SCC0220 - Laboratório Introdução à Ciência da Computação II

Relatório de execução da aula prática 2

Alunos	**NUSP**
Felipe Camargo Cerri	15451119
Gabriel Campanelli Iamato	15452920

Exercício 1 – Cálculo do MDC

Item a

➔ Comentário

Implementamos dois algoritmos distintos para o cálculo da exponenciação: um método iterativo e outro recursivo baseado em divisão e conquista. Logo de início, através da submissão de ambos no runcodes, foi possível perceber a complexidade reduzida da segunda implementação. Isso ficou evidente pois a versão iterativa excedeu o tempo limite em metade dos casos de teste.

A implementação do algoritmo iterativo foi realizada por meio de um loop, que, a cada iteração, multiplica o resultado parcial pela base.

Na versão recursiva, por sua vez, o método de divisão e conquista é responsável por reduzir consideravelmente a complexidade. Ele se baseia em dividir o problema inicial em metades mais facilmente computáveis. Assim, para utilizá-lo de forma recursiva estabelecemos o caso base como a base da exponencial no momento que o expoente se iguala a 1, enquanto as chamadas da própria função ocorriam com o expoente reduzido pela metade, além disso, optamos por armazenar cada chamada em uma variável temporária e elevá-la ao quadrado para evitar chamadas desnecessárias que aumentariam a complexidade de tempo e memória. Finalmente, para tratar casos em que o expoente é ímpar, utilizamos uma condicional que multiplica o resultado parcial pela base.

É importante ressaltar que, assim como requisitado na atividade, nas duas implementações utilizamos a operação de módulo para limitar o resultado para suas quatro casas finais, com o objetivo de evitar overflow.

➔ **Código**

Versão iterativa:

```
#define mod 10000;

long int iterativo(int n, int k) {
    long int result = 1;
    for (int i =0; i<k; i++) {
        result *= n;
        result %= mod;
    }
    return result;
}
```

Versão recursiva:

```
long int divConq(int n, int k) {
    long int result = 1, temp;

    if (k == 1) { //caso base
        return n;
    }
    //faz correção caso k seja impar
    else if (k%2) {
        result *= n;
    }
    //temp evita chamar a funcao desnecessariamente
    temp = divConq(n, k: k/2);
    temp *= temp%mod;
    result *= temp;

    return result%mod;
}
```

➔ **Saída**

Para avaliar empiricamente a eficiência do algoritmo recursivo proposto, realizamos alguns casos teste com diferentes entradas. A comparação do tempo de execução do método convencional e do com divisão e conquista foi realizada com o auxílio da biblioteca “time.h”.

Menor tempo de execução: Divisão e Conquista

```
2 658
Resultado iterativo: 5744, Tempo de Execução: 0.000014
Resultado Divisão e Conquista: 5744, Tempo de Execução: 0.000002
```

Menor tempo de execução: Divisão e Conquista

```
55 17599
Resultado iterativo: 4375, Tempo de Execução: 0.000333
Resultado Divisão e Conquista: 4375, Tempo de Execução: 0.000002
```

Menor tempo de execução: Divisão e Conquista

```
258 82291
Resultado iterativo: 9392, Tempo de Execução: 0.001009
Resultado Divisão e Conquista: 9392, Tempo de Execução: 0.000003
```

Menor tempo de execução: Divisão e Conquista

```
546 280765197
Resultado iterativo: 2016, Tempo de Execução: 0.809466
Resultado Divisão e Conquista: 2016, Tempo de Execução: 0.000001
```

Menor tempo de execução: Divisão e Conquista

```
158 2089261943
Resultado iterativo: 9712, Tempo de Execução: 5.995475
Resultado Divisão e Conquista: 9712, Tempo de Execução: 0.000001
```

A partir dos testes realizados podemos concluir que o método de divisão e conquista, implementado recursivamente, possui uma complexidade de tempo menor, visto que, em todos os casos, o algoritmo teve menor tempo de execução, no entanto, devido às consecutivas chamadas recursivas, a complexidade de memória tende a ser levemente maior. Vale ressaltar que, para casos com números menores a diferença de eficiência de tempo não é significativa, mas à medida que os números crescem, a diferença entre os dois algoritmos passa a ser considerável.


ICMC
USP
SÃO CARLOS