# 進捗管理・報告(2020/9/29)

**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**
　・メモリ効率の良い可逆深さ優先探索の幅優先探索への応用を検討した
https://docs.google.com/document/d/1rNExi09gGYMPBwhGATQzmcRqm4aJ8nLD-bXHmKFY0kI/edit?usp=sharing

　・実装中のプログラム
　　Tree クラスの breadthFirstSearch メソッドのループをうまく実装できず悩み中です。

```
<!-- プログラム -->
class Node
    Node left
    Node right
    Node parent
    int value

    method constructor(int val, Node left, Node right, Node parent)
        call setValue(val)
        call setLeftChild(left)
        call setRightChild(right)
        call setParent(parent)

    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

    method computeDepth(int depth)
        if parent=nil then
            skip
```

```
        else
            depth += 1
            call parent::computeDepth(depth)
        fi parent=nil

class Tree
    Node root

    method setRoot(Node node)
        copy Node node root

    // 探索中のノードとその直前に探索したノードから、次に探索するノードを決定する
    // また、探索はノードの深さによって制限される
    method step(Node cur, Node pre, int exploringLimitDepth)
        local Node next = nil
        local int curDepth = nil
        call cur::computeDepth(curDepth)
        if cur.parent=pre then
            // 通常の判定に加えて、次の判定を追加する
            // 現在探索中のノードの深さが探索深さ制限を超えていたら
            // (>= にしているのは、exploringLimitDepthの値が0でも対応できるようにするため)
            if (cur.left=nil && cur.right=nil) || curDepth>=exploringLimitDepth then
                // 次に探索するノードは直前に探索したノード(親)
                pre <=> next
            else
                if cur.left=nil then
                    // 次に探索するノードは右の子
                    copy Node cur.right next
                else
                    // 次に探索するノードは左の子
                    copy Node cur.left next
                fi cur.left=nil
                uncopy Node cur.parent pre
            fi (cur.left=nil && cur.right=nil) || curDepth>=exploringLimitDepth
        else if cur.right!=nil then
            if cur.right=pre then
                // 次に探索するノードは親
                copy Node cur.parent next
                uncopy Node cur.right pre
            else
                // 次に探索するノードは右の子
                copy Node cur.right next
                uncopy Node cur.left pre
            fi cur.parent=next
        else
            // 次に探索するノードは親
```

```
            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) ||
curDepth>=exploringLimitDepth
        uncall cur::computeDepth(curDepth)
        delocal int curDepth = nil
        pre <=> cur
        cur <=> next
        delocal Node next = nil

    // O(1)の可逆深さ優先探索を行う
    // resultの初期値はroot.leftである必要がある
    // また、探索はノードの深さによって制限される
    // 例：exploringLimitDepthの値が2の場合
    //       → 深さ2以上のノードは探索されない
    method depthFirstSearch(int key, Node result, int exploringLimitDepth)
        local Node pre = nil
        copy Node root pre
        from pre=root loop
            call step(result, pre, exploringLimitDepth)
        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

    method breadthFirstSearch(int key, Node result)
        local int exploringLimitDepth = nil
        exploringLimitDepth += 1
        local Node node = nil
        copy Node root.left node
        call depthFirstSearch(key, node, exploringLimitDepth)
        if node!=root then
                copy Node node result
                uncopy Node result node
            else
                uncopy Node root node
        fi result.value=key
        delocal Node node = nil
        delocal int exploringLimitDepth = 1

        local int exploringLimitDepth = nil
        exploringLimitDepth += 2
```

```
        local Node node = nil
        copy Node root.left node
        call depthFirstSearch(key, node, exploringLimitDepth)
        if node!=root then
            copy Node node result
            uncopy Node result node
        else
            uncopy Node root node
        fi result.value=key
        delocal Node node = nil
        delocal int exploringLimitDepth = 2

        local int exploringLimitDepth = nil
        exploringLimitDepth += 3
        local Node node = nil
        copy Node root.left node
        call depthFirstSearch(key, node, exploringLimitDepth)
        if node!=root then
            copy Node node result
            uncopy Node result node
        else
            uncopy Node root node
        fi result.value=key
        delocal Node node = nil
        delocal int exploringLimitDepth = 3

class Program
    Tree tree
    Node dummy
    Node node1
    Node node2
    Node node3
    Node node4
    int dummyValue
    int key
    Node result

    method main()
        new Node dummy
        new Node node1
        new Node node2
        new Node node3
        new Node node4
        dummyValue -= 1
        call dummy::constructor(dummyValue, node1, nil, nil)
        call node1::constructor(11, node2, nil, dummy)
```

```
call node2::constructor(22, node3, node4, node1)
call node3::constructor(33, nil, nil, node2)
call node4::constructor(44, nil, nil, node2)

new Tree tree
call tree::setRoot(dummy)

key += 44
call tree::breadthFirstSearch(key, result)
```

```
<!-- うまくいかないループ -->
method breadthFirstSearch(int key, Node result)
        local int exploringLimitDepth = nil
        from exploringLimitDepth=0 loop
            exploringLimitDepth += 1
            local Node node = nil
            copy Node root.left node
            call depthFirstSearch(key, node, exploringLimitDepth)
            if node!=root then
                copy Node node result
                uncopy Node result node
            else
                uncopy Node root node
            fi result.value=key
            delocal Node node = nil
        until result.value=key

        local int resultDepth = nil
        call result::computeDepth(resultDepth)
        delocal int exploringLimitDepth = resultDepth
        uncall result::computeDepth(resultDepth)
        delocal int resultDepth = nil
```

**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