UNIFEI - UNIVERSIDADE FEDERAL DE ITAJUBÁ

CURSO DE GRADUAÇÃO EM SISTEMAS DE INFORMAÇÃO

SIN110 - ALGORITMOS E GRAFOS

RESOLUÇÃO DOS EXERCÍCIOS E01 DO DIA 21/08/2015

ITAJUBÁ

2015

Exercícios E01 – 21/08/15

Aluna: Karen Dantas **Número de matrícula:** 31243

1) Enunciado: A função abaixo recebe uma lista encadeada com cabeça e um inteiro x, e promete devolver p tal que p->chave==x ou NULL se tal p não existir. Analise a função verificando sua correção e eficiência.

Algoritmo

```
struct celula{
        int chave;
        struct celula *prox;
};
struct celula * busca (int x, struct celula * y){
1       int achou=0;
2       struct celula * p = y->prox;
3       while (p!=NULL && !achou){
4               if (p->chave==x)
5                       achou=1;
6               p=p->prox;
7       }
8       if (achou)
9               return p;
10       else
11              return NULL;
12 }
```

Correção

Invariante: A cada ciclo do laço *while*, o valor da variável 'x' será diferente dos dados anteriores na lista à variável 'p->chave' atual.

Início: O invariante é válido quando, na linha 2, 'p' recebe 'y->prox' sendo testado logo a seguir se a lista está vazia e, caso não esteja, entra no *while* e verifica se o número de 'p->chave' é igual a 'x' e se não for o *loop* continua.

Se a estratégia de implementação utilizada na inserção da lista for aquela, na qual, o ponteiro para a cabeça da lista, que será o início da mesma, não contiver nenhum dado, ou seja, o primeiro elemento da lista está contido em 'y->prox', significa que o invariante é válido. Caso contrário, se a estratégia usada for a qual a cabeça da lista aponta para o primeiro elemento da lista, o invariante estará incorreto, pois, ao fazer 'p=y->prox' na linha 2 estará pulando o primeiro dado da lista que pode até ser o dado procurado.

Manutenção: O invariante é mantido através da linha 3 até que se encontre o dado procurado ou se conclua que o dado não estava presente na lista, pois, enquanto o dado for diferente de 'x', na linha 6 'p=p->prox'. Logo, a condição de parada é válida. Porém, o invariante se torna inválido quando o dado for encontrado, pois, a variável 'achou' vai receber o valor 1 e, na linha 6, 'p=p->prox' e essa atribuição fará com que se perda o endereço do dado buscado passando-se para o endereço do dado seguinte, ou seja, em 'p->chave', 'x' não vai ser diferente dos dados anteriores, pois, no endereço anterior a ele, o dado é igual a 'x'.

Término: Quando o laço *while* termina sua execução, se o dado não foi encontrado, o invariante fica correto, pois, todos os dados da lista são diferentes de ‘x’. Caso o dado seja encontrado, o valor da variável retornada (‘p->chave’) é diferente de ‘x’ sendo que o dado imediatamente anterior a ‘p->chave’ é igual a ‘x’. Portanto, o algoritmo está incorreto, pois, o valor do endereço ‘p’ retornado (‘p->chave’) é diferente de ‘x’, logo, a função não faz o que é prometido.

Para que o algoritmo ficasse correto, deveria ter colocado um *else* na linha 6 e, se a estratégia utilizada na inserção da lista for a qual a cabeça da lista aponta para o primeiro elemento da lista, na linha 2 deve-se arrumar para ‘struct celula * p = y’.

Eficiência

Pior caso: O pior caso que vou considerar é quando o dado procurado não está na lista.

Algoritmo	Contagem
struct celula * busca (int x, struct celula * y){	----
1 int achou=0;	1
2 struct celula * p = y->prox;	1
3 while (p != NULL && !achou){	n+1
4 if (p->chave==x)	n
5 achou=1;	0
6 p=p->prox;	n
7 }	----
8 if (achou)	1
9 return p;	0
10 else	1
11 return NULL;	1
12 }	----

F(n) = 1+1+ (n+1) + n + n +1+1+1
F(n) = 3n +6

Comportamento Assintótico: O(n)

Melhor caso: O melhor caso considerado é quando o dado procurado é o primeiro da lista.

Algoritmo	Contagem
struct celula * busca (int x, struct celula * y){	----
1 int achou=0;	1
2 struct celula * p = y->prox;	1
3 while (p != NULL && !achou){	1+1
4 if (p->chave==x)	1
5 achou=1;	1
6 p=p->prox;	1
7 }	----
8 if (achou)	1
9 return p;	1
10 else	0
11 return NULL;	0
12 }	----

F(n) = 1+1+1+1+1+1+1+1+1
F(n) = 9

Comportamento Assintótico**:** O(1)

---

2) Enunciado: Prove a correção e determine o tempo de execução, no pior e no melhor caso, para o algoritmo de classificação abaixo.

