Objektum elvű alkalmazások fejlesztése

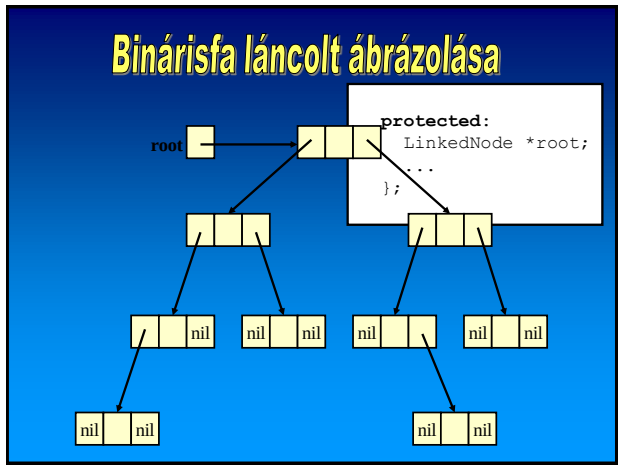
Bináris fa bejárása

Készítette: Gregorics Tibor
Steingart Ferenc

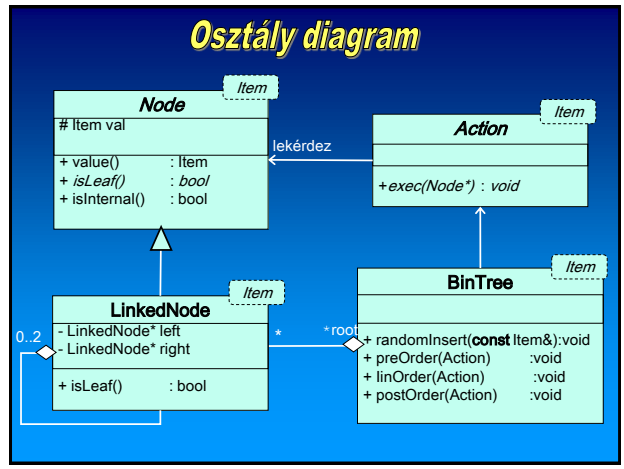
Feladat

Olvassuk be a szabványos bemenetről számokat, építsünk fel ezekből véletlenszerűen egy bináris fát, írjuk ki a csúcsokban tárolt értékeket a szabványos kimenetre különféle bejárási stratégiák alapján, végül határozzuk meg a belső csúcsokban tárolt értékek összegét, és az összes csúcs értékeinek maximumát!

Binárisfa láncolt ábrázolása



Osztály diagram



Absztrakt tevékenység osztály-sablonja

```
template <typename Item>
class Action{
public:
   virtual void exec(Node<Item> *node)=0;
};
```

bintree.hpp

Absztrakt csúcs osztály-sablonja

```
template <typename Item>
class Node {

   public:
      Item value() const {return _val;}
      virtual bool isLeaf() const = 0;
      bool isInternal() const {return !isLeaf();}

   protected:
      Node(const Item& v): _val(v){}
      Item _val;
};
```

bintree.hpp

Láncolt csúcs osztálya

```
template <typename Item> class BinTree;

template <typename Item>
class LinkedNode: public Node<Item>{
   friend class BinTree;

   public:
      LinkedNode(const  Item& v, LinkedNode *l,
                                 LinkedNode *r):
        Node<Item>(v), _left(l), _right(r){}
      bool isLeaf()const
        {return _left==NULL && _right==NULL;}

   private:
      LinkedNode *_left;
      LinkedNode *_right;
};
```

