# 進捗管理・報告(2021/3/2)

**1.** 現在取り組んでいること
可逆コンピューティング
ー可逆アルゴリズム

**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
・可逆アルゴリズムの調査

**3.** 前回からの進捗
・UNIVERSITY OF COPENHAGEN
Reversible Graph Algorithms を調査

・優先度付きキュー**(min**ヒープ**)**
ー時間計算量：非可逆な優先度付きキューと同じ
ーゴミ出力：decreasekey, extractmin, insertにそれぞれ定数量のゴミ

・UNIVERSITY OF COPENHAGENの実装(左)
・UNIVERSITY OF COPENHAGENを先生が書き換えたもの(右)

```
procedure minheapify(int A[], int heapsize, int i,
                     int heapgarbage[], int garbagecounter)
    local int left = 0, int right = 0, int min = 0
    call left(i, left)
    call right(i, right)

    // Find the smallest note between i and the children of i
    if left <= heapsize && A[left] < A[i] then
        min += left
        if right <= heapsize && A[right] < A[min] then
            min -= left
            min += right
        fi right <= heapsize && A[right] < A[left]
    else
        min += i
        if right <= heapsize && A[right] < A[min] then
            min -= i
            min += right
        fi right <= heapsize && A[right] < A[i]
    fi left <= heapsize && A[left] < A[i]

    // Save it in the garbage
    heapgarbage[garbagecounter] += min
    garbagecounter += 1

    // If either of the children is smaller, switch i with the child and
    // call minheapify recursively
    if min != i then
        A[i] <=> A[min]
        call minheapify(A, heapsize, min, heapgarbage, garbagecounter)

        if left = heapgarbage[garbagecounter-1] then
            min -= left
        else
            min -= right
        fi left = heapgarbage[garbagecounter-1]
    fi min = 0

    garbagecounter -= 1

    // Set the current min to 0
    if i = heapgarbage[garbagecounter] then
        min -= i
    fi i = heapgarbage[garbagecounter]

    uncall right(i, right)
    uncall left(i, left)
    delocal int left = 0, int right = 0, int min = 0
```

最も小さい子を探す
インデックスをminに格納
小さい方のインデックスを保存
minの値を0に
(iが最小だった場合)
インデックスiとminの値を交換.
minについて再帰的に実行.

```
/*
 * minheapify ensure the minimum heap property of a tree, where both subtrees of
 * the node at index i has the minimum heap property, yet the node at index i
 * might be larger
 *
 * Parameters:
 * int A[]              A minimum heap array with an inconsistency at index i
 * int heapsize         The heaps current size
 * int i                An index for a node in the heap
 * int j                Initially, set to zero
 * int heapgarbage[ size(A)-1]  がいらない バージョン
 * Postcondition:
 * A minimum heap which has no inconsistency at index i and its two subtrees
 * j は元のindex iにあった要素がどこにいったか(ゴミ)
 */
procedure minheapify(int A[], int heapsize, int i, int j)
    // Find the smallest note between i and the children of i
    if (RIGHT(i) <= heapsize && A[RIGHT(i)] < A[i] && A[RIGHT(i)] < A[LEFT(i)]) ||
       (LEFT(i) <= heapsize && A[LEFT(i)] < A[i]) then
        local int min = LEFT(i) <= heapsize && A[LEFT(i)] < A[i] ? LEFT(i) : RIGHT(i)
        A[i] <=> A[min]
        call minheapify(A, heapsize, min, j)
        call double(j)
        j ^= LEFT(i) = min ? 0 : 1 // set the lowest bit
        delocal int min = j % 2 = 0 ? LEFT(i) : RIGHT(i)
    else
        j ^= 1
    fi j != 1

procedure double(int x)
    local int tmp = x * 2
        tmp <=> x
    delocal int tmp = x / 2
```

iがる左のどちら移動したか

A[RIGHT(i)] > A[LEFT(i)] の条件を追加する必要?

2 xor 1 = 3
3 xor 1 = 2
4 xor 1 = 5
5 xor 1 = 4.
2 xor 0 = 2
3 xor 0 = 3

・違い

ーUNIVERSITY OF COPENHAGEN

・位置を交換したノードについて、直前の場所をインデックスとして十分な大きさ　　の配列(size(A)-1)に格納

・必要なゴミ：大きさsize(A)-1のint型の配列, そのインデックスを表すint型の変数

ー先生の実装

・int型の変数1つを用いて、位置を交換したノードが左右どちらのノードと交換さ　　れたのかを表現

・必要なゴミ：int型の変数

**4.** 今後の課題