Prof. Hebert Coelho
Profa Nádia Félix

Conteúdo

1 Percursos em Árvore Binária de Busca

 (++++)

Faça um programa para inserir elementos inteiros em uma árvore binária de busca. Após apresentar duas linhas, uma com os elementos da árvore na sequência do percurso em pré-ordem e outra com os elementos do percurso em nível. Todos os algoritmos devem ser implementados sem uso de recursividade.

Entrada

A primeira linha contém os elementos inteiros separados por espaço a serem inseridos na árvore.

Saída

Você deverá imprimir duas linhas. A primeira linha apresenta os elementos da árvore separados por espaço na sequência do percurso em pré-ordem. A segunda linha apresenta os elementos da árvore separados por espaço na sequência do percurso em nível.

Exemplos

Entrada	Saída
1 2 3 4 5 6 7 8 9 10	1 2 3 4 5 6 7 8 9 10 1 2 3 4 5 6 7 8 9 10

Entrada	Saída
5 3 7 2 4 6 8	5 3 2 4 7 6 8 5 3 7 2 4 6 8

2 Árvore Binária em Vetor

 (+++)

Uma forma de representar uma árvore binária em um vetor é chamado de HEAP. Os HEAPs foram idealizados para que a menor ou a maior chave tenham acesso em tempo constante O(1), assim são usados como filas de prioridade.
Heap máximo: lista linear em um vetor com chaves $s_1, \ldots, s_n$ com propriedade $s_i \leq s_{\lfloor i/2 \rfloor}$, para $1 < i < n$;
Considerando esta regra, podemos visualizar como se cada elemento do vetor na posição i fosse pai de dois elementos $i*2$ e $i*2+1$. A ideia dos métodos para inserir um elemento, remover um elemento ou construir um *Heap* são apresentados abaixo:
Obs.: suponha que o Heap (vetor) tenha n elementos e comece na posição 1 (alguns ajustes devem ser feitos para o vetor começando em 0).

- **Construção 1:** para $i = \lfloor n/2 \rfloor$ até 1 aplique $descer(i,n)$.
- **Construção 2:** para $i = 2$ até n aplique $subir(i)$

Abaixo são apresentados os métodos de subir e descer em um HEAP máximo.
subir(i):
..... $j = \lfloor i/2 \rfloor$
.....**se** $j \geq 1$ **então**
.....**se** $T[i] > T[j]$ **então**
.....$T[i] \Leftrightarrow T[j]$
.....subir(j)
.
.
descer(i, n):
.....$j = 2 * i$
.....**se** $j \leq n$ **então**
..... **se** $j < n$ **então**
..... **se** $T[j+1] > T[j]$ **então**
..... $j = j+1$
..... **se** $T[i] < T[j]$ **então**
..... $T[i] \Leftrightarrow T[j]$
..... descer(j, n)

Dado um sequencia de valores inteiros, construa 2 HEAPs, o primeiro HEAP usando o algoritmo de construção 1 e o segundo HEAP usando o algoritmo de construção 2. Apresente os elementos de ambos HEAPS na tela.

Entrada

A primeira linha contém os $1 < N < 200$ elementos inteiros separados por espaço para construir os HEAPs.

Saída

Você deverá imprimir duas linhas. A primeira linha apresenta os elementos do HEAP1 construído a partir do algoritmo de construção 1. A segunda linha apresenta os elementos do HEAP2 construído a partir do algoritmo de construção 2.

Exemplos

Entrada	Saída
1 2 3 4 5 6 7 8 9 10	10 9 7 8 5 6 3 1 4 2 10 9 6 7 8 2 5 1 4 3

Entrada	Saída
5 3 7 2 4 6 8	5 3 2 4 7 6 8 5 3 7 2 4 6 8

3 Altura máxima e mínima de uma ABB

 (+++)

Dado um conjunto de chaves, construir uma árvore binária de busca (ABB) de altura máxima com tais chaves (árvore degenerada), e construir uma árvore binária de busca de altura mínima com tais chaves (árvore balanceada). Ao final responder a altura de cada árvore. Você deverá construir as árvores e rodar uma função que determina a altura, não apenas fazer os cálculos.

