Name:	Mauro Vargas
ID:	107212593

CSCI 3104, Algorithms **Escobedo & Jahagirdar**

Final Exam Summer 2020 (60 points) **Summer 2020, CU-Boulder**

`Final/dont_panic.png`

Advice 1: For every problem in this class, you must justify your answer: show how you arrived at it and why it is correct. If there are assumptions you need to make along the way, state those clearly.

Advice 2: Verbal reasoning is typically insufficient for full credit. Instead, write a logical argument, in the style of a mathematical proof.

Honor code: On my honor as a University of Colorado at Boulder student, I have neither given nor sought unauthorized assistance in this work

Initials	MV
Date	07/22/2020

If you violate the CU **Honor Code**, you will receive a 0.

Name: Mauro Vargas

ID: 107212593

CSCI 3104, Algorithms **Escobedo & Jahagirdar**

Final Exam Summer 2020 (60 points) **Summer 2020, CU-Boulder**

Instructions for submitting your solution:

- The solutions **should be typed**, we cannot accept hand-written solutions. Here's a short intro to **Latex**.
- In this homework we denote the asymptomatic *Big-O* notation by $\mathcal{O}$ and *Small-O* notation is represented as o.
- We recommend using online Latex editor **Overleaf**. Download the **.tex** file from Canvas and upload it on overleaf to edit.
- You should submit your work through **Gradescope** only.
- If you don't have an account on it, sign up for one using your CU email. You should have gotten an email to sign up. If your name based CU email doesn't work, try the identikey@colorado.edu version.
- Gradescope will only accept **.pdf** files (except for code files that should be submitted separately on Canvas if a problem set has them) and **try to fit your work in the box provided**.
- You cannot submit a pdf which has less pages than what we provided you as Gradescope won't allow it.

Name: Mauro Vargas

ID: 107212593

CSCI 3104, Algorithms **Escobedo & Jahagirdar**

Final Exam Summer 2020 (60 points) **Summer 2020, CU-Boulder**

1. (5 pts) You are given a list of jobs with start and end times. Assume that only one job can be executed at a time on a processor. Your task is to find out the minimum number of processors required so that the jobs are executed as soon as they arrive (no waiting time for a process). The input to your algorithm will be a list of tuples indicating the start and end times of the respective jobs. The output should be a number indicating the minimum number of processors required. Your algorithm should have a runtime of $\mathcal{O}(n\log(n))$, where n is the number of jobs. Assume you do not have access to any auxiliary functions.

   Example
   Input: $(2,10),(9,11),(15,18),(3,4),(17,19),(5,13)$
   Output: 3
   Explanation: Between the interval $(9,10)$ it can be seen that there will be 3 { (2, 10) , (9, 11), (5, 13)} jobs that need to be executed simultaneously. Thus at the very least 3 processors are required.

   (a) (4 pts) Write down well commented pseudo-code or paste real code to solve the above problem.

```
def f(A):
maxOverlap = 0 # variable to store the maximum
count = 0
B = [] # store start time and end time
for i in range(len(A)):#store start and end times into array B
    B.append([A[i][0], 'start'])
    B.append([A[i][1], 'end'])
#sort the B array with Randomized quicksort to get avg nlog(n)
B = Randomized_Quicksort(B)
for i in range(len(B)): # count number of overlaps
#we increase count if start occur it means a new range start added so
    if (B[i][1] == 'start'):
        count = count + 1
#we decrease count if end occur it means a range is ended
    if (B[i][1] == 'end'):
        count = count - 1
    maxOverlap = max(maxOverlap, count)#updating the value
return maxOverlap
```

```
array = [ (2, 10), (9,11), (15, 18), (3, 4), (17, 19), (5, 13)]
C = f(array)
print(C)
```

(b) (1 pts) Explain your algorithms runtime and space complexity by analyzing your code. For example, stating that a sorting algorithm runs in $\mathcal{O}(n\log(n))$ without any justification is insufficient.

We Implement QuickSort using random pivoting to achieve the average case consistently. QuickSort is like merge sort and applies the divide-and-conquer.

