INFO 6205 – Algorithms and Data
Practice Quiz -Solutions

Student Name: ____

Professor: Nik Bear Brown

Q1 (5 Points) Match the following definitions:

i. Dynamic Programming
ii. Connected Component
iii. Min-cut
iv. Augmenting path
v. Residual graph

Match the term with the definition.

A. A graph cut whose cut set has the smallest number of edges or smallest sum of weights possible
B. A method for solving complex problems by breaking them down into simpler subproblems. We need to solve different parts of the problem (subproblems), then combine the solutions of the subproblems to reach an overall solution.
C. A path in the residual graph from source to sink with a capacity of at least 1.
D. A graph with edges that represent the difference between an edges capacity and its flow.
E. A component of an undirected graph is a subgraph in which any two vertices are connected to each other by paths, and which is connected to no additional vertices in the super-graph.

Solution:

i-B
ii-E
iii-A
iv-C
v,-D

A. Dynamic Programming

A method for solving complex problems by breaking them down into simpler subproblems. We need to solve different parts of the problem (subproblems), then combine the solutions of the subproblems to reach an overall solution.

B Connected Component

A component of an undirected graph is a subgraph in which any two vertices are connected to each other by paths, and which is connected to no additional vertices in the super-graph.

C Min-cut
A graph cut whose cut set has the smallest number of edges or smallest sum of weights possible."

D. Augmenting path

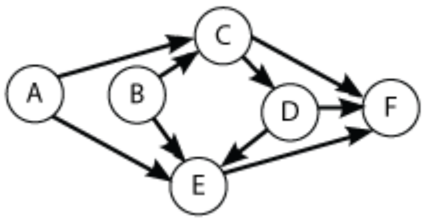
A path in the residual graph from source to sink with a capacity of at least 1.

E. Residual graph

A graph with edges that represent the difference between an edges capacity and its flow.

Q2 (5 Points) Short answer:

2A (3 Points) What is the max-flow from A to F in the directed graph below? All edges are unit capacity.



Solution:

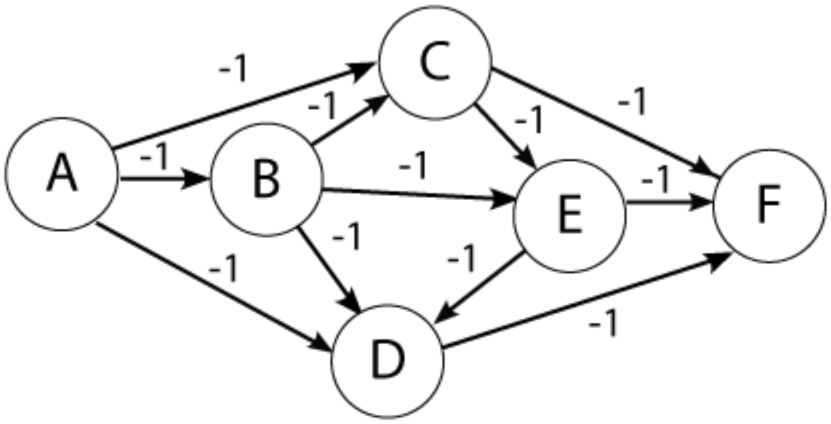
2 by max-flow min-cut

2B (2 Points) What is the role of an *augmenting path* in the *Ford-Fulkerson* algorithm?

Solution:

An augmenting path is an additional possible path for a flow. (It augments the current flow) The augmenting path theorem tells us a flow f is a max flow if and only if there are no new augmenting paths. Ford-Fulkerson stops when no new augmenting paths can be found.

Q3 (5 Points)



Use the Bellman-Ford algorithm to find the shortest path from node A to F in the weighted directed graph above. Select which of the following statements are true. (Multiple choice)

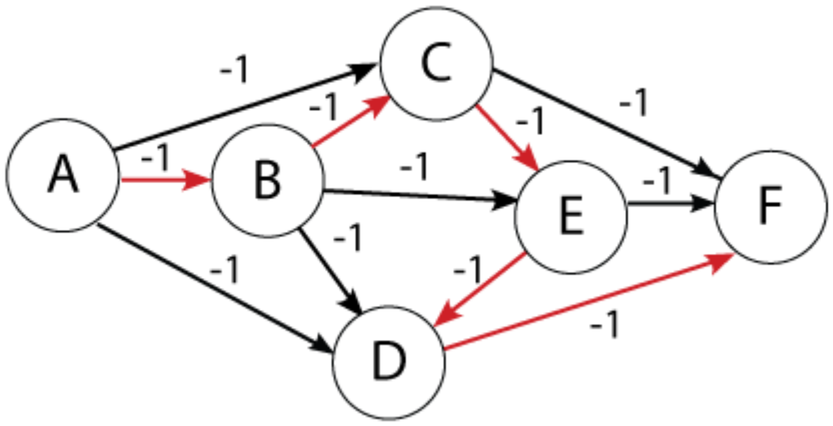
A. The graph has a negative cycle.
B. The lowest cost path to E from A is -3.
C. The lowest cost path to F from A is -4.
D. The lowest cost path to D from A is -2.
E. The lowest cost path to C from A is -2.

