SCC-210
Algoritmos Avançados

Capítulo 1
Introdução e E/S

João Luís G. Rosa

International Collegiate Programming Competition (ICPC)

- ICPC no mundo:
  - Existe desde a década de 1970;
  - Realizada em todo o mundo pela ACM (*Association for Computing Machinery*);
  - No ano de 2009 contou com a participação de mais de 22.000 competidores de 2.000 escolas de 82 países.
- ICPC no Brasil:
  - Maratona de Programação desde 1996;
  - Realizada pela SBC desde o ano 2000;
  - Apoio do CNPq de 2002 a 2004;
  - Patrocínio da Fundação Carlos Chagas desde 2005.

Maratona de Programação

- Regras
  - Times de três estudantes com até cinco anos de estudos universitários;
  - Cada time deve ter um nome. Fica a critério do time escolhê-lo;
  - De 6 a 10 problemas computacionais para serem resolvidos durante 5 horas de competição;
  - Quando um time considera que resolveu um problema, submete aos juízes que, *online,* dizem se a solução está ou não correta;
  - Uma solução correta resolve um conjunto de testes dos juízes, desconhecido dos alunos.

Maratona de Programação

- Formato:
  - Apenas um computador é alocado para o time inteiro;
  - Todos os times iniciam a competição com penalidade de tempo igual a zero;
  - O time que resolver mais problemas, com menor tempo total como critério de desempate, é o vencedor.

Maratona de Programação

- Formato:
  - Apenas é permitido acesso a materiais impressos durante o concurso.
  - Para cada problema o time deve implementar uma solução para resolvê-lo e submeter o código fonte aos juízes para avaliação.
  - As implementações devem ser em C, C++ ou Java (a critério do time).
  - Cada solução aceita pelos juízes é premiada com um balão colorido.

Formato

Maratona de Programação

- Formato:
  - A cada solução aceita, o tempo decorrido desde o início da competição até a submissão da solução correta é somado ao tempo total do time;
  - A cada solução rejeitada, uma penalidade de 20 minutos é somada ao tempo total do problema;
  - Quando um problema for solucionado pelo time, todas as penalidades acumuladas associadas ao problema também serão somadas ao tempo total do time.

7

Maratona de Programação

- Cada problema contém:
  - Informações para contextualização (*background*);
  - O enunciado do problema;
  - Informações sobre a entrada (*Input*);
  - Informações sobre a saída (*Output*);
  - Exemplo de entrada (*Sample Input*);
  - Exemplo de saída (*Sample Output*).

Maratona de Programação

- Os juízes podem dar uma das seguintes respostas a uma solução submetida por um time:
  - Yes;
  - No – Wrong Answer (Incorrect Output);
  - No – Presentation Error (Output Format Error);
  - No – Time Limit Exceeded;
  - No – Runtime Error;
  - No – Compile Error.

Maratona de Programação

- O time não ganhará mais pontos por problemas 90% resolvidos, por elegância na implementação do algoritmo, ou por algoritmos extremamente eficientes.

- *"The fastest programmers, as opposed to the fastest programs, win"* (Programming Challenges).

The 3n+1 problem

PC/Uva IDs: 110101/100, Popularity: A, Sucess rate: low, Level: 1

Consider the following algorithm to generate a sequence of numbers. Start with an integer *n*. If *n* is even, divide by 2. If *n* is odd, multiply by 3 and add 1. Repeat this process with the new value of *n*, terminating when *n* = 1. For example, the following sequence of numbers will be generated for *n* = 22:

22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1

It is *conjectured* (but not yet proven) that this algorithm will terminate at *n* = 1 for every integer *n*. Still, the conjecture holds for all integers up to at least 1,000,000. For an input *n*, the *cycle-length* of *n* is the number of numbers generated up to and *including* the 1. In the example above, the cycle length of 22 is 16. Given any two numbers *i* and *j*, you are to determine the maximum cycle length over all numbers between *i* and *j*, *including* both endpoints.

The 3n+1 problem

Input

The input will consist of a series of pairs of integers *i* and *j*, one pair of integers per line. All integers will be less than 1,000,000 and greater than 0.

Output

For each pair of input integers *i* and *j*, output *i*, *j* in the same order in which they appeared in the input and then the maximum cycle length for integers between and including *i* and *j*. These three numbers should be separated by one space, with all three numbers on one line and with one line of output for each line of input.

