Xueying Chen

113366600

DSA-Fall-2019

Homework # 7

① 22.1-5

square of graph G is obtain by starting with G, and adding edges between any two vertices. whose distance in G is 2.

Adjacent-list

$1 \to 2 \to 4$

$2 \to 5$

$3 \to 6 \to 5$

$4 \to 2$

$5 \to 4$

$6 \to 6$

$\Downarrow$

$1 \to 2 \to 4 \to 5$

$2 \to 5 \to 4$

$3 \to 6 \to 5 \to 4$

$4 \to 2 \to 5$

$5 \to 4 \to 2$

$6 \to 6$

Algorithm

Starting from adj[1], for each $v \in adj[1]$, find adj[v], for each $v' \in adj[v]$, if v' is not in v, that means distance between v and v' is 2. add v' to adj[1], move to next vertex, until we have consumed all vertices.

Worst case when G is complete graph $E = |V|^2$

$O(VE) = O(|V|^3)$

For each vertex, we have to traverse thru all edge in worst case find distance. For example

$1 \to 2 \to 5 \to 4 \to 2 \to 5$.

therefore the time complexity is

$\Theta(VE)$

Matrix representation

	1	2	3	4	5	6
1	0	1	0	1	0	0
2	0	0	0	0	1	0
3	0	0	0	0	1	1
4	0	1	0	0	0	0
5	0	0	0	1	0	0
6	0	0	0	0	0	1

$\Downarrow$

	1	2	3	4	5	6
1	0	1	0	1	1	0
2	0	0	0	1	1	0
3	0	0	0	1	1	1
4	0	1	0	0	1	0
5	0	1	0	1	0	0
6	0	0	0	0	0	1

Similar to adj-list.

Matrix = m[6][6]

↑ Col ↑ Row

for i=1 to 6
  for j=1 to 6
    if m[i][j] == 1
      for k=1 to 6
        if m[j][k] == 1 and m[i][k] ≠ 1
          set m[i][k] = 1

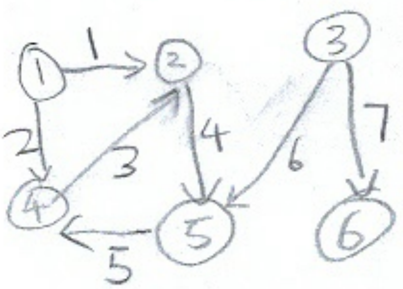
running time = $O(n^3)$

$n = |V|$

② 22.1-7

$|V| \times |E|$

$$B = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} \begin{array}{c} \begin{array}{ccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{array} \\ \begin{bmatrix} -1 & -1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & -1 \\ 0 & -1 & -1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & -1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \end{array} \quad 6 \times 7$$



$$B^T \quad \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \end{array} \begin{array}{c} \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \end{array} \\ \begin{bmatrix} -1 & 1 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & -1 & 0 & 0 \\ 0 & -1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & -1 & 0 \\ 0 & 0 & -1 & 0 & 1 & 0 \\ 0 & 0 & -1 & 0 & 0 & 1 \end{bmatrix} \end{array} \quad 7 \times 6$$

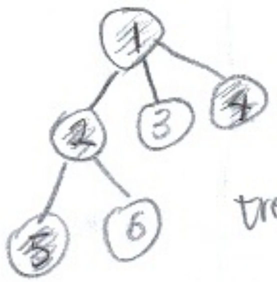
① The diagonal of BB^T represent # of edge connected to V_i, both entering & leaving V_i

② -1 in matrix BB^T represent a edge betwee V_i and V_j

$$BB^T = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} \begin{array}{c} \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \end{array} \\ \begin{bmatrix} 2 & -1 & 0 & -1 & 0 & 0 \\ -1 & 3 & 0 & -1 & -1 & 0 \\ 0 & 0 & 2 & 0 & -1 & -1 \\ -1 & -1 & 0 & 3 & -1 & 0 \\ 0 & -1 & -1 & -1 & 3 & 0 \\ 0 & 0 & -1 & 0 & 0 & 1 \end{bmatrix} \end{array} \quad 6 \times 6$$

