可逆線形探索

理工学部ソフトウェア工学科
2014SE024 家崎 雄太　2014SE067 水野 竣太郎

目次

はじめに

背景:

線形探索アルゴリズムの可逆化に関する議論が不十分である

→効率的な可逆線形探索アルゴリズムの模索が必要である

目的:

可逆線形探索の実装を通じて効率的な可逆探索アルゴリズムの模索をする

アプローチ:

Janusで可逆線形探索を実装する

線形探索アルゴリズムのバリエーション

通常の
線形探索

番兵を用いた
線形探索

整列リストを
用いた
線形探索

線形探索の対象となるレコード列

R_1	R_2	R_3	・・・	R_n
K_1	K_2	K_3	・・・	K_n

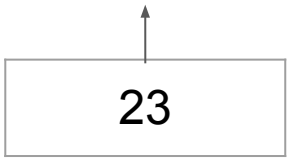
レコード列にあるレコードはそれぞれキーを持っている

探索する時は，入力として

「対象とするレコード列」「レコードの個数」「探したいキー」が必要になる

番兵を用いた線形探索

R_1	R_2	R_3	R_4	R_5	R_6
35	45	64	11	93	23



R_6が持つキー「23」は探したいキー[23]と一致する

→R_6は最後尾（番兵）であるため，元のレコード列中に無かった（失敗した）

可逆化の手段

・埋め込み（Landauer法）

計算過程をゴミ情報として出力

・Bennett法

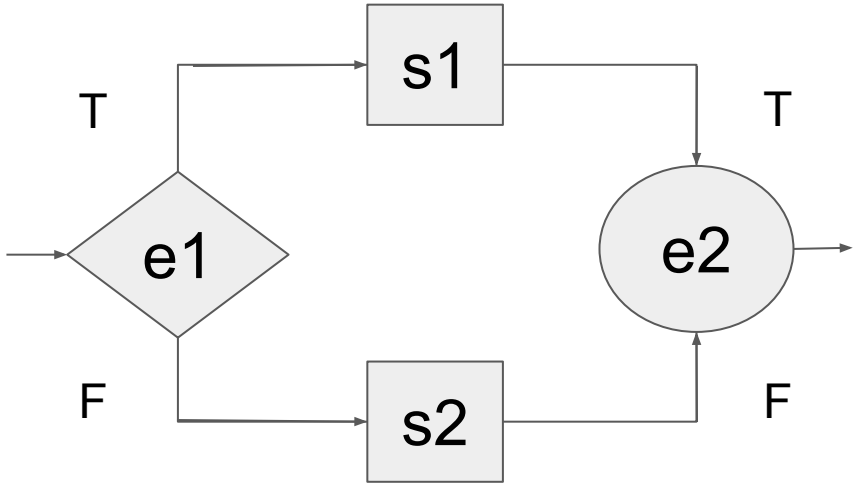
計算過程をゴミ情報としてゴミスタックに格納

関数を逆呼び出しする事でゴミ情報を除去

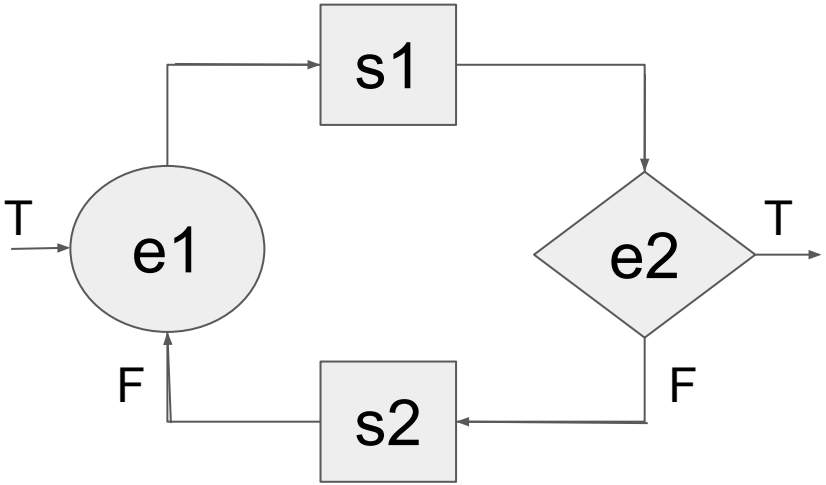
Janus

- 命令型可逆プログラミング言語である
- 可逆性を保証する構文規則を持つ
- **`call`**，**`uncall`**をして関数の呼び出し，逆呼び出しを行う

