

# 簡潔データ構造 第 1 回

marimo

2023 年 4 月 25 日

# 前回やったこと

---

- Word-RAM の定義
- 簡潔データ構造の定義

簡潔ビットベクトル

**Succinct Bit Vector**

# 定義

---

$\{0,1\}^n$  の元を長さ  $n$  の**ビットベクトル**と呼ぶ。 $B \in \{0,1\}^n$  としたとき,

- $B[i]$  は  $B$  の  $i$  番目のビットを表す\*1
- $B[i..j]$  は  $B$  の  $i$  番目から  $j-1$  番目までの連続する部分列を表す
- 上の記法において  $i \geq j$  のときは空とする

---

\*1 0-indexed. 以降  $n$  番目などと言うとき 0-indexed とします

# 演算の定義

---

$B \in \{0,1\}^n$ ,  $j \in \{0,1\}$  とする。このとき次の演算を定める

- $\text{access}(B,i)$ :  $B[i]$  を返す
- $\text{rank}_j(B,i)$ :  $B[0..i]$  の  $j$  の数を返す
- $\text{select}_j(B,i)$ :  $B$  の先頭から  $i$  番目の  $j$  の位置を返す

# 今日の主定理

---

定理 .

長さ  $n$  のビットベクトルが与えられたとき,  $O(n)$  時間の前計算によって構築される  $O(n \lg \lg n / \lg n)$  ビットの索引を用いて rank, select は語長  $\Omega(\lg n)^{*2}$  の Word-RAM 上で**定数時間**で計算できる

---

<sup>2</sup>  $\Omega(f(n))$  は定義を与えていないが, これもランダウの記号のひとつであって漸近的に  $f(n)$  と同程度かこれより大きい関数の集合である

# 定理の概観

---

$o(n)$  bits の索引をつかって定数時間で rank, select を処理できるのは結構非自明なアイデアにやるなら,

- rank に答えるために prefix sum<sup>\*3</sup>を持つ
- select も  $i$  番目の 0,1 の位置を全部持つ

といったものが考えられるが、これらは  $n \lg n$  bits の空間を使うので簡潔ではない  
実際には rank を計算するための索引は工夫した prefix sum を用いる。select の計算にはナイーブな持ち方と  $n$  分探索をするための索引の 2 つを使う。加えて反転したベクトルを考えればよいので  $\text{rank}_1, \text{select}_1$  の索引だけ考えれば OK.

rankの計算



# rankの計算

---

$l = \lg^2 n$ \*4とする。 $B$ を先頭から長さ  $l$  の**大ブロック**に分割する。

つまり,  $j$  番目の大ブロックとは  $B[lj..l(j+1)]$  のこと

ここで整数の配列  $R_L$  を次のように定める

$R_L[i] =$  最初から  $i - 1$  番目の大ブロックまでの 1 の総数

ただし  $R_L[0] = 0$  とする。こうすると  $x = \lfloor i/l \rfloor$  とすれば

$$\text{rank}_1(B, i) = R_L[x] + \sum_{j=lx}^{i-1} B[j]$$

$R_L$  を持つためには  $O(n/\lg^2 n \times \lg n) = O(n/\lg n)$  bits だけの空間を使えばよいので、この索引は簡潔。これだけだとまだ rank に  $O(\lg^2 n)$  時間かかる。

---

4 これは本当はよくないのですが,  $\lg^2 n = (\lg n)^2$  と思ってください

## rankの計算 その2

---

さらに大ブロックを長さ  $s = \frac{1}{2}\lg n$  の**小ブロック**に分割する。

新たな整数の配列  $R_S$  を次のように定める

$R_S[i] = i$ 番目の小ブロックが属す大ブロックについて  
その先頭から直前の小ブロックまでの中の1の総数

ただし大ブロックの先頭の場合は0とする。こうすると  $x = \lfloor i/l \rfloor, y = \lfloor i/s \rfloor$  とすれば

$$\text{rank}_1(B, i) = R_L[x] + R_S[y] + \sum_{j=sy}^{i-1} B[j]$$

$R_S$  を持つためには  $O(n/\lg n \times \lg \lg n) = o(n)$  だけかかるので、OK。まだ  $O(\lg n)$  時間かかる。

# rankの計算 その3

---

細分化の方針だけでは定数時間にはできない。

ここで新しい方針**表引き**をします。

今のところ問題になっている  $B[sy..i]$  の長さは  $\frac{1}{2} \lg n$  以下だから、Word-RAM でこのビット列を読んで整数  $w$  と思うのは定数時間でできる。

