q-fast tries

2020年度前期　高度データ構造

本日は、q-fast trie というデータ構造を取り扱います。

前回のおさらい

- 全体集合を $U = \{0, 1, 2, ..., u\text{-}1\}$ とし，$S \subseteq U$ とする
- u: U の要素数, n: S の要素数

- 先週のデータ構造: p-fast tries [Willard 1984]
  - クエリ・操作時間: $O(\sqrt{\log u})$
  - 領域: $O(n\sqrt{\log u}\, 2^{\sqrt{\log u}})$

- 今週のデータ構造: q-fast tries [Willard 1984]
  - クエリ・操作時間: $O(\sqrt{\log u})$
  - **領域: $O(n)$**

前回の講義では p-fast trie について解説しました。

全体集合のサイズを u としたとき、各クエリ・操作を $O(\sqrt{\log u})$ 時間で実現する $O(n\sqrt{\log u}\, 2^{\sqrt{\log u}})$ 領域の p-fast trie が存在することを示しました。

今週は、このクエリ・操作時間を同じ $O(\sqrt{\log u})$ に保ったまま、S の要素数 n に線形な $O(n)$ 領域のデータ構造 q-fast trie について解説します。

q-fast trie [定義]

- q-fast trie：p-fast trie の領域計算量を $O(n\sqrt{\log u}\,2^{\sqrt{\log u}})$ から $O(n)$ に改良したデータ構造
- ある正整数 c について，S をそれぞれ要素数 c〜$2c$-1 の部分集合に分割する
- 各部分集合 S_i について，AVL tree を作り，$\max(S_{i-1}) < z_i \leqq \max(S_i)$ を満たす整数 z_i と対応づける
- この整数の集合 $Z = \{z_1, z_2, \ldots\}$に対する p-fast trie を作る
- この p-fast trie と複数の AVL tree から構成される q-fast trie のサイズを (h, b, c) と書く

では、q-fast trie について解説します。

基本的には、q-fast trie は、p-fast trie と 複数の AVL tree を組み合わせたデータ構造です。

まず、ある正整数 c について，S をそれぞれ要素数 c〜$2c$-1 の部分集合に分割します。

次に、S の各部分集合 S_i に対する AVL tree を作り、それを $\max(S_{i-1}) < z_i \leqq \max(S_i)$ を満たす整数 z_i と対応づけます。

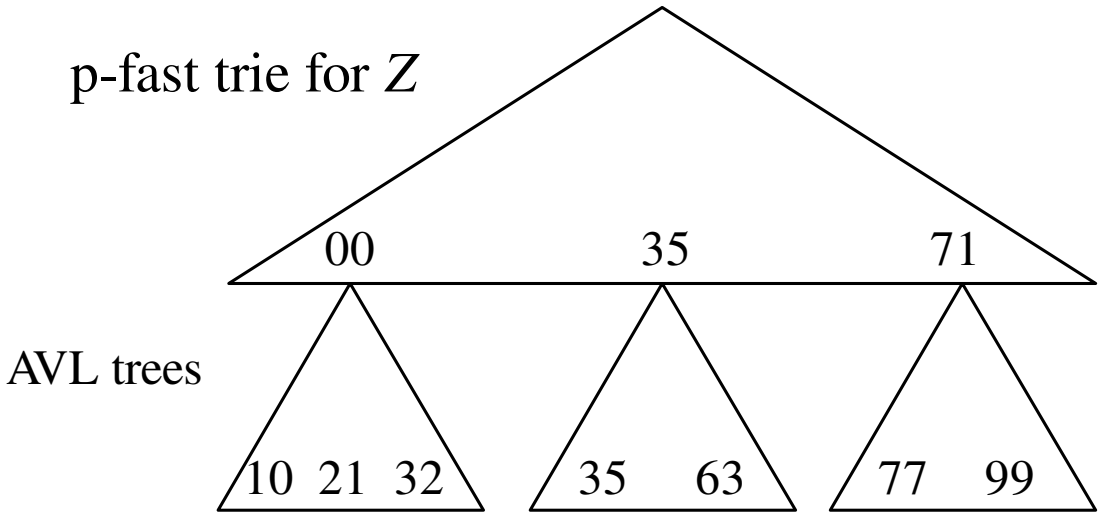
この整数の集合 $Z = \{z_1, z_2, \ldots\}$に対する p-fast trie を作ります。

q-fast trie は、この p-fast trie と、上記の複数の AVL tree から構成されます。

このとき、q-fast trie のサイズを (h, b, c) と書きます。

(h, b) はそれぞれ p-fast trie の高さと最大分岐数です（前回参照）。

q-fast trie [図解]



