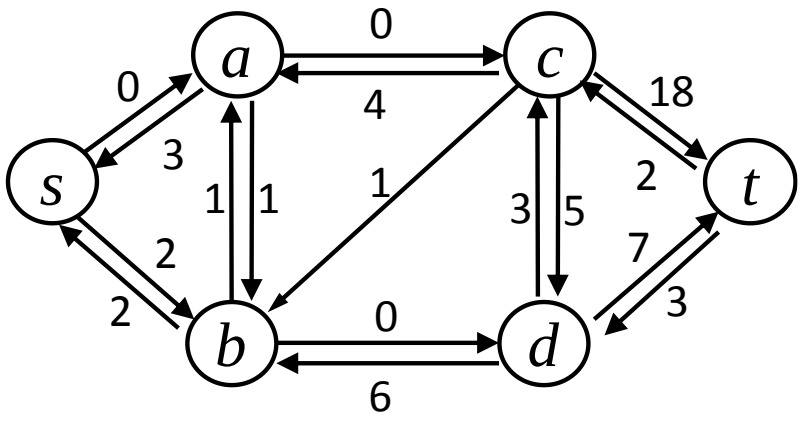
a) Draw the corresponding residual graph. (5 pts)

Each mistake in edge direction or value would cause one-point deduction.



b) Is this a max flow? If yes, indicate why. If no, find max flow. (6 pts)

No, since there is still path like s-b-c-t that can be augmented with flow 1 (3 points) The max flow is 4+1=5 that can be obtained from the residual graph shown below (3 points).



c) What is the min-cut? (5 pts)

Min-cut: ({s,a,b},{c,d,t}) (4 points). For the min-cut you need to show it on the graph or write it in the form (A,B) where A, B are the two parts of the original graph. The value of min-cut equals to $C_{ac}+C_{bc}=4+1=5$. (1 point)

Note that the other cuts like ({s,b},{a,c,d,t}) with value of 6 etc., have larger values than the min-cut with value of 5 and cannot be considered as min-cut for the network.

5) 16 pts

Given a rod of length n inches and an array of prices that contains prices of all pieces of size smaller than or equal to n. Determine the maximum value obtainable by cutting up the rod and selling the pieces. For example, if the length of the rod is 8 and the values of different pieces are given as in the following table, then the maximum obtainable value is 23 (by cutting the rod into two pieces of lengths 2 and 6). Notation: Price(i) is the price of a rod of length i

Length	1	2	3	4	5	6	7	8
Price	1	5	8	9	10	18	17	20

And if the prices are as given in the following table, then the maximum obtainable value is 24 (by cutting the rod into eight pieces of length 1)

Length	1	2	3	4	5	6	7	8
Price	3	5	8	9	10	19	17	20

a) Recursively define the value of an optimal solution (recurrence formula) (6 pts)

The following four solutions are correct –

Soln 1:

OPT(i): best possible price obtained for a rod of length i

OPT(i) = max{p(j) + OPT(i-j)} for all j in {1..i}

Time complexity: $O(n^2)$

Here, the assumption is that the rod needs to be cut and cannot be left uncut.

Soln 2:

OPT(i): best possible price obtained for a rod of length i

OPT(i) = max{OPT(j) + OPT(i-j), p[j]} for all j in {1..i}

Time complexity: $O(n^2)$

Here, the assumption is that the rod need not be cut.

Soln 3:

OPT(i, j): best possible price obtained for a rod of length i with j cuts

OPT(i, j) = max{OPT(k, j-1) + p[i-k], OPT(i, j-1)} for all k in {1..i}

Time complexity: $O(n^3)$

4 points have been deducted for high time complexity.

Many students have used the above definition for OPT (i.e. j cuts as opposed to cuts 1..j) but have not used a third variable k. This is incorrect and marks have been deducted for this since we can have multiple cuts of same length.

Soln 4:

OPT(i, j): best possible price obtained for a rod of length i when the longest cut is no longer than j

OPT(i, j) = max{OPT(i-j, j) + p[j], OPT(i, j-1)}

Time complexity: $O(n^2)$

Grading rubric:

(a) Definition of OPT: 2 marks
Recurrence: 3 marks
Final answer: 1 mark

(b) Initialization: 2 marks
Loop indices: 3 marks
Return final answer: 1 marks

(c) No partial marking
Note that no marks have been given for part c if the recurrence formed in part a was incorrect.

b) Describe (using pseudocode) how the value of an optimal solution is obtained using iteration. Make sure you include any initialization required. (6 pts)

Solution 1 can be written using iteration as

```
int cutRod(int price[], int n)
{
  int val[n+1];
  val[0] = 0;
  int i, j;

  // Build the table val[] in bottom up manner and return the last entry
  // from the table
  for (i = 1; i<=n; i++)
  {
     int max_val = INT_MIN;
     for (j = 1; j <=  i; j++)
       max_val = max(max_val, price[j] + val[i-j]);
```

```
      val[i] = max_val;
    }

   return val[n];
}
// end OR

return cutRod(arr, size)
```

c) What is the complexity of your solution in part b? (4 pts)

$O(n^2)$

6) 16 pts

There are n students in a class. We want to choose a subset of k students as a committee. There has to be m1 number of freshmen, m2 number of sophomores, m3 number of juniors, and m4 number of seniors in the committee. Each student is from one of k departments, where k = m1 +m2 +m3 +m4. Exactly one student from each department has to be chosen for the committee. We are given a list of students, their home departments, and their class (freshman, sophomore, junior, senior).
Describe an efficient algorithm based on network flow techniques to select who should be on the committee such that the above constraints are all satisfied.

a) Describe how to construct a flow network to solve this problem, including the description of nodes, edges, edge directions and their capacities. (10 pts)

b) Describe how the solution to the network flow problem you described in part a corresponds to the problem of determining who should be on the committee. (6 pts)

Solution: Consider the following graph, the source is connected to K departments with capacity one. Each department is connected to a subset of n students, that belongs to the department with capacity one. Each student is connected to the respective group (freshman, sophomore, junior, senior) with capacity one. Each group is connected to the sink with capacity m, for group I. The maximum flow in this problem finds the set of students to be included in the committee. The students' nodes that flow pass through them are shows who should be in the committee

a) The problem has two important constraints :

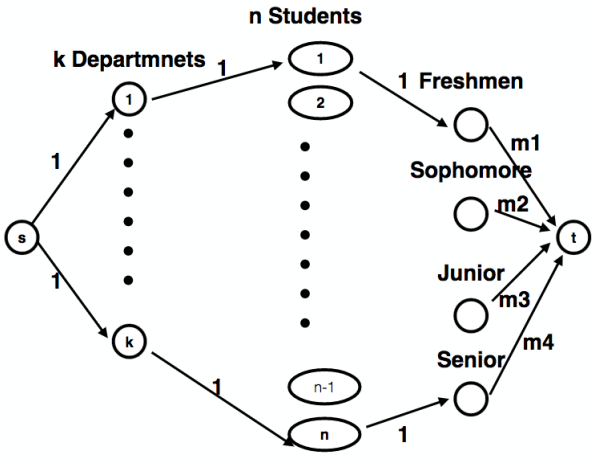
1. Exactly one student from each department must be selected.
   k nodes and k edges with correct capacity expected. (3 points)
2. Number of selected students from each class must be m1 for the freshmen, ..
   4 nodes for classes with capacity m1, m2, m3, m4 expected. (3 points)

For students you can either put some nodes or some edges, But if your graph doesn't give the correct answer you will not get point just for adding n nodes for students. (2 points)
The network flow graph gives the right set of students (2 points)

b) claim: Picking k students for the committee is possible iff The max flow of the given graph in part a is equal to k. (2 points)

Why your graph satisfies the constraint of the problem (2 points)
How the set of students can be determined based on you graph (2 points)

You can either explain these three question or prove the claim, and you will get 6 points.



CS570
Analysis of Algorithms
Spring 2016
Exam II

Name: _____

Student ID: _____

Email Address:_____

_____Check if DEN Student

	Maximum	Received
Problem 1	20	
Problem 2	15	
Problem 3	20	
Problem 4	20	
Problem 5	25	
Total	100	

Instructions:

1. This is a 2-hr exam. Closed book and notes
2. If a description to an algorithm or a proof is required please limit your description or proof to within 150 words, preferably not exceeding the space allotted for that question.
3. No space other than the pages in the exam booklet will be scanned for grading.
4. If you require an additional page for a question, you can use the extra page provided within this booklet. However please indicate clearly that you are continuing the solution on the additional page.

