1. Longest Common Subsequence

int Solution::solve(string A, string B) {

    int m=A.length(),n=B.length();

    int t[m+1][n+1];

    for(int i=0;i<m+1;i++){

        for(int j=0;j<n+1;j++){

            if(i==0 || j==0) t[i][j]=0;

        }

    }

    for(int i=1;i<m+1;i++){

        for(int j=1;j<n+1;j++){

            if(A[i-1]==B[j-1]) t[i][j]=1+t[i-1][j-1];

            else t[i][j]=max(t[i-1][j],t[i][j-1]);

        }

    }

    return t[m][n];

}

1. Stairs – Max no of ways to climb A stairs by climbing only 1 or 2 at a time

int Solution::climbStairs(int n) {

if (n <= 0) return 0;

if (n == 1 || n == 2) return n;

return climbStairs(n-1) + climbStairs(n-2);

}

int climbStairs(int N)

{

int dp[N+1];

dp[0] = 1

dp[1] = 1

for (i = 2; i <= N; i++){

dp[i] = dp[i-1] + dp[i-2];

}

return dp[N];

}

1. Best time to buy & sell a stock 1

int Solution::maxProfit(const vector<int> &arr) {

int n = arr.size();

if (n <= 1) {

return 0;

}

int diff[n-1];

for (int i=0; i < n-1; i++)

diff[i] = arr[i+1] - arr[i];

int max\_diff = diff[0];

for (int i=1; i<n-1; i++)

{

if (diff[i-1] > 0)

diff[i] += diff[i-1];

if (max\_diff < diff[i])

max\_diff = diff[i];

}

return max\_diff > 0 ? max\_diff:0;

}

1. Maximum path in triangle

int Solution::solve(vector<vector<int> > &A) {

int n=A.size();

vector<vector<int> > dp=A;

for(int i=1;i<n;i++){

for(int j=0;j<n;j++){

if(A[i][j]==0){ break; }

if(j==0){ dp[i][j]+=dp[i-1][j]; }

else{

dp[i][j]+=max(dp[i-1][j],dp[i-1][j-1]);

}

}

}

int ans=0;

for(int i=0;i<n;i++){

ans=max(ans,dp[n-1][i]);

}

return ans;

}

1. Longest Palindromic Subsequence

int Solution::solve(string A) {

    int n=A.length();

    string s=A;

    reverse(A.begin(),A.end());

    int t[n+1][n+1];

    for(int i=0;i<n+1;i++) {

        for(int j=0;j<n+1;j++) {

            if(i==0 || j==0) t[i][j]=0;

        }

    }

    for(int i=1;i<n+1;i++) {

        for(int j=1;j<n+1;j++) {

            if(A[i-1]==s[j-1]) t[i][j]= 1+t[i-1][j-1];

            else t[i][j]=max(t[i-1][j],t[i][j-1]);

        }

    } return t[n][n];

}

1. Longest Increasing Subsequence

int Solution::lis(const vector<int> &a) {

    int n=a.size();

    vector<int> dp(n,0);

    dp[0]=1;

    for(int i=1;i<n;i++) {

        dp[i] = 1;

        for(int j=0;j<i;j++){

            if(a[i]>a[j]) {

                dp[i] = max(dp[i],1+dp[j]);

            }

        }

    }

    return \*max\_element(dp.begin(),dp.end());

}

1. Unique paths in a grid

int Solution::uniquePathsWithObstacles(vector<vector<int> > &A) {

int m = A.size();

assert(A.size()>=0 && A.size()<=100 && A[0].size()>=0 && A[0].size()<=100);

if (m == 0) {

return 0;

}

int n = A[0].size();

if (n == 0) {

return 0;

}

int steps[m][n];

int noSteps = 1, i,j;

for (i = 0; i < n; ++i) {

if (A[0][i] == 1) {

noSteps = 0;

}

steps[0][i] = noSteps;

}

noSteps = 1;

for (i = 0; i < m; ++i) {

if (A[i][0] == 1) {

noSteps = 0;

}

steps[i][0] = noSteps;

}

for (i = 1; i < m; i++)

{

for (j = 1; j < n; j++) {

if (A[i][j] == 1) {

steps[i][j] = 0;

} else {

steps[i][j] = steps[i-1][j] + steps[i][j-1];

}

}

}

return steps[m-1][n-1];

}

1. Palindromic Substrings

int Solution::solve(string A) {

int n = A.size();

assert(1 <= n && n <= 1e3);

for(char &x: A) assert('a' <= x && x <= 'z');

bool dp[n][n];

memset(dp, 0, sizeof(dp));

int ans = 0;

for(int i = n-1; i >= 0; --i){

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

if(i == j) dp[i][j] = 1;

else if(i+1 == j) dp[i][j] = (A[i] == A[j]);

else dp[i][j] = dp[i+1][j-1] & (A[i] == A[j]);

if(dp[i][j]) ++ans;

}

}

return ans;

}

1. Dice Rolls

int Solution::solve(int A) {

assert(1<=A && A<=1e6);

int mod = 1e9+7;

int dp[A+1] = {1};

for(int i = 1; i <= A; ++i){

for(int j = 1; j <= min(i, 6); ++j){

dp[i] += dp[i-j];

dp[i] %= mod;

}

}

return dp[A];

