Programação Dinâmica

Samuel Raimundo

Universidade Federal de Viçosa

23 de outubro de 2024

01-INTRUDUÇÃO-FIBONACCI

Sequência de Fibonacci

- $F(0) = 0$
- $F(1) = 1$
- $F(n) = F(n-1) + F(n-2)$, se $n > 1$

01-INTRUDUÇÃO-FIBONACCI

Sequência de Fibonacci

- $F(0) = 0$
- $F(1) = 1$
- $F(n) = F(n-1) + F(n-2)$, se $n > 1$

Números da sequência de Fibonacci

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, ...

01-INTRUDUÇÃO-FIBONACCI

Problema:

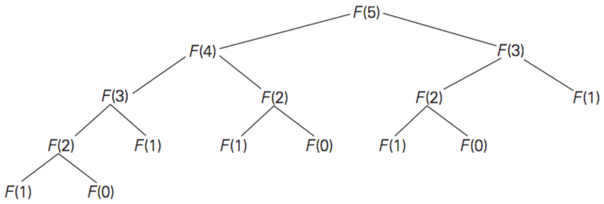
Dado n, qual o valor de $F(n)$?

01-INTRUDUÇÃO-FIBONACCI

Problema:
Dado n, qual o valor de $F(n)$?

```
long long fibo(int n){
    if(n==0) return 0;
    if(n==1) return 1;
    return fibo(n-1)+fibo(n-2);
}
```

01-INTRUDUÇÃO-FIBONACCI



Motivo:

- Um mesmo valor é resolvido várias vezes.
- Veja quantas vezes F(1) é chamado!
- Note que F(2) é resolvido várias vezes...
- Imagine com n grande!

01-INTRUDUÇÃO-FIBONACCI

```
long long tab[MAX] = {0,1};
bool solved[MAX] = {true,true};

long long fibo(int n){
    if(solved[n]) return tab[n];
    tab[n] = fibo(n-1) + fibo(n-2);
    solved[n] = true;
    return tab[n];
}
```

Vantagem:

- *fibo*(n) é calculado uma única vez para cada n.

2 - PROGRAMAÇÃO DINÂMICA - PD

Como?

- Em vez de resolver o mesmo subproblema de novo e de novo...
- ... guardar o resultado dos já resolvidos em uma "tabela" (memoization)
- ... e consultar a tabela para não resolvê-los novamente.

2 - PROGRAMAÇÃO DINÂMICA - PD

Quando?

- Tipicamente em problemas de otimização ("encontre o máximo..", "encontre o mínimo..") e problemas de contagem ("conte quantos...")
- **Subestrutura ótima**: solução ótima do problema pode ser obtida a partir de soluções ótimas de subproblemas menores do mesmo tipo
- **Sobreposição**: vários subproblemas precisam da solução ótima dos mesmos subproblemas menores
- **Casos base**: Estados que podem ser 'facilmente' calculados.

2 - PROGRAMAÇÃO DINÂMICA - PD

PD "Top-down"

- Ideia: dividir o problema em subproblemas menores
- Guardar soluções dos subproblemas em uma tabela à medida que são resolvidos
- Só resolver um subproblema se sua solução ainda não está guardada na tabela
- Recursão + memoization

2 - PROGRAMAÇÃO DINÂMICA - PD

PD "Bottom-up"

- Ideia: resolver antecipadamente os subproblemas que podem ser necessários
- Resolver os problemas em “ordem de tamanho” guardando os resultados em uma tabela
- Ao resolver um problema, seus subproblemas já foram resolvidos

3 - EXEMPLOS: COIN-ROW PROBLEM

Fila de moedas

- Há uma fila de n moedas cujos valores são inteiros positivos $c_1, c_2, \ldots, c_n$ no necessariamente differentos.
- O objetivo é pegar o máximo valor com a restrição de não pegar duas moedas adjacentes.

- Exemplo: 5 1 2 10 6 2
- Solução: 5 1 2 10 6 2, de valor 17

3 - EXEMPLOS: COIN-ROW PROBLEM

Para a n-ésima moeda há duas opções:

- Pegar: ganha c_n e continua o processo da $n\!-\!2$
- Não pegar: continua o processo da $n\!-\!1$

3 - EXEMPLOS: COIN-ROW PROBLEM

Para a n-ésima moeda há duas opções:

- Pegar: ganha c_n e continua o processo da $n-2$
- Não pegar: continua o processo da $n-1$