This study resource was shared via CourseHero.com

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

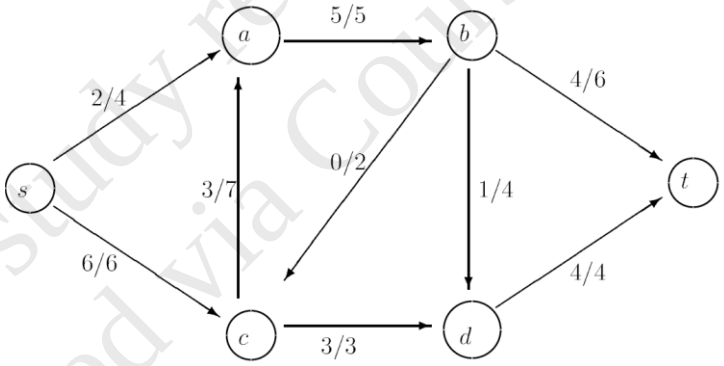
[TRUE]
The number of iterations it takes Bellman-Ford to converge can vary depending on the order of nodes updated within an iteration.

[FALSE]
In a flow network, if the capacity of every edge is odd, then there is a maximum flow in which the flow on each edge is odd.

[TRUE]
Maximum value of an s-t flow could be less than the capacity of a given s-t cut in a flow network.

[FALSE]
Any Dynamic Programming algorithm with n^2 unique sub-problems will run in $O(n^2)$ time.

[TRUE]
The following flow is a maximal flow.
Note: The notation *a/b* describes *a* units of flow on an edge of capacity *b*.



This study resource was shared via CourseHero.com

https://www.coursehero.com/file/14397679/Exam-II-solution-Spring-2016pdf/

[FALSE]

In a circulation network, there is a feasible circulation with demands $\{d_v\}$, if $\sum_v d_v = 0$

[FALSE]

An optimal solution to a 0/1 knapsack problem will always contain the object i with the greatest value-to-cost ratio V_i/C_i.

[TRUE]

The dynamic programming solution presented in class for the 0/1 knapsack problem is not an efficient solution.

[FALSE]

For any edge e that is part of the minimum cut in G, if we increase the capacity of that edge by any integer k>1, then that edge will no longer be part of the minimum cut.

[TRUE]

Ford-Fulkerson Algorithm cannot solve the max-flow problem in a flow network in polynomial time, however, there are other algorithms that can solve this problem in polynomial time.

This study resource was shared via CourseHero.com

2) 15 pts

Suppose you have a DAG with costs $c_e > 0$ on each edge and a distinguished vertex s. Give a dynamic programming algorithm to find the most expensive path in the graph that begins at s. Your solution should include the recurrence formula for the cost of the path, pseudo code to show your implementation, and complexity analysis. Your algorithm should return the most expensive path. For full credit, your algorithm's runtime should be linear.

Let's consider the vertices in topological order. Define OPT(v) as the longest distance from s to v.

Our recurrence is then:
OPT(v) = max_{u ∈ adj(v) } {OPT(u) + w(u,v)},
where adj(v) is the set of vetices, each of which has an edge to node v.

Base case OPT(s) = 0; OPT(v) = -infinity, if v != s and adj(v) is empty

Algorithm:

Do topological ordering and get ordering indices as 1,….,V

```
For v = 1 to V
        If v == s
                OPT(s) = 0
        else if  adj(v) is empty
                OPT(v) = -infinity
        else
                OPT(v) = max_{u ∈ adj(v) } {OPT(u) + w(u,v)},

        End
        Record the node u ∈ adj(v)  that achieves OPT(v), denote it as p(v);
End

/* the above running time is O(|V|+|E|)

longest distance = max_{v} {OPT(v)};
v* = argmax_{v} {OPT(V)};

/*the above running time is O(|V|)
```

This study resource was shared via CourseHero.com

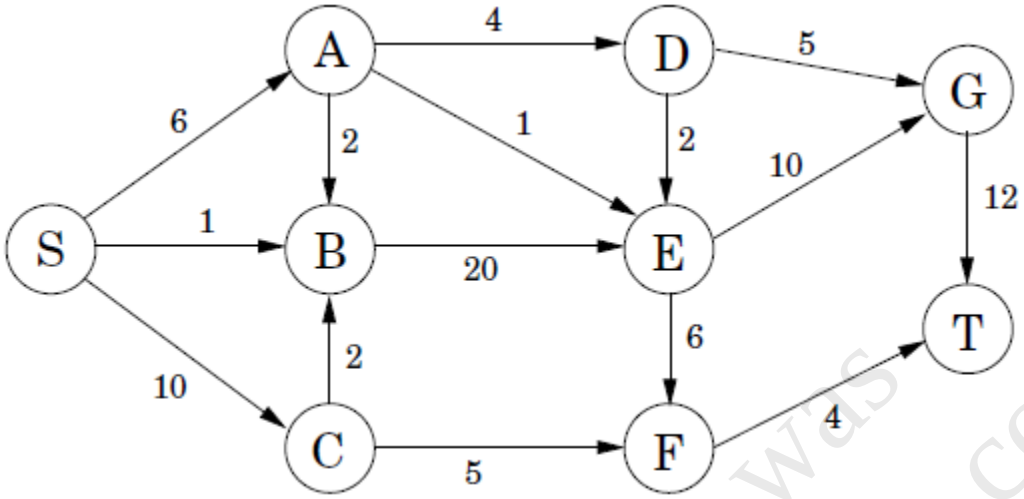
```
/*Return the longest path as follows******************

Path = [p(v*), v*];  /* the longest path
u = v*;
While (p(u)!= s)
        u = p(u);
        Path = [p(u), Path];
End

Return Path;

/*the above running time is O(|V|)
```

Complexity: O(|V|+|E|)

This study resource was shared via CourseHero.com

3) 20 pts

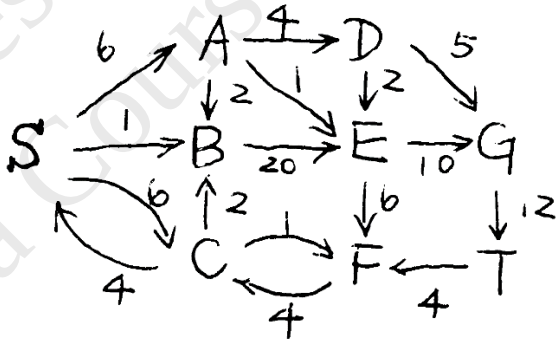
For the following network, with edge capacities as shown,



a) Find the maximum flow from *S* to *T* using the Ford Fulkerson algorithm. You need to show the work involved in each iteration including the residual graph and augmenting path.

Solution:

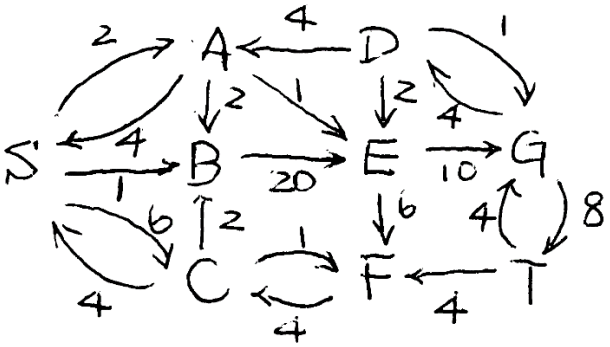
Iteration 1:



Residual graph:

Augmenting path: S-C-F-T (flow value = 4)

Iteration 2:

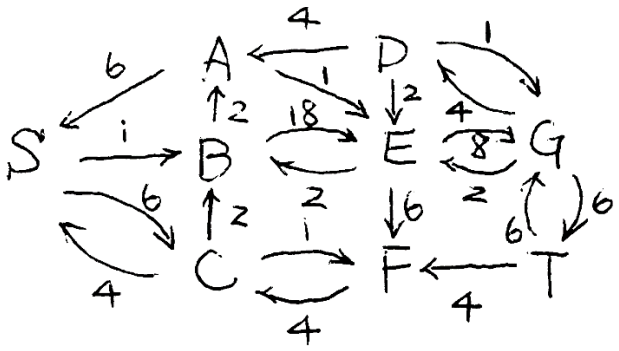


Residual graph:

Augmenting path: S-A-D-G-T (flow value = 4)

This study resource was shared via CourseHero.com

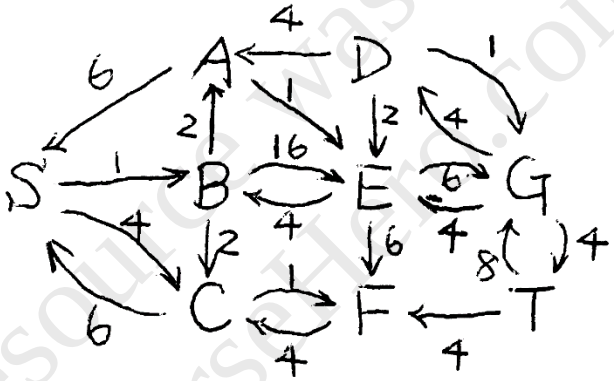
Iteration 3:



Residual graph:

Augmenting path: S-A-B-E-G-T (flow value = 2)

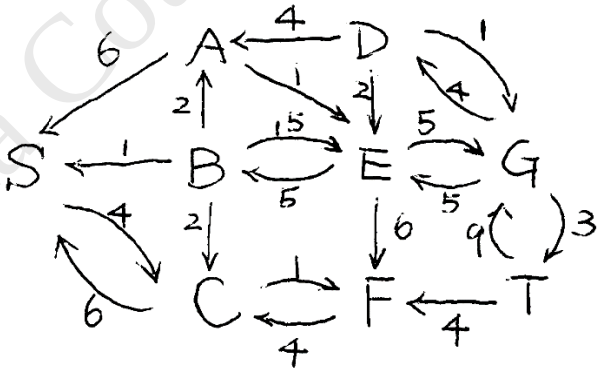
Iteration 4:



Residual graph:

Augmenting path: S-C-B-E-G-T (flow value = 2)

Iteration 5:



Residual graph:

Augmenting path: S-B-E-G-T (flow value = 1)

Since there is no more augmenting path in the last residual graph, the maximum flow value = 4 + 4 + 2 + 2 + 1 = 13

b) Using the solution to part a, find an s-t min cut. You need to show the steps to find the min cut.

