進捗管理・報告(2020/9/22)

1. 現在取り組んでいること
　可逆コンピューティングの研究

2. 進捗状況
　・可逆コンピューティングについてと、この分野で行われてきたことの調査

3. 前回からの進捗
　・ROOPLPP で深さ優先探索を実装した

```
<!-- ここから -->
class Node
    Node left
    Node right
    Node parent
    int value

    method setValue(int val)
        value ^= val

    method setLeftChild(Node node)
        copy Node node left

    method setRightChild(Node node)
        copy Node node right

    method setParent(Node node)
        copy Node node parent

class Tree
    Node root

    method setRoot(Node node)
        copy Node node root

    method step(Node cur, Node pre)
        local Node next = nil
        if cur.parent=pre then
            if cur.left=nil && cur.right=nil then
                pre <=> next
            else
                if cur.left=nil then
                    copy Node cur.right next
```

```
            else
                copy Node cur.left next
            fi cur.left=nil
            uncopy Node cur.parent pre
        fi cur.left=nil && cur.right=nil
    else if cur.right!=nil then
        if cur.right=pre then
            copy Node cur.parent next
            uncopy Node cur.right pre
        else
            copy Node cur.right next
            uncopy Node cur.left pre
        fi cur.parent=next
    else
        copy Node cur.parent next
        uncopy Node cur.left pre
    fi cur.right!=nil
    fi (next.parent=cur && (cur.left=next || cur.left=nil)) || (cur.right=nil && cur.left=nil)
    pre <=> cur
    cur <=> next
    delocal Node next = nil

method search(int key, Node result)
    local Tree tree = nil
    new Tree tree
    local Node pre = nil
    copy Node root pre
    from pre=root do
        skip
    loop
        call tree::step(result, pre)
    until result=root || result.value=key
    if result!=root then
        uncopy Node result.parent pre
    else
        uncopy Node root.left pre
    fi result!=root
    delocal Node pre = nil
    delocal Tree tree = nil
```

```
class Program
    Tree tree
    Node dummy
    Node node1
    Node node2
    Node node3
    Node node4
    int dummyValue
    int node1Value
    int node2Value
    int node3Value
    int node4Value
    int key
    Node result

    method main()
        new Node dummy
        new Node node1
        new Node node2
        new Node node3
        new Node node4
        dummyValue -= 1
        node1Value += 11
        node2Value += 22
        node3Value += 33
        node4Value += 44
        call dummy::setValue(dummyValue)
        call node1::setValue(node1Value)
        call node2::setValue(node2Value)
        call node3::setValue(node3Value)
        call node4::setValue(node4Value)
        call node4::setParent(node2)
        call node3::setParent(node2)
        call node2::setParent(node1)
        call node2::setLeftChild(node3)
        call node2::setRightChild(node4)
        call node1::setParent(dummy)
        call node1::setLeftChild(node2)
        call dummy::setLeftChild(node1)
        new Tree tree
        key += 55
        copy Node node1 result
        call tree::setRoot(dummy)
        call tree::search(key, result)
```

ROOPLPP をアルゴリズムの表現に使ってみての感想

・Janus で木構造を表現していた時は、ノードを配列のインデックスで表しており、コードを見ただけでは何をしているのが分かりにくかった

→ node.parent のように記述できるようになり、見通しがよくなった

難しかったところ

・instance::method のように method を呼ぶ必要がある

→ インスタンスのメソッド中で自身のインスタンスのメソッドを call できない

→ php のように Class::method や this->method のように呼べたらいいなと

・エラーメッセージがないため、ある程度以上のプログラムでエラーが出ると大変

4. 今後の課題

・Analyzing Trade-offs in Reversible Linear and Binary Search Algorithms で提案されている線形探索の問題点を発見し、改善案を提案する。

・木構造の幅優先探索に、深さ優先探索と同じ考え方を適用できるかどうか確かめる

・木構造の観点からより一般的な考察を述べる

5. その他

・ROOPLPP インタープリタ http://153.127.69.31/