集合 $S = \{10, 21, 32, 35, 63, 77, 99\}$ に対する
サイズ (2, 10, 2) の q-fast trie

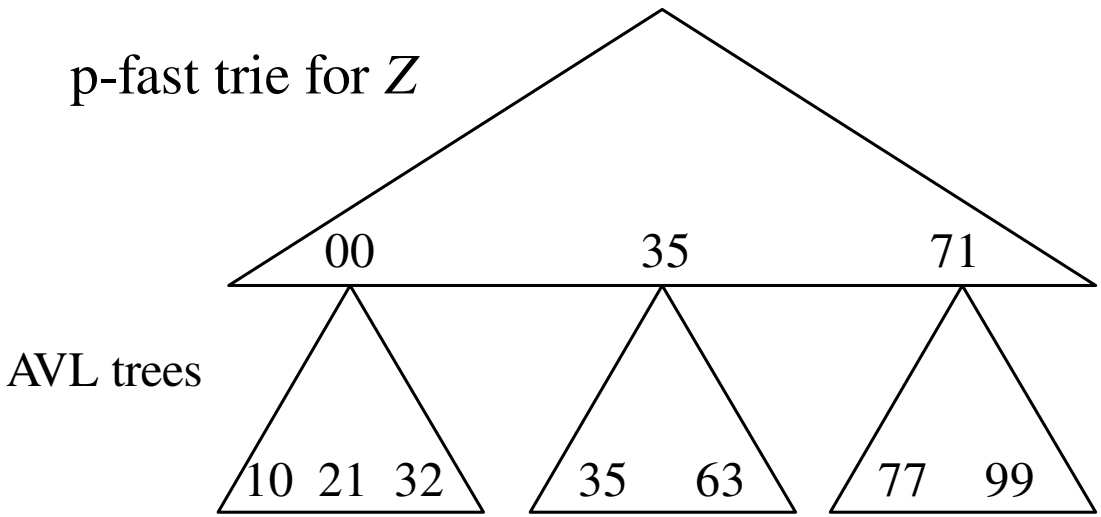
図で示すとこのような感じです。

集合 $S = \{10, 21, 32, 35, 63, 77, 99\}$ に対する サイズ (2, 10, 2) の q-fast trie の概念図です。
それぞれの三角形は、木を表しています。

いま c = 2 なので、S は 2〜3 個の部分集合に分けられていて、それぞれの部分集合は AVL tree で管理されています。

(次スライドへ)

q-fast trie [図解] (続き)



集合 $S = \{10, 21, 32, 35, 63, 77, 99\}$ に対する
サイズ (2, 10, 2) の q-fast trie

各 AVL tree は、直前の部分集合の最大値より大きく、自身の部分集合の最大値以下の値が対応づけられています。

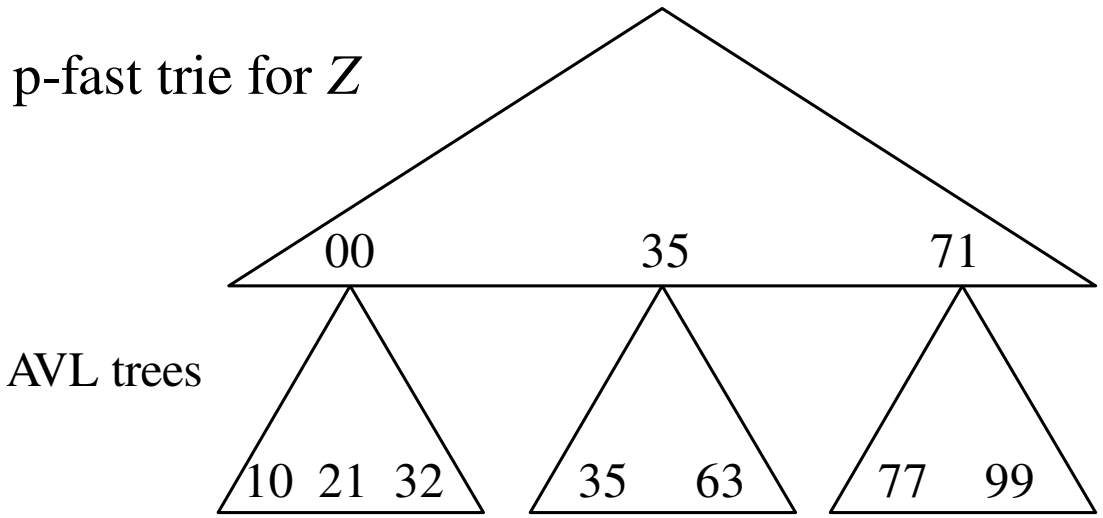
例えば、真ん中の AVL tree には 35 が対応づけられていて、$32 < 35 \leqq 63$ を満たしています。
同様に、左端の AVL tree には 00、右端の AVL tree には 71 が対応づけられています。