Solution:

In the last residual graph in a), starting from *S*, the nodes can be reached are *S*, *C*, and *F*. Thus, the minimum cut from a): {S, C, F} {A, B, D, E, G, T}.

This study resource was shared via CourseHero.com

4) 20 pts

You are given a sequence of *n* numbers (positive or negative): $x_1, x_2, \ldots, x_n$. Your job is to select a subset of these numbers of maximum total sum, subject to the constraint that you can't select two elements that are adjacent (that is, if you pick x_i then you cannot pick either x_{i-1} or x_{i+1}). On the boundaries, if x_1 is chosen, x_2 cannot be chosen; if x_n is chosen, then x_{n-1} cannot be chosen. Give a dynamic programming solution to find, in time polynomial in *n*, the subset of maximum total sum. Please give the optimal cost equation, and pseudo code for your solution. Also state the complexity of your solution.

Solution: Let sum_i be the maximum sum of the numbers $x_1, x_2, \ldots, x_i$ given the adjacency constraint.

$$sum_0 = 0$$
$$sum_1 = max(0, x_1)$$
$$sum_i = max(sum_{i-2} + x_i, sum_{i-1})$$

This last step works because either we include x_i, in which case we also want to include the bestsolution on up to i – 2, or we don't include x_i, in which case we can just use the best solution on i – 1.

Our final answer is then just sum_n.To calculate the set that gives the max sum, we could simply keep pointers back from i to either i – 1 or i – 2 depending on which one was bigger (or we could go back and check which was bigger).We follow those pointers, including appropriate numbers. Because there are n subproblems, and each subproblem takes O(1) time to solve, this runs in O(n) time.

```
01: OPT[-1, 0, 1, …, n] := [0, …, 0]
02: for i = 1, …, n:
03:     OPT[i] := max(OPT[i-2]+x_i, OPT[i-1])
04: S := [], j := n
05: while j > 0:
06:     if OPT[j] = OPT[j-2] + x_j:
07:         S.append(x_j), j := j-2
08:     else:
09:         j := j-1
10: return S
```

This study resource was shared via CourseHero.com

https://www.coursehero.com/file/14397679/Exam-II-solution-Spring-2016pdf/

5) 25 pts

We define a most vital edge of a network as an edge whose deletion causes the largest decrease in the maximum s-t-flow value. Let f be an arbitrary maximum s-t-flow. Either prove the following claims or show through counterexamples that they are false:

(a) A most vital edge is an edge e with the maximum value of c(e).

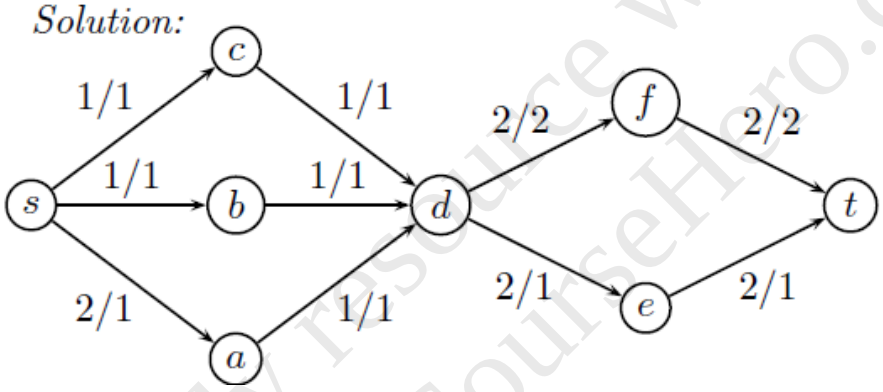
(b) A most vital edge is an edge e with the maximum value of f (e).

(c) A most vital edge is an edge e with the maximum value of f (e) among edges belonging to some minimum cut.

(d) An edge that does not belong to any minimum cut cannot be a most vital edge.

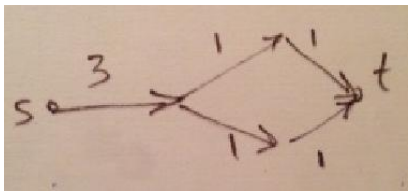
(e) A network can contain only one most vital edge.

Solution:



This is a counterexample for (a), (b) (d) and (e). The specified row f is a maximum flow with the value three. The deletion of any edge decreases the flow value by one. Hence, each edge is a most vital arc ((e) is false). The edge e = (s, b) has neither the maximum value of c(e) nor the maximum value of f(e); still, it is a most vital arc ((a) and (b) are false). The edge (s, a) does not belong to any minimum cut; still, it is a most vital arc ((d) is false).

Part (c) is false and here is the counter example: all the edges except the one connecting s belong to at least one min-cut, and the flow on each of those edges is 1, and therefore also the maximum . However, deleting any of the edges belonging to min-cut can only reduce the max-flow value by 1, in contrast, deleting the edge connecting s can decrease the flow by 2.



Study resource was shared via CourseHero.com

https://www.coursehero.com/file/14397679/Exam-II-solution-Spring-2016pdf/

RCS570 Fall 2017: Analysis of Algorithms Exam II

	Points
Problem 1	20
Problem 2	20
Problem 3	15
Problem 4	15
Problem 5	15
Problem 6	15
Total	100

Instructions:

1. This is a 2-hr exam. Closed book and notes
2. If a description to an algorithm or a proof is required please limit your description or proof to within 150 words, preferably not exceeding the space allotted for that question.
3. No space other than the pages in the exam booklet will be scanned for grading.
4. If you require an additional page for a question, you can use the extra page provided within this booklet. However please indicate clearly that you are continuing the solution on the additional page.
5. Do not detach any sheets from the booklet. Detached sheets will not be scanned.
6. If using a pencil to write the answers, make sure you apply enough pressure so your answers are readable in the scanned copy of your exam.
7. Do not write your answers in cursive scripts.

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

[TRUE/FALSE]
It is possible for a dynamic programming algorithm to have an exponential running time.

[TRUE/FALSE]
In a connected, directed graph with positive edge weights, the Bellman-Ford algorithm runs asymptotically faster than the Dijkstra algorithm.

[TRUE /FALSE]
There exist some problems that can be solved by dynamic programming, but cannot be solved by greedy algorithm.

[TRUE /FALSE]
The Floyd-Warshall algorithm is asymptotically faster than running the Bellman-Ford algorithm from each vertex.

[TRUE /FALSE]
If we have a dynamic programming algorithm with n^2 subproblems, it is possible that the space usage could be $O(n)$.

[TRUE /FALSE]
The Ford-Fulkerson algorithm solves the maximum bipartite matching problem in polynomial time.

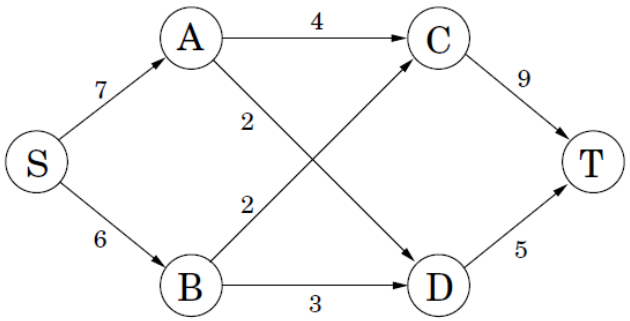
[TRUE /FALSE]
Given a solution to a max-flow problem, that includes the final residual graph G_f. We can verify in a *linear* time that the solution does indeed give a maximum flow.

[TRUE /FALSE]
In a flow network, a flow value is upper-bounded by a cut capacity.

[TRUE/FALSE]
In a flow network, a min-cut is always unique.

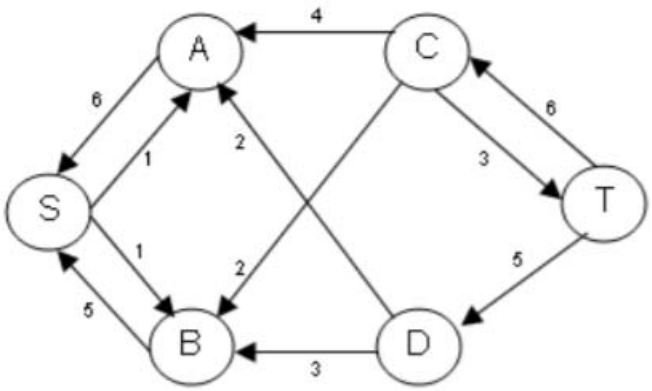
[TRUE/FALSE]
A maximum flow in an integer capacity graph must have an integer flow on each edge.

