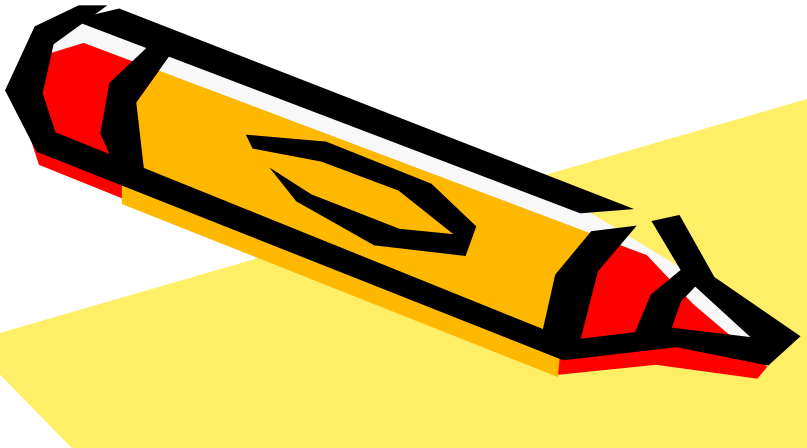
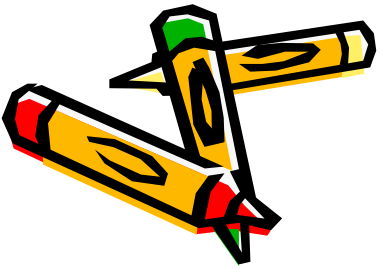


Dynamic Programming

By Harry Wiggins

Introduction

- DP is a confusing name for a programming technique that dramatically reduces the runtime from exponential to polynomial time.
- Essentially we avoid solving the same problem twice, by finding subproblems(like strong induction).
- We demonstrate this with an example.



Strong induction

- Let S(n) be a statement concerning some integer n ie

  $1^2 + 2^2 + \ldots + n^2 = n(n+1)(2n+1)/6$

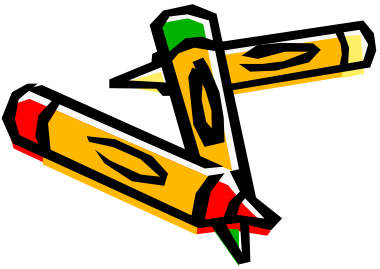
- S(1) is true
- Using the fact that S(1), …, S(n-1) is true we deduce that S(n) is true.
- Hence result holds for all statements.

Problem 1 : Increasing subsequence

Given a sequence of integers, how would you find the maximum subsequence (it need not be consecutive)?

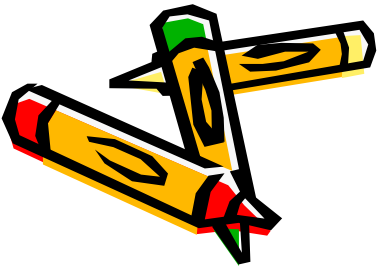
Q : 3 6 4 9 8 10

A : 4 (3 4 8 10)



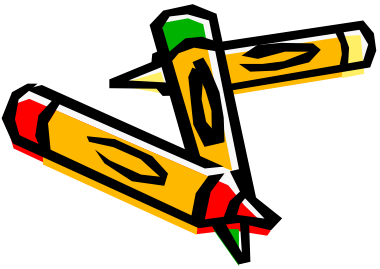
Recursive solution

```
check(start, nmatches,biggest){
    int better, i,best=nmatches;
    for (i =start; i < n;i++){
        if (sequence[i]>biggest){
            better = check(i,nmatches+1,sequence[i]);
            if (better>best) best = better;
            }
        }
    return best;
}
```

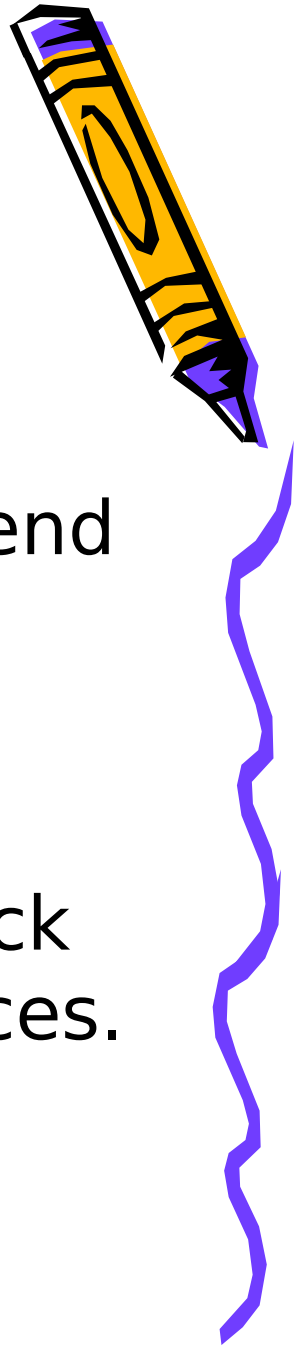


It’s horrible!