$w$  は 0 以上  $\sqrt{n}$  以下なので、次のような表  $T$  を  $O(\sqrt{n} \lg n \lg \lg n) = o(n)$  でもてる。

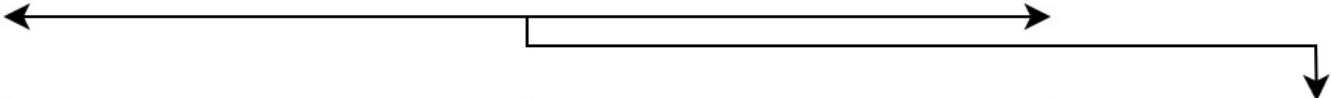
$T[w][i] = w$  を 2 進表現したビット列の、先頭から  $i$  番目までの 1 の数  
ただし、 $i$  番目も含む。

というわけでこれら 3 つの構造を持つことで  $\text{rank}_1$  が定数時間で計算できる。

$$\text{rank}_0(B, i) = i - \text{rank}_1(B, i)$$

を使えば  $\text{rank}_0$  も定数時間で計算できる。

# rankの計算

$B$	011	101	011	110	101	111	100	111	001
$R_S$	0	2	4	0	2	4	0	1	4
									
$R_L$	0			6			13		

$l = 9, s = 3$ の場合の例

準備 :k 分木

k-ary tree

# 木の形式的定義

---

**グラフ** 頂点集合  $V$  と辺集合  $E \subset V^2$  の組  $G = (V, E)$  のこと

**ウォーク** (walk) 辺の列  $e_0, \dots, e_n$  であって,  $e_i = (v_i, v_{i+1})$  を満たすような頂点の列  $v_0, \dots, v_{n+1}$  が存在するもの。

**トレイル** (trail) ウォークであって, 辺が全て相異なるもの

**パス** (path) トレイルであって, 頂点が全て相異なるもの

**連結** (connected) グラフ  $G$  が連結であるとは, 任意の頂点  $u, v$  について  $u$  と  $v$  をつなぐパスが存在すること

**サイクル** (cycle) トレイルであって, 最初と最後の頂点が一致し, さらにそれ以外の頂点が全て相異なるもの

**木** (tree) グラフであって, サイクルを持たず, 連結であるもの

# 関連するものの定義

---

**根付き木** (rooted tree) 木であって、特に 1 つの頂点が**根** (root) として指定されているもの。

**親** (parent) 親とは根へのパス上の隣接する頂点のこと。

**子** (child) 根付き木の頂点  $v$  について、子とは親が  $v$  であるような頂点のこと。

**子孫** (descendant) 子孫とは子と子の子孫のこと。

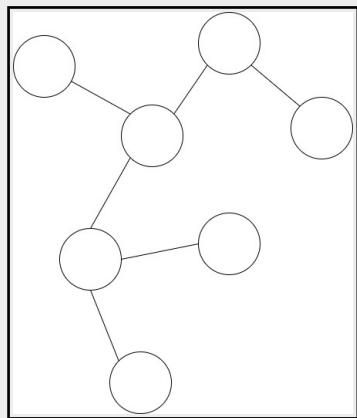
**葉** (leaf) 子を持たない頂点のこと。

**高さ** (height, depth) 根から最も遠い葉までの距離

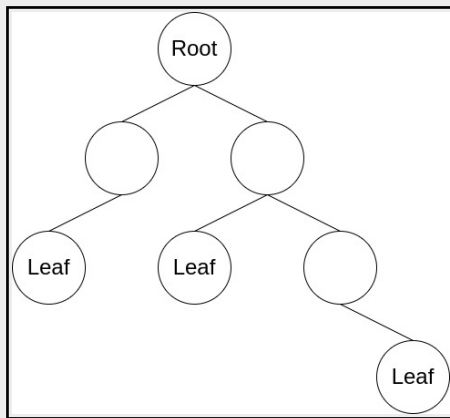
# おきもち

よくわからない人は、紙の上に丸(頂点)をいくつか書いて、その間にループができないように線(辺)をつないで全部がつながるようにしてください。これが木です。

頂点の中から1つ選ぶことにして、これに**根**(root)と名前を付けます。こうすると繋がり関係を保ったまま、垂れ下がるようにできます。このとき、頂点に対して直接下にくるものを**子**、下にある頂点全部を**子孫**、子がないものを**葉**と呼びます。