}

1. Minimum Sum Path in Matrix

int Solution::minPathSum(vector<vector<int> > &A) {

int m,n,i,j;

m = A.size();

if (m == 0) {

return 0;

}

n = A[0].size();

if (n == 0) {

return 0;

}

int tc[m][n];

tc[0][0] = A[0][0];

for (i = 1; i < m; ++i) {

tc[i][0] = tc[i-1][0] + A[i][0];

}

for (j = 1; j < n; ++j) {

tc[0][j] = tc[0][j-1] + A[0][j];

}

for (i = 1; i < m; ++i) {

for (j = 1; j < n; ++j) {

tc[i][j] = std::min(tc[i][j-1],tc[i-1][j])+A[i][j];

}

}

return tc[m-1][n-1];

}

1. Paint House

int Solution::solve(vector<vector<int> > &A) {

int N = A.size();

int i = 0;

if(N == 0) return 0;

for(i = 1; i<N; i++){

A[i][0] += min(A[i-1][1], A[i-1][2]); // if 0 chosen in row 1, min of

//1 and 2 had to be chosen earlier. Similarly for all

A[i][1] += min(A[i-1][0], A[i-1][2]);

A[i][2] += min(A[i-1][0], A[i-1][1]);

}

// now find min of the 3 elements in last row

int ans = min(min(A[i-1][0], A[i-1][1]), A[i-1][2]);

return ans;

}

1. Jump Game

int Solution::canJump(vector<int> &A) {

int next = A.size() - 1;

for (int i=A.size()-2; i>=0; --i) {

if (A[i] >= (next - i)) {

next = i;

}

}

return (next == 0) ? 1:0;

}

1. Minimum difference subset

int Solution::solve(vector<int> &A) {

    int n=A.size();

    int s=0;

    for(int i=0;i<n;i++) {

        s+= A[i];

    }

    bool t[n+1][s+1];

    for(int i=0;i<n+1;i++) {

        for(int j=0;j<s+1;j++) {

            if(i==0) t[i][j]=false;

            if(j==0) t[i][j]=true;

        }

    }

    for(int i=1;i<n+1;i++) {

        for(int j=1;j<s+1;j++) {

            if(A[i-1]<=j) t[i][j]=t[i][j-A[i-1]] || t[i-1][j];

            else t[i][j]=t[i-1][j];

        }

    }

    int i=n;

    vector <int> v;

    for(int j=0;j<=s/2;j++) {

        if(t[i][j]==true) v.push\_back(j);

    }

    int m=INT\_MAX;

    for(i=0;i<v.size();i++){

        m=min(m,s-(2\*v[i]));

    }

    return m;

}

1. Scrambled String

unordered\_map <string,bool> mp;

int Solution::isScramble(const string A, const string B) {

    if(A.length()!=B.length()) return false;

    if(A.length()==0 && B.length()==0) return true;

    if(A.compare(B)==0) return true;

    string key=A;

    //key.push\_back(" ");

    key= key + " ";

    key.append(B);

    if(mp.find(key)!=mp.end()) return mp[key];

    else{

    int n=A.length();

    bool flag= false;

    for(int i=1;i<=n-1;i++){

        if(((isScramble(A.substr(0,i),B.substr(n-i,i))==true) && isScramble(A.substr(i,n-i),B.substr(0,n-i))==true) || ((isScramble(A.substr(0,i),B.substr(0,i))==true) && (isScramble(A.substr(i,n-i),B.substr(i,n-i))==true)))

        {flag=true;break;}

    }return mp[key]=flag;}

}

1. Egg drop problem

int static t[101][10001];

int getans(int A,int B){

    if(B==0 || A==1 || B==1) return B;

    if(t[A][B]!=-1) return t[A][B];

    else {

        int mn=INT\_MAX;

        for(int k=1;k<=B;k++) { int low,high;

            if(t[A-1][k-1]!=-1) low=t[A-1][k-1];

        else {low=getans(A-1,k-1);t[A-1][k-1]=low;}

        if(t[A][B-k]!=-1)  high=t[A][B-k];

        else {high=getans(A,B-k);t[A][B-k]=high;}

            int temp=1+max(low,high);

            mn=min(mn,temp);

        }

        return t[A][B]=mn;

    }

}

int Solution::solve(int A, int B) {

    memset(t,-1,sizeof(t));

    return getans(A,B);

}

1. 0-1 Knapsack

int Solution::solve(vector<int> &A, vector<int> &B, int C) {

    int n=A.size();

    int t[n+1][C+1];

    for(int i=0;i<n+1;i++) {

        for(int j=0;j<C+1;j++) {

            if(i==0 || j==0) t[i][j]=0;

        }

    }

    for(int i=1;i<n+1;i++){

        for(int j=1;j<C+1;j++) {

            if(B[i-1]<=j) t[i][j]=max(A[i-1]+t[i-1][j-B[i-1]],t[i-1][j]);

            else t[i][j]= t[i-1][j];

        }

    } return t[n][C];

}

1. Maximum Path Sum in Binary Tree

int dp(TreeNode\* root, int &res){

        if(root == NULL){

            return 0;

        }

        int leftSum = dp(root->left, res);

        int rightSum = dp(root->right, res);

        int pathSum = max(root->val + max(leftSum, rightSum), root->val);

        int totalSum = max(pathSum, root->val + leftSum + rightSum);

        res = max(res, totalSum);

        return pathSum;

