Universidade de Brasília
Departamento de Ciência da Computação/IE
Semestre: 02/2010
Disciplina: Computação Básica

Trabalho Prático 2

O objetivo desta segunda etapa é que o aluno aprenda a utilizar **estruturas de repetição, estruturas de dados do tipo vetor** em algoritmos e a implementá-las em programas que executem corretamente no computador.

ATENÇÃO: Este trabalho prático é composto de 3 (três) partes, sendo que cada parte possui um problema específico a ser solucionado. O programa com a solução de cada problema deve estar em um arquivo fonte (.c) diferente, e os três arquivos deve ser compactados em um único arquivo .zip ou .rar.

Para **nomenclatura** dos arquivos, utilize:
Parte 1: <*matrícula*>_*2P1.c*
Parte 2: <*matrícula*>_*2P2.c*
Parte 3: <*matrícula*>_*2P3.c*
Os 3 arquivos acima devem ser compactados em um único arquivo: <*matrícula*>_*2.zip ou* <*matrícula*>_*2.rar*

***** PARTE 1 (3 pontos) *****

Problema: Considere um jogo, onde o personagem "*player*" tem a possibilidade de utilizar as seguintes armas para ataque, onde cada arma é identificada no programa por um código (ver fig. abaixo).

 = 1  = 2  = 3

 = 4  = 5  = 6

Durante a execução do jogo, o *player* pode utilizar somente uma arma de cada vez. Contudo uma mesma arma pode ser utilizada novamente para outros ataques. Considere que um vetor guarda os códigos de cada arma que o *player* utilizou durante o jogo. Assim, cada posição do vetor conterá o código da arma (um número inteiro de 1 a 6) utilizada em um determinado ataque.

Entrada:

- Os 15 códigos (nro de 1 a 6) das armas utilizadas pelo player durante o jogo, que serão armazenadas em um vetor. Pode-se assumir que serão informados os códigos corretos.

Saída:

- Dois vetores, sendo que o primeiro deve conter os códigos das armas do vetor de entrada (sem repetição), e o segundo deve conter para cada arma, o número de vezes que cada arma se repetiu no vetor de entrada.

Detalhamento do problema:

Faça um algoritmo, que leia o vetor acima descrito (vetor ARMAS, por ex.), com 15 posições, todas preenchidas com os códigos das armas utilizadas durante o jogo, mostre como saída outros dois vetores, onde o primeiro deve conter os códigos de ARMAS (sem repetição) e o segundo deve conter o número de vezes que cada arma foi utilizada.

Exemplo:

Vetor de entrada, contendo os códigos das armas utilizadas:

6	5	1	6	3	4	3	1	2	1	5	5	1	3	4

Sequências de saída:

Vetor saída 1, contendo os códigos das armas, sem repetição:

6	5	1	3	4	2

Vetor saída 2, contendo o número de vezes que a arma na posição ***i*** do vetor 1 foi utilizada:

2	3	4	3	2	1

Obs: O exemplo acima indica que a arma de código 6 foi utilizada 2 x, a de código 5 foi utilizada 3 x, e assim por diante. Pode acontecer que nem todas as 6 armas tenham sido utilizadas, ou seja, não significa que no vetor de entrada todos os seis códigos estarão cadastrados. Defina os vetores de saída no máximo 6 posições cada, pois este é o nro máximo de armas. Como dito, não significa que as 6 posições dos vetores de saída serão ocupadas.

Sua solução só deve passar pelo vetor ARMAS 1 vez!
Você pode percorrer o vetor de saída 1 quantas vezes necessário! (lembre-se: otimize seu programa!)
A ordem das armas no vetor saída 1 deve ser a mesma do vetor de entrada, porém sem repetição de códigos, como mencionado acima.