2) 20 pts.
You are given the following graph G. Each edge is labeled with the capacity of that edge.



a) Find a max-flow in G using the Ford-Fulkerson algorithm. Draw the residual graph G_f corresponding to the max flow. You do not need to show all intermediate steps. (10 pts)

solution



Grading Rubics:

Residual graph: 10 points in total

For each edge in the residual graph, any wrong number marked, wrong direction, or missing will result in losing 1 point.

The total points you lose is equal to the number of edges on which you make any mistake shown above.

b) Find the max-flow value and a min-cut. (6 pts)

Solution: f = 11, cut: ({S, A, B}, {C, D, T })

Grading Rubics:

Max-flow value and min-cut: 6 points in total

Max-flow: 2 points.

- If you give the wrong value, you lose the 2 points.

Min-cut: 4 points

- If your solution forms a cut but not a min-cut for the graph, you lose 3 points
- If your solution does not even form a cut, you lose all the 4 points

c) Prove or disprove that increasing the capacity of an edge that belongs to a min cut will always result in increasing the maximum flow. (4 pts)

Solution: increasing (B,D) by one won't increase the max-flow

Grading Rubics:

Prove or Disprove: 4 points in total

- If you judge it "True", but give a structural complete "proof". You get at most 1 point
- If you judge it "False", you get 2 points.
- If your counter example is correct, you get the rest 2 points.

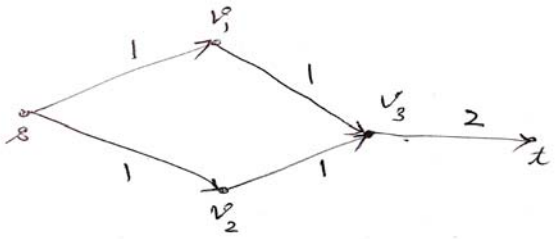
Popular mistake: a number of students try to disprove it by showing that if the min-cut in the original graph is non-unique, then it is possible to find an edge in one min-cut set, such that increasing the capacity of this does not result in max-flow increase.
But they did not do the following thing:
The existence of the network with multiple min-cuts needs to be proved, though it seems to be obvious. The most straightforward way to prove the existence is to give an example-network that has multiple min-cuts. Then it turns out to be giving a counter example for the original question statement.

3) 15 pts.
Given a flow network with the source *s* and the sink *t*, and positive integer edge capacities *c*. Let (S,T) be a minimum cut. Prove or disprove the following statement:
If we increase the capacity of every edge by 1, then (S,T) still be a minimum cut.

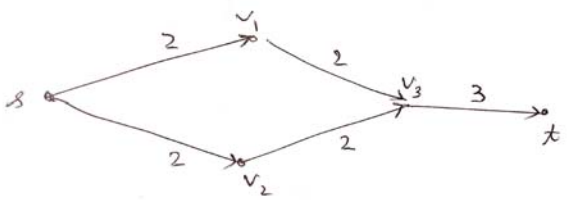
False. Create a counterexample.

An instance of a counter-example:

Initially, a (S,T) min cut is `S:{s,v_1,v_2}` and `T:{v_3,t}`



After increasing capacity of every edge by 1, (S,T) is no longer a min-cut. We now have `S':{s,v_1,v_2,v_3}` and `T':{t}`



Grading rubric:
If you try to prove the statement: -15
For an incorrect counter-example: -9
No credit for simply stating true or false.

4) 15 pts.
Given an unlimited supply of coins of denominations $d_1, d_2, \ldots, d_n$, we wish to make change for an amount C. This might not be always possible. Your goal is to verify if it is possible to make such change. Design an algorithm by *reduction* to the knapsack problem.

a) Describe reduction. What is the knapsack capacity, values and weights of the items? (10 pts.)

Capacity is C. (2 points)
values = weights = d_k. (4points)

- You need to recognize that the problem should be modeled based on the unbounded knapsack problem with description of the reduction: (5 points)
- Explanation of the verification criteria (4 points)
  Opt(j): the maximum change equal or less than j that can be achieved with d1, d2, …, dn
  Opt(0)=0
  Opt(j) = max[opt(j-di)+di] for di<=j
  If we obtain Opt(C) = C, it means the change is possible.

b) Compute the runtime of your verification algorithm. (5 pts)

O(nC) (3 points), Explanation (2 points).

5) 15 pts.
You are considering opening a series of electrical vehicle charging stations along Pacific Coast Highway (PCH). There are n possible locations along the highway, and the distance from the start to location k is $d_k \geq 0$, where $k = 1, 2, \ldots, n$. You may assume that d i < d k for i < k. There are two important constraints:

1) at each location k you can open only one charging station with the expected profit p_k, $k = 1, 2, \ldots, n$.
2) you must open at least one charging station along the whole highway.
3) any two stations should be at least M miles apart.

Give a DP algorithm to find the maximum expected total profit subject to the given constraints.

a) Define (in plain English) subproblems to be solved. (3 pts)

Let OPT(k) be the maximum expected profit which you can obtain from locations 1, 2, …, k.

Rubrics
Any other definition is okay as long as it will recursively solve subproblems

b) Write the recurrence relation for subproblems. (6 pts)

$OPT(k) = \max\{OPT(k-1), p_k + OPT(f(k))\}$
where f(k) finds the largest index j such that $d_j \leq d_k - M$, when such j doesn't exist, f(k)=0

Base cases:
$OPT(1) = p_1$
$OPT(0) = 0$

Rubrics:
Error in recursion -2pts, if multiple errors, deduction adds up
No base cases: -2pts
Missing base cases: -1pts
Using variables without definition or explanation: -2pts
Overloading variables (re-defining n, p, k, or M): -2pts

c) Compute the runtime of the above DP algorithm in terms of *n*. (3 pts)

algorithm solves n subproblems; each subproblem requires finding an index f(k) which can be done in time O(log n) by binary search.
Hence, the running time is O(n log n).

Rubric:
$O(n^2)$ is regarded as okay
O(d_n n) pseudo-polynomial is also okay if the recursion goes over all d_n values
O(n) is also okay

$O(n^3)$,O(nM), O(kn) : no credit

d) How can you optimize the algorithm to have O(*n)* runtime? (3 pts)

Preprocess distances $r_i = d_i - M$. Merge d-list with r-list.

Rubric
Claiming "optimal solution is already found in c" only gets credit when explanation about pre-process is described in either part b or c.
Without proper explanation (*e.g.* assume we have, or we can do): no credit
Keeping an array of max profits: no credit, finding the index that is closest to the installed station with M distance away is the bottleneck, which requires pre-processing.

6) 15 pts.
A group of traders are leaving Switzerland, and need to convert their Francs into various international currencies. There are n traders t 1 ,t 2 ,…,t n and m currencies c 1 ,c 2 ,…,c m . Trader t k has F k Francs to convert. For each currency c j , the bank can convert at most B j Franks to c j . Trader t k is willing to trade as much as S kj of his Francs for currency c j .(For example, a trader with 1000 Francs might be willing to convert up to 200 of his Francs for USD, up to 500 of his Francs for Japanese's Yen, and up to 200 of his Francs for Euros). Assuming that all traders give their requests to the bank at the same time, describe an algorithm that the bank can use to satisfy the requests (if it can).

a) Describe how to construct a flow network to solve this problem, including the description of nodes, edges, edge directions and their capacities. (8 pts)
Bipartite graph: one partition traders $t_1, t_2, \ldots, t_n$. Other, available currency, $c_1, c_2, \ldots, c_m$.
Connect t_k to c_j with the capacity S_{kj}
Connect source to traders with the capacity F_k.
Connect available currency c_j to the sink with the capacity B_j.
Rubrics:
- Didn't include supersource (-1 point)
- Didn't include traders nodes (-1 point)
- Didn't include currencies nodes (-1 point)
- Didn't include supersink (-1 point)
- Didn't include edge direction (-1 point)
- Assigned no/wrong capacity on edges between source & traders (-1 point)
- Assigned no/wrong capacity on edges between traders & currencies (-1 point)
- Assigned no/wrong capacity on edges between currencies & sink (-1 point)

b) Describe on what condition the bank can satisfy all requests. (4 pts)
If there is a flow f in the network with $|f| = F_1 + \ldots + F_n$. then all traders are able to convert their currencies.
Rubrics:
- Wrong condition (-4 points): Unless you explicitly included $|f| = F_1 + \ldots + F_n$, no partial points were given for this subproblem.
- In addition to correct answer, added additional condition which is wrong (-2 points)
- No partial points were given for conditions that satisfy only some of the requests, not all requests.
- Note that ΣS_{kj} and ΣB_j can be larger than ΣF_k in some cases where bank satisfy all requests.
- Note that ΣS_{kj} can be larger than ΣB_j in some cases where bank satisfy all requests.
- It is possible that ΣS_{kj} or ΣB_j is smaller than ΣF_k. In these cases, bank can never satisfy all requests because max flow will be smaller than ΣF_k.

c) Assume that you execute the Ford-Fulkerson algorithm on the flow network graph in part a), what would be the runtime of your algorithm? (3 pts)

$O(n\ m\ |f|)$

Rubrics:

- Flow($|f|$) was not included (-1 point)
- Wrong description (-1 point)
- Computation is incorrect (-1 point)
- Notation error (-1 point)
- Missing big O notation (-1 point)
- Used big Theta notation instead of big O notation (-1 point)
- Wrong runtime complexity (-3 points)
- No runtime complexity is given (-3 points)

CS570
Analysis of Algorithms
Spring 2017
Exam II

Name: _____

Student ID: _____

Email Address:_____

_____Check if DEN Student

	Maximum	Received
Problem 1	20	
Problem 2	10	
Problem 3	10	
Problem 4	15	
Problem 5	15	
Problem 6	15	
Problem 7	15	
Total		

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

[/FALSE] The value of the max flow is not enough and we need the complete flow
Given the value of max flow, we can find a min-cut in linear time.