Solution:

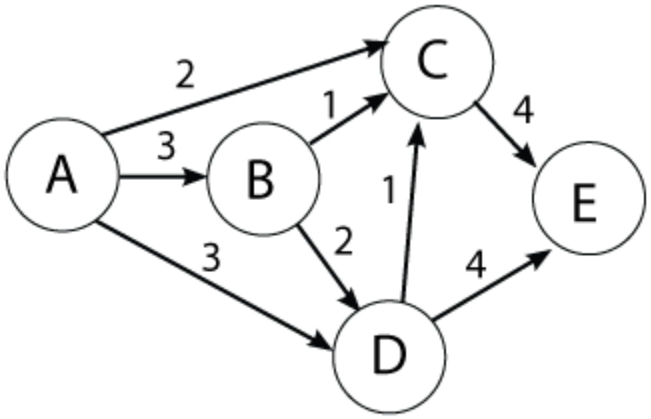
A. The graph has a negative cycle. (F)
B. The lowest cost path to E from A is -3. (T)
C. The lowest cost path to F from A is -4. (F)
D. The lowest cost path to D from A is -2. (F)
E. The lowest cost path to C from A is -2. . (T)

Note, that since all the edge weights are -1, the lowest cost path will have the most edges possible (n-1 or 6 -1) IF there are no negative cycles.
Shortest path: A->B->C->E->D->F at cost -5

	A	B	C	D	E	F
0	0	INF	INF	INF	INF	INF
1	0	-1 (A)	-1 (A)	-1 (A)	INF	INF
2	0	-1 (A)	-2 (B)	-2 (B)	-2 (B)	-2 (D)
3	0	-1 (A)	-2 (B)	-3 (E)	-3 (C)	-3 (C)
4	0	-1 (A)	-2 (B)	-4 (E)	-3 (C)	-4 (D)
5	0	-1 (A)	-2 (B)	-4 (E)	-3 (C)	-5 (D)
6	0	-1 (A)	-2 (B)	-4 (E)	-3 (C)	-5 (D)



Q4 (5 Points)



Use the Ford-Fulkerson algorithm to find the maximum flow from node A to E in the weighted directed graph above. Select which of the following statements are true. (Multiple choice)

A. The path A-D has residual capacity after the Ford-Fulkerson algorithm terminates.
B. The path D-E has residual capacity after the Ford-Fulkerson algorithm terminates.
C. The path D-C has residual capacity after the Ford-Fulkerson algorithm terminates.
D. The path B-C has residual capacity after the Ford-Fulkerson algorithm terminates.
E. The path C-E has residual capacity after the Ford-Fulkerson algorithm terminates.

Solution:

All are false. There is no residual capacity on any edges after the Ford-Fulkerson algorithm terminates.

A. The path A-D has residual capacity after the Ford-Fulkerson algorithm terminates. (F)
B. The path D-E has residual capacity after the Ford-Fulkerson algorithm terminates. (F)
C. The path D-C has residual capacity after the Ford-Fulkerson algorithm terminates. (F)
D. The path B-C has residual capacity after the Ford-Fulkerson algorithm terminates. (F)
E. The path C-E has residual capacity after the Ford-Fulkerson algorithm terminates. (F)

