Programación Dinámica

5 de mayo de 2019

Sucesión de Fibonacci

Definición

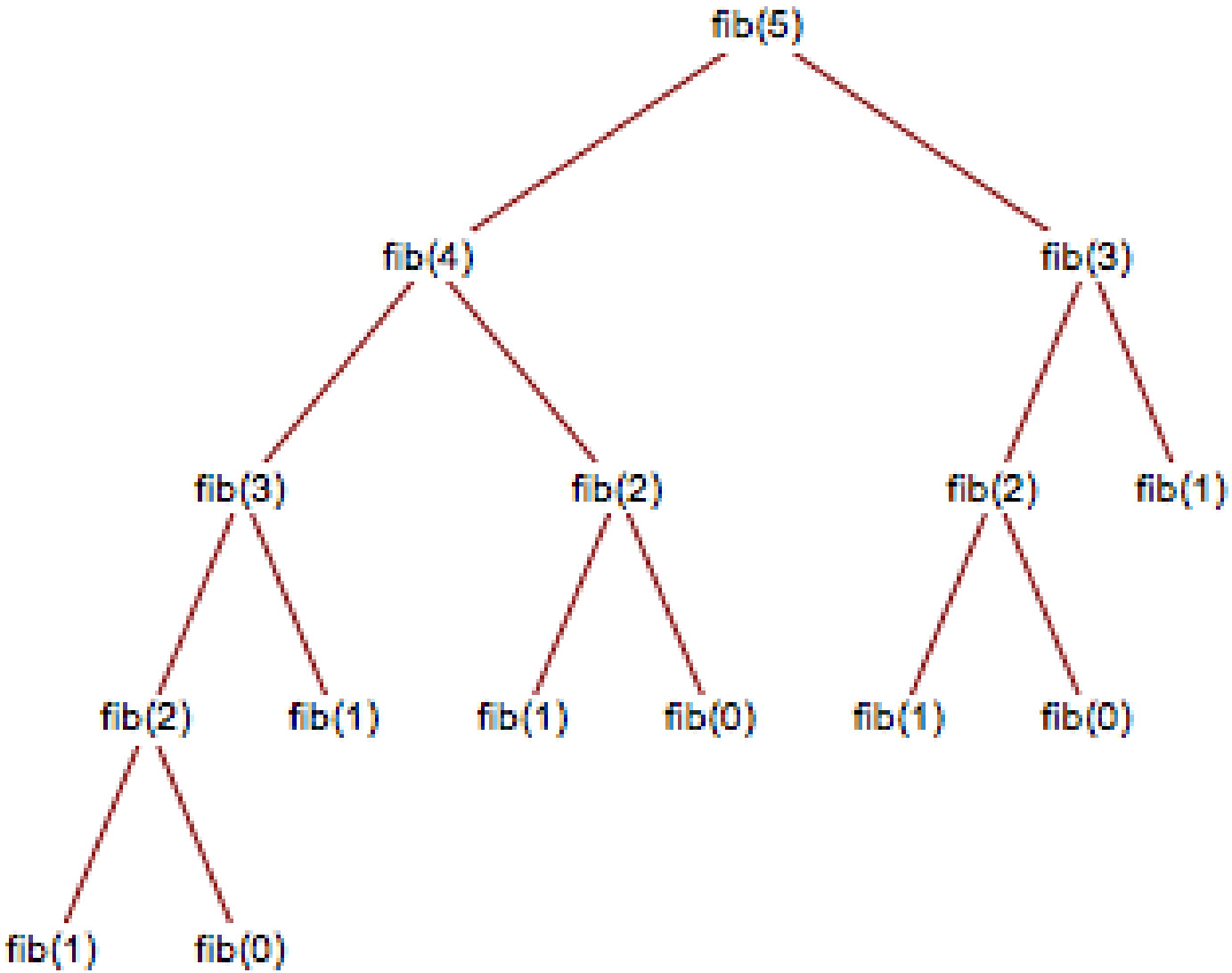
La sucesion de Fibonacci se define como $fib_0 = 0$, $fib_1 = 1$, $fib_n = fib_{n-1} + fib_{n-2}$ (para n $\geq$ 2). Los primeros terminos son 0, 1, 1, 2, 3, 5, 8, 13, 21...

- La definición nos sugiere una forma de computarlo (función recursiva)

Fibonacci recursivo

```
int fib(int n) {
if(n<=1) return n;
return fib(n-1) + fib(n-2);
}
```

Árbol generado por Fib(5)



Fibonacci en programación dinámica

- Podemos hacerlo mejor: en lugar de recalcular la respuesta cada vez, la calculamos una sola vez y guardamos el resultado

Fibonacci - programación dinámica

```
int mem[MAX_N];
int fib(int n){
    // mem esta lleno de -1 al principio
    if(mem[n] >= 0)return mem[n];
    int res;
    if(n<=1){
        res = n;
    }else{
        res = fib(n-1) + fib(n-2);
    }
    mem[n] = res;
    return res;
}
```

Fibonacci - programación dinámica

- Esto es una función recursiva
- En el arreglo mem guardamos los resultados para los n's que ya calculamos (usamos un valor negativo para indicar que no lo calculamos)
- Ahora la complejidad de calcular todos los fibonaccis hasta n es O(n)

Programación dinámica - Definición

- La programación dinámica es, básicamente, una estrategia donde se guardan los resultados ya calculados, sabiendo que pueden hacer falta después.
- Podemos verla como un método para resolver un problema, que lo parte en subproblemas más pequeños, los cuáles resuelve recursivamente, guardando los resultados para no calcularlos más de una vez.

