DP editorial

Longest common subsequence

```
int Solution::solve(string A, string B) {
    int n=A.size();
    int m=B.size();
    vector<vector<int>> dp(n+1,vector<int>(m+1,0));
    for(int i=1;i<=n;i++){
        for(int j=1;j<=m;j++){
            if(A[i-1]==B[j-1]){
                dp[i][j]=max(dp[i][j],dp[i-1][j-1]+1);
            }
            dp[i][j]=max({dp[i][j],dp[i-1][j],dp[i][j-1]});
        }
    }
    return dp[n][m];
}
```

Longest Palindromic Subsequence

Reverse one string and find longest common subsequence between it and original string

```
int Solution::solve(string a) {
    int n=a.size();
    string b=a;
    reverse(b.begin(),b.end());
    //cout<<b<<endl;
    vector<vector<int>> dp(n+1,vector<int>(n+1,0));
    for(int i=1;i<=n;i++){
        for(int j=1;j<=n;j++){
            if(a[i-1]==b[j-1]){
                dp[i][j]=max(dp[i][j],dp[i-1][j-1]+1);
            }
            dp[i][j]=max({dp[i][j],dp[i-1][j],dp[i][j-1]});
            //cout<<dp[i][j]<<endl;
        }
    }
    return dp[n][n];

}
```

method 2

```
int Solution::solve(string a) {
    int n=a.size();
    string b=a;
    reverse(b.begin(),b.end());
    //cout<<b<<endl;
    vector<vector<int>> dp(n+1,vector<int>(n+1,0));
    for(int i=0;i<n;i++){
        dp[i][i]=1;
    }
    int ans=1;
    for(int l=1;l<n;l++){
        for(int i=0;i+l<n;i++){
            int r=i+l;
            if(l==1){
                if(a[i]==a[r]){
                    dp[i][r]=2;

                }
                else{
                    dp[i][r]=1;

                }
                ans=max(ans,dp[i][r]);
                continue;
            }
            if(a[i]==a[r]){
                dp[i][r]=dp[i+1][r-1]+2;
            }
            dp[i][r]=max({dp[i][r],dp[i+1][r],dp[i][r-1]});
            ans=max(ans,dp[i][r]);
        }
    }
    return ans;

}
```

Knapsack Type 1

```


ll dp[101][100001];//max sum with till prefix i with j weight

int main()
{
ios_base::sync_with_stdio(0);
cin.tie(0);
    ll n,w;
    cin>>n>>w;
    ll w1[n+1];

    ll a[n+1];
    for(int i=1;i<=n;i++){
        cin>>w1[i]>>a[i];
    }



    for(int i=1;i<=n;i++){
        for(int j=1;j<=w;j++){
           dp[i][j]=dp[i-1][j];
                if(j-w1[i]>=0){

                    dp[i][j]=max(dp[i-1][j-w1[i]]+a[i],dp[i][j]);
                    //cout<<dp[i][j]<<endl;




        }
    }
    }
ll ans=0;
    for(int i=0;i<=w;i++){
        ans=max(dp[n][i],ans);
    }
    cout<<ans<<endl;


    return 0;
}
```

Knapsack type2

```
ll dp[101][100001];




int main()
{
ios_base::sync_with_stdio(0);
cin.tie(0);
    ll n,w;
    cin>>n>>w;
    ll w1[n+1];

    ll a[n+1];

    //cout<<dp[0][0]<<endl;
    for(int i=1;i<=n;i++){
        cin>>w1[i]>>a[i];
    }
     //min weight to reach this value form 1...i items
    for(int i=0;i<=n;i++){
        for(int j=0;j<=100000;j++){
            dp[i][j]=mod;
            if(j==0){
                dp[i][j]=0;
            }
        }
    }
    //cout<<dp[0][0]<<endl;


    for(int i=1;i<=n;i++){
        for(ll j=1;j<=100000;j++){
            dp[i][j]=dp[i-1][j];
            if(j-a[i]>=0){
                dp[i][j]=min(dp[i][j],w1[i]+dp[i-1][j-a[i]]);
            }
        }




    }

    int ans=0;
    for(int i=1;i<=n;i++){
        for(int j=0;j<=100000;j++){

                if(dp[i][j]<=w){
                    ans=max(ans,j);
                }

        }
    }
    cout<<ans<<endl;




    return 0;
}
```