頂点数 7 の木



頂点数 7 の根付き木

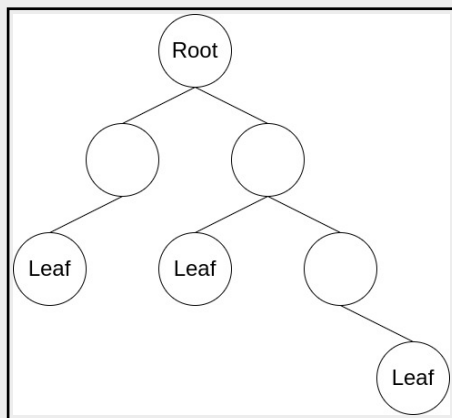


# k 分木

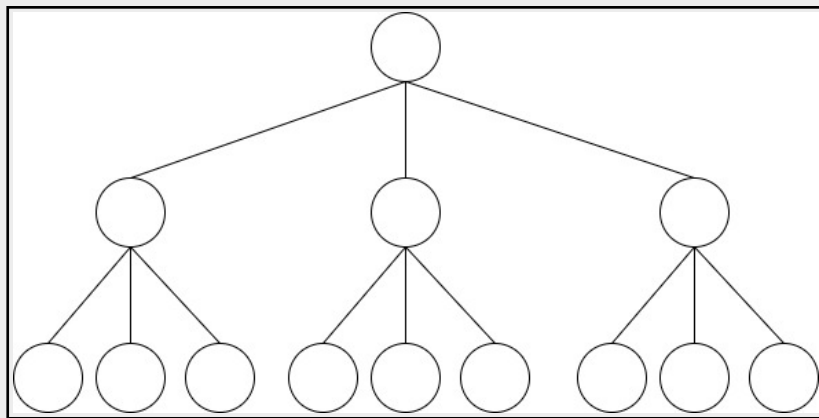
**k 分木** (k-ary tree) 全ての頂点について子の個数が高々  $k$  である木

**完全 k 分木** (perfect k-ary tree) 葉を除く全ての頂点について子の個数がちょうど  $k$  である木

完全 k 分木の各ノードに情報を載せていてこれを持ちたいとき、実は配列が 1 つあれば十分。



2 分木



完全 3 分木

# 完全 $k$ 分木の持ち方

---

配列  $A$  を次のように定める

- $A[0] = \text{root}$  の情報
- $A[i]$  の子孫の情報は  $A[i*k + 1]$  から  $A[i*k + k]$  までに載せる

これで実はいまうまくいっている。幅探索順序というやつ。 $k$  進数との対応を考えるとよいかもしれない。

select**の計算**

# selectの計算

---

$B$  の中の 1 の数を  $m$ ,  $l = \lg^2 n$  とする。

$B$  をそれぞれが 1 をちょうど  $l$  個含むように**大ブロック**に分割する。ただし最後のブロックはその限りでない。

つまり  $s_i = \text{select}_1(li)$ ,  $s_0 = 0$  とすれば,  $i$  番目の大ブロックは  $B[s_i..s_{i+1}]$  のこと。

大ブロックについて, その長さが  $\lg^4 n$  以上のとき疎 (sparse), そうでないとき密 (dense) と呼ぶ

# 疎なブロックの select

---

疎なブロックについてはそのブロック内の 1 の位置を全部持てばよい

これを持つためには  $O(n/\lg^4 n \times \lg^2 n \times \lg n) = O(n/\lg n)$  bits だけ使えば大丈夫なので簡潔

# 密なブロックの select

---

こちらは大変

まず、長さが  $s = \frac{1}{2} \lg n$  の**小ブロック**に分割する。大ブロックに含まれる小ブロックの数は高々  $2 \lg^3 n$  個。

これらを葉にするような完全  $\sqrt{\lg n}$  分木を作る。深さは定数<sup>5</sup>になる。

葉には小ブロックの中の 1 の数を持つ。  $\lg \lg n$  bits 程度で持てる。

内部ノードにはその子孫の小ブロックに含まれる全ての 1 の数を持つ。  $\lg \lg n$  bits 程度で持てる。ノードは全体でも  $O(n/\lg n)$  個なので全部持っても  $o(n)$  bits で簡潔。

加えて各大ブロックについて対応する構造を指すポインタを持つ必要があるが、これも  $O(n/\lg^2 n \times \lg n) = O(n/\lg n)$  bits で持てるので簡潔。