AP stands for Augmenting Path

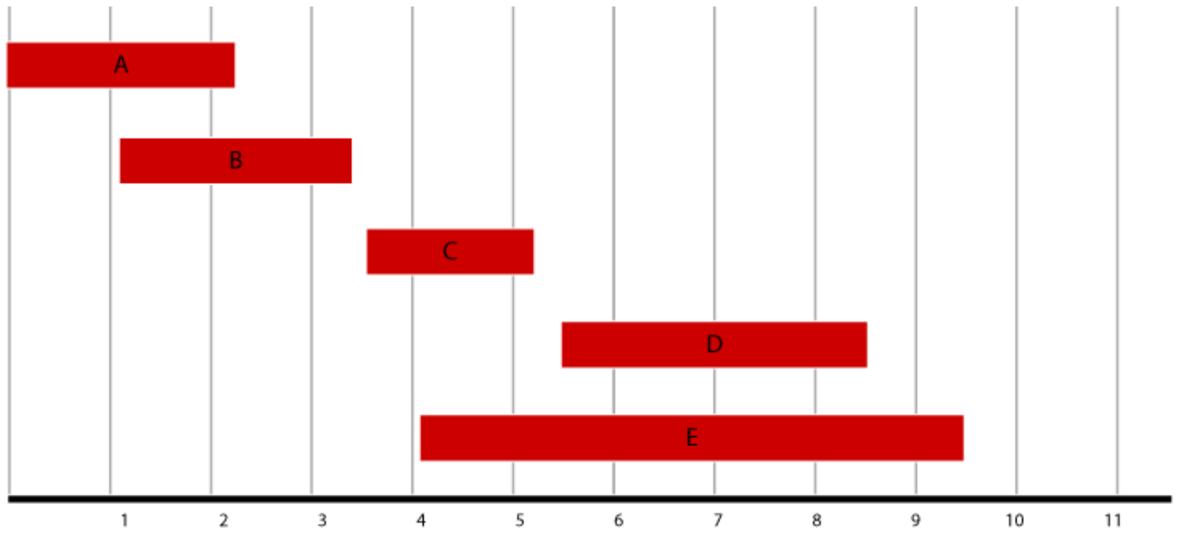
AP1: (A-C: 0-2) -> (C-E: 2-2) Flow 2

AP2: (A-B: 2-1) -> (B-C: 0-1) -> (C-E: 1-3) Flow 1
AP3: (A-B: 1-2) -> (B-D: 1-1) -> (D-C: 0-1) -> (C-E: 0-4) Flow 1
AP4: (A-B: 0-3) -> (B-D: 0-2) -> (D-E: 3-1) Flow 1
AP5: (A-D: 0-3) -> (D-E: 0-4) Flow 3

No more augmenting paths. Max-Flow 8

Q5 (5 Points)

Given the five intervals below, and their associated values; select a subset of non-overlapping intervals with the maximum combined value. Use dynamic programming.



Interval	Value
A	1
B	2
C	1
D	2
E	1

Select which of the following statements are true. (Multiple choice)

A. {B C, D} is the set with maximum combined value.
B. If we only considered the intervals A and B the max value would be 2.
C. The maximum combined value is 6.
D. If we only considered the intervals A through D the max value would be 5.
E. Intervals C and E both may be in the max set if the value of E where 9.

Solution:

A. {B C, D} is the set with maximum combined value. (T)
B. If we only considered the intervals A and B the max value would be 2. (T)
C. The maximum combined value is 6. (F)
D. If we only considered the intervals A through D the max value would be 5. (T)
E. Intervals C and E both may be in the max set if the value of E where 9. (F)

Interval	A	B	C	D	E
Value	1	2	3	5	1
Previous	-	-	B	C	B
Max	1	2	2	3	3

Interval	Max Value
A	Max(1+0,0)
B	Max(2+0,1)
C	Max(1+2,2)
D	Max(2+3,3)
E	Max(1+2,5)

Creating Max Set:

S={}

For all intervals I from highest to lowest:

```
Trace(i):

  If Value[i] + Max[Previous] ≥ Value [i -1]:
    S= S + I + Trace (Previous of i):
else:  Trace(i-1)
```

Steps:

Interval	Trace(i)	S
E	1+2 < 5	
D	2+3 ≥ 5 Jumped to Trace(C)	D
C	1+2 ≥ 3	C
B	2+0 ≥ 2	B
A	Jumped to Trace(-)	

S= {B C, D}

Q6 (5 Points) Do stable matchings always exist for the Stable Roommate Problem? Give a proof that they do or don't.

Solution:

This comes directly from the Kleinberg and Tardos slides "Chapter 1 Introduction: Some Representative Problems."

Q. Do stable matchings always exist?
A. Not obvious a priori.

Stable roommate problem.

- 2n people; each person ranks others from 1 to 2n-1.
- Assign roommate pairs so that no unstable pairs.

