Universidade Federal do ABC
MCTA028-15 - Programação Estruturada
2018.Q3

Projeto Googol – Parte 1

Professores Emílio Francesquini e Carla Negri Lintzmayer

16 de outubro de 2018

1 O problema

Existem muitas versões para a origem do nome Google, a famosa empresa norte-americana que se destaca, entre outras coisas, pelos seus serviços de busca na Web. Uma das explicações mais aceitas vem de fontes ligadas à Universidade de Stanford, local onde tal empresa foi criada. Essa versão da história conta que o nome pretendido era na verdade "Googol" e um dos seus fundadores cometeu um erro de grafia e acabou escrevendo "Google". Googol é o nome dado ao número 10^{100}. A ideia dos fundadores era escolher um nome que refletisse a imensa quantidade de informação que precisaria ser processada para indexar as páginas da Internet.

Por outro lado, podiam muito bem ter sido os zimbabuanos a inventar o nome Google. Existe uma lenda na Internet que diz que a inflação anual estimada em 2008 teria sido por volta de $6,5 \times 10^{108}\%$ ou 650 milhões de googols (googois?). Essa lenda surgiu do equívoco de alguém ter usado a inflação estimada de janeiro a novembro (79.600.000.000%) como sendo a inflação apenas do mês de novembro e partir daí extrapolando a estimativa anual[1]. De um jeito ou de outro, seria muito legal ter uma nota de Cem Trilhões de Dólares (ainda que sejam dólares zimbabuanos).

Figura 1: Nota de Z$100.000.000.000.000.

[1] https://en.wikipedia.org/wiki/Hyperinflation_in_Zimbabwe

Para se ter uma ideia do quão absurdamente grande é um googol, veja a sua comparação com outros números enormes[2]:

- Desde que ocorreu o Big Bang, "só" se passaram 17×10^{39} de ioctossegundos (1 ioctossegundo = 10^{-24} segundo).
- Juntas, todas as pessoas do mundo viveram 5×10^{11} anos, ou 17×10^{18} segundos, ou "apenas" 17×10^{42} ioctossegundos.
- A massa do universo observável é estimada entre 10^{50} e 10^{60} Kg.
- Um Googol é aproximadamente igual a 70! (70 fatorial).

Tendo em vista este tipo de problema, é comum em sistemas computacionais precisarmos de estruturas de dados que comportem números gigantescos. A maneira mais comum é guardar estes números em variáveis do tipo *float* ou *double* que, no final das contas (sem trocadilhos), armazenam uma aproximação do número em notação científica. Isso funciona em grande parte dos casos. Contudo, em alguns ramos da computação, como a criptografia, é preciso calcular e manipular números extremamente grandes (da ordem de 10^{300} ou mais) precisamente, sem aproximações. Comumente, variáveis do tipo inteiro em C vão apenas até 64 bits (ou $2^{64} \sim= 1,8 \times 10^{19}$) e alguns compiladores mais recentes têm suporte a inteiros de até 128 bits ($2^{128} \sim= 3,4 \times 10^{38}$) mesmo em máquinas 64 bits.

Neste projeto queremos ir além. Nós vamos desenvolver uma estrutura de dados cuja única limitação para o tamanho do número a ser armazenado seja a quantidade de memória do computador. Ao final do quadrimestre nosso projeto será capaz de lidar com números inteiros, positivos e negativos, sobre os quais efetuaremos as 4 operações aritméticas básicas ($+$, $-$, $\times$, $\div$) além de calcular o resto da divisão. Nessa primeira parte do projeto, contudo, vamos focar apenas nas operações de **soma e subtração**.

2 Organização do código

Você pode estruturar o seu programa da maneira que achar mais conveniente para cumprir a tarefa dada. Contudo, aqui estão algumas sugestões que podem ajudar.

Para facilitar nossa discussão, vamos carinhosamente chamar um número a ser armazenado e manipulado por nós de "numerão".

2.1 Representação

Existem maneiras muito elaboradas e eficientes para representar numerões em computadores. Uma das mais empregadas (senão a mais) chama-se *complemento de dois*. Apesar da sua eficiência, sugerimos que neste trabalho

[2]Exemplos "emprestados" da Wikipedia: https://pt.wikipedia.org/wiki/Googol.

você fique longe desta técnica e aborde o problema de uma maneira muito mais simples.