この整数の集合 $Z = \{00, 35, 71\}$ に対する p-fast trie を構築して、下の AVL tree と連結して得られるデータ構造が q-fast trie です。

successor with q-fast trie

successor(50, S) の求め方

1. p-fast trie を用いて y = successor(50, Z) を求める



では、この図を用いて、q-fast trie を使った successor クエリの応答方法を解説します。

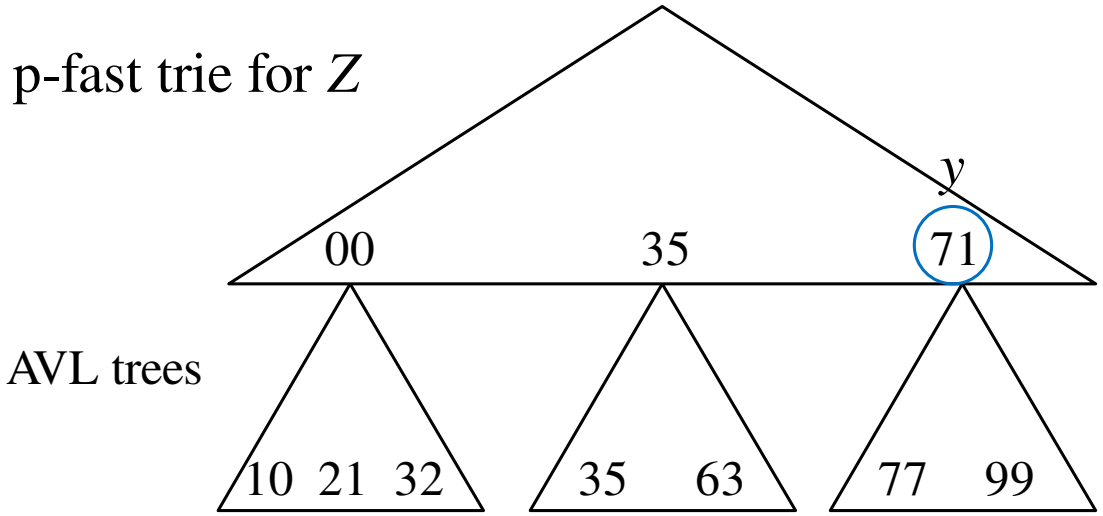
いま S = {10, 21, 32, 35, 63, 77, 99} です。
ここで successor(50, S) を探すことを考えます。

まず、p-fast trie を用いて、successor(50, Z) を探します。

successor with q-fast trie (続き)

successor(50, S) の求め方

1. p-fast trie を用いて y = successor(50, Z) を求める

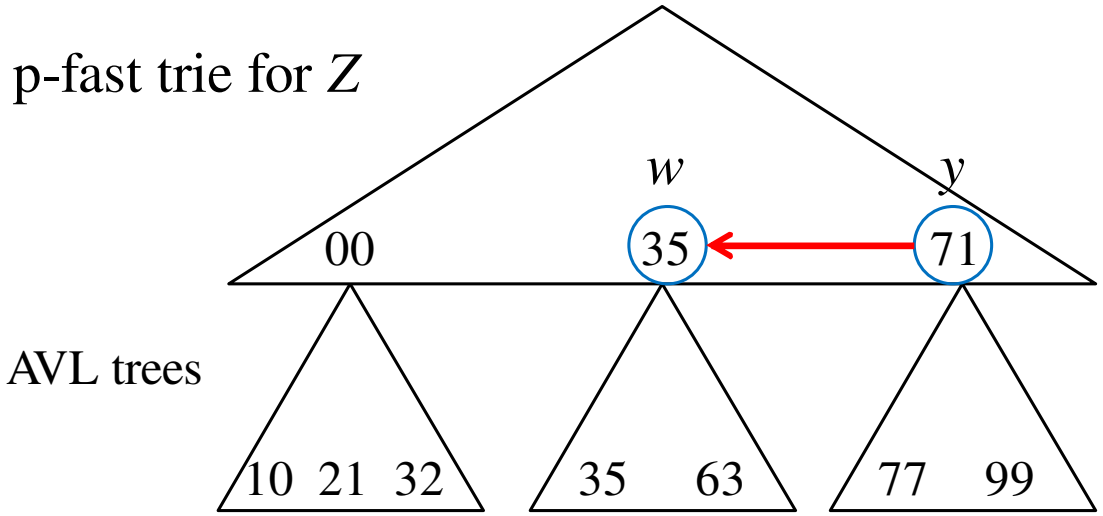


Z = {00, 35, 71} なので、
successor(50, Z) = 71 です。これを y とおきます（y = 71）

successor with q-fast trie (続き)

