SCC0220 - Laboratório Introdução à Ciência da Computação II

Relatório de execução da aula prática 0

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

Exercício 1 – Multiplicação

Item a

Comentário

Foram implementadas duas formas de multiplicação, a convencional e a multiplicação pelo Método de Karatsuba, implementado a partir do pseudo-código fornecido no exercício. Ambas passaram em todos os casos de teste da plataforma *runcodes*,testados cada um por um integrante da dupla.

Para implementar o método de multiplicação convencional, foi necessário iterar sobre cada dígito dos dois números multiplicados. Optamos por armazenar o resultado de cada multiplicação parcial em uma string auxiliar para facilitar o cálculo. A função potência de 10, previamente disponibilizada, foi aplicada para ajustar as casas decimais conforme necessário. Finalmente, atualizamos o resultado final em cada iteração, utilizando a função de soma de strings.

Para a implementação do Método de Karatsuba, foi utilizada a função “strndup”, sugerida no exercício, para a separação das strings, as funções “add”, “sub”, “potencia_de_10”, para operações matemáticas com strings, além do processo natural do Método de Karatsuba, de sucessivas separações das strings em pedaços menores para realização de contas, até que seja possível multiplicar ambas de tamanho 1, e continuar o processo de recursão. Para strings de tamanhos diferentes, foram colocados zeros à esquerda para igualar os tamanhos e auxiliar no funcionamento do algoritmo. Por fim, todos os espaços de memória alocados foram desalocados para evitar *memory leak*.

Código

Multiplicação Convencional:

```c
char* multiplicacao(const char* str1, const char* str2) {
    int tam1 = strlen(str1), tam2 = strlen(str2), mul;
    char *result = malloc(sizeof(char)*(3));
    char aux[3];

    memset (result, '0', 2 );
    result[2] = '\0';

    for (int i = tam1 - 1 ; i >= 0; i--) {
        for (int j = tam2 - 1; j >= 0; j--) {

            mul = (str1[i] - '0')*(str2[j] - '0');
            aux[0] = (mul/10) + '0';
            aux[1] = (mul%10) + '0';
            aux[2] = '\0';

            char* temp = add(result, potencia_de_10(aux, tam1 + tam2 - i - j - 2));
            free(result);
            result = temp;
        }

    }
    int i = 0;
    while (result[i] == '0' && result[i + 1] != '\0') i++;
    char *resposta = malloc(sizeof(char)*(strlen(result)-i+1));
    strcpy(resposta, result+i);
    free(result);

    return resposta;
}
// Função para multiplicar dois números grandes representados por strings
```

Método de Karatsuba:

```c
char* karatsuba(char* str1, char* str2){
    int m = max(strlen(str1), strlen(str2));
    int meio = m/2;
    // Nesses dois IFs, se alguma string for maior que a outra, o algoritmo as iguala c
    // Exemplo: "1" e "02", ficaria "01" e "02"
    if(strlen(str1)>strlen(str2)){
     for (int i = strlen(str2); i >= 0; i--) {
        str2[i + 1] = str2[i];
        }
        str2[0] = '0';
    }
    if(strlen(str2)>strlen(str1)){
     for (int i = strlen(str1); i >= 0; i--) {
        str1[i + 1] = str1[i];
        }
        str1[0] = '0';
    }
    // Se ambos os tamanhos das strings forem 1, será feita a multiplicação entre elas
    if(strlen(str1) == 1 && strlen(str2) == 1){
        int result = (str1[0] - '0') * (str2[0] - '0');
        char * result_char;
        if(result>=10){
            result_char = malloc(3 * sizeof(char));
            result_char[0] = (result/10+'0');
            result_char[1] = ((result - ((result/10)*10))+'0');
            result_char[2] = '\0';

        }
        else{
            result_char = malloc(2 * sizeof(char));
            result_char[0] = (result +'0');
            result_char[1] = '\0';
        }
        return result_char;
    }
```

Continuação método de Karatsuba:

```c
    // Separação das metades inferiores e superiores
    char* metade_superior1 = strndup(str1, meio);
    char* metade_inferior1 = strndup(str1 + meio, strlen(str1)- meio);
    char* metade_superior2 = strndup(str2, meio);
    char* metade_inferior2 = strndup(str2 + meio, strlen(str2) - meio);

    char* z0 = karatsuba(metade_inferior1, metade_inferior2);
    char* soma1 = add(metade_inferior1, metade_superior1);
    char* soma2 = add(metade_inferior2, metade_superior2);
    char* z1 = karatsuba(soma1, soma2);
    char* z2 = karatsuba(metade_superior1, metade_superior2);

    // Cálculo do resultado final
    char* z1_menos_z0_z2 = sub(z1, add(z0, z2));
    char* potencia10_z2 = potencia_de_10(z2, 2 * (m - meio));
    char* potencia10_z1_menos_z0_z2 = potencia_de_10(z1_menos_z0_z2, m - meio);
    char* resultado = add(add(potencia10_z2, potencia10_z1_menos_z0_z2), z0);
    //Liberando espaços de memória
    free(metade_inferior1);
    metade_inferior1 = NULL;
    free(metade_inferior2);
    metade_inferior2 = NULL;
    free(metade_superior2);
    metade_superior2 = NULL;
    free(metade_superior1);
    metade_superior1 = NULL;
    return resultado;
}
```

Saída

Para comprovar a eficiência dos dois métodos de multiplicação , foi utilizada a biblioteca <time.h> da linguagem C para medir o tempo de execução do algoritmo. Os dois números multiplicados, o resultado e o tempo de execução de cada algoritmo estão disponíveis abaixo.

```
872 291
Resultado Karatsuba: 253752, Tempo de Execução: 0.000036
Resultado Convencional: 253752, Tempo de Execução: 0.000010
```

Tempo de execução mais curto (3 dígitos) : Convencional

```
3221212 8823922
Resultado Karatsuba: 28423723433464, Tempo de Execução: 0.000089
Resultado Convencional: 28423723433464, Tempo de Execução: 0.000056
```

Tempo de execução mais curto (7 dígitos) : Convencional

```
19293827162 82819329182
Resultado Karatsuba: 1597901822910290841484, Tempo de Execução: 0.000160
Resultado Convencional: 1597901822910290841484, Tempo de Execução: 0.000103
```

Tempo de execução mais curto (11 dígitos) : Convencional

```
9929292939495929 9283717372839291
Resultado Karatsuba: 92180749362408867181061065746339, Tempo de Execução: 0.000373
Resultado Convencional: 92180749362408867181061065746339, Tempo de Execução: 0.000325
```

Tempo de execução mais curto (16 dígitos) : Convencional

```
39292929292939291939 83828485828293829392
Resultado Karatsuba: 3293866766385312912296859482535346871088, Tempo de Execução: 0.000451
Resultado Convencional: 3293866766385312912296859482535346871088, Tempo de Execução: 0.000502
```

Tempo de execução mais curto (20 dígitos): Karatsuba

A partir dos testes, percebe-se que o Método de Karatsuba performa melhor para números muito grandes, mas para números menores o método convencional é mais eficiente, note que para o exemplo de 20 dígitos o Karatsuba foi mais eficiente em questão de tempo. Em questão de utilização de memória, as chamadas recursivas do método Karatsuba ao passo que aumenta-se a proporcionalidade dos números, tendem a acumular mais memória do que o método convencional.