Algoritmo

```
ShakeSort (A, n)
1     e <- 1
2     para i <- n-1 ate e faça
3           para j <- e ate i faça
4                 se A[j] > A[j+1]
5                       então troca (A[j], A[j+1])
6           para j <- i ate e + 1 faça
7                 se A[j-1] > A[j]
8                       então troca (A[j-1], A[j])
9           e <- e + 1
```

Correção

Invariante: A cada ciclo do *para* da linha 2, as variáveis contidas do lado esquerdo para o direito do vetor em A[1 ... n] serão movidas, se preciso, para seu lugar correto no vetor através do laço da linha 3. E, na linha 6, serão movidos para suas posições corretas os dados contidos do lado direito para o esquerdo do vetor em A[n-1 ... 2]. Seus espaços de percussão são determinados pelas variáveis ‘i’ e ‘e’, e que, consequentemente, se tornam cada vez menores.

Início: O invariante é válido na linha 3 quando se é verificado na linha 4 se o dado contido em A[1 ... n] está na sua posição correta e, se não estiver, é movido para sua posição correta. Na linha 6, o mesmo ocorre na linha 7, só que com dados contidos em A[n-1 ... 2].

Manutenção: O invariante é mantido através da linha 2, na qual, é decrementada a variável ‘i’ que tem que ser maior ou igual à variável ‘e’ que é incrementada na linha 9. Isso, faz com que seus laços internos rodem ⌊n/2⌋ vezes e o vetor seja percorrido em sentido bidirecional. O invariante também é mantido pelos laços das linhas 4 e 6 que são responsáveis por ordenar o vetor e cujas condições de parada dependem dos valores de ‘i’ e ‘e’. Logo, suas condições de parada também são válidas.

Término: Quando o laço *para* da linha 2 termina sua execução, o vetor foi percorrido em sentido bidirecional ⌊n/2⌋ vezes pelos laços da linha 4 e 6 e seus dados foram ordenados. Portanto, o algoritmo está correto.

Eficiência

Pior caso: O pior caso é quando o vetor está ordenado de forma decrescente. Por exemplo: A[5 | 4 | 3 | 2 | 1].

Algoritmo	Contagem
ShakeSort (A, n)	----
1 e <- 1	1
2 **para** i <- n-1 **ate** e **faça**	$\lfloor n/2 \rfloor + 1$
3 **para** j <- e **ate** i **faça**	$2+...+(n-1)+n = \lceil [(n+2).\lfloor n/2 \rfloor]/2 \rceil$
4 **se** A[j] > A[j+1]	$1+...+(n-1) = \lceil \{[(n-1)+1].\lfloor n/2 \rfloor\}/2 \rceil$
5 **então** troca (A[j], A[j+1])	$1+...+(n-1) = \lceil \{[(n-1)+1].\lfloor n/2 \rfloor\}/2 \rceil$
6 **para** j <- i **ate** e + 1 **faça**	$n+(n-1)+...+3 = \lceil [(n+3).\lfloor n/2 \rfloor]/2 \rceil$
7 **se** A[j-1] > A[j]	$(n-1)+...+2 = \lceil \{[(n-1)+2].\lfloor n/2 \rfloor\}/2 \rceil$
8 **então** troca (A[j-1], A[j])	$(n-1)+...+2 = \lceil \{[(n-1)+2].\lfloor n/2 \rfloor\}/2 \rceil$
9 e <- e + 1	$\lfloor n/2 \rfloor$

$F(n) = 1+ \lfloor n/2 \rfloor+1 + \lceil [(n+2).\lfloor n/2 \rfloor]/2 \rceil + \lceil \{[(n-1)+1].\lfloor n/2 \rfloor\}/2 \rceil + \lceil \{[(n-1)+1].\lfloor n/2 \rfloor\}/2 \rceil + \lceil [(n+3).\lfloor n/2 \rfloor]/2 \rceil + \lceil \{[(n-1)+2].\lfloor n/2 \rfloor\}/2 \rceil + \lceil \{[(n-1)+2].\lfloor n/2 \rfloor\}/2 \rceil + \lfloor n/2 \rfloor$

$= 2 + 2\lfloor n/2 \rfloor + \lceil \{[(n+2)+ n + n + (n+3) + (n+1) + (n+1)] . \lfloor n/2 \rfloor/2\} \rceil$

$= 2 + 2\lfloor n/2 \rfloor + \lceil \{[(6n + 7) . \lfloor n/2 \rfloor]/2\} \rceil$

$= 2 + 2\lfloor n/2 \rfloor + \lceil (\lfloor 6n^2 + 7n/2 \rfloor)/2 \rceil$

$= 2 + 2\lfloor n/2 \rfloor + \lceil 6n^2 + 7n/4 \rceil$

$= 2 + 2\lfloor n/2 \rfloor + \lceil 6n^2/4 + 7n/4 \rceil$

$= 2 + n + \lceil 3n^2/2 \rceil + \lceil 7n/4 \rceil$

$F(n) = \lceil 3n^2/2 \rceil + \lceil 11n/4 \rceil + 2$

Comportamento Assintótico: $O(n^2)$

Melhor caso: O melhor caso considerado é quando vetor já estiver ordenado.