Janusにおけるif文と繰り返し文



if e_1 then s_1 else s_2 fi e_2



from e_1 do s_1 loop s_2 until e_2

書き換え規則（予稿_図3.3）

$$\mathcal{T}[\![\ \texttt{x = c;}\ ]\!] = \begin{array}{l}\texttt{push(x,g)}\\ \texttt{x \^{}= c}\end{array}$$

a:代入

$$\mathcal{T}[\![\ \texttt{int x = c;}\ ]\!] = \begin{array}{l}\texttt{local int x = c}\\ \quad\texttt{push(x, g)}\\ \texttt{delocal int x = 0}\end{array}$$

b:変数宣言

$$\mathcal{T}\left[\!\left[\begin{array}{l}\texttt{if(}\ e_1\texttt{)\{}\\ \quad s_1\\ \texttt{\}}\\ \texttt{else\{}\\ \quad s_2\\ \texttt{\}}\end{array}\right]\!\right] = \begin{array}{l}\texttt{if}\ \mathcal{T}[\![e_1]\!]\ \texttt{then}\\ \quad\mathcal{T}[\![s_1]\!]\\ \quad\texttt{push(1,g)}\\ \texttt{else}\\ \quad\mathcal{T}[\![s_2]\!]\\ \quad\texttt{push(0,g)}\\ \texttt{fi top(g) = 1}\end{array}$$

c:条件式

$$\mathcal{T}\left[\!\left[\begin{array}{l}\texttt{while(}\ e_2\texttt{)\{}\\ \quad s_1\\ \texttt{\}}\end{array}\right]\!\right] = \begin{array}{l}\texttt{push(1,g)}\\ \texttt{from top(g) = 1 loop}\\ \quad\mathcal{T}[\![s_1]\!]\\ \quad\texttt{push(0,g)}\\ \texttt{until !}\mathcal{T}[\![e_2]\!]\end{array}$$

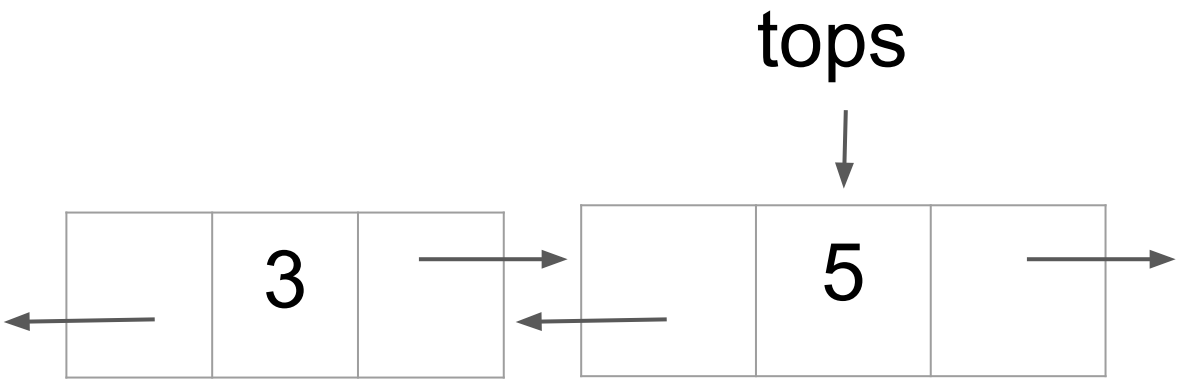
d:繰り返し文1

$$\mathcal{T}\left[\!\left[\begin{array}{l}\texttt{for (i = 0;}\ e_2\ \texttt{i++)\{}\\ \quad s_1\\ \texttt{\}}\end{array}\right]\!\right] = \begin{array}{l}\texttt{push(1,g)}\\ \texttt{from top(g) = 1 loop}\\ \quad\mathcal{T}[\![s_1]\!]\\ \quad\texttt{i+=1}\\ \quad\texttt{push(0,g)}\\ \texttt{until !}\mathcal{T}[\![e_2]\!]\end{array}$$

