New Reversible Computing Algorithms for Shortest Paths Problem

1. 研究分野

可逆コンピューティング
ー可逆アルゴリズム

2. 目的

時間・空間計算量の観点で効率の良い可逆最短経路アルゴリズムを提案する

3. 背景

可逆アルゴリズム効率化の必要性
衛生的で健全なアルゴリズムの開発

4. アプローチ

・heap
・Bellman-Ford
・Dijkstra
・Floyd-Warshall

5. 結果

・min_heap

この論文.

```
procedure minheapify(int A[], int heapsize, int i, int lastindex, int pow)
    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]
    //if i isn't the smallest element, switch and call minheapify
    if min != i then
        A[i] <=> A[min]
        call minheapify(A, heapsize, min, lastindex,pow)
    else
        lastindex += min
    fi min != i
    //set the current min to 0
    if min != i then
        if lastindex / 2 = i && lastindex % 2 = 1 then
            min -= 1
        fi lastindex / 2 = i && lastindex % 2 = 1
        if lastindex % 2 = 0 then
            min -= lastindex / pow
        else
            min -= (lastindex-1) / pow
        fi lastindex % 2 = 0
        local int tmp = pow * 2
            tmp <=> pow
        delocal int tmp = pow / 2
    fi min = 0
    if i = lastindex then
        min -= lastindex
    fi i = lastindex
    uncall right(i, right)
    uncall left(i, left)
    delocal int left = 0, int right = 0, int min = 0
```

init:0 / init:1 / 最終的なインデックス

卒論

```
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
```

- 配列を用意しなくても、2つの値で逆移を特定可能。
- 先生の方法に似ている。

・Bellman-Ford
→ 論文にはrelaxの部分のみ示されている(Bellmam-Fordアルゴリズムは、relax以外に、重みの初期化と負閉路が存在することを確かめることが必要。この論文のrelaxは2~4行目の処理を行う)
(参考：アルゴリズムイントロダクション第二巻p249)

```
BELLMAN-FORD(G, w, s)
1  INITIALIZE-SINGLE-SOURCE(G, s)
2  for i = 1 to |G.V| − 1
3      for each edge (u, v) ∈ G.E
4          RELAX(u, v, w)
5  for each edge (u, v) ∈ G.E
6      if v.d > u.d + w(u, v)
7          return FALSE
8  return TRUE
```

→ INITIALIZE-SINGLE-SOURCEにO(V)時間、O(V)のゴミ出力が必要
→ 5〜8行目にはゴミ出力は必要ない

→ relaxは緩和を行う関数(参考：アルゴリズムイントロダクション第二巻p246)

```
RELAX(u, v, w)
1  if v.d > u.d + w(u, v)
2      v.d = u.d + w(u, v)
3      v.π = u
```

以下のrelaxは、O(VE)時間でO(VE)のゴミを出力する
大きさVEの配列garbage[]と、その場所を示すgarbagecounterが必要

```
1  procedure relax(int G[], int nodes[], int garbage[], int garbagecounter)
2      local int i = 0
3      from i = 0 do
4          local int j = 0, int k = 0
5          from j = 0 do
6              if nodes[G[j][0]][2] >= nodes[k][2] + G[j][1] then
7                  nodes[G[j][0]][1] <=> garbage[garbagecounter]
8                  nodes[G[j][0]][1] += k
9                  garbagecounter += 1
10                 nodes[G[j][0]][2] <=> garbage[garbagecounter]
11                 garbagecounter += 1
12                 nodes[G[j][0]][2] += nodes[k][2] + G[j][1]
13             fi nodes[G[j][0]][2] = nodes[k][2] + G[j][1]
14             j += 1
15             if (k < size(nodes) - 1) && (j = nodes[k+1][0]) then
16                 k += 1
17             fi j = nodes[k][0]
18         until j = size(G)
19         delocal int j = size(G), int k = size(nodes) - 1
20         i += 1
21     until i = size(nodes) - 2
22     delocal int i = size(nodes) - 2
```

6. 有用性

7. 限界・短所

8. 次に何を読めばいいか？