50) Matrix Chain Multiplication

13 December 2024 05:48

Given a sequence of matrices, find the most efficient way to multiply these matrices together. The efficient way is the one that involves the least number of multiplications. The dimensions of the matrices are given in an array **arr[]** of size **n** (such that **n** = number of matrices + 1) where the i^{th} matrix has the dimensions **(arr[i-1] x arr[i])**.

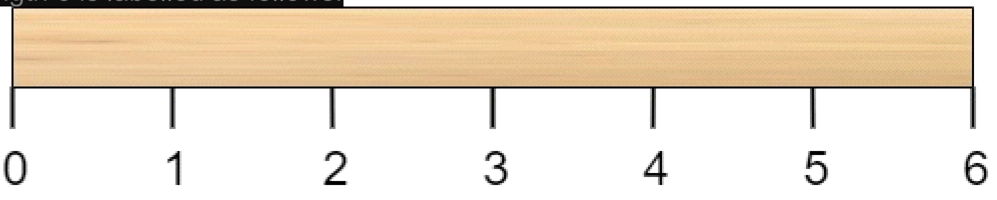
From <https://www.geeksforgeeks.org/problems/matrix-chain-multiplication0303/1?utm_source=youtube&utm_medium=collab_striver_ytdescription&utm_campaign=matrix-chain-multiplication>

```
int helper(int i, int j, vector<int> &arr, vector<vector<int>> &dp){
    if(i == j) return 0;
    int mini = 1e9;
    if(dp[i][j] != -1) return dp[i][j];
    for(int k = i; k < j; k++){
        int steps = arr[i-1]*arr[j]*arr[k] + helper(i, k, arr, dp) + helper(k+1, j, arr, dp);
        mini = min(steps, mini);
    }
    return dp[i][j] = mini;
}
int matrixMultiplication(vector<int> &arr) {
    int n = arr.size();
    vector<vector<int>> dp(n, vector<int>(n, -1));
    return helper(1, n-1, arr, dp);
}
```

51) Minimum cost to cut a stick

13 December 2024 06:23

Given a wooden stick of length n units. The stick is labelled from 0 to n. For example, a stick of length **6** is labelled as follows:



Given an integer array cuts where cuts[i] denotes a position you should perform a cut at.
You should perform the cuts in order, you can change the order of the cuts as you wish.
The cost of one cut is the length of the stick to be cut, the total cost is the sum of costs of all cuts.
When you cut a stick, it will be split into two smaller sticks (i.e. the sum of their lengths is the length of the stick before the cut). Please refer to the first example for a better explanation.

From <https://leetcode.com/problems/minimum-cost-to-cut-a-stick/description/>

```
    int helper(int i, int j, vector<int>& cuts, vector<vector<int>> &dp){
        if(i > j) return 0;
        int mini = 1e9;
        if(dp[i][j] != -1) return dp[i][j];
        for(int k = i; k <= j; k++){
            int cost = cuts[j+1] - cuts[i-1] + helper(i, k-1, cuts, dp) + helper(k+
1, j, cuts, dp);
            mini = min(mini, cost);
        }
        return dp[i][j] = mini;
    }
    int minCost(int n, vector<int>& cuts) {
        int c = cuts.size();
        cuts.push_back(n);
        cuts.insert(cuts.begin(), 0);
        sort(cuts.begin(), cuts.end());
        vector<vector<int>>dp(c+1, vector<int>(c+1, -1));
        return helper(1, c, cuts, dp);
    }
```

52) Burst Balloons

13 December 2024 06:52

You are given n balloons, indexed from 0 to n - 1. Each balloon is painted with a number on it represented by an array nums. You are asked to burst all the balloons.
If you burst the ith balloon, you will get nums[i - 1] * nums[i] * nums[i + 1] coins. If i - 1 or i + 1 goes out of bounds of the array, then treat it as if there is a balloon with a 1 painted on it.
Return *the maximum coins you can collect by bursting the balloons wisely*.

From <https://leetcode.com/problems/burst-balloons/description/>

```
    int helper(int i, int j, vector<int>& nums, vector<vector<int>> &dp){
        if(i > j) return 0;
        int maxi = -1e9;
        if(dp[i][j] != -1) return dp[i][j];
        for(int k = i; k <= j; k++){
            int cost = nums[i-1]*nums[k]*nums[j+1] + helper(i, k-1, nums, dp) + 
helper(k+1, j, nums, dp);
            maxi = max(maxi, cost);
        }
        return dp[i][j] = maxi;
    }
    int maxCoins(vector<int>& nums) {
        int n = nums.size();
        nums.push_back(1);
        nums.insert(nums.begin(), 1);
        vector<vector<int>>dp(n+1, vector<int>(n+1, -1));
        return helper(1, n, nums, dp);
    }
```

