Homework 5: Binary Tree and Application

CHÚ Ý:

1. Lựa chọn các gói bài tập để thực hiện:

   Combo 1: Bài 1, Bài 2 (cơ bản)

   Combo 2: Bài 2, Bài 3 (nâng cao)

   Combo 3: Bài 4 (nâng cao)

   Combo 4: Bài 5 (nâng cao)

2. Trong bài nộp có file .doc hoặc .txt thuyết minh về gói bài tập thực hiện và những nội dung cần giải thích về bài làm: thuật toán được chọn, tài liệu tham khảo, định dạng input, yêu cầu hệ thống...

<Binary Tree>

Bài 1. Tạo giao diện BinaryTreeInterface như sau:

```
public interface BinaryTreeInterface<T> {
     T root();
     int size();            // number of node in tree
     boolean isEmpty();
     int numChildren(T p);  // nmber of children of element p;

     T parent(T p);  //return parent of p
     T left(T p);    //return left child of p
     T right(T p);   //return right child of p
     T sibling(T p); //return sibling of p
}
```

1.1 Xây dựng kiểu dữ liệu BinaryTree sử dụng mảng, cài đặt giao diện BinaryTreeInterface đã xây dựng ở trên với lược đồ gợi ý như sau:

```
public class ArrayBinaryTree<E,T> implements BinaryTreeInterface<T> {
     private E [] array;
     private int n = 0;
     private int defaultsize = 100;

     public ArrayBinaryTree(){
          array = (E[]) new Object[defaultsize];
     }
     //update methods
     public void setRoot(E element) {
          // Set element to root of an empty tree (at index 1)
     }
     public void setLeft(int p, E element) {
          // Set left child of p (at index 2p)
     }
     public void setRight(int p, E element) {
          // Set right child of p (at index 2p+1)
     }
}
```

1.2 Xây dựng cấu trúc dữ liệu BinaryTree sử dụng cấu trúc móc nối, cài đặt giao diện BinaryTreeInterface đã xây dựng ở trên với lược đồ gợi ý như sau:

```
public class LinkedBinaryTree<E,T> implements BinaryTreeInterface<T> {

     protected static class Node<E> {
          private E element;         // an element stored at this node
          private Node<E> parent;    // a reference to the parent node (if any)
          private Node<E> left;      // a reference to the left child
          private Node<E> right;     // a reference to the right child

          // Constructs a node with the given element and neighbors. */
          public Node(E e, Node<E> above, Node<E> leftChild, Node<E> rightChild){
               // Sinh viên hoàn thiện phương thức
          }
     }
     //update methods
     public Node<E> addRoot(E element) {
          // Add element to root of an empty tree
     }

     public Node<E> addLeft(Node p, E element) {
          // Add element to left child node of p if empty
     }

     public Node<E> addRight(Node p, E element) {
          // Add element to right child node of p if empty
     }

     public void set(Node p, E element) {
          // set element to nnode p
     }
```

Lưu ý: T là position trong cây, nếu cài đặt bằng Array thì T là kiểu **int**, nếu cài đặt bằng Linked thì T là kiểu **Node<E>**, E là kiểu generic giá trị của phần tử (data) trong cây.

1.3 Tạo một cây nhị phân sử dụng hai cấu trúc dữ liệu trên. Viết hàm in và in ra màn hình và file output.txt cây nhị phân vừa tạo theo chiều ngang. Ví dụ:

```
          7
     3
          2
1
          6
     5
          8
```

<ExpressionTree>

Bài 2. Biểu thức số học (Arithmetic Expression) có thể biểu diễn theo một dạng khác nhau tùy thuộc vào vị trí tương đối của toán tử so với các toán hạng. Phổ biến bao gồm các dạng biểu thức: trung tố (infix), hậu tốt (postfix), tiền tố (prefix). Ví dụ:

- Biểu thức trung tố: (6/3 + 2) * (7 - 4) (toán hạng đứng hai bên toán tử)
- Biểu thức hậu tố: 6 3 / 2 + 7 4 - * (toán hạng đứng sau toán tử)

- Biểu thức tiền tố: * + / 6 3 2 – 7 4 (toán hạng đứng trước toán tử)

2.1 Xây dựng lớp ExpressionTree mở rộng lớp LinkedBinaryTree với việc bổ sung các phương thức duyệt cây theo lược đồ gợi ý như sau:

```
public class ExpressionTree<E> extends LinkedBinaryTree{

     public void preorderPrint(Node<E> p) {
          //pre-order traversal of tree with root p
     }

     public void postorderPrint(Node<E> p) {
          //post-order traversal of tree with root p
     }

     public void inorderPrint(Node<E> p) {
          //in-order traversal of tree with root p
     }
}
```

2.2 Sử dụng các phương thức để in ra các dạng biểu diễn của biểu thức (tiền tố, trung tố, hậu tố) được biểu diễn bằng một cây nhị phân biểu thức cho trước.

Vào: Cây nhị phân biểu thức

Ra : Các dạng biểu diễn của biểu thức

2.3 Sử dụng các phương thức để tính toán giá trị của biểu thức biểu diễn bởi một cây nhị phân cho trước.

Vào: Cây nhị phân biểu thức

Ra : Giá trị của biểu thức

PHẦN BÀI TẬP NÂNG CAO VÀ ỨNG DỤNG

Bài 3. Viết chương trình dựng cây nhị phân biểu diễn biểu thức số học được cho dưới dạng chuỗi.