Name: Mauro Vargas

ID: 107212593

CSCI 3104, Algorithms **Escobedo & Jahagirdar**

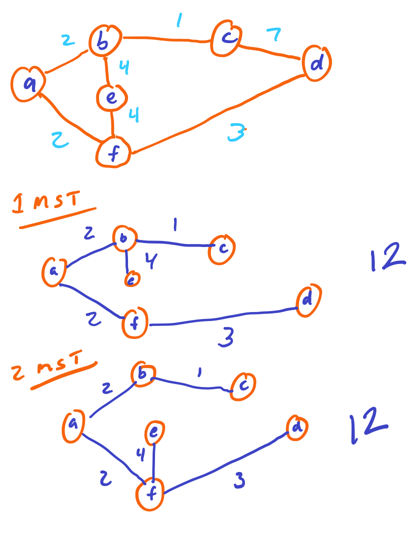
Final Exam Summer 2020 (60 points) **Summer 2020, CU-Boulder**

2. (5 pts) You are designing a network, to connect several offices in different locations. You can contact your network cable provider about the cost of connecting any pair of offices. Your task is to design an algorithm, so that all offices are connected and the cost associated with building the network is minimized.
The input to your algorithm is a Graph in the form of an adjacency list or matrix. The nodes in this graph represent the offices and edge weights represent the cost associated with connecting a pair of offices. The output of the algorithm will be a list of office pairs such that the total connection cost is minimized.

   (a) (2 pts) Provide an example with at least 3 offices and 3 connections. Your example must include at least 2 unique solutions.

Problem 2 a

Tuesday, July 21, 2020 4:05 PM



(b) (3 pts) Provide well commented pseudo code or actual code to solve the above problem.

Kruskal's algorithm considers each edge in sorted order by weight. An arrow points to the edge under consideration at each step of the algorithm. If the edge joins two distinct trees in the forest, it is added to the forest, thereby merging the two trees.

```
MST-KRUSKAL(G,W)
A = [] Empty array
for each vertex v ∈ G.V
MAKE-SET(V)
sort the edges of G.E into non decreasing order by weight w
for each edge(u,v) ∈ G.E  taken in non decreasing order by weight
if FIND-SET(u) ≠ FIND-SET(v)
A = A ∩ FIND-SET(v)
Union(u,v)
return A
```

Name: Mauro Vargas

ID: 107212593

CSCI 3104, Algorithms **Escobedo & Jahagirdar**

Final Exam Summer 2020 (60 points) **Summer 2020, CU-Boulder**

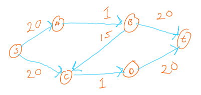
3. (10 pts) Consider the graph $G = (V, E)$ with edge capacities $c(e)$, $\forall e \in E$, the edge capacity represents the maximum amount of flow that can pass through the edge. In addition to edge capacities, the above graph has vertex capacities $c(v)$, $\forall v \in V - \{s, t\}$ (The source and the sink vertex do not have vertex capacities). Each vertex capacity represents the maximum amount of flow that can pass through the vertex.

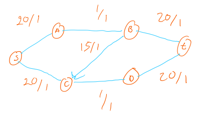
   (a) (4 pts) Given a source vertex $s \in V$ and a sink $t \in V$. How will you modify the given graph G, so that you can find the max-flow from s to t with a known algorithm? Also, show an example input and modified graph with at least 5 vertices. This must be a drawing!

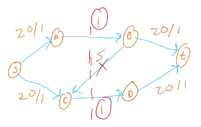
Problem 3a

Max Flow - min-cut Theorem: The value of the max flow equals the capacity of min cut.

$C_f(U \to V) = (c(U \to V) - f(U \to V) + f(V \to U) = 0$







$|f| = c(A \to B) + \cancel{c} c(C \to D$

$|f| = 1 + 1$

$|f| = 2$ max flow

(b) (4 pts) Provide an algorithm to find the max-flow in the above graph from s to t.

```
FORDFULKERSON(G,S,T)
#set all flows to zero
for each edge(u,v) in G.E
(u,v).f = 0
#search algorithm
while there exists a path p from s to t in the residual network G_f
c_f(p) = min{c_f(u,v) : (u,v} is in p } #forest
for each edge(u,b) in p #loop through path
#update weights in G and G_f
if(u,b) in E
(u,v) in E
(u,v).f = (u,v).f + c_f(p) #forware
(u,v).f = (v,s).f - c_f(p) #backwards
```

(c) (2 pts) Find the max-flow in your example graph by stepping though your algorithm. Provide a new graph image each time the flow though an edge is altered.

see Problem 3a picture

4. (10 pts) There are n rooms numbered from $\{1, 2, ..n\}$ in Williams Village. Each of the rooms has a teleportation device with a value $v_i \quad \forall i \in \{1, 2, ...n\}$. The teleportation device placed in the room i with value v_i teleports to a room numbered $(v_i \ \% \ n) + 1$.

