

Proving tree algorithms for succinct data structures

Reynald Affeldt ¹ Jacques Garrigue ²
Xuanrui Qi ³ Kazunari Tanaka ²

¹ 産業技術総合研究所

² 名古屋大学多元数理科学研究科

³ Tufts University

November 22, 2018

<https://github.com/affeldt-aist/succinct>

Introduction

Rank&Select
Plan
Definitions

LOUDS

Implementation
First attempt
Second try

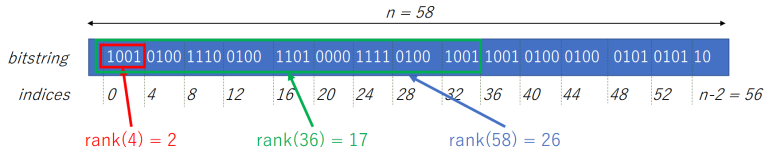
Conclusion

- 時間・空間複雑さが共に良いデータ表現
- 「復号化の要らない圧縮」
- ビッグ・データでは多用されている
- 応用例
 - データマイニングのデータの圧縮
 - グーグル日本語 IME の辞書

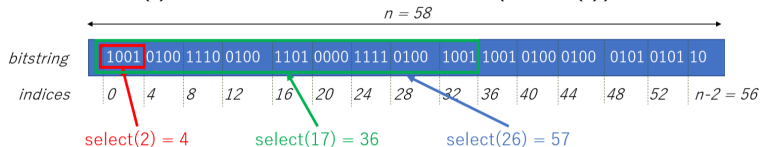
Rank と Select

ビット列への高速のアクセスを実現するために、二つの基本的な関数を最適化する。定数時間で実装可能。

- $\text{rank}(i) = i$ ビット目までの 1 の数



- $\text{select}(i) = i$ 番目の 1 の位置: $\text{rank}(\text{select}(i)) = i$



[Tanaka A., Affeldt, Garrigue 2016] で実装を CoQ で証明

簡潔データ構造における木構造の表現と利用

二つの視点

表現 rank と select を使って, 木構造をビット列で表現する

利用 木構造 (red-black tree) を使って, 動的変化が可能な
ビット列を実装

- 両実装の基本性質を Coq/SSREFLECT で証明
- 前者を後者の上で利用できる

CoQ での基本定義

`rank` は簡単に定義できる. `select` はその逆関数.

```
Variables (T : eqType) (b : T) (n : nat).
Definition rank i s := count_mem b (take i s).
Definition Rank (i : nat) (B : n.-tuple T) :=
  #|[set k : [1,n] | (k <= i) && (tacc B k == b)]|.
Lemma select_spec (i : nat) (B : n.-tuple T) :
  exists k, ((k <= n) && (Rank b k B == i)) ||
    (k == n.+1) && (count_mem b B < i).
Definition Select i (B : n.-tuple T) :=
  ex_minn (select_spec i B).
```

`pred s y` は `y` 以前の最後の `b`. `succ s y` は `y` 以後の最初の `b`.

```
Definition pred s y := select (rank y s) s.
Definition succ s y := select (rank y.-1 s).+1 s.
```

添字を正しく合わせるのが大変.

ここでは添字を1から数えるが、本によって異なる.

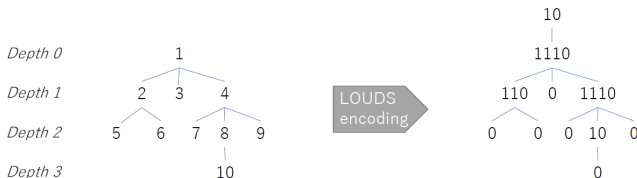
LOUDS

Conclusion

L.O.U.D.S.

Level-Order Unary Degree Sequence

[Navarro 2016, Chapter 8]



Depth 0	Depth 1	Depth 2	Depth 3
1	234	56789	10

	Depth 0	Depth 1	Depth 2	Depth 3
10	1110	11001110	000100	0

- 幅優先に並べた各ノードの次数の一進数表記
- 各ノードの子の数を表す 1 の後に 0 を付ける
- 丁度長さ $2n + 2$ のビット列で木の分岐構造が書ける
- 辞書などに応用 (グーグル日本語 IME)

基本操作の実装

木の中のパスと LOUDS 列の中の位置の間に同型を定義する.

必要な操作は

- 根の位置 (疑似ルートの追加で 2, 位置は 0 から数える)
- i 番目の子ノードの位置
- 親ノードの位置
- 子供の数

Variable B : seq bool.

Definition LOUDS_child $v\ i$:=

select false (rank true ($v + i$) B).+1 B .

Definition LOUDS_parent v :=

pred false B (select true (rank false v B) B).

Definition LOUDS_children v := succ false B v .+1 - v .+1.

基本操作の実装

木の中のパスと LOUDS 列の中の位置の間に同型を定義する.

必要な操作は

- 根の位置 (疑似ルートの追加で 2, 位置は 0 から数える)
- i 番目の子ノードの位置
- 親ノードの位置
- 子供の数

Variable B : seq bool.

Definition LOUDS_child v i :=

select false (rank true (v + i) B).+1 B.

Definition LOUDS_parent v :=

pred false B (select true (rank false v B) B).

Definition LOUDS_children v := succ false B v.+1 - v.+1.

問題: 構造的な対応になっていない

幅優先操作においてパスpより前に現れるノードの数を $\text{count_smaller } t_p$ とする

```

Definition LOUDS_position (t : tree) (p : seq nat) :=
  (count_smaller t p + (count_smaller t (rcons p 0)).-1).+2.
(*           0 の数           1 の数           疑似ルート *)

```

```

Definition LOUDS_subtree B (p : seq nat) :=
  foldl (LOUDS_child B) 2 p.