[/FALSE]
The Ford-Fulkerson algorithm can compute the maximum flow in polynomial time.

[/FALSE]
A network with unique maximum flow has a unique min-cut.

[TRUE/]
If all of the edge capacities in a graph are an integer multiple of 3, then the value of the maximum flow will be a multiple of 3.

[/FALSE]
The Floyd-Warshall algorithm always fails to find the shortest path between two nodes in a graph with a negative cycle.

[/FALSE] 0/1 knapsack does not have a polynomial time dynamic programming solution—not even a weakly polynomial solution. 0/1 knapsack has a pseudopolynomial time dynamic programming solution which is basically an exponential time solution (with respect to its input size)
0/1 knapsack problem can be solved using dynamic programming in polynomial time, but not **strongly** polynomial time.

[/FALSE]
If a dynamic programming algorithm has *n* subproblems, then its running time complexity is *O(n)*.

[/FALSE]
The Travelling Salesman problem can be solved using dynamic programming in polynomial time

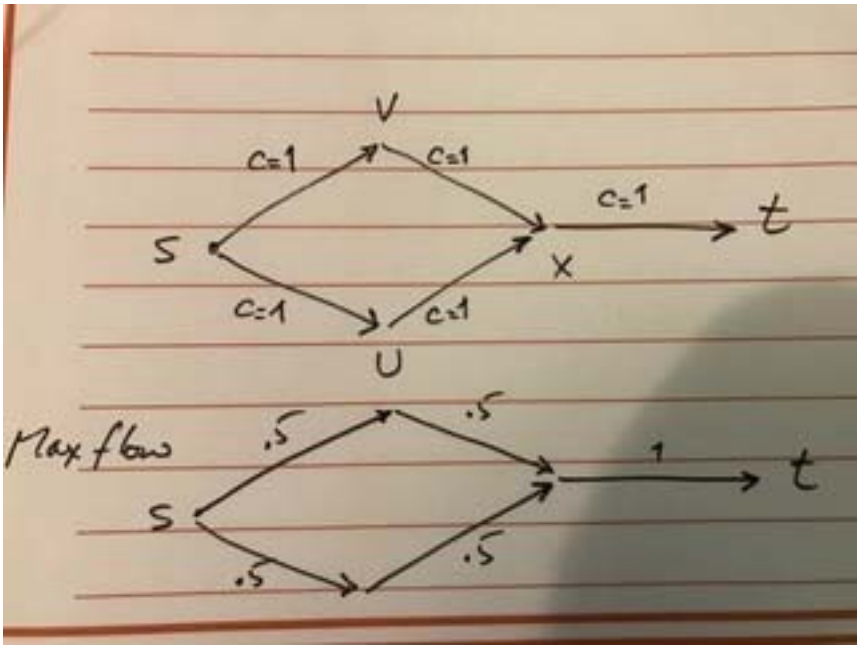
[/FALSE]
If flow in a network has a cycle, this flow is not a valid flow.

[FALSE/] We can't do this for *undirected* graphs
We can use the Bellman-Ford algorithm for undirected graph with negative edge weights.

2) 10 pts

Given $N(G(V,E),s,t,c)$, a flow network with source s, sink t, and positive integer edge capacities *c(e)* for every *e* □ *E*. Prove or disprove the following statement:
A maximum flow in an integer capacity graph must have integral (integer) flow on each edge.

Counterexample:



Grading:
A correct counter example gets full 10 points, otherwise 0 points is given.

Common Mistake:
0 is an integer so it will not count as a non-integral flow.

3) 10 pts

The subset sum problem is defined as follows: Given a set of n positive integers $S=\{a_1, a_2, ..., a_n\}$ and a target value T, does there exist a subset of S such that the sum of the elements in the subset is equal to T? Let's define a boolean matrix M where M(i, j) is true if there exists a subset of $\{a_1, a_2, ..., a_i\}$ whose sum equals j. Which one of the following recurrences is valid? (circle one)

1) $M(i, j) = M(i-1, j)$ Ú $M(i, j- a_i)$

2) $M(i, j) = M(i-1, j)$ Ú $M(i-1, j- a_i)$ This is the correct solution

3) $M(i, j) = M(i-1, j-1)$ Ú $M(i-1, j)$

4) $M(i, j) = M(i, j- a_i)$ Ú $M(i-1, j - a_i)$

Solution:
For computing M(i, j) you either include a_i or you don't. If you include a_i then we have M(i, j) = M(i-1, j-a_i). If you don't include a_i, then M(i, j) = M(i-1, j).

Grading:
Recurrence 2 gets full 10 points. Recurrences 1 and 4 get 5 points since they have one of the two cases correct. Recurrence 3 gets 0 points.

4) 15 pts

This problem involves partitioning a given input string into disjoint substrings in the cheapest way. Let $x_1x_2 \ldots x_n$ be a string, where particular value of x_i does not matter. Let C(i,j) (for $i \leq j$) be the given precomputed cost of each substring $x_i \ldots x_j$. A partition is a decomposition of a string into disjoint substrings. The cost of a partition is the sum of costs of substrings. The goal is to find the min-cost partition.
An example. Let "ab" be a given string. This string can be partitioning in two different ways, such as, "a", "b", and "ab" with the following costs C(1,1) + C(2,2) and C(1,2) respectively.

a) Define (in plain English) subproblems to be solved. (5 pts)

Let subproblem OPT(j) be the min-cost partition $x_1x_2 \ldots x_j$

b) Write the recurrence relation for subproblems. (7 pts)

$OPT(j) = \min_{1 \leq k < j} (OPT(k-1) + C(k, j))$
$OPT(0) = 0$

c) Compute the runtime of the algorithm. (3 pts)

$\Theta(n^2)$.

Common mistake:

If you write the recurrence based on two parameters of i,j for the corresponding subsequence xi…xj like opt(i,j) you might lose points from 4 to 7 points in part b, depending on your solution. In this case, you probably came up with complexity of $O(n^3)$ instead of $O(n^2)$ which is not optimum compared to the right solution and you might lose other points (2 to 3 points) in part c.

5)

You have two rooms to rent out. There are n customers interested in renting the rooms. The i^{th} customer wishes to rent one room (either room you have) for d[i] days and is willing to pay bid[i] for the entire stay. Customer requests are non-negotiable in that they would not be willing to rent for a shorter or longer duration. Devise a dynamic programming algorithm to determine the maximum profit that you can make from the customers over a period of D days.

Let OPT(d1, d2, i) be the maximum profit obtainable with d1 remaining days for room 1 and d2 remaining days for room 2 using the first i customers.

a) Write the recurrence relation for subproblems. (7 pts)

$$OPT(d1, d2, i) = \max(bid[i] + OPT(d1 - d[i], d2, i - 1),\ bid[i] + OPT(d1, d2 - d[i], i - 1),\ OPT(d1, d2, i - 1))$$

Initial conditions
$OPT(d1,d2,0) = 0$
$OPT(d1,d2,i) = -\infty$ if $d1 < d[i]$ or $d2 < d[i]$

b) Compute the runtime of the algorithm. (4 pts)

$O(nD^2)$

Grading Rubric:

Common mistakes:
1. Most common mistake is to model each room separately, so having two recurrence for two rooms. This is not true, and will lead to the same solution for each recurrence.
2. Another common mistake is to model the problem using 2D days (simply add up), this is also not true. i.e. If a customer wants to stay for 4 days and there are 2 days left in both rooms, then this approach of combining two rooms into one incorrectly suggests that the customer can stay.
3. Also, there are some students model the problem using three variables, (days, customers, rooms), this is not true.