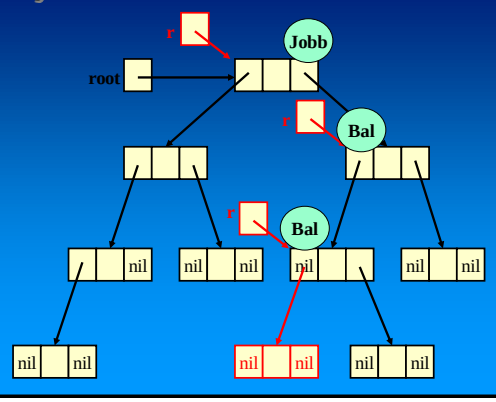
bintree.hpp

Bináris fa osztály-sablonja

```
template <typename Item>
class BinTree{
   public:
      BinTree():root(NULL){srand(time(NULL));}
      ~BinTree();

      void randomInsert(const Item& e);

      void preOrder  (Action<Item>*todo){ pre(_root,todo);}
      void inOrder   (Action<Item>*todo){  in(_root,todo);}
      void postOrder(Action<Item>*todo){post(_root,todo);}

   protected:
      LinkedNode<Item>* _root;
      friend class Action;
      void pre  (LinkedNode<Item>*r, Action<Item>*todo);
      void in   (LinkedNode<Item>*r, Action<Item>*todo);
      void post(LinkedNode<Item>*r, Action<Item>*todo);
};
```

bintree.hpp

Új csúcs beszúrása bináris fába



Új csúcs beszúrása bináris fába

```
void BinTree<Item>::randomInsert(const Item& e)
{
    if(_root==NULL) _root =
                new LinkedNode<Item>(e,NULL,NULL);
    else {
        LinkedNode<Item> *r = _root;
        int d = rand();
        while(d&1 ? r->_left!=NULL : r->_right!=NULL){
            if(d&1) r = r->_left;
            else    r = r->_right;
            d = rand();
        }
        if(d&1) r->_left =
                new LinkedNode<Item>(e,NULL,NULL);
        else     r->_right=
                new LinkedNode<Item>(e,NULL,NULL);
    }
}
```

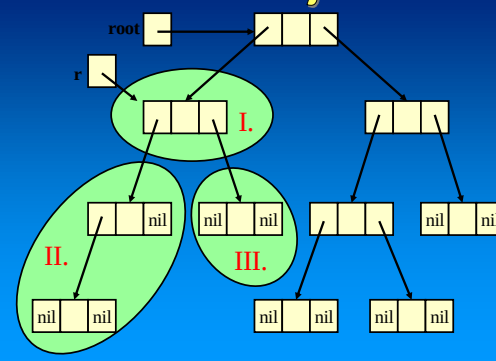
bintree.hpp

Bináris fa felépítése

```
int main()
{
   BinTree<int> t;
   int i;
   while(cin >> i){
      t.randomInsert(i);
   }
```

test.cpp

Preorder bejárás

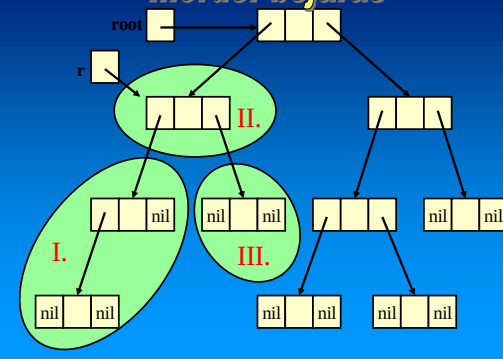


Preorder bejárás

```
template <typename Item>
void BinTree<Item>::
pre(LinkedNode<Item> *r, Action<Item> *todo)
{
    if(r==NULL) return;
    todo->exec(r);
    pre(r->_left,  todo);
    pre(r->_right, todo);
}
```

bintree.hpp

Inorder bejárás

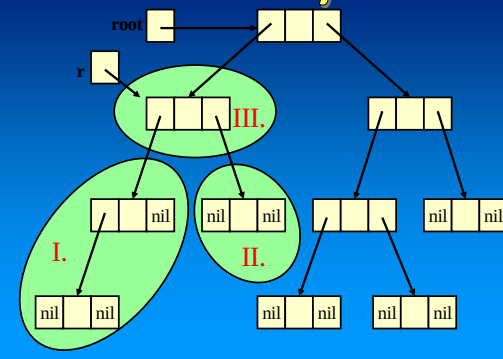


Inorder bejárás

```
template <typename Item>
void BinTree<Item>::
in(LinkedNode<Item> *r, Action<Item> *todo)
{
    if(r==NULL) return;
    in(r->_left,  todo);
    todo->exec(r);
    in(r->_right, todo);
}
```