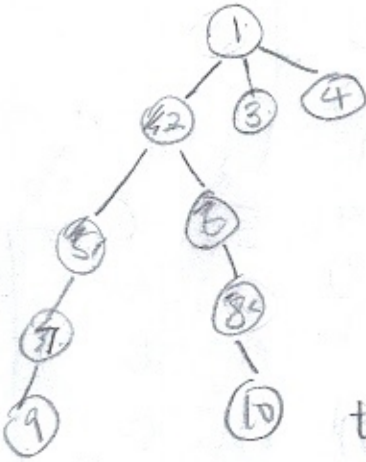
③ 22.2-8

Diameter = largest of all shortest path between 2 nodes



Diameter = 4 (nodes 5 & node 4

tree #2



Diameter = 7 (node 9 & 10)

tree #2

Take tree #2 for example.

Diameter at node 1 = 1+Sum (Max2(height(2), height(3), height(4

Sum of the height of 2 sub tree

for example height(2) = 4

height(3) = 1

height(4) = 1

Diameter at node 1 = 1+ sum(4, 1) = 6

Largest diameter = Max (Diameter(1), Diameter(2)
Diameter(3), Diameter(4))

largest Diameter (2) = Max(Diameter(2), Diameter(5), Diameter(6))

Diameter at 2 = 7

③ Continue.

Algorithm:

input t

```
int Diameter(t)

        h1 = 0    // largest height
        h2 = 0    // second largest height
        for child in t
              // find 1st and 2nd largest height in
              // sub trees of t.
              h = height(child)
              h1 = max(h1, h2, h)
              h2 = 2ndMax(h1, h2, h)
        Largest_Diameter = 1 + h1 + h2
        for child in t.
              d = Diameter(child)
              Largest_diameter = Max(Largest_diameter, d)

        return largest_diameter.
```

Algorithm Analysis:

Max() & 2ndMax() function are constant.

For the first for loop, we have to traverse n-1 node to find height of each child. For example, if the are 3 children, and each child has $\frac{n-1}{3}$ nodes. the

$$\text{Loop\#1} = 3\,\Theta\left(\frac{n-1}{3}\right) = \Theta(n)$$

For the second loop, we have recursion, we have to find diameter in $n-1$ nodes except for the root node

$$T(n) = T(n-1) + \theta(n)$$

and this is $\theta(n^2)$

4


```
// Original Implementation using recursion
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)


DFS-VISIT(G,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-VISIT(G, v)
  u.color = BLACK
  time = time + 1
  u.f = time


// Implementation using Stack.
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)


DFS-VISIT(G,u)

  Stack S // initialize stack to empty
  S.push(u)

  while ! S.isEmpty()
    u' = S.pop()
        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'
            S.push(v)
    u'.color = BLACK
    time = time + 1
    u'.f = time
```

⑤ i) using BFS

since all weight are same. we can applie BFS algorithm until we reach P. shortest path is the height of BFS spanning tree − 1

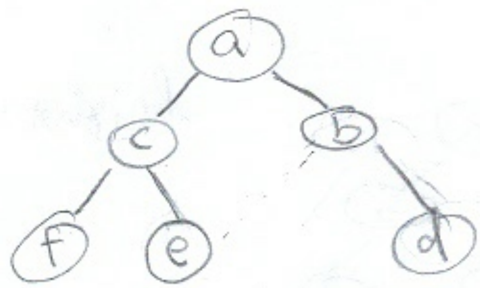
height = 1

visit a

height = 2

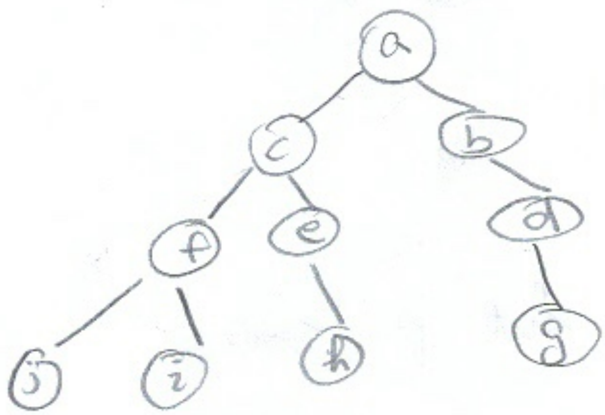
visit c & d.

