Universidade Tecnológica Federal do Paraná
Departamento Acadêmico de Informática
CSF13 – Fundamentos de Programação 1
Prof. Bogdan Tomoyuki Nassu
Profa. Leyza Baldo Dorini

(***) 1. (Adaptado da OBI 2011)

Sebastião Bueno Coelho, apelidado de SBC pelos familiares e amigos, passou as férias de janeiro de 2011 no sítio de seus avós. Durante sua estadia, uma das atividades prediletas do SBC era nadar no rio que havia no fundo da casa onde morava.

Uma das características do rio que mais impressionava SBC era um belo caminho, feito inteiramente com pedras brancas. Há muito tempo, o avô de SBC notara que os habitantes do sítio atravessavam o rio com grande frequência e, por isso, construiu um caminho no rio com pedras posicionadas em linha reta; ao fazê-lo, tomou muito cuidado para que o espaçamento das pedras fosse de exatamente um metro.

Hoje em dia, a única utilidade do caminho é servir de diversão para os sapos que vivem no rio, que pulam de uma pedra a outra agitadamente. Um certo dia, enquanto descansava e nadava nas águas, SBC assistiu atentamente às acrobacias dos bichos e notou que cada sapo sempre pulava (zero, uma ou mais vezes) uma quantidade fixa de metros.

SBC sabe que você participa da OBI todos os anos e resolveu desafiar-te com o seguinte problema: Dado o número de pedras no rio, o número de sapos, a pedra inicial sobre a qual cada sapo está (cada pedra é identificada por sua posição na sequência de pedras) e a distância que cada sapo pula, determinar as posições onde pode existir um sapo depois que SBC chega no rio.

Entrada:

O programa recebe como entradas dois inteiros *N* e *M* (ambos no intervalo [1, 100], que representam o número de pedras no rio e o número de sapos, respectivamente. São então lidos *M* pares de valores, cada par contendo a posição inicial de um sapo e a distância fixa de um pulo, ambos valores no intervalo [0, *N*]. Considere que a primeira pedra corresponde à posição 0, a segunda pedra à posição 1, etc. Desta forma, a última pedra tem a posição *N*-1.

Saída:

O programa deve mostrar *N* valores, que indicam a possibilidade ou não de ter um sapo em cada pedra. Para as pedras que podem ter um sapo você deve imprimir 1, e para as pedras que com certeza não podem ter nenhum sapo você deve imprimir 0.

Veja os exemplos na próxima página!

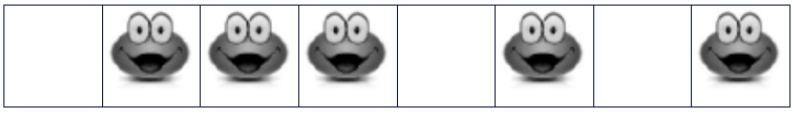
Exemplos:

Entrada	Saída
5 2 2 2 3 4	10111

Neste exemplo, SBC indicou a existência de 5 pedras no rio e 2 sapos. Os sapos estavam inicialmente nas pedras 2 e 3. SBC também lhe disse que o primeiro sapo da entrada sempre pula 2 metros, e o segundo sempre pula 4 metros. A figura a seguir ilustra as possíveis pedras que podem ser ocupadas pelos sapos quando eles começam a pular.

Entrada	Saída
8 3 2 3 1 2 5 2	01110101

Neste exemplo, SBC indicou a existência de 8 pedras no rio e 3 sapos. Os sapos estavam inicialmente nas pedras 2, 1 e 5. SBC também lhe disse que o primeiro sapo da entrada sempre pula 3 metros, o segundo e terceiro sempre pulam 2 metros. Dessa forma, o primeiro sapo pode estar nas pedras 2 ou 5; o segundo sapo pode estar nas pedras 1, 2, 4 ou 6; e o terceiro sapo pode estar nas pedras 5, 3, 1 e 7. A figura a seguir ilustra as possíveis pedras que podem ser ocupadas pelos sapos quando eles começam a pular.



(***) 2. A contagem de palavras em um texto é uma funcionalidade básica encontrada em vários *softwares* para edição de texto. Existem também *sofwares* e *websites* dedicados unicamente a esta simples tarefa!

