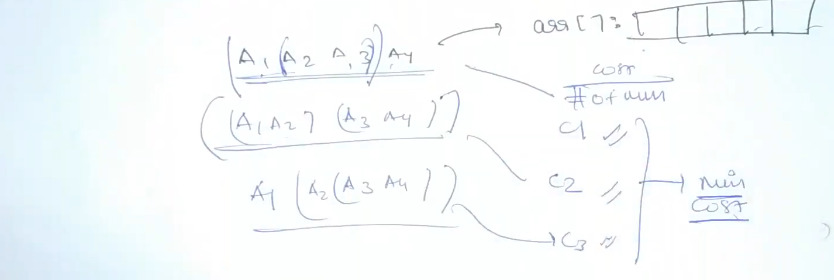
**1) MCM Identification and general format.**

Array or String will be given. Suppose i is the first index of the array and j is the last index of the array. Then we will split the array in i to k, and k+1 to j. this will give the two-temp array. We will combine both temp array to find out resultant array.

If two matric a x b and c x d is given. We can multiply these 2 matrices if and only if b == c and new resultant matrix will be of dimension a x d

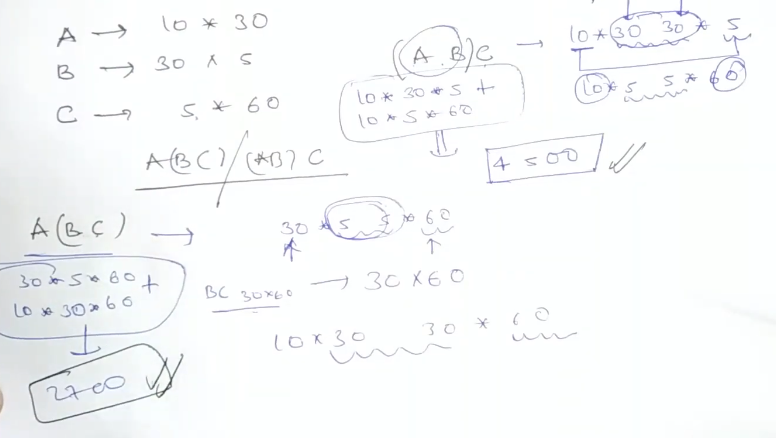
And cost of the matrix multiplication will be a \* b\* d( here b and c is equal so we will consider only once).



Here A1, A2, A3, A4 matrix is given. Dimension of the matrix we will get it by using the array. We will put the bracket in between matrix and find the cost of the matrix.

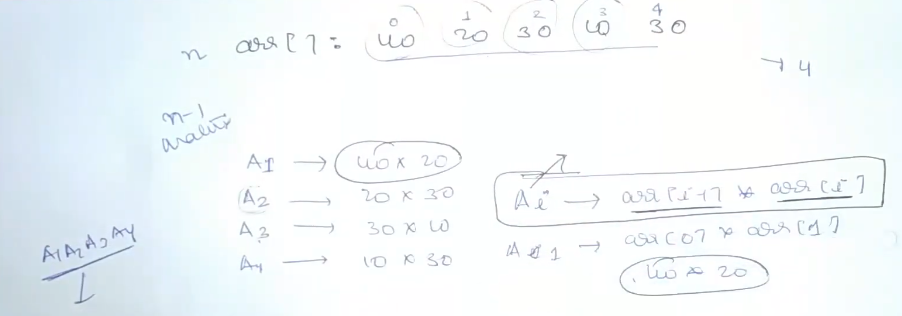
I above example, suppose we have put the bracket like 1st then cost is coming C1, if we are putting bracket like 2nd then cost is coming C2 and if we are putting bracket like 3rd one then cost is coming C3.

Then we have to find out which one is minimum out of these 3. That will be the minimum cost of the MCM.



Here in this example we have matrix A(10 \* 30), B(30 \* 5), C(5 \* 60). We have to find out the min cost of this MCM.

If we put the bracket like 1st scenario then we are getting the cost 4500 and in 2nd scenario we are getting the cost 2700. We will return the min of these 2 value that is 2700



How we will find the dimension of the matrix ?

Suppose above array is given and we are given 4 matrix. A1, A2, A3, A4

Then dimension of the matrix will be

A1 = 40 \* 20

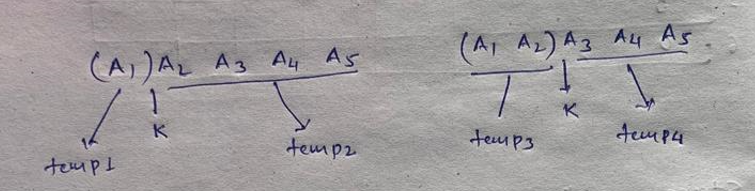
A2 = 20 \* 30

A3 = 30 \* 10

A4 = 10 \* 30

So basically, dimension of the matrix Ai will be

Ai = arr[i-1] \* arr[i]



Here we will divide the array into 2 parts. One is 0th index to K and another kth index to n. cost of multiplying 2 matrix is arr[i-1] \* arr[k] \* arr[j]

And then we will find out the temp answers and in last we will return the minimum from these temp answers.