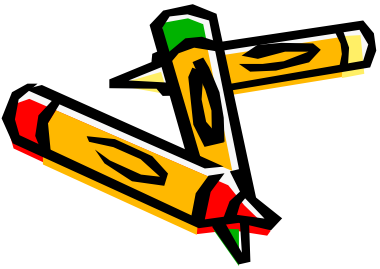
- Exponential time (ie if consider an increasing sequence it cost 2^n)
- Could potentially end up with a huge stack.
- Doing the same calculations more than once.



Find the subproblem



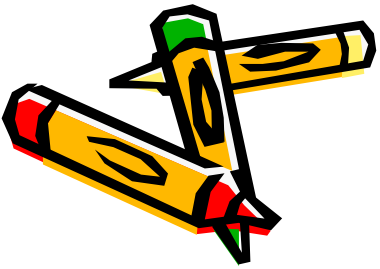
- It seems the length of the longest subsequence from position p to the end is important.
- Most of the time a bit of storage is necessary for execution efficiency.
- So we have a best array(keeping track of longest tail increasing subsequences.



The solution

```
best[n-1]=1;
for (int i=n-2;i>=0;i--){
best[i]=1;
for (int j=i+1;j<n;j++){
    if (num[i]<num[j])
        best[i]=max(best[i],1+best[j]);
}
cout << max(best.begin(),best.end()) << endl;
```

What if you want the actual sequence?

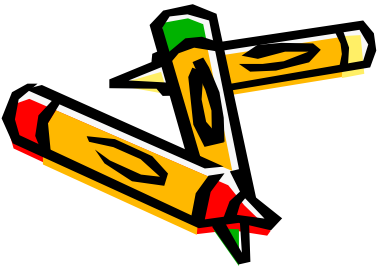


Problem 2 : String matching

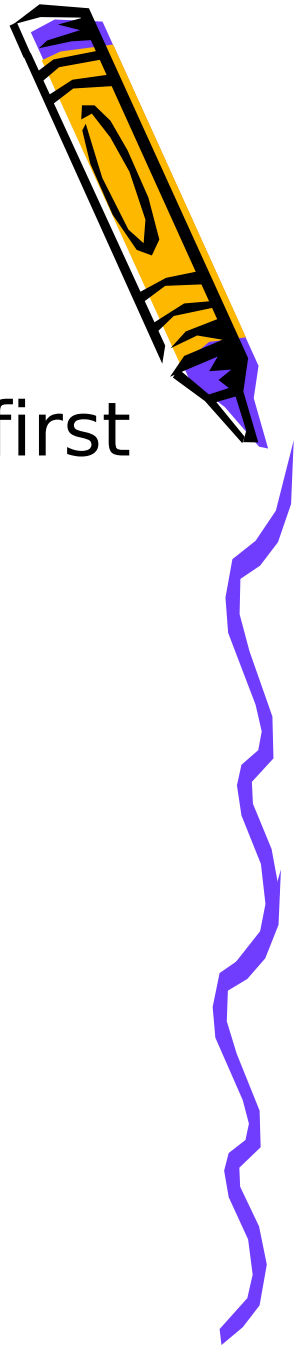
Your friend gives you two words. You want to remove letters from each word so that you end up with the same word. How would you determine the longest such possible word?

Q : America and Alabama

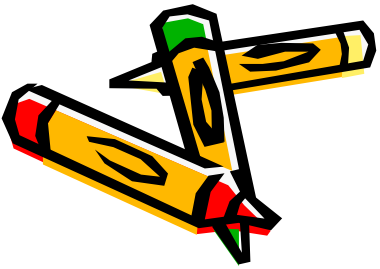
A : 3 (ama)



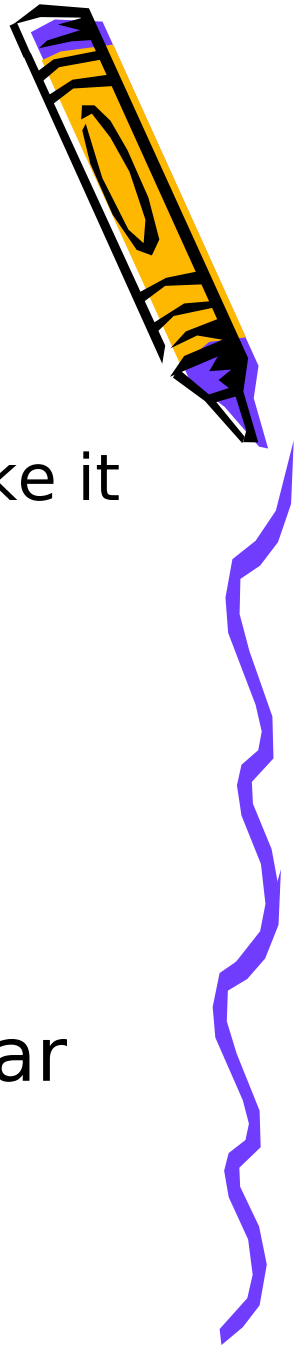
Identify the subproblem



- match[a][b] ie the biggest match of first a characters of first string and first b characters of second string.
- match[a][b] = max(match[a-1][b], (1+match[a-1][b-1])*δ_{match})
- Order of calculation values
- Answer is match[n][m]
- This example is 2D previous was 1D.



