cmput 304 2024 study questions 2 show your work

1.
```
L = createEdgeList(G)                         # (3)
P = {}                                        # (4)
for v in G: P[v] = v                          # (2)
while len(L) > 0:                             # (9)
    t = extractmin(L)                         #       (7)
    a, b = t[0], t[1]                         #       (0)
    ra, rb = UF.myfind(a,P), UF.myfind(b,P)   #       (6)
    if ra != rb:                              #       (1)
        print(a,b,t[2])                       #              (5)
        UF.myunion(ra,rb,P)                   #              (8)
```

2. When we create the edge list L, we construct L from G which has n vertices. We have a for loop that iterates through all vertices $v \in G$ which takes $O(n)$ time. Then we iterate through the edges of v, and there could be $n-1$ of these, so we are now taking at least $O(n^2)$ time. The rest of the steps take constant time, so creating L takes $O(n^2)$ time.

   Next we begin a while loop that iterates through L, L has $O(n^2)$ edges so this now takes at least $O(n^2)$ time. Then calling extractmin does another iteration through L, so it takes $O(n^2)$ time here. In total, we now have an algorithm that takes $O(n^4)$ time. It is simple to verify then that the rest of the steps in this while loop are dominated in time complexity from extractmin, and so the loop takes $O(n^4)$ time. In the case that the number n is too large to fit into one memory location, we have to multiply our runtime by the time necessary to perform any basic operation (addition, subtraction, assignment) by k where k is the size (number of memory locations) needed to store n, so by $\Theta(\lg n)$, so the total runtime is in $O(n^4 \lg n)$.

3. a) `fib(n = 0)` and `fib(n = 1)` both get inside the `if` condition and instantly `return n` and terminate, therefore $T(0) = T(1) = 1$.

b) For any $n \geq 2$, `fib(n)` doesn't get inside the `if` condition and instead call `return fib(n-1) + fib(n-2)`. In other words, it makes a total of: one call to `fib()` on the top layer, plus $T(n-1)$ calls as a result of `fib(n-1)`, plus $T(n-2)$ as a result of `fib(n-2)`. Hence, $T(n) = 1 + T(n-1) + T(n-2)$.

c) Primer: $f(0) = 0, f(1) = 1, f(2) = 1, f(3) = 2, f(4) = 3, f(5) = 5$.

Proof by strong induction on n:

Base case:

We know $T(0) = T(1) = 1$. This means $T(0) = 2f(1) - 1$ and $T(1) = 2f(2) - 1$. It also means, $T(2) = 1 + T(1) + T(0) = 3 = 2f(3) - 1$, so $T(n) = 2f(n+1) - 1$ for $n = 2$.

Inductive step:

Assume the property $T(n) = 2f(n+1) - 1$ holds for $n = [2, k]$ with $k \geq 2$, we want to show that it also holds for $n = k + 1$.

$$\begin{aligned} T(k+1) &= 1 + T(k) + T(k-1) && \text{by property from (a)} \\ &= 1 + 2f(k+1) - 1 + 2f(k) - 1 && \text{by the inductive assumption} \\ &= 2\left[f(k+1) + f(k)\right] - 1 \\ &= 2f(k+2) - 1 && \text{by the Fibonacci property} \end{aligned}$$

The proof by induction is complete. $T(n) = 2f(n+1) - 1$.

d) The Fibonacci sequence can be approximated as:

$$f(n) = \frac{1}{\sqrt{5}}\left(g^n - (1-g)^n\right)$$

Let n_0 be a large enough number such that for $n \geq n_0$, $(1-g)^n \approx 0$. So,

$$f(n) \approx \frac{1}{\sqrt{5}} g^n \qquad (\text{for } n > n_0)$$

So, for $n \geq n_0$, we have

$$T(n) = 2f(n+1) - 1 \approx 2\left(\frac{1}{\sqrt{5}} g^{n+1}\right) - 1 \approx \frac{2}{\sqrt{5}} g^{n+1}$$

Let $c = \frac{2}{\sqrt{5}} g$. Then $cg^n = \frac{2}{\sqrt{5}} g^{n+1}$, and both of these are satisfied for all $n \geq n_0$: $cg^n \geq T(n) \geq 0$ and $0 \leq cg^n \leq T(n)$.

So $T(n) \in \mathcal{O}(g^n)$ and $T(n) \in \Omega(g^n)$. Therefore, $T(n) \in \Theta(g^n)$.