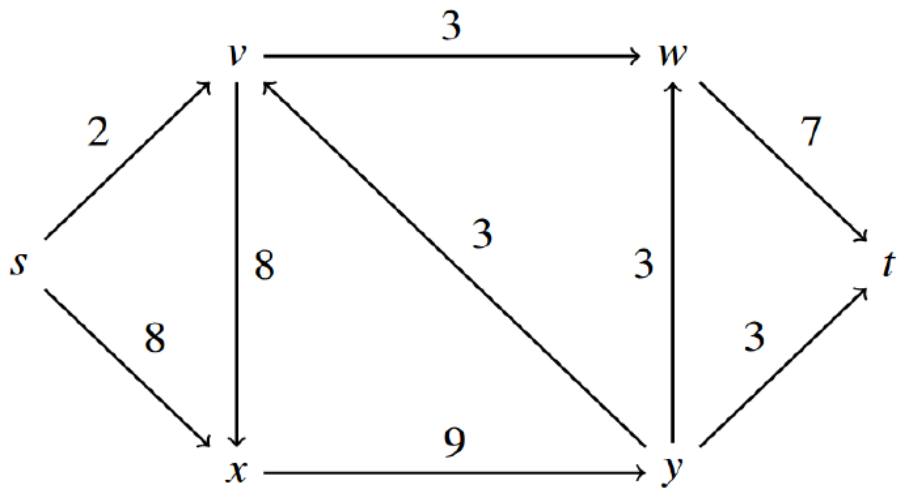
Credits:
1. Without boundary condition, or incorrect boundary condition : -2 points

2. Every missing recurrence subproblem : -3 points. i.e. the correct recurrence should be max {f_1, f_2, f_3}, but one of them is missing.
3. Incorrect time complexity: -4 points (no partial credit)

Partial credits:
For common mistakes mentioned above, give maximum of 3 points (depends on the correctness of recurrence equation and run time complexity, if the recurrence equation is not correct then no credit was given even though the complexity is correct)

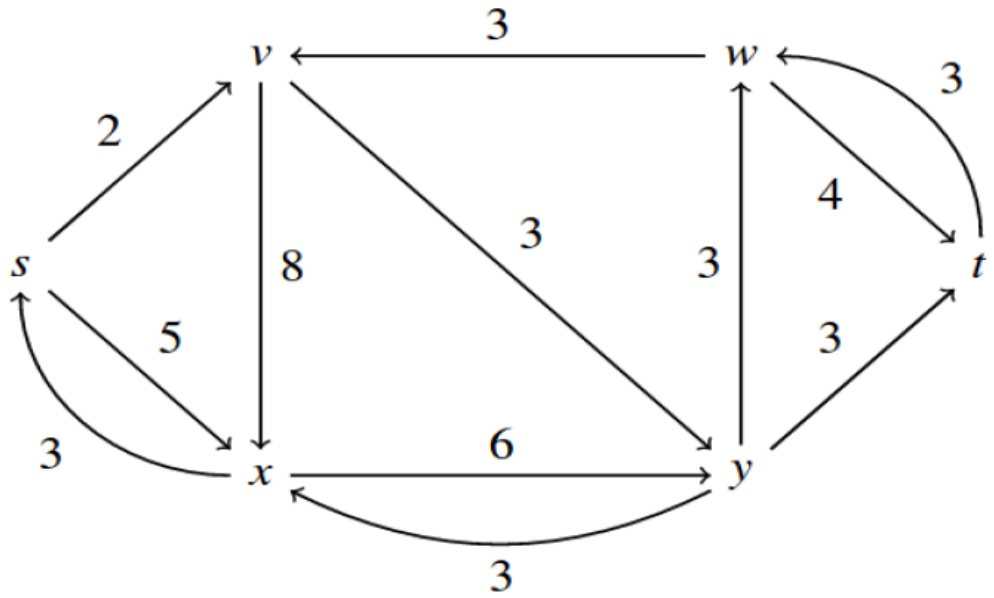
6) 15 pts

You are given the following graph. Each edge is labeled with the capacity of that edge.



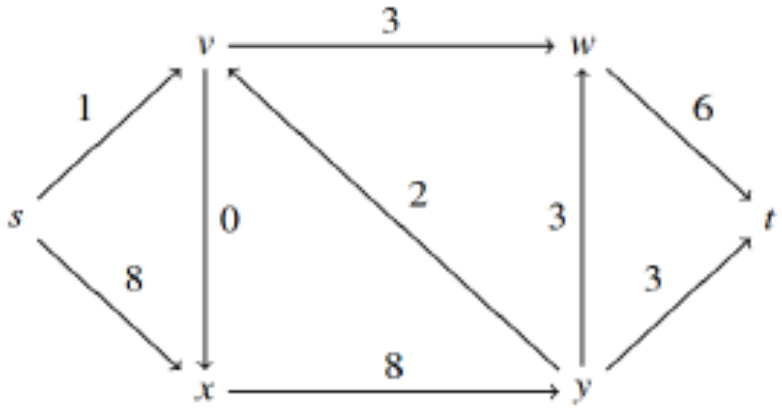
a) Draw the corresponding residual graph after sending as much flow as possible along the path s→x→y→v→w→t. (5 pts)

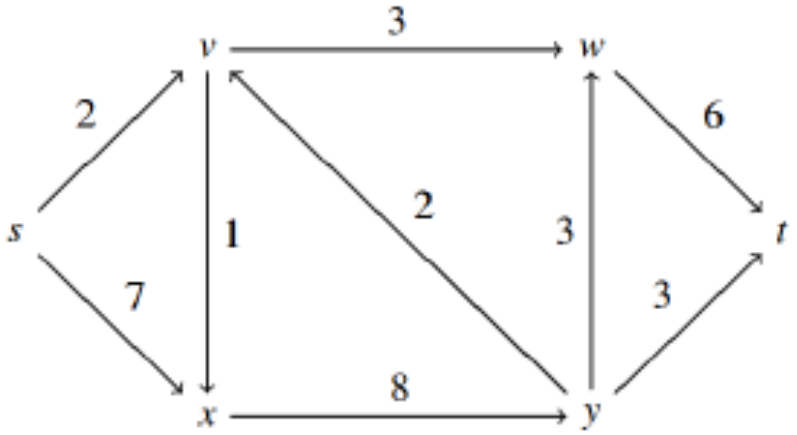
3

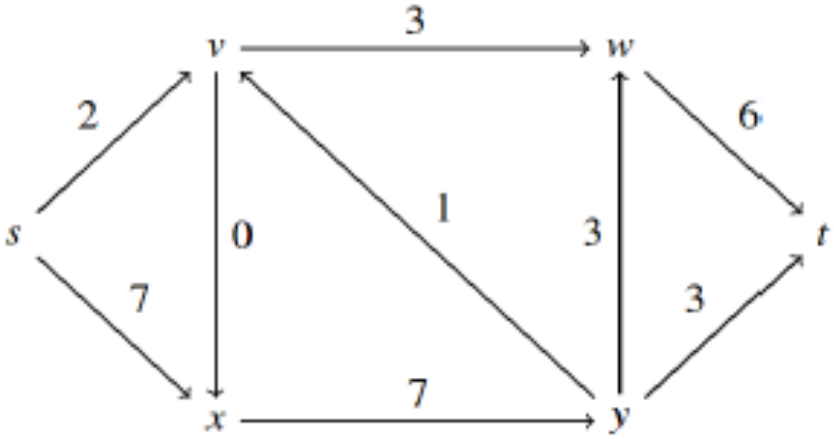


b) Find the value of a max-flow. (5 pts)

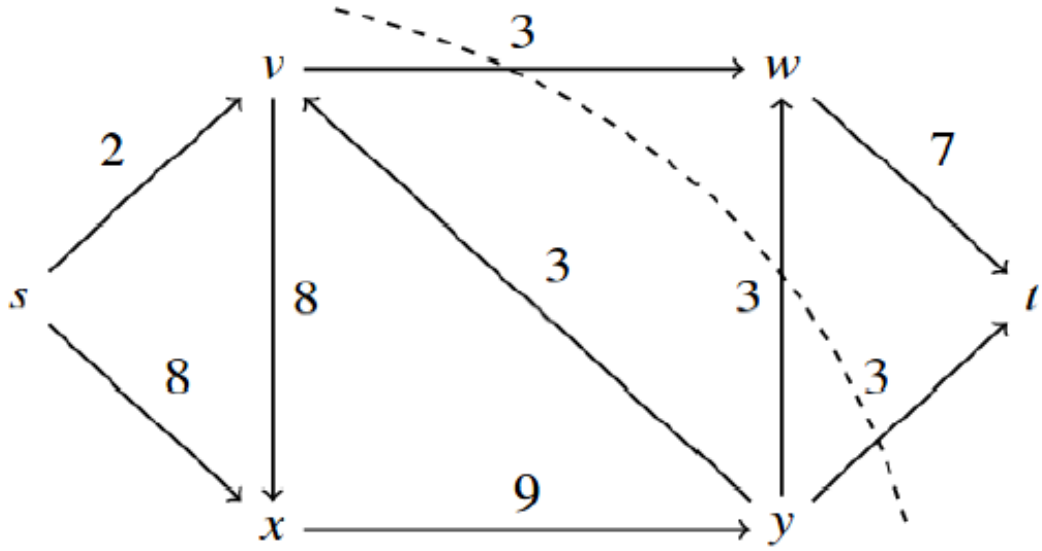
9, multiple answers







c) Find a min-cut? (5 pts)



7) 15 pts

Consider a drip irrigation system, which is an irrigation method that saves water and fertilizer by allowing water to drip slowly to the roots of plants. Suppose that the location of all drippers are given to us in terms of their coordinates (x^d, y^d). Also, we are given locations of plants specified by their coordinates (x^p, y^p).

