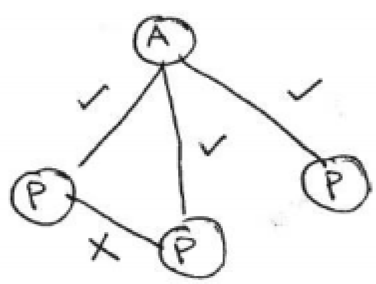
1. (3 points) Suppose all golf players are either professionals or amateurs and between any pair of golf players there may or may not be a rivalry. Suppose we are given *n* golf players and a list of *r* pairs of golf players for which there are rivalries (note that, you can represent this information as an undirected graph). Give an *O*(*n* + *r*)-time algorithm that determines whether it is possible to designate some of the golf players as professionals and the remainders as amateurs such that each rivalry is between a professional and an amateur. If it is possible to perform such a designation, your algorithm should produce it. You do not need to provide pseudo-code. Explain in detail how your algorithm will work.

   Hint: You can perform BFS to visit all vertices and compute distances. What would odd and even distances mean?

   Answer: -

   Create an undirected graph where every vertex represents a golf participant and every aspect represents a rivalry. The diagram will have n vertices and r edges. Perform as many BFS's as wanted to go to all vertices. Assign all golf gamers whose distance is even to be experts and all golf gamers whose distance is peculiar to be amateurs. Then test every aspect to confirm that it goes between a expert and an amateur. If in the given plan contention continually exists between experts and amateurs and no longer between amateurs and amateurs or professions and professionals, then node designation after BFS will be enough. However, if competition additionally exists between specialists and specialists or between amateurs and amateurs then no designation of nodes such that competition exists between specialists and amateurs is possible. This can be finished through going via every aspect to take a look at that contrary cease nodes of this aspect are of distinct type. For example, in the following graph, whilst checking every part after BFS, there is additionally a contention between expert and professional. That ability no designation of nodes, such that competition is usually between professionals and amateurs, is possible in this case.

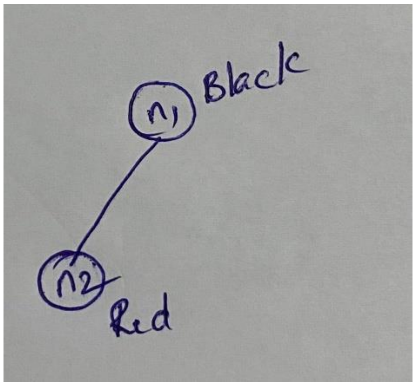
   

   By this can say time complexity is O(n + r) time when we use BFS, it takes O(n) time to identify each golf player as a professional or amateur and O(r) time to check each edge. Therefore total time taken is O(n + r).

2. (3 points) Consider a red-black tree formed by inserting n nodes with RB-INSERT. Argue that if n > 1, the tree has at least one red node.

   Answer:

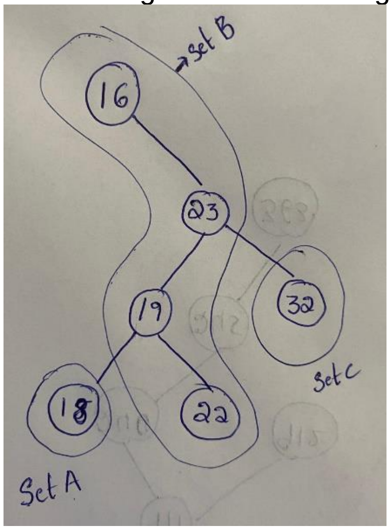
   Suppose for n =2 as root node is black as every time if the node is inserted it's always red in color. If violations occur due to it recoloring is done which leaves one node as red. By this we can conclude we have at least one red node.

3. (3 points) Professor Bunyan thinks he has discovered a remarkable property of binary search trees. Suppose that the search for key k in a binary search tree ends up in a leaf. Consider three sets: A, the keys to the left of the search path; B, the keys on the search path; and C, the keys to the right of the search path. Professor Bunyan claims that any three keys a ∈ A, b ∈ B, and c ∈ C must satisfy a ≤ b ≤ c. Give a smallest possible counterexample to the professor's claim.

   Answer:

   Following example is counter example to to Professor Bunyan's claim.
   Where set A = {18}, -- Left to search key
   Set B = {16,23,19,22}
   Set C = {32} – Right to search key
   Which we got while searching for key 22.

   From above example, as given a ∈ A, b ∈ B, and c ∈ C must satisfy a ≤ b ≤ c, but in our case 18 ∈A, 16 ∈ B but 18 > 16, by this we can conclude **claim is wrong**.

