range reporting

2020年度・高度データ構造

本日は range reporting という問題と、それを解くためのアルゴリズムとデータ構造を扱います。

range reporting

- 全体集合を $U = \{0, 1, ..., u\text{-}1\}$ とする.
- 部分集合 $S \subseteq U$ のサイズを n とする.
- 任意の整数 a, b (ただし $a \leqq b$) について,
  $[a, b] = \{a, a+1, ..., b\}$ とする.
- r_report(a, b, S): $S \cap [a, b]$ を求める操作.
- 例）$U = \{0, 1, ..., 15\}$, $S = \{3, 6, 7, 12, 14\}$ のとき,
  r_report(4, 10, S) = $S \cap [4, 10] = \{6, 7\}$,
  r_report(8, 13, S) = $S \cap [8, 13] = \{12\}$.

まずは，range reporting 問題を定義します。

いつものように，全体集合を U とし，その部分集合 S を考えます。n は S の要素数です。

整数 a, b について，[a, b] を a から b までの区間中の整数の集合とします。

r_report(a, b, S) は，S 中の [a, b] の範囲に含まれる要素を返す操作です。

例えば，U を 0 から 15 までのすべての整数の集合，S = {3, 6, 7, 12, 14} としたとき，r_report(4, 10, S) と r_report(8, 13, S) が返す集合はそれぞれ {6, 7} と {12} となります。

以降，この問題を解く手法について考えていきます。

range report with successor

定理 1

member と successor の 時間計算量が k のとき，
r_report(a, b, S) を $O(km)$ 時間で計算できる．
ただし，$m = |S \cap [a, b]| + 1$.

まず，これまでの講義で何度も出てきた successor クエリを使って range reporting を解く方法を紹介します。

member と successor の時間計算量を k とします。
このとき，r_report(a, b, S) を O(km) 時間で計算することがえきます。
ただし，m は r_report(a, b, S) の答えの個数＋1です。

(次ページに続く)

range report with successor

定理 1

member と successor の 時間計算量が k のとき，
r_report(a, b, S) を $O(km)$ 時間で計算できる．
ただし，$m = |S \cap [a, b]| + 1$.

以下のアルゴリズムは r_report を $O(km)$ 時間で計算する：

1. $R \leftarrow \phi$;
2. **if** member(a, S) = true **then** $R \leftarrow R \cup \{a\}$;
3. $x \leftarrow$ successor(a, S);
4. **while** $x \leqq b$ **do**
5. $\quad R \leftarrow R \cup \{x\}$;
6. $\quad x \leftarrow$ successor(x, S);
7. **return** R;

やり方は，以下に記したアルゴリズムの通りです。

R という集合に答えを記録していきます。最初は空集合です（1行目）

区間の最初の値 a が S の要素であれば，a を R に加えます（2行目）

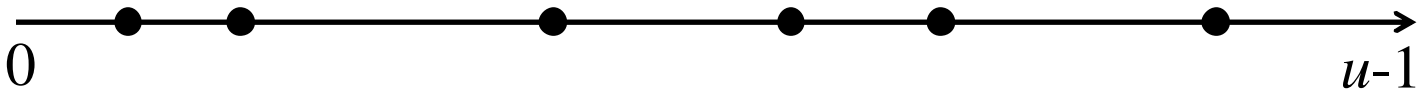
x に a の successor を代入し（3行目），x が区間の最後の値 b を超えない間，x の successor を R に追加していきます（4〜6行目）

最後に，R を出力します（7行目）

range report with successor

定理 1

member と successor の 時間計算量が k のとき,
r_report(a, b, S) を $O(km)$ 時間で計算できる.
ただし, $m = |S \cap [a, b]| + 1$.



- S の各要素

図で表すとこのような感じです。

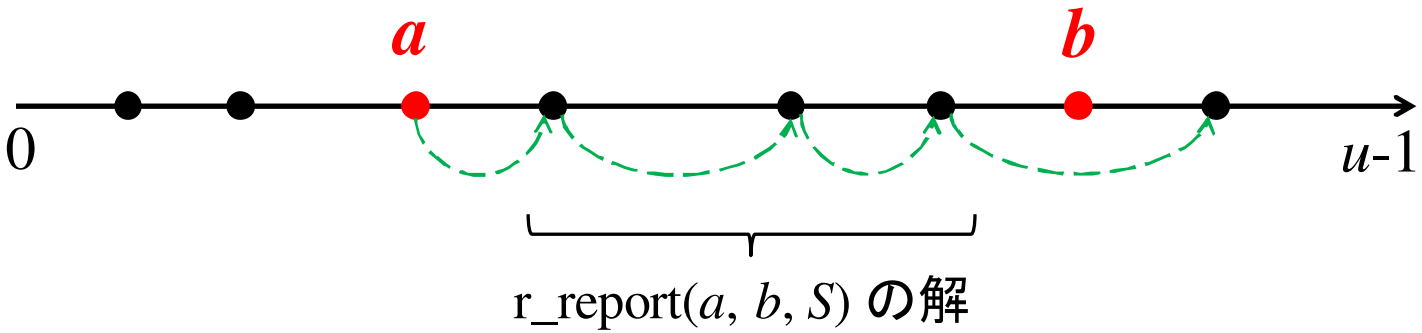
0 から u-1 (u = |U|) の数直線上に, S の要素を●で表示しています。

(次ページへ)

range report with successor

定理 1

member と successor の 時間計算量が k のとき，
r_report(a, b, S) を $O(km)$ 時間で計算できる．
ただし，$m = |S \cap [a, b]| + 1$.



- S の各要素

区間 [a, b] がクエリとして与えられたら（赤の点），a から順に S 中の successor を求め，b を超えたら終了し，[a, b] の範囲内の S の要素を返せばよい，ということです。

range report with successor

定理 1

member と successor の 時間計算量が k のとき，
r_report(a, b, S) を $O(km)$ 時間で計算できる．
ただし，$m = |S \cap [a, b]| + 1$.

- よって，1回の member と高々 m 回の successor を計算することで，r_report(a, b, S) を計算できる．
- したがって，$O(km)$ 時間で計算できる．

したがって，1回の member と高々 m 回の successor を計算することで，r_report(a, b, S) を O(km) 時間で計算できることになります。

ちなみに，m の値に＋1 しているのは，解の個数が 0 のときに時間計算量 O(km) の m が 0 にならないようにするためです。

今週のデータ構造

- 定理 1 より,
  - 平衡2分探索木で $O(m \log n)$ 時間, $O(n)$ 領域
  - van Emde Boas tree で $O(m \log\log u)$ 時間, $O(u)$ 領域
  - y-fast trie で $O(m \log\log u)$ 時間, $O(n)$ 領域

  で r_report を計算できる.

これまでの講義で扱ったデータ構造の successor の時間計算量を O(km) の k に代入すると, r_report をそれぞれこのような時間で計算できます。

いずれも, 答えの数 m に log n あるいは loglog u という項が掛かった時間を要します。

今週のデータ構造

- 定理 1 より,
  - 平衡2分探索木で $O(m \log n)$ 時間, $O(n)$ 領域
  - van Emde Boas tree で $O(m \log\log u)$ 時間, $O(u)$ 領域
  - y-fast trie で $O(m \log\log u)$ 時間, $O(n)$ 領域

  で r_report を計算できる.

【今週のデータ構造】

- **optimal range reporting [Alstrup et al. 2001]** :
  $O(m)$ 時間・$O(n)$ 領域で r_report を計算できる.