53) Boolean Parenthesization

14 December 2024 01:53

Given a boolean expression **s** of length **n** with following symbols.
Symbols
  'T' ---> true
  'F' ---> false
and following operators filled between symbols
Operators
  & ---> boolean AND
  | ---> boolean OR
  ^ ---> boolean XOR
Count the number of ways we can parenthesize the expression so that the value of expression evaluates to true.
Note: The answer can be large, so return it with **modulo 1003**

From <https://www.geeksforgeeks.org/problems/boolean-parenthesization5610/1?itm_source=geeksforgeeks&itm_medium=article&itm_campaign=practice_card>

```
#define ll long long
const int mod = 1003;

int f(int i, int j, int isTrue, string &exp, vector<vector<vector<ll>>> &dp) {
  // Base cases
  if (i > j) return 0;

  if (i == j) {
    if (isTrue) return exp[i] == 'T' ? 1 : 0;
    else return exp[i] == 'F' ? 1 : 0;
  }

  // Check DP table
  if (dp[i][j][isTrue] != -1) return dp[i][j][isTrue];

  ll ways = 0;

  // Iterate through operators
  for (int ind = i + 1; ind <= j - 1; ind += 2) {
    ll lT = f(i, ind - 1, 1, exp, dp) % mod;
    ll lF = f(i, ind - 1, 0, exp, dp) % mod;
    ll rT = f(ind + 1, j, 1, exp, dp) % mod;
    ll rF = f(ind + 1, j, 0, exp, dp) % mod;

    if (exp[ind] == '&') {
      if (isTrue) ways = (ways + (lT * rT) % mod) % mod;
      else ways = (ways + (lF * rT) % mod + (lT * rF) % mod + (lF * rF) % mod) % mod;
    } else if (exp[ind] == '|') {
      if (isTrue) ways = (ways + (lT * rT) % mod + (lT * rF) % mod + (lF * rT) % mod) % mod;
      else ways = (ways + (lF * rF) % mod) % mod;
    } else if (exp[ind] == '^') {
      if (isTrue) ways = (ways + (lT * rF) % mod + (lF * rT) % mod) % mod;
      else ways = (ways + (lT * rT) % mod + (lF * rF) % mod) % mod;
    }
  }

  // Store result in DP table
  return dp[i][j][isTrue] = ways;
}

int countWays(int n, string s) {
  vector<vector<vector<ll>>> dp(n, vector<vector<ll>>(n, vector<ll>(2, -1)));
  return f(0, n - 1, 1, s, dp);
}
```

54) Palindrome Partitioning II

14 December 2024 02:40

Given a string s, partition s such that every
substring
 of the partition is a
palindrome
.
Return *the **minimum** cuts needed for a palindrome partitioning of* s.

From <https://leetcode.com/problems/palindrome-partitioning-ii/description/>

```
    bool isPalindrome(int i, int j, string &s) {
        while (i < j) {
            if (s[i] != s[j]) return false;
            i++;
            j--;
        }
        return true;
    }
    int minPartitions(int i, int n, string &str, vector<int> &dp) {
        if (i == n) return 0;
        if (dp[i] != -1) return dp[i];
        int minCost = INT_MAX;
        for (int j = i; j < n; j++) {
            if (isPalindrome(i, j, str)) {
                int cost = 1 + minPartitions(j + 1, n, str, dp);
                minCost = min(minCost, cost);
            }
        }
        return dp[i] = minCost;
    }
    int minCut(string s) {
        int n = s.size();
        vector<int> dp(n, -1);
        return minPartitions(0, n, s, dp) - 1;
    }
```

55) Partition array for maximum sum

14 December 2024 02:50

Given an integer array arr, partition the array into (contiguous) subarrays of length at most k. After partitioning, each subarray has their values changed to become the maximum value of that subarray.

From <https://leetcode.com/problems/partition-array-for-maximum-sum/description/>

```
int f(int ind, vector<int> &num, int k, vector<int> &dp) {
    int n = num.size();
    if (ind == n) return 0;
    if (dp[ind] != -1) return dp[ind];
    int len = 0;
    int maxi = INT_MIN;
    int maxAns = INT_MIN;
    for (int j = ind; j < min(ind + k, n); j++) {
        len++;
        maxi = max(maxi, num[j]);
        int sum = len * maxi + f(j + 1, num, k, dp);
        maxAns = max(maxAns, sum);
    }
    return dp[ind] = maxAns;
}
int maxSumAfterPartitioning(vector<int>& num, int k) {
    int n = num.size();
    vector<int> dp(n, -1);
    return f(0, num, k, dp);
}
```