(*Xem giải thích thêm về các dạng biểu diễn biểu thức số học ở cuối bài tập*)

Bài 4. Xem yêu cầu project P-8.68 sách M. Goodrich, trang 358 và thực hiện project.

Bài 5. Xem yêu cầu project P-8.69 sách M. Goodrich, trang 358 và thực hiện project.

Giải thích thêm về các dạng biểu diễn biểu thức số học (dùng cho Bài 3)

Biểu thức số học (Arithmetic Expression) có thể biểu diễn theo một dạng khác nhau tùy thuộc vào vị trí tương đối của toán tử so với các toán hạng. Phổ biến bao gồm các dạng biểu thức: trung tố (infix), hậu tốt (postfix), tiền tố (prefix). Ví dụ:

- Biểu thức trung tố: (6/2 + 3) * (7 - 4) (toán hạng đứng hai bên toán tử)
- Biểu thức hậu tố: 6 2 / 3 + 7 4 - * (toán hạng đứng sau toán tử)
- Biểu thức tiền tố: * + / 6 2 3 – 7 4 (toán hạng đứng trước toán tử)

Độ ưu tiên của các toán tử như sau:

- Toán tử nhân (*) = toán tử chia (/)
- Toán tử (+) = toán tử trừ (-)
- Nhân, chia ưu tiên cao hơn cộng, trừ

Bài toán dựng cây nhị phân biểu diễn biểu thức số học được cho dưới dạng chuỗi được phân tích như sau:

Input: biểu thức dạng trung tố đã được tách thành các tokens. Mỗi token là thành phần của biểu thức gồm dấu ngoặc, toán hạng, toán tử.

Ví dụ input cho biểu thức (6/3 + 2) * (7 - 4) là

String[] tokens = {"(","6", "/", "3","+","2",")", "*", "(", "7", "-", "4",")"}

Ouput: cây nhị phân biểu diễn biểu thức (sử dụng cấu trúc liên kết)

Algorithm: (1 trong 2 cách)

1. Dựng cây trực tiếp từ các token input
2. Chuyển biểu thức từ dạng trung tố sang hậu tố hoặc tiền tố sử dụng Stack, rồi dựng cây.

Một số thuật toán hỗ trợ:

a) Thuật toán chuyển từ dạng trung tố sang dạng hậu tố:

*Đọc từng token trong biểu thức infix từ **trái qua phải**, với mỗi token ta thực hiện các bước sau:*

- *Nếu là toán hạng: cho ra output.*
- *Nếu là dấu mở ngoặc "(": cho vào stack.*
- *Nếu là dấu đóng ngoặc ")": lấy các toán tử trong stack ra và cho vào output cho đến khi gặp dấu mở ngoặc "(". Dấu mở ngoặc cũng phải được đưa ra khỏi stack.*
- *Nếu là toán tử:*
  - *Chừng nào ở đỉnh stack là toán tử và toán tử đó có độ ưu tiên **lớn hơn hoặc bằng** toán tử hiện tại thì lấy toán tử đó ra khỏi stack và cho ra output.*
  - *Đưa toán tử hiện tại vào stack .*

Sau khi duyệt hết biểu thức infix, nếu trong stack còn phần tử thì lấy các token trong đó ra và cho lần lượt vào output. Cuối cùng là đảo ngược biểu thức một lần nữa và ta sẽ thu được kết quả

b) Thuật toán chuyển từ dạng trung tố sang dạng tiền tố:

*Đọc từng token trong biểu thức infix từ **phải qua trái**, với mỗi token ta thực hiện các bước sau:*

- *Nếu là toán hạng: cho ra output.*
- *Nếu là dấu đóng ngoặc ")": cho vào stack.*
- *Nếu là dấu mở ngoặc "(": lấy các toán tử trong stack ra và cho vào output cho đến khi gặp dấu đóng ngoặc ")". Dấu đóng ngoặc cũng phải được đưa ra khỏi stack.*
- *Nếu là toán tử:*
  - *Chừng nào ở đỉnh stack là toán tử và toán tử đó có độ ưu tiên **lớn hơn hoặc bằng** toán tử hiện tại thì lấy toán tử đó ra khỏi stack và cho ra output.*
  - *Đưa toán tử hiện tại vào stack*

Sau khi duyệt hết biểu thức infix, nếu trong stack còn phần tử thì lấy các token trong đó ra và cho lần lượt vào output. Cuối cùng là đảo ngược biểu thức một lần nữa và ta sẽ thu được kết quả.

c) Thuật toán dựng cây nhị phân biểu thức:

Từ biểu thức xâu kí tự đầu vào, dựng cây nhị phân biểu diễn biểu thức theo các bước sau:

1. ***Chuyển biểu thức từ dạng trung tố sang hậu tố***
2. ***Xây dựng cây nhị phân từ dạng hậu tố:***

   Lặp qua từng token trong chuỗi hậu tố

- *Tạo một đối tượng BinaryTreeNode nhãn của node là token này*
- *Nếu là toán hạng: Push node vào stack*
- *Nếu là toán tử:*
  - *Pop một toán hạng ra khỏi stack và đặt làm RightChild của node*
  - *Pop toán hạng kế tiếp ra khỏi stack và đặt làm LeftChild của node*
  - *Push node vào stack*

Sau khi vòng lặp kết thúc, phần tử cuối cùng còn lại trong stack là node gốc của cây biểu thức.