(3) (e) List L w/ n integers, maybe (-) #'s

(1) remove 1'st int and keep score

(2) add 1'st int and remove 2

```
int maxScore(L):
    n = L.length();   score = 0;
    for i=0; n
        if ( L[i+1] > L[i] || i+1 = size)
            choose (1) <-> score = score; L.pop();
        else if ( L[i+1] < L[i] )
            choose (2) <-> score = score + L.pop(); L.pop() // again
    return score
```

Scanned with Mobile Scanner

(3) (d) Because there is a special pass, we can keep track of all shortest paths by subtracting the largest ~~other~~ weight from any possible path.

↳ We can write A algorithm that uses Dijkstra's algorithm on A as the source and return a vector ~~g~~ consisting of all the vertices w/ their shortest edges. The following algorithm has u.d for every neighbor of A set to the smallest value it could be. If we can subtract the maximum toll, or weight. We can design the Algorithm so that it runs Dijkstra's algorithm after the larger weight from any possible unique path from A to B had been set to zero.

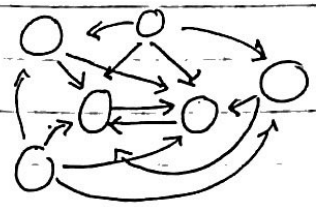
```
Dijkstra (G, w, s)
Initialize-single-source (G, s)
S = ∅
Q = G.V
while Q ≠ ∅
    u = Extract min (Q)
    S = S ∪ {u}
    foreach vertex v ∈ G.adj[u]
        Relax (u, v, w)
```

→ Algorithm step 1: Find all possible unique, non overlapping paths from A to B and set the maximum weight of each of those paths to zero.

→ Run Dijkstra's algorithm on A as the source and the cheapest path will be revealed to B.

Scanned with Mobile Scanner

(3) (c)



There are n ~~[illegible]~~ rows and n columns

m(i,j)=1 if i knows j that means for celebrity, all of j is 1 meaning m(:,j)=1

~~Pair<int, int> find celebrities (m);~~

~~for size = m.rows.length(); start = 1;~~

~~maybe = find someone (m, start);~~

~~Pair<int, int> answer = <0,0>;~~

~~stack S; n = m.rows.size();~~

~~int celebrity1;~~

~~int celebrity2;~~

~~for (int i=0; n)~~

~~S.push(i);~~

~~A = S.top S.POP()~~

~~B = S.top S.pop()~~

~~int find someone (m, j)~~

~~for i = 0 to m.rows.length()~~

~~if m(i, j) = 1~~

~~return i~~

~~bool knows(a, b, m)~~

~~return m[a][b];~~

~~Pair<int, int> find celebrities (m);~~

~~n = m.size(); a = 0, b = n-1; temp = n-1~~

~~while (a < b) {~~

~~if (knows(a, b, m))~~

~~a++~~

~~else b--~~

~~}~~

~~for (i=0; n)~~

~~for j = 0; size -1~~

~~for i = 0; size -1~~

~~int find someone (m, j)~~

~~for (int i = j to m.size)~~

~~if m[j][i] = 1~~

~~return i~~

assuming every row represents a possible person

Pair<int, int> find celebrities (m)

n = m.size(); pair<int, int> answer = <0,0>; start = 0; i = 0;

meaning if m is nxn, n is n

~~for (i = 0; n)~~ while (i < n){

if (m[start][i] == 1) {start ++;}

~~{start ++}~~ (if start != i)

else {i++, start = 0} // reset start if i increments

if (start == n)

answer.first = i; break // first celebrity found

} ~~while~~ start = 0; i = i+1 // reset start and increment i

while (i < n)

if (m[start][i] == 1)

start ++; (if start != i)

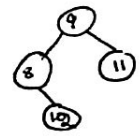
else i++, start = 0

if (start == n)

answer.second = i; break

// this is O(n) b/c two while loops run up to n.