 }

int Solution::maxPathSum(TreeNode\* root) {

    int res = INT\_MIN;

        dp(root, res);

        return res;

}

1. Chain of Pairs

int Solution::solve(vector<vector<int> > &A) {

int N = A.size();

vector<int> count(N, 1);

for(int i = 1; i<N; i++){

for(int j = 0; j<i; j++){

if((A[i][0] > A[j][1]) && (count[i] < count[j]+1)) count[i] = count[j]+1;

}

}

return \*max\_element(count.begin(), count.end());

}

1. Min sum Path in Triangle

int Solution::minimumTotal(vector<vector<int> > &A) {

   int n=A.size();

   int arr[n];

   for(int i=0;i<n;++i)arr[i]=A[n-1][i];

   //Go with Bottom up manner

   for(int i=n-2;i>=0;--i){

       for(int j=0;j<A[i].size();++j){

          arr[j]=min(arr[j],arr[j+1])+A[i][j];

       }

   }

   return arr[0]

1. Increasing path in matrix

void help(vector<vector<int>>&A,int i,int j,int n, int m, vector<vector<bool>>&vis)

{

vis[i][j]=true;

if(i<n-1 and vis[i+1][j]==false and A[i+1][j]>A[i][j])

help(A,i+1,j,n,m,vis);

if(j<m-1 and vis[i][j+1]==false and A[i][j+1]>A[i][j])

help(A,i,j+1,n,m,vis);

}

int Solution::solve(vector<vector<int> > &A) {

int n = A.size();

int m = A[0].size();

vector<vector<bool>> vis(n,vector<bool>(m,false));

help(A,0,0,n,m,vis);

if(vis[n-1][m-1])//dope

return n+m-1;

return -1;

}

1. Edit distance

|  |
| --- |
| int min(int x, int y, int z) |
|  | { |
|  | return min(min(x, y), z); |
|  | } |
|  |  |
|  | int editDistDP(string str1, string str2, int m, int n) |
|  | { |
|  |
|  | int dp[m + 1][n + 1]; |
|  |
|  |
|  | for (int i = 0; i <= m; i++) { |
|  | for (int j = 0; j <= n; j++) { |
|  |
|  |
|  | if (i == 0) |
|  | dp[i][j] = j; // Min. operations = j |
|  |  |
|  |
|  |
|  | else if (j == 0) |
|  | dp[i][j] = i; // Min. operations = i |
|  |  |
|  |
|  |
|  | else if (str1[i - 1] == str2[j - 1]) |
|  | dp[i][j] = dp[i - 1][j - 1]; |
|  |  |
|  |
|  |
|  | elsec |
|  | dp[i][j] = 1 + min(dp[i][j - 1], |
|  | dp[i - 1][j], |
|  | dp[i - 1][j - 1]); |
|  | } |
|  | } |
|  |  |
|  | return dp[m][n]; |
|  | } |
|  |
|  |
|  | int main() |
|  | { |
|  |  |
|  | string str1 = "sunday"; |
|  | string str2 = "saturday"; |
|  |  |
|  | cout << editDistDP(str1, str2, str1.length(), str2.length()); |
|  |  |
|  | return 0; |
|  | } |

1. Max Sum without adjacent elements

int Solution::adjacent(vector<vector<int> > &A) {

int i,j,n,it;

n=A[0].size();

vector<int> ans(n+1);

ans[n]=0;

ans[n-1]=max(A[0][n-1],A[1][n-1]);

for(i=n-2;i>=0;i--)

ans[i]=max(ans[i+1],ans[i+2]+max(A[0][i],A[1][i]));

return ans[0];

}

1. Unique BST

/\*\*

\* Definition for binary tree

\* struct TreeNode {

\* int val;

\* TreeNode \*left;

\* TreeNode \*right;

\* TreeNode(int x) : val(x), left(NULL), right(NULL) {}

\* };

\*/

std::vector<TreeNode\*> generate\_trees(int root\_val, int left, int right) {

if (right < left) return std::vector<TreeNode\*>{};

std::vector<TreeNode\*> subroots, left\_subtree, right\_subtree;

for (int i = left; i < root\_val; ++i) {

std::vector<TreeNode\*> temp = generate\_trees(i, left, root\_val - 1);

left\_subtree.insert(left\_subtree.end(), temp.begin(), temp.end());

}

for (int j = root\_val + 1; j <= right; ++j) {

std::vector<TreeNode\*> temp = generate\_trees(j, root\_val + 1, right);

right\_subtree.insert(right\_subtree.end(), temp.begin(), temp.end());

}

if (right\_subtree.size() == 0)

right\_subtree.push\_back(NULL);

if (left\_subtree.size() == 0)

left\_subtree.push\_back(NULL);

for (auto &left\_subroot : left\_subtree) {

for (auto &right\_subroot : right\_subtree) {

TreeNode\* subroot = new TreeNode(root\_val);

subroot->left = left\_subroot;

subroot->right = right\_subroot;

subroots.push\_back(subroot);

}

}

return subroots;

}

vector<TreeNode\*> Solution::generateTrees(int A) {  
 std::vector<TreeNode\*> roots;

for (int i = 1; i <= A; ++i) {

std::vector<TreeNode\*> temp = generate\_trees(i, 1, A);

roots.insert(roots.end(), temp.begin(), temp.end());

}

return roots;