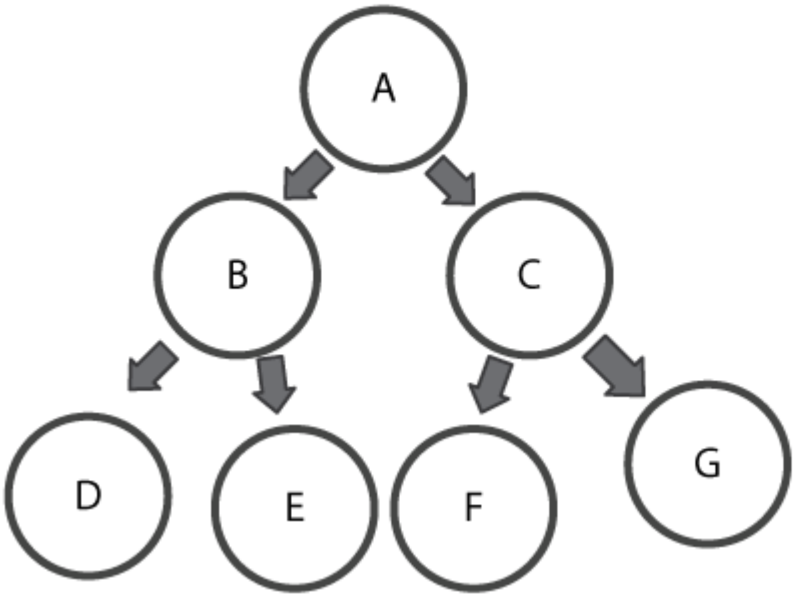
Observation. Stable matchings do not always exist for stable roommate problem.
Prove by counter example (see below or see https://en.wikipedia.org/wiki/Stable_roommates_problem)

	1st	2nd	3rd
Adam	B	C	D
Bob	C	A	D
Chris	A	B	D
Doofus	A	B	C

A-B, C-D $\Rightarrow$ B-C unstable
A-C, B-D $\Rightarrow$ A-B unstable
A-D, B-C $\Rightarrow$ A-C unstable

Q7 (5 Points)

Topologically sort the DAG below:



Select which of the following valid topological orderings? (Multiple choice)

A. {A,B,C,D,E,F,G}
B. {A,B,C,E, D,F,G}
C. {A,C,B,D,E,F,G}
D. {B,C,A,D,E,F,G}
E. { G,D,E,F,A,B,C}

Solution:

A. {A,B,C,D,E,F,G} (T)
B. {A,B,C,E, D,F,G} (T)
C. {A,C,B,D,E,F,G} (T)
D. {B,C,A,D,E,F,G} (F)
E. { G,D,E,F,A,B,C} (F)

```
L ← Empty list that will contain the sorted elements
S ← Set of all nodes with no incoming edges
**while** S is non-empty **do**
    remove a node n from S
    add n to *tail* of L
    **for each** node m with an edge *e* from n to m **do**
        remove edge e from the graph
        if m has no other incoming edges then
            insert m into S
if graph has edges then
    return error (graph has at least one cycle)
else
    return L (a topologically sorted order)
```

L={}
Remove A (no incoming edges)   L={A}
Remove B (no incoming edges)   L={A,B}
Remove C (no incoming edges)   L={A,B,C}
Remove D (no incoming edges)   L={A,B,C,D}
Remove E (no incoming edges)   L={A,B,C,D,E,}
Remove F (no incoming edges)   L={A,B,C,D,E,F}
Remove G (no incoming edges)   L={A,B,C,D,E,F,G}
Done

L={A,B,C,D,E,F,G}

Note there are many valid topologically sorted orders (e.g. L={A, C,B D,E G, F})

Q8 (5 Points)

Given the weights and values of the four items in the table below, select a subset of items with the maximum combined value that will fit in a knapsack with a weight limit, *W,* of 6. Use dynamic programming. _Show your work._

Item $_i$	Value v_i	Weight w_i
1	3	3
2	2	2
3	4	3
4	3	2

Capacity of knapsack W=6

Select which of the following statements are true. (Multiple choice)

A. Either {1,3} OR {3,4} is a subset of items with the maximum combined value.
B. The maximum combined value is 8.

C. Item 3 will be in the subset of items with the maximum combined value.
D. The maximum combined value of items 1, 2 and 3 is 7 if the capacity of knapsack is W=6
E. The maximum combined value of items 1, and 2 is 5 if the capacity of knapsack is W=6

Solution:

A. Either {1,3} OR {3,4} is a subset of items with the maximum combined value. (T)
B. The maximum combined value is 8. (F)
C. Item 3 will be in the subset of items with the maximum combined value. (T)
D. The maximum combined value of items 1, 2 and 3 is 7 if the capacity of knapsack is W=6 (T)
E. The maximum combined value of items 1, and 2 is 5 if the capacity of knapsack is W=6 (T)

Capacity of knapsack W=6
Algorithm: given two arrays w[3, 2, 3, 2] and v[3, 2, 4, 3]:

```
for $I$ ← 1 to $n$:
    for $x$ ← 1 to $w$:
        if $w[I] > x$:
            OPT[$I, x$] ← OPT[$i – 1, x$]
        else
            OPT[$I, x$] ← max(OPT[$i – 1, x$]; OPT[$i – 1, x – w[i]$] + $v[i]$])
```

6	3	5	7 <--A Note 7	7 ? <--B
5	3	5	6	7 <--Note 7
4	3	3	4 <--B	5
3	3 <--A	3 ? <--	4	4
2	0	2	2	3
1	0	0 <--B	0	0
	1	**2**	**3**	**4**

We used items 1,3 (trace back A) or used 3,4 (trace back B) for a combined value of 7 in the knapsack. Note we can get to 7 by taking or not taking item 4.