Scanned with Mobile Scanner



(3) (b)

```
Node deleteMax (Node root): {
    int maxval= findMax (root);
    return deleteNode (root, maxval);
}
```

```
int findMax (Node root) {
    int result= root->val;
    int leresult= findmax (root->left);
    int riresult= findMax (root->right);
    if (leresult > result)
        result= leresult;
    if (riresult > result)
        result = riresult;
    return result;
}
```

```
Node minVal (Node node) {
    Node current = node;
    while (current && current->left != NULL)
        current = current->left;
    return current;
}
```

```
Node deleteNode (int value, Node Parent) {
if (value < parent->val) {
    if (left != NULL) return
```

turns out these were right

```
Node deleteNode (Node root, int key) {
    if (root == NULL) return root;
    if (key < root->val) // pretty useless
        root->left= deleteNode (root->left, key);
    else if (key > root->val)
        root->right= deleteNode (root->right, key);
    else
        if (root->left == NULL) {
            Node temp= root->right;
            free (root);
            return temp;
        }
        else if (root->right == NULL) {
            Node temp = root->left;
            free (root);
            return temp;
        }
        Node temp= minVal (root->right);
```

```
Node deleteNode (Node root, int key) {
if(root== NULL). return root;
if(key < root->Val) root-> left= deleteNode (root->left, key);
else if(key > root->val) root ->right= delete Node (root->right, key);
else { if (root->left==NULL){ Node temp= root->right;
        free(root) return temp; }
    else if (root->right==NULL){ Node temp = root->left;
        free(root); return temp; }
    Node temp= minVal (root -> right);
    root -> key = temp -> key;
    root -> right = delete Node (root -> right, temp -> key); }
return root;
}
```

Scanned with Mobile Scanner

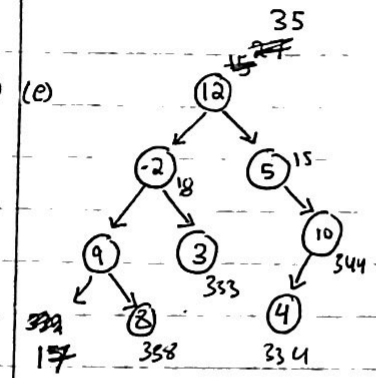
(3) (a)

```
pair <int, int> twoLargestPathSums (Node root): {
    pair <int, int> finalvec = <0, 0>;
    int sum1 = 0; sum1 = root->val;
    visit (root, sum1, finalvec);
    return finalvec;
}


//Need a recursive function to visit children of root
void visit (Node root, int sum1, pair <int, int> finalvec){
    if (root->left == NULL && root->right == NULL) // safe to say this will not be original parent root
        if(sum1 > finalvec.first && sum1 != finalvec.second)
            finalvec.first = sum1
        elseif( sum1 > finalvec.second && sum1 != finalvec.first)
            finalvec.second = sum1
    if (root->left != NULL)
        sum1 += root->left->val // update sum1 as we go
        visit (root->left, sum1, finalvec)
        sum1 -= root->left->val // delete value of children if we no longer need it
    elseif (root->right != NULL)
        sum1 += root->right->val
        visit (root->right, sum1, finalvec)
        sum1 -= root->right->val
}
```

Scanned with Mobile Scanner

(2) (e)



root = 12 12 → L = -2 -2 → L = 9 9 → L = NULL

visit 9 9 += 330 → 339 9 → R = 8 8 → L = NULL

visit 8 8 += 330 = 338 visit -2 -2 += 9 += 3 = 10

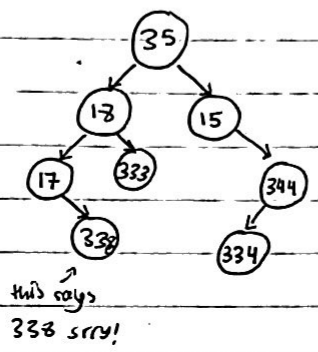
-2 → R = 3 3 → L = NULL visit 3 3 += 330 ⇒ 333

visit 12 12 += 18 += 15 ⇒ 35 12 → R = 5 5 → L = NULL

visit 5 5 += 10 ⇒ 15 5 → R = 10 10 → L = 4 4 += 330 = 334

Visit 10 10 += 334 = 344

New tree Answer



this says 338 srry!