Entrada

A primeira linha contém os $1 < N < 10000$ elementos inteiros separados por espaço para construir as árvores binárias de busca.

Saída

Seu programa deve imprimir dois inteiros, o primeiro inteiro representa a altura da ABB de altura máxima e o segundo inteiro representa a altura da ABB de altura mínima.

Exemplo

Entrada	Saída
`1 2 3 4 5 6 7 8 9 10`	`10` `4`

Entrada	Saída
`5 3 7 2 4 6 8`	`7` `3`

Entrada	Saída
`5 3 7 2 4 6 8 1 15 17 18 30 25 31 40` `55 100 66 88 32 76 77 80`	`23` `5`

4 Percurso diferente em ABB

 (+)

Seja um percurso definido pelas seguintes operações:
Ordem A:

1. Visitar a raiz;
2. percorrer a subárvore esquerda do nó v na ordem A;
3. percorrer a subárvore direita do nó v na ordem B;

Ordem B:

1. percorrer a subárvore esquerda do nó v na ordem B;
2. Visitar a raiz;
3. percorrer a subárvore direita do nó v na ordem A;

Dado um conjunto de chaves, faça um programa que insira os elementos em uma árvore binária de busca, e a partir da raiz execute o percurso em ordem A.

Entrada

A primeira linha contém os $1 < N < 10000$ elementos inteiros separados por espaço para construir a árvore binária de busca.

Saída

Você deverá imprimir uma linha com os elementos da ABB seguindo o percurso A.

Exemplo

Entrada	Saída
1 2 3 4 5 6 7 8 9 10	1 2 3 4 5 6 7 8 9 10

Entrada	Saída
5 3 7 2 4 6 8	5 3 2 4 6 7 8

5 Construindo uma árvore binária

 (+++++)

Uma árvore binária (não precisa ser de busca) pode ser construída de forma única de várias maneiras (veja exercício 3.32 do livro Estruturas de dados e Seus algoritmos e estude todas as formas possíveis). Dados um percurso em ordem simétrica e um percurso em nível, construa a árvore binária referente a tais dados e imprima a árvore com a função de impressão abaixo:

```
void printTree( arvore p, int h){
.....if (p != NULL){
.....printTree(p− >dir, h + 1);
.....printf (“ %∗d \n”, h*5, p− >chave);
.....printTree(p− >esq, h+1);
.....}
}
```

Obs.1: A chamada inicial da função printTree(raiz, 0), ou seja, deve ser passado a raiz da árvore e o valor 0(zero) para h.
Obs.2: printTree imprime a árvore lateralmente, com raiz a esquerda.

Entrada

A primeira linha contém os $1 < N < 10000$ elementos inteiros separados por espaço referente ao percurso em ordem simétrica na árvore binária. A primeira linha contém os $1 < N < 10000$ elementos inteiros separados por espaço referente ao percurso em nível na árvore binária.

Saída

Você deverá imprimir os elementos da árvore binária construída com a função printTree(raiz,0). Obs.: Desconsidere os pontos nas saídas de exemplo abaixo, foram utilizados apenas para que os espaços ficassem corretos.

Exemplo

Entrada	Saída
1 2 3 4 5 6 7 8 9 10 1 2 3 4 5 6 7 8 9 10	.....10 .....9 .....8 .....7 .....6 .....5 .....4 .....3 ....2 1

Entrada	Saída
2 3 4 5 6 7 8 5 3 7 2 4 6 8	.....8 ....7 .....6 5 .....4 ....3 2

Entrada	Saída
4 5 1 7 3 8 1 5 3 4 7 8	.....8 ....3 .....7 1 ....5 4