e:繰り返し文2

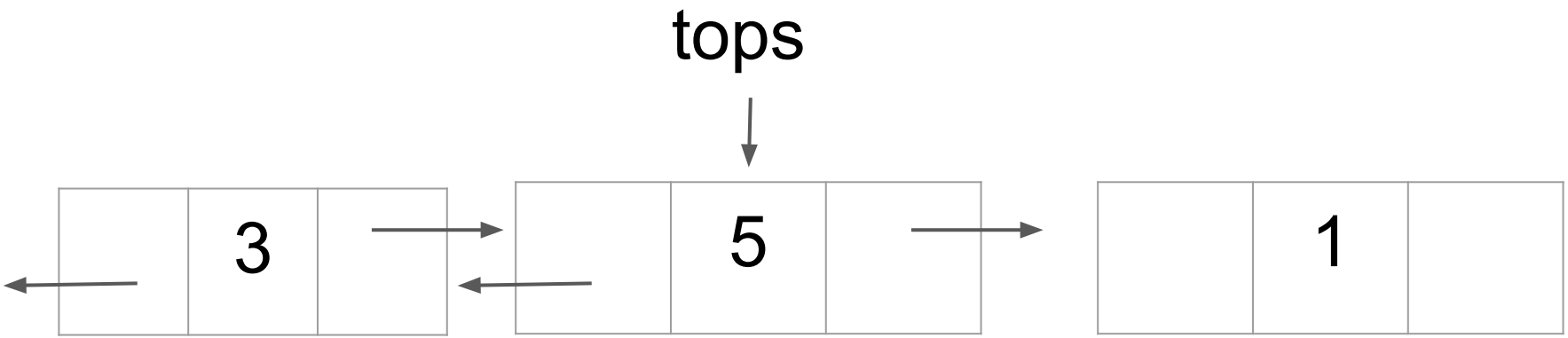
スタックの操作

prev	data	next



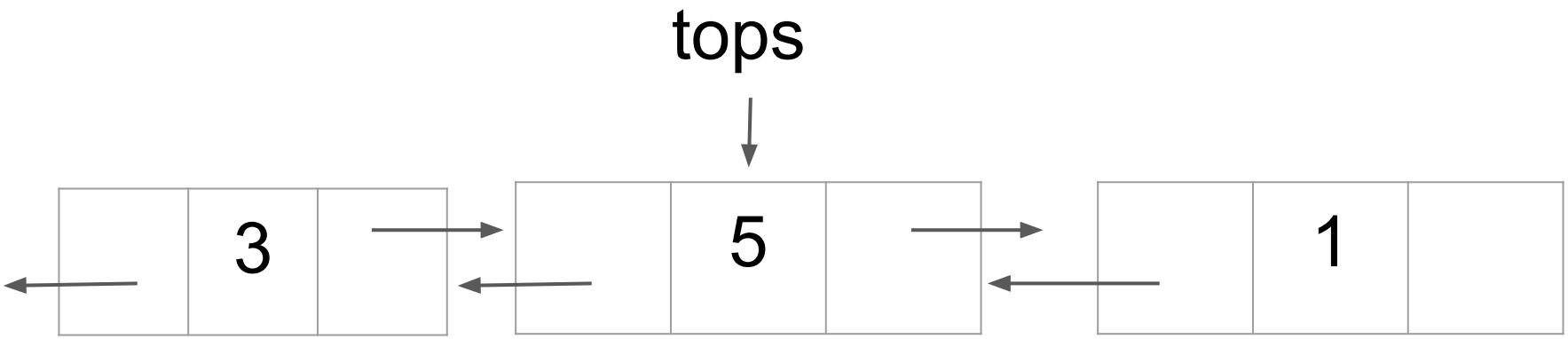
スタックを双方向リストで表現

スタックの操作



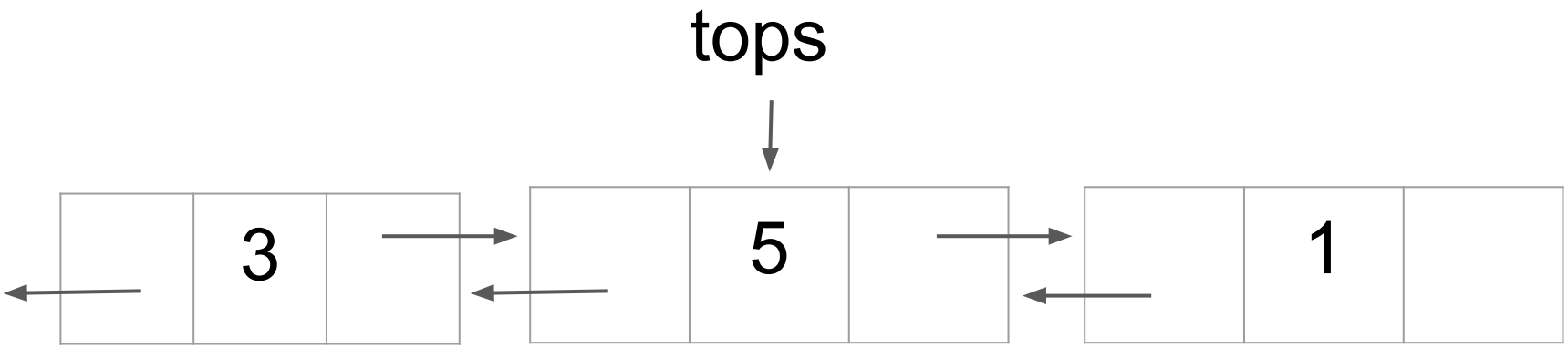
スタックに1をpushする

スタックの操作



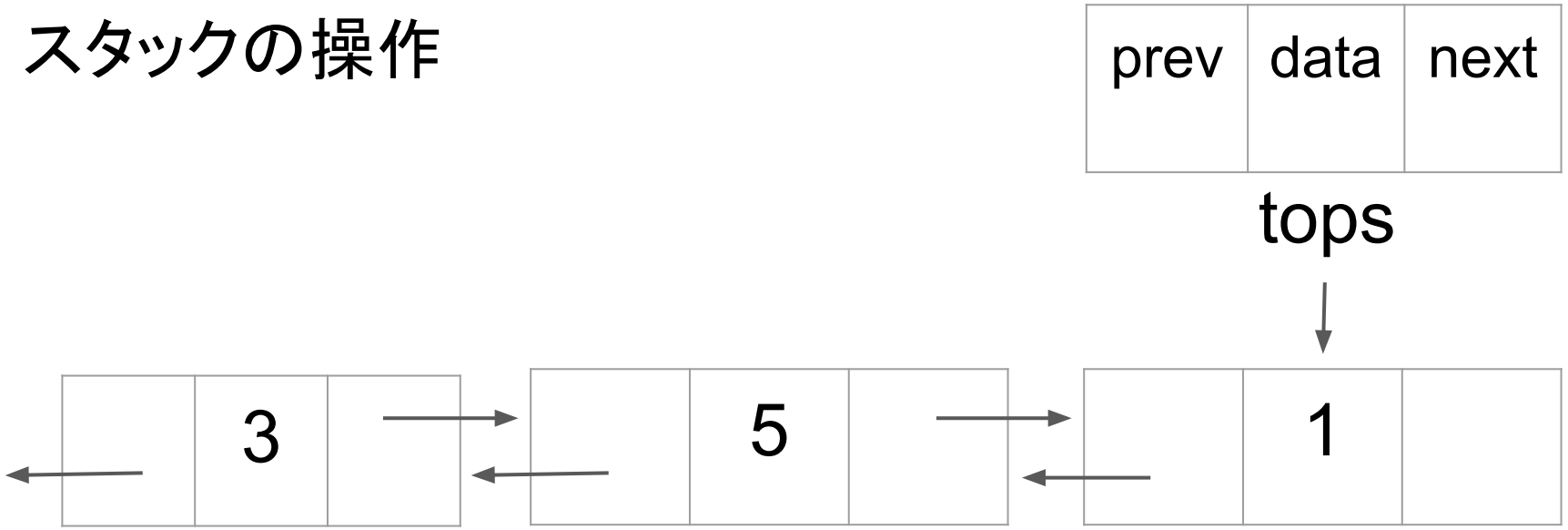
1のprevにtopsを代入

スタックの操作



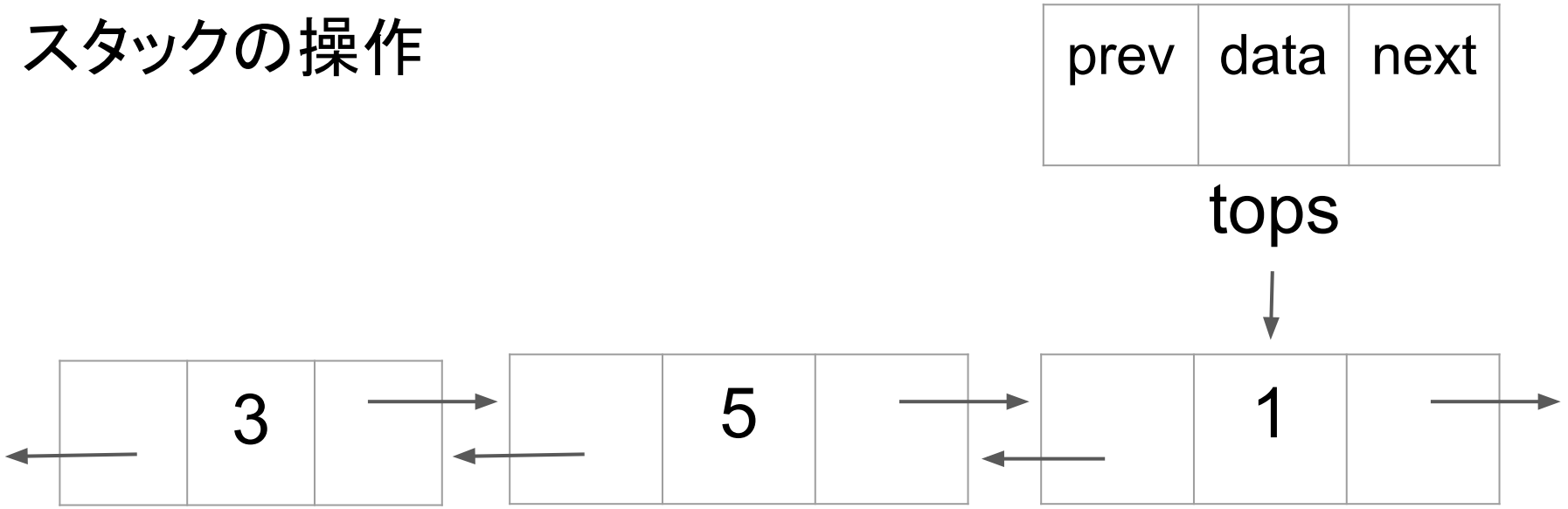
5のnextに1の場所を代入

スタックの操作



topsを1のprevで0クリア
topsに5のnextを代入

スタックの操作



スタックの操作

```
1  procedure spush(int data[],int next[],int prev[],int tops,int x)
2    local int i = 0
3      from i = 0 loop
4        i += 1
5      until data[i] = 0