どんな範囲 $[a, b]$ を指定されても,
各要素を定数時間で出力できる！

これに対し，今回紹介するデータ構造では，O(m) 時間・O(n) 領域で r_report を計算することができます。

つまり，どんな範囲 [a, b] をクエリで指定されても，[a, b] の範囲内にある S の各要素を定数時間で出力できる，ということです。

素朴な $O(m)$ 時間データ構造

- 各範囲 $[a, b]$ について，$S \cap [a, b]$ の要素を保持しておけば，$O(m)$ 時間で r_report が解ける．
- a, b の組み合わせは $O(u^2)$ 通り．
- 各範囲の中の答えの数は $O(n)$ なので，この素朴な方法は $O(nu^2)$ 領域を要する．

上記のことからも，
$O(m)$ 時間・$O(n)$ 領域の凄さがわかる

O(m) 時間・O(n) 領域という結果の凄さを強調するために，まずは O(m) 時間で r_report を計算する素朴な（自明な）方法について考えてみます。

各範囲 [a, b] ついて，クエリの答えをすべて覚えておけば，当然 O(m) 時間でこの問題を解くことができます。

一方，a と b の組み合わせは O(u^2) 通りあります。

各範囲 [a, b] に含まれる S の要素数は高々 n なので，この素朴な方法は O(nu^2) 領域を要します。

これに対して，前述の手法はわずか O(n) 領域で同じ O(m) クエリ時間を達成しているわけですから，いかに優れた結果であるかがわかるかと思います。

アイデア

- 本講義では，$O(n \log u)$ 領域のデータ構造を紹介する．
  - $O(n)$ 領域のデータ構造も，同じアイデアに基づいている．

【アイデア】

- 範囲 $[a, b]$ に含まれる要素を，必ずしも前から順番に出力する必要はない．
- つまり，どんな順番で出力していってもよいので，最終的に $S \cap [a, b]$ が計算できればよい．

本講義では，O(n log u) 領域のデータ構造を紹介します。なお，O(n) 領域のデータ構造も，同様のアイディアに基づいています。

手法のアイデアは以下の通りです。

先ほどの successor を用いた手法では，答えを先頭（左）から順番に出力していっていました。

一方，r_report(a, b, S) の定義から，必ずしも順番に出力する必要はありません。

つまり，どんな順番で出力してもよいので，最終的に [a, b] の範囲内にある S の各要素を見つけることができれば良いわけです。

find_any

- find_any(a, b, S) : $S \cap [a, b]$ 中の要素を
  「どれかひとつ」求める操作.
  $S \cap [a, b] = \phi$ のときは，find_any(a, b, S) = nil とする.

- 例）$S = \{3, 6, 7, 12, 14\}$, $a = 4$, $b = 10$ のとき,
  $S \cap [4, 10] = \{6, 7\}$ なので,
  find_any(4, 10, S) は 6 か 7 のどちらかを返せばよい.

そこで，find_any という操作を考えます。

find_any(a, b, S) は，範囲 [a, b] に含まれる S 中の「任意の要素」を求める操作です。

もし答えがないときには，find_any(a, b, S) = nil とします。

具体例を見てみましょう。

S = {3, 6, 7, 12, 14}，a = 4, b = 10 のとき，[4, 10] に含まれる S の要素は 6 と 7 なので，find_any(4, 10, S) は 6 または 7 を返せばよい，ということになります。

range reporting with find_any

定理 2

find_any の時間計算量が h のとき，
r_report(a, b, S) を $O(hm)$ 時間で計算できる．
ただし，$m = |S \cap [a, b]| + 1$.

【証明】以下のアルゴリズムを考える．

r_report(a, b, S):

1. $x \leftarrow$ find_any(a, b, S);
2. **if** $x \neq$ nil **then**
3. r_report(a, x-1, S);
4. **output** x;
5. r_report(x+1, b, S);

find_any の 時間計算量を h としましょう。

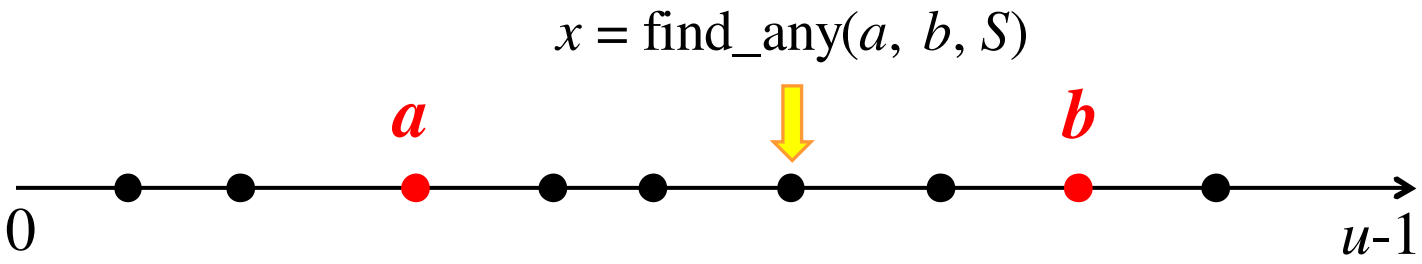
ことのとき，r_report(a, b, S) を O(hm) 時間で計算することができます。

証明のために，以下のアルゴリズムを考えます。
このアルゴリズムは，まず find_any で任意の解 x を求めたあと，x を分岐点として前半と後半に分けて，最適的に find_any を呼び出しています。

range reporting with find_any

定理 2

find_any の時間計算量が h のとき，
r_report(a, b, S) を $O(hm)$ 時間で計算できる．
ただし，$m = |S \cap [a, b]| + 1$.



- S の各要素

図で確認してみます。

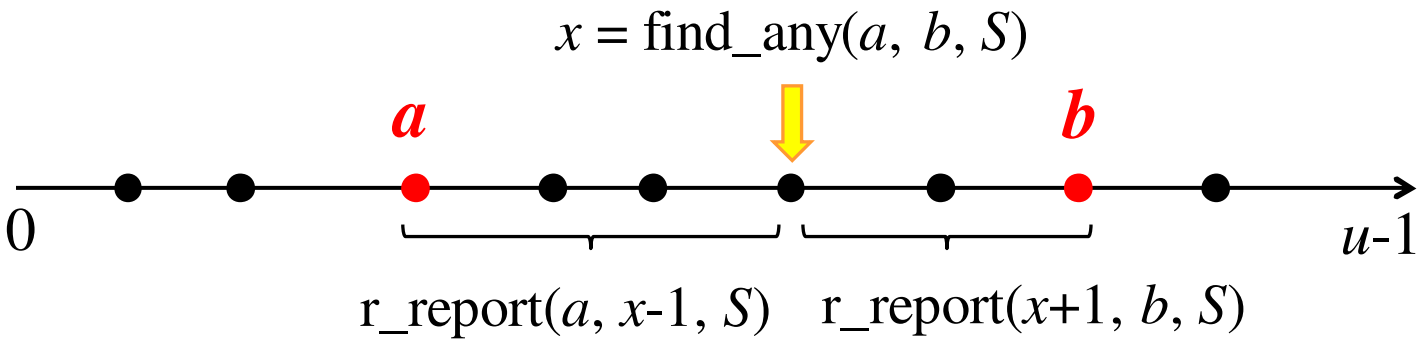
区間 [a, b] がクエリとして与えられたあと，find_any(a, b, S) = x をまず求めます。
いま，x は黄色の矢印が指している要素です。

(次につづく)

range reporting with find_any

定理 2

find_any の時間計算量が h のとき，
r_report(a, b, S) を $O(hm)$ 時間で計算できる．
ただし，$m = |S \cap [a, b]| + 1$.



- S の各要素

続いて，x を除いた前半の区間 [a, x-1] と後半の区間 [x+1, b] について，再帰的に r_report を呼び出していく，という流れです。

range reporting with find_any

定理 2

find_any の時間計算量が h のとき，
r_report(a, b, S) を $O(hm)$ 時間で計算できる．
ただし，$m = |S \cap [a, b]| + 1$.

