整数の集合を効率的に管理するデータ構造

2 年 4 組 20 番 児玉大樹

1　はじめに

プログラミング言語 C++ には、全順序集合を効率的に管理するためのデータ構造として、標準ライブラリに `std::set` が実装されている。`std::set` の各種操作にかかる時間計算量は、要素数を N として $\Theta(\log N)$ だが、管理する集合があらかじめ上界の与えられた非負整数の集合ならば、van Emde Boas tree（以下、vEB tree と略記する）と呼ばれるデータ構造を用いてより高速に操作を行うことができる。本稿では、vEB tree を実装し、`std::set` と性能を比較する。

2　導入

2.1　計算量

計算量は、アルゴリズムやデータ構造の効率を評価するための尺度であり、**ランダウ記法**を用いて「入力サイズ n に対し、このアルゴリズムは時間計算量 $O(n)$ で動作する」のように記述される。以下に本稿で用いるランダウ記法の定義を示す。

- $f(n) = O(g(n))$ であるとは、ある定数 C および自然数 n_0 が存在して、n_0 より大きい任意の自然数 n に対し $|f(n)| \leq C|g(n)|$ が成り立つことをいう。
- $f(n) = \Omega(g(n))$ であるとは、ある定数 C および自然数 n_0 が存在して、n_0 より大きい任意の自然数 n に対し $|f(n)| \geq C|g(n)|$ が成り立つことをいう。
- $f(n) = \Theta(g(n))$ であるとは、$f(n) = O(g(n))$ かつ $f(n) = \Omega(g(n))$ であることをいう。

2.2　全順序集合に対する操作

管理する集合を S とおく。本稿では、以下の 5 つの操作を扱う。

- INSERT(x)
  $x \notin S$ ならば S に x を追加する。$x \in S$ ならば何もしない。
- DELETE(x)
  $x \in S$ ならば S から x を取り除く。$x \notin S$ ならば何もしない。
- MEMBER(x)
  $x \in S$ かどうか判定する。
- PREDECESSOR(x)
  S に含まれる x 以下の要素のうち最大のものを求める。存在しないならばそのことを報告する。
- SUCCESSOR(x)
  S に含まれる x 以上の要素のうち最小のものを求める。存在しないならばそのことを報告する。

2.3 集合に対する制約

vEB tree を適用するため、ある自然数 U が存在して、S は $\{0, 1, \ldots, U-1\}$ の部分集合であると仮定する。この U を**普遍集合サイズ**と呼ぶ。vEB tree の計算量は U に依存する。

3 `std::set` の仕組み

3.1 二分木

二分木とは、根付き木であって、全ての頂点が高々 2 つの子を持つものである。2 つの子はそれぞれ**左の子、右の子**と呼ばれる。頂点がただ 1 つの子を持つ場合、左の子、右の子のうちいずれか一方の子を持つ。

3.2 二分探索木

二分探索木とは、二分木であって、頂点に値が対応しており、全ての頂点 n について以下の条件がいずれも成り立つものである。

- n の左の子が存在するならば、その値は n の値以下である。
- n の右の子が存在するならば、その値は n の値以上である。

3.3 平衡二分探索木

二分探索木は、頂点数を N とおくと、深さが最大で $\Theta(N)$ になる。二分探索木の条件を保ったまま、深さを小さく維持しようとするのが**平衡二分探索木**である。`std::set` は平衡二分探索木を用いて実装されている。

`std::set` の最悪時間計算量は、集合の要素数を N として $\Theta(\log N)$ である。また、空間計算量は $\Theta(N)$ である。

4 vEB tree

4.1 vEB tree の構造

簡単のため $U = 2^K$ とする。$K \leq 1$ すなわち $U \leq 2$ のときは、長さ U の配列を用いて、$x = 0, \ldots, U-1$ それぞれについて $x \in S$ かどうかを管理しておくことにより、全ての操作を $O(1)$ の計算量で行うことができる。以降は $K \geq 2$ とする。

$U_{\mathrm{low}} = 2^{\lfloor \frac{K}{2} \rfloor}, U_{\mathrm{high}} = 2^{\lceil \frac{K}{2} \rceil}$ とおく。普遍集合サイズ u の vEB tree を $\mathrm{vEB}(u)$ と表すことにすると、$\mathrm{vEB}(U)$ は以下の情報から構成される。

- min：S の最小値（存在しない場合は NIL）
- max：S の最大値（存在しない場合は NIL）
- cluster：U_{high} 個の $\mathrm{vEB}(U_{\mathrm{low}})$ を指すポインタ
- summary：cluster の情報をまとめた $\mathrm{vEB}(U_{\mathrm{high}})$

以下、$i = 0, \ldots, U_{\mathrm{high}} - 1$ に対し、$\mathrm{cluster}_i$ で $i+1$ 番目のポインタが指す $\mathrm{vEB}(U_{\mathrm{low}})$ を表すことにする。また、$x \in \{0, \ldots, U-1\}$ に対し、x を U_{low} で割った商を $\mathrm{high}(x)$、余りを $\mathrm{low}(x)$ とおく。

$\mathrm{cluster}_0, \ldots, \mathrm{cluster}_{U_{\mathrm{high}}-1}$ は $S \setminus \{\mathrm{min}\}$ の要素を管理する。$x \in S \setminus \{\mathrm{min}\}$ は、$\mathrm{cluster}_{\mathrm{high}(x)}$ 内に $\mathrm{low}(x)$ として管理する。

summary は $i \in \{0, \ldots, U_{\mathrm{high}} - 1\}$ かつ $\mathrm{cluster}_i$ が管理する集合が空でないような i の集合を管理する vEB tree である。

4.2 各種操作の実現

4.2.1 INSERT(x)

4.2.2 DELETE(x)

4.2.3 MEMBER(x)

4.2.4 PREDECESSOR(x)

4.2.5 SUCCESSOR(x)