successor(50, S) の求め方

2. 連結リストを用いて w = predecessor(y, Z) を求める



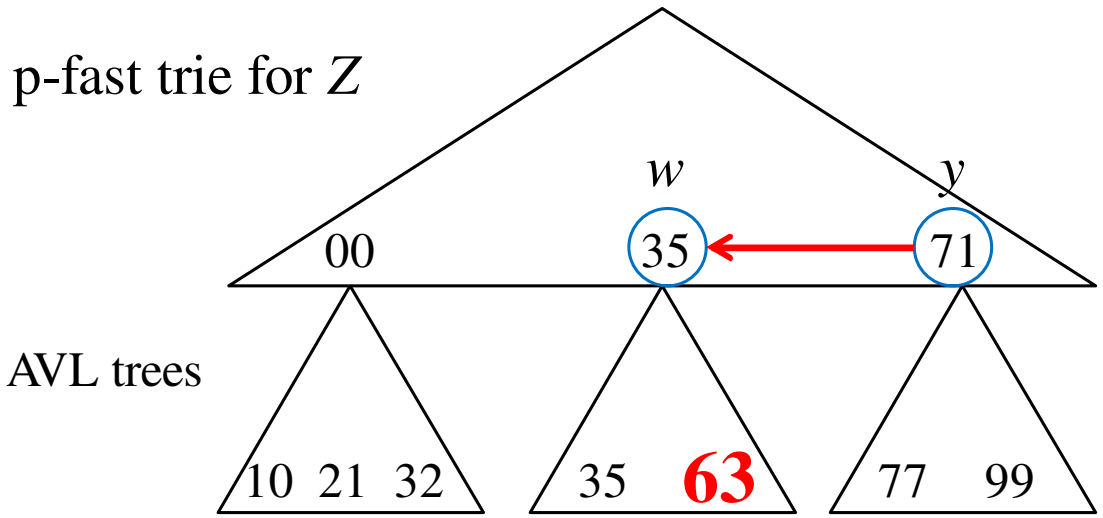
次に、predecessor(y, Z) を求めます。

いま、y = 71 を表す p-fast trie の葉にいるので、双方向連結リストを辿ることで predecessor(y, Z) = 35 が求まります。
これを w とします (w = 35)

successor with q-fast trie (続き)

successor(50, S) の求め方

3. w と y を根とする AVL tree から successor(50, S) を求める



続いて、w と y を根とする2つの AVL tree を探索することで、successor(50, S) を求めます。場合分けがあります。

(1) w の AVL tree に 50 の successor があるとき、その値を返す

(2) w の AVL tree に 50 の successor がないとき
(2-1) y の AVL tree に 50 の successor があるとき、その値を返す
(2-2) y の AVL tree に 50 の successor がないとき、nil を返す

図は場合 (1) の例で、w 中に 50 の successor 63 があるので、それを返します。

ここでは具体例を用いて説明しましたが、この方法で一般の場合でも、正しく successor を探すことができるのは明らかです。

successor with q-fast trie

補題

サイズ (h, b, c) の q-fast trie を用いて,
successor(x, S) を $O(h + \log b + \log c)$ 時間で
計算できる.
このとき q-fast trie の領域計算量は $O(n(1+hb/c))$.

【証明】

- p-fast trie の探索 ➔ $O(h+\log b)$ 時間
- AVL tree の探索 ➔ $O(\log c)$ 時間
- p-fast trie の領域 ➔ $O(n/c \times bh)$
- AVL tree の領域の合計 ➔ $O(n)$

q-fast trie を使った successor について、左の補題が成り立ちます。

サイズ (h, b, c) の q-fast trie を用いて, successor(x, S) を O(h + log b + log c) 時間で計算できます。このとき q-fast trie の領域計算量は O(n(1+hb/c)) となります。

まず時間計算量を証明します。

前回の講義で解説したように、p-fast trie の探索は O(h + log b) 時間です。

それぞれの AVL tree は c 以上 2c-1 以下の要素を含むので、その高さは O(log c) です。よって、AVL tree の探索は O(log c) 時間です。

この2つを足して、O(h + log b + log c) 時間で successor を計算できる、ということになります。

successor with q-fast trie (続き)

補題

サイズ (h, b, c) の q-fast trie を用いて,
successor(x, S) を $O(h + \log b + \log c)$ 時間で
計算できる.
このとき q-fast trie の領域計算量は $O(n(1+hb/c))$.

