DYNAMIC PROGRAMMING

Introducción

La programación dinámica se trata de un paradigma, uno difícil de aplicar, el cual está enfocado en relaciones de recurrencias.

Existe una importante cantidad de problemas que pueden ser resueltos usando este paradigma, así que es necesario prestar mucha atención esta sección aunque resulte muy difícil entender la teoría lo cual es algo que suele ocurrir muchas veces para los programadores que están iniciando en este mundo.

Algo importante a tener en cuenta es la abreviatura para hacer referencia a *dynamic programming* la cual será DP.

Antes de continuar es necesario hacer mención sobre la existencia de dos enfoques de DP:

1. Top down (basado directamente en backtracking).
2. Bottom up (técnica de tabulación).

En este caso el enfoque a conocer será ***Bottom up***, debido a que es la forma pura de DP.

Concepto

Se debe de entender como una técnica basado en una fórmula de recurrencia de los algunos estados. La solución que se va a conseguir por medio de esta técnica será obtenida usando soluciones anteriores de sub-problemas.

Entonces, lo primero a entender son las sub-soluciones para los sub-problemas. Es debido a lo anterior que se entenderá como un estado (state) una manera de describir una situación, que será la solución para el problema, de manera que si nosotros queremos encontrar la solución para el estado 'i' (solución al problema), primero debemos de encontrar sub-soluciones a estados anteriores y esto abre a la percepción de descomponer el problema original en sub-problemas, tal que la siguiente solución será construida usando las soluciones a sub-problemas anteriores.

Esto nos hace entender que primero debemos tener las soluciones de los sub-problemas más pequeños para poder ir construyendo con ellas otras soluciones cada vez más grandes hasta conseguir la solución del problema original, finalmente quedaría resumido.

Solución Sub-problema + … + Solución Sub-problema = Solución problema original

La clave de la habilidad a adquirir será con respecto a la identificación de los estados de los problemas para poder determinar las relaciones entre el problema actual con sus sub-problemas.

Bottom-Up DP – método de tabulación

Esta es la forma pura del DP y es implementada mediante el uso de una tabla.

Como ya se hizo mención anteriormente, lo primero a realizar será descomponer el problema original en problemas muchísimos más pequeños para poder resolver primero estos. Sin embargo, aun teniendo los problemas más pequeños (sub-problemas simples) no podemos hacer que la computadora los resuelva mágicamente para cada tipo de situación, sino que forzosamente debemos de tener una solución base que siempre se cumpla para poder resolver los siguientes sub-problemas, y aquí es donde entra el concepto de ***casos base*.**

Casos base

Son soluciones a uno o más sub-problema(s) (el o los más próximo(s)), tal que se sepa que dichas soluciones **siempre** serán las mismas y no van a cambiar. Es por esto lo anterior que los caso(s) base serán, entonces, la solución al primer estado que se distinga del problema.

Un ejemplo sería

Dado una lista de N monedas (diferente valor), encontrar la mínima cantidad de número de monedas cuya suma sea igual al número S (número objetivo).

Para este problema la forma de obtener el caso base será, entonces, descomponiendo el problema original en problemas más pequeños y aquí es fácil observar que los estados serán formar sumas pequeñas con las monedas para después formar la suma solicitada.

Y el caso base, o sea la solución a un estado que conocemos y no va cambiar es la cantidad necesaria de monedas para formar un suma de 0, así que para formar 0 necesitamos 0 monedas, ya una vez con esto podemos pasar al siguiente estado que será formar una suma de 1 por lo cual tendremos que checar entre nuestras monedas si tenemos alguna capaz de formar 1, no vale alguna moneda que sobrepase esa cantidad.

Es de esta manera que se puede hacer la identificación de los casos base.

Pasos para una solución

De forma general podemos nombrar los siguientes pasos para dar con una solución.

1. Determinar los parámetros únicos que describen al problema, paso similar a backtracking.
2. Si tenemos N parámetros para representar los states se debe tener una tabla DP de dimensión N con una entrada por estado.
3. Una vez encontrado los casos base se podrá inicializar la tabla y se podrá encontrar la siguiente solución de los sub-problemas, este proceso se va repetir hasta llenar la tabla y hará mediante loops (for, while).

Ejemplo con problema – Fibonacci

Dado un número (posición), encontrar el número de Fibonacci que pertenezca a la posición dada por el usuario usando DP.

