可逆探索アルゴリズム

M2018SE009　増田大輝

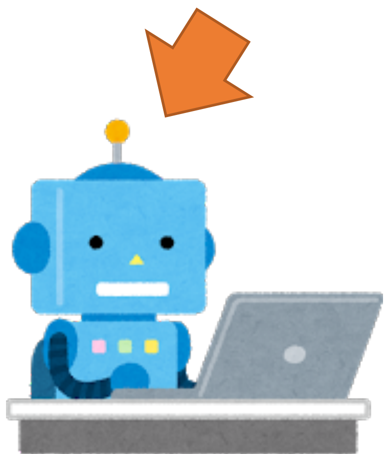
指導教員　　横山哲郎

目次

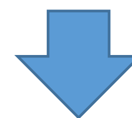
1. はじめに
2. 研究課題
3. 関連研究
4. 線形探索アルゴリズムの可逆化
5. 木構造の探索アルゴリズムの可逆化
6. 今後の課題
7. おわりに

1. はじめに（1/3）

消費エネルギーの拡大



計算機の可逆化により
消費エネルギー効率改善



可逆の塔の実現

可逆アプリケーション
可逆アルゴリズム
可逆プログラミング言語
可逆論理ゲート
物理層

1. はじめに(2/3)

可逆アルゴリズムの解析

複雑なアルゴリズムへの応用

可逆アルゴリズムの知見

今までの研究

効率的な可逆線形探索アルゴリズムの設計・実装[3]

今回の研究内容

可逆探索アルゴリズムの解析

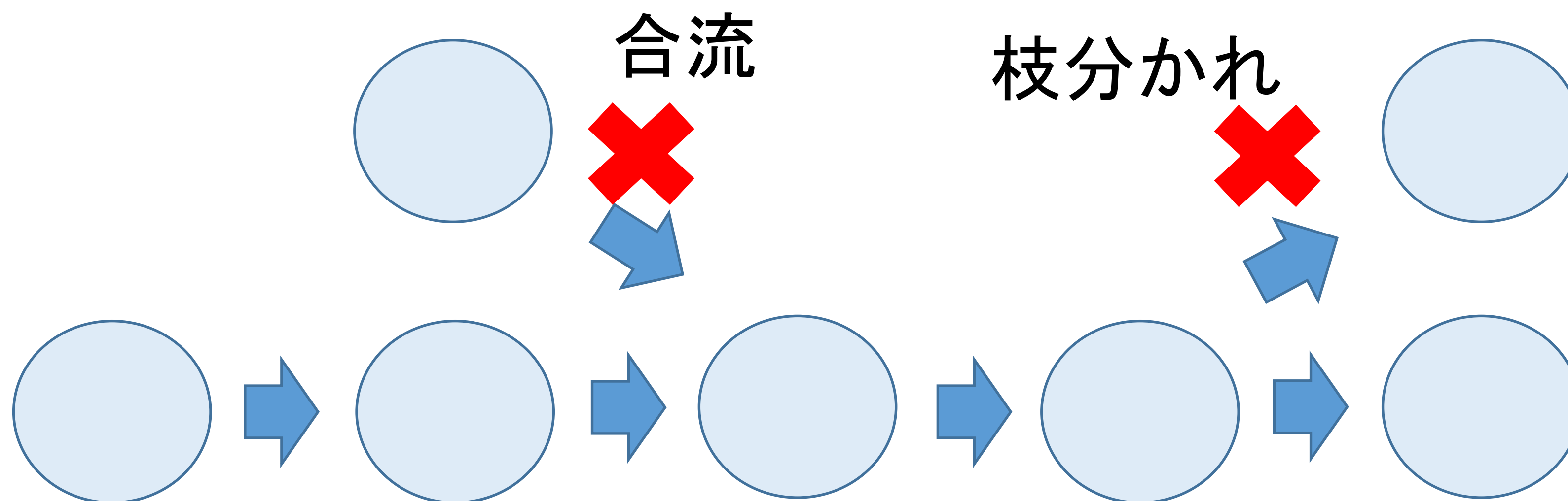
[3]:家崎雄太，水野竣太郎：可逆線形探索，南山大学2017年度卒業論文(2018).

