ELEMI ALKALMAZÁSOK FEJLESZTÉSE II.

Bináris fa bejárása

Készítette: Gregorics Tibor
Steingart Ferenc

Tartalom

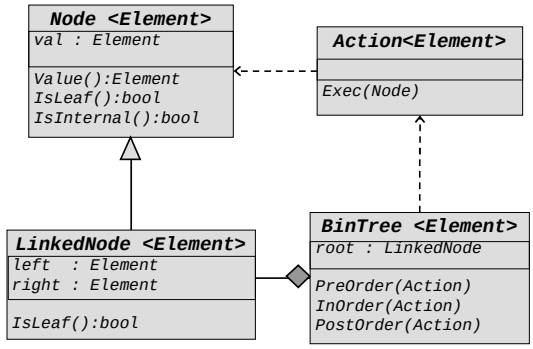
- Bináris fa-típus láncolt ábrázolással
- Rekurzív bejárás tevékenység objektumokkal

1. Feladat

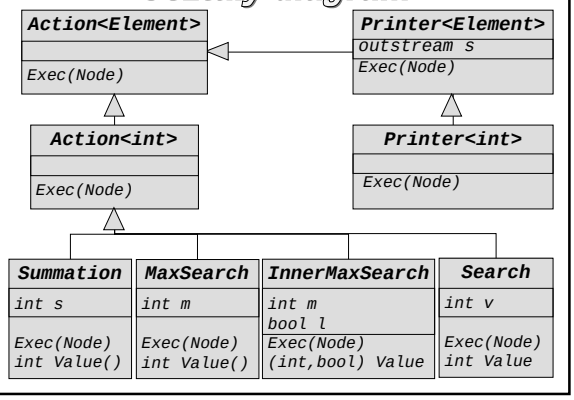
Olvassuk be a standard bemenetről érkező számokat, építsünk fel ezekből véletlenszerűen egy bináris fát, írjuk ki a standard kimenetre különféle bejárási stratégiák mellett a csúcsokban tárolt értékeket, határozzuk meg ezek összegét, a maximumát, majd csak a belső csúcsok értékeinek maximumát, végül keressünk egy páros számot!

A feladat megoldásához több egymáshoz szorosan kapcsolódó osztály-sablont hozunk létre.

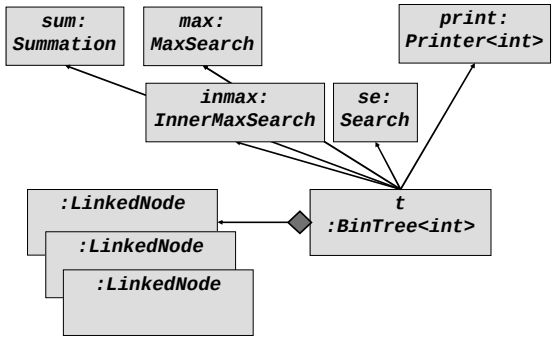
Osztály diagram



Osztály diagram



Objektum diagram



Bináris fa osztály-sablonja publikus rész

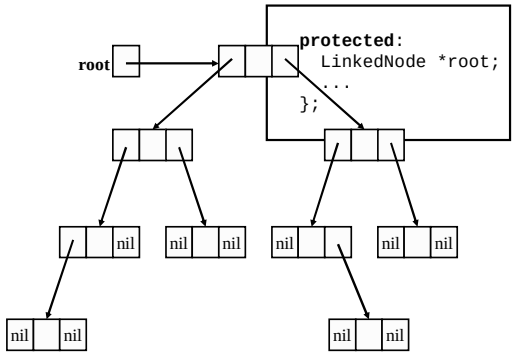
```
template <class Element>
class BinTree{
public:
   BinTree();
   ~BinTree();

   void RandomInsert(const Element& e);

   void PreOrder (Action<Element> *todo);
   void InOrder  (Action<Element> *todo);
   void PostOrder(Action<Element> *todo);
```

bintree.h

Binárisfa láncolt ábrázolása



Absztrakt csúcs osztály-sablonja

```
template <class Element>
class Node {
public:
   const Element& Value() const {return val;}
   virtual bool  IsLeaf() const = 0;
   bool IsInternal() const {return!IsLeaf();}
protected:
   Node(const Element &v): val(v){}
   Element val;
};
```

