Solution 10

1.

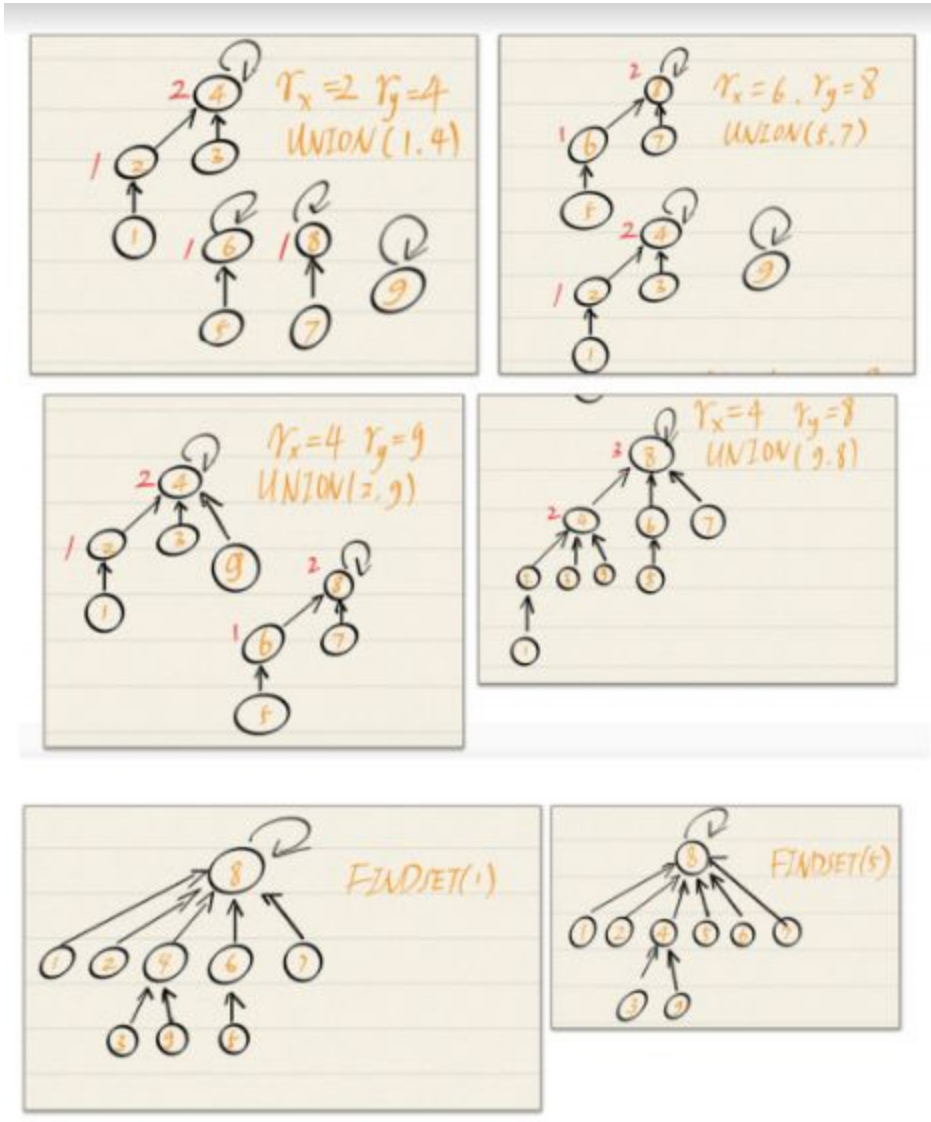
a. The cost of a minimum spanning tree is 23.
b. There are 2 different minimum spanning trees.

2.

v	v.key	v.pi
a	INF, 0	NIL
b	INF, 13, 7, 6	NIL, a, d, h
c	INF, 1	NIL, a
dd	INF, 3	NIL, c
e	INF, 2	NIL, a
f	INF, 9, 3	NIL, e, d
g	INF, 8, 5, 4	NIL, c, e, f
h	INF, 4	NIL, a

3

```
UNION( x, y )
rx = FIND-SET(x)
ry = FIND-SET(y)
if rx = ry : return
if rx.rank > ry.rank
    ry.p = rx
else:
  rx.p = ry
  if rx.rank == ry.rank:
      ry.rank = ry.rank +1
```


$r_x=2$ $r_y=4$
UNION(1,4)
$r_x=6$, $r_y=8$
UNION(5,7)
$r_x=4$ $r_y=9$
UNION(2,9)
$r_x=4$ $r_y=8$
UNION(9,8)
FINDSET(1)
FINDSET(5)

4.