Minimum edit distance

```


int Solution::minDistance(string A, string B) {
    int n=A.size();
    int m=B.size();
    long long dp[n+1][m+1];
    for(int i=0;i<=n;i++){
        for(int j=0;j<=m;j++){
            dp[i][j]=1e9;
        }
    }
    dp[0][0]=0;
    dp[1][0]=1;
    dp[0][1]=1;
    for(int i=1;i<=n;i++){
        dp[i][0]=i;
    }
    for(int i=1;i<=m;i++){
        dp[0][i]=i;
    }
    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];
            }
            else{
                 dp[i][j]=dp[i-1][j-1]+1;
            }
            dp[i][j]=min({dp[i][j],dp[i-1][j]+1,dp[i][j-1]+1});
        }
    }
    return dp[n][m];
```

Scramble String

```
int  dp[101][101][101];
int A[26];
string s1,s2;
int solve2(int i,int j,int s){
    if(dp[i][j][s]!=-1){
        return dp[i][j][s];
    }
    if(s==1){
        if(s1[i]==s2[j]){
            return dp[i][j][s]=1;
        }
        else{
            return dp[i][j][s]=0;
        }
    }
    string a=s1.substr(i,s);
    string b=s2.substr(j,s);
    sort(a.begin(),a.end());
    sort(b.begin(),b.end());
    if(a!=b){
            dp[i][j][s]=0;
            return dp[i][j][s];
        }
        for(int k=1;k<s;k++){
            if(solve2(i,j,k)&&solve2(i+k,j+k,s-k))return dp[i][j][s]=1;
            if(solve2(i,j+s-k,k)&&solve2(i+k,j,s-k))return dp[i][j][s]=1;
        }
        return dp[i][j][s]=0;






}
//string a,b;

int solve(string a,string b){
    if(a.size()!=b.size())return 0;
    string a1=a;
    string b1=b;
    int n=a.size();
    if(n==0)return 1;
    if(a==b)return 1;
    sort(a1.begin(),a1.end());
    sort(b1.begin(),b1.end());
    if(a1!=b1)return 0;
    for(int i=1;i<n;i++){
        // string s1=a.substr(0,i);
        // string s2=b.substr(0,i);
        // string s3=a.substr(i,n-i);
        // string s4=b.substr(i,n-i);
        if(solve(a.substr(0,i),b.substr(0,i))&&solve(a.substr(i,n-i),b.substr(i,n-
i))){
            return 1;
        }

        if(solve(a.substr(0,i),b.substr(n-i,i))&& solve(a.substr(i,n-
i),b.substr(0,n-i))){
            return 1;
        }



    }
    return 0;






}
int Solution::isScramble(const string A, const string B) {
    s1=A;
```

```
     s2=B;
     for(int i=0;i<51;i++){
         for(int j=0;j<51;j++){
             for(int k=0;k<51;k++){
                 dp[i][j][k]=-1;
             }
         }
     }
     if(A.size()!=B.size())return 0;
     return solve2(0,0,A.size());
    ///return solve(A,B);




}
```

Regular Expression Match

Implement wildcard pattern matching with support for '?' and '*' for strings A and B.

'?' : Matches any single character. '*' : Matches any sequence of characters (including the empty sequence). The matching should cover the entire input string (not partial).

```
int Solution::isMatch(const string A, const string B) {
   int n=A.size();
   int m=B.size();
   int c=0;
   for(int j=0;j<m;j++){
      if(B[j]!='*')c++;
   }
   if(c>n)return 0;
   if(c==0 && m>0)return 1;
   int dp[n+1][m+1];
   for(int i=0;i<n;i++){
      for(int j=0;j<m;j++)dp[i][j]=0;
   }
   dp[0][0]=1;
   for(int j=1;j<=m;j++){
      if(B[j-1]=='*'){
         dp[0][j]=dp[0][j-1];
      }
   }
   for(int i=1;i<=n;i++){
      for(int j=1;j<=m;j++){
         if(B[j-1]=='*'){
            dp[i][j]=dp[i][j-1]|dp[i-1][j];
         }
         else if(B[j-1]=='?'|| A[i-1]==B[j-1]){
            dp[i][j]=dp[i-1][j-1];
         }
         else{
            dp[i][j]=0;
         }
         //cout<<dp[i][j]<<endl;
      }
   }
   return dp[n][m];

}
```

Regular Expression II

Implement regular expression matching with support for '.' and '*'.

'.' Matches any single character.

'*' Matches zero or more of the preceding element.

