THE WORLD OF DYNAMIC PROGRAMMING

Recently I came across the an algorithm that can find the maximum sub array in a given 1-D array. Its called the Kadane's algorithm.It comes under the domain of dynamic programming. I would like to write about a few things I understood about Dynamic programing so that it may help someone else to get a start and at the same time to improve my own knowledge on the topic.

Dynamic programming is an optimization problem or a programming technique, where the given problem is split into sub problems and so on.

To understand more about dynamic programming we need to look at how it obtains optimization. Dynamic programming builds its optimal solution by creating optimal solutions for its subproblems.This is also called optimal substructure.

It follows the principal of Optimality which states that "A problem is said to satisfy the Principle of Optimality if the sub solutions of an optimal solution of the problem are themselves optimal solutions for their subproblems."

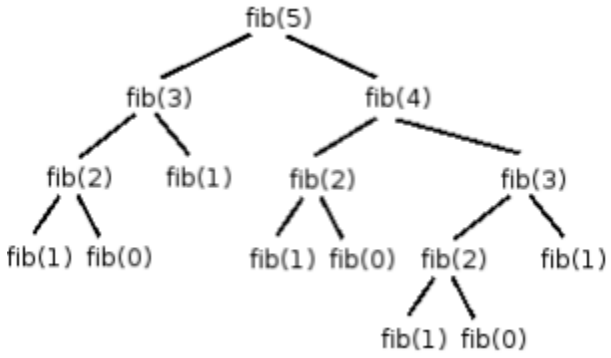
Or to put it in simpler terms optimal substructures means that the optimal solution to a problem is made up of optimal solutions to its sub problems.

Another technical term used in dynamic programming is Overlapping subproblems. This means that there are a few subproblems in total and
many recurring instances of each.

Dynamic programming uses a bottom up design process. A very good example of this will be the fibonacci series generation.

Lets first look at the recursive solution to generate fibonacci series. Instances like f(1) ,f(0) repeat multiple times. If these values can be stored they don't have to be calculated multiple times.

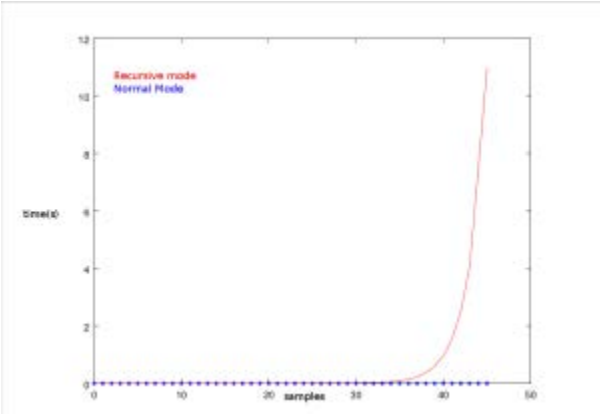
The left side shows the number of steps taken by a dynamic program and the right side shows the steps taken bya recursive program to calculate fibonacci series.



An example of Dynamic Programming and Recursion

It is also to be noted that dynamic programming is not the same as divide and conquer, because dynamic programming proceeds by splitting the problem space in every possible manner.

I have tested the efficiency of the of both kinds of Fibonacci code. And here are my findings with respect to time taken for running the code.



And below is the code used for testing this.

```
#include<stdio.h>
```

```
#include <time.h>



int fibonacci1(int n)

{

 int a=0,b=1;

 int i,c;

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

   {

     c=a+b;

     a=b;

     b=c;

     //printf("%d\n",c);

   }

 return b;

}
```

```
int recfib(int n)

{

  if ( n == 0 )

    return 0;

  else if ( n == 1 )

    return 1;

  else

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

}

int main()

{

  clock_t t;

  double normaltime=0,rectime=0;

  int sample;

  for(sample=0;sample<100;sample++)

    {
```

```
        t=clock();

        fibonacci1(sample);

        t = clock() - t;

        normaltime = ((double)t)/CLOCKS_PER_SEC; // in seconds

        t=clock();

        recfib(sample);

        t = clock() - t;

        rectime = ((double)t)/CLOCKS_PER_SEC; // in seconds

        printf("%f,%f\n",normaltime,rectime);

      }

    return 0;

  }
```

Source : http://thebeautifullmind.com/