**2) MCM Recursive approach**

Base condition for the recursion will be

**if** (i >= j) {

**return** 0;

}

Now do the recursive call from 0 to k and k+1 to n for every i

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

**int** temp = solve(arr, i, k) + solve(arr, k + 1, j) + arr[i - 1] \* arr[k] \* arr[j];

min = Math.*min*(min, temp);

}

And cost of multiplying two matrix is arr[i-1] \* arr[k] \* arr[j]

<https://github.com/hareramcse/Datastructure/blob/master/DP/src/com/hs/dp/mcm/MCMRecursive.java>

3) **MCM with memoization**

In Memoization we take the matrix and store the result of the method call. Before doing method call we check the matrix if the method call output is available in the matrix or not. If it is available then we return the value from matrix itself. Ifvalue is not available then we do the method call and store the value into the matrix.

<https://github.com/hareramcse/Datastructure/blob/master/DP/src/com/hs/dp/mcm/MCMWithMemoization.java>

**4) Palindrome Partitioning ||**

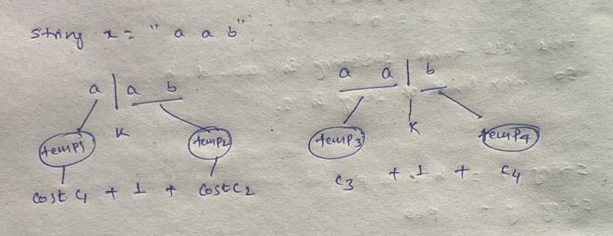
String str = "aab";

Given a string str, partition str such that every substring of the partition is a palindrome.

Return *the minimum cuts needed* for a palindrome partitioning of str

Sol:

Here we will divide the array into 2 parts



One is from i to k and other is from k to n. once divided String is palindrome we stop dividing.

After 1st divide, aab becomes, a and ab. a is palindrome so we stop dividing a

After 2nd divide, ab becomes a and b. here a and b is palindrome.

Cost(ab) = Cost(a) + 1 + cost(ab) = 0 + 1 + cost(ab) = 1 + cost(ab)

Cost(ab) = cost(a) + 1 + cost(b) = 0 + 1 + 0 = 1

So cost(aab) = 1 + 1 = 2

So we did 2 cut and all the String of aab are palindrome.

In the 2nd image.

After 1st divide, aab becomes, aa and b

Here aa and b both are palindrome. So we stop dividing further.

Cost(aab) = cost(aa) + 1 + cost(b)

Here aa is palindrome so it will return 0 so cost(b)

Cost(aab) = 0 + 1 + 0 = 1

So here we did only one cut and all String are palindrome.

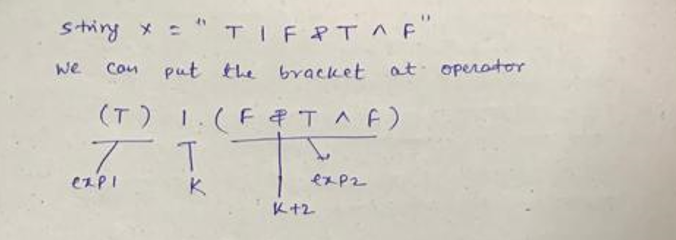
<https://github.com/hareramcse/Datastructure/blob/master/DP/src/com/hs/dp/mcm/PalindromPartitionTabular.java>

5) Evaluate expression to true

String s = "T|F&T^T";

String s is given. We need to find out how many number of ways we can rearrange the operator to make this expression true. Here T stands for True and F stands for False. We can have maximum three operator or, and, xor. We can put the bracket in-between the String to get the value of the String True.

Sol:



Once we put the operator at k index of the String it divides into 2 parts. Expression1 and expression2.

Then we can evaluate both expression and return the result. We can put the operator at operator index and next operator will come at k+2 index.

Suppose we have expression (exp1) ^ (exp2). This is XOR operator.

In XOR operator

T ^ T = False

F ^ F = False

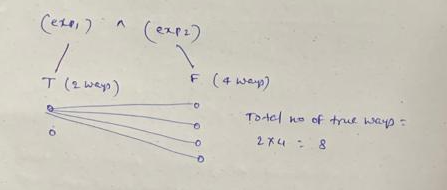
T ^ F = True

F ^ T = True

So here in this case either of the exp1 or exp2 can be true or false to make the whole expression true.