【証明】

- p-fast trie の探索 ➔ $O(h+\log b)$ 時間
- AVL tree の探索 ➔ $O(\log c)$ 時間
- p-fast trie の領域 ➔ $O(n/c \times bh)$
- AVL tree の領域の合計 ➔ $O(n)$

次に領域についてです。

それぞれの部分集合は少なくとも c 個の要素を含むので、高々 n/c 個の部分集合が存在します。よって、AVL tree の個数も高々 n/c です。

前回の講義で、要素数 m の p-fast trie の領域は O(mbh) であることを解説しました。

Z の要素数は AVL tree の個数に等しいので、m = n/c となり、Z に対する p-fast trie の領域は O(n/c × bh) であることがわかります。

すべての AVL tree の頂点数は明らかに S の要素数 n に等しいです。よって、AVL tree の領域は合計で O(n) になります。

足し合わせて、領域は O(n(1 + hb/c)) ということになります。

successor with q-fast trie

定理

successor(x, S) を$O(\sqrt{\log u})$ 時間で計算できる $O(n)$ 領域の q-fast trie が存在する.

【証明】

- $b = 2^{\sqrt{\log u}}$, $h = \sqrt{\log u}$, $c = hb = \sqrt{\log u}\, 2^{\sqrt{\log u}}$ とすると,
  補題より $O(h + \log b + \log c) = O(\sqrt{\log u})$ 時間
  領域は $O(n(1+hb/c)) = O(n)$

前ページの補題から、直ちにこの定理が得られます。

successor(x, S) を O(√log u) 時間で計算できる O(n) 領域の q-fast trie が存在します.

パラメタ b と h の値は、前回の p-fast trie と同様に $b = 2^{\sqrt{\log u}}$, $h = \sqrt{\log u}$ と設定します。

また、パラメタ c を c = hb とします。

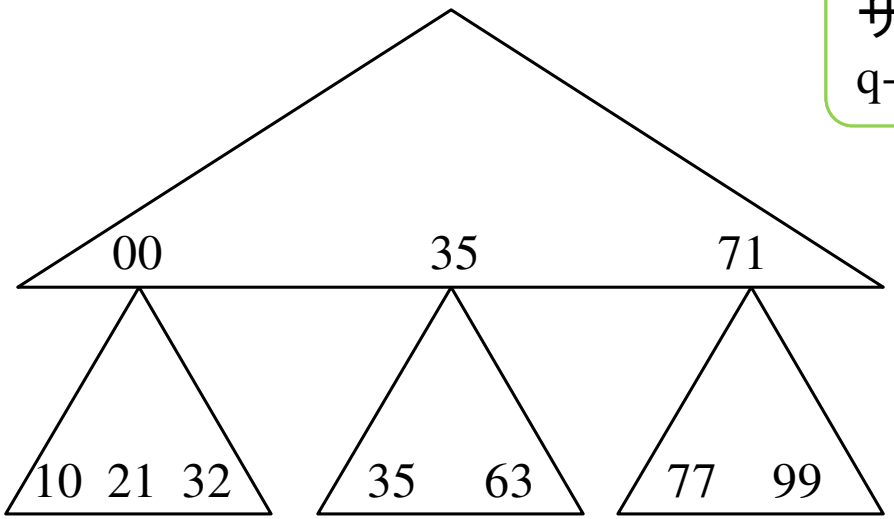
前ページの補題から、successor の時間は
O(h + log b + log c)
= O(h + log b + log (hb))
= O(h + log b + log h + log b)
= O(h + log b)
= O(√log u)
となります。

また、c = hb なので、領域は
O(n(1+hb/c)) = O(n) となります。

以上でこの定理の証明は終わりです。

insertion to q-fast trie

- insertion によって AVL tree のサイズが $2c$ に なったら，サイズ c の 2つの AVL tree に分割
- insert(5, S)



