

Codeium: Refactor | Explain

```

3  public class MatrixchainMultiplication {
4      //Given a sequence of matrices, find the most efficient way to multiply these matrices together.
5      //The problem is not actually to perform the multiplications, but merely to decide in which order
6      // to perform the multiplications.
7      //We have many options to multiply a chain of matrices because matrix multiplication is associative.
8      //In other words, no matter how we parenthesize the product, the result will be the same.
9      //For example, if we had four matrices A, B, C, and D, we would have:
10     // (ABC)D = (AB)(CD) = A(BCD) = ....
11     //However, the order in which we parenthesize the product affects the number of simple arithmetic
12     // operations needed to compute the product, or the efficiency.
13     //For example, suppose A is a 10 x 30 matrix, B is a 30 x 5 matrix, and C is a 5 x 60 matrix.
14     //Then,
15
16     // (AB)C = (10x30x5) + (10x5x60) = 1500 + 3000 = 4500 operations
17     // A(BC) = (30x5x60) + (10x30x60) = 9000 + 18000 = 27000 operations.
18     //Clearly the first parenthesization requires less number of operations.
19     //Given an array p[] which represents the chain of matrices such that the ith matrix Ai is of
20     //dimension p[i-1] x p[i]. We need to write a function MatrixChainOrder() that should return the
21     //minimum number of multiplications needed to multiply the chain.
22
23     //Approach: Recursive
24     Codeium: Refactor | Explain | X
25     public static void main(String[] args) {
26         int[] arr = {1, 2, 3, 4, 3};
27         System.out.println(matrixChainOrder(arr, 1, arr.length - 1));
28         System.out.println("matrixChainOrderMemo ->"+matrixChainOrderMemo(arr, 1, arr.length - 1));
29         System.out.println("matrixChainOrderDP ->"+matrixChainOrderDP(arr));
30     }

```

Codeium: Refactor | Explain | Generate Javadoc | X

```

31     public static int matrixChainOrder(
32         int[] arr, int i, int j) {
33         if (i >= j) {
34             return 0;
35         }
36
37         int min = Integer.MAX_VALUE;
38         for (int k = i; k < j; k++) {
39             int count = matrixChainOrder(arr, i, k) +
40                 matrixChainOrder(arr, k + 1, j)
41                 + arr[i - 1] * arr[k] * arr[j];
42             if (count < min) {
43                 min = count;
44             }
45         }
46         return min;
47     }

```

49

//Approach: Memoization

Codeium: Refactor | Explain | ✕

50

public static int matrixChainOrderMemo(

51

int[] arr, int i, int j) {

52

int[][] dp = new int[arr.length + 1][arr.length + 1];

53

return matrixChainOrderMemo(arr, i, j, dp);

54

}

55

Codeium: Refactor | Explain | Generate Javadoc | ✕

56

public static int matrixChainOrderMemo(

57

int[] arr, int i, int j, int[][] dp) {

58

if (i >= j) {

59

return 0;

60

}

61

62

if (dp[i][j] != 0) {

63

return dp[i][j];

64

}

65

66

int min = Integer.MAX\_VALUE;

67

for (int k = i; k < j; k++) {

68

int count = matrixChainOrderMemo(arr, i, k, dp) +

69

matrixChainOrderMemo(arr, k + 1, j, dp) +

70

arr[i - 1] \* arr[k] \* arr[j];

71

if (count < min) {

72

min = count;

73

}

74

}

75

dp[i][j] = min;

76

return dp[i][j];

77

}

78

```

79 //Approach: Dynamic Programming
80 Codeium: Refactor | Explain | ✕
81 public static int matrixChainOrderDP(int[] arr) {
82     int n = arr.length;
83     int[][] dp = new int[n][n];
84
85     for (int i = 1; i < n; i++) {
86         dp[i][i] = 0;
87     }
88
89     for (int l = 2; l < n; l++) {
90         for (int i = 1; i < n - l + 1; i++) {
91             int j = i + l - 1;
92             if (j == n) {
93                 continue;
94             }
95             dp[i][j] = Integer.MAX_VALUE;
96             for (int k = i; k < j; k++) {
97                 int count = dp[i][k] + dp[k + 1][j] + arr[i - 1] * arr[k] * arr[j];
98                 if (count < dp[i][j]) {
99                     dp[i][j] = count;
100                 }
101             }
102         }
103     }
104     return dp[1][n - 1];
105 }
106

