情報数理科学第5回課題　08-192025 林橘平

Ex 6-4

コード(BinaryTree.hの該当部分のみを抽出)

```
template<class Node>
int BinaryTree<Node>::size2(Node *u) {
        Node  *a = u, *prev = a->parent, *next;
        if ((a->left == nil) && (a->right == nil)) return 1;
        int n = 0;
        while (a != nil) {
                if (prev == a->parent) {
                        n++;
                        if (a->left != nil) next = a->left;
                        else if (a->right != nil) next = a->right;
                        else next = a->parent;
                } else if (prev == a->left) {
                        if (a->right != nil) next = a->right;
                        else next = a->parent;
                } else {
                        next = a->parent;
                }
                prev = a;
                a = next;
        }
        return n;
}
```

checkEx6_4.cpp

```
#include "BinaryTree.h"

class MyBinaryTree : public ods::BinaryTree<ods::BTNode1>{
  typedef ods::BTNode1 Node;
public:
  MyBinaryTree(ods::BTNode1* nil_, ods::BTNode1* r_)
:ods::BinaryTree<ods::BTNode1>(nil_){
    r = r_;
  }
  void traverse(){
    traverse(r);
  }
  void traverse(Node* u){
```

```
    if (u == nil) return;
    std::cout << u << std::endl;
    traverse(u->left);
    traverse(u->right);
  }
  void traverse2() {
    Node *u = r, *prev = nil, *next;
    while (u != nil) {
      std::cout << u << std::endl;
      if (prev == u->parent) {
        if (u->left != nil) next = u->left;
        else if (u->right != nil) next = u->right;
        else next = u->parent;
      } else if (prev == u->left) {
        if (u->right != nil) next = u->right;
        else next = u->parent;
      } else {
        next = u->parent;
      }
      prev = u;
      u = next;
    }
  }
  void bfTraverse() {
    ods::ArrayDeque<Node*> q;
    if (r != nil) q.add(q.size(),r);
    while (q.size() > 0) {
      Node *u = q.remove(0);
      std::cout << u << std::endl;
      if (u->left != nil) q.add(q.size(),u->left);
      if (u->right != nil) q.add(q.size(),u->right);
    }
  }
};

int main(){
  ods::BTNode1 *r = new ods::BTNode1();
  ods::BTNode1 *n1 = new ods::BTNode1();
  ods::BTNode1 *n2 = new ods::BTNode1();
  ods::BTNode1 *n3 = new ods::BTNode1();
  ods::BTNode1 *n4 = new ods::BTNode1();
  std::cout << "r = " << r << ",n1=" << n1 << ",n2=" << n2 << ",n3=" << n3
<< ",n4=" << n4 << std::endl;
  r->left = n1; n1->parent = r;
  r->right = n2; n2->parent = r;
  n2->left = n3; n3->parent = n2;
  n2->right = n4; n4->parent = n2;
  /*
               r
              / \
             n1  n2
```

```
                  /  \
                 n3  n4
  */
  MyBinaryTree t(0, r);
  // t.size2(r) -> 5
  std::cout << "t.size2(r)=" << t.size2(r) << std::endl;
  // t.size(n1) -> 1
  std::cout << "t.size2(n1)=" << t.size2(n1) << std::endl;
  // t.size(n2) -> 3
  std::cout << "t.size2(n2)=" << t.size2(n2) << std::endl;
  // t.size(n3) -> 1
  std::cout << "t.size2(n3)=" << t.size2(n3) << std::endl;
  // t.size(n3) -> 2
  std::cout << "t.size2(n4)=" << t.size2(n4) << std::endl;

}
```