サイズ (2, 10, 2) の q-fast trie

続いて、q-fast trie を用いた要素の追加 (insertion) を考えていきましょう。

左図に示すサイズ (2, 10, 2) の q-fast trie を使って説明します。

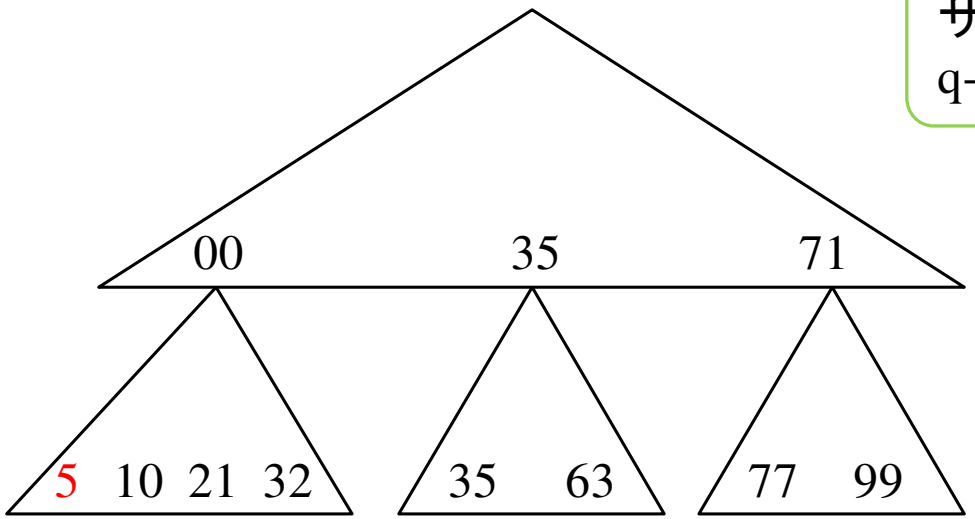
c = 2 であることを覚えておきましょう。
(p-fast trie の高さ h = 2 と分岐の最大数 10 は図には陽に示していませんが、以降の解説には影響ありません)

この q-fast trie に新たな要素 5 を追加する insert(5, S) を考えます。

insertion to q-fast trie

- insertion によって AVL tree のサイズが $2c$ に なったら，サイズ c の 2つの AVL tree に分割
- insert(5, S)

サイズ (2, 10, 2) の q-fast trie



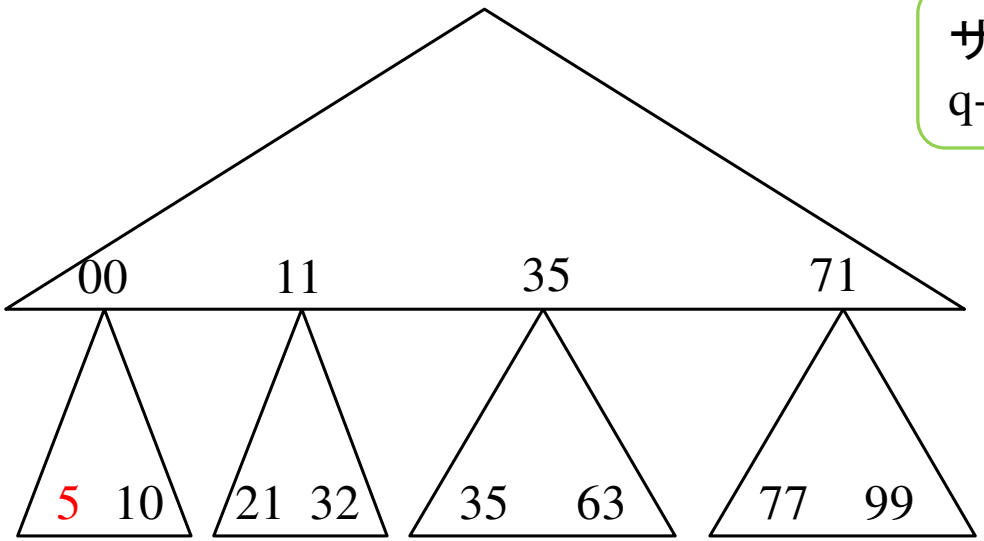
5 は S = {10, 21, 32, 35, 63, 77, 99} のどの要素よりも小さいので、左図のように、一番左の AVL tree に格納されることになります。

ここで、c = 2 なので、一番左の AVL tree は 2c = 4 個の要素を含んでしまいました。

q-fast trie の定義では、各 AVL tree は高々 2c-1 = 3 個の要素しか含んではいけない決まりだったので、この AVL tree をサイズ c の2つの AVL tree に分割します。

insertion to q-fast trie

- insertion によって AVL tree のサイズが $2c$ に なったら，サイズ c の 2つの AVL tree に分割
- insert(5, S)



サイズ (2, 10, 2) の q-fast trie

分割後の AVL tree です。

{5, 10} に対する AVL tree には、元々の値 00 を対応付けておきます。

一方、{21, 32} に対する AVL tree には、定義を満たす任意の値（ここでは 11）を対応付けます。(10 < 11 ≦ 32 を満たしていることを確認しましょう)

最後に、この 11 を p-fast trie が管理する集合 Z に追加します。

以上が q-fast trie を使った insert のアルゴリズムです。

insertion to q-fast trie

定理

q-fast trie における insert(x, S) に要する時間計算量は $O(\sqrt{\log u})$ である.