```

30

matrixChainOrderMemo ->30

matrixChainOrderDP ->30

```

3 Codeium: Refactor | Explain
4 public class BooleanParenthesis {
5     //Given a boolean expression with following symbols.
6     // Symbols 'T' ---> true 'F' ---> false And
7     // following operators filled between symbols // Operators & ---> boolean AND | ---> boolean OR ^ ---> boolean XOR
8     // Count the number of ways we can parenthesize the expression so that the value of expression evaluates to true.
9     // For Example:
10    // Expression: T|T&F^T
11    // Ways: ((T|T)&(F^T)), (T|(T&(F^T))), (((T|T)&F)^T)
12    // Expression: T^F|F
13    // Ways: (T^(F|F)), ((T^F)|F)
14    // Expression: T|F^T&T
15    // Ways: (T|(F^(T&T))), (((T|F)^T)&T), (T|((F^T)&T)), (T|((F^T)&T))
16
17    //Approach: Recursive
18    Codeium: Refactor | Explain | ✕
19    public static void main(String[] args) {
20        String s = "T|T&F^T";
21        System.out.println(countWays(s, 0, s.length() - 1, true));
22    }
23
24    Codeium: Refactor | Explain | Concepts | Undo | ✕

```

```

22  public static int countWays(
23      String s, int i, int j, boolean isTrue) {
24      if (i > j) {
25          return 0;
26      }
27
28      if (i == j) {
29          if (isTrue) {
30              return s.charAt(i) == 'T' ? 1 : 0;
31          } else {
32              return s.charAt(i) == 'F' ? 1 : 0;
33          }
34      }
35
36      int ans = 0;
37      for (int k = i + 1; k < j; k += 2) {
38          int lT = countWays(s, i, k - 1, true);
39          int lF = countWays(s, i, k - 1, false);
40          int rT = countWays(s, k + 1, j, true);
41          int rF = countWays(s, k + 1, j, false);
42
43          if (s.charAt(k) == '&') {
44              if (isTrue) {
45                  ans += lT * rT;
46              } else {
47                  ans += lT * rF + lF * rT + lF * rF;
48              }
49          } else if (s.charAt(k) == '|') {
50              if (isTrue) {
51                  ans += lT * rT + lT * rF + lF * rT;
52              } else {
53                  ans += lF * rF;
54              }
55          } else if (s.charAt(k) == '^') {
56              if (isTrue) {
57                  ans += lT * rF + lF * rT;
58              } else {
59                  ans += lT * rT + lF * rF;
60              }
61          }
62      }
63      return ans;
64  }
65

```

```

66 //Approach: Memoization
67 Codeium: Refactor | Explain | ✕
68 public static int countWaysMemo(
69     String s, int i, int j, boolean isTrue) {
70     int[][][] dp = new int[s.length() + 1][s.length() + 1][2];
71     return countWaysMemo(s, i, j, isTrue, dp);
72 }
73 Codeium: Refactor | Explain | Generate Javadoc | ✕
74 public static int countWaysMemo(
75     String s, int i, int j, boolean isTrue, int[][][] dp) {
76     if (i > j) {
77         return 0;
78     }
79     if (i == j) {
80         if (isTrue) {
81             return s.charAt(i) == 'T' ? 1 : 0;
82         } else {
83             return s.charAt(i) == 'F' ? 1 : 0;
84         }
85     }
86     if (dp[i][j][isTrue ? 1 : 0] != 0) {
87         return dp[i][j][isTrue ? 1 : 0];
88     }
89
90     int ans = 0;
91     for (int k = i + 1; k < j; k += 2) {
92         int lT = countWaysMemo(s, i, k - 1, true, dp);
93         int lF = countWaysMemo(s, i, k - 1, false, dp);
94         int rT = countWaysMemo(s, k + 1, j, true, dp);
95         int rF = countWaysMemo(s, k + 1, j, false, dp);
96
97         if (s.charAt(k) == '&') {
98             if (isTrue) {
99                 ans += lT * rT;
100             } else {
101                 ans += lT * rF + lF * rT + lF * rF;
102             }
103         } else if (s.charAt(k) == '|') {
104             if (isTrue) {
105                 ans += lT * rT + lT * rF + lF * rT;
106             } else {
107                 ans += lF * rF;
108             }
109         } else if (s.charAt(k) == '^') {
110             if (isTrue) {
111                 ans += lT * rF + lF * rT;
112             } else {
113                 ans += lT * rT + lF * rF;
114             }
115         }
116     }
117     return dp[i][j][isTrue ? 1 : 0] = ans;
118 }
119
120