【証明つづき】

- 上記のアルゴリズム r_report の再帰回数は，find_any の呼び出し回数に等しい．
- find_any の呼び出し回数は高々 $2m-1$ 回である．

13 ページのアルゴリズムを見直してみましょう。

このアルゴリズムは，再帰で呼び出されるたびに，1行目で find_any を必ず実行します。

よって，このアルゴリズムの再帰回数は find_any の呼び出し回数と等しくなります。

find_any の呼び出し回数は高々 2m-1 なので，r_report(a, b, S) に要する時間は h(2m-1) = O(hm) 時間となります。

演習問題

- アルゴリズム r_report 中の find_any の
  呼び出し回数は高々 $2m-1$回であることを示せ.

※ $m = |S \cap [a, b]| + 1$ なので,
r_report の答えの個数は $m-1$ であることに注意.

提出〆切：８月６日（木）２３：５９

さて，ここで本日の演習問題です。

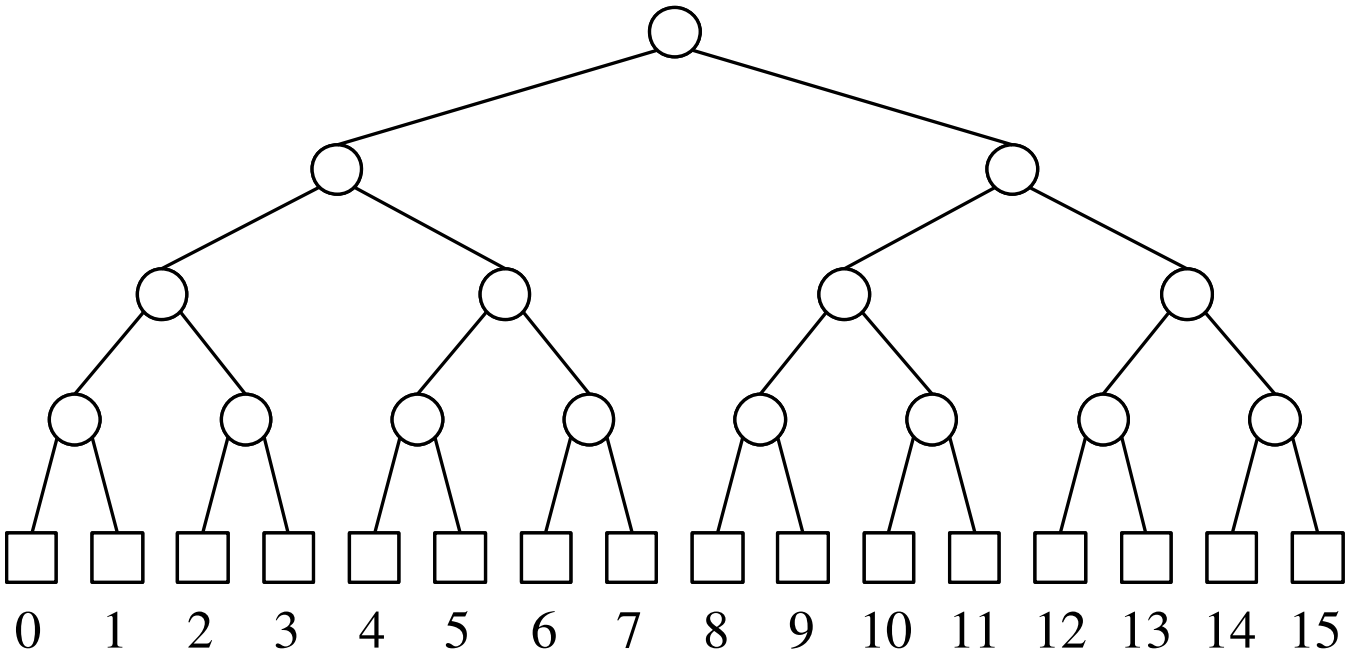
前ページで，アルゴリズム r_report 中の find_any の呼び出し回数は高々 2m－1回であると述べました。

このことを証明してください。

演習問題の提出期限は 8/6 23:59 です。

find_any のためのデータ構造

- 全体集合 U に対する平衡2分木 T を考える.



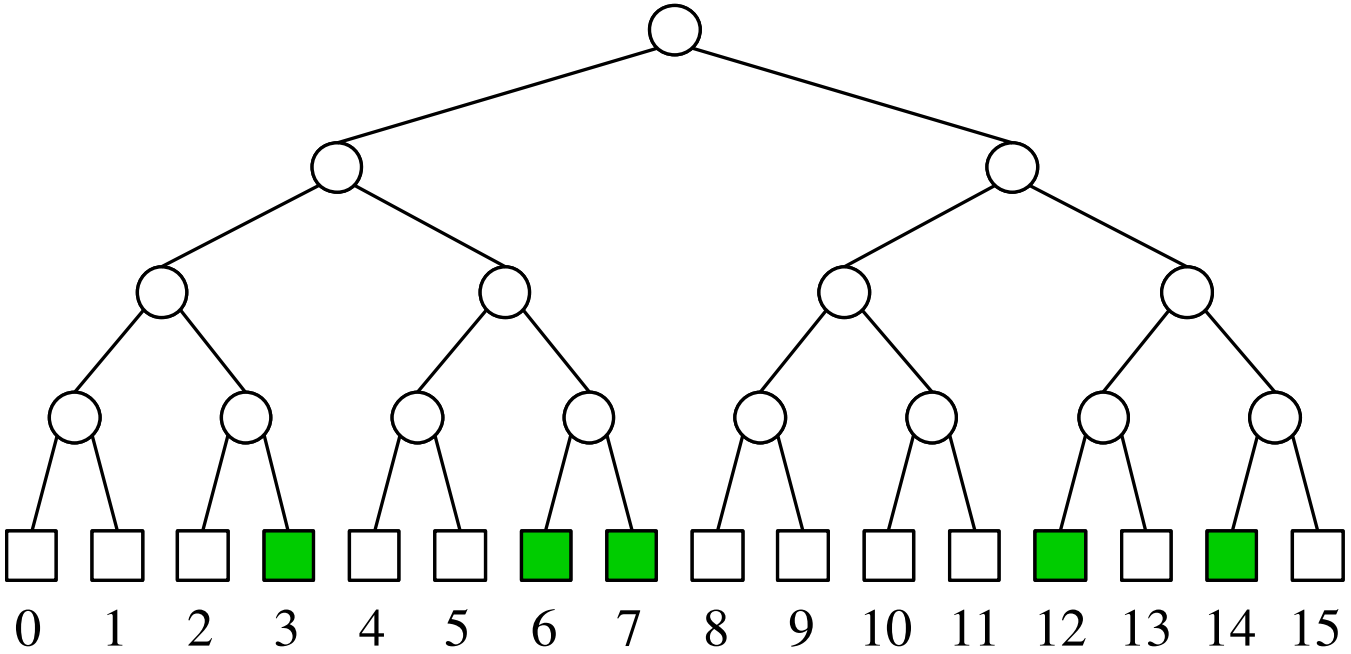
では，ここから先は find_any の具体的な実現方法について解説していきます。

