**DP Basic**

DP = careful brute force

DP = memorization + recursion + guessing

DP = shortest path in a DAG (Directed Acyclic Graph)

Time of DP = time of sub problem \* number of sub problem

Five Easy Steps for DP from

http://www.youtube.com/watch?v=ENyox7kNKeY

1. Define Subproblems

2. Guess part of the solution

3. Relate Subproblem Solutions

4. Recursion & Memorization (Decide bottom-up or top-down)

5. Solve Original Problem

1. **Coin Change**

Source: <http://www.geeksforgeeks.org/dynamic-programming-set-7-coin-change/>

To count total number of solutions, we can divide all set solutions in two sets.  
1) Solutions that do not contain m-th coin (or Sm).  
2) Solutions that contain at least one Sm.  
Let count(S [], m, n) be the function to count the number of solutions, then it can be written as sum of count(S [], m-1, n) and count(S [], m, n-Sm).

Let say we have set of note, S = {1, 2, 3} and amount, n = 5.  
The function C ({1, 3}, 1) is called two times. If we draw the complete tree, then we can see that there are many sub problems being called more than once.

C () --> count()

C({1,2,3}, 5)

/ \

/ \

C({1,2,3}, 2) C({1,2}, 5)

/ \ / \

/ \ / \

C({1,2,3}, -1) C({1,2}, 2) C({1,2}, 3) C({1}, 5)

/ \ / \ / \

/ \ / \ / \

C({1,2},0) C({1},2) C({1,2},1) C({1},3) C({1}, 4) C({}, 5)

/ \ / \ / \ / \

/ \ / \ / \ / \

. . . . . . C({1}, 3) C({}, 4)

/ \

/ \

. .

|  |  |
| --- | --- |
| **Recursive Solution:**  *int count( int S[], int m, int n )*  *{*  *if (n == 0)*  *return 1;*    *if (n < 0)*  *return 0;*    *if (m <=0 && n >= 1)*  *return 0;*    *return count(S, m - 1, n) + count(S, m, n-S[m-1]);*  *}* | **DP Solution:**  *int count( int S[], int m, int n )*  *{*  *int table[n+1];*  *memset(table, 0, sizeof(table));*  *table[0] = 1;*    *for(int i=0; i<m; i++)*  *for(int j=S[i]; j<=n; j++)*  *table[j] += table[j-S[i]];*    *return table[n];*  *}* |

1. **Longest Palindrome Substring**

Source: <http://leetcode.com/2011/11/longest-palindromic-substring-part-i.html>

*#define MAX\_STR\_LEN 100*

*bool table[MAX\_STR\_LEN][MAX\_STR\_LEN];*

*//Time = O(N2) and space = O(N2)*

*string longestPalindromeDP(string s) {*

*int n = s.length();*

*int longestBegin = 0;*

*int maxLen = 1;*

*for (int i = 0; i < MAX\_STR\_LEN; i++)*

*for (int j = 0; j < MAX\_STR\_LEN; j++)*

*table[i][j] = false;*

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

*table[i][i] = true;*

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

*{*

*if (s[i] == s[i+1])*

*{*

*table[i][i+1] = true;*

*longestBegin = i;*

*maxLen = 2;*

*}*

*}*

*for (int len = 3; len <= n; len++)*

*{*

*for (int i = 0, j = len-1; j < n; i++, j++)*

*{*

*if (s[i] == s[j] && table[i+1][j-1])*

*{*

*table[i][j] = true;*

*longestBegin = i;*

*maxLen = len;*

*}*

*}*

*}*

*return s.substr(longestBegin, maxLen);*

*}*

1. **Longest Palindrome Subsequence**

Source: <http://www.geeksforgeeks.org/dynamic-programming-set-12-longest-palindromic-subsequence/>

Let X[0..n-1] be the input sequence of length n and L(0, n-1) be the length of the longest palindromic subsequence of X[0..n-1].

If last and first characters of X are same, then L(0, n-1) = L(1, n-2) + 2.  
Else L (0, n-1) = MAX (L (1, n-1), L (0, n-2)).

Following is a general recursive solution with all cases handled.

// every single character is a palindrome of length 1

L (i, i) = 1 for all indexes i in given sequence

// IF first and last characters are not same

If (X[i] != X[j]) L(i, j) = max{L(i + 1, j),L(i, j - 1)}

// If there are only 2 characters and both are same

Else if (j == i + 1) L(i, j) = 2

// If there are more than two characters, and first and last

// characters are same

Else L (i, j) = L(i + 1, j - 1) + 2

|  |  |
| --- | --- |
| **Recursive Solution:**  *int lps(char \*seq, int i, int j)*  *{*  *// If there is only 1 character*  *if (i == j)*  *return 1;*    *//If there are 2 char which are same*  *if (seq[i] == seq[j] && i + 1 == j)*  *return 2;*    *// If the first and last charmatch*  *if (seq[i] == seq[j])*  *return lps (seq, i+1, j-1) + 2;*    *// If first and last char not match*  *return max(lps(seq, i, j-1), lps(seq, i+1, j) );*  *}* | **DP Solution:**  *int dp[100][100];*  *//Time = O(N2) and space = O(N2)*  *int subSequence\_LongestPalindrome(string Str){*  *int n = Str.length();*  *//Base case*  *for (int i = 0; i < n; i++)*  *dp[i][i] = 1;*  *for (int len=2; len<=n; len++){*  *for (int i = 0, j = len-1; j < n; i++, j++){*  *if (Str[i] == Str[j]){*  *if(len == 2) dp[i][j] = 2;*  *else dp[i][j] = dp[i+1][j-1] + 2;*  *}*  *else{*  *dp[i][j] = max(dp[i][j-1], dp[i+1][j]);*  *}*  *}*  *}*  *return dp[0][n-1];*  *}* |

1. **Longest Biotonic Subsequence**