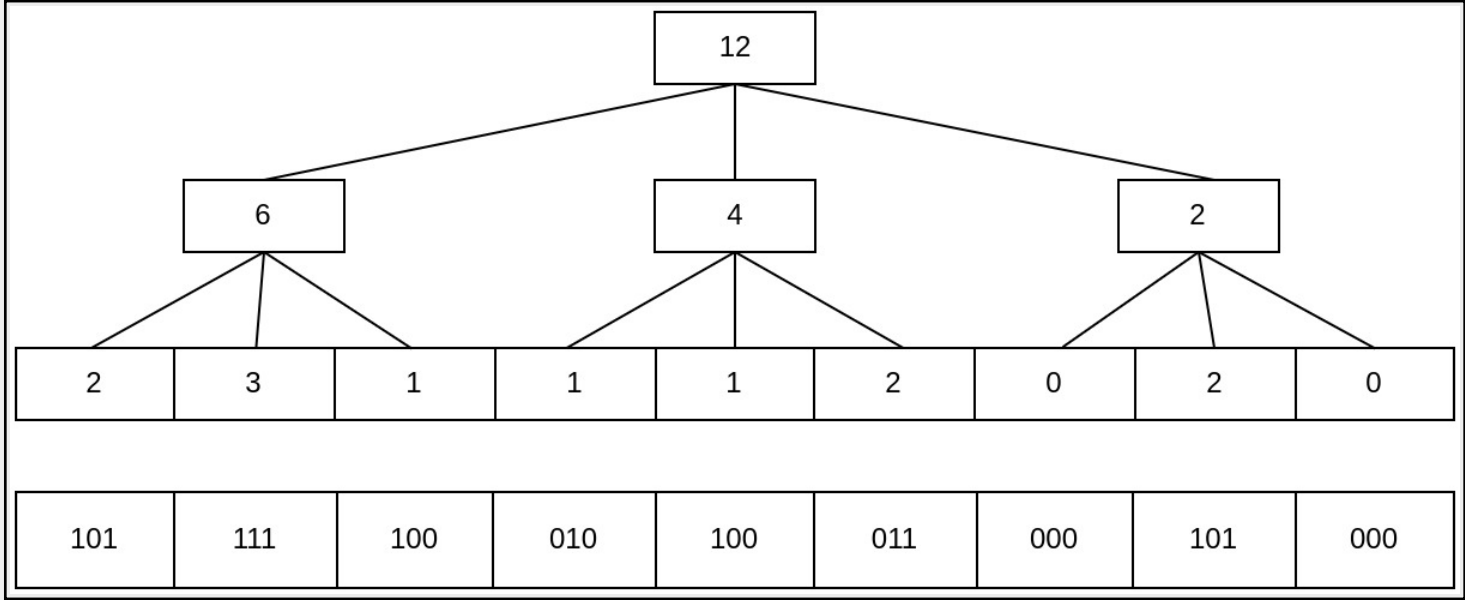
# selectの計算

---

この索引で  $\text{select}_1(B, i)$  を計算する方法をまだ言っていないので言います

1.  $i$  が属する大ブロックを求める
2. 大ブロックが密か疎か
  - (a) 疎の場合, 答えが入っているのでそのまま返す
  - (b) 密の場合, 次の手続きでもって求める
    - (i) 今いるノードの子孫の情報は連続する  $O(\sqrt{\lg n} \lg \lg n) = o(\lg n)$  bits にあるので整数として読む
    - (ii) 降りるべき子孫は表引きで分かる。表のサイズは  $O(2^{\sqrt{\lg n} \lg \lg n} \times \lg \lg n) = o(n)$  bits で簡潔。
    - (iii) あとは葉まで降りる。最後も表を引けばわかる。

# selectの計算



selectの密なブロックの索引の例。  $s = 3$ としている。



おまけ

長さ  $n$  のビットベクトルを持つ簡潔データ構造のサイズは  $n + o(n)$  bits である。

ただ、1 の数が  $m$  という制限を付けると情報理論的下限は

$$\lg \text{binom}(n, m) \sim m \lg \frac{n}{m} + (n - m) \lg \frac{n}{n - m} - O(\lg n)$$

になる。これは  $m$  が小さい、もしくは  $n$  に近いときは  $n$  よりも小さくなる。

じゃあ別のデータ構造が必要ですね。それが Fully Indexable Dictionary です。

知りたい人がそれなりにいればやります