これまでの講義で何度か行ったように，全体集合 U に対する平衡2分木を考え，これを T と名付けます。

T の葉は 0 から u-1 までの整数に対応しています。この例では，U = {0, 1, ..., 15} です。

S に対応する葉

- S の要素に対応する葉に着目する.
  - 下の例では, $S = \{3, 6, 7, 12, 14\}$

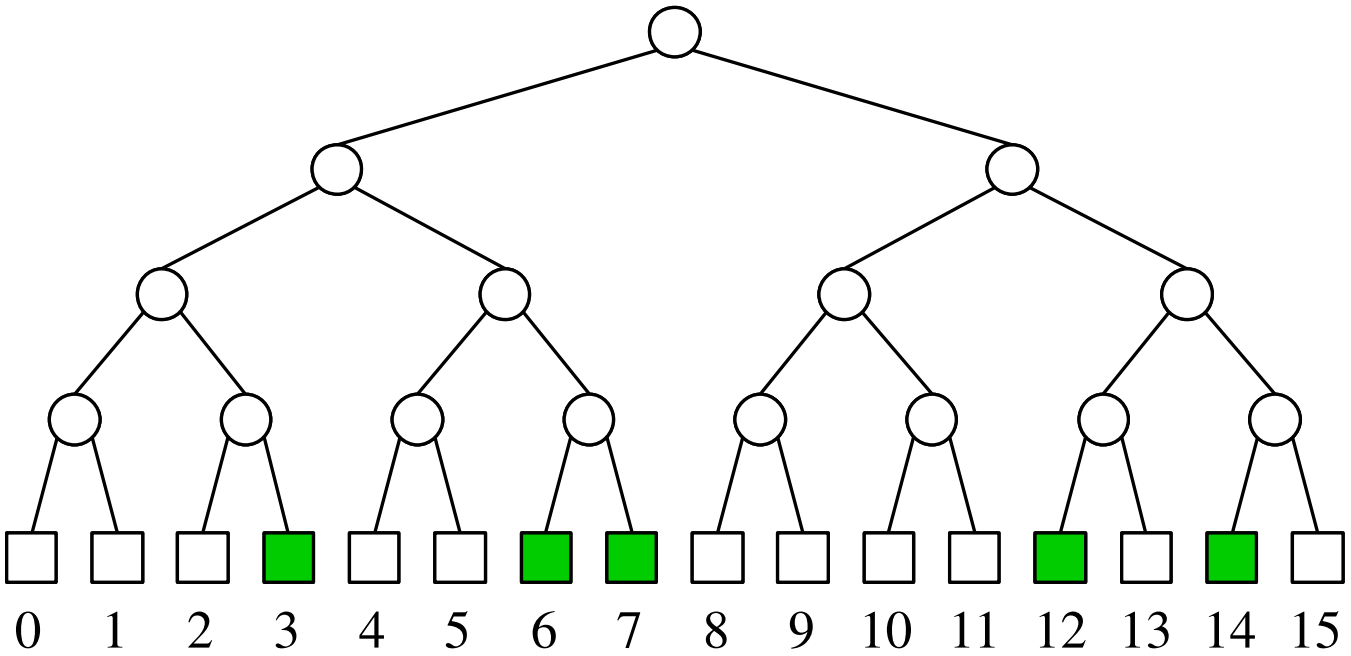


S の要素に対応する T の葉に注目します。

この例では，緑の葉が S の要素に対応しています。

range reportと最近共通祖先 (lca)の関係

- $v = \mathrm{lca}(a, b)$ とする．このとき，$S \cap [a, b] \subseteq S_v$ である．
  - S_v は v を根とする部分木に含まれる S の要素の集合．



T の2頂点 x と y の最近共通祖先 (lowest common ancestor) を lca(x, y) と書くことにします。

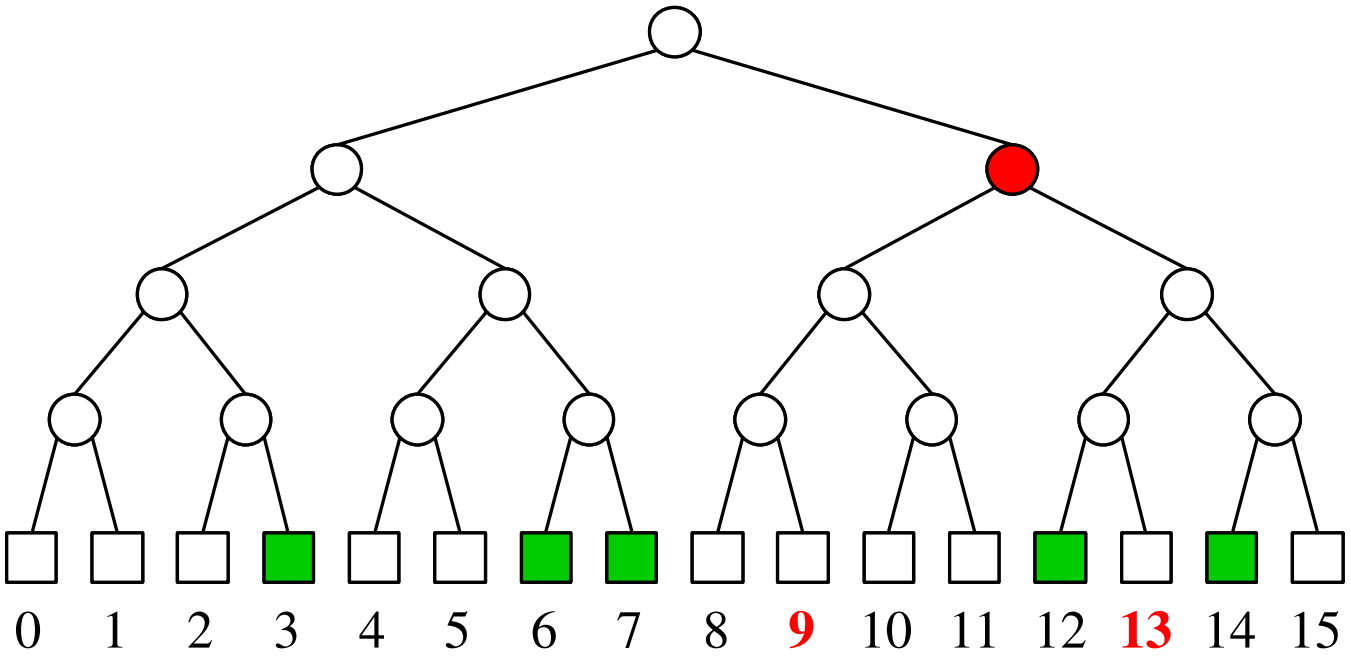
クエリ区間 [a, b] に対して，v = lca(a, b) とします。ここで，lca(a, b) の引数 a と b はそれぞれ a と b に対応する T の葉です。

