Programação dinâmica

Professor: Wladimir Araújo Tavares

A programação dinâmica é uma técnica de programação que ajuda a resolver de maneira eficiente uma classe de problemas que possuem sobreposição de subproblemas e uma substrutura ótima. A seguir, apresentaremos um problema com sobreposição de subproblemas.

Problema 1

Considere o seguinte problema:

Seja s_n o conjunto de palavras de tamanho n formada pelos caracteres $\{a, b, c\}$ que não possui dois caracteres a's consecutivos. Qual é a cardinalidade de s_n?

Por exemplo,

- $s_1 = \{a, b, c\}$, $|s_1| = 3$
- $s_2 = \{ab, ac, ba, bb, bc, ca, cb, cc\}$, $|s_2| = 8$

Note que os seguintes casos podem acontecer: * bs_{n-1}, pode começar com a letra b seguido por uma palavra em s_{n-1} * cs_{n-1}, pode começar com a letra b seguido por uma palavra em s_{n-1} * abs_{n-2} ou acs_{n-2}, pode começar por ab ou ac seguido por uma palavra em s_{n-2}.

Note que o mesmo subproblema aparece mais de uma vez. Neste caso, podemos tirar proveito da programação dinâmica para desenvolver um algoritmo eficiente.

Podemos definir $x_n = |s_n|$ de maneira recursiva da seguinte maneira:

$x_1 = 2$ $x_2 = 8$ $x_n = 2x_{n-1} + 2x_{n-2}$

Implementação

Bottom-up

```
int num_palavra(int n){
    long long dp[n+1];

    dp[1] = 3;
    dp[2] = 8;
    for(int i = 3; i <= n; i++){
        dp[i] = 2*dp[i-1] + 2*dp[i-2];
    }

    return dp[n];
}
```

Top-down

```
#include "bits/stdc++.h"
#define MAXN 20
using namespace std;

int dp[MAXN];

int num_palavra(int n){

    if( dp[n] != -1)
        return dp[n];
    else{
        dp[n] = 2*num_palavra(n-1) + 2*num_palavra(n-2);
        return dp[n];
    }
}

int main()
{
    int n;
    cin >> n;
    memset(dp, -1, sizeof(dp) );
    dp[1] = 3;
    dp[2] = 8;
    cout << num_palavra(n) << endl;
    return 0;
}
```

Problema 2

Dois jogadores disputavam um prêmio que seria dado a quem primeiro fizesse 3 pontos no jogo. Quando o primeiro jogador tinha 2 pontos e o segundo tinha 1 pontos, foi preciso interromper o jogo. Supondo que ambos os jogadores tem a mesma chance de ganhar um 1 ponto. Qual é a probabilidade do jogador 1 vencer a partida?

Seja $p_{i,j}$ a probabilidade do jogador 1 vencer a partida dado que o jogador 1 tem i pontos e o jogador 2 tem j pontos.

O valor de $p_{i,j}$ pode ser definido de maneira recursiva da seguinte maneira:

$p_{nj} = 1.0, \quad \forall j, 1 \leq j < n (1)$

$p_{in} = 0.0, \quad \forall i, 1 \leq i < n (2)$

$p_{ij} = 0.5p_{i+1,j} + 0.5p_{i,j+1}, \quad \forall i, j, i \neq n, j \neq n (3)$

Podemos preencher a tabela de subproblemas com as informações das equações

(1) e (2)

$p_{i,j}$	1	2	3
1			0.0
2			0.0
3	1.0	1.0	*

Agora, podemos calcular $p_{2,2}$:

$p_{2,2} = 0.5 * p_{3,2} + 0.5 * p_{2,3} = 0.5 * 1 + 0.5 * 0 = 0.5$

$p_{i,j}$	1	2	3
1			0.0
2		0.5	0.0
3	1.0	1.0	*

Agora, podemos calcular $p_{2,1}$ e $p_{1,2}$

$p_{2,1} = 0.5 * p_{3,1} + 0.5 * p_{2,2} = 0.5 * 1.0 + 0.5 * 0.5 = 0.75$

$p_{1,2} = 0.5 * p_{2,2} + 0.5 * p_{1,3} = 0.5 * 0.5 + 0.5 * 0 = 0.25$

$p_{i,j}$	1	2	3
1		0.25	0.0
2	0.75	0.5	0.0
3	1.0	1.0	*

Agora, podemos calcular $p_{1,1}$:

$p_{1,1} = 0.5 * p_{2,1} + 0.5 * p_{1,2} = 0.5 * 0.75 + 0.5 * 0.25 = 0.375 + 0.125 = 0.5$

$p_{i,j}$	1	2	3
1	0.5	0.25	0.0
2	0.75	0.5	0.0
3	1.0	1.0	*

No problema acima, temos uma grande sobreposição de subproblemas. Se não utilizarmos a programação dinâmica, o mesmo subproblema será resolvido múltiplas vezes.

Implementação

```
#include "bits/stdc++.h"
#define MAXP 100
using namespace std;

double dp[MAXP+1][MAXP+1];

double prob(int i, int j, int n){

    if( i == n) return 1.0;
    else if( j == n ) return 0.0;
    else if( dp[i][j] > 0) return dp[i][j];
    else {
        dp[i][j] = 0.5*prob(i+1, j, n) + 0.5*prob(i, j+1, n);
        return dp[i][j];
    }
}

int main()
{
    int n;

    for(int i = 0; i <= MAXP; i++)
        for(int j = 0; j <= MAXP; j++)
            dp[i][j] = -1;

    cout << prob(15,13,20) << endl;

    return 0;
}

//0.725586
```

Problema 3

Dado um vetor a de tamanho n que a soma dos seus elementos é igual a M, e dado algum $K \leq M$, queremos saber se existe um subconjunto dos números do vetor a tal que a soma desse subconjunto dá exatamente K?

Por exemplo, se $a = \{2, 3, 1, 4\}$ e $K = 7$, podemos encontrar o subconjunto {3,4} cuja soma é igual a 7.

Primeiramente, vamos adicionar um elemento a mais no conjunto com valor 0. No exemplo anterior, o vetor ficaria igual a $a = \{0, 2, 3, 1, 4\}$

Vamos definir o nosso subproblema da seguinte maneira $m[i][j] = true$, se é possível encontrar um subconjunto do conjunto $\{a[0], a[1], \ldots, a[i]\}$ tal que a

soma do conjunto é igual a j.

O caso base do nosso problema será:

$m[0][0] = true$ $m[0][j] = false, \quad \forall j, j > 0$

No caso geral, temos a seguinte recorrência:

$m[i][j] = m[i-1][j], j < a[i]$

$m[i][j] = m[i-1][j]$ or $m[i-1][j-a[i]]$, caso contrário

Perceba o seguinte, se j for menor que o valor de $a[i]$, o elemento i não pode ser usado para nenhum subconjunto com a soma igual a j. Se j for maior que $a[i]$, então precisamos checar se a soma j era possível antes ou se a soma $j - a[i]$ era possível antes.

Bottom-up

a = {0, 2,3,1,4}

Caso Base:

$m[i][j]$	0	1	2	3	4	5	6	7
0	T	F	F	F	F	F	F	F

$i = 1, a[1] = 2$

$m[i][j]$	0	1	2	3	4	5	6	7
0	T	F	F	F	F	F	F	F
1	T	F	T	F	F	F	F	F

$i = 2, a[2] = 3$

$m[i][j]$	0	1	2	3	4	5	6	7
0	T	F	F	F	F	F	F	F
1	T	F	T	F	F	F	F	F
2	T	F	T	T	F	T	F	F

$i = 3, a[3] = 1$

$m[i][j]$	0	1	2	3	4	5	6	7
0	T	F	F	F	F	F	F	F
1	T	F	T	F	F	F	F	F

$m[i][j]$	0	1	2	3	4	5	6	7
2	T	F	T	T	F	T	F	F
3	T	T	T	T	T	T	T	F

i = 4, a[3] = 4

$m[i][j]$	0	1	2	3	4	5	6	7
0	T	F	F	F	F	F	F	F
1	T	F	T	F	F	F	F	F
2	T	F	T	T	F	T	F	F
3	T	T	T	T	T	T	T	F
4	T	T	T	T	T	T	T	T