A dripper can only provide water to plants within distance l. A single dripper can provide water to no more than n plants. However, we recently got some funding to upgrade our system with which we bought k monster drippers, which can provide water supply to three times the number of plants compared to standard drippers. So, we now have i standard drippers and k monster drippers.

Given the locations of the plants and drippers, as well as the parameters l and n, decide whether every plant can be watered simultaneously by a dripper, subject to the above mentioned constraints. Justify carefully that your algorithm is correct and can be obtained in polynomial time.

Solution:

Create a graph G=(V, E)

V = {source s, nodes p_1, …, p_j for sets of plants, nodes d_1, …, d_i for standard droppers, nodes for d_{i+1}, …, d_{i+k}, and sink t} (1 pt)

E = {

$c(s, p_x) = 1$, for x = 1 to j (1 pt)

$c(p_x, d_y) = 1$, for x = 1 to j, y = 1 to i+k, (1 pt) and p_x, d_y are close enough (1 pt)

$c(d_y, t) = n$, for y = 1 to I (2 pt)

$c(d_y, t) = 3n$, for y = i to i+k (2 pt)

}

Use Ford-Fulkerson algorithm to find maximum integer flow f in G and justification. (7 pt)#

CS570 Fall 2018: Analysis of Algorithms Exam I

	Points		Points
Problem 1	20	Problem 5	20
Problem 2	20	Problem 6	8
Problem 3	16		
Problem 4	16		
	Total	**100**	

Instructions:

1. This is a 2-hr exam. Closed book and notes
2. If a description to an algorithm or a proof is required please limit your description or proof to within 150 words, preferably not exceeding the space allotted for that question.
3. No space other than the pages in the exam booklet will be scanned for grading.
4. If you require an additional page for a question, you can use the extra page provided within this booklet. However please indicate clearly that you are continuing the solution on the additional page.
5. Do not detach any sheets from the booklet. Detached sheets will not be scanned.
6. If using a pencil to write the answers, make sure you apply enough pressure so your answers are readable in the scanned copy of your exam.
7. Do not write your answers in cursive scripts.

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

[TRUE/FALSE]
Dynamic programming only works on problems with non-overlapping subproblems.

[TRUE/FALSE]
If a flow in a network has a cycle, this flow is not a valid flow.

[TRUE/FALSE]
Every flow network with a non-zero max s-t flow value, has an edge e such that increasing the capacity of e increases the maximum s-t flow value.

[TRUE/FALSE]
A dynamic programming solution always explores the entire search space for all possible solutions.

[TRUE/FALSE]
Decreasing the capacity of an edge that belongs to a min cut in a flow network always results in decreasing the maximum flow.

[TRUE/FALSE]
Suppose f is a flow of value 100 from s to t in a flow network G. The capacity of the minimum $s-t$ cut in G is equal to 100.

[TRUE/FALSE]
One can **efficiently** find the maximum number of edge disjoint paths from s to t in a directed graph by reducing the problem to max flow and solving it using the Ford-Fulkerson algorithm.

[TRUE/FALSE]
If all edges in a graph have capacity 1, then Ford-Fulkerson runs in linear time.

[TRUE/FALSE]
Given a flow network where all the edge capacities are even integers, the algorithm will require at most C/2 iterations, where C is the total capacity leaving the source s.

[TRUE/FALSE]
By combining divide and conquer with dynamic programming we were able to reduce the space requirements for our sequence alignment solution at the cost of increasing the computational complexity of our solution.

2) 20 pts.
Suppose that we have a set of students $S_1, \cdots, S_s$, a set of projects $P_1, \cdots, P_p$, a set of teachers $T_1, \cdots, T_t$. A student is interested in a subset of projects. Each project pi has an upper bound $K(P_i)$ on the number of the students that can work on it. A teacher T_i is willing to be the leader of a subset of the projects. Furthermore, he/she has a upper bound $H(T_i)$ on the number of students he/she is willing to supervise in total. We assume that no two teachers are willing to supervise the same project. The decision problem here is whether there is really a feasible assignment without violating any of the constrains in K and in H.

a) Solve the decision problem using network flow techniques. (15 pts)

The source node is connected to each student with the capacity of 1. Each student is connected to his/her desired projects with a capacity of 1.
Each project is connected to teachers who are willing to supervise it with a capacity of K(P_i). Each teacher is connected to the sink node with the capacity of H(T_i). If there is a max flow that is equal to the number of students, there is a solution to the problem. You can also convert the order of nodes and edges (source->teachers->projects->students->sink) as long as the edges are correct. You can have 2 sets of nodes for projects (inputs and outputs) as long as the output nodes do not constrain the flow.

b) Prove the correctness of your algorithm. (5 pts)

We need to show that
A – If there is a feasible assignment of students and teachers to projects, we can find a max flow of value s (# of students) in G
B – If we find a max flow of value s in G, we can find a feasible assignment of students and teachers to projects.

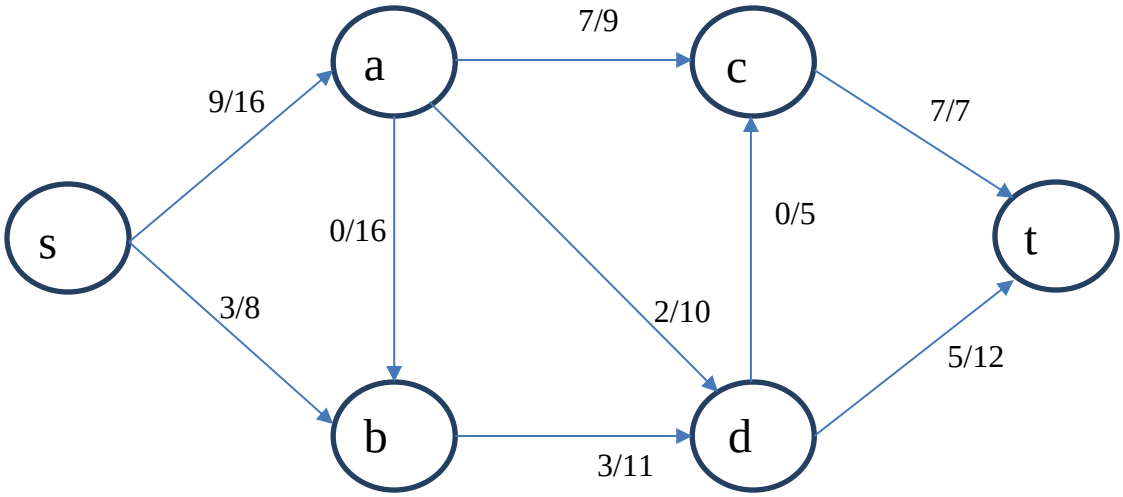
Rubric:
1) incorrect/missing nodes (incorrect nodes * -3)
2) incorrect/missing edges (incorrect edges * -2)
3) extra demand values (extra demands * -1)
4) not mentioning that max flow = number of students (-3)
5) if the proof is only provided for one side (-2)
6) if proof is not sufficient (-1)

3) 16 pts.