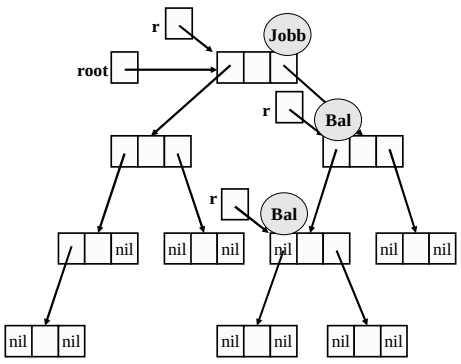
bintree.h

Láncolt csúcs osztálya

```
class LinkedNode: public Node<Element>{
friend class BinTree;
public:
   LinkedNode(const Element& v,
        LinkedNode *l, LinkedNode *r):
        Node<Element>(v), left(l), right(r){}
   virtual bool IsLeaf()const
        {return left==nil && right==nil;}
private:
   LinkedNode *left;
   LinkedNode *right;
};
};
```

bintree.h

Új csúcs beszúrása bináris fába



Új csúcs beszúrása bináris fába

```
void BinTree<Element>::RandomInsert(const Element& e)
{
    if(root==nil) root = new LinkedNode(e,nil,nil);
    else {
        LinkedNode *r = root;
        int d = rand();
        while(d&1 ? r->left!=nil : r->right!=nil){
            if(d&1) r = r->left;
            else    r = r->right;
            d = rand();
        }
        if(d&1) r->left  = new LinkedNode(e,nil,nil);
        else    r->right = new LinkedNode(e,nil,nil);
    }
}
```

bintree.h

Bináris fa osztály-sablonja publikus rész

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

   void RandomInsert(const Element& e);

   void PreOrder (Action<Element> *todo)
                              {Pre(root, todo);}
   void InOrder  (Action<Element> *todo)
                              {In(root, todo);}
   void PostOrder(Action<Element> *todo)
                              {Post(root, todo);}
```

bintree.h

Absztrakt tevékenység osztály-sablonja

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

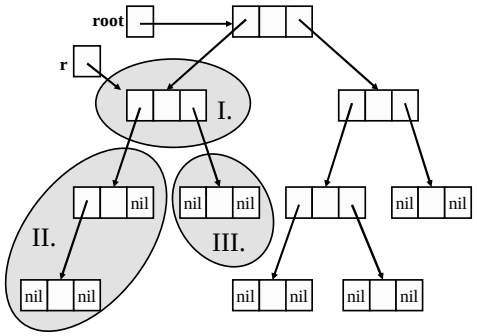
bintree.h

Bináris fa osztály-sablonja rejtett rész

```
protected:
   LinkedNode *root;
   void Pre (LinkedNode *r, Action<Element>*todo);
   void In  (LinkedNode *r, Action<Element>*todo);
   void Post(LinkedNode *r, Action<Element>*todo);
};
```

bintree.h

Preorder bejárás

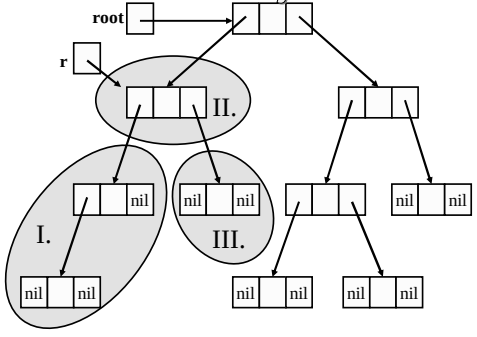


Preorder bejárás

```
template <class Element>
void BinTree<Element>::
Pre(LinkedNode *r,Action<Element> *todo)
{
    if(r==nil) return;
    todo->Exec(r);
    Pre(r->left,  todo);
    Pre(r->right, todo);
}
```

bintree.h

Inorder bejárás

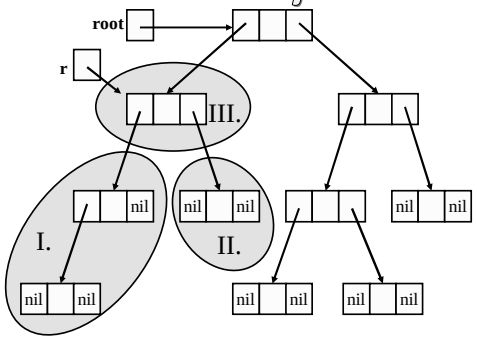


Inorder bejárás

```
template <class Element>
void BinTree<Element>::
In(LinkedNode *r,Action<Element> *todo)
{
    if(r==nil) return;
    In(r->left,  todo);
    todo->Exec(r);
    In(r->right, todo);
}
```

bintree.h

Postorder bejárás



Postorder bejárás

```
template <class Element>
void BinTree<Element>::
Post(LinkedNode *r,Action<Element> *todo)
{
    if(r==nil) return;
    Post(r->left,  todo);
    Post(r->right, todo);
    todo->Exec(r);
}
```

bintree.h

Kiírás tevékenység osztály-sablonja

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

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

t.PostOrder(&print);
```

test.cpp

Destruktor

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

Új tevékenység : törlés

bintree.h

Törlés tevékenység osztálya

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

```
Post(r,d);

   LinkedNode<int> *r;
   DelAction  *d;

   d->Exec(r);
```

bintree.h

Tevékenységek rekurzív megfogalmazása

Összegzés

$s:=\Sigma_{i=1..n} f(i)$
$f:[1..n]\rightarrow H$
helyett

$s := r(n)$
ahol
$r:[0..n]\rightarrow H$
$r(0) := 0$
$r(i) := r(i\text{-}1)+f(i)$

Maximum kiválasztás

$m:=\mathbf{max}_{i=1..n}\{ f(i) \}$
$f:[1..n]\rightarrow H$
helyett

$m := r(n)$
ahol
$r:[1..n]\rightarrow H$
$r(1) := f(1)$
$r(i) := \mathbf{max}\{ r(i\text{-}1), f(i) \}$

Az $i=1..n$ helyett valamelyik fabejárást alkalmazzuk,
az $f(i)$ a fa i-edik csúcsának értéke,
a $\beta(i)$ az i-edik csúcsra megfogalmazott feltétel.