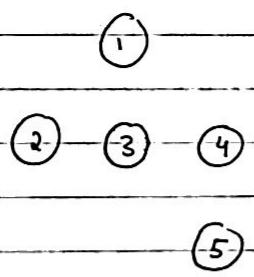
for (f)

```
Link(x,y)
if x.rank > y.rank
    y.p = x
else x.p = y
    if x.rank == y.rank
        y.rank = y.rank + 1
```

```
Find-set(x)
if x ≠ x.p
    x.p = Findset(x.p)
return x.p
```

```
Union(x,y)
Link(Find-set(x), Find-set(y))
```

(f)



initially all rank = 0

Union (5,4) 5.p = 4

y.rank = y.rank + 1

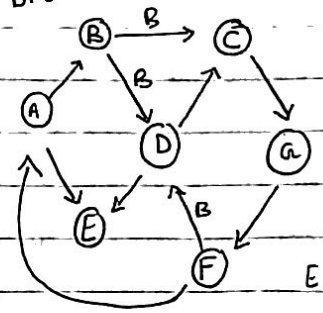
So u.rank = u.rank + 1 and 4.rank = 1

So before we perform this union rank of 1 must be raised through other unions

Union (2,1)	2.p = 1	1.rank = 1
union (3,1)	3.p = 1	1.rank = 1
union (5,4)	5.p = 4	4.rank = 1
union (4,1)	4.p = 1	1.rank = 2

Scanned with Mobile Scanner

(2) (c) DFS



Start at D; Alphabetical-order
No Back edges. mark everything white. No time stamps

D.color = white Visit D D.color = Gray C.π = D Visit C

C.color = Gray G.π = C visit G G.color = Gray

F.π = G Visit F F.color = GRAY A.π = F visit A

A.color = Gray B.π = A Visit B B.color = Gray

E(B,C) is back edge. E(B,D) is back edge B.color = Black

E.π = A Visit E E.color = Gray E.color = Black A. Color = Black

E(F,D) is back edge F.color = Black G.color = Black C.color = Black

D.color = Black

In order of what got marked black last to first: If something got marked black later that means it was discovered first and it got traversed through

D	C	G	F	A	E	B

last First

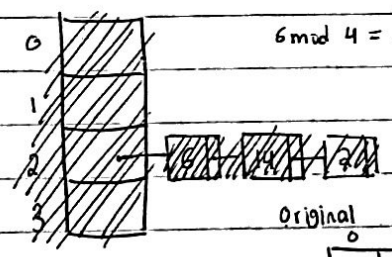
Order of visit: D → C → G → F → A → B → E

(d) Load Factor = $\frac{\text{total \# of items stored}}{\text{size of array}}$

default size of H is 4 → n=4

insert sequence: 6, 14, 2, 1, 15, 3

hash function ~~6~~ x mod n

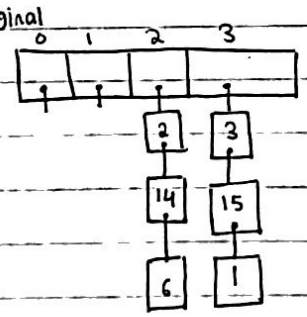


6 mod 4 = 2 load factor: $\frac{1}{4}$ ✓ 14 mod 4 = 2 load factor = $\frac{2}{4}$ ✓

2 mod 4 = 2 load factor = $\frac{3}{4}$ ✓ 1 mod 4 = 3 load factor = $\frac{4}{4}$ ✓

15 mod 4 = 3 load factor = $\frac{5}{4}$ ✓ 3 mod 4 = 3 load factor = $\frac{6}{4}$ ✗

Original



New hash table size n = 8

New traversal order 2, 14, 6, 3, 15, 1

2 mod 8 = 2 load factor will never go above 1.5 w/ 6 elements and 8 spaces

14 mod 8 = 6

6 mod 8 = 6

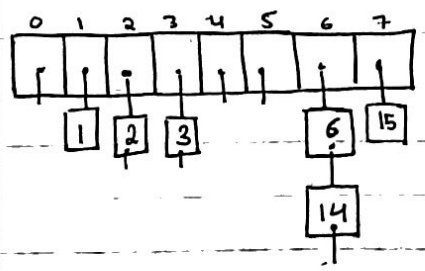
3 mod 8 = 3

15 mod 8 = 7

1 mod 8 = 1

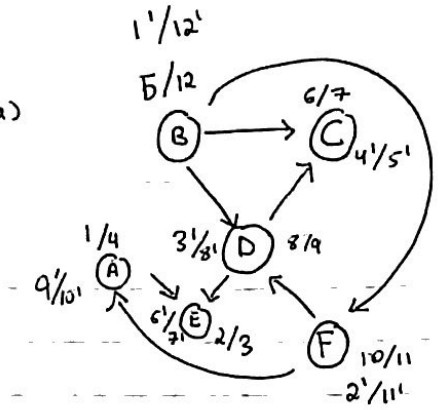
Answer ⇩

New



Scanned with Mobile Scanner

(2) (a)