   A room in Williams Village is said to be beautiful if you can get back to the room you started using the teleportation devices. The task is to count the number of beautiful rooms in Williams Village.
   The input to your algorithm is the list of values associated with the teleportation device placed in the rooms. The output should be a number indicating the number of beautiful rooms.

   Example 1
   Input: [2, 3, 4, 5]
   Output: 4
   Explanation: All the 4 rooms that are beautiful and the corresponding paths to get back to them are as follows.

   - **Room 1** The teleportation path for Room1 is Room1, Room3, and back to Room1
   - **Room 2** The teleportation path for Room2 is Room2, Room4, and back to Room2
   - **Room 3** The teleportation path for Room3 is Room3, Room1, and back to Room3
   - **Room 4** The teleportation path for Room4 is Room4, Room2, and back to Room4

   Example 2
   Input: [1, 2, 3, 6]
   Output: 2
   Explanation: It can be seen that only Room3 and Room4 are beautiful.

Name: Mauro Vargas

ID: 107212593

CSCI 3104, Algorithms **Escobedo & Jahagirdar**

Final Exam Summer 2020 (60 points) **Summer 2020, CU-Boulder**

(a) (2 pts) Give a high-level explanation of how you plan to solve this problem. (How would you explain your potential solution to a younger sibling or someone who has never taken a computer science class?) Minimum 1 paragraph.

first search colors vertices during the search to indicate their state. Each vertex is initially white, is grayed when it is discovered in the search, and is blackened when it is finished, that is when its adjacency list has been examined completely. DFS search has time stamps each vertex. Each vertex v has two timestamps: the first timestamp v.d records when v is first discovered (and grayed), and the second timestamp v.f records when the search finishes ecamining v's adjacency list (and blackens v)

Name: Mauro Vargas

ID: 107212593

(b) (5 pts) Provide well commented pseudo code or actual code to solve the above problem.

```
DFS(G)
for each vertex u in G.V
u.color = white
u.pi = Nil
time = 0
for each vertex u in G.V
if u.color == white
DFS-VISIT(G,u)

def DFS-Visit(u)
time = time + 1 # white vertex u has just been discovered
u.d = time
u.color = gray
for each v in G.Adj[u] # explore edge (u,v)
if v.color == White
v.pi = u
DFS-Visted(G,v)
u.color = Black #blacken u; it is finished
time = time + 1
u.f = time
```