height = 3



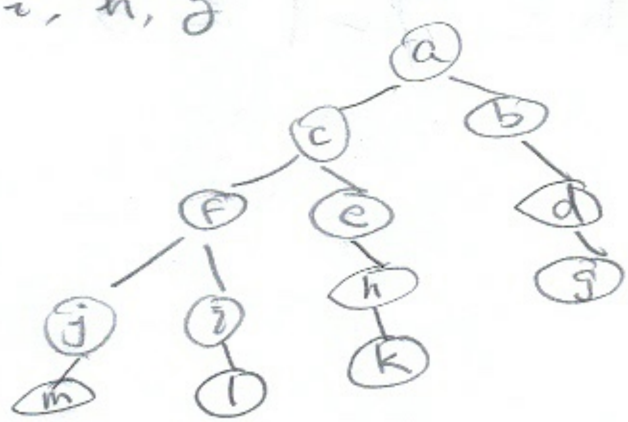
visit f, e, d.

height = 4



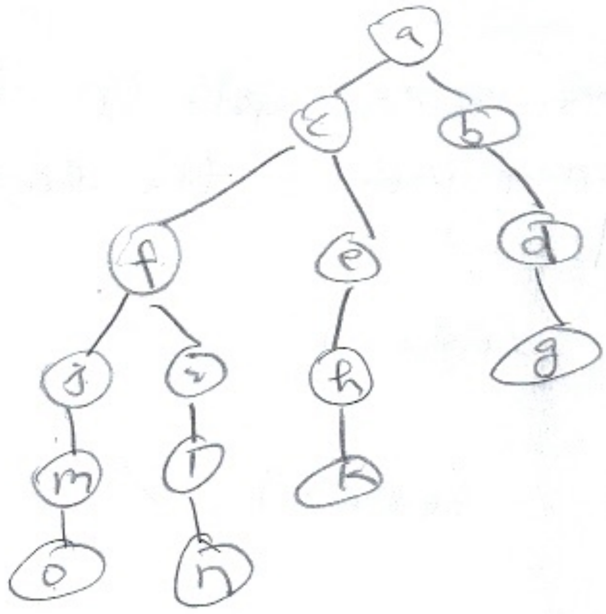
visit j i, h, g

height = 5



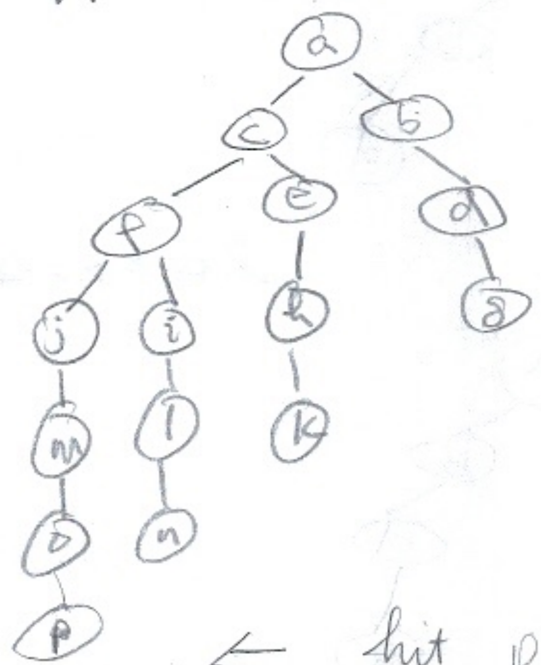
Visit m, l, k

height = 6



Visit o n

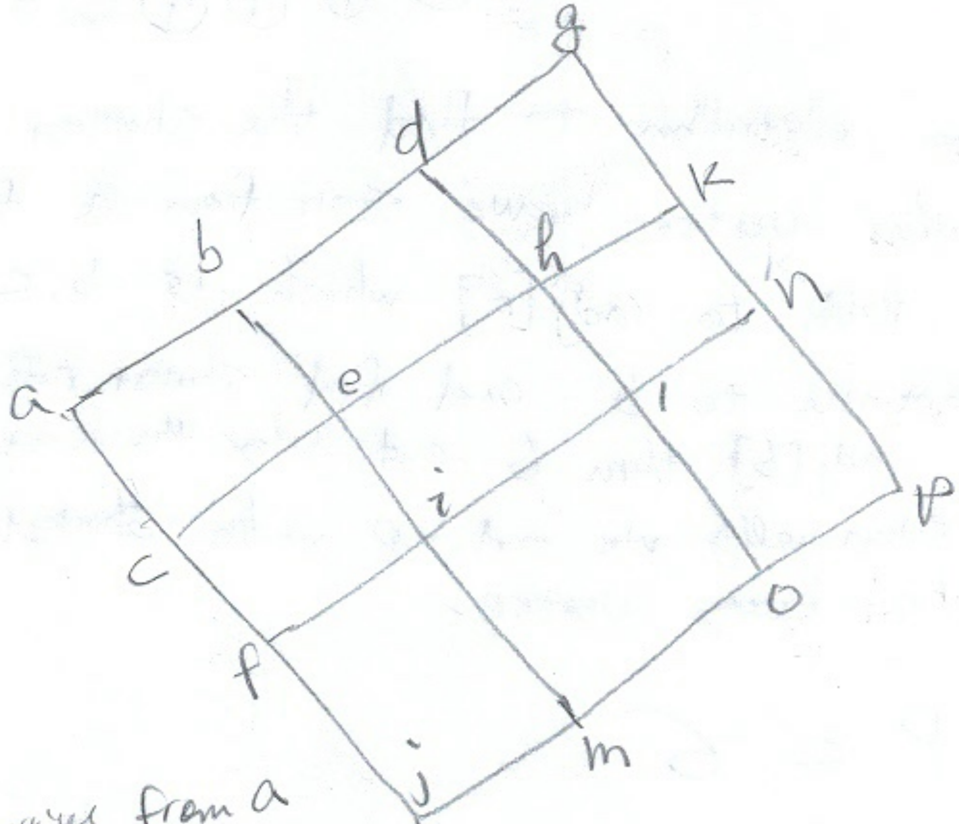
height = 7



← hit p stop

Shortest path = 7 - 1 = 6

ii)



starting from a

selected vertex	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p
b	1	1	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
c	1	1	2	2	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
d	1	1	2	2	2	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
e	1	1	2	2	2	3	3	∞	∞	∞	∞	∞	∞	∞	∞
f	1	1	2	2	2	3	3	3	∞	∞	∞	∞	∞	∞	∞
g	1	1	2	2	2	3	3	3	3	∞	∞	∞	∞	∞	∞
h	1	1	2	2	2	3	3	3	3	4	∞	∞	∞	∞	∞
i	1	1	2	2	2	3	3	3	3	4	4	∞	∞	∞	∞
j	1	1	2	2	2	3	3	3	3	4	4	4	∞	∞	∞
k	1	1	2	2	2	3	3	3	3	4	4	4	∞	∞	∞
l	1	1	2	2	2	3	3	3	3	4	4	4	5	∞	∞
m	1	1	2	2	2	3	3	3	3	4	4	4	5	5	∞
n	1	2	2	2	2	3	3	3	3	4	4	4	5	5	∞

	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p
a	1	1	2	2	2	3	3	3	3	4	4	4	5	5	6

Apply dijkstra algorithm to find the shortest path to every other vertices. we start from a find the shortest path to adj[a] which is b, c, then we continue to b, and find shortest path from a to adj[b] thru b and relax the cost if neccessary, eventually we end up with shortest path from a to every vertice.

$a \to P = 6$