1. はじめに（3/3）

可逆アルゴリズム

可逆という制約を持った計算順序

可逆：　すべての状態が直前と直後の状態を高々1つしか持たないもの

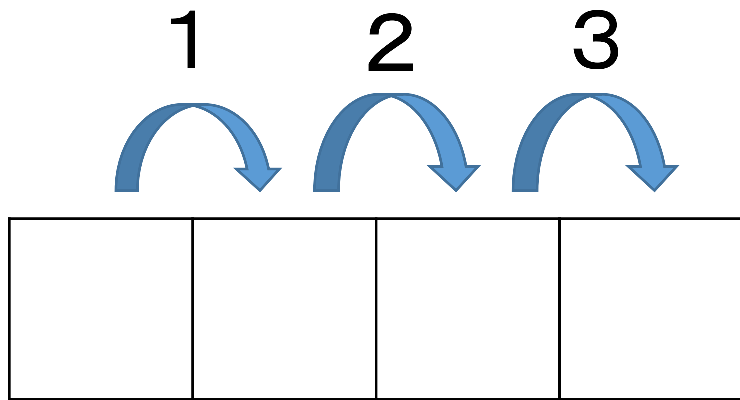


2. 研究課題

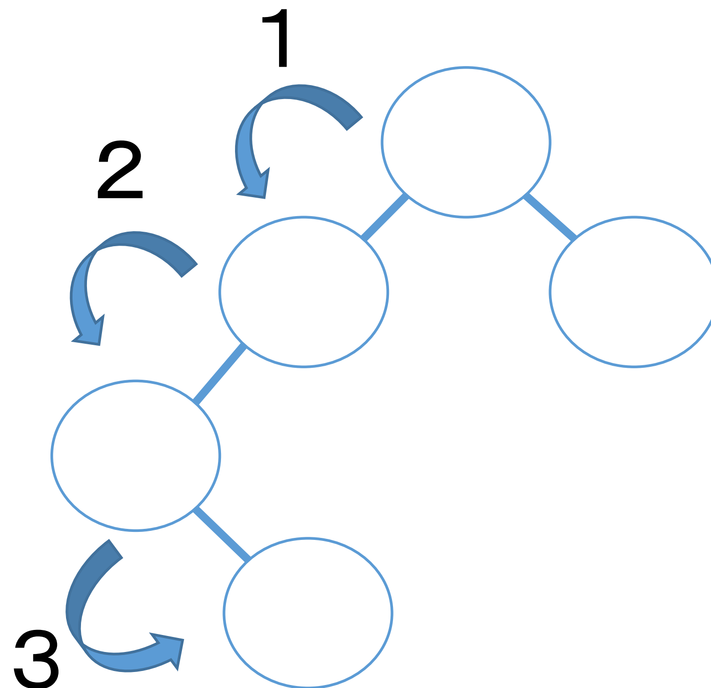
目的：効率的な可逆探索アルゴリズムの設計・実装と解析

可逆化と解析の対象

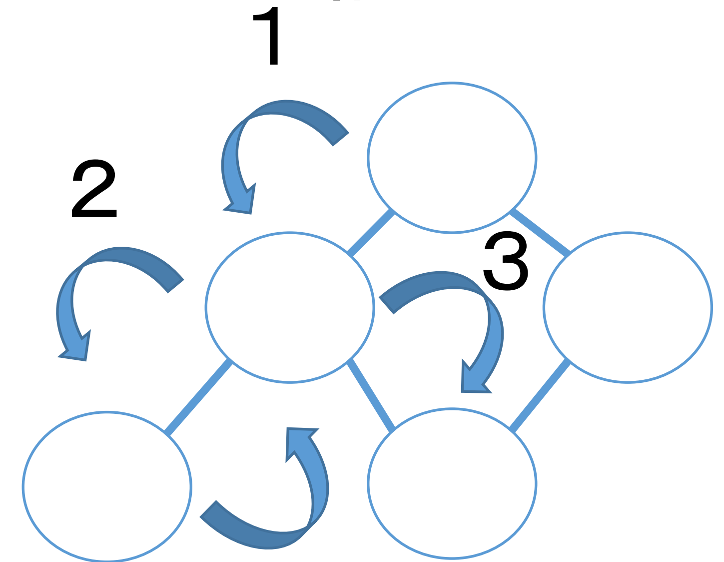
線形探索



木構造の探索

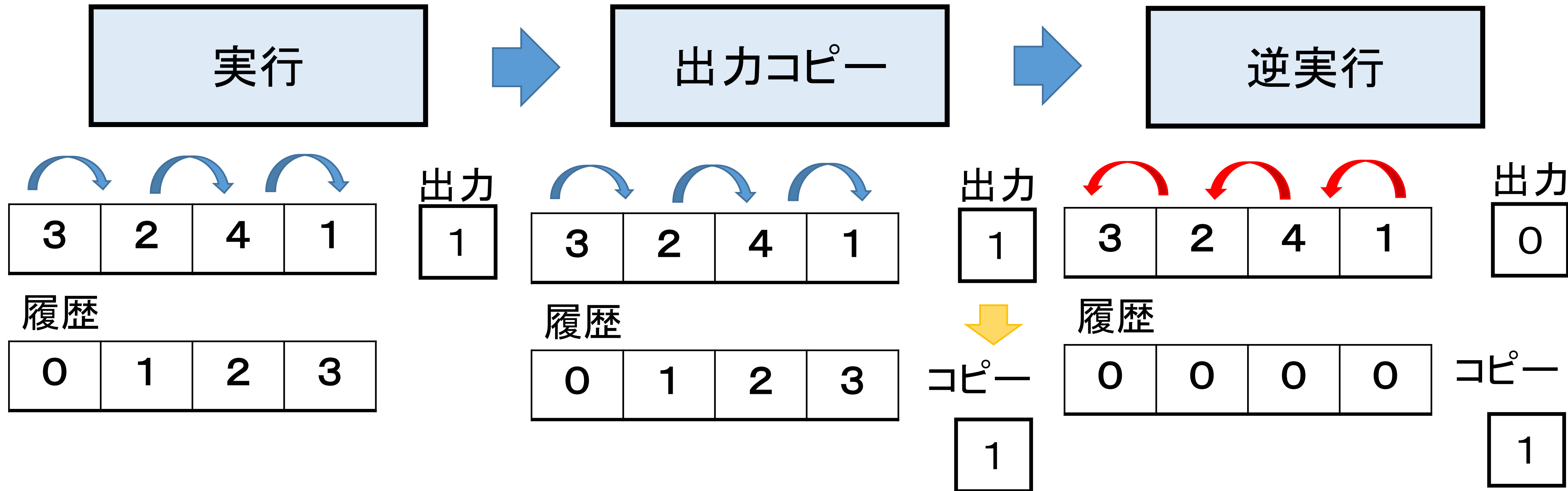


グラフ構造の探索



3. 関連研究

Bennett のアルゴリズムの可逆化の一般解法

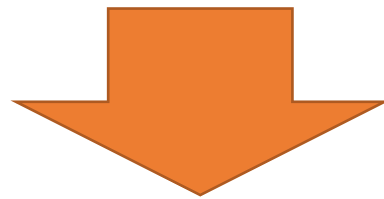


[5]:Bennett, C.H.:Logical reversibility of computation, IBM Journal of Research and Development, vol.17, no.6, pp.525-532(1973).

３．関連研究

可逆ソートアルゴリズム

- 効率的なごみ出力や特殊なコード共有法などの
  可逆プログラミング特有の技術手法の発見
- 空間使用量・パス数・ごみ出力の点で効率的なアルゴリズムの提案



可逆アルゴリズム作成の手順や方法の知見

4. 線形探索アルゴリズムの可逆化

アプローチ

以前の論文[3]の効率的な可逆線形探索を基としたプログラムを解析

視点

- 元の出力以外のメモリ使用量
- 元の出力以外のごみ出力
- 入力ファイルの走査回数

[3]: 家崎雄太, 水野竣太郎: 可逆線形探索, 南山大学2017年度卒業論文(2018).