The 3n+1 problem

Sample Input

1 10
100 200
201 210
900 1000

Sample Output

1 10 20
100 200 125
201 210 89
900 1000 174

Entrada e Saída

Principais Funções

- **#include<stdio.h>**
  - **printf** - impressão formatada em **stdout**;
  - **sprintf** - impressão formata em strings;
  - **gets** - leitura de strings de **stdin** (depreciado);
  - **fgets** - leitura de strings de **streams**;
  - **scanf** - leitura formatada de **stdin**;
  - **sscanf** - leitura formatada de strings;
  - **getchar** - leitura de caractere de **stdin**.

Função **printf**

- **int printf(const char * format,...);**
  - É uma função, retorna o número de caracteres impressos ou **EOF** na ocorrência de erro;
  - **Especificador de formato**: %[flags][width][.precision][length]specifier
    - **Specifier**: c, d, f, o (octal), s, u (decimal sem sinal), x ou X (hexadecimal);
    - **Length**: h (short), l (long int) e L (long double);
    - **Precision**: número de casas decimais;
    - **Width**: número mínimo de caracteres a serem impressos;
    - **Flags**: -,+, espaço, #, 0.

Exemplo: **printf**

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

int main()
{
    printf("Caracteres: %c %c, ASCII: %d\n", 'a', 65, 'Z');
    printf("Decimais: %d %ld\n", 1977, 650000);
    printf("Precedido com brancos: %10d \n", 1977);
    printf("Precedido com zeros: %010d \n", 1977);
    printf("Em decimal: %d, octal: %o e hexadecimal %#x\n", 100, 100, 100);
    printf("Ponto flutuante: %4.2f %+.0e %E\n", 3.1416, 3.1416, 3.1416);
    printf("Truque com a largura: %*d\n", 5, 10);
    printf("%s\n", "Uma string");
    system("pause");
    return 0;
}
```

Saídas

```
Caracteres: a A, ASCII: 90
Decimais: 1977 650000
Precedido com brancos:       1977
Precedido com zeros: 0000001977
Em decimal: 100, octal: 144 e hexadecimal 0x64
Ponto flutuante: 3.14 +3e+000 3.141600E+000
Truque com a largura:    10
Uma string
```

Função **gets**

- **`char * gets ( char * str );`**
  - Depreciado por não permitir especificar o tamanho da string;
  - Realiza a leitura de caracteres até encontrar um caractere de nova linha (**‘\n’**) ou fim de arquivo;
  - Remove o caractere **‘\n’** de **stdin**, mas não o coloca em **str**;
  - Insere o caractere **‘\0’** no final de **str**.

Função **fgets**

- **char * fgets (char * str, int num, FILE * stream);**
  - Realiza a leitura de caracteres até **num**-1 caracteres ou encontrar um caractere de nova linha ou fim de arquivo;
  - O caractere **‘\n’** é considerado válido e é inserido em **str** (permite identificar se ainda há algo no buffer de entrada - diferente de **gets**!);
  - Insere o caractere **‘\0’** no final de **str**.

Função fgets

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

int main() {
    char str[20];

    gets(str);
    printf("-%s-\n", str);
    fgets(str, 20, stdin);
    printf("-%s-\n", str);
    system("pause");
    return 0;
}
```

Retorno: **`gets`** e **`fgets`**

- Para ambos **`gets`** e **`fgets`**:
  - Em caso de sucesso, as funções retornam o parâmetro **`str`**;
  - Se o caractere de fim de arquivo é encontrado e nenhum caractere foi lido, então um ponteiro **`NULL`** é retornado;
  - Se um erro é encontrado **`NULL`** é retornado;
  - **`ferror()`** e **`feof()`** podem ser utilizadas para diferenciar entre erros e fim de arquivo.

Função **scanf**

- **`int scanf(const char * format, ...);`**
- ***`format`*** pode conter:
  - **Especificador de formato**: %[*][width] [modifiers]type
    - **type**: c, d, f, o (octal), s, u, x, X (hexa), n (nr. de valores lidos);
    - **modifiers**: h (short), l (long) e L (long double);
    - **width**: especifica o número máximo de caracteres;
    - *: faz com que os dados sejam lidos de stdin, mas ignorados

Função **scanf**

- ***format*** pode conter:
  - **Caracteres em branco**: casa com zero ou mais caracteres brancos (‘ ‘, ‘\n’ e ‘\t’);
  - **Caracteres diferente de branco, exceto ‘%’**: faz com que esses caracteres, se casarem com a entrada sejam ignorados. Se não casarem com a entrada **scanf** falha e retorna deixando demais caracteres em stdin;

