進捗管理・報告(2021/3/29)

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
　New Reversible Computing Algorithms for Shortest Paths Problem の Bellman-Ford アルゴリズムについて、緩和部分(relax)の効率化を考える

```
1  procedure ralax(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
```

図1. relax

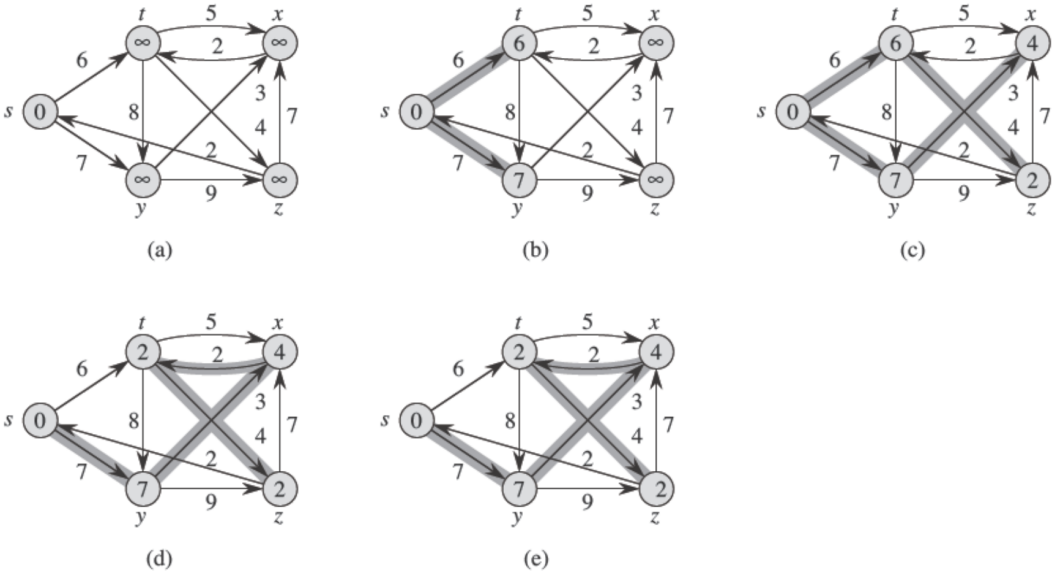

図2. 振る舞い

・案1

無限を覚えるコストが大きい → 十分に大きな値(例えばint型の最大値)がゴミにたまる

→ 最初にグラフを探索して、ノードの最大値を無限の代わりに使用する

結果

ノードの最大値だけでは難しそう

→ sを始点として Bellman-Ford アルゴリズムを実行した直後のグラフについて、zを始　　点としてBellman-Ford アルゴリズムを実行する

→ グラフの最大値は7なので、z以外の全てのノードの値は0になる

→ relax(x, z, 7)において、x.d > z.d + 7 が False になってしまう

＋ 各ノードにおいて、たどったかたどってないかを表す1ビットの値があれば可能

→ 各ノードを初めて探索するときは、各ノードの値を上書きする

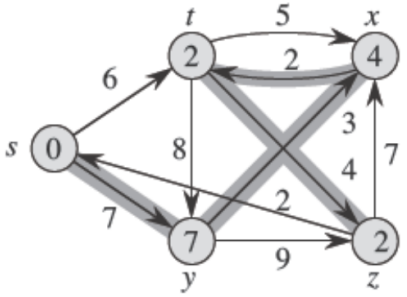

図3. sを始点として Bellman-Ford アルゴリズムを実行

・案2

ノードの値が変わるときに、元々のノードが保持する値と、エッジの値の差をとる

→ 現在：値が上書きされる際に、直前の先行点・ノードの値をゴミとして保存

結果

全てのエッジの値が異なるなら可能

→ 下の図のようなグラフの場合、どこから来たのかを一意に決定できない

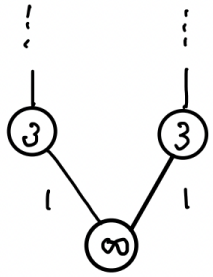

図4. できない場合のグラフ例

4. 今後の課題

New Reversible Computing Algorithms for Shortest Paths Problemを引き続き調査