４．線形探索アルゴリズムの可逆化

結果

（元の出力以外のごみ出力が0となるようなアルゴリズム）

配列

- レコードの構造や入出力の変化により入力の走査回数に変化
- 探索の成否が入力の走査回数に影響

リスト

- データの書き換えの有無によってメモリ使用量に変化

考察

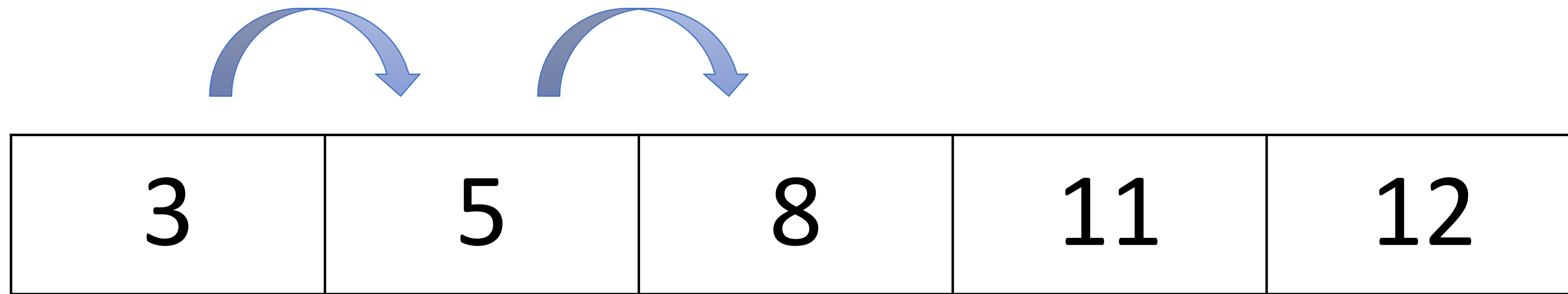