Escreva uma função que recebe como parâmetro apenas uma *string*, e retorna o número de palavras que existem nela, supondo que uma palavra contém apenas letras – ou seja, números, espaços, pontuação e quaisquer outros caracteres são tratados como divisores entre palavras. Suponha que a *string* é terminada em `'\0'`, e não use funções da biblioteca-padrão. Os tamanhos da *string* e do *buffer* que a contém são desconhecidos a priori.

(**) 3. Escreva uma função que recebe um vetor de inteiros e um vetor de saída com a mesma capacidade. A função deve colocar nas posições iniciais do vetor de saída os elementos do vetor de entrada, mas sem repetições, e retornar o número de elementos inseridos no vetor de saída. As posições que sobrarem no vetor de saída terão conteúdo indeterminado. Os dados do vetor original devem ser mantidos intactos. Por exemplo, se os elementos do vetor original são {0, 1, 2, 3, 4, 3, 2, 4, 5, 3, 2, 6, 1, 0}, o novo vetor deve conter nas posições iniciais os valores {0, 1, 2, 3, 4, 5, 6}, com as demais posições tendo conteúdo indeterminado, e a função deve retornar 7.

(***) 4. Escreva uma função que recebe como parâmetros um vetor de inteiros e o seu tamanho, e retorna o tamanho da maior sequência não decrescente encontrada. Por exemplo, considere o vetor abaixo:

1, 5, 6, 4, 9, 13, 13, 13, 55, 54, 40, 28

A maior sequência não decrescente tem 6 números, e vai do 4 até o 55 (inclusive). Em outro exemplo, considere o vetor abaixo:

10, 9, 8, 7, 6, 5, 4, 3, 2, 1

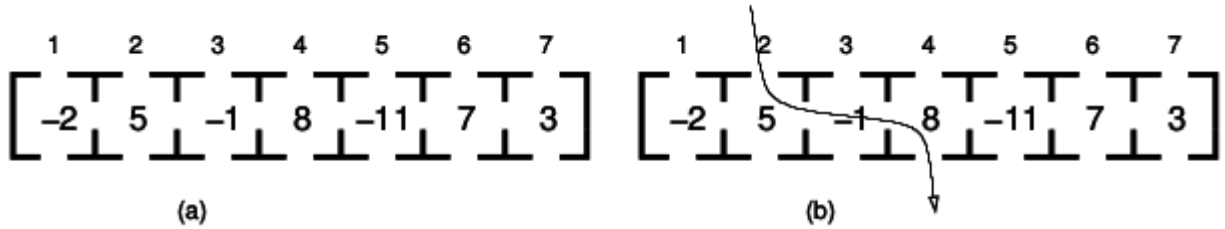
Como o vetor está em ordem decrescente, a maior sequência não decrescente tem tamanho 1 – qualquer um dos números isolados seria uma sequência de tamanho 1!

Além de retornar o tamanho da sequência, a função deve retornar, em 2 parâmetros passados por referência, as posições do início e do fim da sequência. O protótipo da função deve ser:

```
int tamMaiorSeqNDec (int* val, int n, int* inicio, int* fim);
```

(☠) 5. (Adaptado da OBI 2014)

Bruninho está programando um personagem virtual para o próximo desafio de um jogo de aventura em que, numa das fases, o personagem tem que entrar em um corredor, percorrer algumas salas e depois sair do corredor. Ele pode entrar apenas uma vez, e passar por cada sala apenas uma vez. Todas as salas possuem uma porta de entrada e uma de saída, como ilustra a parte (a) da figura abaixo. Ao passar por uma sala o jogador ganha um certo número de vidas (que pode ser negativo!). O objetivo é passar pelo corredor coletando a maior quantidade possível de vidas! Por sorte, sempre existe ao menos uma sala onde se ganha um número positivo de vidas.



No exemplo acima, o personagem de Bruninho pode ganhar, no máximo, 12 vidas, por exemplo, entrando pela sala 2 e saindo pela sala 4, como mostrado na parte (b) da figura. Nesta tarefa, você deve escrever um programa que, dados os números de vidas correspondentes a cada sala do corredor, calcule a quantidade máxima de vidas que será possível ganhar.

Entrada:

O programa recebe como entrada o número de salas no corredor, que estará entre 1 e 1024, e depois, para cada sala, um número inteiro no intervalo [-100, 100] que indica o número de vidas que se ganha naquela sala.

Saída:

O programa deve mostrar o número máximo de vidas que é possível ganhar.

Exemplos:

Entrada	Saída
7 -2 5 -1 8 -11 7 3	12
10 50 42 -35 2 -60 5 30 -1 40 31	105