Programación dinámica - Estados

- En el contexto de programación dinámica, llamamos estado a cada combinación de parámetros de la misma. En el caso de Fibonacci, el estado es el valor de n.
- Para calcular el costo de la programación dinámica, la cuenta suele ser **cantidad de estados * costo de calcular el valor de un estado**

Fibonacci - versión iterativa

```
int mem[MAX_N];
int calcFib(int n){
    fib[0]=0;
    fib[1]=1;
    for(int i = 2 ; i<=n ; i++){
        fib[i] = fib[i-1] + fib[i-2];
    }
}
```

- Todo problema de programación dinámica se puede resolver recursivamente (cómo se mostró antes) o iterativamente.
- La forma recursiva se suele llamar "memoización" (memoization en inglés)

Programación dinámica iterativa vs. recursiva

- Tanto la forma iterativa como la recursiva tienen ventajas y desventajas
- Forma iterativa:
  - Suele ser más eficiente en tiempo, porque se evitan las llamadas a función.
  - A veces permite ahorrar memoria
  - El código suele resultar más corto.
- Forma recursiva:
  - Para algunos es el modo más "natural" de pensar la programación dinámica.
  - No requiere pensar en el orden en el que debemos calcular los sub-problemas, que no siempre es obvio.
  - Es más eficiente cuando hay muchos valores de la tabla que no es necesario calcular (no es muy común, pero puede ocurrir)

Problemas de optimización

- La programación dinámica se puede usar también para resolver algunos problemas de optimización (encontrar la mejor solución, de acuerdo a cierto criterio)
- ¿Se acuerdan del problema de la moneda? (Greedy). El caso general se puede resolver con programación dinámica.

Problema de la moneda

Enunciado

Tenemos n tipos de monedas de valores v_0 , v_1 , ... , v_{n-1} y queremos saber la menor cantidad de monedas que necesitamos para sumar exactamente m.

Problema de la moneda

Enunciado

Tenemos n tipos de monedas de valores v_0 , v_1 , ... , v_{n-1} y queremos saber la menor cantidad de monedas que necesitamos para sumar exactamente m.

- Veamos el código que soluciona el problema...

Problema de la Moneda - Programación dinámica

```
int v[MAXN];
int dp[MAXN][MAXM];
/*
coin = indice en arreglo de monedas

res = cuanta plata me queda por pagar
*/
int solve(int coin, int res){
    if(res<0) return INF;
    if(coin == -1){
        if(res == 0){
            return 0;
        }else{
            return INF;
        }
    }
    if(dp[coin][res] >= 0) return dp[coin][res];

    int dp[coin][res] =
            min(solve(coin-1,res), 1 + solve(coin, res - v[coin]);

    return dp[coin][res]
}
```

Moneda - Generar solución

- En muchos problemas de optimización, además del valor de la solución óptima, se nos pide que demos explícitamente la solución (en el caso de moneda, qué tengo que tomar de cada tipo)
- Podemos aprovechar que tenemos guardado el resultado de todos los sub-problemas. Empezamos desde el estado dp[coins,res]
  - Si dp[coin,res] = dp[coins-1,res] es porque decidimos no elegir la moneda y recursionamos para (coins-1, res)
  - Si no, es porque tenemos que agregar la moneda coin. La agregamos al resultado y recursionamos para (coins, res - v_{coins})
  - Seguimos hasta llegar a (0,0)

Problema de la Moneda - Código para generar solución

```
int dp[MAXN][MAXM];
vector<int> monedas;
void gen_sol(int coin, int res){
    if(res==0)return;
    if(coin!=0 && dp[coin][res] == dp[coins-1][res]){
        gen_sol(coin-1,res);
    }else{
        monedas.push_back(coin);
        gen_sol(coin,res-v[coin]);
    }
}

int generate(int coin, int res){
    memset(dp,-1,sizeof(dp));
    solve(coin,res);
    gen_sol(coin,res);
    // en el vector monedas tenemos las monetas que necesitamos
}
```

Generar soluciones

- Este patrón para generar solución óptima se puede aplicar a cualquier programación dinámica que resuelva un problema de optimización
- Partimos del estado que nos interesa
- En cada paso usamos los valores calculados para ver qué decisión coincide con el resultado del estado
- Guardamos en la solución la decisión que tomamos
- Recursionamos para los subproblemas

Conclusión

- Programación dinámica es uno de los tópicos más comunes en el contexto de programación competitiva, por lo que es muy importante dedicarle tiempo a practicarlo
- Un concepto muy relacionado a programación dinámica del cual no hablamos es **backtracking**. Es lo mismo que programación dinámica, pero sin guardar información de los subproblemas resueltos (solo la recursión que prueba todas las posibilidades). Se usa backtracking cuando la estructura del problema es tal que cada sub-problema se resuelve una sóla vez.

Problemas

- `https://atcoder.jp/contests/dp/tasks`
- `https://www.a2oj.com/category?ID=33`

En particular, vamos a realizar A y B de **https://atcoder.jp/contests/dp/tasks** en clase.