4. (3 points) Find an optimal parenthesization of a matrix-chain product whose sequence of dimensions is {5; 10; 3; 12; 5; 50; 6}.
Answer:
We have:
p0 = 5
p1 = 10
p2 = 3
p3 = 12
p4 = 5
p5 = 50
p6 = 6

The corresponding matrices are:
A1 – 5x10
A2 – 10x3
A3 – 3x12
A4 – 12x5
A4 – 5x50
A5 – 50x6

From the algorithm, we have for all x, m[x,x] = 0
m[1,2] = m[1,1] + m[2,2] + p0 + p1 + p2
m[1,2] = 0 + 150
m[1,2] = 150

m[3,4] = m[3,3] + m[4,4] + p2 * p3 * p4 m[3,4] = 0 + 180
m[3,4] = 180

m[4,5] = m[4,4] + m[5,5] + p3 * p4 * p5 m[4,5] = 0 + 3000
m[4,5] = 3000

m[5,6] = m[5,5] + m[6,6] + p4 * p5 * p6 m[5,6] = 0 + 1500
m[5,6] = 1500

m[1,3] = min of { m[1,1] + m[2,3] + p0 * p1 * p3 = 750 }
{ m[1,2] + m[3,3] + p0 * p2 * p3 = 330 }

m[2,4] = min of **{ m[2,2] + m[3,4] + p1 * p2 * p4 = 330}**
{ m[2,3] + m[4,4] + p1 * p3 * p4 = 960}

m[3,5] = min of { m[3,3] + m[4,5] + p2 * p3 * p5 = 4800}
{ m[3,4] + m[5,5] + p2 * p4 * p5 = 930 }

m[4,6] = min of **{ m[4,4] + m[5,6] + p3 * p4 * p6 = 1860 }**
{ m[4,5] + m[6,6] + p3 * p5 * p6 = 6600 }

m[1,4] = min of { m[1,1] + m[2,4] + p0 * p1 * p4 = 580 }

{ m[1,2] + m[3,4] + p0 * p2 * p4 = 405 }
{ m[1,3] + m[4,4] + p0 * p3 * p4 = 630 }

m[2,5] = min of **{ m[2,2] + m[3,5] + p1 * p2 * p5 = 2430 }**
{ m[2,3] + m[4,5] + p1 * p3 * p5 = 9360 }
{ m[2,4] + m[5,5] + p1 * p4 * p5 = 2830 }

m[3,6] = min of { m[3,3] + m[4,6] + p2 * p3 * p6 = 2076 }
{ m[3,4] + m[5,6] + p2 * p4 * p6 = 1770 }
{ m[3,5] + m[6,6] + p2 * p5 * p6 = 1830 }

m[1,5] = min of { m[1,1] + m[2,5] + p0 * p1 * p5 = 4930 }
{ m[1,2] + m[3,5] + p0 * p2 * p5 = 1830 }
{ m[1,3] + m[1,4] + p0 * p3 * p5 = 6330 }
{ m[1,4] + m[1,5] + p0 * p4 * p5 = 1655 }

m[2,6] = min of **{ m[2,2] + m[3,6] + p1 * p2 * p6 = 1950 }**
{ m[2,3] + m[4,6] + p1 * p3 * p6 = 2940 }
{ m[2,4] + m[5,6] + p1 * p4 * p6 = 2130 }
{ m[2,5] + m[6,6] + p1 * p5 * p6 = 5430 }

m[1,6] = min of { m[1,1] + m[2,6] + p0 * p1 * p6 = 2250 }
{ m[1,2] + m[3,6] + p0 * p2 * p6 = 2010 }
{ m[1,3] + m[4,6] + p0 * p3 * p6 = 2550 }
{ m[1,4] + m[5,6] + p0 * p4 * p6 = 2055 }
{ m[1,5] + m[6,6] + p0 * p5 * p6 = 3155 }

M	1	2	3	4	5	6
6	2010	1950	1770	1840	1500	0
5	1655	2430	930	3000	0	
4	405	330	180	0		
3	330	360				
2	150					
1	0					

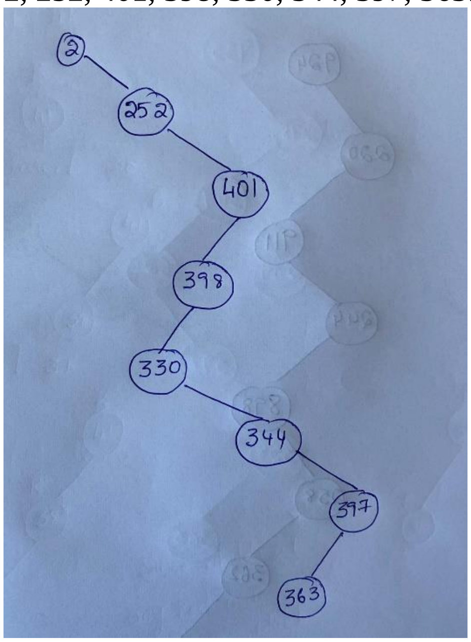
And using this, S table is constructed as follows:

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

The minimum cost is therefore **2010** and the optimal parenthesization is:
((A1 * A2) * (A3 * A4) * (A5 * A6))

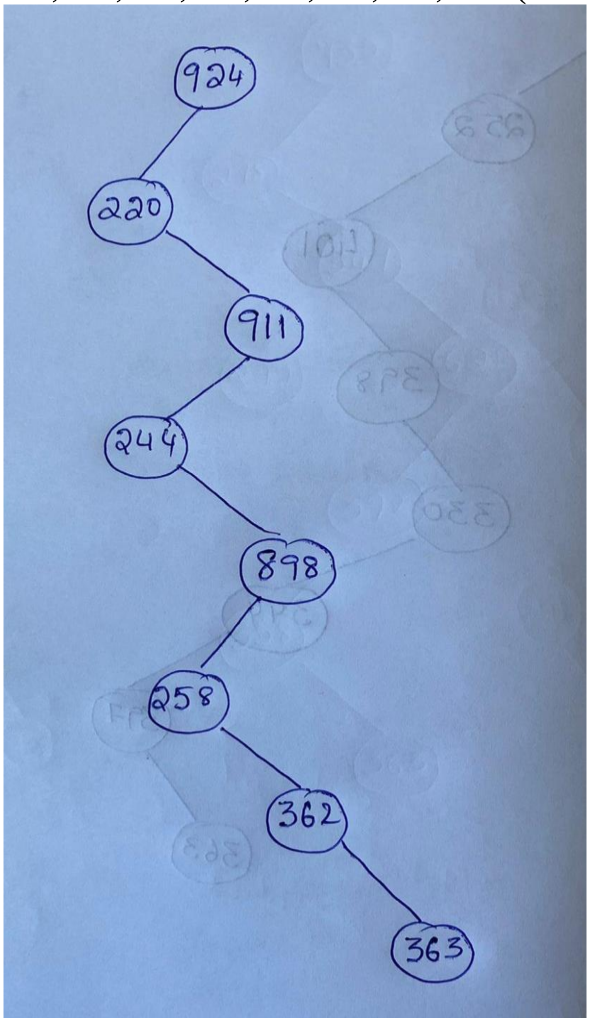
5. (3 points) Suppose that we have numbers between 1 and 1000 in a binary search tree, and we want to search for the number 363. Which of the following sequences could *not* be the sequence of nodes examined?
   Anwser: - c,e

   a. 2, 252, 401, 398, 330, 344, 397, 363. (**Examined**)

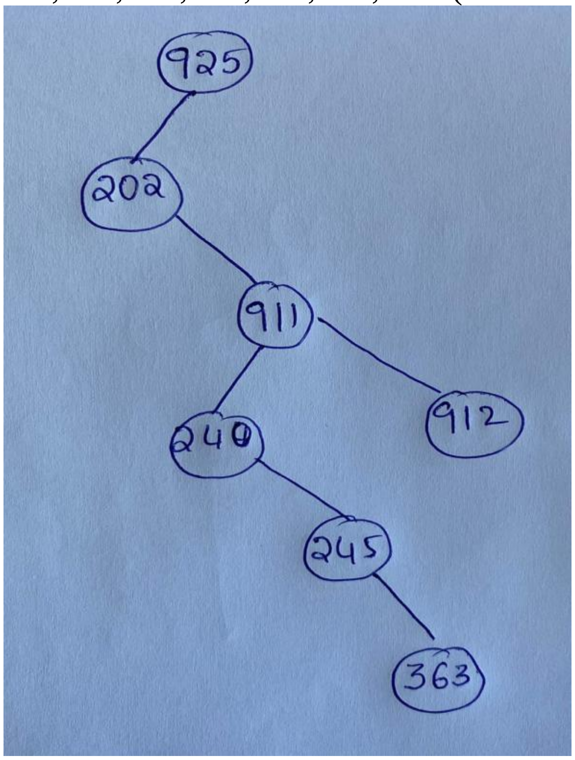
   If we construct a binary search tree with given sequence of keys and start searching for **363** we visit each every key BST in given sequence.

b. 924, 220, 911, 244, 898, 258, 362, 363. (**Examined**)