}

1. Shortest Common Superstring

Didn’t get it, Bitmasking

1. Palindrome partitioning 2

 int t[501][501];

bool ispallindrome(string &s,int i,int j){

while(i<=j){

    if(s[i]!=s[j]) return false;

    i++,j--;

}

return true;

}

int solve(string &s,int i,int j){

    if(t[i][j]!=-1) return t[i][j];

if(i>=j) return 0;

if(ispallindrome(s,i,j)) return 0;

int mn=INT\_MAX;

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

    int temp=solve(s,i,k)+solve(s,k+1,j)+1;

    if(temp<mn) mn=temp;

}

t[i][j]=mn;

return t[i][j];

}

int Solution::minCut(string str) {

    int i=0,j=str.length()-1;

        memset(t,-1,sizeof(t));

        return solve(str,i,j);

}

;

1. Longest Valid Parenthesis

int Solution::longestValidParentheses(string A) {

int ans = 0;

stack<int>sk;

sk.push(-1);

for(int i=0;i<A.size();i++)

{

if(A[i]=='(')

sk.push(i);

else

{

if(!sk.empty())

sk.pop();

if(!sk.empty())

ans = max(ans,i-sk.top());

else

sk.push(i);

}

}

return ans;

}

int Solution::longestValidParentheses(string s) {

stack<int> st;

int n=s.size();

int ans=0;

vector<int> dist(n,0);

for(int i=0;i<n;i++){

if(s[i]=='(')

st.push(i);

else if(!st.empty()){

int len=i-st.top()+1;

st.pop();

dist[i]=len;

if(i-len>0)

dist[i]+=dist[i-len];

ans=max(ans,dist[i]);

}

}

return ans;

1. Coins in a line

int rec(vector<int> &a,int i,int j,vector<vector<int> > &m)

{

if(j<i) return 0;

if(i==j) return a[i];

if(i+1==j) return max(a[i],a[j]);

if(m[i][j]!=-1) return m[i][j];

return m[i][j]=max(a[i]+min( rec(a,i+2,j,m) , rec(a,i+1,j-1,m) ),a[j]+min( rec(a,i+1,j-1,m),rec(a,i,j-2,m) ) );

}

int Solution::maxcoin(vector<int> &a) {

vector<vector<int> > m(a.size(),vector<int> (a.size(),-1));

return rec(a,0,a.size()-1,m);

}

1. Length of Longest Subsequence

int Solution::longestSubsequenceLength(const vector<int> &A) {

// aka Longest Bitonic Subsequence(LBS)

// create Longest Inc Subseq(LIS) and Longest Dec Subseq(LDS)

// LBS = max of LIS and LDS - 1

int N = A.size();

if(N == 0) return 0;

vector<int> LIS(N, 1);

vector<int> LDS(N,1);

for(int i = 1; i<N; i++){

for(int j = 0; j<i; j++){

//standard LIS condition

if((A[i] > A[j]) && (LIS[i] <= LIS[j])) LIS[i] = LIS[j] + 1;

}

}

for(int i = N-2; i>=0; i--){

for(int j = N-1; j>i; j--){

//standard LDS condition

if((A[i] > A[j]) && (LDS[i] <= LDS[j])) LDS[i] = LDS[j] + 1;

}

}

int max = LIS[0] + LDS[0] - 1;

for (int i = 1; i < N; i++)

if (LIS[i] + LDS[i] - 1 > max)

max = LIS[i] + LDS[i] - 1;

return max;

}

1. Kingdom War

 vector<vector<int>> dp(m+1 , vector<int>(n+1 , 0));

    int maxe;

    maxe=dp[m-1][n-1]=mat[m-1][n-1];

    for(int i=m-1;i>=0;i--)

    {

        for(int j=n-1;j>=0;j--)

        {

            dp[i][j]= mat[i][j] + dp[i+1][j] + dp[i][j+1] -  dp[i+1][j+1];

             maxe=max(maxe , dp[i][j]);

        }

    }

    return maxe;

}

// dp[i][j+1] -> denotes the grid of values from below part of (i , j+1) -> including itself

    // dp[i+1][j]->denotes the grid of values from below part of (i+1 , j) -> including itself

    // as both contains grid(i+1 , j+1) values , it comes twice so we

    // will substract dp[i+1][j+1];

1. Rod Cutting

Didn’t get it.

1. Evaluate expression to True

unordered\_map<string,int>mp;

int solve(string s,int i,int j, bool isTrue){

if(i>j)

return false;

if(i==j){

if(isTrue==true)

return s[i]=='T';

else

return s[i]=='F';

}

string temp=to\_string(i)+" "+to\_string(j)+" "+to\_string(isTrue);

if(mp.find(temp)!=mp.end())

    return mp[temp];

int ans=0;

for(int k=i+1;k<=j-1;k+=2){

    int lt=solve(s,i,k-1,true);

    int lf=solve(s,i,k-1,false);

    int rt=solve(s,k+1,j,true);

    int rf=solve(s,k+1,j,false);

    if(s[k]=='&'){

        if(isTrue==true)

            ans=ans + lt\*rt;

        else

            ans=ans + lf\*rt + lt\*rf + lf\*rf;

    }

if(s[k]=='|'){

        if(isTrue==true)

            ans= ans + lt\*rt + lt\*rf + lf\*rt;

        else

            ans=ans+ lf\*rf;