- 可逆アルゴリズム特有の影響を考慮する必要性

構造	出力	整列
配列	レコード位置	有

成功時

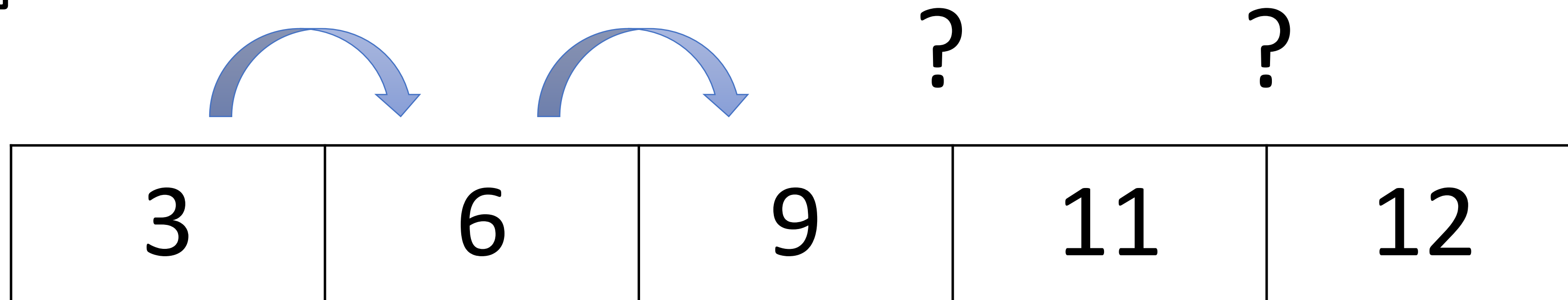
Key = 8

出力



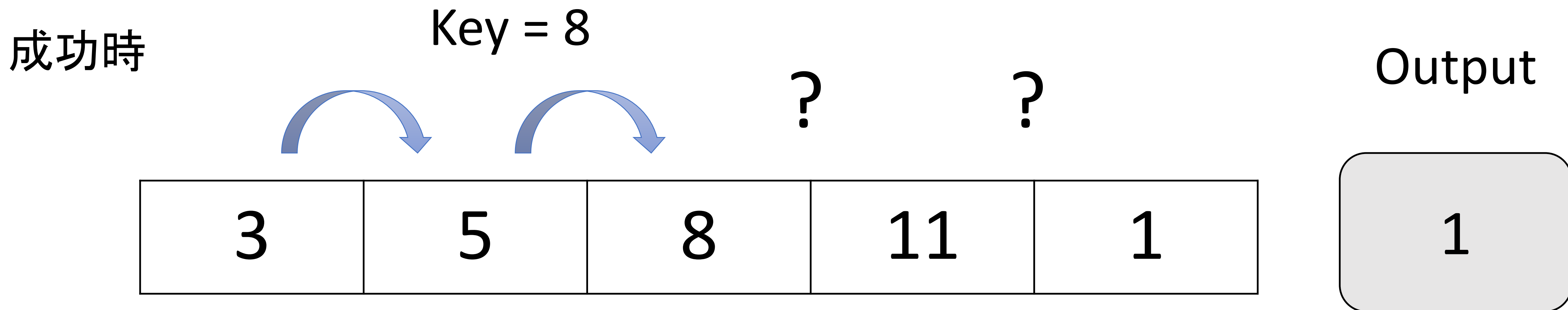
R_3

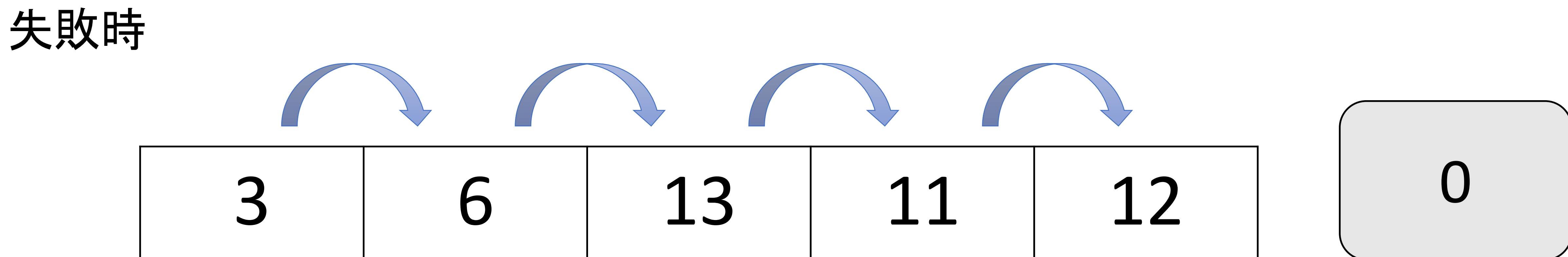
失敗時