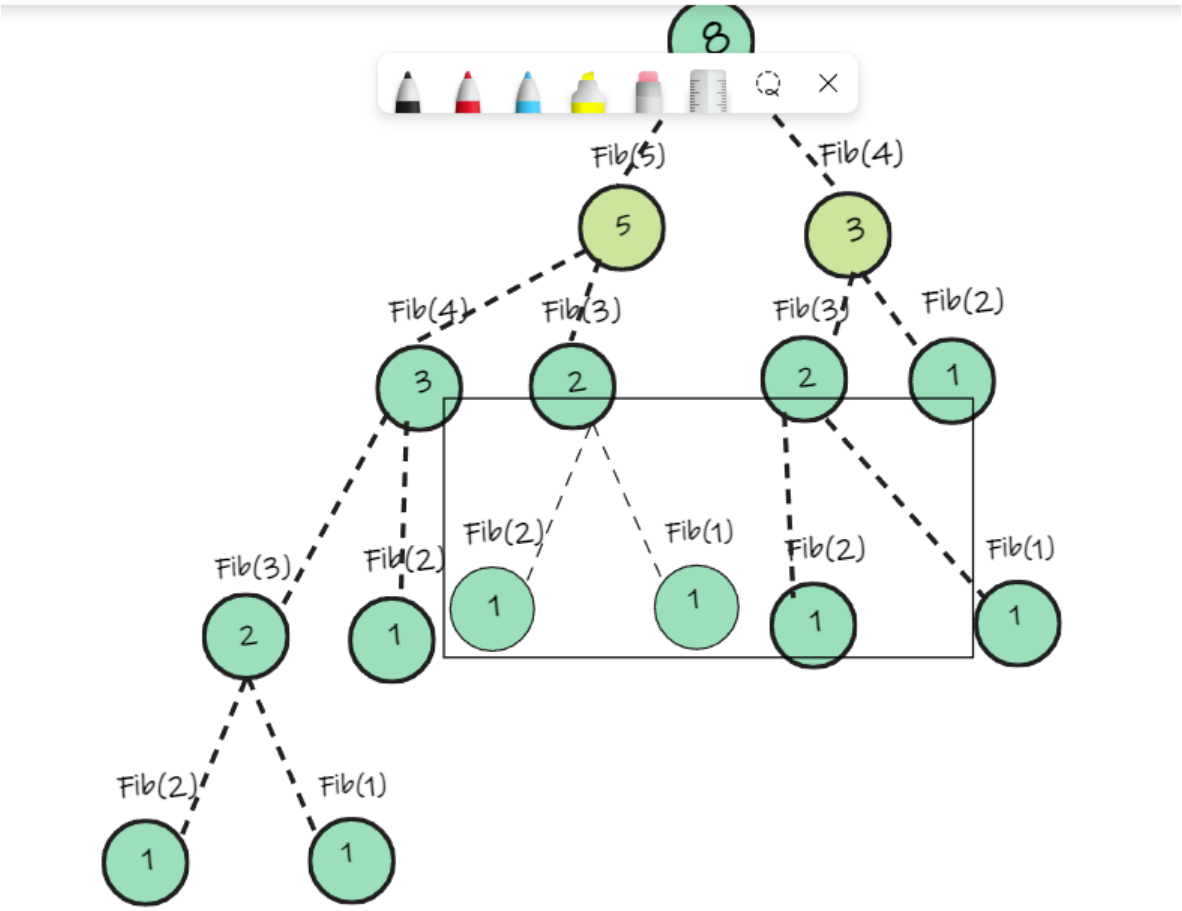
Interpretación: el usuario nos va a proporcionar un número N el cual deberá de ser interpretado como una posición, tal que sea la salida sea el número de Fibonacci N. Así que se tendrá que aplicar el criterio de la búsqueda de los states para armar la solución.

Sabemos que la sucesión de Fibonacci se calcula usando los dos números anteriores, entonces se estaría necesitando de dos números anteriores para obtener el siguiente número. De aquí es fácil ver cuáles son los states, serían simplemente la generación de estos números de la sucesión hasta llegar a la posición solicitada por el usuario.

Los states serán entonces cada posición, debido a que cada una de estas se estará generando un número de Fibonacci.

Identificación de los pasos:

1. Parámetros: como parámetro solo será uno que es el número dado por el usuario, como es un solo parámetro la tabla será de una sola dimensión.
2. Casos base: en este caso tenemos más de un caso base y será para las posiciones de la tabla: 0, 1, 2. Esto es debido a que nosotros sabemos que siempre los tres primeros de la sucesión de Fibonacci son 0, 1, 1, los cuales siempre serán los tres primeros y ya los conocemos de ante mano así que estos valores son inicializados en la tabla DP.
3. Colocar casos base: inicializar la tabla de DP con los casos base para poder obtener la solución del siguiente state.



Ahora, el paso siguiente es el más importante y es la generación de una fórmula para poder computar los states. Para esto debemos de fijarnos sobre el comportamiento de la tabla y como este debe de estar funcionando para ir creando los siguientes números.

Por ejemplo, calcular fib(6)

Esta es nuestra tabla actualmente

0	1	1				

Se podrá ver que en cada iteración del loop será posible mandar a llamar a los últimos dos celdas (con valores) para realizar la suma del nuevo valor

Entonces quedaría celda_4 = dp(2) + dp (1) =>> celda_4=1+1 =>> celda_4=2

0	1	1	2			

Ahora, la celda_5

Celda_5 = dp (3) + dp (2) =>> celda_5=2+1 =>> celda_5=3

0	1	1	2	3		

Celda_6

Celda_6= dp (4)+fib(3) =>> celda_6=3+2 =>> celda_6=5

0	1	1	2	3	5	

Celda_7

Celda_7= dp (5)+ dp (4) =>> celda_7=5+8 =>> celda_7=8

0	1	1	2	3	6	8

Es muy importante hacer la observación que el resultado es el que se encuentra en la última posición. En cada problema de DP se debe de hacer un análisis para ver en dónde queda ubicado el resultado

Así quedaría implementada la tabla, entonces se puede deducir la formula usada para conocer estos valores

Dp[n] = dp (n-1)+ dp (n-2)

El cual va estar dentro de un arreglo de tamaño n

Es por esto que se puede decir que su complejidad de espacio es ‘n’

Space complexity = O(n) Time complexity = O(n)

```cpp
long long dp[100];

long long fibonacci(int n)
    {
        // inicializando con los casos base
        dp[0]=0;
        dp[1]=1;
        dp[2]=1;

        for(int i=3;i<=n;i++)
            {
                dp[i] = dp[i-1] + dp[i-2];
            }
        // al final devuelvo el resultado
        return dp[n]; // Porque está indexado en 0
    }

int main()
    {
        cout<<(fibonacci(50));
        return 0;
    }
```

Este es el ejemplo más sencillo sobre la idea de DP y tabular para conseguir la solución. Ahora, se dará un paso más y será otro problema en donde se verá el empleo de DP para ir construyendo la solución e ir escogiendo la solución más óptima en cada state para al final obtener la solución más óptima.