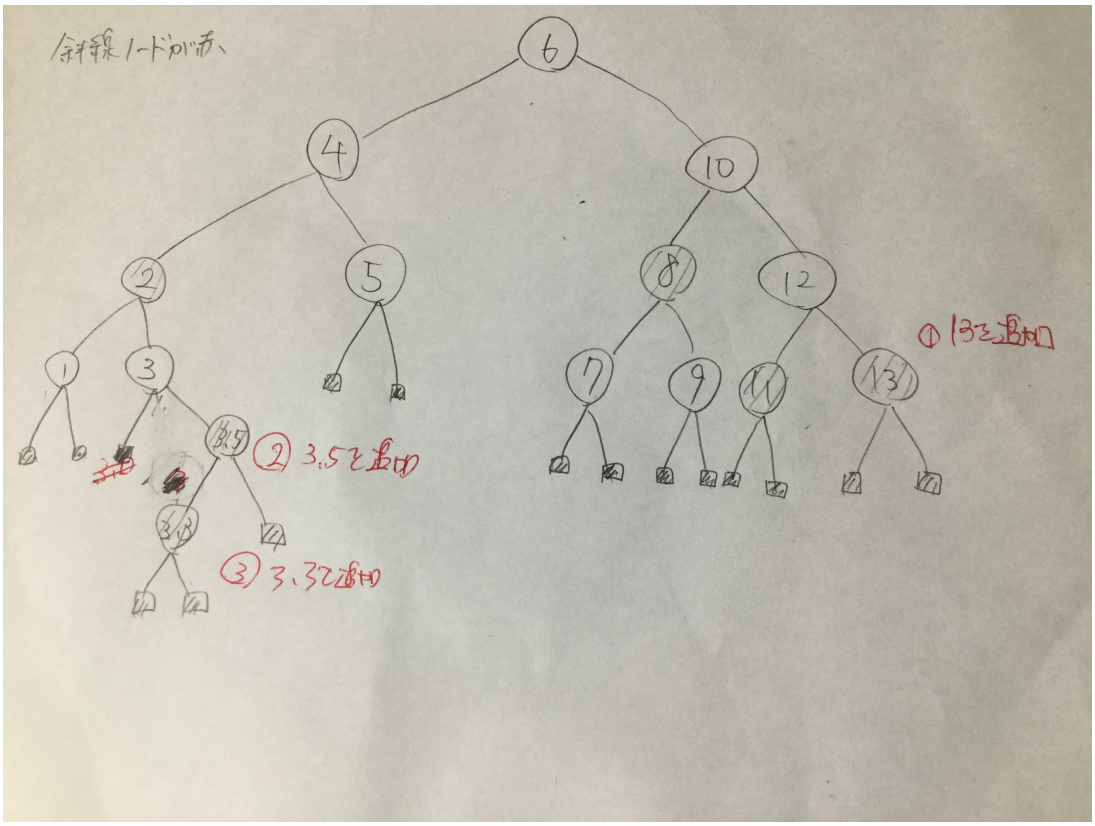
実行結果

```
 ~/m/cpp g++ -Wall -o checkEx6_4 checkEx6_4.cpp
 ~/m/cpp  ./checkEx6_4
r =
0x7fec72501010,n1=0x7fec724011d0,n2=0x7fec724011f0,n3=0x7fec72401210,n4=0x
7fec72401230
t.size2(r)=5
t.size2(n1)=1
t.size2(n2)=3
t.size2(n3)=1
t.size2(n4)=1
```

考察

size2(Node *u)において、uが葉である場合とそうでない場合を分けて考えた。 uが葉である場合(if ((a->left == nil) && (a->right == nil)))の時、サイズは1だから1を返す。uが葉でない場合は、再帰を使わずに木全体のサイズを求めるsize2()を参考にした。size2()では根rから探索を始めていたが、size2(Node *u)ではuから始めた。それに伴い、prev = u->parentとした。後のループはsize2()と同様である。左の枝から順にノードをカウンタnを用いて数えていけば求めるsizeが求まる。実行結果を見ても正しいsizeを返していることがわかる。

Ex 9-2


斜線ノードが赤、
6
4
10
2
5
8
12
1
3
7
9
11
13
3,5
3
① 13と通知
② 3,5と通知
③ 3,3と通知