S = {1,3} OR {3,4}

Q9 (5 Points)

Master Theorem: For each of the following recurrences, give an expression for the runtime T(n) if the recurrence can be solved with the Master Theorem. Otherwise, indicate that the Master Theorem does not apply.

i. $T(n) = 3T(n/2) + n$

ii. $T(n) = 2T(n/2) + n$

iii. $T(n) = n^3T(n/2) + n$

iv. $T(n) = 8T(n/3) + n^3$

v. $T(n) = 27T(n/3) + n^3$

Match the θ with the recurrence

A. $\theta(n^3)$
B. Does not apply
C. $\theta(n^{\log 3})$
D. $\theta(n^3 \log n)$
E. $\theta(n \log n)$

Solution:

i- $\theta(n^{\log 3})$ which is C
Ii- $\theta(n \log n)$ which is E
Iii- Does not apply which is B
Iv- $\theta(n^3)$) which is A
v- $\theta(n^3 \log n)$ which is D

i. $T(n) = 3T(n/2) + n \Longrightarrow T(n) = \Theta(n \lg 3)$ (Case 1)

$k = \log_b a = \log_2 3 = \log(3)/\log(2) > 1$
f(n) is $\theta(n^{\log_b a})$ which is $\theta(n^{\log 3})$
Therefore the runtime T(n) is $\theta(n^{\log 3})$

ii. $T(n) = 2T(n/2) + n$

$a=b=2$; $k = \log_b a = \log_2 2 = \log(2)/\log(2) = 1$
Case 2 again. $p=0$. $f(n) = n^2$ which is $\theta(n)$
Therefore the runtime T(n) is $\theta(n \log n)$

iii. $T(n) = n3T(n/2) + n$

Does not apply (a is not constant)

.Iv. $T(n) = 8T(n/3) + n^3$ (Case 3)

$k = \log_b a = \log_3 8 = \log(8)/\log(3) < 3$
f(n) is $\theta(n^3)$
Therefore the runtime T(n) is $\theta(n^3)$

v. $T(n) = 27T(n/3) + + n^3$

$k = \log_b a = \log_3 27 = \log(27)/\log(3) = 3$
Case 2 again. $p=0$. $f(n) = n^3$
Therefore the runtime $T(n)$ is $\theta(n^3 \log n)$

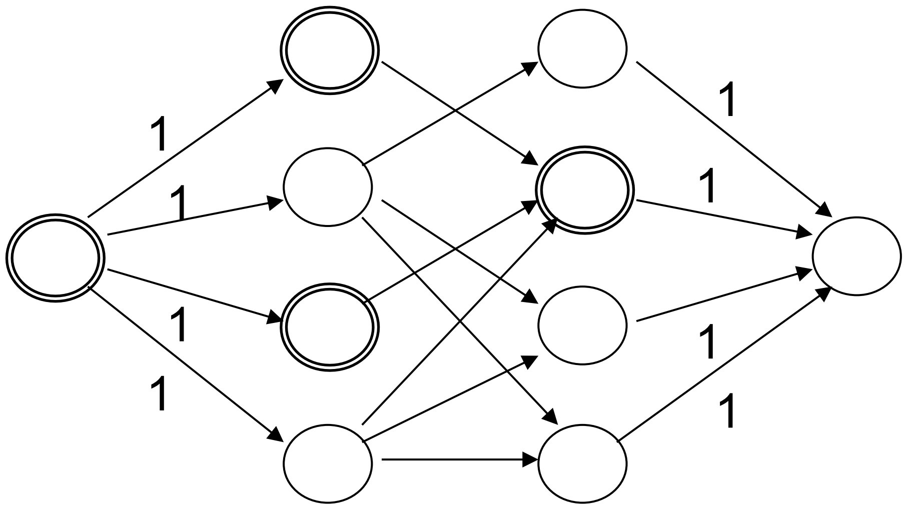
Q10 (10 Points) **The bipartite matching** problem which is defined as follows: Given a bipartite graph with two sets of vertices L and R and some edges E between L and R, produce the largest set of pairings such that a vertex in L may be paired with **at most one** other vertex in R.

Can max-flow be used to solve this problem? If so, how?

Type in your answer.