So no of true ways = LT \* RF + LF \* RT ( LT = Left True, RF = Right False, LF = Left False, RT = Right True)

So to find out the True ways…….we need to find out True ways and False ways also.



Suppose exp1 is true in 2 ways and exp2 is true in 4 ways. So whole expression can be true in 2\*4 = 8 ways. Because T^F and F^T both are true. That is why we are using \* here.

So to find out whole expression true how many times…..we need to find out how may ways exp1 can be true and false. Similarly how many ways exp2 can be false and true.

On ith and jth index only T or F will be there and on Kth index operator will be there.

Base condition will be

**if** (i > j)

**return** 0;

Method Signature will be

Solve(String s, int i, int j, isTrue)

This method will be recursivally called for all the operator one by one and one time with true and another time with false.

So with this 4 recursive call we will get lt, lf, rt, rf

And now we will check all the operator how many ways it can be true one by one. And based on the above explanation we will find out how many times whole string can be true.

<https://github.com/hareramcse/Datastructure/blob/master/DP/src/com/hs/dp/mcm/EvaluateExpressionToTrueParenthesis.java>

6) Scramble String

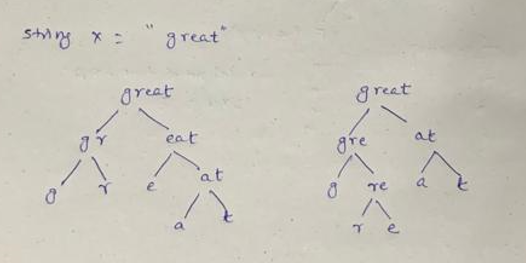
String S1 = "coder";

String S2 = "ocred";

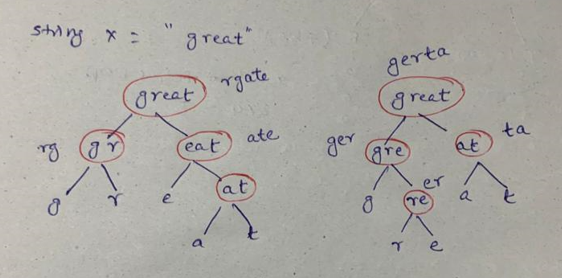
Given two strings s1 and s2 of **the same length**, return true if s2 is a scrambled string of s1, otherwise, return false.

Sol:

We can make the binary tree of the string in any form but leaf node should not be empty.



Here we have made the binary tree in 2 ways. We can make in any ways.



Here we can swap the non leaf node and swap its child node. After swapping the child node replace it with the non leaf node like above.

In the above case great and rgate are scramble strings.

Similarly great and gerta are scramble string.

Non equal length of string cant be scramble string

Empty and equal strings are scramble string

Check the anagram of the string( sort the char of the strings and check if both are equals. If equal then both are scramble string)

Do the substring of s1 and s2 from 0 to i and i to n. and check swap and non swap scenario.

<https://github.com/hareramcse/Datastructure/blob/master/DP/src/com/hs/dp/mcm/ScrambleStringMemoized.java>

7) Egg dropping problem

You are given k identical eggs and you have access to a building with n floors labeled from 1 to n.

You know that there exists a floor f where 0 <= f <= n such that any egg dropped at a floor **higher** than f will **break**, and any egg dropped **at or below** floor f will **not break**.

Each move, you may take an unbroken egg and drop it from any floor x (where 1 <= x <= n). If the egg breaks, you can no longer use it. However, if the egg does not break, you may **reuse** it in future moves.

Return *the****minimum number of drops****that you need to determine****with certainty****what the value of*f is.

Sol:

We can use binary search here.

There will be 2 base condition here.

1. If there is 0 or 1 floor then we need only 0 or 1 drops.
2. If there is 1 egg and n floors then we need n trials.

After this we will use binary search..

**int** breakMove = eggDrop(dp, k - 1, mid - 1);

**int** safeMove = eggDrop(dp, k, n - mid);

res = Math.*min*(res, Math.*max*(breakMove, safeMove) + 1);

if egg breaks then no of eggs will be k-1 and no of floor will be mid-1 because if it breaks at mid-1 then it will break from above floor also. So we will go down from here.

if egg does not break then no of egg will be k only and no of floor will be n-mid because if it did not break from mid then we will check from above the mid floor. So above the mid floor how many floor are there…n-mid only.

<https://github.com/hareramcse/Datastructure/blob/master/DP/src/com/hs/dp/mcm/EggDroppingTabular.java>