進捗管理・報告(2020/5/14)

1. 現在取り組んでいること

・流れの可視化（卒業論文の続き）研究
・可逆コンピューティング研究

2.進捗状況

流れの可視化

・研究内容をまとめたHTMLの作成
（http://tetsuo.jp/lab/seminar/2018/tfda.html）
・卒業論文の修正

可逆コンピューティング

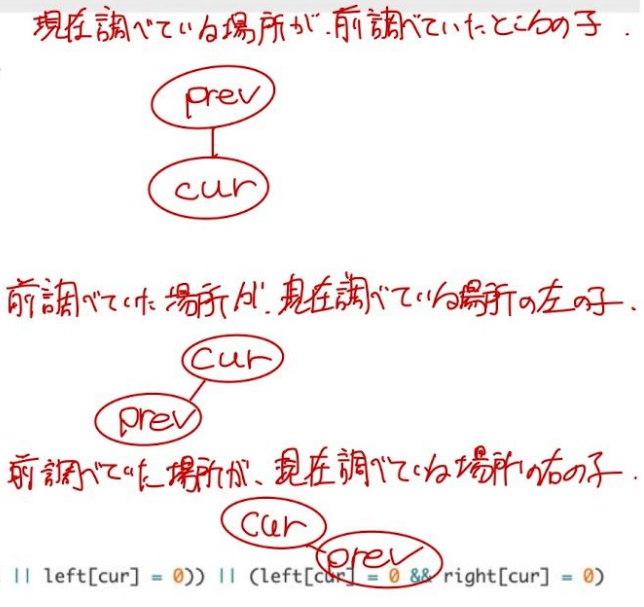
・可逆コンピューティングについてと、この分野で行われてきたことの調査
（https://www.overleaf.com/project/5e731b5059008200012799bd）
・増田先輩の実装した可逆深さ優先探索のアルゴリズムを解析した

3.前回からの進捗

可逆コンピューティング

・先生の実装した可逆深さ優先探索アルゴリズムについて理解し、問題点を修正した

```
5  // a step of a tree traversal
6  procedure step(int left[], int right[], int parent[], int cur, int prev)
7    local int next = 0
8    if parent[cur] = prev then
9      if left[cur] = 0 && right[cur] = 0 then
10       next <=> prev        // zero clear prev
11     else if left[cur] = 0 then
12         next ^= right[cur]
13         prev ^= parent[cur]  // zero clear
14       else
15         next ^= left[cur]
16         prev ^= parent[cur]  // zero clear
17     fi left[cur] = 0
18   fi left[cur] = 0 && right[cur] = 0
19   else if left[cur] = prev then
20     if right[cur] != 0 then
21       next ^= right[cur]
22       prev ^= left[cur]  // zero clear
23     else
24       next ^= parent[cur]
25       prev ^= left[cur]  // zero clear
26     fi right[cur] != 0
27   else
28     next ^= parent[cur]
29     prev ^= right[cur]  // zero clear
30   fi parent[left[cur]] = cur
31   fi (parent[next] = cur && (left[cur] = next || left[cur] = 0)) || (left[cur] = 0 && right[cur] = 0)
32   prev <=> cur
33   cur <=> next
34   delocal int next = 0
35
36 procedure search(int left[], int right[], int parent[], int cur, int prev, int val[], int k)
37   from prev = 0 loop
38     show(cur)
39     call step(left, right, parent, cur, prev)
40   until val[cur] = k || cur = 0
```

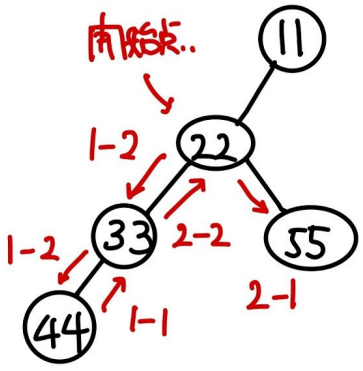


走査手順
1　現在調べている場所が、前調べていたところの子だったら
1-1　左右どちらとも子がない　→　次は親を調べる
1-2　左に子がある　→　次は左の子を調べる
1-3　右に子がある　→　次は右の子を調べる

2　前調べていた場所が、現在調べている場所の左の子
2-1　右の子がある　→　次は右の子を調べる
2-2　右の子がない　→　次は親を調べる

3　前調べていた場所が、現在調べている場所の右の子
3-1　次は親を調べる

例：



赤線：走査順序(キー：55)

1 . 開始する点はmain関数で任意に定める
2 . 左の子を探索
3 . 子を持たない時は戻って右の子を探索

修正点：
・先生のプログラムにはアサーションが常に真になってしまう箇所があった

```
20    if right[cur] != 0 then
21       next ^= right[cur]
22       prev ^= left[cur]  // zero clear
23    else
24       next ^= parent[cur]
25       prev ^= left[cur]  // zero clear
26    fi right[cur] != 0
27  else
28    next ^= parent[cur]
29    prev ^= right[cur]  // zero clear
30  fi parent[left[cur]] = cur
31  fi (parent[next] = cur && (left[cur] = next || left[cur] = 0)) || (left[cur] = 0 && right[cur] = 0)
32  prev <=> cur
33  cur <=> next
34  delocal int next = 0
```