Összegzés tevékenység osztálya

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

test.cpp

Maximumkeresés tevékenység osztálya

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

test.cpp

Maximumkeresés alkalmazása

```
try{
   MaxSearch max(t.RootValue());
   t.PreOrder(&max);
   cout << "Fa maximális eleme:" << max.Value();
}catch(BinTree::Exceptions e){
   if( e==BinTree::NOROOT )
      cout << "Nincs elem a fában!" ;
}
cout << endl;
```

test.cpp

```
Element RootValue()
{
  if( root==nil ) throw NOROOT;
  return root->Value();
}
...
```

bintree.h

Tevékenységek rekurzív megfogalmazása

Feltételes maximum keresés

$l,m:=\mathbf{feltmax}_{i=1..n}\{ f(i) \mid \beta(i) \}$
$f:[1..n]\rightarrow H$
$\beta:[1..n]\rightarrow L$
helyett

$l,m := r(n)$
ahol
$r:[0..n]\rightarrow L\times H$
$r(0) := (\ hamis, *\ )$

$$r(i) := \begin{cases} r(i-1) & \text{ha } \neg\beta(i) \\ (r(i-1)_1, \mathbf{max}\{r(i-1)_2, f(i)\}) & \text{ha } \beta(i)\wedge r(i-1)_1 \\ (igaz, f(i)) & \text{ha } \beta(i)\wedge\neg r(i-1)_1 \end{cases}$$

Feltételes maximum keresés tevékenység osztálya

```
class InnerMaxSearch: public Action<int>{
public:
   struct Result {
      int  m;
      bool l;
   };

   InnerMaxSearch(){r.m = 0; r.l = false;}
   void Exec(Node<int> *node)
   {
      ...
   }
   Result Value(){return r;}
private:
   Result r;
}
```

test.cpp

Feltételes maximum keresés tevékenység

```
void Exec(Node<int> *node)
{
   if(node->IsInternal()){
      if(!r.l){
         r.l = true;
         r.m = node->Value();
      }else if(node->Value()>r.m){
         r.m = node->Value();
      }
   }
}
```

test.cpp

Tevékenységek rekurzív megfogalmazása

Keresés

$l,i := \textbf{keres}_{\substack{i=1..n \\ \beta:[1..n]\to L}} \beta(i)$

helyett

$l,i := r(i)$

ahol

$r:[0..n] \to L \times H$

$r(0) := (hamis, 0)$

$$r(i) := \begin{cases} r(i-1) \text{ ha} & r(i-1)_1 \vee \neg\beta(i) \\ (igaz, i) \text{ ha} & \neg r(i-1)_1 \wedge \beta(i) \end{cases}$$

Lineáris keresés tevékenység osztálya

```
class Search: public Action<int>{
public:
   enum Exceptions{FOUND};

   Search(){};
   void Exec(Node<int> *node)
   {
      if(node->Value()%2==0){
            v = node->Value();
            throw FOUND;
      }
   }
   int Value(){return v;}
private:
   int v;
};
```

test.cpp

Bináris fa felépítése

```
int main()
{
   BinTree<int> t;
   int i;
   cin >> i;
   while(i!=0){
      t.RandomInsert(i);
      cin >> i;
   }
```

test.cpp

Bejárások

```
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

Összegzés és maximum kiválasztás

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

try{
   MaxSearch max(t.RootValue());
   t.PreOrder(&max);
   cout << "Fa maximális eleme:" << max.Value();
}catch(BinTree::Exceptions e){
   if( e==BinTree::NOROOT )
      cout << "Nincs elem a fában!" ;
}
cout << endl;
```

test.cpp

Feltételes maximum keresés

```
InnerMaxSearch inmax;
t.PreOrder(&inmax);
cout << "Belső csúcsok maximális értéke:";
if( inmax.Value().l )cout << inmax.Value().m;
else                 cout << "Nincs belső csúcs!";
cout << endl;
```

test.cpp

Keresés

```
Search se;
try{
   t.PreOrder(&se);
   cout << "Nincs páros szám!";
}catch(Search::Exceptions e){
   if( e==Search::FOUND )
      cout << "Első páros szám:" << se.Value();
}
cout << endl;
```

test.cpp