**Dynmaic Programming For Practice**

Sharing some topic wise good Dynamic Programming problems and sample solutions to observe on how to approach.

1.**Unbounded Knapsack** **or Target sum**  
Identify if problems talks about finding groups or subset which is equal to given target.

<https://leetcode.com/problems/target-sum/>  
<https://leetcode.com/problems/partition-equal-subset-sum/>  
<https://leetcode.com/problems/last-stone-weight-ii/>  
<https://leetcode.com/problems/shortest-subarray-with-sum-at-least-k/>

All the above problems can be solved by 01 Knapsack or Target sum algo with minor tweaks.  
Below is a standard code for 01 knapsack or target sum problems.

int 01 knacsack(vector<int>& nums,vector<int>& v, int w) // nums array , w total amount that have to collect

{ // v value array

int n=nums.size();

vector<vector<bool>> d(n+1,vector<bool>(w+1,0));

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

{

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

{

if(j<nums[i-1])

{

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

}

else if(nums[i-1]<=j)

{

d[i][j]=max(v[i-1]+d[i-1][j-nums[i-1]],d[i-1][j]);

}

}

}

return d[n][w];

}

**Funtion for Target sum**

int countsubset(vector<int>& nums, int w)

{

int n=nums.size();

vector<vector<bool>> d(n+1,vector<bool>(w+1));

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

{

d[i][0]=1;

}

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

{

d[0][i]=0;

}

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

{

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

{

if(j<nums[i-1])

{

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

}

else if(nums[i-1]<=j)

{

d[i][j]=d[i-1][j-nums[i-1]] + d[i-1][j];

}

}

}

return d[n][w];

}

2.**Unbounded Knapsack**  
Identify if problems talks about finding groups or subset which is equal to given target and repetition is allowed.

<https://leetcode.com/problems/coin-change-2/>  
<https://leetcode.com/problems/coin-change/>

All the above problems can be solved by unbounded Knapsack algo with minor tweaks.  
Below is a standard code for 01 knapsack or target sum problems.

int unboundedknacsack(vector<int>& nums,vector<int>& v, int w)

{

int n=nums.size();

vector<vector<bool>> d(n+1,vector<bool>(w+1,0));

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

{

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

{

if(j<nums[i-1])

{

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

}

else if(nums[i-1]<=j)

{

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

}

}

}

return d[n][w];

}

**or**

int change(int amount, vector<int>& coins)

{

vector<vector<int>> d(coins.size()+1,vector<int>(amount+1));

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

{

d[i][0]=1;

}

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

{

d[0][i]=0;

}

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

{

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

{

if(j<coins[i-1])

{

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

}

else if(j>=coins[i-1])

{

d[i][j]=(d[i][j-coins[i-1]]+d[i-1][j]);

}

}

}

return d[coins.size()][amount];

}

3.**Longest Increasing Subsequence (LIS)**

Identify if problems talks about finding longest increasing subset.

<https://leetcode.com/problems/minimum-cost-to-cut-a-stick/>  
<https://leetcode.com/problems/longest-increasing-subsequence/>  
<https://leetcode.com/problems/largest-divisible-subset/>  
<https://leetcode.com/problems/perfect-squares/>  
<https://leetcode.com/problems/super-ugly-number/>

<https://leetcode.com/problems/russian-doll-envelopes/>  
<https://leetcode.com/problems/maximum-height-by-stacking-cuboids/description/>

@Nam\_22 mentioning above two question .

All the above problems can be solved by longest Increasing subsequence algo with minor tweaks.  
Below is a standard code for LIS problems.

int lengthOfLIS(vector<int>& nums)

{

vector<int> d(nums.size(),1);

int m=0;

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

{

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

{

if(nums[j]<nums[i] && d[i]<d[j]+1)

{

d[i]=d[j]+1;

}

}

m=max(d[i],m);

}

return m;

}

**longest bitonic subsequence**

int lbs(vector<int> v)

{

vector<int> lis(v.size(),1);

vector<int> lds(v.size(),1);

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

{

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

{

if(v[j]<v[i] && lis[i]<lis[j]+1)

{

lis[i]=lis[j]+1;

}

}

}

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

{

for(int j=v.size()-1;j>i;j--)

{

if(v[j]<v[i] && lds[i]<lds[j]+1)

{

lds[i]=lds[j]+1;

}

}

}

int m=0;

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

{

m=max(m,lis[i]+lds[i]-1);

}

return m;

}

4.**Longest Common Subsequence**

Identify if problems talks about finding longest common subset.