    }

    if(s[k]=='^'){

        if(isTrue==true)

            ans=ans+ lf\*rt + lt\*rf;

        else

            ans=ans+ lt\*rt + lf\*rf;

    }

}

ans=ans%1003;

return mp[temp]=ans;

}

int Solution::cnttrue(string A) {

mp.clear();

return solve(A,0,A.length()-1,true);

}

1. Best time to buy and sell a stock 2

int Solution::maxProfit(const vector<int> &A) {

if(A.size()==0)

return 0;

int min=A[0];

int sum=0;

for(int i=1;i<A.size();i++){

if(A[i]<min)

{

min=A[i];

}

if(A[i]>min){

sum=sum+A[i]-min;

min=A[i];

}

}

return sum;

}

1. Max product subarray

int Solution::maxProduct(const vector<int> &A)

{

int maxim=A[0];

int minim=A[0];

int choice1=1,choice2=1;

if(A.size()==0)

return -1;

int ans=0;

if(A.size()==1)

return A[0];

for(int i=1;i<A.size();i++)

{

choice1=A[i]\*minim;

choice2=A[i]\*maxim;

minim=min(A[i],min(choice1,choice2));

maxim=max(A[i],max(choice1,choice2));

ans=max(ans,maxim);

}

return ans;

}

1. Ways to color a 3XN board

Didn’t get it.

1. Coin Sum Infinite

int M=1e6+7;

int Solution::coinchange2(vector<int> &A, int B) {

int n=A.size();

vector<int> dp(B+1);

for(int i=1;i<=n;i++){

dp[0]=1;

for(int j=1;j<=B;j++){

if(A[i-1]>j)continue;

else {

dp[j]=(dp[j-A[i-1]]+dp[j])%M;

}

}

}

return dp[B];

}

(Aditya Verma Coin Change 1 gave MLE)

1. Distinct Subsequences

 int n = A.length(), m = B.length();

    int dp[n+1][m+1];

    memset(dp,0,sizeof(dp));

    for(int i = 0 ; i <= n ; i++)

        dp[i][0]=1;

    for(int i = 1 ; i <= n ; i++)

    {

        for(int j = 1 ; j <= m ; j++)

        {

            if(A[i-1] == B[j-1])

                dp[i][j] = dp[i-1][j-1]+dp[i-1][j];

            else

                dp[i][j] = dp[i-1][j];

        }

    }

    return dp[n][m];

1. Smallest sequence with given primes

vector<int> Solution::solve(int a, int b, int c, int n) {

vector<int>dp(n+1);

dp[0]=1;

int l1=0,l2=0,l3=0;

for(int i=1;i<=n;i++){

dp[i]=min(a\*dp[l1],min(b\*dp[l2],c\*dp[l3]));

if(dp[i]==a\*dp[l1])

l1++;

if(dp[i]==b\*dp[l2])

l2++;

if(dp[i]==c\*dp[l3])

l3++;

}

dp.erase(dp.begin());

return dp;

}

//only have **A, B, C** or a combination of them as their prime factors.

1. K-th Manhattan distance neighbourhood

vector<int> x={1,-1,0,0};

vector<int> y={0,0,1,-1};

int arch(int i,int j,int k,vector<vector<int> > &B,vector<vector<vector<int>>> &dp){

if(i>=B.size() || j>=B[0].size() || i<0 || j<0) return 0;

if(k==0) return B[i][j];

if(dp[i][j][k]!=-1) return dp[i][j][k];

int ans1=B[i][j];

for(int ii=0;ii<4;ii++){

ans1=max(ans1,arch(i+x[ii],j+y[ii],k-1,B,dp));

}

return dp[i][j][k]=ans1;

}

vector<vector<int> > Solution::solve(int A, vector<vector<int> > &B) {

int n=B.size();

int m=B[0].size();

vector<vector<int>> ans(n,vector<int> (m,0));

vector<vector<vector<int>>> dp(n,vector<vector<int>> (m,vector<int> (A+1,-1)));

for(int i=0;i<n;i++){

for(int j=0;j<m;j++){

ans[i][j]= arch(i,j,A,B,dp);

}

}

return ans;

}

1. Word break

bool wordBreak(string s, vector < string > & wordDict) {

set < string > word\_set(wordDict.begin(), wordDict.end());

vector < bool > dp(s.length() + 1);

dp[0] = true;

for (int i = 1; i <= s.length(); i++) {

for (int j = 0; j < i; j++) {

if (dp[j] and word\_set.find(s.substr(j, i - j)) != word\_set.end()) {

dp[i] = true;

break;

}

}

}

return dp[s.length()];

}

1. Maximum size square sub matrix

int Solution::solve(vector<vector<int> > &A) {

int n=A.size();

int m=A[0].size();

int dp[n][m];

memset(dp, -1, sizeof dp);

for(int i=0; i<n ; i++){

for(int j=0; j<m ; j++){

if(A[i][j]==0) dp[i][j]=A[i][j];

if(i==n-1) dp[i][j]=A[i][j];

if(j==m-1) dp[i][j]=A[i][j];

}

}

for(int i=n-2; i>=0 ; i--){

for(int j=m-2; j>=0; j--){

if(dp[i][j]==-1)

dp[i][j]=min(dp[i+1][j], min(dp[i][j+1], dp[i+1][j+1]))+1;