The m table:

M	1	2	3	4	5	6
1	0	315	1782	1827	1752	5127
2		0	567	1377	1527	2427
3			0	1890	2240	4340
4				0	1350	9450
5					0	3000
6						0

The s table:

S	1	2	3	4	5	6
1		1	1	1	1	1
2			2	3	4	5
3				3	4	5
4					4	5
5						5
6						

The multiplication order is: A1 [(((A2 * A3) A4) A5) A6]

5.

Use a counter example to prove the claim gives suboptimal solution. Consider the following case A1 is 5x6, A2 is 6x3, A3 is 3x2. If you use her claim, then the result is (A1A2) A3 = 120. But we can group in another way,(A1(A2A3)) = 96. So, her claim is not correct.

6.

Solution partly from Peifu Yu and Yudong Zhou:

Formula (3 point):

dp[j] = max(p[j], p[i] + dp[j-i] - c) (i = 1….. j -1) j > 0

dp[0] = 0

Algorithm (3 point)

```
MEMOIZED-CUT-ROD(p, n)
    let r[0..n] be a new array
    for i = 0 to n
        r[i] = -∞
    return MEMOIZED-CUT-ROD-AUX(p, n, r)

MEMOIZED-CUT-ROD-AUX(p, n, r)
    if r[n] ≥ 0
        return r[n]
    if n == 0
        q = 0
    else q = -∞
        for i = 1 to n-1
            q = max(q, p[i] + MEMOIZED-CUT-ROD-AUX(p, n - i, r)-c)
        q=max(q,p[n])
    r[n] = q
    return q
```

Graph (2 point)

subproblem graph :

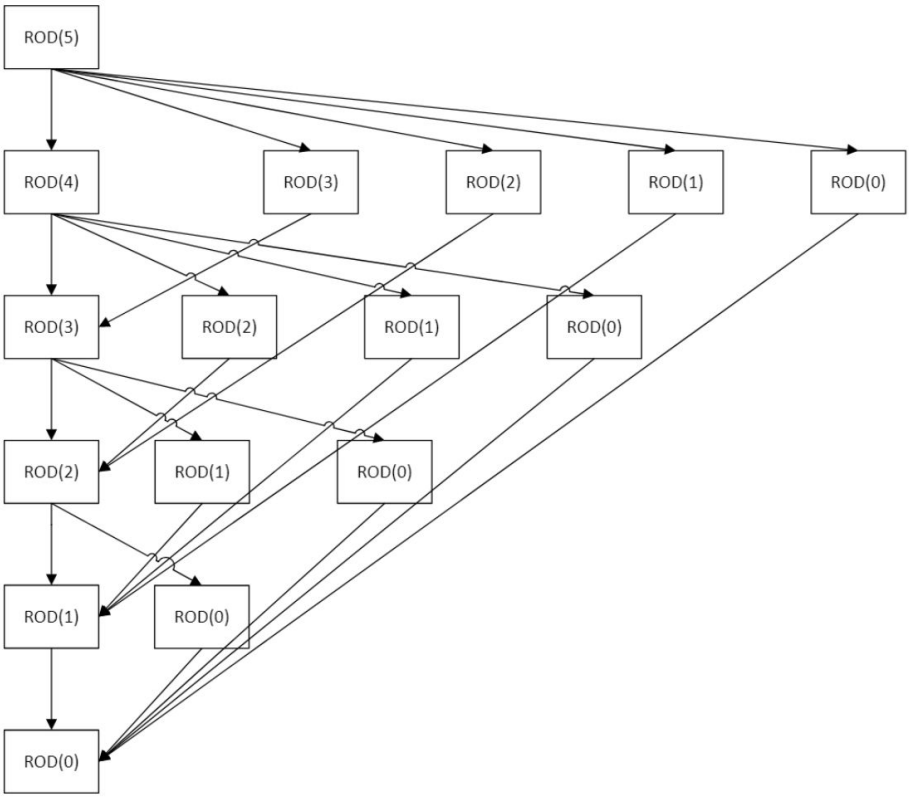


Table (2 point)

	0	1	2	3	4	5
r	0	1	6	8	11	13

	0	1	2	3	4	5
s	0	1	2	3	2	2

7.

