UNIVERSIDADE DE SÃO PAULO

Instituto de Ciências Matemáticas e de Computação

SCC0218 – Algoritmos Avançados e Aplicações

Fibonacci em O($log_2 n$) (Divisão e Conquista)

Elias Italiano Rodrigues – 7987251
Rodolfo Megiato de Lima – 7987286

1. Introdução

Este trabalho consiste em implementar um algoritmo para calcular o n-ésimo termo da sequência de Fibonacci com ordem de complexidade O($log_2 n$).

2. Desenvolvimento

Para desenvolver o algoritmo, foram utilizados os subsídios 1 e 2 dados na descrição do trabalho. A partir do subsídio 1, foi obtida a seguinte fórmula:

$$\begin{bmatrix} F_n \\ F_{n-1} \end{bmatrix} = A^{n-1} \times B \text{ , onde } A = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \text{ e } B = \begin{bmatrix} F_1 \\ F_0 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

O algoritmo para a fórmula acima tem a complexidade O($log_2 n$), uma vez que, pelo subsídio 2, a fórmula para calcular a n-ésima potência de uma matriz A é:

$$\begin{cases} pow(A,n) = A, \text{ se } n = 1 \\ pow(A,n) = pow^2(A,n/2), \text{ se } n \text{ é par} \\ pow(A,n) = A \times pow^2(A,\lfloor n/2 \rfloor), \text{ se } n \text{ é ímpar} \end{cases}$$

A justificativa da complexidade O($log_2 n$) vem do método de divisão e conquista aplicado na recorrência acima. Isso pode ser concluído observando-se que:

- A cada chamada recursiva, o tamanho do problema é dividido em dois gerando uma árvore de recursão de altura $log_2 n$.
- Em cada nível k da árvore é realizada uma operação de multiplicação de matrizes considerada de complexidade O(1), pois o resultado de $pow^2(A,\lfloor n/2^k \rfloor)$ é feito atribuindo-se $R \leftarrow pow(A,\lfloor n/2^k \rfloor)$ e então $R \leftarrow R \times R$ se n é par, ou $R \leftarrow A \times (R \times R)$ se n é ímpar.
- Portanto: se em cada nível da árvore é feita uma operação O(1) e sua altura é $log_2 n$, a ordem complexidade é $O(1) \cdot \log_2 n \Rightarrow O(\log_2 n)$

A implementação do projeto foi feita em linguagem C. Não houve a necessidade de criação de estruturas de dados, pois foram utilizadas somente matrizes que são simples arranjos de números inteiros em linguagem C.

Foi necessário usar variáveis inteiras de 64 bits para guardar os valores das operações, pois o próprio limite da entrada (10^{11}) já estoura o limite de representação de inteiros com 32 bits. Além

disso, no pior caso, mesmo aplicando *mod* 10^6 nas multiplicações, teríamos por algum instante um valor na ordem de 10^{12}.

O código é composto de 3 funções que juntas executam o algoritmo. São elas:

```
unsigned long long int **matrix_mul(unsigned long long int **A, unsigned long long int **B, unsigned long long int mod);
```

```
unsigned long long int **matrix_pow(unsigned long long int **M, unsigned long long int n, unsigned long long int mod);
```

```
unsigned long long int F(unsigned long long int n, unsigned long long int mod);
```

Confira a documentação no código-fonte do projeto.

3. Conclusão

Conferimos em páginas da Internet[1][2] alguns valores para a sequência de Fibonacci e pudemos constatar que o algoritmo está funcionando. Também retornou a saída esperada, segundo o monitor da turma, para o caso $n = 10^{11}$ que é 937501.

4. Referências

[1] http://www.maths.surrey.ac.uk/hosted-sites/R.Knott/Fibonacci/fibtable.html
[2] http://php.bubble.ro/fibonacci/