Solution:

- Connect a source vertex to each vertex in L and a sink vertex to each vertex in R. Make edges directional from source to sink.
- Have weights of 1 on each edge from source to left vertices, weight of 1 on each edge from left to right vertices and weights of 1 on each edge from right vertices to sink
- Run the Ford-Fulkerson algorithm



Unit or infinite capacity edges between sets L and R

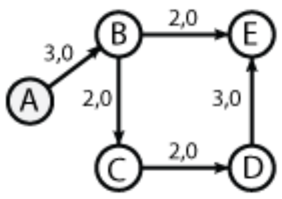
Max flow formulation (from K&T slides)

i. Create digraph $G' = (L \cup R \cup \{s, t\}, E')$.
ii. Direct all edges from L to R, and assign infinite (or unit) capacity.

iii. Add source s, and unit capacity edges from s to each node in L.
iv. Add sink t, and unit capacity edges from each node in R to t.

Q11 (5 Points) Short answer:

Use the Preflow-Push maximum flow algorithm to find the maximum flow from node A to E in the weighted directed graph below.



Note: Forward, backward (e.g. 3,0) represents the forward and backward flow.
You must run push–relabel until B pushed a pre-flow from B to either C or E.

Answer the following:

What is the initial height of A? (5 points)
What is the max-flow? (5 points)
Is the height of B ever greater than the height of A? (5 points)

Solution:

What is the initial height of A? 5
What is the max-flow? 2
Is the height of B ever greater than the height of A? Yes, it must eventually push 1 back to the source A.

Q12 (10 Points)

Let *G* = *(V*, *E)* be a directed graph, and suppose that for each node *v*, the number of edges into *v* is equal to the number of edges out of *v*. There are no nodes without edges.

A (5 Points) Let *x*, *y* be two nodes of *G*, if the capacity of each edge is 1, is there an algorithm to detect how many mutually edge disjoint paths exist between x and y? Why?

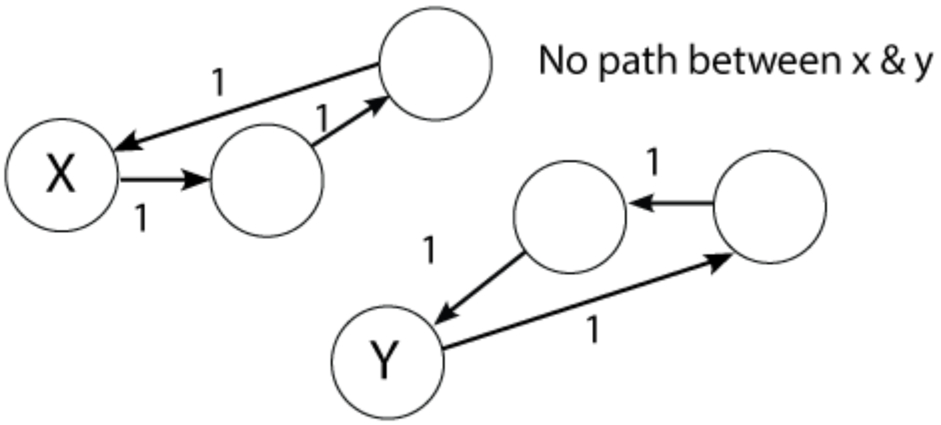
B (5 Points) Let *x*, *y* be two nodes of *G*, if the capacity of each edge is 1, and that suppose that the max-flow is *k,* how many mutually edge disjoint paths exist between x and y? Why?

Type your answer.

Solution:

A (10 Points) Let x, y be two nodes of G, if the capacity of each edge is 1, how many mutually edge disjoint paths exist between x and y? Why?

The number of mutually edge disjoint paths will depend on the graph. There could be none (see example below) up to the max-flow between x and y. FULL CREDIT was given if you realized the number of mutually edge disjoint paths equaled the max-flow and that we didn't know what that flow was.



B (10 Points) Let *x*, *y* be two nodes of *G*, if the capacity of each edge is 1, and that suppose that the max-flow is *k,* how many mutually edge disjoint paths exist between x and y? Why?

This question is basically asking you to recall the mutually edge disjoint path max-flow formulation in the K&T slides.

If the max-flow is *k,* then the max number edge-disjoint x-y paths equals max flow value.

Pf. $\leq$

- Suppose there are k edge-disjoint paths $P_1, \ldots, P_k$.
- Set f(e) = 1 if e participates in some path P_i ; else set f(e) = 0.
- Since paths are edge-disjoint, f is a flow of value k. ▪

Q13 (5 Points)

Arrange the following functions in increasing order of growth:

- $55n$
- $3n^3$
- 0.0001^n
- $2n^2$
- $n^2 \log(n)$
- $\frac{n}{2}$
- $n\sqrt{n}$

- 5^n

Solution:

Polynomial time:

$\frac{n}{2}$, 55n, n √n, $2n^2$, $3n^3$

Note: n √n is the same as $n^{1.5}$

The $n^2 \log(n)$ is between n^2 and n^3

Exponential time:

0.0001^n(decreasing), 5^n (increasing)

Final order:

0.0001^n, $\frac{n}{2}$, 55n, n √n, $2n^2$, $n^2 \log(n)$, $3n^3$, 5^n

Q14 (5 Points)
Write a recurrence relation for Quick Sort best case as described below. If the recurrence can be solved with the Master Theorem, use it to provide the runtime. Otherwise, indicate that the Master Theorem does not apply.