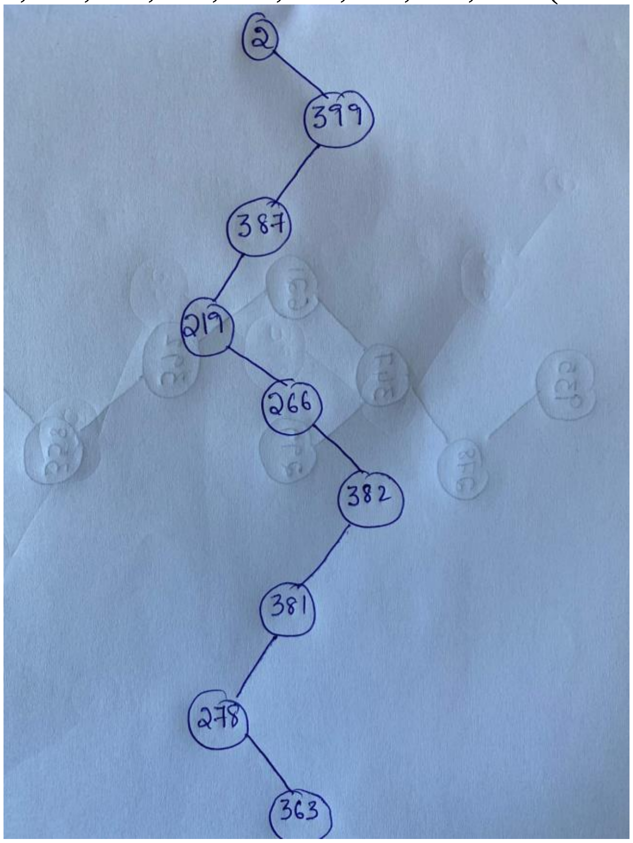
If we construct a binary search tree with given sequence of keys and start searching for **363** we visit each every key BST in given sequence.

c. 925, 202, 911, 240, 912, 245, 363. (**Not Examined**)



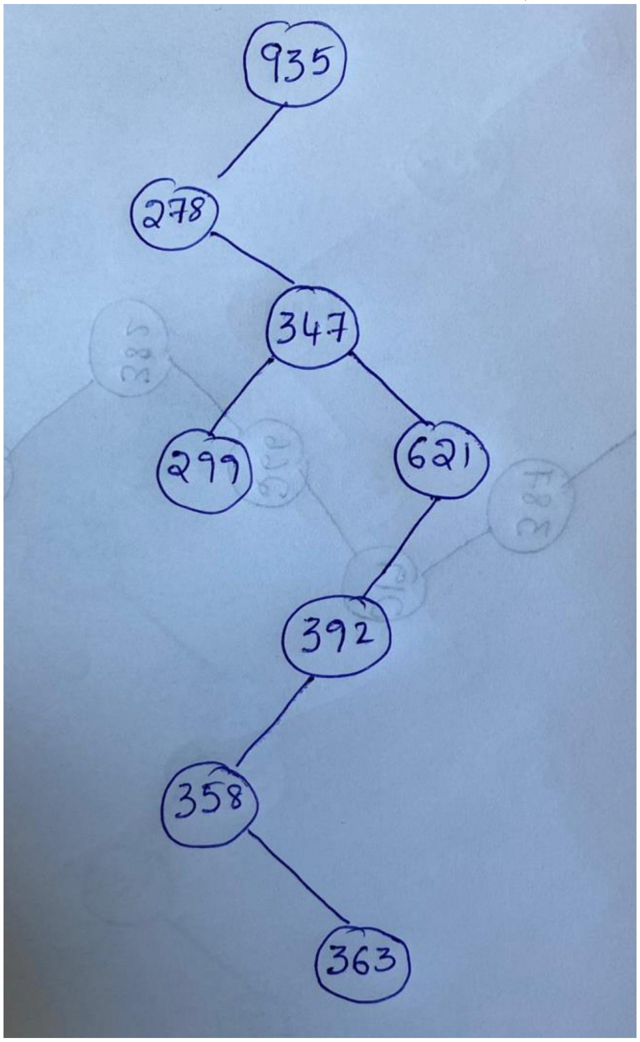
If we construct a binary search tree with given sequence of keys and start searching for **363** we don't visit each every key BST in given sequence because **912** is greater than 363 and a right child of **911.**

d. 2, 399, 387, 219, 266, 382, 381, 278, 363. (**Examined**)



If we construct a binary search tree with given sequence of keys and start searching for **363** we visit each every key BST in given sequence.

e. 935, 278, 347, 621, 299, 392, 358, 363. (**Not Examined**)



If we construct a binary search tree with given sequence of keys and start searching for **363** we visit not each every key BST in given sequence because **299** is smaller than **363** and left child of **347.**