S_v を v を根とする部分木に含まれる S の要素からなる集合とします。

このとき，r_report(a, b, S) の解は S_v の部分集合となっています。

range reportと最近共通祖先 (lca)との関係

- $S \cap [9, 13] = \{12\} \subset S_{\mathrm{lca}(9, 13)} = \{12, 14\}$



具体例で確認してみましょう。

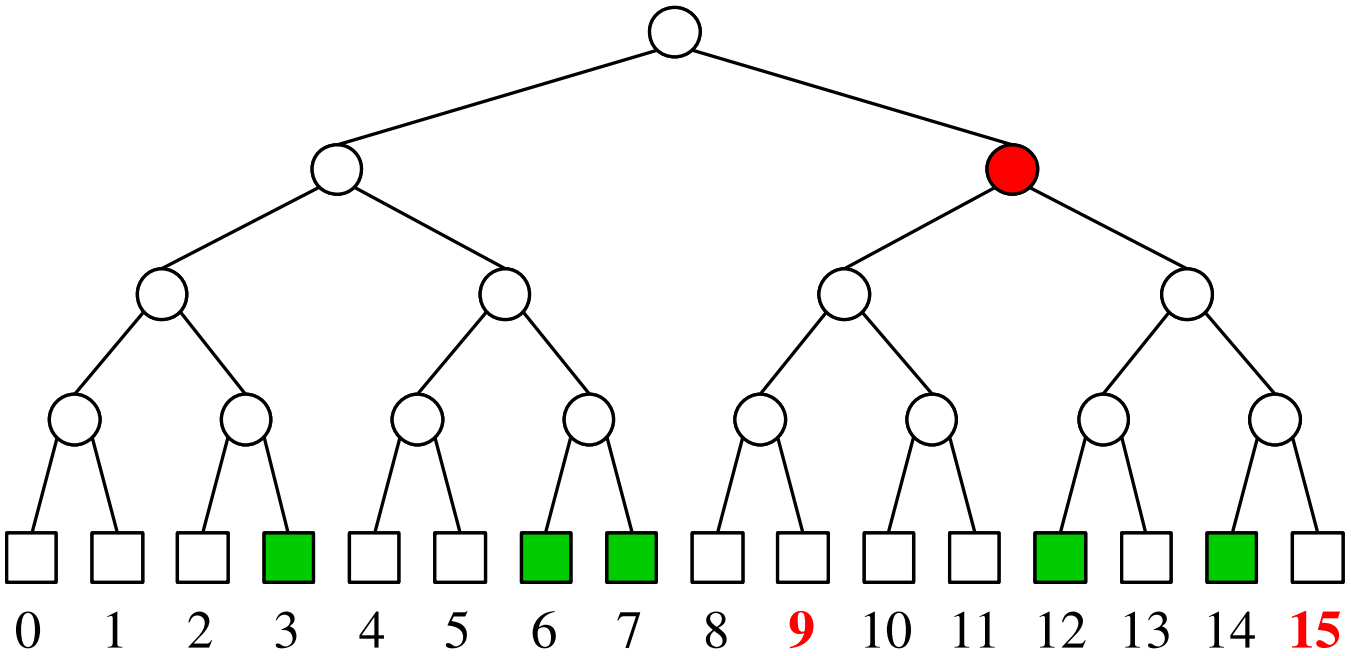
区間 [9, 13] に対して，
r_report(9, 13, S) = {12} です。

一方，9 と 13 の葉の lca（赤い内部頂点）の下にある緑の葉の集合は {12, 14} です。

確かに，{12} ⊂ {12, 14} となっていることが確認できます。

range reportと最近共通祖先 (lca)との関係

- $S \cap [9, 15] = \{12, 14\} = S_{\mathrm{lca}(9, 15)}$

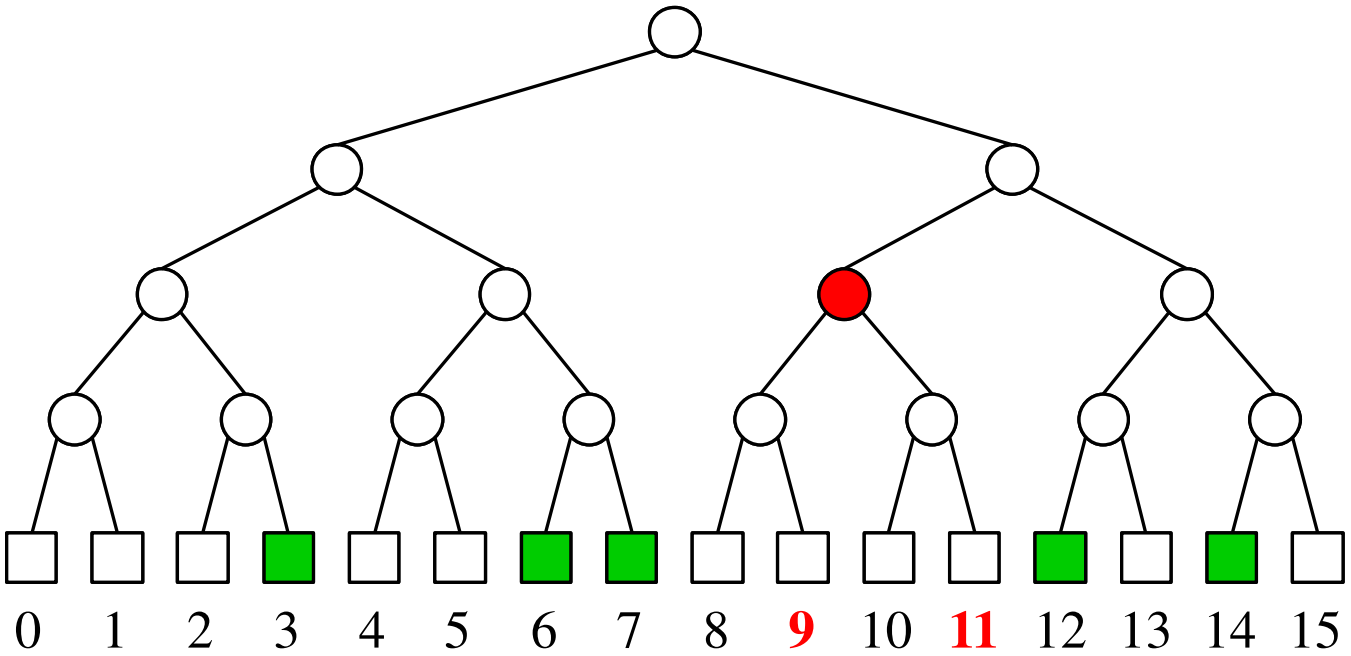


この図は，r_report(a, b, S) の解が S_v そのものになる例を示しています。

クエリ区間 [9, 15] に対して，r_report(9, 15, S) = {12, 14} であり，また，S_v = {9, 15} です。

range reportと最近共通祖先 (lca)との関係

- $S \cap [9, 11] = S_{\mathrm{lca}(9, 11)} = \phi$.



解なしの場合についても確認しておきましょう。

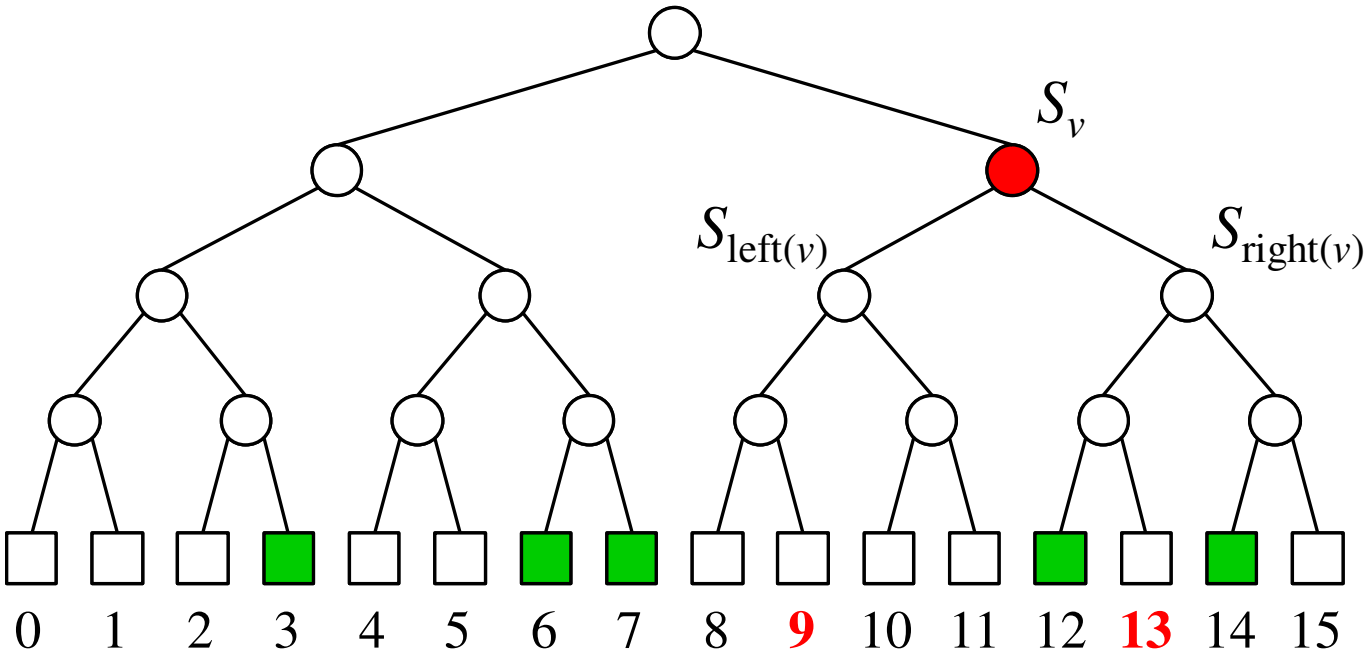
クエリ区間 [9, 11] について，r_report(9, 11, S) = Φ です。