```

```

121 //Approach: Tabulation
122 Codeium: Refactor | Explain | X
123 public static int countWaysTab(String s, int i, int j, boolean isTrue) {
124     int[][][] dp = new int[s.length() + 1][s.length() + 1][2];
125     for (int gap = 0; gap < s.length(); gap++) {
126         for (i = 0, j = gap; j < s.length(); i++, j++) {
127             if (i == j) {
128                 if (isTrue) {
129                     dp[i][j][1] = s.charAt(i) == 'T' ? 1 : 0;
130                 } else {
131                     dp[i][j][0] = s.charAt(i) == 'F' ? 1 : 0;
132                 }
133             } else {
134                 for (int k = i + 1; k < j; k += 2) {
135                     int lT = dp[i][k - 1][1];
136                     int lF = dp[i][k - 1][0];
137                     int rT = dp[k + 1][j][1];
138                     int rF = dp[k + 1][j][0];
139
140                     if (s.charAt(k) == '&') {
141                         if (isTrue) {
142                             dp[i][j][1] += lT * rT;
143                         } else {
144                             dp[i][j][0] += lT * rF + lF * rT + lF * rF;
145                         }
146                     } else if (s.charAt(k) == '|') {
147                         if (isTrue) {
148                             dp[i][j][1] += lT * rT + lT * rF + lF * rT;
149                         } else {
150                             dp[i][j][0] += lF * rF;
151                         }
152                     } else if (s.charAt(k) == '^') {
153                         if (isTrue) {
154                             dp[i][j][1] += lT * rF + lF * rT;
155                         } else {
156                             dp[i][j][0] += lT * rT + lF * rF;
157                         }
158                     }
159                 }
160             }
161         }
162     }
163     return dp[0][s.length() - 1][isTrue ? 1 : 0];

```

**.MatrixChainMultiplication.BooleanParenthesis**

```

3 public class EggDroppingProblem {
4     //Given a certain number of floors and a certain number of eggs, find the minimum number of attempts needed to
5     // find the threshold floor from which the egg breaks.
6     //An egg that survives a fall can be used again.
7     //A broken egg must be discarded.
8     //For example, if the threshold is 16 and we have 2 eggs, then the minimum number of attempts is 4.
9     //We can drop from floor 10, 16, 13, 14, 15
10    //Approach: Recursive
11    Codeium: Refactor | Explain | X
12    public static void main(String[] args) {
13        int floors = 16;
14        int eggs = 2;
15        System.out.println(minAttempts(floors, eggs));
16    }
17
18    Codeium: Refactor | Explain | Generate Javadoc | X
19    public static int minAttempts(int floors, int eggs) {
20        if (floors == 0 || floors == 1 || eggs == 1) {
21            return floors;
22        }
23
24        int min = Integer.MAX_VALUE;
25        for (int i = 1; i <= floors; i++) {
26            int count = 1 + Math.max(minAttempts(i - 1,
27                                    eggs - 1),
28                                    minAttempts(floors - i, eggs));
29            if (count < min) {
30                min = count;
31            }
32        }
33        return min;
34    }
35
36    //Approach: Memoization
37    Codeium: Refactor | Explain | X
38    public static int minAttemptsMemo(int floors, int eggs) {
39        int[][] dp = new int[floors + 1][eggs + 1];
40        return minAttemptsMemo(floors, eggs, dp);
41    }
42
43    Codeium: Refactor | Explain | Generate Javadoc | X
44    public static int minAttemptsMemo(int floors, int eggs, int[][] dp) {
45        if (floors == 0 || floors == 1 || eggs == 1) {
46            return floors;
47        }
48
49        if (dp[floors][eggs] != 0) {
50            return dp[floors][eggs];
51        }
52
53        int min = Integer.MAX_VALUE;
54        for (int i = 1; i <= floors; i++) {
55            int count = 1 + Math.max(minAttemptsMemo(i - 1, eggs - 1, dp),
56                                    minAttemptsMemo(floors - i, eggs, dp));
57            if (count < min) {
58                min = count;
59            }
60        }
61        dp[floors][eggs] = min;
62        return dp[floors][eggs];
63    }