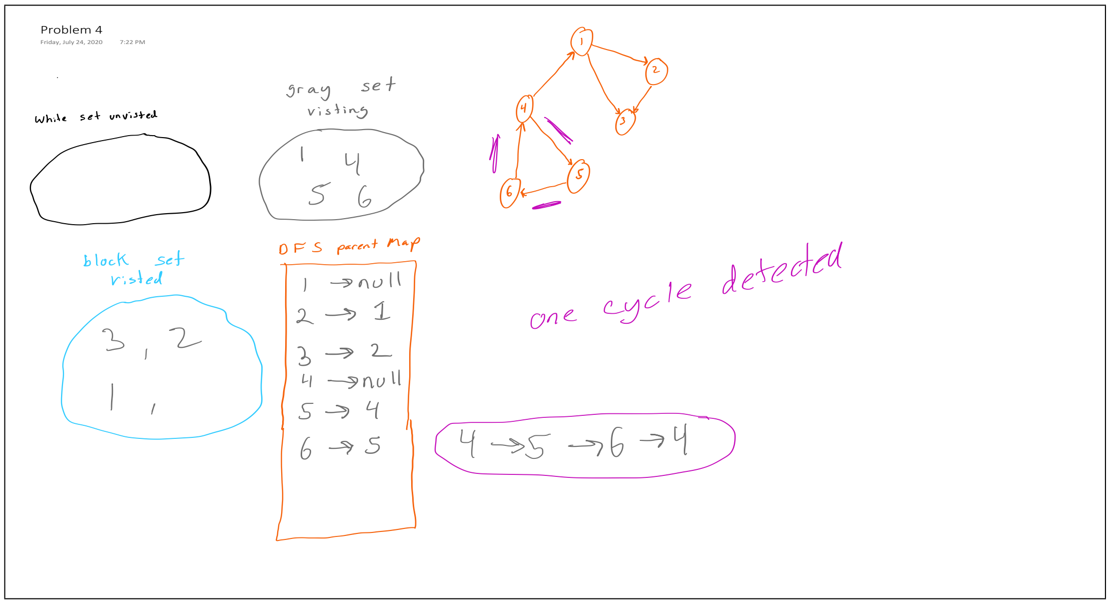
Name: Mauro Vargas

ID: 107212593

CSCI 3104, Algorithms **Escobedo & Jahagirdar**

Final Exam Summer 2020 (60 points) **Summer 2020, CU-Boulder**

(c) (3 pts) Give an example input with at least 5 rooms and structure of how your algorithm explores the rooms. This must be an image! Your input can **not** be an example already given.



Name: Mauro Vargas

ID: 107212593

CSCI 3104, Algorithms **Escobedo & Jahagirdar**
Final Exam Summer 2020 (60 points) **Summer 2020, CU-Boulder**

5. (10 pts) You are at your home in Boulder enjoying the summer and your friend challenges you to ride your bike for at least s miles starting from your home and ending at any of the scenic outlooks. You have a map of Boulder that has information about roads connecting various view points and your house. The task is to determine if there is a sequence of scenic outlooks that you can travel to starting from your house such that you have travelled at least s miles. You can not travel on the same edge more than 1 time and once you visit an outlook you cannot travel to it again.
The input to your problem is a graph $G = (V, E)$ in the form of an adjacency list or adjacency matrix and the source vertex s corresponding to your house. Your algorithm should output a boolean value indicating if it's possible to complete the challenge of riding at least s miles starting from your house.

   (a) (2 pts) Give a high-level explanation of how you plan to solve this problem. (How would you explain your potential solution to a younger sibling or someone who has never taken a computer science class?) Minimum 1 paragraph.

   We will first visit every neighborhood and discover how many miles for every different path, note to always take the minimum distance. If a neighborhood is reachable we will mark a 't' on the map else we will put a 'f' depending on how many 's' miles we are limited to. After we have traveled to all the neighborhood we can check our map and figure out which neighborhoods are possible to reach with 's' miles.

Name: Mauro Vargas

ID: 107212593

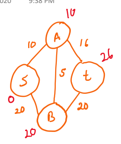
CSCI 3104, Algorithms **Escobedo & Jahagirdar**

Final Exam Summer 2020 (60 points) **Summer 2020, CU-Boulder**

(b) (6 pts) Provide well commented pseudo code or actual code to solve the above problem.

Problem 5

Friday, July 24, 2020 9:38 PM



distance Parent Bool Possible to travel

Vertex	d	π	p
S ✗	0	Nil	True
A ✗	~~∞~~ 10	~~Nil~~ S	True
B ✗	~~∞~~ ~~20~~ 15	~~Nil~~ ~~S~~ A	True
t	~~∞~~ 26	~~Nil~~ A	False

Say number of miles was 20

m=20

U = S

1. d[V] > d[U] + w

   ∞ > 0 + 10

   d[V] = 10

   π[V] = S

2. d[V] > d[U] + w

   ∞ > 0 + 20

   d[V] = 20

   π[V] = S

U = A break loop

3. d[V] > d[V] + w

   ∞ > 10 + 16

   d[V] = 26

   π[V] = A

4. d[V] > d[V] + w

   20 > 10 + 5

   d[V] = 15

   π[V] = A

U = B break loop

5. d[V] > d[U] + w

   26 > 15 + 20 False

U = t break loop

No adj nodes.

Pseudo code

```
(1) S = ∅
(2) Q = V[G]
(3) while Q != ∅
      U = etract min (Q) ← binary heap
      S = S ∪ {U}
      for each V ∈ Adj[U] is Not in Q
          if d[V] > d[U] + w
              d[V] = d[U] + w
              π[V] = U
          if d[V] >= m
              P[V] = True
          else
              P[V] = False
```

(c) (2 pts) Explain your algorithms runtime and space complexity by analyzing your code. For example, stating that a sorting algorithm runs in $\mathcal{O}(n\log(n))$ without any justification is insufficient.

I used a modify version of Dijstra's Algorithm which has a run time of $\mathcal{O}(V^2)$. If the input graph is represented using adjacency list, it can be reduced to $E\log V$ by using a binary heap to search for our min vertex.