***** PARTE 2 (3 pontos) *****

Problema: Dado um número natural ***n***, queremos saber ***quantos*** e ***quais são*** os números primos ***menores*** que ***n***. Um número natural maior que 1 é primo se os seus únicos divisores são o 1 e ele mesmo. Para realizar essa tarefa, você deverá implementar um algoritmo especial que utiliza vetores descrito mais adiante.

Entrada:

- Um número natural ***n***. Faça o teste de consistência (com mensagem de erro) para que ***n*** seja sempre positivo e menor do que 100000.

Saída:

- A quantidade de primos menores que ***n*****,** seguido dos números primos menores que ***n.***

Detalhamento do problema:

Dado um ***n,*** utilizaremos um vetor também de tamanho ***n.*** Ao término do algoritmo, a ***i***-ésima posição do vetor deverá ser:

- -1 → Se ***i*** **não** for primo
- 1 → Se ***i*** for primo

Inicialmente, todas as posições devem ser marcadas com 0, para indicar que aquela posição ainda não foi visitada. Em seguidas, as posições de índice 0 e 1 devem ser marcadas com -1. Em seguida:

1. O algoritmo deverá procurar a menor posição que ainda não foi visitada e marcá-la com 1. Digamos que seja a posição de índice ***i*** no vetor.
2. Em seguida, deverá marcar com -1 ***todas as posições*** que são múltiplas de ***i***. Ou seja, as posições do vetor ***2i, 3i, 4i, …, ki***; onde ***ki < n***
3. Volta para o primeiro passo e marca com 1 a próxima posição livre.

Se o algoritmo tiver sido implementado corretamente, é garantido que apenas as posições cujo índice é primo estarão marcadas com 1. Basta contar o número de posições marcadas com 1, que isso fornecerá a quantidade de primos menores que ***n*** e, em seguida, imprimir essas posições.

Exemplo de entrada:

Informe n:
10

Exemplo de saída:

Existem 4 números primos menores que 10.
2
3
5
7

Exemplo de funcionamento algoritmo:

Para ***n = 10***, utilizaremos as posições de 0 a 9 do vetor. Inicialmente todas as posições devem ser marcadas com 0:

Índice	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Vetor	0	0	0	0	0	0	0	0	0	0

Em seguida, as posições 0 e 1 devem ser marcadas com -1:

Índice	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Vetor	-1	-1	0	0	0	0	0	0	0	0

A próxima posição livre é a posição 2, então ela é marcada com 1:

Índice	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Vetor	-1	-1	1	0	0	0	0	0	0	0

Em seguida, todas as posições múltiplas de 2 são marcadas com -1. Então vamos marcar as posições 4, 6 e 8:

Índice	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Vetor	-1	-1	1	0	-1	0	-1	0	-1	0

A próxima posição livre é a posição 3, então ela é marcada com 1:

Índice	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Vetor	-1	-1	1	1	-1	0	-1	0	-1	0

Em seguida, todas as posições múltiplas de 3 são marcadas com -1. Vamos marcar as posições 6 e 9. A posição 6 já foi marcada anteriormente, mas para simplificar podemos marcá-la novamente:

Índice	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Vetor	-1	-1	1	1	-1	0	-1	0	-1	-1

A próxima posição livre é a posição 5:

Índice	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Vetor	-1	-1	1	1	-1	1	-1	0	-1	-1

Não há múltiplos de 5 no vetor, então vamos para a próxima posição não marcada, que é o 7:

Índice	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Vetor	-1	-1	1	1	-1	1	-1	1	-1	-1

Com isso, todas as posições do vetor foram marcadas, e apenas as posições de índice primo foram marcadas com 1.

Observações adicionais:

- Como o ***n*** máximo é 100000, basta declarar um vetor de de 100000 posições. **int v[100000];**
- O algoritmo básico admite uma série de pequenas otimizações. Por exemplo, após marcar um posição indíce ***i*** primo com 1 e o seus múltiplos com -1, onde i^2 é maior ou igual a ***n***, é garantido que todas as posições restantes correspondem a números primos. Isso advém de um resultado de Teoria dos Números, que garante que se um número ***k*** não é primo, ele possui um divisor maior que 1 e menor ou igual à raiz quadrada de ***k***. Porém, antes de tentar otimizar o algoritmo certifique-se de que a versão básica está funcionando corretamente!