1.**subsequence**  
<https://leetcode.com/problems/longest-common-subsequence/>  
<https://leetcode.com/problems/distinct-subsequences/>  
<https://leetcode.com/problems/shortest-common-supersequence/>  
<https://leetcode.com/problems/distinct-subsequences/>  
<https://leetcode.com/problems/interleaving-string/>

int longestCommonSubsequence(string text1, string text2)

{

int n1 = text1.size();

int n2 = text2.size();

vector<vector<int>> dp(n1+1,vector<int>(n2+1,0));

for(int i=1;i<=n1;i++)

{

for(int j=1;j<=n2;j++)

{

if(text1[i-1] == text2[j-1])

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

else

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

}

}

return dp[n1][n2];

}

2.**substring**

<https://leetcode.com/problems/maximum-length-of-repeated-subarray/>

int longestCommonSubstring(string text1, string text2)

{

int n1 = text1.size();

int n2 = text2.size();

vector<vector<int>> dp(n1+1,vector<int>(n2+1,0));

int r=0;

for(int i=1;i<=n1;i++)

{

for(int j=1;j<=n2;j++)

{

if(text1[i-1] == text2[j-1])

{

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

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

}

else

dp[i][j] = 0;

}

}

return dp[n1][n2];

}

3.**palindrome**

<https://leetcode.com/problems/longest-palindromic-substring/>  
<https://leetcode.com/problems/longest-palindromic-subsequence/>  
<https://leetcode.com/problems/minimum-insertion-steps-to-make-a-string-palindrome/>  
<https://leetcode.com/problems/delete-operation-for-two-strings/>

int lps(string s1)

{

int n1=s1.length();

string s2=s1;

reverse(s2.begin(),s2.end());

int n2=s2.length();

vector<vector<int>> dp(n1+1,vector<int>(n2+1,0));

for(int i=1;i<=n1;i++)

{

for(int j=1;j<=n2;j++)

{

if(s1[i-1]==s2[j-1])

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

else

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

}

}

return dp[n1][n2];

}

4.**Print**

string longestCommonSubsequence(string a)

{

string b=a;

reverse(b.begin(),b.end());

int n1=a.size();

int n2=b.size();

vector<vector<int>> d(n1+1,vector<int>(n2+1,0));

for(int i=1;i<=n1;i++)

{

for(int j=1;j<=n2;j++)

{

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

{

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

}

else

{

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

}

}

}

string v;

int i=n1,j=n2;

while(i>0 && j>0)

{

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

{

v.push\_back(a[i-1]);

i--;

j--;

}

else

{

if(d[i-1][j]>d[i][j-1])

{

i--;

}

else

{

j--;

}

}

}

reverse(v.begin(),v.end());

return v;

}

5.**Gap Method Problems**

General Dp problem which is solved by Gap method

<https://leetcode.com/problems/count-different-palindromic-subsequences/>  
<https://leetcode.com/problems/palindrome-partitioning-ii/>  
<https://leetcode.com/problems/minimum-score-triangulation-of-polygon/>

And Leetcode stones problem set are also included.

All the above problems can be solved by gap methodwith minor tweaks.  
Below is a standard code for gap method code.

**count palindromic subsequence**

int countPalindromicSubsequences(string s)

{

int d[s.length()][s.length()];

for(int g=0;g<s.length();g++)

{

for(int i=0,j=g;j<s.length();i++,j++)

{

if(g==0)

{

d[i][j]=1;

}

else if(g==1)

{

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

{

d[i][j]=3;

}

else

{

d[i][j]=2;

}

}

else

{

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

{

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

}

else

{

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

}

}

}

}

return d[0][s.length()-1];

}

6.**Kadans algo**

Identify if problems talks about finding the maximum subarray sum.

<https://leetcode.com/problems/best-time-to-buy-and-sell-stock/>  
<https://leetcode.com/problems/best-time-to-buy-and-sell-stock-ii/>  
<https://leetcode.com/problems/maximum-absolute-sum-of-any-subarray/>  
<https://leetcode.com/problems/arithmetic-slices/>  
<https://leetcode.com/problems/arithmetic-slices-ii-subsequence/>  
<https://leetcode.com/problems/longest-turbulent-subarray/>  
<https://leetcode.com/problems/k-concatenation-maximum-sum/>  
<https://leetcode.com/problems/k-concatenation-maximum-sum/>  
<https://leetcode.com/problems/length-of-longest-fibonacci-subsequence/>  
<https://leetcode.com/problems/ones-and-zeroes/>  
<https://leetcode.com/problems/maximum-sum-circular-subarray/>

All the above problems can be solved by gap method with minor tweaks.  
Below is a standard code for gap method code.

int kad(vector<int> v)

{

int c=v[0],o=v[0];

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

{

if(c >= 0)

{

c=c+v[i];

}

else

{

c=v[i];

}

if(o<c)

{

o=c;

}

}

return o;

}

7.**Catalan**

Identify if problems talks about counting the number of something.  
eg node,bracket etc.

<https://leetcode.com/problems/unique-binary-search-trees/>

All the above problems can be solved by catalan with minor tweaks.  
Below is a standard code for catalan code.

int cat(int n)

{

int dp[n+1];

dp[0]=1;

dp[1]=1;

for(int i = 2; i < n+1; i++)

{

dp[i]=0;

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

{

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

}

}

return dp[n];

}