```



```

61 //Approach: Dynamic Programming
62 Codeium: Refactor | Explain | X
63 public static int minAttemptsDP(int floors, int eggs) {
64     int[][] dp = new int[floors + 1][eggs + 1];
65     for (int i = 1; i <= floors; i++) {
66         dp[i][1] = i;
67     }
68     for (int i = 1; i <= eggs; i++) {
69         dp[1][i] = 1;
70     }
71     for (int i = 2; i <= floors; i++) {
72         for (int j = 2; j <= eggs; j++) {
73             dp[i][j] = Integer.MAX_VALUE;
74             for (int k = 1; k <= i; k++) {
75                 int count = 1 + Math.max(dp[k - 1][j - 1], dp[i - k][j]);
76                 if (count < dp[i][j]) {
77                     dp[i][j] = count;
78                 }
79             }
80         }
81     }
82     return dp[floors][eggs];
83 }
84 }
85 }
86 }
87 }
88 }
89 }

```

.MatrixChainMultiplication.EggDroppingProblemn

Codeium: Refactor | Explain

```
3 public class PallindromePartition {
4     // Given a string, a partitioning of the string is a palindrome partitioning if every substring of the partition is a palindrome.
5     // For example, "aba|b|bbabb|a|b|aba" is a palindrome partitioning of "ababbbabbababa".
6     // Determine the fewest cuts needed for palindrome partitioning of a given string.
7     // For example, minimum 3 cuts are needed for "ababbbabbababa".
8
9     //Approach: Recursive
10    Codeium: Refactor | Explain | X
11    public static void main(String[] args) {
12        String s = "ababbbabbababa";
13        System.out.println(minCuts(s, 0, s.length() - 1));
14    }
15
16    Codeium: Refactor | Explain | Generate Javadoc | X
17    public static int minCuts(String s, int i, int j) {
18        if (i >= j) {
19            return 0;
20        }
21
22        if (isPallindrome(s, i, j)) {
23            return 0;
24        }
25
26        int min = Integer.MAX_VALUE;
27        for (int k = i; k < j; k++) {
28            int count = minCuts(s, i, k) + minCuts(s, k + 1, j) + 1;
29            if (count < min) {
30                min = count;
31            }
32        }
33        return min;
34    }
```

```

34 public static boolean isPallindrome(String s, int i, int j) {
35     while (i < j) {
36         if (s.charAt(i++) != s.charAt(j--)) {
37             return false;
38         }
39     }
40     return true;
41 }

```

42  
43 //Approach: Memoization

Codeium: Refactor | Explain | X

```

44 public static int minCutsMemo(String s, int i, int j) {
45     int[][] dp = new int[s.length() + 1][s.length() + 1];
46     return minCutsMemo(s, i, j, dp);
47 }

```

Codeium: Refactor | Explain | Generate Javadoc | X

```

49 public static int minCutsMemo(String s, int i, int j, int[][] dp) {
50     if (i >= j) {
51         return 0;
52     }
53
54     if (isPallindrome(s, i, j)) {
55         return 0;
56     }
57
58     if (dp[i][j] != 0) {
59         return dp[i][j];
60     }
61
62     int min = Integer.MAX_VALUE;
63     for (int k = i; k < j; k++) {
64         int count = minCutsMemo(s, i, k, dp) +
65                     minCutsMemo(s, k + 1, j, dp) + 1;
66         if (count < min) {
67             min = count;
68         }
69     }
70     dp[i][j] = min;
71     return dp[i][j];
72 }
73

```

```

73
74 //Approach: Dynamic
75 Codeium: Refactor | Explain | X
76 public static int minCutsDP(String s) {
77     int n = s.length();
78     int[][] dp = new int[n][n];
79
80     for (int i = 1; i < n; i++) {
81         dp[i][i] = 0;
82     }
83
84     for (int l = 2; l < n; l++) {
85         for (int i = 1; i < n - l + 1; i++) {
86             int j = i + l - 1;
87             if (isPalindrome(s, i, j)) {
88                 dp[i][j] = 0;
89             } else {
90                 dp[i][j] = Integer.MAX_VALUE;
91                 for (int k = i; k < j; k++) {
92                     int count = dp[i][k] + dp[k + 1][j] + 1;
93                     if (count < dp[i][j]) {
94                         dp[i][j] = count;
95                     }
96                 }
97             }
98         }
99     }
100     return dp[1][n - 1];
101 }
102
103 //Approach: Try to find it in most optimized way
104
105 }
106
107