A minimum spanning tree can can also be treated as a graph. Once we have the original minimum spanning tree and the new edge with reduced weight, we can add this edge to the minimum spanning tree. Given the property of a MST, the modified MST graph contains a cycle. Now we can use a modified DFS to detect the cycle and also find all the nodes and edges in the cycle. Then remove the maximum edge in the cycle, and the remaining graph will be the new minimum spanning tree.

8.

This question can be solved using minimum spanning tree algorithms.

Let us first make the edge weights negative, so edge weight w_{ij} becomes -w_{ij} (e.g. of 4 becomes -4).

Next, let us define the path cost as the maximum edge weight on the path.

Thus, the original problem of finding maximum bandwidth (the bandwidth of a path is the lowest value on the path) now becomes finding the minimum cost between any two nodes in this new graph.

We then run the minimum spanning tree algorithm. We can prove that the path between a and b in the minimum spanning tree is the path that gives us the minimum cost between two nodes.

Let T be the minimum spanning tree. Let p be the path between two nodes, i and j, in the minimum spanning tree T. If p is not the minimum cost path in the original graph there is another path p' whose cost is less.

Now we remove a maximum weight edge, e, in the path from i to j in the minimum spanning tree T. We now have two connected component in the graph. Let S bet the nodes in one of the components and thus V-S are the nodes in the other component. We look at the minimum cost edge that "crosses the cut" . (i.e. the minimum weight edge that goes from a node in S to a node in V-S). We know it has to be less than e_w (the edge weight of the edge e), since the minimum cost path from i to j was less than e_w.

This contradicts T being a minimum spanning tree. Therefore, there does not exist another path p' whose cost is less.

The time complexity of this algorithm is the same as MST algorithm: O(ElogV)

9.

We assume that the last hotel is the destination, as we don't know the distance between the last hotel to cousin's home.

First we define a helper function to ensure that we don't travel more than 300 miles per day.

$$f(i,k) = \begin{cases} 0 & m_i - m_k \leq 300 \\ \infty & m_i - m_k > 300 \end{cases}$$

Then we define the recursive cost function to compute the lowest cost:

$$C(i) = \begin{cases} h_i & m_i \leq 300 \\ \min\limits_{1 \leq k < i} \left(h_i + C(k) + f(i,k)\right) & m_i > 300 \end{cases}$$

As with all dynamic programming problems, they can be solved in either a bottom-up fashion or a top-down fashion.

Bottom Up Algorithm

```
MIN-COST(n, h, m)
 1:  a ← Initialize an array of size n
 2:  for i ← 1 to n do
 3:      if m[i] ≤ 300 then
 4:          a[i] ← h[i]
 5:      else
 6:          min ← ∞
 7:          for j ← 1 to i − 1 do
 8:              if m[i] − m [j] > 300 then
 9:                  continue
10:              temp ← C(j) + h[i]
11:              if temp < min then
12:                  min ← temp
13:          a[i] ← min
14:  return a[n]
```

Top Down Algorithm

```
MEMOIZED-OPTIMAL-COST(n, h, m)
 1:  memo ← Initialize an array of size n
 2:  for i ← 1 to n do
 3:      memo[i] ← ∞
 4:  return TPD-COST(n, h, m, memo)

TPD-COST(n, h, m, memo)
 1:  if memo[n] < ∞ then
 2:      return memo[n]
 3:  if m[n] ≤ 300 then
 4:      memo[n] = h[n]
 5:      return memo[n]
 6:  q = ∞
 7:  for i ← 1 to n do
 8:      if m[n] − m[i] ≤ 300 then
 9:          q = min(q, TDP-COST(i, h, m, memo)+h[n])
10:  memo[n] = q
11:  return memo[n]
```