Programação Dinâmica

Prática

André, Salles

Departamento de Informática
Universidade Federal de Viçosa

INF 333 - 2024/1

Introdução - 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

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

Problema

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

Introdução - Fibonacci

Fibonacci - recursivo

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

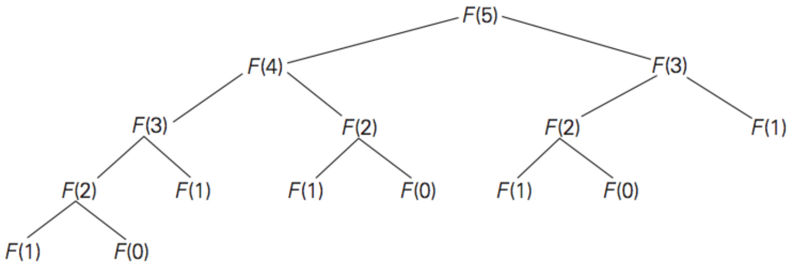
Exercício #1

Implementar a versão do Fibonacci recursivo.

- Fibo(10) = 55
- Fibo(20) = 6765
- Fibo(30) = 832040
- Fibo(40) = ?

- Note como começa a ficar lento a partir de 40...

Introdução - 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!

Introdução - Fibonacci

Fibonacci - recursivo (com tabela)

```
long long tab[MAX] = {0, 1};        //números de fibonacci
bool solved[MAX] = {true, true};    //quem já foi calculado

long long int 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*

Introdução - Fibonacci

- Outra versão, sem array de booleanos
- `tab[n]` é inicializado com -1 e só é calculado uma vez

Fibonacci - recursivo (com tabela)

```
long long tab[MAX];  //inicializar com 0, 1, -1, -1, -1,...

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

Introdução - Fibonacci

Exercício #2

Implementar a versão do Fibonacci com tabela.

- Fibo(10) = 55
- Fibo(20) = 6765
- Fibo(30) = 832040
- Fibo(40) = ?

- Note a diferença de desempenho em relação à outra versão

Fibonacci - sem recursividade

```
long long tab[MAX];

long long fibo(int n)
{
  tab[0] = 0;
  tab[1] = 1;

  for(int i=2; i<=n; i++)
    tab[n] = tab[n-1] + tab[n-2];
  return tab[n];
}
```

Vantagens:

- Elimina a recursividade
- Não é necessário inicializar a tabela

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.

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

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

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

Programação Dinâmica - PD

As duas versões eficientes mostradas para o cálculo do Fibonacci ilustram a PD top-down e a PD bottom-up.

Exemplo - Troco (problema de decisão)

Problema do troco

Dado um conjunto de M valores de moeda $\{v_1, v_2, \ldots, v_m\}$ e um estoque ilimitado de cada valor, é possível somar exatamente N?

Exemplo

- $M = 3$, valores $\{3, 7, 15\}$
  - $N = 5$: impossível
  - $N = 6$: $3 + 3$
  - $N = 7$: 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 $3 + 3 + 3 + 3 + 3$

Exemplo - Troco (problema de decisão)

Exercício #3

Implementar uma solução por força bruta (*backtracking*)

- o que é uma solução parcial?
- quais as alternativas em cada chamada?
- quais os critérios de parada?

Exemplo - Troco (problema de decisão)

Algoritmo força bruta

```
//possivel devolver n com as m primeiras moedas de v?
bool troco(int n, int m, int v[]) {

  if (n<0) return false;  //devolveu mais que podia
  if (n==0) return true;  //devolveu exatamente o valor
  if (m==0) return false; //nao tem mais moeda pra devolver

  if (troco(n-v[m-1],m,v))  //tenta devolver a moeda v_m
    return true;
  if (troco(n,m-1,v))       //tenta sem usar a moeda v_m
    return true;

  return false;             //nao consegue devolver n
}
```

Exemplo - Troco (problema de decisão)

Testar com este exemplo

- $M = 3$, valores $\{4, 6, 10\}$
  - $N = 100$
  - $N = 101$
  - $N = 120$
  - $N = 121$
  - $N = 10000$
  - $N = 10001$

- Note que começa a ficar lento com N grande...

Exemplo - Troco (problema de decisão)

Exercício #4

Transforme seu (*backtracking*) numa PD

- `tab[n] =` possível devolver *n*?

- Inicializar a tabela com -1 (não resolvido), e preencher com 1 (true = possível) ou 0 (false = impossível) à medida que vai resolvendo

Exemplo - Troco (problema de decisão)

Testar com o mesmo exemplo

- $M = 3$, valores $\{4, 6, 10\}$
  - $N = 100$
  - $N = 101$
  - $N = 120$
  - $N = 121$
  - $N = 10000$
  - $N = 10001$

- Será rápido mesmo para valores grandes de n

Exemplo - Troco (problema de decisão)

Exercício #5

Implementar uma PD bottom-up

Algoritmo

```
tab[0] = true;
tab[1..n] = false;
for(i=0;i<m;i++)                        //para cada moeda
    for(j=0;j<=n-v[i];j++)
        if (tab[j])                     //se e' possivel j
            tab[j+v[i]] = true;         //e' possivel j + moeda
```

todos os valores marcados com `true` *são possíveis*

Exemplo - Troco (problema de **localização**)

Exercício #5b

Se for possível, informar como

Algoritmo

```
tab[0] = 0;
tab[1..n] = -1;
for(i=0;i<m;i++)                   //para cada moeda
    for(j=0;j<=n-v[i];j++)
        if (tab[j]!=-1)            //se e' possivel j
            tab[j+v[i]] = v[i];    //e' possivel j + moeda
```

`tab[n]` *informa o valor da moeda usada para atingir n;*

`tab[n - tab[n]]` *informa o valor da moeda anterior;*

e assim por diante, bastar "voltar" em `tab` *de n até* 0

Exemplo - Troco (problema de **otimização**)

Exercício #5c

Informar a quantidade mínima

Algoritmo

```
tab[0] = 0;
tab[1..n] = INF;
for(i=0;i<m;i++)                    //para cada moeda
    for(j=0;j<=n-v[i];j++)
        if (tab[j]!=INF)            //se e' possivel j
            tab[j+v[i]] = min(tab[j+v[i]],tab[j]+1);
```

`tab[n]` *informa o mínimo de moedas para n*

Exemplo - Troco sem repetição (problema de decisão)

Problema do troco sem repetição (SUBSET SUM)

Dado um conjunto de M moedas $\{v_1, v_2, \ldots, v_m\}$, é possível somar exatamente N?

Exemplo

- $M = 3$, valores $\{3, 3, 7\}$
    - $N = 5$: impossível
    - $N = 6$: $3 + 3$
    - $N = 7$: 7
    - $N = 8$: impossível
    - $N = 9$: impossivel
    - $N = 10$: $3 + 7$
    - $N = 11$: impossível
    - $N = 12$: impossível
    - $N = 13$: $3 + 3 + 7$
    - $N > 13$: impossível

Exemplo - Troco (problema de decisão)

Exercício #6

Modificar sua solução *backtracking* do troco para resolver o SUBSET SUM.

- Dica: devolvendo ou não a moeda, ela não poderá ser usada mais

Exercício #7

Modificar sua PD bottom-up do troco para resolver o SUBSET SUM.

- Dica: inverter o “for j” para não usar um ‘true’ recém-marcado