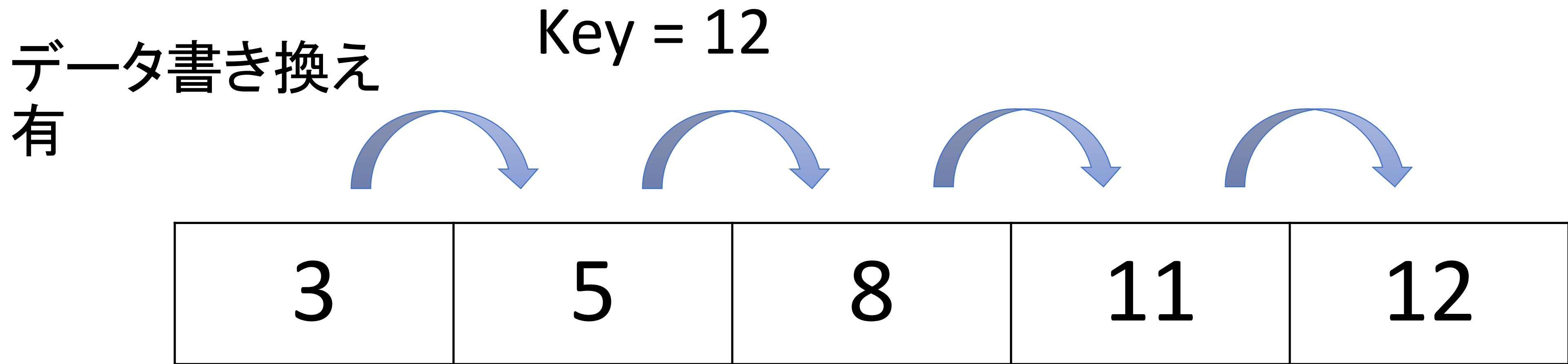
-1

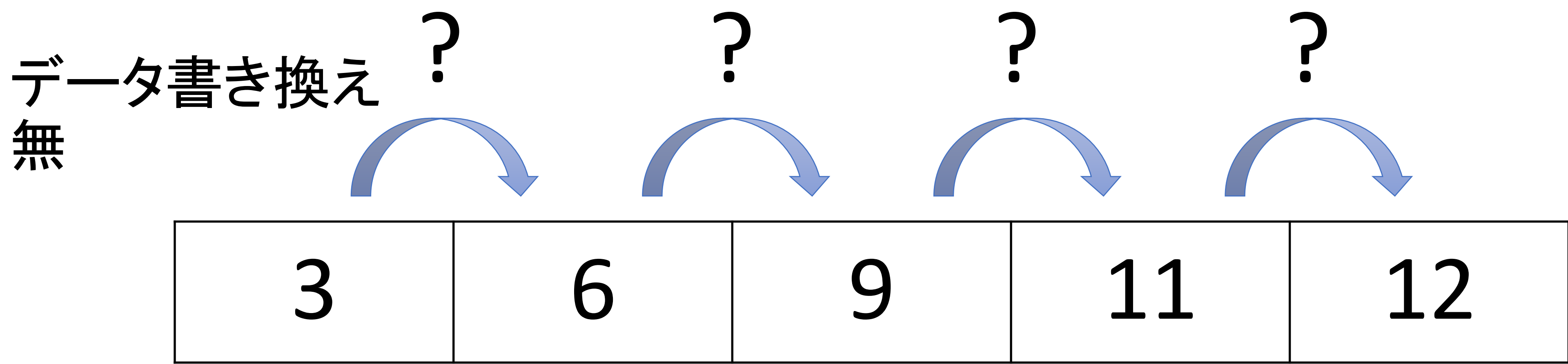
構造	出力	整列
配列	フラグ	無





構造	出力	整列
リスト	フラグ	無





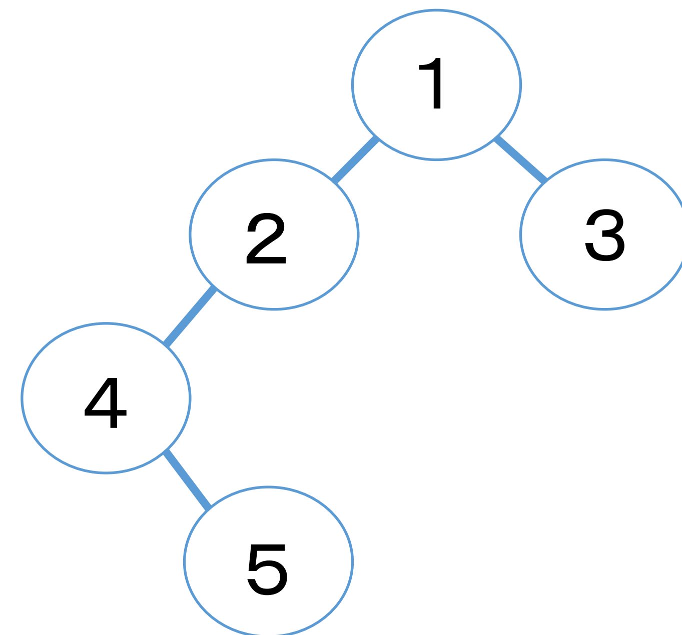
5. 木構造の探索アルゴリズムの可逆化(1/4)