一方，lca(9, 11) の部分木の中には緑の葉，つまり S の要素がないので，S_v = Φ です。

よって，この場合も r_report(a, b, S) ⊆ S_v という関係は成り立っています。

range reportと最近共通祖先 (lca)の関係

- v の左の子を left(v), 右の子を right(v) とする.
  $S_v \neq \phi$ のとき, $S \cap [a, b] \neq \phi$ ならば,
  $a \leqq \max S_{\text{left}(v)} < b$ または $a < \min S_{\text{right}(v)} \leqq b$ が成り立つ.



前ページまでの議論をさらに推し進めると，以下のような観察が得られます。

クエリ区間 [a, b] に対して v = lca(a, b) とします。
また，left(v), right(v) をそれぞれ v の左の子と右の子とします。

S_v が空でないとき，かつクエリの解集合 S ∩ [a, b] も空でないならば，
a ≦ max S_{left(v)} < b または
a < min S_{right(v)} ≦ b
が成り立ちます。

この図の例では，最初の不等式が成立しています。

一般には，両方の不等式が成立することもありますが，ここで重要なことは，少なくとも一方の不等式は必ず成り立つということです。

$O(1)$ 時間・$O(n \log u)$ 領域 find_any

補題 1

find_any(a, b, S) を $O(1)$ 時間で計算できる
$O(n \log u)$ 領域のデータ構造が存在する.

【データ構造の概要】

1. $v = \mathrm{lca}(a, b)$ を計算する.
2. $S_v = \phi$ ならば, nil を返す.
3. $S_v \neq \phi$ ならば, $[a, b]$ の範囲にある
   $\max S_{\mathrm{left}(v)}$, $\min S_{\mathrm{right}(v)}$ のどちらかを返す.

以上の観察から，この補題を得ます。

find_any を定数時間で計算できる O(n log u) 領域のデータ構造が存在します。

データ構造（アルゴリズム）の概要です。

まず，a と b の最近共通祖先 v = lca(a, b) を求めます（1行目）

もし S_v が空ならば，直ちに r_report(a, b, S) も空であることがわかるので，nil を返します（2行目）

もし S_v が空でないならば，[a, b] の範囲にある max S_{left(v)} または min S_{right(v)} のどちらかを返します（3行目）

このアルゴリズムの正当性は，前ページまでの議論から明らかです。

$O(n \log u)$ 領域の証明

補題 1

find_any(a, b, S) を $O(1)$ 時間で計算できる
$O(n \log u)$ 領域のデータ構造が存在する.

【領域計算量の証明】

- $S_v \neq \phi$ を満たす内部節点 v の集合を P とする.
- このとき, $|P| = O(n \log u)$ が成り立つ.

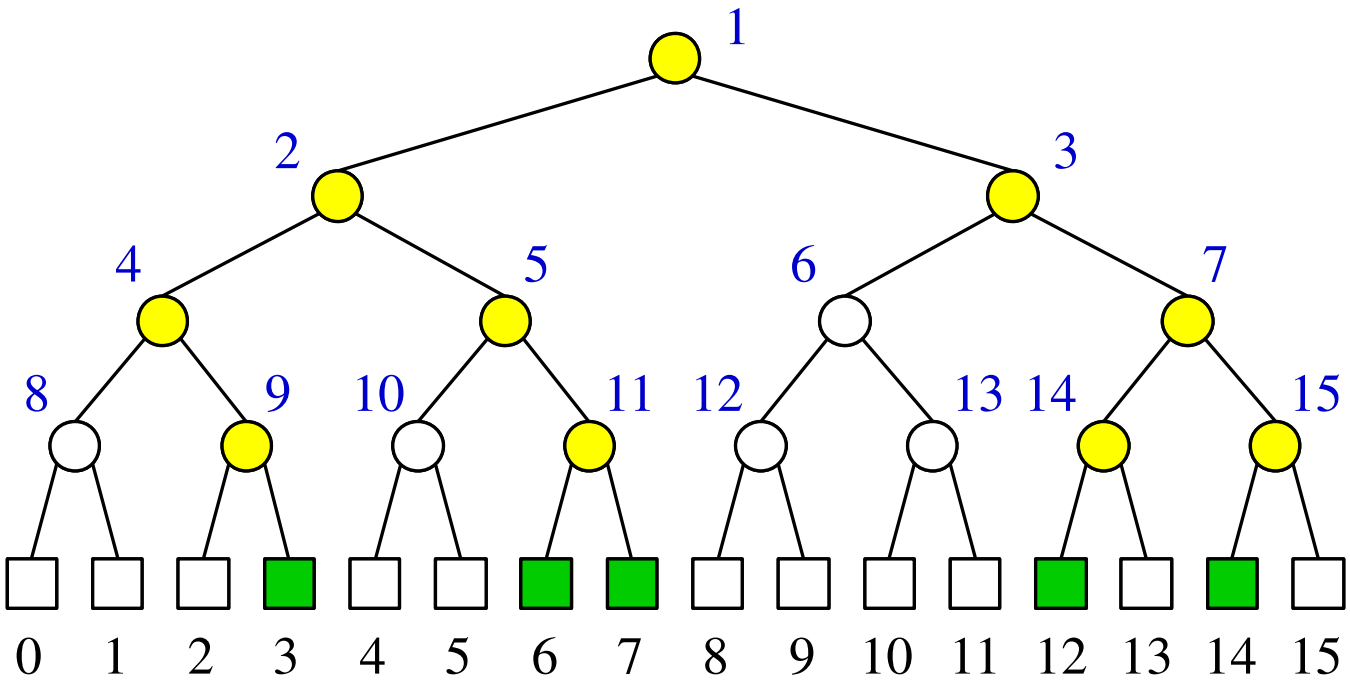
領域計算量について考察します。

S_v が空でない内部頂点 v の集合を P とします。

このとき，|P| は O(n log u) で上から抑えられます（第7回 x-fast trie のときと同じ議論です）

$O(1)$ 時間・$O(n \log u)$ 領域 find_any

- 下の例では，$P = \{1, 2, 3, 4, 5, 7, 9, 11, 14, 15\}$.



T の内部頂点それぞれに，1 から u-1 (= 15) までの番号を振っておきます。

この図の例では，P は黄色の内部頂点の番号の集合です。

黄色の内部頂点は，必ずその部分木に緑の葉を含みます。

緑の葉はちょうど n 個あり，この木 T の高さは log u なので，黄色の頂点の個数は明らかに O(n log u) です。

$O(n \log u)$ 領域の証明

補題 1

find_any(a, b, S) を $O(1)$ 時間で計算できる
$O(n \log u)$ 領域のデータ構造が存在する.

【領域計算量の証明】

- P の各要素を cuckoo hash table に格納する.
  このとき必要な領域は $O(n \log u)$ である.