*** PARTE 3 (4 pontos) ***

Problema: O problema da soma de subconjuntos é NP-completo e portanto, em um certo sentido, é "intratável". Essa discussão será deixada para a disciplina de Projeto e Análise de Algoritmos e aqui nós veremos apenas uma variante simples desse problema. O problema é o seguinte: dado um conjunto de números positivos ***v*** e um número ***k***, existe algum subconjunto de ***v*** cuja soma de elementos é maior ou igual à ***k***? Se sim, mostre um subconjunto de ***v*** com o número mínimo de elementos.

Entrada:

- O número ***n*** de elementos do conjunto. Assuma que ***n*** sempre será menor que 1000.
- Os elementos ***inteiros positivos e distintos*** do conjunto.
- O número inteiro positivo ***k.***

Saída:

- ***Sim****,* se há um subconjunto de ***v*** cuja soma é maior ou igual à ***k***. ***Não***, caso contrário.

- Se a resposta for ***Sim***, deverá ser mostrado um subconjunto de ***v*** com o número mínimo de elementos.

Exemplo de entrada:

Informe n:
7
Informe os números:
20
10
70
60
50
40
30
Informe k:
150

Exemplo de saída:

Sim
70
60
50

Exemplo de entrada 2:
Informe n:
5
Informe os números:
5
7
2
1
3
Informe k:
19

Exemplo de saída 2:

Não

Observações adicionais sobre a PARTE 3:

- Um problema pode ter mais de uma solução. No exemplo 1, um outro subconjunto de ***v*** com 3 elementos e cuja soma é maior ou igual a 150 é {70,50,30}, cuja soma é exatamente 150. Qualquer solução é válida.
- Ainda no Exemplo 1, observe que o conjunto {70,60,50,40} tem soma maior ou igual a 150, mas possui 4 elementos. Nós queremos um subconjunto com o menor número de elementos possível. Note que no Exemplo 1, ***nenhum*** subconjunto com 2 elementos tem soma maior ou

igual a 150.

- Lembre-se que um conjunto não tem elementos repetidos. Assim, {70, 70, 70} **não é** uma solução válida para o Exemplo 1.
- Todo conjunto é subconjunto dele próprio. Para o Exemplo 2, se ***k*** fosse 18, a única solução possível seria {7,5,3,2,1}, que é o conjunto todo.
- **Assuma que o usuário sempre informará números inteiros distintos e positivos. Isto é, não é necessário fazer teste de consistência neste problema.**

Observações gerais (comuns aos 3 problemas):

1. As unidades de entrada e saída a serem utilizadas serão o teclado e o monitor de vídeo, respectivamente.

2. Para facilitar o manuseio do programa, deverão ser dadas mensagens solicitando os dados de entrada, e mensagens explicativas das informações de saída fornecidas pelo programa.

3. Use variáveis **Mnemônicas** no seu programa fonte.

4. A saída dos programas devem seguir os exemplos indicados acima, utilizando mensagens similares.

5. Após mostrar os dados de saída de cada problema inserir o comando getchar no programa para que os dados permaneçam na tela aguardando o <enter>.

6. Incluir cabeçalho como comentário (ou seja, entre /* */), no programa fonte, de acordo com os critérios de avaliação dos trabalhos (Disponível no Moodle).

7. Sigam as determinações quanto às entradas e saídas do programa. Serão descontados pontos de programas que tiverem de ter a entrada e/ou a saída corrigidas. Se tiverem alguma dúvida quanto à formatação de entrada e de saída, entrem em contato com os monitores com antecedência em relação ao prazo de entrega do trabalho.

8. A data de entrega do programa é: 05/12/2010 (domingo) até às 23:55 hs, via moodle.