木構造:グラフ構造の中の閉路のない構造

- 各要素は親子関係を持つ

幅優先探索
根に近い節から順に探索
(1,2,3,4,5の順に探索)

深さ優先探索
1. 葉に行きつくまで探索を継続
2. 見探索の節へ戻り探索を再開
(1,2,4,5,3の順に探索)



５．木構造の探索アルゴリズムの可逆化（2/4）

深さ優先探索の可逆化の課題

- 現在の位置情報から次の探索地点が親か子かを一意に定められない

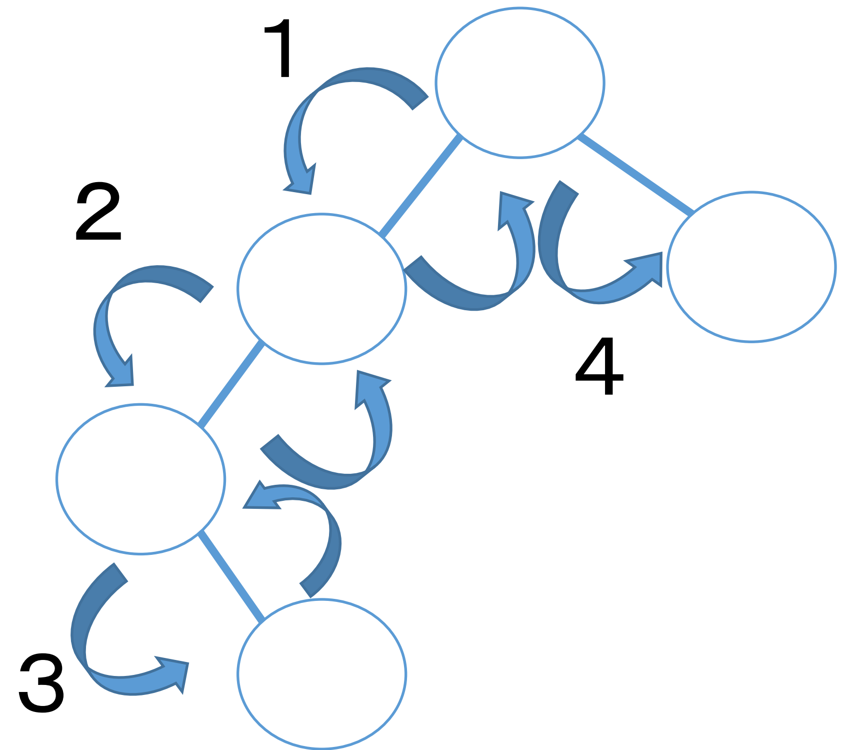
解決法

- 探索した方向を記憶
- 左側の枝を伝って探索

```
procedure func(int in[][], int k, int f, int visit)
    if visit != -1 then
        local int temp = visit
        call func(in, k, f, in[temp][1])
        call func(in, k, f, in[temp][2])
        if in[temp][0] = k then
            f ^= -1
            f ^= temp
        fi in[temp][0] = k
        delocal int temp = visit
    fi visit != -1
```

5. 木構造の探索アルゴリズムの可逆化(3/4)

```
procedure func(int in[][], int k, int f, int visit)
   if visit != -1 then
      local int temp = visit
      call func(in, k, f, in[temp][1])  左の子を探索
      call func(in, k, f, in[temp][2])  右の子を探索
      if in[temp][0] = k then
         f ^= -1
         f ^= temp
      fi in[temp][0] = k
      delocal int temp = visit
   fi visit != -1
```