```

```
Theorem LOUDS_positionE t (p : seq nat) :  
  let B := LOUDS t in valid_position t p ->  
  LOUDS_position t p = LOUDS_subtree B p.
```

```
Theorem LOUDS_parentE t (p : seq nat) x :
  let B := LOUDS t in valid_position t (rcons p x) ->
  LOUDS_parent B (LOUDS_position t (rcons p x)) = LOUDS_position t p.
```

```
Theorem LOUDS_childrenE t (p : seq nat) :
  let B := LOUDS t in valid_position t p ->
  children t p = LOUDS_children B (LOUDS_position t p).
```

様々な問題点

- 幅優先走査が木の構造から離れている
- 構造的な帰納法が使えない
- 欲しい性質が簡単に証明できる「自然な対応」がない
- 証明すると添字が中々合わない

結果的には

- LOUDS に関する証明は 800 行以上
- 50 行を越える補題を多数含む
- もっと自然な対応があるはず

Second try

Introduction

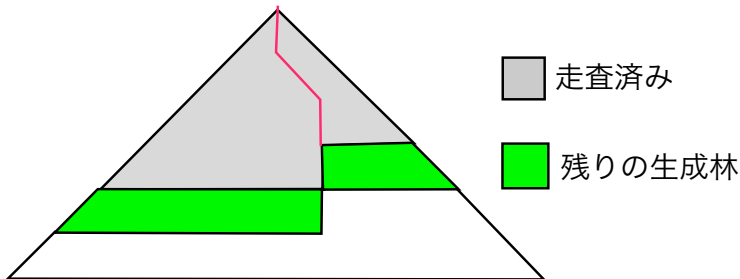
Rank&Select
Plan
Definitions

LOUDS

Implementation
First attempt
Second try

Conclusion

- 幅優先走査にパスの概念を導入する
- 帰納的な対応を得るために、木ではなく、林から生成
- 生成林は同じレベルから取らなくてもいい!



走査と残り

Introduction

Rank&Select

Plan

Definitions

LOUDS

Implementation

First attempt

Second try

Conclusion

Variable (A B : Type) (f : tree A -> B).

(* パス p より前のノードの幅優先走査 *)

Fixpoint lo_traversal_lt (w : forest A) (p : seq nat) : seq B.

(* パス p 以降の幅優先走査を行うための林 *)

Fixpoint lo_traversal_res (w : forest A) (p : seq nat) : forest A.

(* 両者の関係 *)

Lemma lo_traversal_lt_cat w p1 p2 :

lo_traversal_lt w (p1 ++ p2) =

lo_traversal_lt w p1 ++ lo_traversal_lt (lo_traversal_res w p1) p2.

(* All paths lead to Rome *)

Theorem lo_traversal_lt_max t p :

size p >= height t ->

lo_traversal_lt [:: t] p = lo_traversal_lt [:: t] (nseq (height t) 0).

この定理がlo_traversal_ltの一意性を保証する

LOUDS の順序と位置

(* LOUDS_lt もパスを使って定義する *)

Definition LOUDS_lt w p := flatten
 (lo_traversal_lt (node_description \o children_of_node) w p).

(* 通常の幅優先走査で定義された LOUDS と LOUDS_lt の関係 *)

Theorem LOUDS_lt_ok (t : tree A) :
 LOUDS t = true :: false :: LOUDS_lt [:: t] (nseq (height t) 0).

(* LOUDS での位置 *)

Definition LOUDS_position w p := size (LOUDS_lt w p).

(* 幅優先走査におけるノードの位置 *)

Definition LOUDS_index w p := size (lo_traversal_lt id w p).

Lemma LOUDS_position_select w p p' :
 valid_position (head dummy w) p ->
 LOUDS_position w p =
 select false (LOUDS_index w p) (LOUDS_lt w (p ++ p')).

Lemma LOUDS_index_rank w p p' n :
 valid_position (head dummy w) (rcons p n) ->
 LOUDS_index w (rcons p n) =
 size w + rank true (LOUDS_position w p + n) (LOUDS_lt w (p ++ n :: p')).

証明した性質

Introduction

Rank&Select
Plan
Definitions

LOUDS

Implementation
First attempt
Second try

Conclusion

Theorem LOUDS_childE (t : tree A) (p p' : seq nat) x :
 let B := LOUDS_lt [:: t] (rcons p x ++ p') in
 valid_position t (rcons p x) ->
 LOUDS_child B (LOUDS_position [:: t] p) x =
 LOUDS_position [:: t] (rcons p x).

Theorem LOUDS_parentE (t : tree A) p p' x :
 let B := LOUDS_lt [:: t] (rcons p x ++ p') in
 valid_position t (rcons p x) ->
 LOUDS_parent B (LOUDS_position [:: t] (rcons p x)) =
 LOUDS_position [:: t] p.

Theorem LOUDS_childrenE (t : tree A) (p p' : seq nat) :
 let B := LOUDS_lt [:: t] (rcons p 0 ++ p') in
 valid_position t p ->
 children t p = LOUDS_children B (LOUDS_position [:: t] p).

Introduction

Rank&Select
Plan
Definitions

LOUDS

Implementation
First attempt
Second try

Conclusion

よくなった所

- 全ての証明がパスに関する帰納法
- 自然に共通の補題が現われる
- 全体で 500 行に短縮, 長いものでも 25 行以下

問題点

- 相変わらず, 長い補題がある (lo_traversal_lt_max等)
- あらゆる所にパスが現われる

今後の展望

- 他の幅優先走査に応用できるか

コード

<https://github.com/affeldt-aist/succinct>