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 [1]。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 [1]。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 [2]。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 が左の子を持つならば、その部分木に含まれる頂点の値は n の値以下である。
- n の右の子を持つならば、その部分木に含まれる頂点の値は 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 [2]。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` は通常、平衡二分探索木の一種である赤黒木を用いて実装されている [3]。`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、普遍集合サイズ U がある非負整数 K に対し $U = 2^K$ と表されると仮定する。U がこのように表されない場合についても、普遍集合サイズを $U' = 2^{\lceil \log_2 U \rceil}$ で置き換えれば、$U \leq U' \leq 2U$ より計算量には影響しない。

$K \leq 1$ 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。

$U_{\text{low}} = 2^{\lfloor \frac{K}{2} \rfloor}, U_{\text{high}} = 2^{\lceil \frac{K}{2} \rceil}$ とおく。普遍集合サイズ u の vEB tree を vEB(u) と表すことにすると、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 vEB(U_{low}) を指すポインタ
- summary：cluster の情報をまとめた vEB(U_{high})

「何もない」ことを表すための値として、NIL を導入した。$a = \text{NIL}$ かつ $b \neq \text{NIL}$ ならば、$a = b, a < b, a > b$ のいずれも偽であるとする。

以下、$i = 0, \ldots, U_{\text{high}} - 1$ に対し、cluster_i で $i + 1$ 番目のポインタが指す $\text{vEB}(U_{\text{low}})$ 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 $\text{high}(x)$、余りを $\text{low}(x)$ とおく。

$\text{cluster}_0, \ldots, \text{cluster}_{U_{\text{high}}-1}$ は $S \setminus \{\text{min}\}$ の要素を管理する。$x \in S \setminus \{\text{min}\}$ は、$\text{cluster}_{\text{high}(x)}$ 内に $\text{low}(x)$ として管理する。summary は $i \in \{0, \ldots, U_{\text{high}} - 1\}$ かつ 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。

4.2 各種操作の実現

前節で述べた通り、$K \leq 1$ の場合は全ての操作を $O(1)$ の計算量で行うことができる。本節では $K \geq 2$ の場合の操作について説明する。

4.2.1 INSERT(x)

[1] $\text{min} = \text{NIL}$ のとき

min, max を x で置き換え、終了する。

[2] $\text{min} = x$ のとき

既に $x \in S$ であるので、終了する。

[3] $\text{min} \neq \text{NIL}$ かつ $\text{min} \neq x$ のとき

まず、$x < \text{min}$ ならば x と min の値を交換する。

次に $\text{cluster}_{\text{high}(x)}$ に対して $\text{INSERT}(\text{low}(x))$ を呼び出す。呼び出す前の段階で $\text{cluster}_{\text{high}(x)}$ が管理する集合が空だったならば、summary に対して $\text{INSERT}(\text{high}(x))$ を呼び出す。

最後に、$\text{max} < x$ ならば max を x で置き換える。

4.2.2 DELETE(x)

[1] $\text{min} = \text{NIL}$ または $x < \text{min}$ または $\text{max} < x$ のとき

$x \notin S$ であるので、終了する。

[2] $\text{min} = \text{max} = x$ のとき

min, max を NIL で置き換え、終了する。

[3] $\text{min} = x < \text{max}$ のとき

summary の min を i とおき、cluster_i の min を x' とおく。min を $i \times U_{\text{low}} + x'$ で置き換え、cluster_i に対して $\text{DELETE}(x')$ を呼び出す。これにより cluster_i が管理する集合が空になったならば、summary に対して $\text{DELETE}(i)$ を呼び出す。

[4] $\text{min} < x = \text{max}$ のとき

$\text{cluster}_{\text{high}(x)}$ に対して $\text{DELETE}(\text{low}(x))$ を呼び出す。これにより $\text{cluster}_{\text{high}(x)}$ が管理する集合が空になったならば、summary に対して $\text{DELETE}(\text{high}(x))$ を呼び出す。

次に、summary が管理する集合が空になったならば、max を min で置き換える。そうでなければ、summary の max を i、cluster_i の max を x' とおいて、max を $i \times U_{\text{low}} + x'$ で置き換える。

[5] $\text{min} < x < \text{max}$ のとき

$\text{cluster}_{\text{high}(x)}$ に対して $\text{DELETE}(\text{low}(x))$ を呼び出す。これにより $\text{cluster}_{\text{high}(x)}$ が管理する集合が空になったならば、summary に対して $\text{DELETE}(\text{high}(x))$ を呼び出す。

4.2.3 MEMBER(x)

[1] min = NIL または $x <$ min または max $< x$ のとき
$x \notin S$ と報告する。
[2] min $= x$ または max $= x$ のとき
$x \in S$ と報告する。
[3] min $< x <$ max のとき
$\text{cluster}_{\text{high}(x)}$ に対して MEMBER(low(x)) を呼び出した結果を報告する。

4.2.4 PREDECESSOR(x)

[1] min = NIL または $x <$ min のとき
NIL を返す。
[2] max $\leq x$ のとき
max を返す。
[3] min $\leq x <$ max のとき
$\text{cluster}_{\text{high}(x)}$ の min が low(x) 以下ならば、$\text{cluster}_{\text{high}(x)}$ に対して PREDECESSOR(low(x)) を呼び出した結果 x' とおいて、high(x) $\times U_{\text{low}} + x'$ を返す。
そうでないとき、summary に対して PREDECESSOR(high(x) -1) を呼び出した結果を i とおく。$i =$ NIL ならば min を返し、$i \neq$ NIL ならば cluster_i の max を x' とおいて、$i \times U_{\text{low}} + x'$ を返す。