Nossa sugestão é que você armazene os dígitos do numerão em um vetor, por exemplo, de inteiros. Assim, o númerão 12345 poderia ser representado pelo vetor $[5, 4, 3, 2, 1]$. Note que sugerimos que no vetor a ordem dos dígitos seja dada dos dígitos menos significativos para os mais significativos. Isso facilita na hora de fazer a soma, por exemplo, pois o algoritmo da soma tradicional (papel e lápis) trabalha dos dígitos menos significativos para os mais significativos. Por outro lado, isso força você escrever um código que varre o vetor na ordem inversa à tradicional para imprimir os numerões (pois a ordem dos dígitos será a inversa da tradicional) ou, na segunda parte deste projeto, para fazer a divisão (já que o algoritmo trabalha com os dígitos mais significativos do dividendo primeiro). Vá pensando nesse caso!

Também será necessário escolher uma maneira de representar números negativos e positivos. Duas maneiras simples de fazê-lo são:

- Fazer o sinal do numerão ser o mesmo do último dígito. Por exemplo: $-310 \rightarrow [0, 1, -3]$, $247 \rightarrow [7, 4, 2]$.
- Reservar uma posição no vetor para indicar o seu sinal, por exemplo a posição 0. Caso contenha 0 o número é positivo, caso contenha 1 é negativo. Exemplo: $-310 \rightarrow [1, 0, 1, 3]$, $247 \rightarrow [0, 7, 4, 2]$.

Sugerimos também que você crie funções para criar um numerão a partir de um inteiro (`int`) ou a partir de uma string (`char []`) e para imprimir o numerão armazenado. Isso facilitará bastante os seus testes.

Assim, supondo que você vá armazenar o numerão em um vetor de inteiros, seu programa pode ter as seguintes funções:

```
1 /* Função que recebe um inteiro n e o armazena em num, retornando a
  ↪  quantidade de dígitos: */
2 int criaNumeraoDeInt(int n, int num[]);
3
4 /* Função que recebe uma string str que contém um número e armazena
  ↪  os dígitos dele em num, retornando a quantidade de dígitos: */
5 int criaNumeraoDeString(char str[], int num[]);
6
7 /* Função que recebe um numerão com tamNum dígitos e o imprime: */
8 void imprimeNumerao(int num[], int tamNum);
```

2.2 Operações

Será necessário criar uma função para cada operação (nesta primeira fase, soma e subtração) que será realizada sobre os numerões. Recomendamos (porém a escolha é sua) que você faça funções que alterem um dos dois

numerões envolvidos na operação ao invés de ter um novo numerão apenas para resultado da operação. Neste caso você provavelmente também vai querer criar uma função de cópia entre numerões.

```c
/* Soma 'a' e 'b' e guarda o resultado (a+b) em 'a'
   'b' não é modificado
   retorna a quantidade de dígitos do resultado */
int soma(int a[], int tamA, int b[], int tamB);

/* Subtrai 'b' de 'a' e guarda o resultado (a-b) em 'a'
   'b' não é modificado
   retorna a quantidade de dígitos do resultado */
int subtrai(int a[], int tamA, int b[], int tamB);

/* Copia 'b' para 'a'
   'b' não é modificado
   retorna a quantidade de dígitos copiados */
int copiaNumerao(int a[], int b[], int tamB);
```

Exemplo de uso:

```c
/* criando dois numeroes baseados em inteiros: */
int a[TAM_MAX], b[TAM_MAX], tamA, tamB;
tamA = criaNumeraoDeInt(4, a);
tamB = criaNumeraoDeInt(3, b);

/* somando os números e imprimindo o resultado, que está no próprio
↪  'a': */
tamA = soma(a, tamA, b, tamB);
imprimeNumerao(a, tamA); /* imprime 7 */

int c[TAM_MAX], tamC;
tamC = copiaNumerao(c, a, tamA);
tamC = subtrai(c, tamC, b, tamB);
imprimeNumerao(a, tamA); /* imprime 7 */
imprimeNumerao(c, tamC); /* imprime 4 */
```

Para a implementação das operações propriamente ditas, utilize os algoritmos de soma e subtração que você está acostumado (papel e lápis). Eles são surpreendentemente eficientes, mesmo para números extremamente grandes.

Repare que existem algumas dependências entre as operações. Por exemplo, para implementar a multiplicação você vai precisar da soma. Para implementar a divisão você vai precisar de subtração. Por isso, nessa primeira parte do projeto o foco será apenas na soma e na subtração de numerões. É, contudo, essencial que você teste à exaustão o seu código para assegurar-se

que ele se comporta como deveria. Descobrir que a sua função de subtração não funciona corretamente na segunda parte do projeto ao mesmo tempo que você está quebrando a cabeça para entender o porquê da sua divisão não estar dando o resultado esperado será muito mais difícil.