Exemplo 1: **scanf**

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

int main() {
    int n, m;
    char s[10];

    scanf("%*d %d", &n);
    printf("Valor lido: %d\n", n);

    scanf("%7d%s", &n, s);
    printf("Valor lido: %d, %s\n", n, s);

    scanf("%d %d", &n, &m);
    printf("Valor lido: %d, %d\n", n, m);

    scanf("%d.%d", &n, &m);
    printf("Parte inteira %d, parte fracionaria %d\n", n, m);

    system("pause");
    return 0;
}
```

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

int main() {
    int n, m;
    char s[10];
    char c;

    scanf("%d", &n);
    scanf("%c", &c);
    printf("Valor lido: %d, %c\n", n, c);

    scanf("%s", &s);
    c = getchar();
    printf("Valor lido: %s, %c\n", s, c);

    scanf("%d\n", &n);
    scanf("%c", &c);
    printf("Valor lido: %d, %c\n", n, c);

    scanf("%s ", &s);
    c = getchar();
    printf("Valor lido: %s, %c\n", s, c);

    system("pause");
    return 0;
}
```

Função **scanf**

- A máscara %[] realiza a leitura de um string, cujos caracteres válidos são especificados entre os colchetes:
  - scanf(“%[abc]”, s); // aabbccabc válido
- O Símbolo ^ indica o conjunto complementar (qualquer caractere menos os presentes na lista):
  - scanf(“%[^abc], s); // defghijklmn... válido

Função **scanf**

- Uma coisa importante sobre o scanf é o parâmetro de retorno:
  - Em caso de sucesso, mesmo que parcial, scanf retorna o número de itens lidos com sucesso.
    - Esse número pode ser um valor menor ou igual ao número de leituras esperado.
  - Em caso de falha antes de que qualquer dado seja lido com sucesso, a constante EOF é retornada.

scanf("%s") vs. gets(fgets)

- **scanf("%s")** opera de forma diferente do **gets(fgets)**:
  - Para o **scanf**, **"%s"** significa uma seqüência de caracteres diferente dos caracteres brancos. Portanto um **scanf("%s")** pode ler somente uma palavra de uma frase;
  - No **gets** e **fgets**, a linha toda é lida.

Cuidado: `fflush(stdin)`

- **Cuidado com `fflush(stdin)`, pois não funciona em todos os compiladores!**
  - **“`fflush` is defined only for output streams. Since its definition of "flush" is to complete the writing of buffered characters (not to discard them), discarding unread input would not be an analogous meaning for `fflush` on input streams.”**

Exemplo 1: The 3n+1 Problem

Sample input

```
1 10
100 200
201 210
900 1000
```

```
int main() {
    int i, int f;

    while (scanf("%d %d", &i, &j) != EOF) {
        // processa o caso de teste
    }
}

int main() {
    int i, int f;

    while (scanf("%d %d", &i, &j) == 2) {
        // processa o caso de teste
    }
}
```

Exemplo 2: Minesweeper

Sample input

```
4 4
*...
....
.*..
....
3 5
**...
.....
.*...
0 0
```

```
int main() {
     char mx[102][102];
     int m, n, l, c;

     while (1) {
          scanf("%d %d", &n, &m);
          if (m == 0 && n == 0)
               break;
          for (l=1; l<=n; l++)
             for(c=1; c<=m; c++)
                  scanf(" %c", &mx[l][c]);
          // processa o caso de teste
     }
}
```

Exemplo 3: The Trip

Sample input

```
3
10.00
20.00
30.00
4
15.00
15.01
3.00
3.01
0
```

```
int main() {
    float alunos[1000];
    int n, i;

    while (1) {
        scanf("%d", &n);
        if (n == 0)
            break;
        for (i=0; i<n; i++)
            scanf("%f", &alunos[i]);
        // processa o caso de teste
    }
}
```

Exemplo 4: Crypt Kicker

Sample input

```
4
and
jane
puff
spot
xsb qymm xsb rquat
xxx yyyy zzz w
```

```
scanf("%d\n", &dict_size);
for (i = 0; i < dict_size; i++)
    scanf("%s\n", words[i]);
while (gets(line)) {
      line_words_last = 0;
      pos = 0;
      while (sscanf(&line[pos], "%s%n",
            line_words[line_words_last++], &inc) != EOF) {
                  pos += inc;
      }
      //processa o caso de teste
}
```

Referências

- Batista, G. & Campello, R.
  - Slides disciplina *Algoritmos Avançados*, ICMC-USP, 2007.
- Skiena, S. S. & Revilla, M. A.
  - *Programming Challenges – The Programming Contest Training Manual*. Springer, 2003.