}

}

int maxi=0;

for(int i=0; i<n ; i++){

for(int j=0; j<m ; j++){

maxi= max(maxi,dp[i][j]);

}

}return maxi\*maxi;

}

1. Tiling with dominoes left
2. Queen Attack

vector<string> s;

int n,m;

bool isvalid(int x,int y){

    if(x<0 || y<0 || x>=n || y>=m)return false;

    return true;

}

int checkup(int i,int j){

    if(isvalid(i,j) ){

        if(s[i][j]=='1')return 1;

        else return checkup(i-1,j);

    }

    return 0;

}

int checkdown(int i,int j){

    if(isvalid(i,j) ){

        if(s[i][j]=='1')return 1;

        else return checkdown(i+1,j);

    }

    return 0;

}

int checkleft(int i,int j){

    if(isvalid(i,j) ){

        if(s[i][j]=='1')return 1;

        else return checkleft(i,j-1);

    }

    return 0;

}

int checkright(int i,int j){

    if(isvalid(i,j) ){

        if(s[i][j]=='1')return 1;

        else return checkright(i,j+1);

    }

    return 0;

}

int checkdup(int i,int j){

    if(isvalid(i,j) ){

        if(s[i][j]=='1')return 1;

        else return checkdup(i+1,j+1);

    }

    return 0;

}

int checkddown(int i,int j){

    if(isvalid(i,j) ){

        if(s[i][j]=='1')return 1;

        else return checkddown(i-1,j-1);

    }

    return 0;

}

int checkrdup(int i,int j){

    if(isvalid(i,j) ){

        if(s[i][j]=='1')return 1;

        else return checkrdup(i-1,j+1);

    }

    return 0;

}

int checkrddown(int i,int j){

    if(isvalid(i,j) ){

        if(s[i][j]=='1')return 1;

        else return checkrddown(i+1,j-1);

    }

    return 0;

}

vector<vector<int> > Solution::queenAttack(vector<string> &A) {

 n=A.size();

 m=A[0].length();

    s=A;

vector<vector<int>> mat(n,vector<int> (m,0));

for(int i=0;i<n;i++){

    for(int j=0;j<m;j++){

        int sum=0;

        sum+=checkup(i-1,j);

        sum+=checkdown(i+1,j);

        sum+=checkleft(i,j-1);

        sum+=checkright(i,j+1);

        sum+=checkdup(i+1,j+1);

        sum+=checkddown(i-1,j-1);

        sum+=checkrdup(i-1,j+1);

        sum+=checkrddown(i+1,j-1);

   mat[i][j]=sum;

    }

}

    return mat;

}

1. Double increasing series

vector<vector<int>> dp(25,vector<int>(100005,-1));

const int mod=1e9+7;

int fun(int i, int j)

{

if(i==1) return j;

if(dp[i][j]!=-1) return dp[i][j];

int z=0;

z=(z+fun(i-1,j/2))%mod;

if(j-1>0) z=(z+fun(i,j-1))%mod;

return dp[i][j]=z%mod;

}

int Solution::solve(int a, int b){

return fun(b,a);

}

1. Intersecting chords in a circle

int Solution::chordCnt(int n) {

    long long int dp[n+1];

    for(int i=0;i<n+1;i++){

        dp[i] = 0;

    }

int mod = 1e9+7;

    dp[0] = 1;

    dp[1] = 1;

    dp[2] = 2;

    for(int i=3;i<n+1;i++){

        for(int j=0;j<i;j++){

            dp[i]=(dp[i]+dp[i-j-1]\*dp[j])%mod;

        }

    }

    return dp[n]%mod;

}

//didn’t understand the recurrence relation

1. Dungeon princess

int rec(int row,int col,vector<vector<int>> &dp,vector<vector<int> > &A){

     int m=A.size();

    int n=A[0].size();

    if(row>=m || col>=n){

        return INT\_MIN;

    }

    int ans=0;

    if(row==m-1 && col==n-1){

        return min(0,ans+A[row][col]);

    }

    if(dp[row][col]!=-1){

        return dp[row][col];

    }

    ans=min(0,A[row][col]+max(rec(row+1,col,dp,A),rec(row,col+1,dp,A)));

    return dp[row][col]=ans;

}

int Solution::calculateMinimumHP(vector<vector<int> > &A) {

    int m=A.size();

    int n=A[0].size();

    vector<vector<int>> dp(m+1,vector<int>(n+1,-1));

    int ans=rec(0,0,dp,A);

    return abs(ans)+1;

}

1. Ways to decode

const int mod = 1000000007;

const int maxn = 100009;

int dp[maxn];

int solve(int index, string & A) {

if (index >= A.size())

return 1;

if (dp[index] != -1)

return dp[index];

if (A[index] == '0')

return dp[index] = 0;

int ans = solve(index + 1, A);

ans %= mod;

if (index + 1 < A.size()) {

int num = ((A[index] - '0') \* 10) + (A[index + 1] - '0');

if (num >= 10 && num <= 26)

ans += solve(index + 2, A);

ans %= mod;

}

return dp[index] = ans;

ans %= mod;

}

int Solution::numDecodings(string A) {

memset(dp, -1, sizeof(dp));

return solve(0, A);

}

1. Min jumps array

int Solution::jump(vector<int> &arr) {

int n=arr.size(), mx=0;

if(n==1)

return 0;

if(arr[0]==0)

return -1;

int mx\_reach=arr[0], step=arr[0], jump=1;

for(int i=1; i<n; ++i)