５. 木構造の探索アルゴリズムの可逆化（4/4）

利点

- 漸近的メモリ使用量、ごみ出力量ともにO(n)からO(1)に改善

欠点

- 漸近的時間計算量がO(n)からΘ(n)に悪化

考察

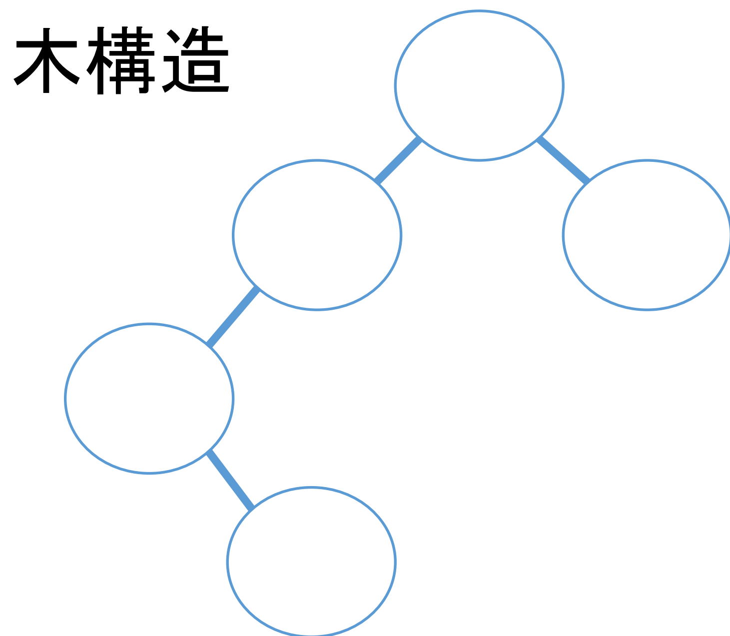
深さ優先探索

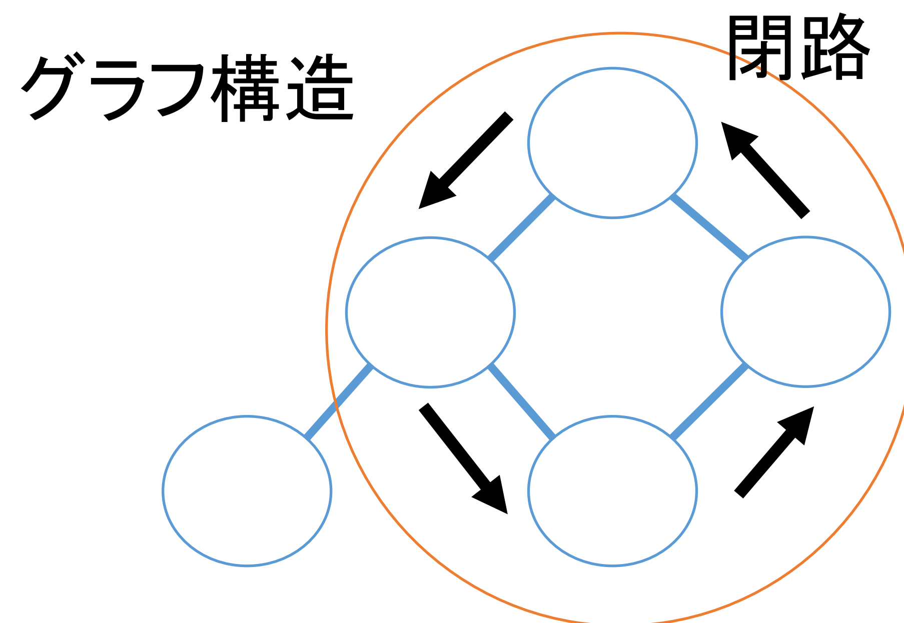
時間的・空間的計算量でトレードオフ関係があることを発見

６. 今後の課題

- グラフ探索アルゴリズムの効率的な可逆化
- グラフ探索アルゴリズムと木構造の探索アルゴリズムを解析

閉路の有無の可逆アルゴリズムに対する影響度





7. おわりに

目的

- 可逆線形探索の解析
- 木構造の深さ優先探索の可逆化
- グラフ探索アルゴリズムの可逆化

成果

- 可逆線形探索のトレードオフ関係を発見
- 木構造の可逆深さ優先探索の効率的な手法を提案

課題

- グラフ探索アルゴリズムの可逆化
- 木構造とグラフ構造の可逆化した場合の影響の解明