$F(n) = \max\{c_n + F(n-2), F(n-1)\}$ *for* $n > 1$,
$F(0) = 0,\ F(1) = c_1$

3 - EXEMPLOS: COIN-ROW PROBLEM

```cpp
int coin_row(const vector<int> &coins,
             vector<int> &memo, int x){
    if(x<0) return 0;
    if(memo[x]!=-1) return memo[x];
    memo[x] = max(coin_row(coins,memo,x-1),
                  coin_row(coins,memo,x-2)+coins[x]);
    return memo[x];
}
```

3 - EXEMPLOS: TROCO (PROBLEMA DE DECISÃO)

Problema do Troco

Dado um conjunto de M valores de moeda $v_1, v_2, ..., v_m$ e um estoque ilimitado de cada um, é possível somar exatamente N?

3 - EXEMPLOS: TROCO (PROBLEMA DE DECISÃO)

Exemplo

- $M = 3$, valores 3, 7, 15
  - $N = 7$: 7
  - $N = 8$: impossível
  - $N = 9$: $3 + 3 + 3$
  - $N = 10$: $3 + 7$
  - $N = 11$: impossível
  - $N = 12$: $3 + 3 + 3 + 3$
  - $N = 13$: $3 + 3 + 7$
  - $N = 14$: $7 + 7$
  - $N = 15$: 15 ou $3 + 3 + 3 + 3 + 3$

3 - EXEMPLOS: TROCO (PROBLEMA DE DECISÃO)

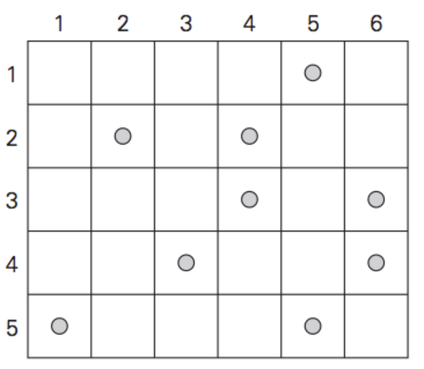
```cpp
bool troco(const vector<int> &coins, int k){
    vector<bool> tab(k+1,false);
    tab[0] = true;
    for(int i=0;i<k;i++){
        if(tab[i]){
            for(int c: coins){
                if(c+i<=k){
                    tab[i+c] = true;
                }
            }
        }
    }
    return tab[k];
}
```

3 - COIN-COLLECTING PROBLEM

Coletor de moedas

- Várias moedas são espalhadas numa grade de *nxm* casas
- Um robô, localizado no canto superior-esquerdo, deve coletar o máximo de moedas e levá-las até o canto inferior-direito
- Ele coleta as moedas das casas que visita e em cada passo pode ir de sua posição atual para a casa à direita ou a casa abaixo
- O objetivo é determinar o máximo de moedas que o robô pode coletar (e o caminho pelo qual ele consegue isto)

3 - COIN-COLLECTING PROBLEM



3 - COIN-COLLECTING PROBLEM

Para cada casa escolher se é melhor vir de cima ou da esquerda

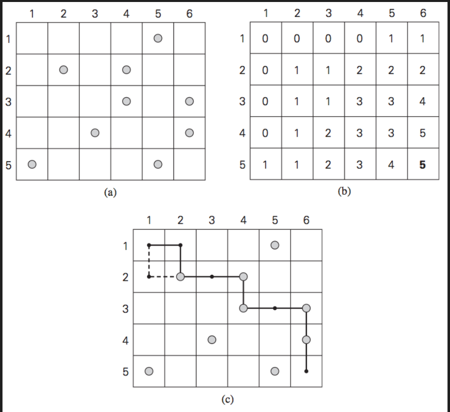
3 - COIN-COLLECTING PROBLEM

Para cada casa escolher se é melhor vir de cima ou da esquerda

$F(i, j) = maxF(i-1, j), F(i, j-1) + c_{ij}$ *for* $1 \leq i \leq n$, $1 \leq j \leq m$

$F(0, j) = 0$ *for* $1 \leq j \leq m$ and $F(i, 0) = 0$ *for* $1 \leq i \leq n$

3 - COIN-COLLECTING PROBLEM



3 - COIN-COLLECTING PROBLEM

```
int coin_collection(const vector<vector<int>> &tab,
                    vector<vector<int>> &memo,int i,int j){

    if(i<0 or j<0) return 0;
    if(memo[i][j]!=-1) return tab[i][j];

    int ans = max(coin_collection(tab,memo,i-1,j),
                  coin_collection(tab,memo,i,j-1)) + tab[i][j];

    memo[i][j] = ans;
    return memo[i][j];
}
```