議論を続けます。

P の各要素の番号の集合を，cuckoo hash (前回講義参照) に格納しておきます。

このとき，cuckoo hash の性質から，必要な領域は O(n log u) となります。

領域計算量に関する議論は以上です。

$O(1)$ 時間の証明

補題 1

find_any(a, b, S) を $O(1)$ 時間で計算できる
$O(n \log u)$ 領域のデータ構造が存在する.

【データ構造の概要】

1. $v = \mathrm{lca}(a, b)$ を計算する.
2. $S_v = \phi$ ならば, nil を返す.
3. $S_v \neq \phi$ ならば, $[a, b]$ の範囲にある
   $\max S_{\mathrm{left}(v)}$, $\min S_{\mathrm{right}(v)}$ のどちらかを返す.

次に，クエリの時間計算量について議論します。

2行目と3行目の場合分けを行うために，S_v が空であるか否かを判定する必要があります。

いま，目標は find_any クエリに O(1) 時間で応答することなので，この判定も O(1) 時間で行う必要があります。

$O(1)$ 時間の証明

- $P = \{v: S_v \neq \phi\}$ なので，$S_v \neq \phi$ の判定を行うには，$v \in P$ かどうかを調べればよい．
- つまり，member(v, P) を計算すればよい．
- いま，P を cuckoo hash table に格納しているので，member(v, P) は $O(1)$ 時間で計算できる．

P とは，S_v が空でない内部頂点 v の集合です。

したがって，S_v が空であるか否かを調べるためには，v が P の要素であるか否かを調べれば良いことになります。

つまり，member(v, P) を実行すればよいわけです。

いま，集合 P を cuckoo hash table に格納しているので，member(v, P) は O(1) 時間で計算できます。

$O(1)$ 時間の証明

補題 1

find_any(a, b, S) を $O(1)$ 時間で計算できる
$O(n \log u)$ 領域のデータ構造が存在する.

【データ構造の概要】

1. $v = \mathrm{lca}(a, b)$ を計算する.
2. $S_v = \phi$ ならば, nil を返す.
3. $S_v \neq \phi$ ならば, $[a, b]$ の範囲にある
   $\max S_{\mathrm{left}(v)}$, $\min S_{\mathrm{right}(v)}$ のどちらかを返す.

最後に残ったのは，1行目の時間計算量の解析です。

a, b が与えられたときに，その最近共通祖先 lca(a, b) を定数時間で計算する必要があります。

most significant bit

定義 1

任意の非負整数 $x \in U$ について，$\mathrm{msb}(x)$ を
x の2進表現の値が 1 である最大の桁とする．
すなわち，$\mathrm{msb}(x) = \max\{i : 2^i \leqq x\}+1$.

- 例）$x = 45$ の2進表現は 101101 なので，
  $\mathrm{msb}(x) = 6$ である．

そのための道具として，most significant bit (msb) という概念を導入します。

全体集合 U の要素である非負整数 x について，msb(x) を x の2進表現の値が 1 である最大の桁とします。

具体例を見てみましょう。

x = 45 のとき，その2進表現は 101101 です。赤で強調した桁が 1 になっている最大の桁なので，msb(x) = 6 となります。

最近共通祖先の計算

補題 2

T の任意の葉 $x, y \in U$ について，
x と y の $\mathrm{msb}(x \oplus y)$ 番目の祖先は
x と y の最近共通祖先 $\mathrm{lca}(x, y)$ である．
ここで，$x \oplus y$ は x と y の2進表現の XOR．

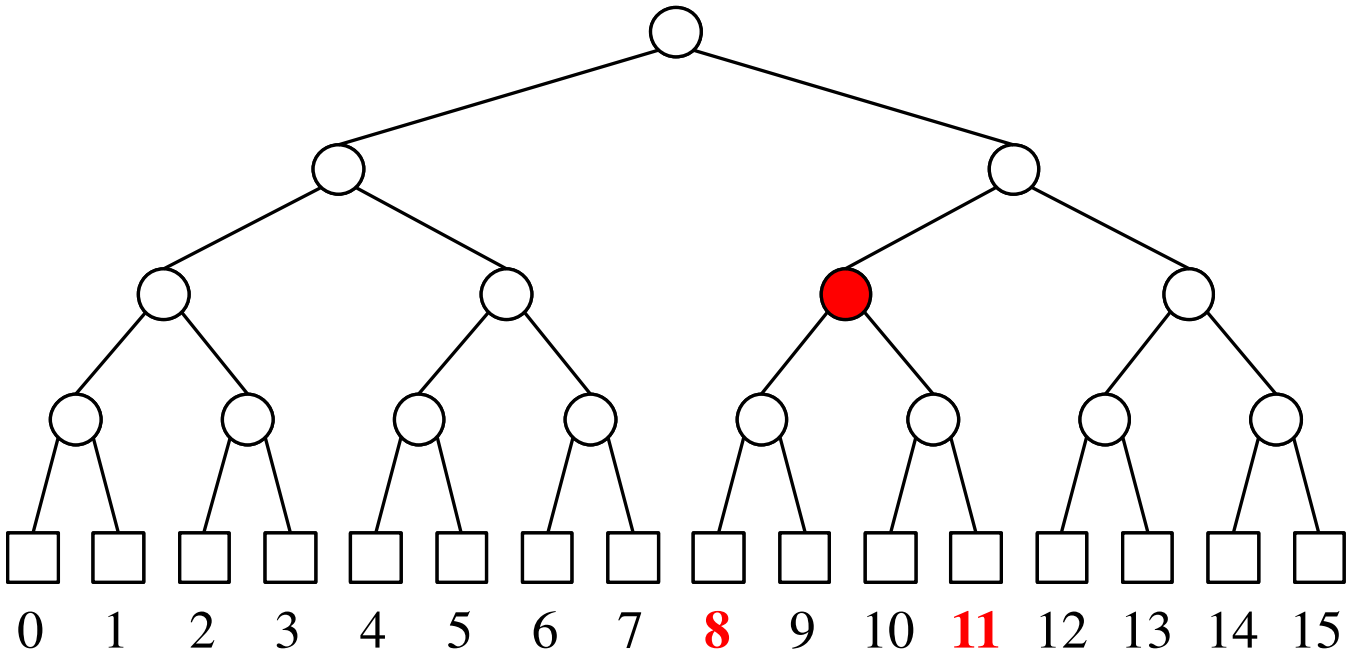
この msb を用いることで，lca(a, b) を高速に計算できます。

整数 x と y の2進表現の XOR (exclusive or, 排他的論理和) を $x \oplus y$ と書くことにします。

このとき，T の任意の x と y について，x と y の msb($x \oplus y$) 番目の祖先は，lca(x, y) と一致します。

最近共通祖先の計算

- 葉 8 と 11 を考える. $8 \oplus 11 = 1000 \oplus 1011 = 0011$ なので, lca(8, 11) は 2 番目の祖先.