The matching should cover the entire input string (not partial).

```

int Solution::isMatch(const string A, const string B) {
  string s=A;
  string p=B;
  int n=p.size();
  int m=s.size();
  s="0"+s;
  p="0"+p;

  int dp[n+1][m+1];
  memset(dp,0,sizeof(dp));
  dp[0][0]=1;

  for(int i=1;i<=n;i++){
      for(int j=0;j<=m;j++){
          if(p[i]==s[j]||p[i]=='.'){
              if(j>0)    dp[i][j]|=dp[i-1][j-1];
          }else if(p[i]=='*'){
              dp[i][j]|=dp[i-2][j];   //use 0 times
              dp[i][j]|=dp[i-1][j];   //use 1 times
              if(s[j]==p[i-1]||p[i-1]=='.')   if(j>0)    dp[i][j]|=dp[i][j-1];
//use multiple times
          }
      }
  }

  return dp[n][m];
}
```

Length of Longest Subsequence

Given an 1D integer array A of length N, find the length of longest subsequence which is first increasing then decreasing.

```
int calc(vector<int> A,int n,vector<int> &dp){
    dp.resize(n);
    dp[0]=1;
    int m[n];
    m[1]=A[0];
    int len=1;
    for(int i=1;i<n;i++){
        if(A[i]>m[len]){
            len++;
            dp[i]=len;
            m[len]=A[i];

        }
        else{
            int p=upper_bound(m+1,m+len+1,A[i])-m;
            if(m[p-1]==A[i]){
                p--;
            }
            dp[i]=p;
            m[p]=A[i];
        }
    }

    return 1;
}
int calc2(vector<int> A,int n,vector<int> &dp){
    dp.resize(n);
    dp[n-1]=1;
    for(int i=n-2;i>=0;i--){
        dp[i]=1;
        for(int j=i+1;j<n;j++){
            //dp[i]=1;
            if(A[i]>A[j]){
                dp[i]=max(dp[i],dp[j]+1);
            }
        }
    }
    return 1;
}

int Solution::longestSubsequenceLength(const vector<int> &A) {
    int n=A.size();
    if(n==0)return 0;
    //if(n==0)return 0;
    vector<int> d1;
    vector<int> d2;
    calc(A,n,d1);
    //reverse(A.begin(),A.end());
    calc2(A,n,d2);
    int ans=0;
    for(int i=0;i<n;i++){
        ans=max(ans,d1[i]+d2[i]-1);

    }
    //if(ans==1)return 0;
    return ans;
}
```

Smallest sequence with given Primes

```


#define pb push_back
int a,b,c,d;
vector<int> Solution::solve(int A, int B, int C, int D) {
    a=A;
    b=B;
    c=C;
    d=D;
    vector<int> v;
    set<int> s;
    s.insert(A);
    s.insert(B);
    s.insert(C);
    while(v.size()!=D){
        auto x=s.begin();
        int p=*x;
        s.erase(x);
        s.insert(p*A);
        s.insert(p*B);
        s.insert(p*C);
        v.pb(p);
    }
    return v;

}
```

Tiling With Dominoes

Given an integer A you have to find the number of ways to fill a 3 x A board with 2 x 1 dominoes.

Return the answer modulo 109 + 7 .

```
int Solution::solve(int A) {
     int n=A;
     int mod=1000000007;
    int a[n+1],b[n+1];
    a[0]=1;
    a[1]=0;
    b[0]=0;
    b[1]=1;
    for(int i=2;i<=n;i++){
        a[i]=a[i-2]%mod+2*b[i-1]%mod;
        b[i]=a[i-1]%mod+b[i-2]%mod;
        a[i]%=mod;
        b[i]%=mod;


    }
    return a[n]%mod;

}
```

Ways to Decode

A message containing letters from A-Z is being encoded to numbers using the following mapping:

'A' -> 1 'B' -> 2 ... 'Z' -> 26 Given an encoded message A containing digits, determine the total number of ways to decode it modulo 109 + 7.

```
int Solution::numDecodings(string A) {
     int n=A.size();
     if(n==1){
         if(A[0]=='0')return 0;
         return 1;
     }
     int mod=1000000007;
     long long dp[n+1];
     memset(dp,0,sizeof(dp));
     if(A[0]=='0'&& A[1]=='0'){
         dp[0]=0;
     }
     else{
     dp[0]=1;
     }
     if(A[0]!='0')
```

```
    dp[1]=1;
    else
    dp[1]=0;
    for(int i=2;i<=n;i++){

        //cout<<p<<endl;
        if(A[i-2]=='1'|| (A[i-2]=='2'&& A[i-1]<'7')){
            dp[i]+=(dp[i-2])%mod;
            dp[i]%=mod;
        }
        if(A[i-1]!='0'){
        dp[i]+=dp[i-1]%mod;
        //cout<<dp[i]<<endl;
        dp[i]%=mod;
        }


    }
    return dp[n];


}
```