{

if(i==n-1)

return jump;

mx\_reach=max(mx\_reach, i+arr[i]);

step--;

if(step==0)

{

jump++;

if(i>=mx\_reach)

return -1;

step=mx\_reach-i;

}

}

return -1;

}

1. Tushar’s Birthday party

int Solution::solve(const vector<int> &A, const vector<int> &B, const vector<int> &C) {

int maxi = INT\_MIN;

for(int i = 0; i<A.size(); i++)

if(A[i]>maxi)

maxi = A[i];

int n = B.size();

int dp[maxi+1];

for(int i = 1; i<=maxi; i++)

dp[i] = INT\_MAX;

dp[0] = 0;

for(int i = 0; i<n; i++)

for(int j = B[i]; j<=maxi; j++)

dp[j] = min(dp[j], C[i]+dp[j-B[i]]);

int ans = 0;

for(int i = 0; i<A.size(); i++)

ans+=dp[A[i]];

return ans;

}

1. Longest Arithmetic Progression

int Solution::solve(const vector<int> &A) {

int n = A.size();

assert(n>=1 && n<=1000);

for(int a:A)

assert(a>=1 && a<=1000000000);

if (n < 3) return n;

vector<vector<int> > dp(n, vector<int>(n, -1));

map<int, int> pos;

for (int i = 0; i < n; i++) {

for (int j = i + 1; j < n; j++) {

dp[i][j] = 2;

int dif = A[j] - A[i];

int need = 2 \* A[i] - A[j];

if (pos.count(need) == 0) continue;

dp[i][j] = max(dp[i][j], dp[pos[need]][i] + 1);

}

pos[A[i]] = i;

}

int ans = 2;

for (int i = 0; i < n; i++) {

for (int j = i + 1; j < n; j++) {

ans = max(ans, dp[i][j]);

}

}

return ans;

}

1. N-digit numbers with sum s

int N = 1e9+7;

long long f(int n, int sum, vector<vector<long long>> &dp){

    if(n == 0 and sum == 0) return 1;

    if(n == 0 or sum == 0) return 0;

    if(9\*n < sum) return 0;

    if(dp[n][sum] != -1) return dp[n][sum] % N;

    int res = 0;

    for(int i=0; i<=9; i++){

        if(n == 1 and i == 0) continue;

        if(sum >= i){

            if(f(n-1, sum-i, dp)){

                res = (res + f(n-1, sum-i, dp)) % N;

            }

        }

        else break;

    }

    return dp[n][sum] = res % N;

}

int Solution::solve(int n, int sum) {

    if(9\*n < sum) return 0;

    vector<vector<long long>> dp(n+1, vector<long long>(sum+1, -1));

    int ans = f(n, sum, dp);

    return ans;

}

1. Sub Matrices with sum zero

int subarray\_sum\_k(vector<int> &nums , int target)

{

    int n=nums.size();

    unordered\_map<int , int> mp;

    int count=0;

    mp[0]=1;

    long long sum=0;

    for(int i=0;i<n;i++)

    {

        sum+=nums[i];

        if(mp.find( sum - target)!=mp.end())

        {

            count+=(mp[sum-target]);

        }

        mp[sum]++;

    }

    return count;

}

int Solution::solve(vector<vector<int> > &mat) {

    // This question is similar to , extension of  finding the no. of subarrays

    // having sum = target

    // find the all possible sub matrices using the sum from i - to n

    // from  j=i , to n;

    int  m=mat.size();

    if(m==0)

    {

        return 0;

    }

    int n=mat[0].size();

    long long res=0;

    for(int i=0;i<m;i++)

    {

         vector<int> nums(n , 0 );

        for(int j=i;j<m;j++)

        {

            for(int k=0;k<n;k++)

            {

                nums[k]+=(mat[j][k]);

            }

            int ans = subarray\_sum\_k(nums , 0);

            res+=ans;

        }

    }

    return res;

}

1. Largest area of rectangle with permutations

The solution is similar to the regular maximum area in the matrix problem with one minor change. As we can permute among columns, just sort the updated columns in descending order and calculate the maximum area we can get using them.

int Solution::solve(vector<vector<int> > &A) {

    int n = A.size(), m = A[0].size(), res = 0;

    vector<int> curr(m);

    for(int i = 0; i<m; i++) curr.push\_back(A[0][i]);

    sort(curr.begin(), curr.end(), greater<int>());

    for(int i = 0; i<m; i++) {

        res = max(res, curr[i]*(i+1));*

*}*

*for(int i = 1; i<n; i++) {*

*for(int j = 0; j<m; j++) {*

*A[i][j] = (A[i][j] == 0) ? 0 : A[i-1][j] + 1;*

*curr[j] = A[i][j];*

*}*

*sort(curr.begin(), curr.end(), greater<int>());*

*for(int j = 0; j<m; j++) {*

*res = max(res, curr[j]*(j+1));

        }

    }

    return res;

}

1. Equal Average Partition

Didn’t get it, based on subset sum, see yt

1. Flip Array

int Solution::solve(const vector<int> &A) {

int sum=0;

for(int i=0;i<A.size();i++) sum+=A[i];

sum/=2;

vector<int> dp(sum+1, INT\_MAX);

dp[0]=0;

for(int i=0;i<A.size();i++)

{

for(int j=sum;j>=A[i];j--)

{

if(dp[j-A[i]]!=INT\_MAX) dp[j]=min(dp[j], dp[j-A[i]]+1);

}

}

for(int i=sum;i>=0 ;i--)

{

if(dp[i]!=INT\_MAX) return dp[i];

}

}

1. Max rectangle in binary array

Didn’t get it.