4. Let $\nu(j)$, $\mu(j)$ be the value of a and b respectively after line 4 has executed exactly j times. We have $\nu(j+1) = \mu(j)$ (for $j \geq 0$) because a is assigned to be b of the previous iteration. Denote that property (1). We also have $\mu(j) = \nu(j-1) + \mu(j-1)$ (for $j \geq 1$) because b is assigned to be the sum of a and b from the previous iteration. By (1), this implies $\mu(j) = \nu(j-1) + \nu(j-2)$ (for $j \geq 2$). Denote that property (2).
   We want to show that $\nu(j) = f(j)$. Proof by strong induction on j:
   Base case:
   With $j = 0$, line 4 is never executed, which means $a = 0$ as initialized, so $\nu(0) = 0 = f(0)$. With $j = 1$, line 4 is executed once, which means $a = 1$, so $\nu(1) = 1 = f(1)$.
   Inductive case:
   Assume the property $\nu(j) = f(j)$ holds for $j = [2, k]$ with $k \geq 2$, we want to show that it also holds for $j = k + 1$.

$$\begin{aligned} \nu(j+1) &= \mu(j) && \text{by (1)} \\ &= \nu(j) + \nu(j-1) && \text{by (2)} \\ &= f(j) + f(j-1) && \text{by the inductive assumption} \\ &= f(j+1) && \text{by the Fibonacci property} \end{aligned}$$

   The proof by induction is complete. $\nu(j) = f(j)$.

5. a) The naive algorithm works: iterate through n, add the pair of binary bits, generate carry overs and incorporate them in the next bit. This algorithm reads each of the $2n$ bits. We want to prove that we cannot do better—we cannot add two n-bit numbers while reading just $2n - 1$ or fewer bits.
   Suppose for contradiction: an algorithm ALG can add two numbers (a and b of n-sized bits) without reading one of the bits. Let $n = 1$, then ALG don't read one of a and b. WLOG, suppose ALG don't read b, which means ALG don't know if $b = 0$ or $b = 1$ when outputting. We can treat this as

a game between ALG and an adversary ADV, where ADV reveals a, then ALG reveals its output, only then does ADV reveals b. Let ADV first reveals $a = 0$, then let ALG reveals its output, then ADV reveals b to be the opposite of ALG's output, rendering ALG wrong. This is in contradiction with the assumption that ALG can add two numbers without reading one bit. The proof is complete.

b) Any algorithm to add two numbers must read each of the $2n$ bits. Let $f(n)$ be the runtime of the most optimal algorithm: $f(n) \geq 2n$. Let $c = 1$ and $n_0 = 1$, then we have $f(n) \geq 2n \geq cn \geq 0$ for all $n > n_0$. So $f(n) \in \Omega(n)$.

6. a) Prim's Trace:

		a	b	c	d	e	f	g	h	j	k
tree											
a update		*	4a		4a	6a					
ab 4											
b update		*	*	8b	4a	3b	6b				
be 3											
e update		*	*	8b	4a	*	6b		4e	3e	
ej 3											
j update		*	*	8b	4a	*	6b		4e	*	12j
ad 4											
d update		*	*	8b	*	*	6b		1d	*	12j
dh 1											
h update		*	*	8b	*	*	6b		*	*	12j
bf 6											
f update		*	*	8b	*	*	*	11f	*	*	11k
bc 8											
c update		*	*	*	*	*	*	2c	*	*	11k
cg 2											
g update		*	*	*	*	*	*	*	*	*	7g
gk 7											

$\Rightarrow$ Set of edges: (ab, be, ej, ad, dh, bf, bc, cg, gk)

b) Prim's Trace:

		a	b	c	d	e	f	g	h	j	k
tree											
k update							11k	7k		12k	*
gk 7											
g update				2g			11g	*		12k	*
cg 2											
c update			8c	*			9c	*		12k	*
bc 8											
b update		4b	*	*		3b	6b	*		12k	*
be 3											
e update		4b	*	*	5e	*	6b	*	4e	3e	*
ej 3											
j update		4b	*	*	5e	*	6b	*	4e	*	*
ab 4											
a update		*	*	*	4a	*	6b	*	4e	*	*
ad 4											
d update		*	*	*	*	*	6b	*	1d	*	*
dh 1											
h update		*	*	*	*	*	6b	*	*	*	*
bf 6											

$\Rightarrow$ Set of edges: (gk, cg, bc, be, ej, ab, ad, dh, bf)

c) Prim's Trace:

		a	b	c	d	e	f	g	h	j	k
tree											
a update		*	4a		4a	6a					
ad 4											
d update		*	4a		*	5d			1d		
dh 1											
h update		*	4a		*	4h			*	4h	
hj 4											
j update		*	4a		*	3j	10f		*	*	12j
ej 3											
e update		*	3e		*	*	8e		*	*	12j
be 3											
b update		*	*	8b	*	*	6b		*	*	12j
bf 6											
f update		*	*	8b	*	*	*	11f	*	*	11f
bc 8											
c update		*	*	*	*	*	*	2c	*	*	11f
cg 2											
g update		*	*	*	*	*	*	*	*	*	7g
gk 7											

⇒ Set of edges: (ad, dh, hj, eg, be, bf, bc, cg, gk)

d) Prim's Trace:

		a	b	c	d	e	f	g	h	j	k
tree											
k update							11k	7k		12k	*
gk 7											
g update				2g			11k	*		12k	*
cg 2											
c update			8c	*			9c	*		12k	*
bc 8											
b update		4b	*	*		3b	6b	*		12k	*
be 3											
e update		4b	*	*	5e	*	6b	*		3e	*
ej 3											
j update		4b	*	*	5e	*	6b	*	4j	*	*
hj 4											
h update		4b	*	*	1h	*	6b	*	*	*	*
dh 1											
d update		4d	*	*	*	*	6b	*	*	*	*
ad 4											
a update		*	*	*	*	*	6b	*	*	*	*
bf 6											