Name: Mauro Vargas

ID: 107212593

CSCI 3104, Algorithms **Escobedo & Jahagirdar**
Final Exam Summer 2020 (60 points) **Summer 2020, CU-Boulder**

6. (20 pts) There are an even number of items arranged along a line. Each item has a value v_i associated with it. You are competing against an opponent to collect items such that total value of your items collected is maximized. At each round of the game, one player is allowed to pick an item from either end of the line.

   Determine the maximum total value that can be obtained, if you are allowed to play first assuming that the opponent is equally clever. The input to your algorithm will be a list of values. The output should be a number indicating the maximum amount of value that you can obtain.

   Example
   Input: $22, 43, 10, 20$
   Output: 63
   Explanation: As the first player I would first pick 20 from the right end. The opponent would then pick 22. Out of the two elements 43, 10 remaining, I can pick 43. Thus 20+43=63 is the maximum total value that can be obtained.
   Note: Observe in the above example that the greedy choice of picking the max valued item does not work. If I had picked 22 instead of 20 in the first round, the maximum value that I could have obtained is 32.

   (a) (5 pts) State the base case and recursive relation that can be used to solve the above problem using dynamic programming.

   Method 1: Recursion.
   A simple solution is to consider all subsets of items and calculate the maximum total value that can be obtained for every subset. Sub-problems are evaluated redundantly, meaning this problem has Overlapping sub-problems property. Since there is overlapping sub-problems we can use Dynamic Programming to avoid computations of same sub-problems by constructing a temporary array B[][] in a bottom-up manner.
   Correct Method 2: Dynamic Programming

   Base case 1: No Items

(b) (5 pts) Provide an example input consisting of at least 6 items. Show how the dynamic programming data structure used to store previous computations is built based on the recursive relation.

```
[[1, 2, 4, 6, 9, 12],
[0, 2, 3, 6, 8, 12],
[0, 0, 3, 4, 8, 10],
[0, 0, 0, 4, 5, 10],
[0, 0, 0, 0, 5, 6],
[0, 0, 0, 0, 0, 6]]

input: array = [1,2,3,4,5,6]
out put: 12
```

(c) (8 pts) Write down well commented pseudo-code or paste real code to solve the above problem using Dynamic Programming or Memoization based on the above recurrence relation.

```
def f(A, k):
    B = [[0 for x in range(k)] for y in range(k)] #create table
    if k == 0: # base case: No items
        return None
    for  index in range(0, k):
        i = 0 #Keep track of out index
        j = index #Keep track of inner index
        while j < k: #while we have not reached the end of the array
            if ((i + 2) <= j):
                a = B[i + 2][j] #store in Matrix
            else:
                a = 0
            if ((i + 1) <= (j - 1)):
                b = B[i + 1][j - 1] #store in Matrix
            else:
                b = 0
            if i <= (j - 2):
                c = B[i][j-2] #store in Matrix
            else:
                c = 0
            B[i][j] = max(A[i] + min(a,b), A[j] + min(b,c))#find the max value
            i = i + 1 # move index i
            j = j + 1 # move index j
    return B[0][k-1] # return max value

array = [1,2,3,4,5,6]
n = len(array)
D = f(array,n)
print(D)
```

Name: Mauro Vargas

ID: 107212593

CSCI 3104, Algorithms **Escobedo & Jahagirdar**

Final Exam Summer 2020 (60 points) **Summer 2020, CU-Boulder**

(d) (2 pts) Discuss the space and runtime complexity of the code, providing necessary justification.

Time Complexity: $\mathcal{O}(\mathcal{N} * \mathcal{W})$.
Where 'N' is the number of weight element and 'W' is capacity. As for every weight element we traverse through all weight capacities $1 <= w <= W$. Since we do not have any weights our time complexity will just be $\mathcal{O}(\mathcal{N})$.
Auxiliary Space: $\mathcal{O}(\mathcal{N} * \mathcal{W})$.
The use of 2-D array of size "N*W", since we don't have any weights our space complexity will be $\mathcal{O}(\mathcal{N})$.

Name: Mauro Vargas

ID: 107212593

7. **Extra Credit (3 pts) will only be considered if your final exam score is less than 100%**

   Write two double spaced pages about the job or position you hope to have when you graduate from college (If you already have a job lined up, write about your next career goal). Your essay must include i) a tractable career goal, ii) a reason why you want to be hired/accepted to this position, iii) a step-by-step plan of how you intend to achieve your goal.