ASSIGNMENT 5 (RITIKA GOYAL) (301401516)

Ques1:

Every node can have at most 2t children and every node can have at most 2t -1 keys. To get the largest we consider maximum.

Root -> at most (2t-1) keys

Nodes at height 1 -> at most 2t children so total keys 2t (2t - 1)

Nodes at height 2 -> at most $(2t)^2$ children so total keys $(2t)^2$ (2t - 1)

. .

. .

. .

Nodes at height h -> at most $(2t)^h$ children so total keys $(2t)^h$ (2t – 1)

Total maximum keys in tree with minimum degree t and height h :

$$= (2t-1) + 2t\,(2t-1) + (2t)^2\,(2t-1) + \ldots\ldots\ldots. + (2t)^h\,(2t-1)$$

$$= (2t-1)\,[\,1 + 2t + (2t)^2 + \ldots\ldots. + (2t)^h\,]$$

$$= \frac{(2t-1)\left(1-(2t)^{h+1}\right)}{1-2t}$$

$$= (2t)^{h+1} - 1 \text{ keys}$$

Ques 2:

Input: An element x not in set S
Output: creates a new set containing x which is its representative.
MAKE-SET (x)

```
    Let each node of linked list contains three fields: next, value, set_ptr.
    Let S be the linked list with two field: head and tail
    Node n;
    n.set_ptr = S;
    n.next = null;
    n.value = x;
    S.head = n;
    S.tail = n;
    Return S;
```

Input: an element x present in Set
Output: returns a pointer to the representative.
FIND-SET (x)

```
    Let each node of linked list contains three fields: next, value, set_ptr.
    Let S be the linked list with two field: head and tail
    Return x.set_ptr.head.value;
```

UNION(x,y)

```
    Let each node of linked list contains three fields: next, value, set_ptr.
    Let the linked list has two field: head and tail

    Sx = x.set_ptr;
    Sy = y.set_ptr;

    Sx.tail.next = Sy.head;
    Sx.tail = Sy.tail;

    While Sy.head not equal to null
        Sy.head.set_ptr = Sx;
        Sy.head = Sy.head.next;

    Sy.tail = null;

    Return Sx;
```

Ques3:

Pseudo-Code:

Let arr_G be the 2-d array which is adjacency matrix of G.
Adjacent-matrix-representation_transpose (G, arr_G)

```
    Let arr_G' be the new matrix that stores transpose of G and is adjacency matrix of G'.
    n = |V|
    for i = 0 to n-1
        for j = 0 to n-1
            arr_G' [j][i] = arr_G[i][j];
    return arr_G';
```

Let list_G be the adjacency list of graph G.

```
Adjacent-list-representation_transpose (G, list_G)
    Let list_G' be the new adjacency list
    n = |V|
    for i = 1 to n
        for each j attached to G[i]
            list_G'[j].add(i)
    return list_G';
```

Running Time of adjacent matrix:

The transpose is calculated by using to for loops in the function. Updating the elements into new matrix takes constant time. So, the sunning time complexity is $= T(|V|^2) + T(1)$

$= O(|V|^2) + O(1)$

$= O(|V|^2)$

Running Time of adjacent list:

The first loop traverses through all the vertices and the second for loop check all edges attached to each vertex v and other operation for updating the value is constant. So, the running time of this algorithm is $= T(|E| + |V|) + T(1)$

$= O(|E| + |V|) + O(1)$

$= O(|E| + |V|)$

Ques4:

Let list_G be the adjacency list of graph G.

```
Adjacent-list-representation_transpose (G, list_G)
    Let list_G' be the new adjacency list
    Let temp be an array of size |V|.
    n = |V|
    for i=1 to n
        temp[i] = false;
    for i = 1 to n
        for each j in list_G[i]
            if temp[j] = false and  i != j
                list_G'[i].add(j)
                temp[j] = true;
    return list_G';
```

Ques5:

Pseudo-code:

Input: Graph G and Adjacency matrix A

```
BFS (G, A, s)
    for each v in V(G) do
        v.color = white;
        v.d = ∞;
        v. π = nil;
    s.color = grey;
    s.d =0;
    s. π = nil;
    Q = ϕ;
    Enqueue (Q,s);

    While Q ≠ ϕ
        u = Dequeue(Q)
        for i = 1 to |V|
            if i.color = white and A[u][i] = 1 then
                i.color = grey;
                i.d = u.d + 1;
                i. π = u;
                Enqueue(Q,i);
        u.color = black;
```

Running Time:

After using adjacency matrix, The first for loop iterates over every vertex giving $O(|V|)$ and then the nested loops check each vertex with every other vertex to check the adjacency giving $O(|V^2|)$ and other operations are constant in time. So,

Running time : $T(|V|) + T(|V^2|) + T(1)$

$= O(|V|) + O(|V^2|) + O(1)$

$= O(|V^2|)$

Ques 6:

Pseudo-code:

```
DFS (G)
    Let S be stack which is empty.
    time = 0;
    for each u in V do
        u.color = white;
        u. π = nil;
    for each u in V
        if u.color = white then
            S.push(u);
            time = time+1;
            u.d = time;
            u.color = gray;
            While S.Empty = false
                v = S.pop();
                for all r adjacent to v
                    if r.color = white then
                        r.color = gray;
                        time = time+1
                        r.d = time;
                        r. π = v;
                        S.push(r);
                time = time+1;
                v.f = time;
                v.color = black;
```