【証明】

- p-fast trie での探索 ➔ $O(h + \log b)$ 時間
- p-fast trie での lowkey/highkey の更新 ➔ $O(h)$ 時間
- AVL tree への挿入➔ $O(\log c)$ 時間
- AVL tree の分割 ➔ $O(\log c)$ 時間

q-fast trie 上での insert 操作に関して、この定理が成り立ちます。

すなわち、q-fast trie 上での insert に要する時間計算量は O(√log u) で抑えられます。

証明の流れは以下のようになっています。

まず、新たな要素 x が入るべき AVL tree を探すために、p-fast trie を探索します。
これは O(h + log b) 時間でできます。

また、前ページの例のように、x の追加によって Z に新たな要素が加わる場合があります。
このとき、p-fast trie で辿ったパス上にある頂点の lowkey / highkey を更新が生じる可能性がありますが、これは O(h) 時間でできます。

AVL tree への x の挿入は O(log c) 時間でできます。

(次スライドへ)

insertion to q-fast trie (続き)

定理

q-fast trie における insert(x, S) に要する時間計算量は $O(\sqrt{\log u})$ である.

【証明】

- p-fast trie での探索 ➔ $O(h + \log b)$ 時間
- p-fast trie での lowkey/highkey の更新 ➔ $O(h)$ 時間
- AVL tree への挿入➔ $O(\log c)$ 時間
- AVL tree の分割 ➔ $O(\log c)$ 時間
  どうやればこの時間でできるでしょう？

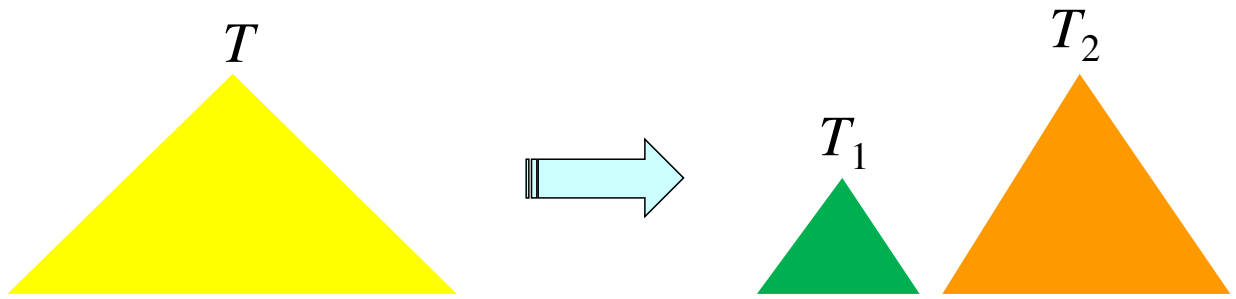
最後に、AVL tree の要素数が 2c になった場合に、先ほどの例のように、ALV tree を分割する必要があります。

仮にこの分割を O(log c) 時間でできるとすると、合計で O(h + log b + log c) = O(√log u) 時間で q-fast trie 上の insert を実現できることになります。

残る課題は、AVL tree を O(log c) 時間で分割する方法です。

演習問題【AVL tree の分割】

- 集合 $X = \{x_1, x_2, ..., x_k\}$ に対する AVL-tree を T とする.
- T を $X_1 = \{x_1, ..., x_j\}, X_2 = \{x_{j+1}, ..., x_k\}$ を格納する2つの AVL tree T_1 と T_2 に分割する.
  - ただし $j < k$ であり, X_1 の各要素は X_2 のどの要素よりも小さい.
- この分割を $O(\log k)$ 時間で行うことができる.
  そのアイディアを示せ.



今回の演習は、この AVL tree の分割方法についてです。

前ページの問題をこのように一般化します。

T を集合 X = {x_1, x_2, ..., x_k} に対する AVL tree とします。
この集合 X の要素は、昇順に並んでいるとします（つまり、x_1 < x_2 < ... < x_k）

