

Lista de Pontos Extras da disciplina Análise de Algoritmos do ano 2017

Professor: Herbert Oliveira Rocha

Aluno: Rodrigo Nascimento do Valle

Questão 3 - Implementar Dijkstra com a fila de prioridade usando heap fibonacci.

A implementação está na pasta de programas com o nome AAListaPontosExtrasDijkstraFibonacci

Questão 5) Números primos

5.1) Implementar uma solução correta em tempo polinomial

Uma implementação desse algoritmo pode ser encontrado na pasta de programas com o nome de AAListaPontosExtrasNumerosPrimos

Questão 6 - Calcular a complexidade do algoritmo fibonacci na versão recursivo

fib(n){

if(n<=1){

return n;

}

else{

return fib(n -1) + fib(n -2);

}

}

T(0) = T(1) = 1

T(n-2) ~ T(n-1)

T(n) = 2T(n-1) +c

4T(n-2) + 3c

8T(n-3) + 7c

2^k T(n-k) + (2^k - 1)c

n - k = 0, k = n

T(n) = 2^nT(0) + (2^n -1)c

T(n) = (1 + c)2^n - c

Daí: limite mínimo: 2^n/2 e limite máximo: 2^n.

Então, a complexidade será O(2^n).

Questão 7) Implementar um algoritmo para calcular triângulos polígonos convexos

Uma implementação desse algoritmo pode ser encontrada na pasta de programas com o nome de AAListaPontosExtrasTriangulosPoligonosConvexos.

Questão 8 - Um exemplo de prova de corretude de algoritmos por indução

Dada uma sequência de números reais an, an-1, an-2,..., a1,a0 e um número real x, compute o valor do polinômio

Pn(x) = an xn +an-1 xn-1 +an-2 xn-2+…+ a1 x+a0

Hipótese de indução

Dados uma sequencia de números reais *an*−1, . . . , *a*1, *a*0, e um número real *x*, sabemos calcular o valor de

Pn-1(x) = an-1 xn-1 +an-2 xn-2+…+ a1 x+a0

Base de indução

P0(x) = a0

Passo de indução

Pn(x) = an xn + Pn-1(x)

Algoritmo

p[0] = a[0];  
for(i=1;i<=n;i++){  
 xi = 1;  
 for(j=1;j<=i;j++){  
 xi = xi\*x;   
 }  
 p[i] = a[i]\*xi + p[i-1];  
}

printf("O resultado %lf\n",p[n]);

Este algoritmo realiza (n(n+1))/2 multiplicações e n adições. Vamos fortalecer a nossa hipótese de indução para reduzir o número de multiplicações realizadas.

Hipótese de indução

Dados uma sequência de números reais *an*−1, . . . , *a*1, *a*0, e um número real *x*, sabemos calcular o valor de

Pn-1(x) = an-1 xn-1 +an-2 xn-2+…+ a1 x+a0 e xn-1

Base de indução

P0(x) = a0

Passo de indução

Pn(x) = an\*x\*xn-1 + Pn-1(x)

Algoritmo

p[0] = a[0];  
xi = 1;  
for(i=1;i<=n;i++){  
 xi = xi\*x;  
 p[i] = a[i]\*xi + p[i-1];  
}  
printf("O resultado %lf\n",p[n]);

Este novo algoritmo realiza 2n multiplicações e n adições. Podemos alterar a nossa hipótese de indução para garantir um resultado ainda melhor.

Hipótese de indução

Dados uma sequência de números reais *an*,*an*−1, . . . , *a*1, e um número real *x*, sabemos calcular o valor de

P'n-1(x) = an xn-1 +an-1 xn-1+…+ a2 x+a1

Base de indução

P0(x) = an

Passo de indução

Pn(x) = x\*P'n-1(x) + *a*0

Algoritmo

p[0] = a[n];  
for(i=1;i<=n;i++){  
 p[i]=x\*p[i-1]+a[n-i];  
}

O cálculo do valor do polinômio pode ser descrito da seguinte maneira:

p[0] = a[n]

p[1] = a[n]\*x + a[n-1]

p[2] = a[n]\*x2 + a[n-1]x + a[n-2]

p[3] = a[n]\*x3 + a[n-1]x2 + a[n-2]x + a[n-3]

...

p[n] = a[n]\*xn + a[n-1]xn-1 + a[n-2]xn-2 + ...+a[1]x+a[0]

O algoritmo também pode ser descrito também pela seguinte expressão:

a[n]\*xn + a[n-1]xn-1 + a[n-2]xn-2 + ...+a[1]x+a[0]

=((((((a[n]\*x + a[n-1])\*x + a[n-2])x + a[n-3])...)x+a[1])x +a[0])

Este algoritmo é conhecido como regra de Horner.

Questão 9) Implementar um algoritmo de multiplicação com números grandes usando divisão e conquista.

Uma implementação desse algoritmo pode ser encontrada na pasta de programas com o nome de AAListaPontosExtrasMultiplicaçãoGrandes.

Questão 10 - Prova da complexidade do algoritmo Radix Sort

Quando cada dígito no alcance de 1 até k, e k não é muito grande, COUNTING-SORT será a escolha óbvia. No caso do counting sort, cada número de n-dígitos leva tempo de O(n+k). Existem d passes, então o tempo total para o Radix Sort é (n+k). Existem d passes, então o tempo total realmente será (dn+kd). Quando d é constante e k = O(n), o algoritmo excuta em tempo linear.

Um exemplo do algoritmo radix sort está na lista de programas com o nome de AAListaPontosExtrasRadixSort.

Questão 12 - Explicar porque as operações na árvore binário possuem tempo de execução O(nlogn)

Inserções simples são O(logn). Pelo fato da altura da árvore de n elementos ser logn, o programa precisa fazer n comparações para inserir, retirar ou procurar um único elemento. Então ele precisará fazer n vezes esse esforço no pior caso para qualquer operação. Então n\*logn.

Questão 13 - Implementar o algoritmo de Huffman com a descrição da sua complexidade:

Uma implementação desse algoritmo pode ser encontrado na pasta de programas com o nome de AAListaPontosExtrasHuffman.

Questão 15) Implementar o algoritmo de lista com saltos:

Uma implementação desse algoritmo pode ser encontrado na pasta de programas com o nome de AAListaPontosExtraSkipList.