Algoritmo	Contagem
ShakeSort (A, n)	----
1 e <- 1	1
2 **para** i <- n-1 **ate** e **faça**	$\lfloor n/2 \rfloor + 1$
3 **para** j <- e **ate** i **faça**	$2+...+(n-1)+n = \lceil [(n+2).\lfloor n/2 \rfloor]/2 \rceil$
4 **se** A[j] > A[j+1]	$1+...+(n-1) = \lceil \{[(n-1)+1].\lfloor n/2 \rfloor\}/2 \rceil$
5 **então** troca (A[j], A[j+1])	0
6 **para** j <- i **ate** e + 1 **faça**	$n+(n-1)+...+3 = \lceil [(n+3).\lfloor n/2 \rfloor]/2 \rceil$
7 **se** A[j-1] > A[j]	$(n-1)+...+2 = \lceil \{[(n-1)+2].\lfloor n/2 \rfloor\}/2 \rceil$
8 **então** troca (A[j-1], A[j])	0
9 e <- e + 1	$\lfloor n/2 \rfloor$

$F(n) = 1+ \lfloor n/2 \rfloor+1 + \lceil [(n+2).\lfloor n/2 \rfloor]/2 \rceil + \lceil \{[(n-1)+1].\lfloor n/2 \rfloor\}/2 \rceil + \lceil [(n+3).\lfloor n/2 \rfloor]/2 \rceil + \lceil [(n-1)+2].\lfloor n/2 \rfloor\}/2 \rceil + \lfloor n/2 \rfloor$

$= 2 + 2\lfloor n/2 \rfloor + \lceil \{[(n+2)+ n + (n+3) +(n+1)] . \lfloor n/2 \rfloor/2\} \rceil$

$= 2 + 2\lfloor n/2 \rfloor + \lceil \{[(4n + 6) . \lfloor n/2 \rfloor]/2\} \rceil$

$= 2 + 2\lfloor n/2 \rfloor + \lceil (\lfloor 4n^2 + 6n/2 \rfloor)/2 \rceil$

$= 2 + 2\lfloor n/2 \rfloor + \lceil 4n^2 + 6n/4 \rceil$

$= 2 + 2\lfloor n/2 \rfloor + \lceil 4n^2/4 + 6n/4 \rceil$

$= 2 + n + n^2 + \lceil 3n/2 \rceil$

$F(n) = n^2 + \lceil (5n/2) \rceil + 2$

Comportamento Assintótico**:** $O(n^2)$

---

3) Enunciado: Projete um algoritmo, prove sua correção e determine a complexidade de tempo e comportamento assintótico, para o problema: *Dado um inteiro N, verifique se N é primo.*

Algoritmo

```
int verificaNumPrimo(int n){
1       int i, contador=0;
2       if (n<0)
3               n=n*-1;
4       for (i=1; i<=n; i++){
5               if (n % i == 0)
6                   contador++;
7       }
8       if (contador==2)
9               return 1;
10      else
11              return 0;
12 }
```

Correção

Invariante: Para de 1 até num, será verificado quantos divisores naturais o número possui e ele somente será primo se for divisível por apenas dois números.

Início: O invariante é válido, pois na linha 4 o laço *for* garantirá que será verificado o resto da divisão inteira entre o número e seus antecessores a partir de 1. E, se o resto da divisão for igual a zero, o contador será incrementado.

Manutenção: O invariante é mantido até que o número seja dividido por ele mesmo. Logo, a condição de parada é válida.

Término: Quando o laço *for* termina sua execução, é verificado na linha 8, através da variável 'contador', quantas vezes o resto da divisão entre o número e seus antecessores foi igual a zero. Se o 'contador' for igual a dois, significa que o número é primo, pois, os números primos são divisíveis por 1 e por eles mesmos, ou seja, possui apenas dois divisores. Caso contrário, o número não é primo. Portanto, o algoritmo está correto.

Complexidade de tempo

Para o cálculo da complexidade irei considerar que o número passado para a função é primo.

Algoritmo	Contagem
int verificaNumPrimo(int n){	----
1 int i, contador=0;	1
2 if (n<0)	1
3 n=n*-1;	1
4 for (i=1; i<=n; i++){	n+1
5 if (n % i == 0)	n
6 contador++;	2
7 }	----
8 if (contador==2)	1
9 return 1;	1
10 else	0
11 return 0	0
12 }	----

F(n) = 1+1+1+(n+1)+n+2+1+1
F(n) = 2n+8

Comportamento Assintótico: O(n)