具体例で確認してみましょう。

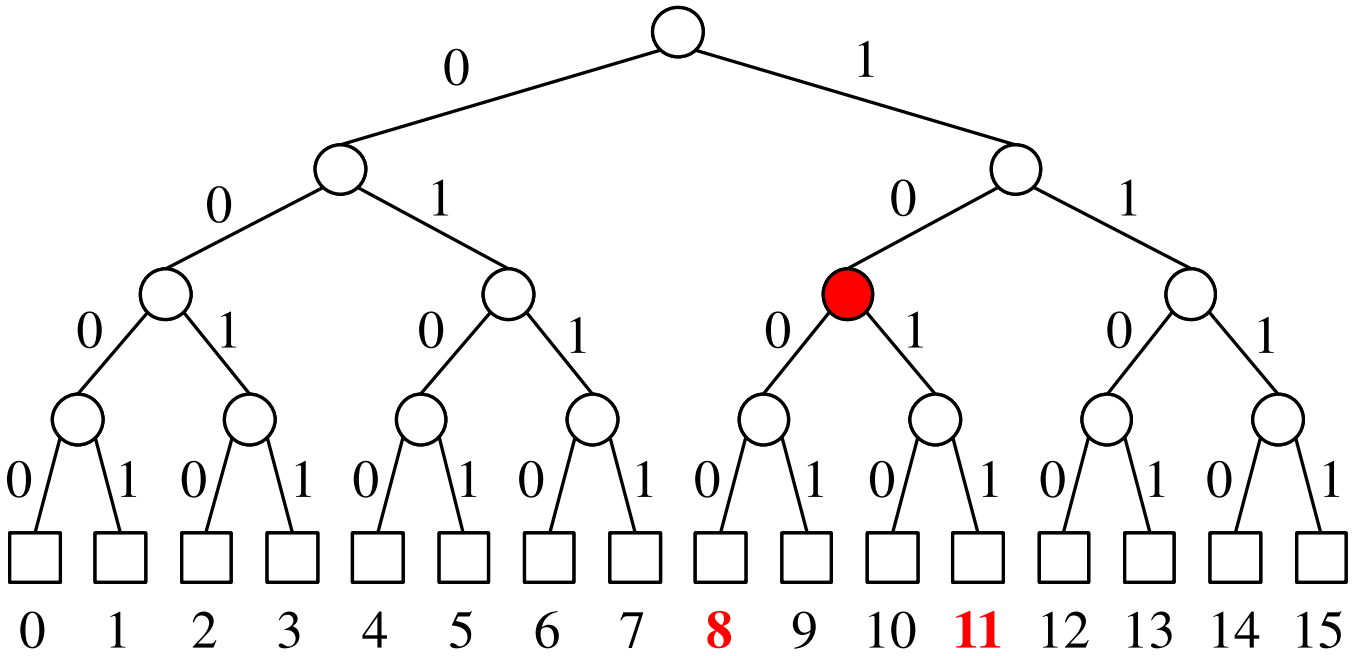
8 と 11 を考えます。これらを2進表現して XOR をとると, 0011 となります。

よって, msb(0011) = 2 番目の祖先が lca(8, 11) になるというわけです。

なぜしょう？（次ページへ）

最近共通祖先の計算

- 左の子への辺を 0，右の子への辺を 1 でラベル付けすると，根から葉へのパスが葉の2進表現に対応する.



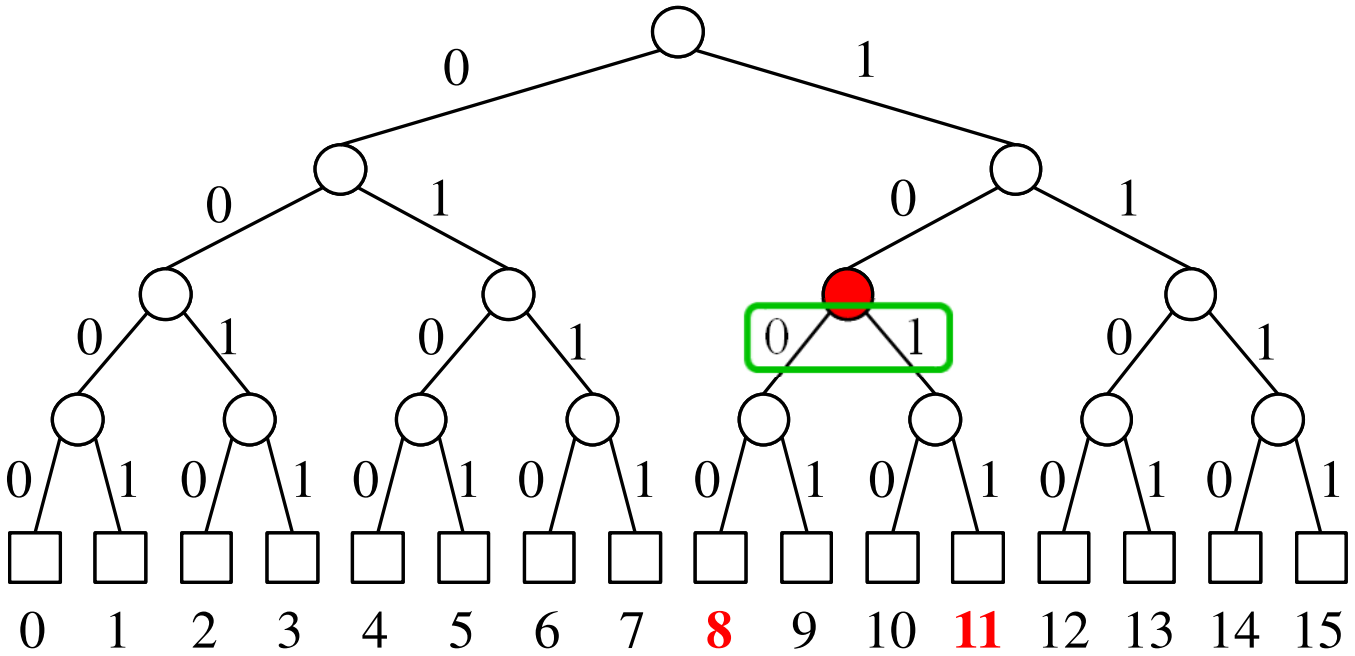
このことを理解するために，各頂点において，左の子への辺を 0，右の子への辺を 1 でラベル付けします。

すると，根から葉へのパスが葉の2進表現に対応します。

例えば，8 に対しては 1000，11 に対しては 1011 です。

最近共通祖先の計算

- msb(8 ⊕11) は 8 と 11 の2進表現の
  最初に食い違ったビットを示している.



ここで，msb(8 ⊕ 11) は，8 と 11 の2進表現の最初に食い違ったビットの位置を示していることになります。

よって，msb を持ちいることで，lca を計算できることになります。

最近共通祖先の計算

補題 2

T の任意の葉 $x, y \in U$ について,
x と y の $\mathrm{msb}(x \oplus y)$ 番目の祖先は
x と y の最近共通祖先 $\mathrm{lca}(x, y)$ である.
ここで, $x \oplus y$ は x と y の2進表現の XOR.

- msb と ⊕ は定数時間で計算可能.
  - msb の計算は [Fredman & Willard, 1990] を参照.

msb と ⊕ はそれぞれ定数時間で計算可能であることが知られています。

$O(1)$ 時間の証明

補題 1

find_any(a, b, S) を $O(1)$ 時間で計算できる
$O(n \log u)$ 領域のデータ構造が存在する.

【データ構造の概要】

1. $v = \mathrm{lca}(a, b)$ を計算する. ➔ $O(1)$ 時間
2. $S_v = \phi$ ならば, nil を返す. ➔ $O(1)$ 時間
3. $S_v \neq \phi$ ならば, $[a, b]$ の範囲にある
   $\max S_{\mathrm{left}(v)}$, $\min S_{\mathrm{right}(v)}$ のどちらかを返す.
   ➔ $O(1)$ 時間

というわけで，1〜3までのすべてのステップをそれぞれ O(1) 時間で実行できるので，結果として find_any を O(1) 時間で応答可能である，ということが示せました。

$O(m)$ 時間・$O(n \log u)$ 領域 range report

定理 3

r_report(a, b, S) を $O(m)$ 時間で計算できる
$O(n \log u)$ 領域のデータ構造が存在する.
ただし, $m = |S \cap [a, b]| + 1$.

【証明】

- 定理 2 と補題 1 より.

最後に, この定理が成り立ちます。

r_report(a, b, S) を O(m) 時間で計算する O(n log u) 領域のデータ構造が存在します。

今週の講義は以上です。