Notes: B comes before C B→C

B comes before D which comes before C & E

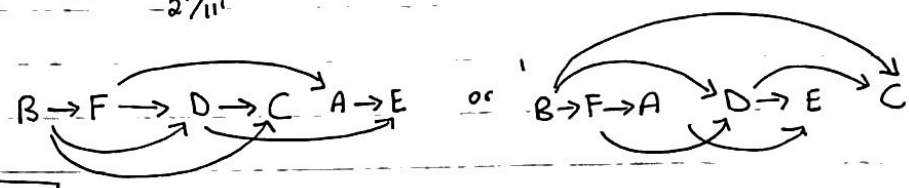
B comes before F which comes before D (wcb) C & E

B cb F wcb A wcb E

B→F→A→E

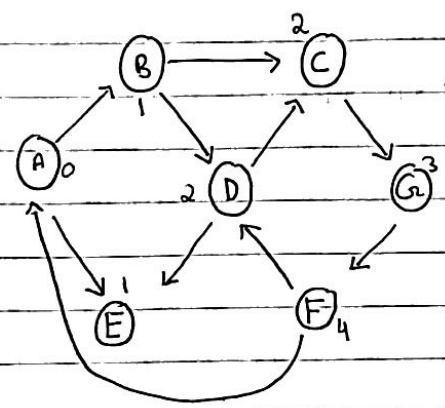
B→D→E
B→D→C

B→F→D→C
B→F→D→E



$\boxed{\text{BFADEC}}$

(b) BFS



Starting vertex A. in alphabetical order

* Discover all vertices at distance k before discovering vertices at distance k+1. everything white

A.color = grey A.d = 0 A.π = NIL

Q = ∅ Q = (A) u = A B.color = Grey

B.d = 1 B.π = A Q = {B} E.Color = Grey

E.d = 1 E.π = A Q = {B, E} A.color = Black

u = B C.color = Grey C.d = 2 C.π = B

Q = {E, C} D.color = Grey D.d = 2 D.π = B

Q = {E, C, D} B.color = Black u = E E.color = Black

u = C G.color = grey G.d = 3 G.π = C

Q = {D, G} C.color = Black u = D D.color = Black

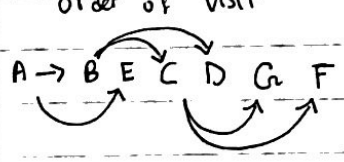
u = G F.color = grey F.d = 4 F.π = G Q = {F}

G.color = Black u = F F.color = Black

In order of what got painted Black first:

$\boxed{\text{A B E C D G F}}$

order of visit

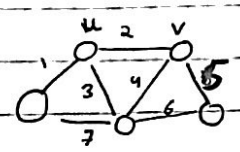


Scanned with Mobile Scanner

was not originally included if (u,v) = 5

(1) (h) ~~False~~

True



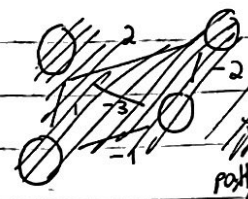
if (u,v) is second smallest, in any case one of the edges that connect u, v can be dropped and (u,v) being added will result in a more minimal tree.

$1+3+4+5=13$
$1+2+4+6=13$
$1+2+5+6=14$ X

all bigger than $1+2+3+5=12$

so this minimum spanning tree includes 2.

(i) ~~True~~ False



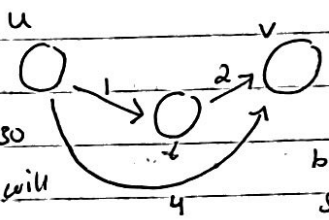
~~Even if we traverse along using Bell-man Ford, the shortest path is recognized to be original shortest path. $W_1+330+W_2+330 \ldots W_n+330 = (W_1 \ldots + W_n) + 330n$~~

$E_1 + E_2 \ldots + E_n$ = original path.

$E_1 + 330 + E_2 + 330 \ldots + E_n + 330$

$= (E_1 + E_2 + \ldots E_n) + 330n$

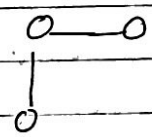
if $w(E_1)$ and $w(E_2 \ldots E_n) << 330$ reducing the amount of edges will result in shorter paths.