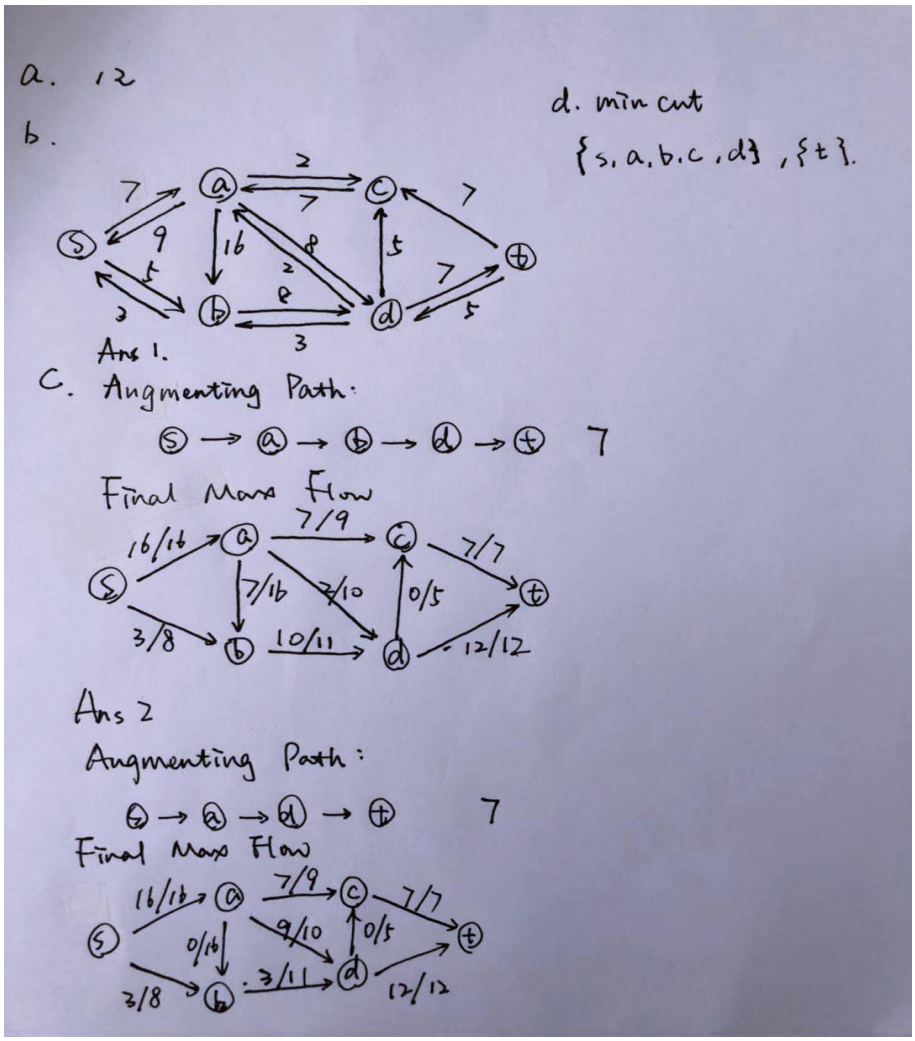
Consider the below flow-network, for which an s-t flow has been computed. The numbers *x/y* on each edge shows that the capacity of the edge is equal to *y*, and the flow sent on the edge is equal to *x*.

a. What is the current value of flow? (2 pts)
b. Draw the residual graph for the corresponding flow-network. (6 pts)
c. Calculate the maximum flow in the graph using the Ford-Fulkerson algorithm. You need to show the augmenting path at each step and final max flow. (6 pts)
d. Show the min cut found by the max flow you found in part c. (2 pts)

Note: extra space provided for this problem on next page



Solution:



Rubric:
a) Incorrect value -> 0 point
b) Every incorrect/missing edge or capacity -> -1point
c) Every possible flow which has a capacity of 19 is correct.
Answers without correct value(other than) 19 will get 0 point.
Answers with correct value/final max flow/final residual graph without augmenting path/detail steps will get 3 points
Answers with correct value/final max flow/final residual graph with augmenting path&detail steps will get 6 points
Answers with a correct augmenting path but incomplete max flow will get 1 point.
d) Incorrect set -> 0 point

4) 16 pts

A symmetric sequence or palindrome is a sequence of characters that is equal to its own reversal; e.g. the phrase "a man a plan a canal panama" forms a palindrome (ignoring the spaces between words).

a) Describe how to use the **sequence alignment algorithm** (described in class) to find a longest symmetric subsequence (longest embedded palindrome) of a given sequence. (14 pts)
Example: CTGACCTAC 's longest embedded palindromes are CTCCTC and CACCAC

Solution: given the sequence S, reverse the string to form S^r . Then find the longest common substring between S and S^r by setting all mismatch costs to ∞ and δ=1 (or anything greater than zero) .

b) What is the run time complexity of your solution? (2 pts)
$O(n^2)$

Rubric 4a)

- 10 points for reversing the string and using the sequence alignment algorithm
  - If the recurrence is given without explicitly mentioning the sequence alignment algorithm is fine.
  - Recurrence is discussed in class
- 2 points for allocating the correct mismatch value.
- 2 points for allocating the correct gap score.
- Any other answer which does not use sequence alignment algorithm is wrong. **The question explicitly asks to use the sequence alignment algorithm.** (0 to 2 points)
  - 2 points for correctly mentioning any other algorithm to get palindrome within a string
- Wrong answers
  - Dividing the string into two and reversing the second half.
    - Will not work, as the entire palindrome can be located within the first or the second half
  - Aligning the original string with the copy of the same string
    - This will return the original string and not the palindrome
  - Any other algorithm which finds the palindrome in a string

Rubric 4b)

2 points

0 points for any other answer

5) 20 pts
Let's say you have a dice with m sides numbered from 1 to m, and you are throwing it n times. If the side with number i is facing up you will receive i points. Consider the variable Z which is the summation of your points when you throw the dice n times. We want to find the number of ways we can end up with the summation Z after n throws. We want to do this using dynamic programming.

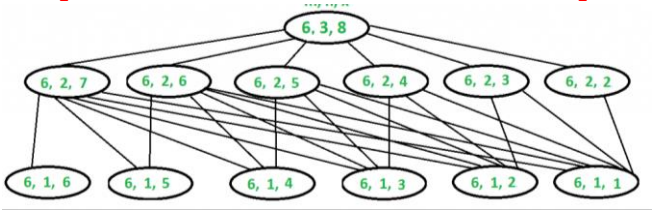
a) Define (in plain English) subproblems to be solved. (4 pts)

OPT(n , Z) = number of ways to get the sum Z after n rolls of dice.

b) Write the recurrence relation for subproblems. (6 pts)

$OPT(n, Z) = \Sigma_{i=1 \text{ to } m} OPT(n-1, Z-i)$

Example: Given, m=6, Calculate: OPT(3,8)
Subproblems are as follows (represented as(m,n,Z):



Rubric:
-6 if not summing over m
-4 if summing over m but OPT is somewhat incorrect.

c) Using the recurrence formula in part b, write pseudocode to compute the number of ways to obtain the value SUM. (6 pts)
Make sure you have initial values properly assigned. (2 pts)

```
OPT(1,i) = 1 for i=1 to MIN(m,SUM)
OPT(1,i) = 0, for i=MIN(m,SUM) to MAX(m,SUM)
For i = 2 to n
     For j = 1 to SUM
          OPT(i, j) = 0
          For k =  1 to MIN (m, i)
               OPT(i, j) = OPT(i, j) + OPT(i-1,j-k)
          Endfor
     Endfor
Endfor
```

```
Return OPT(n,SUM)

Example, Given, m=6, Calculate: OPT(3,8)
OPT Table looks as follows:
```

0	0	0	0	0	0	0	0	0
0	1	1	1	1	1	1	0	0
0	0	1	2	3	4	5	6	5
0	0	0	1	3	6	10	15	**21**

Alternatively, you can initialize OPT(0,0) = 1 and start the iteration of i from 1.

Rubric:
-1 for each initialization missed.
-6 for any incorrect answer. (e.g incorrect recurrence relation, incorrect loops, etc)
i.e. no partial grading for any pseudocode producing incorrect answer.

d) Compute the runtime of the algorithm described in part c and state whether your solution runs in polynomial time or not (2 pts)

Time Complexity: O(m * n * Z) where m is number of sides, n is the number of times we roll the dice, and Z is given sum. This is pseudo polynomial since the complexity depends on the numerical value of input terms.

Rubric:
Not graded if part c is incorrect.
-2 if stated as polynomial time.
-2 if incorrect complexity equation.

6) 8 pts

a) Given a flow network G=(V, E) with integer edge capacities, a max flow *f* in G, and a specific edge *e* in E, design a linear time algorithm that determines whether or not *e* belongs to a min cut. (6 pts)

Solution:
Reduce capacity of e by 1 unit. - O(1)
Find an s-t path containing e and subtract 1 unit of s-t flow on that path. For this, considering e's adjacent nodes to be (u,v), you can run BFS from source to find an s-u path and BFS from v to find an v-t path which gives you an s-u-v-t path crossing e. - O(|V|+|E|)
Then construct the new residual graph $G_{f'}$ - O(|E|)
If there is an augmenting s-t path in $G_{f'}$ then e does not belong to a min cut, otherwise it does.

b) Describe why your algorithm runs in linear time.

You can find this using BFS from source(only one iteration of the FF algorithm) - O(|V| + |E|)
All the above steps are linear in the size of input, therefore the entire algorithm is linear.

Rubric:
If providing the complete algorithm: full points
- If not decreasing the s-t flow by 1: -1 points
- If running the entire Ford-Fulkerson algorithm to find the flow: -2 points
- If providing the overall idea without description of implementation steps: -3 or -4 points depending on your description

If the provided algorithm is not scalable to graphs having multiple(more than two) min-cuts(for example using BFS to find reachable/unreachable nodes from source/sink): -4 points
- Incomplete or incorrect description of the above algorithm (-1 or -2 additional points depending on your description)

If removing the edge and finding max-flow: -3 or -4 points depending on your description (since this algorithm doesn't run in linear time)
If only considering saturated edges to check for min-cut edges: -7 points
Adding the capacity of e and checking the flow: no points (this approach is not checking for min-cut)
Checking all possible cuts in the graph: no points (this takes exponential time)
Other algorithms: no points