bintree.hpp

Postorder bejárás

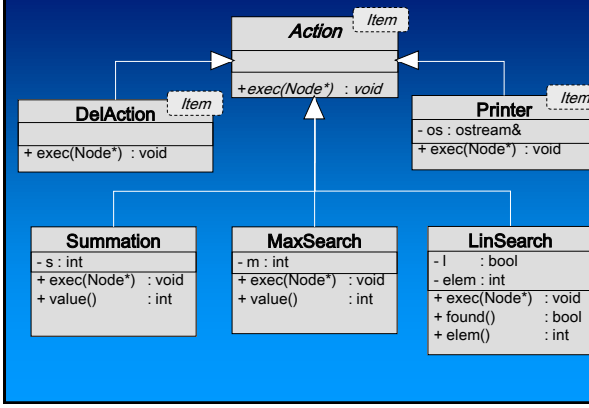


Postorder bejárás

```
template <typename Item>
void BinTree<Item>::
post(LinkedNode<Item> *r, Action<Item> *todo)
{
    if(r==NULL) return;
    post(r->_left,  todo);
    post(r->_right, todo);
    todo->exec(r);
}
```

bintree.hpp

Tevékenység osztályok



Kiírás tevékenység osztály-sablonja

```
template <typename Item>
class Printer: public Action<Item>{
public:
   Printer(ostream &o): os(o){};
   void Exec(Node<Item> *node)
        {os << '['<< node->Value() << ']';}
private:
   ostream& os;
};
```

test.cpp

Kiíratás bejárásokkal

```
Printer<int> print(cout);

cout << "Preorder bejárás:";
t.PreOrder(&print);
cout << endl;

cout << "Inorder bejárás:";
t.InOrder(&print);
cout << endl;

cout << "Postorder bejárás:";
t.PostOrder(&print);
cout << endl;
```

test.cpp

Destruktor

```
class DelAction: public Action<Item>{
public:
   void Exec(Node<Item> *node)
        {delete node;}
};
```

BinTree<Item>-ben

```
template <typename Item>
BinTree<Item>::~BinTree()
{
   DelAction del;
   Post(root, &del);
}
```

Csak POSTORDER-rel

bintree.hpp

Összegzés megfogalmazása tevékenységobjektummal

Összegzés

$t : enor(E)$ $\quad f:E\to H,\ felt:E\to \mathbb{L}$

$s := \Sigma_{e \in t,\ felt(e)} f(e)$

$s := 0$

ha $felt(e)$ akkor
$s := s + f(e)$

Összegzés tevékenység osztálya

```
class Summation: public Action<int>{
public:
   void Summation():s(0){}
   void Exec(Node<int> *node){
      { if(node->IsInternal()) s+=node->Value();}
   int Value(){return s;}
private:
   int s;
};
```

$s := 0$

ha $felt(e)$ akkor
$s := s + f(e)$

```
Summation sum;
t.PreOrder(&sum);
cout << "Fa elemeinek összege:"
     << sum.Value() << endl;
```

test.cpp

Maximum kiválasztás tevékenység osztálya

```
class MaxSearch: public Action<int>{
public:
   Max(int &i) : m(i){}
   void Exec(Node<int> *node)
     {m = m>=node->Value() ? m : node->Value();}
   int Value(){return m;}
private:
   int m;
};
```

$m := initialvalue$

$m := \boldsymbol{max}\{m, e\}$

```
MaxSearch max(t.RootValue());
t.PreOrder(&max);
cout << "Maximális elem:" << max.Value();
```

```
template <class Item> class BinTree {
…
public:
   enum Exceptions{NOROOT};
   Item RootValue() const {
     if( root==NULL ) throw NOROOT;
     return root->Value();
   }
```

test.cpp

Lineáris keresés tevékenység osztálya

```
class LinSearch: public Action<int>{
public:
   void LinSearch():l(false){}
   void Exec(Node<int> *node)
      {l = l && (elem = node->Value())%2==0 );}
   bool  Found(){return l;}
   int   Elem() {return elem;}
private:
   bool l;
   int  elem;
};
```

l := hamis

l, elem := felt(e), e

páros-e

test.cpp