Relatorio de Laboratório de Introdução à Ciência da Computação II – Avaliativo 2

Leonardo Kenzo Tanaka e Pedro Teidi de Sá Yamacita

10 de setembro de 2025

1 Análise do Código da IA

```c
#include <stdio.h>

int main() {
    int n, k;
    int resultado = 1;
    int i;

    // Leitura dos valores de entrada
    scanf("%d", &n);
    scanf("%d", &k);

    // Validacao das entradas
    if (n <= 0 || n > 99) {
        return 1;
    }
    if (k <= 0) {
        return 1;
    }

    // Calculo de n^k mod 1000 de forma iterativa
    for (i = 0; i < k; i++) {
        resultado = (resultado * n) % 1000;
    }

    // Saida do resultado (sem zeros a esquerda)
    printf("%d\n", resultado);
    return 0;
}
```

1.1 Análise da versão IA

O código acima apresenta a forma iterativa de calcular potências, gerado por inteligência artificial, mais especificamente pelo modelo Claude Sonnet 4.

É possível perceber que a IA realizou o cálculo da potência de maneira clássica, ao simplesmente multiplicar o número por ele mesmo k vezes, utilizando a operação de módulo por 1000 para evitar o estouro da variável e obter apenas os três últimos algarismos da potência.

No entanto, para valores muito grandes de k, como 10^9, o código se torna extremamente lento, pois realiza essa quantidade de iterações, resultando em complexidade O(k).

2 Análise do Código Iterativo

```c
#include <stdio.h>

int main()
{
    int n, k, resultado = 1;
```

```
6
7       //Leitura das variaveis
8       scanf(" %d %d", &n, &k);
9       if(n < 0 || n > 99 || k < 0 || k > 1000000000){
10          return 1;
11      }
12
13      //Dividir para conquistar
14      if (k % 2 == 0)
15      {
16          for (int i = 0; i < k / 2; i++)
17          {
18              resultado = (resultado * n) % 1000;
19          }
20          printf("%d", (resultado * resultado) % 1000);
21      }
22      else
23      {
24          for (int i = 0; i < k / 2; i++)
25          {
26              resultado = (resultado * n) % 1000;
27          }
28          printf("%d", (resultado * resultado * n) % 1000);
29      }
30  }
```

2.1 Análise do Código Iterativo

O código acima apresenta a forma iterativa para calcular potências. Como é possível notar, usamos o método de dividir para conquistar como descrito nas intruções da atividade mas de forma iterativa.

Além disso, utilizamos o módulo por 1000 para evitar o estouro da variável e obter apenas os três últimos algarismos da potência. No entanto, apesar dessa alteração na lógica de resolução, o programa ainda se mostra muito lento para valores de k muito elevados, uma vez que sua complexidade é de O(k/2).

3 Comparação Entre os Dois Códigos

A principal diferença entre os dois códigos está na lógica de resolução. O código iterativo gerado pela IA segue a forma clássica do cálculo de potenciação, simplesmente multiplicando a base n um total de k vezes. Dessa forma, sua complexidade é O(k), o que o torna ineficiente e muito lento para valores elevados de k.

Por outro lado, o segundo código iterativo utiliza um método que divide o problema em dois subproblemas menores. Assim, sua complexidade passa a ser O(k/2), tornando-o mais eficiente que o anterior. Ainda assim, o programa permanece lento para valores muito altos de k.

```
1       // Calculo de n^k mod 1000 de forma iterativa - IA
2       for (i = 0; i < k; i++) {
3           resultado = (resultado * n) % 1000;
4       }
5
6       //Dividir para conquistar - Iterativo
7       if (k % 2 == 0){
8           for (int i = 0; i < k / 2; i++){
9               resultado = (resultado * n) % 1000;
10          }
11      }
12      else{
13          for (int i = 0; i < k / 2; i++){
14              resultado = (resultado * n) % 1000;
15          }
16      }
```

Apesar disso, ambos os códigos adotam a mesma estratégia para lidar com o estouro de variável, mantendo apenas os três últimos algarismos do resultado final por meio do uso do módulo 1000.

4 Recursiva x Iterativa x Iterativa IA

Dentre todos os modelos para realizar o cálculo da potência, o mais eficiente é o recursivo, pois nele utilizamos de fato o método "dividir para conquistar". Dessa forma, sua complexidade é O(log(k)), permitindo que mesmo para valores elevados de k o cálculo seja realizado rapidamente.

Em comparação com as outras versões, o código gerado pela IA possui complexidade O(k), o iterativo O(k/2) e o recursivo O(log(k)). Portanto, o método recursivo é o mais eficiente tanto em tempo de execução quanto em uso de memória.

```c
#include <stdio.h>
#include <math.h>

int Potenciacao(int n, int k);

int main(){
    int n, k, resultado;

    //Le os valores de n e k dentro da faixa
    scanf(" %d %d", &n, &k);
    if(n < 0 || n > 99 || k < 0 || k > 1000000000){
        return 0;
    }

    //Calcula a potenciacao e printa os 3 ultimos digitos
    resultado = Potenciacao(n, k);
    printf("%d", resultado);

    return 0;
}

int Potenciacao(int n, int k){

    //Usando o metodo dividir para conquistar
    if(k == 1){
        return n;
    }
    else if(k == 0){
        return 1;
    }
    int potencia = Potenciacao(n, k / 2);

    //Usa o mod de 1000 para retornar somente os 3 ultimos digitos
    if(k % 2 == 0){
        return ((potencia * potencia) % 1000);
    }
    else if(k % 2 == 1){
        return ((potencia * potencia * n) % 1000);
    }
}
```