Problem 3 : Hidden DP



Given a word what is the least number of letters you need to insert anywhere to make it a palindromic word?

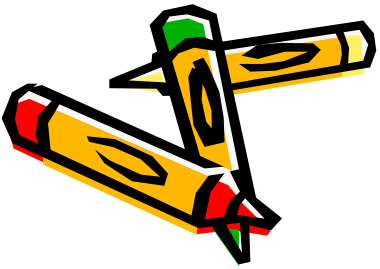
Q : BANANA

A : 1 (add B at end)

Did you know : aibohphobia is the fear of palindromes?

Solution

- Answer is length of string – length of biggest palindromic subset.
- Note it’s a match between string and its reverse.
- Easy exercise to show that the longest such match will be a palindrome.
- (IOI 2000 Day 1 Question 1)



Problem 4 : Coin problem

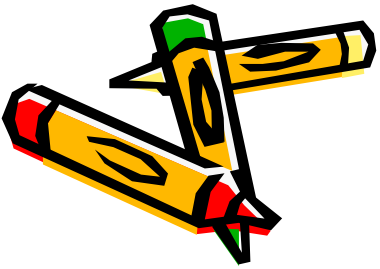
Given a coin system how would you calculate the least number of coins you need to pay a certain amount?

Answer : Clearly need extra array least[n] that stores the least to pay amount n.

least[n] = min(1+least[n-c1],1+least[n-c2], ...)

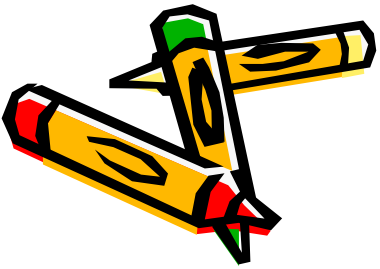
Problem 5 : 1998 IOI Day 2 Question 3

You are given a regular N-gon with numbers on node and operators on edges. Every turn you remove an edge and combine cross edges with the node with the value equal to the operation on edge applied to those nodes. How would you maximize the final value?

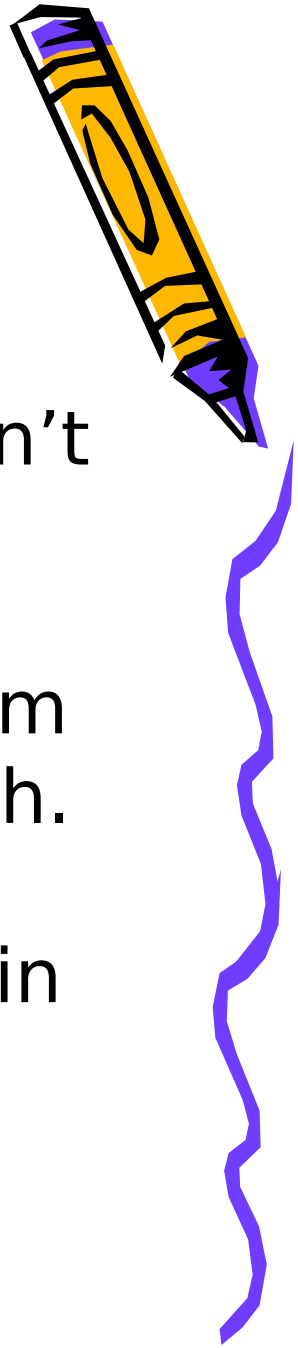


Answer?

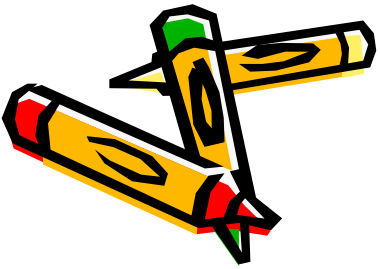
- The subproblem is to keep track of the maximum value possible from node a to node b.
- Update function best[a][b] = max(operation a best[a+1][b], operation best[a][b-1] b)
- Order do best[a][a+1], then best[a][a +2], etc.
- Answer is best[1][1]



Problem 6 : Integer knapsack

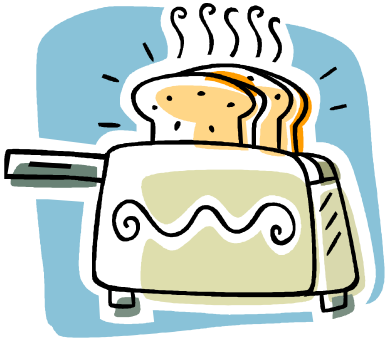


You are designing a contest which isn’t allowed to be longer than a certain predetermined length. You are also given a set of problems. Each problem has a point value and a certain length. Find the contest which has the maximum number of points but within the length constraint.



Need to be careful!

- Can’t re-use a problem.
- Subproblem is most points for length l contest after using m problems.
- best[l][m]=max(best[l][m-1],best[l-length(m)][m-1])
- However we can be a bit space efficient.
- If we update in the right order we only need an one-dimensional array.



The End.