4.2.5 SUCCESSOR(x)

[1] max = NIL または max $< x$ のとき
NIL を返す。
[2] $x \leq$ min のとき
min を返す。
[3] min $< x \leq$ max のとき
$\text{cluster}_{\text{high}(x)}$ の max が low(x) 以上ならば、$\text{cluster}_{\text{high}(x)}$ に対して SUCCESSOR(low(x)) を呼び出した結果 x' とおいて、high(x) $\times U_{\text{low}} + x'$ を返す。
そうでないとき、summary に対して SUCCESSOR(high(x) $+1$) を呼び出した結果を i、cluster_i の min を x' とおいて、$i \times U_{\text{low}} + x'$ を返す。

4.3 計算量解析

4.3.1 時間計算量

前述した関数において、2 つ以上の関数を呼び出して再帰を行う場合がある。しかし、管理している集合の要素数が 1 以下のときに $O(1)$ で処理を行うことができることに注目し、これらを除外すると、高々 1 つの関数しか呼び出さないことがわかる。
よって、時間計算量は再帰の深さにのみ依存する。vEB(U) が管理する vEB tree の普遍集合サイズは $2^{\lceil \frac{K}{2} \rceil}$ 以下なので、再帰の深さは以下の値で上から抑えられる。

- K を $\lceil \frac{K}{2} \rceil$ で置き換えることを繰り返すとき、K が 1 以下になるまでに必要な置き換えの回数。

$K \geq 2$ ならば $\lceil \frac{K}{2} \rceil \leq \frac{2K}{3}$ であるから、この値は $\log_{\frac{3}{2}} K$ 以下である。これと $K = \log_2 U$ から、再帰の深さは $O(\log \log U)$ であることがわかる。以上から、vEB tree の各種操作にかかる時間計算量は $O(\log \log U)$ である。

4.3.2 空間計算量

全ての $x \in S$ に対し、$\min = x$ となるような vEB tree がただ一つ存在するため、空でない集合を管理している vEB tree は $|S|$ 個である。よって、cluster を管理するのにハッシュテーブルなどの連想配列を用いることで、空間計算量を $O(|S|)$ にすることができる。しかし、理論的な時間計算量は $O(\log \log U)$ と変わらないが実用上のパフォーマンスは落ちてしまうため、本稿では空間計算量の改善を行わず、$\Theta(U)$ のものを実装する。

5 計測

Library Checker というサイトでテストケースが用意されているため、そちらを利用する。`std::set` を用いたコード、vEB tree を用いたコード、入出力のみを行うコードを提出した。以下に URL を記載する。

`std::set` `https://judge.yosupo.jp/submission/101567`

vEB tree `https://judge.yosupo.jp/submission/101579`

入出力のみ `https://judge.yosupo.jp/submission/101636`

実行結果は `https://github.com/KodamaD/Nada2022Report/blob/main/docs/data.md` にまとめた。入出力にかかる時間を除いた実行時間を比較すると、$|S|$ が大きいテストケースでは vEB tree が約 4 倍から 10 倍高速であるのに対し、$|S|$ が小さいテストケースでは `std::set` の方が高速であった。原因としては、vEB tree の構築にかかる時間が長いことが考えられる。vEB tree は $|S|$ ではなく U に計算量が依存するので、集合に対する操作がほとんどないテストケースにおいても構築に約 133 ミリ秒を要している。

6 さらなる高速化

これまでは $K \leq 1$ の場合のみに長さ U の配列を用いて集合を管理していたが、これを $K \leq 6$ の場合に拡張し、配列ではなく 64 bit 整数を用いて管理することで高速化を図る。bit 演算と GCC の組み込み関数を用いて各種操作を高速に実現することができる。

この高速化を施したコードは `https://judge.yosupo.jp/submission/101582` である。実行結果は前節に記載した表にまとめた。高速化を施す前のコードと比較すると、約 2.5 倍から 4 倍高速になっていることがわかる。$|S|$ が大きいテストケースでは、`std::set` の約 10 倍から 50 倍ほど高速である。依然として $|S|$ が小さいテストケースでは `std::set` の方が高速だが、構築にかかる時間を 33 ミリ秒に落とすことに成功した。

7 まとめ

vEB tree を実装し、`std::set` と性能を比較した。$|S|$ が大きいテストケースでは vEB tree が非常に高速であるのに対し、$|S|$ が小さいテストケースでは vEB tree の構築にかかる時間がボトルネックになり、`std::set` の方が高速であった。

vEB tree と同等の計算量を実現するデータ構造として、Y-Fast Trie と呼ばれるものが存在する [2]。今後はこのようなデータ構造を実装し、vEB tree や `std::set` との性能を比較したい。

参考文献

[1] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein (著)、浅野哲夫・岩野和生・梅尾博司・山下雅文 (訳)『アルゴリズムイントロダクション』第 3 版 近代科学社 2013 年

[2] Pat Morin（著)、堀江慧・陣内佑・田中康隆（訳)『みんなのデータ構造』ラムダノート株式会社 2018 年

[3] cppreference.com “std::set”
(`https://en.cppreference.com/w/cpp/container/set`)