SCC-210 – Algoritmos Avançados

Introdução e E/S

João Batista

International Collegiate Programming Competition (ICPC)

- ICPC no mundo:
  - Existe desde 1970;
  - Realizada em todo o mundo pela ACM (*Association for Computing Machinery*);
  - No ano de 2012 contou com a participação de mais de 30.000 competidores de 2.200 universidades de 85 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 IBM.

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.

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

Algoritmos Avançados

Entrada e Saída

João Batista

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), e ou E (notação científica), p (ponteiro), g ou G (mais curto entre f ou e);
  - **Length**: h (short), l (long), ll (long long) e L (long double);
  - **Precision**: número de casas decimais (número ou *) ;
  - **Width**: número mínimo de caracteres a serem impressos (número ou *);
  - **Flags**: - (justificado à esquerda) ,+ (força + para positivos), espaço (um espaço é impresso para números sem sinal), # (vários usos como forçar a impressão do ponto decimal, 0x para hexa e 0 para octal), 0 (força 0 à esquerda).

Exemplo: printf

```c
/* printf example */
#include <stdio.h>

int main()
{
   printf ("Characters: %c %c \n", 'a', 65);
   printf ("Decimals: %d %ld\n", 1977, 650000L);
   printf ("Preceding with blanks: %10d \n", 1977);
   printf ("Preceding with zeros: %010d \n", 1977);
   printf ("Some different radixes: %d %x %o %#x %#o \n", 100, 100, 100, 100, 100);
   printf ("floats: %4.2f %+.0e %E \n", 3.1416, 3.1416, 3.1416);
   printf ("Width trick: %*d \n", 5, 10);
   printf ("%s \n", "A string");
   return 0;
}
```

Exemplo: printf

```c
/* printf example */
#include <stdio.h>

int main()
{
   printf ("Characters: %c %c \n", 'a', 65);
   printf ("Decimals: %d %ld\n", 1977, 650000L);
   printf ("Preceding with blanks: %10d \n", 1977);
   printf ("Preceding with zeros: %010d \n", 1977);
   printf ("Some different radixes: %d %x %o %#x %#o \n", 100, 100, 100, 100, 100);
   printf ("floats: %4.2f %+.0e %E \n", 3.1416, 3.1416, 3.1416);
   printf ("Width trick: %*d \n", 5, 10);
   printf ("%s \n", "A string");
   return 0;
}
```

```
Characters: a A
Decimals: 1977 650000
Preceding with blanks:       1977
Preceding with zeros: 0000001977
Some different radixes: 100 64 144 0x64 0144
floats: 3.14 +3e+000 3.141600E+000
Width trick:    10
A string
```

Função gets

char * gets (char * str);

- Realiza a leitura de caracteres até encontrar um caractere de nova linha (“\n”) ou fim de arquivo;
  - Depreciado por não permitir especificar o tamanho da string;
  - 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!);
  - Remove o caractere “\n” de stdin;
  - 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), (ll long 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 diferentes 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") versus gets (fgets)

- scanf("%s") opera de forma diferente do gets (fgets):
  - Para o scanf, "%s" significa uma sequência 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
```

```c
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

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

```
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
}
```

Minesweeper

PC/Uva IDs: 110102/10189, Popularity: A, Sucess rate: high, Level: 1

Have you ever played Minesweeper? This cute little game comes with a certain operating system whose name we can't remember. The goal of the game is to find where all the mines are located within a *M* x *N* field. The game shows a number in a square which tells you how many mines there are adjacent to that square. Each square has at most eight adjacent squares. The 4 x 4 field on the left contains two mines, each represented by a “*'' character. If we represent the same field by the hint numbers described above, we end up with the field on the right:

```
*...      *100
....      2210
.*..      1*10
 ....      1110
```

Minesweeper

Input

The input will consist of an arbitrary number of fields. The first line of each field contains two integers n and m ($0 < n, m \leq 100$) which stand for the number of lines and columns of the field, respectively. Each of the next n lines contains exactly m characters, representing the field.
Safe squares are denoted by “.” and mine squares by “*”, both without the quotes. The first field line where $n = m = 0$ represents the end of input and should not be processed.

Output

For each field, print the message Field #x: on a line alone, where x stands for the number of the field starting from 1. The next n lines should contain the field with the “.” characters replaced by the number of mines adjacent to that square. There must be an empty line between field outputs.

Minesweeper

Sample Input

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

Sample Output

```
Field #1:
*100
2210
1*10
1110

Field #2:
**100
33200
1*100
```

Exercício Laboratório: Graphical Editor

PC/Uva IDs: 1101105/10267, Popularity: B, Sucess rate: low, Level: 1

Graphical editors such as Photoshop allow us to alter bit-mapped images in the same way that text editors allow us to modify documents. Images are represented as an M x N array of pixels, where each pixel has a given color.

Your task is to write a program which simulates a simple interactive graphical editor.

Input

The input consists of a sequence of editor commands, one per line. Each command is represented by one capital letter placed as the first character of the line. If the command needs parameters, they will be given on the same line separated by spaces.

Pixel coordinates are represented by two integers, a column number between 1...M and a row number between 1...N, where $1 \leq M, N \leq 250$. The origin sits in the upper-left corner of the table. Colors are specified by capital letters.

Exercício Laboratório: Graphical Editor

- The editor accepts the following commands:

I M N	Create a new *M* x *N* image with all pixels initially colored white (O).
C	Clear the table by setting all pixels white (O). The size remains unchanged.
L X Y C	Colors the pixel (*X*, *Y*) in color (C).
V X Y1 Y2 C	Draw a vertical segment of color (C) in column *X*, between the rows *Y*1 and *Y*2 inclusive.
H X1 X2 Y C	Draw a horizontal segment of color (C) in the row *Y*, between the columns *X*1 and *X*2 inclusive.
K X1 Y1 X2 Y2 C	Draw a filled rectangle of color C, where (*X*1, *Y*1) is the upper-left and (*X*2, *Y*2) the lower right corner.
F X Y C	Fill the region *R* with the color C, where *R* is defined as follows. Pixel (*X*, *Y*) belongs to *R*. Any other pixel which is the same color as pixel (*X*, *Y*) and shares a common side with any pixel in *R* also belongs to this region.
S Name	Write the file name in MSDOS 8.3 format followed by the contents of the current image.
X	Terminate the session.

Exercício Laboratório: Graphical Editor

- **Output**

  On every command S NAME, print the filename *NAME* and contents of the current image. Each row is represented by the color contents of each pixel. See the sample output.

  Ignore the entire line of any command defined by a character other than I, C, L, V, H, K, F, S, or X, and pass on to the next command. In case of other errors, the program behavior is unpredictable.

Exercício Laboratório: Graphical Editor

Sample Input

```
I 5 6
L 2 3 A
S one.bmp
G 2 3 J
F 3 3 J
V 2 3 4 W
H 3 4 2 Z
S two.bmp
X
```

Sample Output

```
one.bmp
OOOOO
OOOOO
OAOOO
OOOOO
OOOOO
OOOOO
two.bmp
JJJJJ
JJZZJ
JWJJJ
JWJJJ
JJJJJ
JJJJJ
```