6      data[i] ^= x
7      next[i] ^= -1
8      prev[i] ^= tops
9      next[tops] ^= -1
10     next[tops] ^= i
11     tops ^= prev[i]
12     tops ^= i
13   delocal int i = tops
```

キーが格納されていない
場所を探す

0クリアする

予稿_図3.4

埋め込みによる番兵を用いた線形探索

```
1  procedure s_banpei(int data[],int next[],int
       prev[],int tops,int key,int f,stack g)
2    push(1, g)
3    from top(g) = 1 loop
4      local int x = data[tops]
5        call spop(data,next,prev,tops,x)
6        push(x, g)
7      delocal int x=0
8      push(0, g)
9    until data[tops] = key
10   if tops != 0 then
11     push(f, g)
12     f ^= 1
13     push(1, g)
14   else
15     push(0, g)
16   fi top(g) = 1
```

```
while(data[*tops] != key){
    int x = data[*tops];
    spop(data,next,prev,tops,&x);
}
```

dの書き換え規則

```
f = 1;
```

aの書き換え規則

予稿_図3.5

Bennett法

```
1  procedure s_banpei_b(int data[],int next[],int prev[],int tops,int key,int f)
2    local stack g = nil
3      local int ff = 0
4        call s_banpei(data,next,prev,tops,key,ff,g)
5        f ^= ff
6        uncall s_banpei(data,next,prev,tops,key,ff,g)
7      delocal int ff = 0
8    delocal stack g = nil
```