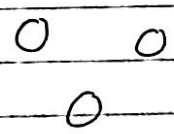
original shortest path is (u,i), (i,v)
but $1+330 + 2+330 > 4+330$
so new shortest path would be (u,v)

(j) An undirected graph is connected if every vertex is reachable from all other vertices. If a graph has n vertices and k connected components, it must have n-k edges at least

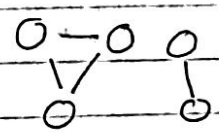
True



n = 5
k = 2
n-k edges ✓



n = 3
k = 3
n-k edges ✓



n = 5 at least
k = 2 n-k edges ✓

Final Ans.

(a) False
(b) True
(c) True
(d) False
(e) False
(f) True
(g) False
(h) True
(i) False
(j) True

Scanned with Mobile Scanner

Nafis Abeer EC330 Final U93285639

(1) (a) $n^{\cos n + \sin n}$ is $O(n)$

False

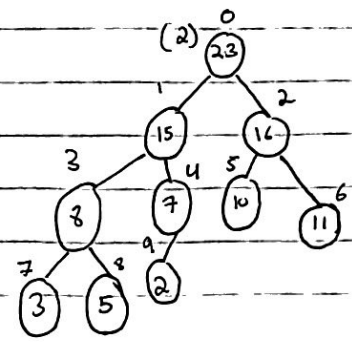
iff $\lim_{n\to\infty} \frac{n^{\cos n+\sin n}}{n} = 0$ ← this oscillates (diverges) ← f

not true always

therefore f is $\Theta(0)$

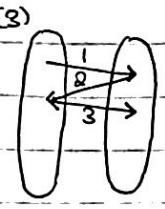
(b) [23, 15, 16, 8, 7, 10, 11, 3, 5, 2]

True

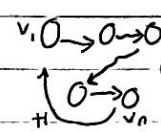


(c) A cycle of length 3 would cause a bipartite to have an odd cycle which is not allowed

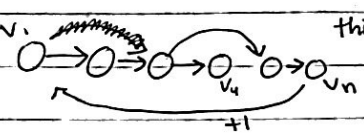
True



(d) Directed acyclic graph G w/ n vertices and at least n-1 edges



all vertices are connected and +1 edge makes it strongly connected



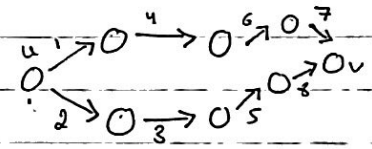
this is also a directed acyclic graph w/ n-1 edges but the +1 edge does not make it strongly connected b/c vertex v_4 can't reach any other vertex

* directed graph is strongly connected if every two vertices are reachable from one another

False

(e) Graph G (V, E, w), only +(w) unique edge weights

there could still be multiple shortest paths

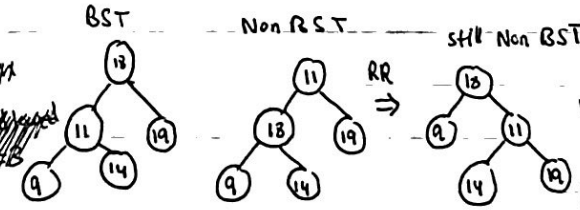


⇒ $\delta(u,v) = 13$ long which can be accomplished by going down both 1 & 2

False

(f) ~~[illegible]~~

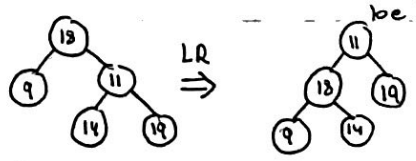
True



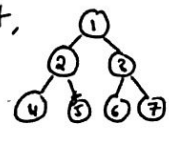
If 2 trees dont have the same inorder traversal, they can't be turned into each other

(g) False preorder prints root then traverses left side the right side.

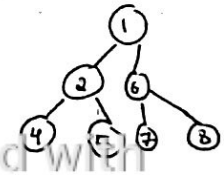
As long as left side values are smaller than right side of root, it is possible



min heap



	1	2	3	4	5	6	7
free order:	1	2	4	5	3	6	7
vals:	1	2	4	5	6	7	3 ✓



Scanned with Mobile Scanner