Quick Sort Pivot Algorithm

Based on our understanding of partitioning in quick sort, we will now try to write an algorithm for it, which is as follows.

```
Step 1 − Choose the highest index value has pivot
Step 2 − Take two variables to point left and right of the list
excluding pivot
Step 3 − left points to the low index
Step 4 − right points to the high
Step 5 − while value at left is less than pivot move right
Step 6 − while value at right is greater than pivot move left
Step 7 − if both step 5 and step 6 does not match swap left and
right
Step 8 − if left ≥ right, the point where they met is new pivot
```

Solution:

Quick Sort best case splits in to two equal sets every time.

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

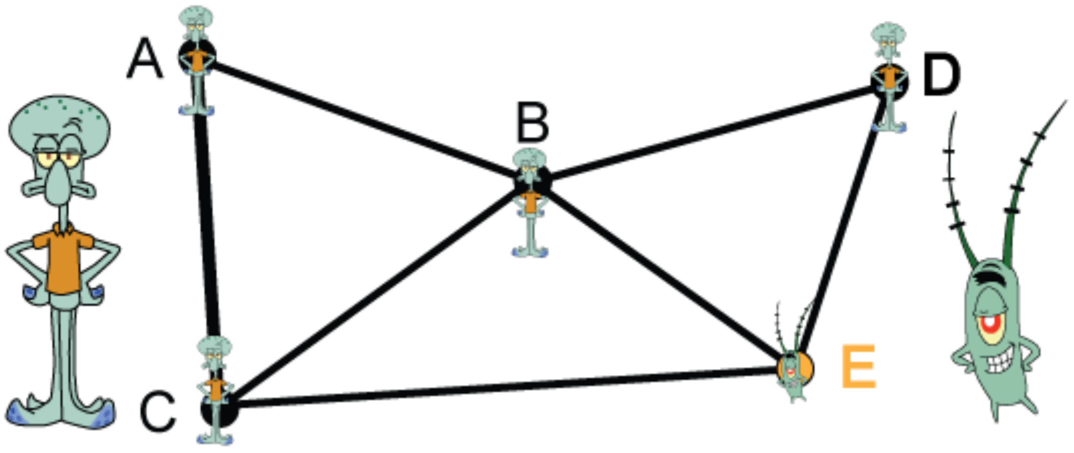
a=2, b=2; $\log_b a=\log_2(2)=1$ f(n) = n^1 which is θ($n^{\log ba}$) θ(n^1)

(CASE B) $f(n)$ and $n^{\log_b a}$ are same size

In Case B or θ($n^{\log_b a} \log^{p+1} n$) or θ($n^1 og^{0+1} n$) or θ(n $\log^1$ n)

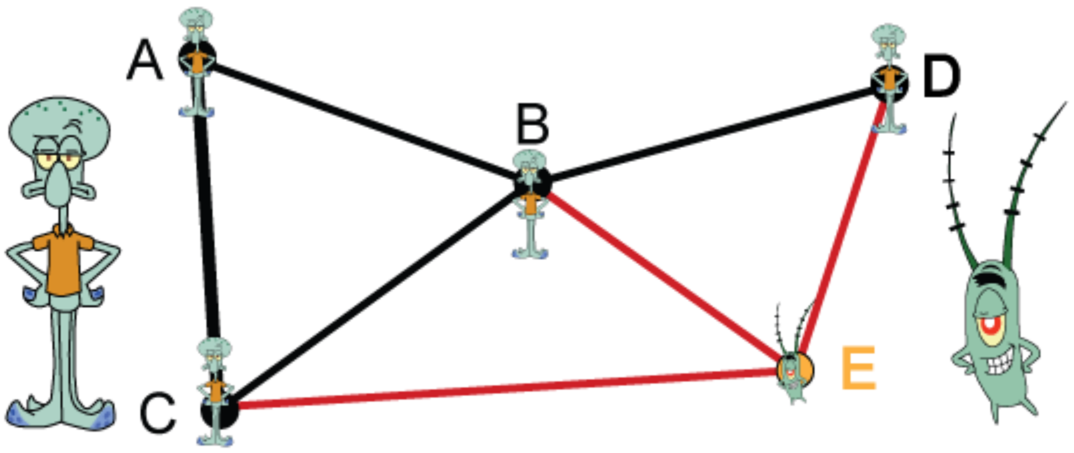
Therefore the runtime T(n) is θ(n log n)

Q15 (5 Points) The graph below is composed of two sets (A,B,C,D) (i.e. those vertices with Squidward) and (E) (i.e. those with Sheldon James Plankton). What is the cut set of the graph?