⇒ Set of edges: (gk, cg, bc, be, ej, hj, dh, ad, bf)

e) Kruskal's Trace:

dh 1

cg 2

be 3

ej 3

ab 4

ad 4

reject he

reject hj

reject de

reject ae

bf 6

gk 7

bc 8

reject ef

reject cf

reject fj

reject fg

reject fk

reject jk

$\Rightarrow$ Set of edges: (dh, cg, be, ej, ab, ad, bf, gk, bc)

7. d) We want to show that K is a MST. If $K = M$ where M is a MST, well then we have shown what we want.

   k) M_1 was constructed so that $M_1 = M + e_k - e_j$. e_j was chosen from the cycle C, which is the cycle of $M + e_k$. Removing an edge from a cycle of $M + e_k$ guarantees that $M + e_k - e_j$ is still spanning.

   l) Removing an edge from a cycle of $M + e_k$ doesn't change the connectedness of M_1, since it was an edge from a cycle. A tree is simply just a connected, non-cyclic graph, so we have a MST.

   m) If it were less, then M would not be a minimum spanning tree to start with.

   n) This is a tautology, but never the less it is an important step in the proof for logical deduction in step (o).

   o) From m and n it follows by definition that M_1 is a MST.

p) From steps f) and on, we can repeat the argument until the step (f) can not be done anymore (since we have but finitely many edges), and therefore we conclude at the end that $M_1 = K$, is a spanning tree.

8. Assuming we choose e_k by smallest weight first, we would choose edge FG first. Then the cycle $FGEBD$ is the cycle C in the proof. We want to find e_j such that $e_j \in C \setminus K$. There is but one possibility for this e_j being edge BD.

9. (0, A):

   0 does not prefer A to D

   A prefers 0 to 1

   $\Rightarrow$ not unhappy

   (0, B):

   0 prefers B to D

   B does not prefer 0 to 3

   $\Rightarrow$ not unhappy

   (0, C):

   0 prefers C to D

   C does not prefer 0 to 2

   $\Rightarrow$ not unhappy

   (1, B):

   1 does not prefer B to A

   B does not prefer 1 to 3

   $\Rightarrow$ not unhappy

   (1, C):

   1 does not prefer C to A

   C does not prefer 1 to 2

   $\Rightarrow$ not unhappy

   (1, D):

   1 does not prefer D to A

D does not prefer 1 to 0

$\Rightarrow$ not unhappy

(2, A):

2 prefers A to C

A prefers 2 to 1

$\Rightarrow$ unhappy

(2, B):

2 prefers B to C

B prefers 2 to 3

$\Rightarrow$ unhappy

(2, D):

2 does not prefer D to C

D prefers 2 to 0

$\Rightarrow$ not unhappy

(3, A):

3 prefers A to B

A prefers 3 to 1

$\Rightarrow$ unhappy

(3, C):

3 prefers C to B

C does not prefer 3 to 2

$\Rightarrow$ not unhappy

(3, D):

3 prefers D to B

D does not prefer 3 to 0

$\Rightarrow$ not unhappy

10. Bipartite preference system and matching with (0, B) and (1, A):

    Check for the following (0, A) and (1, B):

    (0, A):

0 prefers A to B

A does not prefer 0 to 1

⇒ not unhappy

(1, B):

1 prefers B to A

B does not prefers 1 to 0

⇒ not unhappy

Since there is no unhappy couple, it is stable.

Bipartite preference system and matching with (0, A) and (1, B):

Check for the following (0, B) and (1, A):

(0, B):

0 does not prefer B to A

B prefers 0 to 1

⇒ not unhappy

(1, A):

1 does not prefer A to B

A prefers 1 to 0

⇒ not unhappy

Since there is no unhappy couple, it is stable.

11. a) u = 1, v = 0, w = 0, x = 1 because there are no unhappy matches. Therefore, it is stable

    b) Following possible assignments of u, v, w, x can give valid and unstable matching:

    ⇒ u = 0, v = 1, w = 0, x = 1 because (0, A) is unhappy.

    ⇒ u = 0, v = 1, w = 1, x = 0 because (0, A) is unhappy.

    ⇒ u = 1, v = 0, w = 1, x = 0 because (1, B) is unhappy.

12. a) The new proposals made: First 0 would propose to C (even though they are already connected). Now the dotted line means that 1 can not propose to

A, so 1 instead proposes to D, (again they are connected). Then 2 proposes to A, and 3 proposes to A as well (since 3 can no longer propose to D). Now, A has two incoming proposals, and says maybe to 3 and no to 2 (again by the listing of preferences). B has no incoming proposals. C has only one incoming proposal, and so says maybe to 0. D has only one incoming proposal, and so says maybe to 1.