What is DP?

DP is another technique for problems with optimal substructure:

An optimal solution to a problem contains optimal solutions to subproblems

Note that this doesn’t necessarily mean that every optimal solution to a subproblem will contribute to the main solution.

- For divide and conquer (top down), the subproblems are independent so we can solve them in any order.

- For greedy algorithms (bottom up), we can always choose the “right” subproblem by a greedy choice.

- In dynamic programming, we solve many subproblems and store the results: not all of them will contribute to solving the larger problem.

Because of optimal substructure, we can be sure that at least some of the subproblems will be useful

Steps for solving DP problems

1. Define subproblems

2. Write down the recurrence that relates subproblems

3. Recognize and solve the base cases

Problem 1: Fibonacci Sequence

Naïve Recursive Function

int Fib(int n){

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

return 1;

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

DP solution O(n):

Fib[1] = Fib[2] =1;

for(i=3;i<N;i++)

Fib[i] = Fib[i-1]+Fib[i-1]

Problem 2: Given n, find the number of different ways to write n as the sum of 1, 3, 4

Example: for n = 5, the answer is 6

5 = 1+1+1+1+1

= 1+1+3

= 1+3+1

= 3+1+1

= 1+4

Define subproblems

D\_n be the number of ways to write n as the sum of 1, 3, 4

Find the recurrence

D\_n = D\_{n−1} + D\_{n−3} + D\_{ n−4}

Solve the base cases

D\_0 = 1

D\_n = 0 for all negative n

Alternatively, can set: D\_0 = D\_1 = D\_2 = 1, D\_3 = 2

Implementation

D[0] = D[1] = D[2] = 1; D[3] = 2;

for(i = 4; i <= n; i++)

D[i] = D[i-1] + D[i-3] + D[i-4];

Problem 3: given two strings x and y, find the longest common subsequence (LCS) and print its length

Example:

x: ABCBDAB

y: BDCABC

“BCAB” is the longest subsequence found in both sequences, so the answer is 4

Analysis

There are 2^m subsequences of X.

Testing a subsequence (length k) takes time O(k + n).

So brute force algorithm is O(n\* 2^m)

Divide and conquer or Greedy algorithm?

No, can’t tell what initial division or greedy choice to make.

Thus, none of the approaches we have learned so far work here!!!

Intuition: A LCS of two sequences has as a preﬁx a LCS of preﬁxes of the sequences.

So, We concentrate on LCS for smaller problems, i.e simply removes the last (common) element.

Define subproblems

Let D\_{i,j} be the length of the LCS of x\_{1...i} and y\_{1...j}

Find the recurrence

If x\_i = y\_j , they both contribute to the LCS

D\_{i,j}= D\_{i-1,j-1} + 1

either x\_i or y\_j does not contribute to the LCS, so one can be dropped. Otherwise,

D\_{i,j} = max{ D\_{i-1,j} , D\_{i,j-1}}

Find and solve the base cases: D\_{i,0} = D\_{0,j} = 0

Implementation:

for(i = 0; i <= n; i++) D[i][0] = 0;

for(j = 0; j <= m; j++) D[0][j] = 0;

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

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

if(x[i] == y[j])

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

else

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

}

}

Recovering the LCS

Modify the algorithm to also build a matrix D[1 : : : n; 1 : : :m], recording how

the solutions to subproblems were arrived at.

Problem 4: Longest Non Decreasing Subsequence

Given an array 1 , 2 , 5, 2, 8, 6, 3, 6, 9, 7

Find a subsequence which is non decreasing and of maximum length

1-5-8-9 Forms a non decreasing subsequence

So does 1-2-2-6-6-7 but it is longer

Subproblem: Length of LNDS ending at i^{th} Loc

Implementation:

for(i=0;i<100;i++){

max= 0;

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

if(A[i] >= A[j] && L[j] > max)

max = L[j];

}

L[i] = max+1;

}

Problem 5: Given a tree, color nodes black as many as possible without coloring two adjacent nodes

Subproblems:

- we arbitrarily decide the root node r

- B\_v : the optimal solution for a subtree having v as the root, where we color v black

- W\_v : the optimal solution for a subtree having v as the root, where we don’t color v

-The answer is max{B\_r , W\_r}

Find the recurrence

observation: once v’s color is determined, its subtrees can be solved independently

If v is colored, its children must not be coloredl

B\_v = 1 + \sum\_{u \in child(v)} W\_u

If v is not colored, its children can have any color

W\_v = 1 + \sum\_{u \in child(v)} B\_u

Base cases: leaf nodes