Sugestão para o desenvolvimento:

- Comece trabalhando apenas com números positivos.
- Implemente a operação de soma e teste-a à exaustão!
- Faça a subtração. Primeiramente trabalhe apenas os casos de $a - b$ onde $a > b$. Em seguida você vai ter que adicionar o suporte aos números negativos. Teste à exaustão!
- Faça testes exaustivos na soma com números negativos e positivos. Perceba que:
  - $a + b$ com $a \geq 0$ e $b < 0$ é igual a $a - |b|$.
  - $a - b$ com $b < 0$ é igual a $a + |b|$.
  - $a + b$ com $a < 0$ e $b < 0$ é igual a $-(|a| + |b|)$.
  - ...
- Note que utilizando essas transformações é possível utilizar o código que você já tinha feito apenas para números positivos para tratar o caso de números negativos também!
- Coloque logo no início da sua função de subtração algo do tipo:

```
1  int subtrai(int a[], int tamA, int b[], int tamB) {
2      if (b é negativo) {
3          /* b = -c, a-b = a - (-c) = a + c */
4          muda o sinal de b para positivo;
5          tam = soma(a, tamA, b, tamB);
6          para manter b inalterado, volta o sinal para negativo;
7          return tam;
8      }
9      ...
10 }
```

- Note que na função acima, caso `a` também seja negativo, a função `soma` pode aplicar novamente uma das transformações que discutimos e chamar a função `subtrai` de volta para fazer a conta $b - |a|$, ou seja, altere os parâmetros para eles "caberem" nas suas implementações que só funcionavam com números positivos.

3 Entradas e saídas

O seu programa deve ler os dados da entrada padrão e escrever os resultados na saída padrão. Para isto você pode usar as funções `scanf` e `printf`.

- A entrada será dada por uma expressão numérica dada em formato pós-fixo. Exemplos: "5 3 +" , "7 3 -" .
- Cada um dos elementos da expressão estará em uma linha separada.
- Cada linha de entrada trará um número (tamanho arbitrário), uma operação (+, −) ou a palavra "FIM". Por conveniência vamos limitar os números a 1000 dígitos (ou seja, 1001 caracteres no máximo quando incluirmos números negativos).
- Note que operações feitas com dois números de 1000 dígitos podem resultar em números com mais de 1000 dígitos. O limite de 1000 dígitos é apenas válido para os números dados como entrada. Não há limite quanto ao tamanho dos números obtidos como resultado de operações.
- Você pode assumir que a entrada é sempre uma expressão válida.
- Seu programa deve parar a execução quando receber a entrada "FIM".
- Toda vez que um operando for lido, a operação referente deve ser aplicada aos dois elementos especificados nas duas linhas anteriores do arquivo de entrada. O resultado da operação deve ser então impresso na tela.

Nos exemplos a seguir, entrada e saída aparecem intercaladas. O texto em vermelho indica a saída esperada.

3.1 Exemplo 1

```
3
5
+
8
8
2
-
6
12157865549787498543194363453256756324532235218789765476
3
-
12157865549787498543194363453256756324532235218789765473
3245899634153843978545123478312549796317362134
```

```
9831266328646530372529734097691549478543524556538421 78
+
9831266361105526714068173883142784261669022519712043 12
9831266361105526714068173883142784261669022519712043 12
-1
-
9831266361105526714068173883142784261669022519712043 13
FIM
```

3.2 Exemplo 2

```
-123582
-2
-
-123580
FIM
```

3.3 Implementação de referência

Uma implementação de referência está disponível (apenas o binário :p). Você pode baixá-la juntamente com os testes abertos nos links abaixo:

- Implementação de referência
  - Versão Linux 64.
  - Versão Windows (32 bits).
- Testes abertos.

4 Entrega e avaliação

Este projeto poderá ser feito em grupos de até três pessoas. A entrega deverá ser feita pelo Moodle **apenas** por um dos integrantes do grupo. O arquivo enviado deve conter um cabeçalho contendo o nome completo e RA de cada um. Verifique a página da disciplina para informações sobre as datas de entrega e acesso ao Moodle.

Todos os trabalhos entregues passarão por uma correção automática no Moodle. A nota será proporcional ao número de testes que passarem com sucesso. Contudo, o monitor e os professores poderão, ao seu próprio critério, alterar as notas para cima ou para baixo. Isto é, passar na metade do testes não significa que você tirou nota 5 automaticamente: indica apenas a provável nota que será atribuída ao seu trabalho.

Lembre-se: Plágios serão severamente punidos com a reprovação na disciplina.

4.1 Política de atrasos

Trabalhos entregues com atraso serão aceitos, porém sofrerão uma redução na sua nota máxima conforme tabela abaixo:

Dias em Atraso	**Nota Máxima**
0	10
1	7
2	6
3	5
>3	0

Bom trabalho!