右の子を調べて、上へ戻ったときて、
このアサーションが真になる。

よって、プログラムを次のように変更した

```
1  procedure step(int left[], int right[], int parent[], int cur, int prev)
2    local int next = 0
3    if parent[cur] = prev then
4      if left[cur] = -1 && right[cur] = -1 then
5        next <=> prev        // zero clear prev
6      else if left[cur] = -1 then
7          next ^= right[cur]
8          prev ^= parent[cur]  // zero clear
9        else
10         next ^= left[cur]
11         prev ^= parent[cur]  // zero clear
12     fi left[cur] = -1
13   fi left[cur] = -1 && right[cur] = -1
14   else if right[cur] != -1 then
15     if right[cur] = prev then
16       next ^= parent[cur]
17       prev ^= right[cur]  // zero clear
18     else
19       next ^= right[cur]
20       prev ^= left[cur]  // zero clear
21     fi parent[cur] = next
22   else
23     next ^= parent[cur]
24     prev ^= left[cur]  // zero clear
25   fi right[cur] != -1
26   fi (parent[next] = cur && (left[cur] = next || left[cur] = -1)) || (left[cur] = -1 && right[cur] = -1)
27   prev <=> cur
28   cur <=> next
29   delocal int next = 0
30
31 procedure search(int left[], int right[], int parent[], int cur, int prev, int val[], int k)
32   from prev = -1 loop
33     show(cur)
34     call step(left, right, parent, cur, prev)
35   until val[cur] = k || cur = -1
```

ここは同じ．

if 現在いる位置に右の子はあるかどうか調べる．
if 右の子が前調べていたところ． 2-1
次は．親を調べる．
else 次は．右の子を調べる． 2-2
else 次は．親を調べる．

親が次調べる場所．

右の子があるか．

走査手順(1は変更なし)

1　現在調べている場所が、前調べていたところの子だったら

1-1　左右どちらとも子がない　→　次は親を調べる

1-2　左に子がある　→　次は左の子を調べる

1-3　右に子がある　→　次は右の子を調べる

2　現在調べている位置に右の子がある時

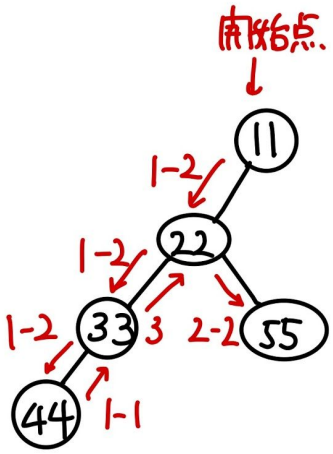
2-1　右の子が前調べていたところ　→　次は親を調べる

2-2　右の子が前調べていないところ　→　次は右の子を調べる

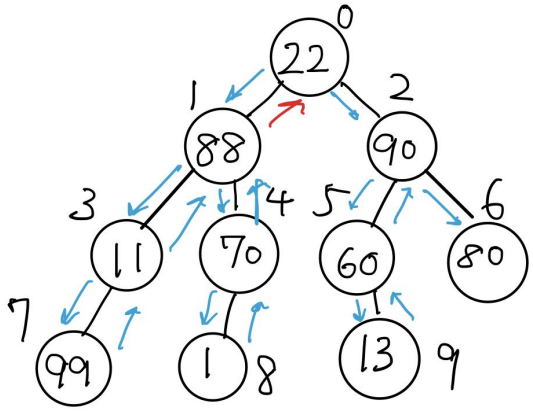
3　現在調べている位置に右の子がない時

3-1　次は親を調べる

例：



赤線：走査手順(キー：55)



矢印：走査手順(キー：80)

・赤の矢印が、前のアルゴリズムではエラーになった箇所

この木の探索結果(キー：80)

cur = 0
cur = 1
cur = 3
cur = 7
cur = 3
cur = 1
cur = 4
cur = 8
cur = 4
cur = 1
cur = 0
cur = 2
cur = 5
cur = 9
cur = 5
cur = 2
cur = 6
k = 80
prev = 2

このアルゴリズムの特徴：

・探索が成功した場合その時点で再帰を終了する

・prevとcurの二つの変数を使うことで場所を一意に定める

・スタックや再帰を使わず場所を表す変数のみを使用するので、空間計算量はO(1)

・与えられた入力以外の出力ゴミ出力量は0

→ただし、先輩のプログラムと違い変数の数が増えた

・時間計算量はO(n)では無い...?

4.今後の課題

可逆コンピューティング

・調査の続き

Reversibility for efficient computing　という論文が、可逆コンピューティングについて詳しくまとめているので、引き続きこれを読む。

Energy-Efficient Algorithms　という論文を調査する。

・空間計算量がO(1)のアルゴリズムについて、改善できる点を考えてみる

→prevは消すことができる...?、時間計算量をO(n)にできる...?