ある j が与えられたとき、X の前半 j 個を含む部分集合を X_1、残りの後半部分を含む部分集合を X_2 とします。

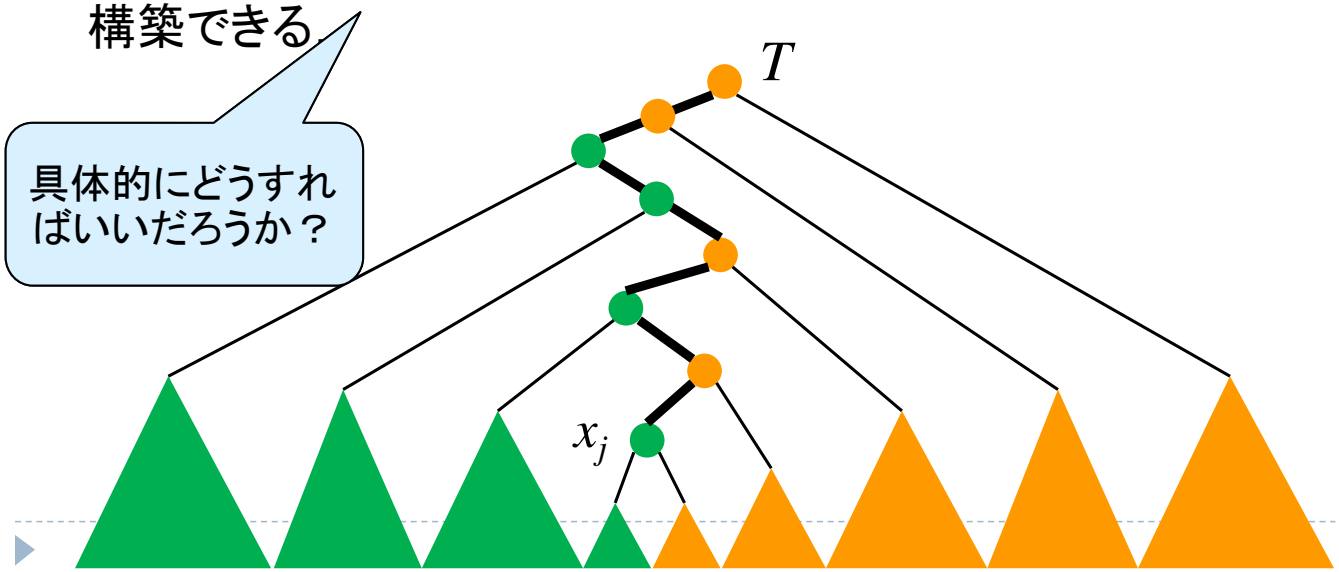
X に対する AVL tree T を、X_1 に対する AVL tree T_1 と、X_2 に対する AVL tree X_2 に分割します。

この分割を O(log k) 時間で行うことができます（ここで k = |X|）

その方法のアイディアを示してください。

ヒント

- T 上で x_j を探索したときのパスを考える.
- このパスで T を分割することによって, x_j 以下の要素を含む部分木(緑)と, x_jより大きい要素を含む部分木(橙)がそれぞれ $O(\log k)$ 個できる.
- これらを 上手く結合することで, $O(\log k)$ 時間で T_1 と T_2を構築できる.



この演習問題は難易度が高めなので、ヒントを出します。

ALV tree T の上で x_j (X_1 の最大の要素)を探索したときのパスを考えます。
図では、このパスを太線で表しています。

このパスを境にして、x_j 以下の要素を含む部分木(緑部分)と、x_j より大きい要素を含む部分木(オレンジ)に分けることができます。

このとき、緑の部分木の個数、およびオレンジの部分木の個数は、どちらも O(log k) です。
(まずこの理由を考察してみましょう)

さらに、これらを上手く結合することで、O(log k) 時間で T_1 と T_2 を構築することができます。

この「上手い結合」の方法を、第2回で学んだAVL tree の性質をフルに活用することで実現できます。

その具体的な方法を解説してください、というのが今回の演習課題です。

お知らせ

- 今回の演習課題はやや難易度の高い問題なので，
  解答のための時間を2週間とり．

  演習問題の**提出〆切を６／25（木）23：59**とします。

- これにともない，来週６／19(金) は **休講** とします。

今回の演習課題はやや難易度の高い問題なので，
解答のための時間を2週間とり，

演習問題の**提出〆切を６／25（木）23：59**とします。
(次のページの提出方法を守ってください)

これにともない，来週 ６／19 (金) は **休講** とします。

今週の講義内容は以上です。

演習問題の pdf について

以下の要領に従って演習問題の pdf を作成してください。

- ファイル名は 学籍番号_lastname.pdf とする。
  例）2IE00099Z_inenaga.pdf
- pdf の1ページ目の最初に
  「第XX回 演習問題」「学籍番号」「氏名」を明記する。
- 留学生は英語で作成しても構いません。
  Foreign students may write their answers in English
  if they feel it easier and more comfortable.

演習問題の pdf を作成する際には、この要領に従ってください。