1. Regular Expression Match

int Solution::isMatch(const string a, const string b) {

    int as=a.size(),bs=b.size();

    vector<bool> prev(bs+1,false),curr(bs+1,false);

    prev[0]=true;

    for(int j=1;j<=bs;j++)

        if(b[j-1]=='\*')

            prev[j]=prev[j-1];

    for(int i=1;i<=as;i++){

        for(int j=1;j<=bs;j++){

            if(a[i-1]==b[j-1] or b[j-1]=='?')

                curr[j]=prev[j-1];

            else if(b[j-1]=='\*')

                curr[j]=curr[j-1] or prev[j];

            else

                curr[j]=false;

        }

        prev=curr;

    }

    return curr[bs];

}

1. Interleaving Strings

int Solution::isInterleave(string A, string B, string C) {

int n=A.size();int m=B.size();

if(A.size()+B.size()!=C.size()) return 0;

int dp[n+1][m+1];

memset(dp,0,sizeof(dp));

dp[0][0]=1;

for(int i=0;i<=n;i++){

for(int j=0;j<=m;j++){

if(i>0 && A[i-1]==C[i+j-1]) dp[i][j]=max(dp[i][j],dp[i-1][j]);

if(j>0 && B[j-1]==C[i+j-1]) dp[i][j]=max(dp[i][j],dp[i][j-1]);

}

}

return dp[n][m];

}

OR

int Solution::isInterleave(string A, string B, string C) {

    int n=0,m=0;

    for(int i=0;i<C.length();i++){

        if(C[i]==A[n] && C[i]==B[m]) {

            if(C[i+1]==A[n+1]) n++;

            else m++;

        }

        else if(C[i]==A[n]) n++;

        else if(C[i]==B[m]) m++;

        else return 0;

    } return 1;

}

1. Max Edge Queries

Didn’t get it.

1. Merge Elements

int dp[201][201];

int minCost(int i,int j,vector<int>&A){

    if(i==j) return 0;

    if(dp[i][j]!=-1)

      return dp[i][j];

   int tot = 0;

   int x = INT\_MAX;

   for(int k =i;k<=j;k++ )

     tot+= A[k];

  for(int k =i+1;k<=j;k++){

      x = min(x,tot+minCost(i,k-1,A)+minCost(k,j,A));

  }

  return dp[i][j] = x;

}

int Solution::solve(vector<int> &A) {

    int n = A.size();

    memset(dp,-1,sizeof(dp));

 return minCost(0,n-1,A);

}

1. Dice Throw

int Solution::findDiceSum(int A, int B, int C) {

long dp[A + 1][C + 1];

long mod = (long) 1e9 + 7;

for (int i = 0; i <= A; i++)

for (int j = 0; j <= C; j++)

dp[i][j] = 0;

dp[0][0] = 1;

for (int i = 1; i <= A; i++) {

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

dp[i][j] = (dp[i][j - 1] + dp[i - 1][j - 1]) % mod;

if (j - B - 1 >= 0)

dp[i][j] = (dp[i][j] - dp[i - 1][j - B - 1] + mod) % mod;

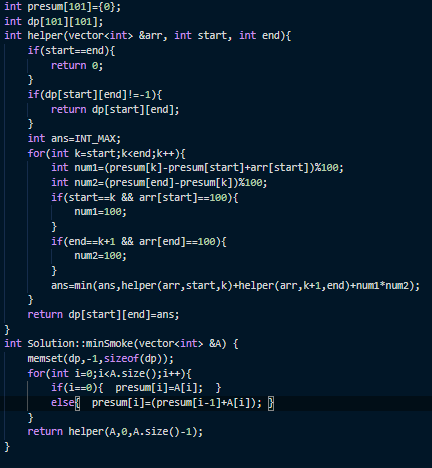
}

}

return (int) dp[A][C];

}

1. Potions



1. Collect resources

int r,c,a[1005][1005],b[1005][1005];

long long dp[1005],mx[1005];

int Solution::solve(vector<vector<int> > &A, vector<vector<int> > &B) {

long long ans=0,s=0;

r=A.size();

c=A[0].size();

for(int i=0;i<r;i++)

{

for(int j=0;j<c;j++)

{

a[i][j]=A[i][j];

ans+=a[i][j];

}

}

for(int i=0;i<r;i++)

{

for(int j=0;j<c;j++)

{

b[i][j]=B[i][j];

b[i][j]-=a[i][j];

}

}

dp[0]=0;

mx[0]=0;

for(int i=1;i<=c;i++)

{

s+=b[0][i-1];

dp[i]=s;

mx[i]=max(mx[i-1],dp[i]);

}

for(int i=1;i<r;i++)

{

dp[0]=mx[0];

s=0;

for(int j=1;j<=c;j++)

{

s+=b[i][j-1];

dp[j]=s+mx[j];

}

for(int j=1;j<=c;j++)

mx[j]=max(mx[j-1],dp[j]);

}

return (ans+mx[c])%1000000007;

}