SCC0220 - Laboratório Introdução à Ciência da Computação II

Relatório do trabalho prático 1

Alunos	**NUSP**
Alec Campos Aoki	15436800
Juan Henriques Passos	15464826

Trabalho Prático 1 – Karatsuba

Multiplicação Convencional

Comentário

Na multiplicação convencional, realizamos a multiplicação caractere por caractere. Pegamos os caracteres das strings e os transformamos em um inteiro (usando -'0'); dividindo esse inteiro por 10, temos que seu quociente é o carry (valor a ser somado ao produto seguinte) e que seu resto é o valor que escreveremos na string resposta (usando o +'0'). O carry é salvo no espaço seguinte na string resposta, espaço esse que tem seu valor somado ao resultado do seu produto correspondente (é por causa dessa operação que inicializamos a string resposta com zeros usando a função *memset*). Como os dígitos de cada número são multiplicados entre si, temos que para números de N dígitos a complexibilidade dessa operação é $O(N^2)$.

Código

```
char* multiplicacao(const char* str1, const char* str2){
   int tam_str1 = strlen(str1); //strlen() já não conta o \0
   int tam_str2 = strlen(str2);

   int tamstrResposta = tam_str1 + tam_str2 + 2; //\0 + 1
    //o maior tamanho possível do produto entre dois números é a soma do tamanho desses
dois números + 1

   char *strResposta = (char *)malloc(tamstrResposta*sizeof(char));
   if(strResposta == NULL) exit(1);

   /*preenchendo strResposta com zeros*/
   memset(strResposta, '0', tamstrResposta + 1);
   strResposta[tamstrResposta-1] = '\0';
```

```
   for(int i=tam_str2-1; i>=0; i--){

       int contadorResposta = tamstrResposta-2; //-2 pois ignoramos o espaço do \0 e pq o
index começa no 0

       int carry = 0;
       int sum = 0; //"quociente"

       for(int j=tam_str1-1; j>=0; j--){
           //OBS: - '0' transforma char em int

                               int   produtoTemporario   =   (str1[j]-'0')*(str2[i]-'0')   +
(strResposta[contadorResposta]-'0');
              //+ (strResposta[contadorResposta]-'0') soma ao produto dos dois dígitos o
carry do produto anterior, armazenado, durante a iteração anterior, no próximo espaço a
ser utilizado no vetor Resposta

           sum = produtoTemporario%10;
           strResposta[contadorResposta] = sum+'0';
           //exemplo: 3*6 = 18
           // 18/10 -> carry = 1, sum = 8
           // sum será o dígito armazenado no vetor Resposta

                carry  =  produtoTemporario/10;  //como  é  um  int,  automaticamente  pega  o
quociente

           if(strResposta[contadorResposta-1] == '9' && carry == 1){
               strResposta[contadorResposta-1] = '0';
                 strResposta[contadorResposta-2] = (strResposta[contadorResposta-2]-'0' +
1)+'0';
           }
           else{
               strResposta[contadorResposta-1] = ((strResposta[contadorResposta-1]-'0') +
carry)+'0'; //guarda o carry no próximo espaço do vetor Resposta
           }
           contadorResposta--;
       }
       tamstrResposta--;
```

```
    }

    return strResposta;
}
```

📤 Saída

```
1234567891011
1110987654321

1371589685334334871208531
0.000020ms
```

Karatsuba

📝 Comentário

Aplicamos o algoritmo Karatsuba recursivamente, seguindo sua implementação matemática exata. Utilizamos a multiplicação convencional no caso base, quando ao menos um dos números tinha somente um dígito. A complexibilidade desse algoritmo é aproximadamente $O(N^{1.58})$ (para números de N dígitos), no entanto, observamos que seu tempo de execução é consideravelmente maior que a multiplicação convencional.

Isso se deve ao fato de que detalhes como a implementação não são levados em conta na anáise teórica de algoritmos (como é o caso da notação *big O*). Como observado pelo código da função karatsuba, ela é muito mais complexa (e utiliza muitas mais funções e operações secundárias, além de ser recursiva) que a função multiplicação, o que aumenta seu tempo de execução consideravelmente. Isso no mostra que o resultado de uma análise teórica de um algoritmo pode ser muito diferente de sua aplicação prática na realidade.

📝 Código

```
char *karatsuba(char *str1, char *str2){
    /*Caso base*/
    if (strlen(str1) <= 1 || strlen(str2) <= 1){
        return multiplicacao(str1, str2);
    }

    if(strlen(str1) > strlen(str2)){
        int tam = strlen(str1);
        char* tmp2 = calloc(tam + 1, sizeof(char));
        memset(tmp2, '0', tam);
        strcpy(tmp2 + (tam - strlen(str2)), str2);
        str2 = tmp2;
```

```
    }
    else if(strlen(str1) < strlen(str2)){
        int tam = strlen(str2);
        char* tmp1 = calloc(tam + 1, sizeof(char));
        memset(tmp1, '0', tam);
        strcpy(tmp1 + (tam - strlen(str1)), str1);
        str1 = tmp1;
    }

    /*Separando os números no meio*/
    int maiorTamanho = strlen(str1);
    int metade = (maiorTamanho+1) / 2; // conversão automática
     //fazemos maiorTamanho+1 para que a parte superior do número seja sempre maior que a
inferior

    char *mSuperior_str1 = (char *)malloc((metade + 1) * sizeof(char)); //+1 por causa do \0
    if(mSuperior_str1 == NULL) exit(1);
    mSuperior_str1[metade] = '\0';

    char *mInferior_str1 = (char *)malloc((maiorTamanho - metade + 1) * sizeof(char));
    if(mInferior_str1 == NULL) exit(1);
    mInferior_str1[maiorTamanho - metade] = '\0';

    char *mSuperior_str2 = (char *)malloc((metade + 1) * sizeof(char)); //+1 por causa do \0
    if(mSuperior_str2 == NULL) exit(1);
    mSuperior_str2[metade] = '\0';

    char *mInferior_str2 = (char *)malloc((maiorTamanho - metade + 1) * sizeof(char));
    if(mInferior_str1 == NULL) exit(1);
    mInferior_str2[maiorTamanho - metade] = '\0';

    /*Atribuição dos números aos vetores criados*/
    for (int i = 0; i < metade; i++){
        mSuperior_str1[i] = str1[i];
        mSuperior_str2[i] = str2[i];
    }
    int contador = 0;
    for (int i = metade; i < maiorTamanho; i++){
```

```
        mInferior_str1[contador] = str1[i];
        mInferior_str2[contador] = str2[i];
        contador++;
    }

    /*Karatsuba em si*/
    char *ac = karatsuba(mSuperior_str1, mSuperior_str2);
    char *bd = karatsuba(mInferior_str1, mInferior_str2);
            char    *ad_plus_bc    =    sub(sub(karatsuba(add(mSuperior_str1,    mInferior_str1),
add(mSuperior_str2, mInferior_str2)), ac), bd);

     metade = maiorTamanho/2;   //atualizamos o valor de metade pois agora queremos saber seu
valor exato

    char *termo1 = potencia_de_10(ac, metade * 2);
    char *termo2 = potencia_de_10(ad_plus_bc, metade);
    char *termo3 = bd;

    free(mSuperior_str1);
    free(mSuperior_str2);
    free(mInferior_str1);
    free(mInferior_str2);

    return (add(add(termo1, termo2), termo3));
}
```

Saída

```
1234567891011
1110987654321

137158968533433487120853 1
0.000375ms
```