関数呼び出し

保存

関数逆呼び出し

予稿_図3.5

手動による番兵を用いた線形探索

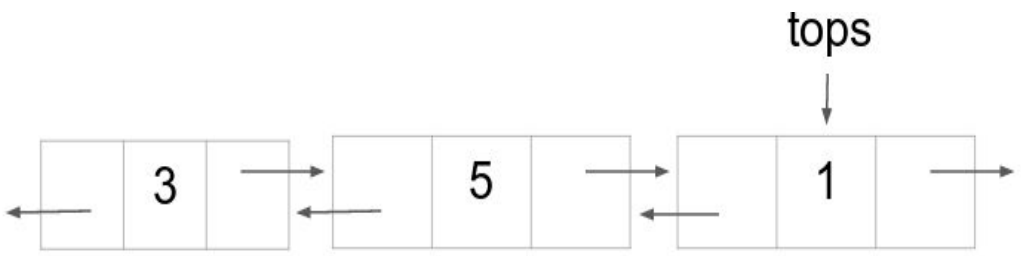
```
1  procedure s_banpei(int data[],int next[],int
       prev[],int tops,int key,int f,int data2[])
2    local int c = tops
3      from tops = c loop
4        local int x = data[tops]
5          call spop(data,next,prev,tops,x)
6        delocal int x = data2[2*data2[2*tops+1]]
7      until data[tops] = key
8      if tops != 0 then
9        f ^= 1
10     fi tops != 0
11     from data[tops] = key loop
12       local int x = data2[2*data2[2*tops+1]]
13         call spush(data,next,prev,tops,x)
14       delocal int x = data[tops]
15     until tops = c
16   delocal int c = tops
```

dataとnextを交互に格納

予稿_図3.6

手動による番兵を用いた線形探索

data[]={ 3, 1, 5}
next[]={ 2, -1,1}
prev[]={-1, 2, 0}
tops=1



の場合

data2[]={3,2,1,-1,5,1}

data2[2*data2[2*tops+1]]はpop前のスタックのtop

5
1

手動による番兵を用いた線形探索

```
1  procedure s_banpei(int data[],int next[],int
       prev[],int tops,int key,int f,int data2[])
2    local int c = tops
3      from tops = c loop
4        local int x = data[tops]
5          call spop(data,next,prev,tops,x)
6        delocal int x = data2[2*data2[2*tops+1]]
7      until data[tops] = key
8      if tops != 0 then
9        f ^= 1
10     fi tops != 0
11     from data[tops] = key loop
12       local int x = data2[2*data2[2*tops+1]]
13         call spush(data,next,prev,tops,x)
14       delocal int x = data[tops]
15     until tops = c
16   delocal int c = tops
```

探索しているキーと
探したいキーが
同じになるまで

番兵か判定

予稿_図3.6

おわりに

- 線形探索の可逆化に関する議論が不足
- 一般解法では中間ゴミ情報が発生
- Janusで3種類の線形探索を可逆化
- 双方向リストでスタックを実現
- ゴミ情報の量とステップ数の観点で効率化

参考文献

[1]Knuth,K.E：``The Art of Computer Programming Volume 3 Sorting and Searching Second Edition 日本語版”,KADOKAWA（2015）

[2]Axelsen H.B., Yokoyama T. (2015) Programming Techniques for Reversible Comparison Sorts. In: Feng X., Park S. (eds) Programming Languages and Systems. Lecture Notes in Computer Science, vol 9458. Springer, Cham

[3]Landauer, R.: Information is Physical, *Physics Today* ,Vol. 44, No. 5(1991), pp. 23-29.

[4] Bennett,C.H.：Logical Reversibility of computation. *IBM J. Res. Dev.*, 17(6):525-532, 1973.