```

`.MatrixChainMultiplication.PallindromePartition`

Codeium: Refactor | Explain

```
3 public class ScrambledString {
4     //Given two strings s1 and s2 of the same length, determine if s2 is a scrambled string of s1.
5     //A string is a scrambled string of another string if it can be obtained by swapping some characters of the other string.
6     //Example: s1 = "great", s2 = "rgeat" -> true
7     //Example: s1 = "abcde", s2 = "caebd" -> false
8 }
```

Codeium: Refactor | Explain | Generate Javadoc | ✕

```
9 public static void main(String[] args) {
10     String s1 = "great";
11     String s2 = "rgeat";
12     System.out.println(isScrambled(s1, s2));
13 }
14
15 //Approach: Recursive
```

Codeium: Refactor | Explain | ✕

```
16 public static boolean isScrambled(String s1, String s2) {
17     if (s1.equals(s2)) {
18         return true;
19     }
20
21     if (s1.length() <= 1) {
22         return false;
23     }
24
25     int n = s1.length();
26     boolean flag = false;
27     for (int i = 1; i < n; i++) {
28         if ((isScrambled(s1.substring(0, i), s2.substring(n - i))
29             && isScrambled(s1.substring(i), s2.substring(0, n - i))) ||
30             (isScrambled(s1.substring(0, i), s2.substring(0, i)) &&
31             isScrambled(s1.substring(i), s2.substring(i)))) {
32             flag = true;
33             break;
34         }
35     }
36     return flag;
37 }
38 }
```

```

39 //Approach: Memoization
    Codeium: Refactor | Explain | X
40 public static boolean isScrambledMemo(String s1, String s2) {
41     int n = s1.length();
42     int[][][] dp = new int[n + 1][n + 1][n + 1];
43     return isScrambledMemo(s1, s2, dp, 0, 0, n);
44 }
45
    Codeium: Refactor | Explain | Generate Javadoc | X
46 public static boolean isScrambledMemo(
47     String s1, String s2, int[][][] dp, int i, int j, int len) {
48     if (i >= len || j >= len) {
49         return false;
50     }
51
52     if (s1.substring(i).equals(s2.substring(j))) {
53         return true;
54     }
55
56     if (len == 1) {
57         return s1.charAt(i) == s2.charAt(j);
58     }
59
60     if (dp[i][j][len] != 0) {
61         return dp[i][j][len] == 1;
62     }
63
64     boolean flag = false;
65     for (int k = 1; k < len; k++) {
66         if ((isScrambledMemo(s1, s2, dp, i, j + len - k, k) &&
67             isScrambledMemo(s1, s2, dp, i + k, j, len - k)) ||
68             (isScrambledMemo(s1, s2, dp, i, j, k) &&
69             isScrambledMemo(s1, s2, dp, i + k, j + k, len - k))) {
70             flag = true;
71             break;
72         }
73     }
74     dp[i][j][len] = flag ? 1 : -1;
75     return flag;
76 }
77

```

```

77
78 //Approach: Dynamic Programming
79 Codeium: Refactor | Explain | ✕
80 public static boolean isScrambledDP(String s1, String s2) {
81     int n = s1.length();
82     boolean[][][] dp = new boolean[n][n][n + 1];
83
84     for (int len = 1; len <= n; len++) {
85         for (int i = 0; i <= n - len; i++) {
86             for (int j = 0; j <= n - len; j++) {
87                 if (len == 1) {
88                     dp[i][j][len] = s1.charAt(i) == s2.charAt(j);
89                 } else {
90                     for (int k = 1; k < len; k++) {
91                         if ((dp[i][j + len - k][k] && dp[i + k][j][len - k]) ||
92                             (dp[i][j][k] && dp[i + k][j + k][len - k])) {
93                             dp[i][j][len] = true;
94                             break;
95                         }
96                     }
97                 }
98             }
99         }
100     }
101     return dp[0][0][n];
102 }
103
104
105 }
106

```

`.MatrixChainMultiplication.ScrambledString`  
**true**