Solution:

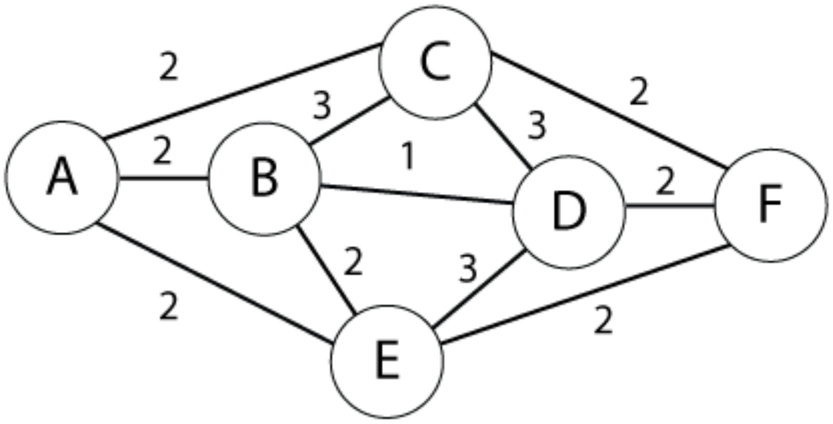
The cut set is (D-E, B-E, C-E)



A cut set is a set of edges whose end points are in different subset of the partition. Edges are said to be crossing the cut if they are in its cut-set. A set of edges of a graph, which, if removed (or "cut"), disconnects the graph (i.e., forms a disconnected graph).

Q16 (5 Points)

Use Kruskal's algorithm to find a minimum spanning tree for the connected weighted graph below. *Show your work.*



Solution:

Kruskal's algorithm pseudocode:

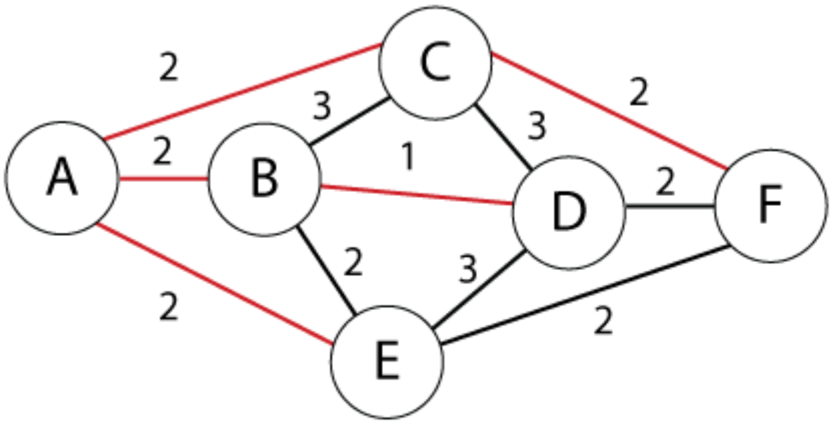
```
A.    Create a forest T (a set of trees), where each vertex in the graph is a separate tree
B.    Create a set S containing all the edges in the graph
C.    Sort edges by weight

While S is nonempty and T is not yet spanning:
   I.    remove an edge with minimum weight from S
   II.   if that edge connects two different trees, then add it to the forest, combining two trees into a
         single tree, otherwise discard that edge.
```

Steps:

- I. Connect B-D (1)
- II. Connect A-C (2) (Note, many choices at weight 2)
- III. Connect A-B (2)
- IV. Connect A-E (2)
- V. Connect C-F (2)

Stop. MST formed (5 edges, 6 vertices)

MST = {B-D, A-C, A-B, A-E, C-F} or many others as long as MST weight is 9
MST weight = 1+2+2+2+2=9



Kruskal's algorithm is O(E log V) time. Where E is the set of edges and V is the set of vertices.

Q17 (10 Points)

Suppose you are given a connected graph G, with edge costs that are all distinct. Prove that G has a unique minimum spanning tree.

Solution:

Suppose by way of contradiction that T and T' are two distinct minimum spanning trees of G. Since T and T' have the same number of edges, but are not equal, there is some edge e' in T' but not in T. If we add e' to T, we get a cycle C. Let e be the most expensive edge on this cycle. Then by the Cycle Property, e does not belong to any minimum spanning tree, contradicting the fact that it is in at least one of T or T'.