やるだけ

@catupper

目次

目次 3

第 I 章　やるだけについて 5
- 1　「やるだけ」とは？ 5
- 2　この記事のコンセプト 5
- 3　おわび 6

第 II 章　やるのは一回だけで充分 7
- 1　この章でやること 7
- 2　ポテンシャル解析 7
- 3　フィボナッチヒープ 9
- 4　平衡 2 分探索木　スプレー木 12
- 5　まとめ 15

参考文献 17

第 1 章

やるだけについて

「この問題やるだけですよ。」

ある界隈の人々

1 「やるだけ」とは？

そもそも「やるだけ」という言葉は、特に考えなくてもやれば解けるくらいに簡単な問題を評価するときの言葉です。他人が答えをだすのに苦しんでいる問題に対して「やるだけ」と評価することは、俗に「やるだけハラスメント」と言われています。ある界隈[1]で「やるだけ」と言うと、もっぱら後者のような使われかたをしています。やるだけハラスメントを受けた人は非常に心が冷え込みますが、やるだけと言うほうもそれなりの実力が必要で、ある意味名誉ある行為なのです。むやみやたらに「やるだけ」を連呼しているだけではダメです。しっかりとやるだけで問題を解くところまでやって初めて「やるだけハラスメント」の完成です。

2 この記事のコンセプト

この記事は「やるだけ」という文字列に絡めて、あらゆるアルゴリズムや競技プログラミングの話をしていく物です。内容自体はそこまでハラスメントしていないので、ゆるい気分で読んでいただけると思います。まぁ、この記事は書くだけでしたけどね！[2]

[1] 競技プログラミング界隈だけでなく数学界隈でも横行しているらしい。
[2] 参考までに、この部誌は提出期限を 2 ヶ月オーバーして完成しました。

3　おわび

そういうコンセプトで色々なことを書いていくつもりで、「やるだけ問題集/回答集」という章がもう一つ付くつもりだったのですが、筆者の都合で、出す予定だった問題を公開できなくなってしまいました。それ故に次の 2 章でこの記事は終わりです。こんなイントロダクションに章を 1 つ割いておきながら次で終わりです。やるだけではありませんでした。ごめんなさい。

第 II 章

やるのは一回だけで充分

「一回で充分です！!」

じょしらく第 5 巻

1　この章でやること

この章では、競技プログラミングをする上では特にやっておく必要のないこと、一回やっておけば充分なものについて話します。つまるところ、計算量解析の仕方や、アルゴリズムの証明などです。この記事は特に、ポテンシャルを用いた計算量解析の仕方と、スプレー木のスプレーのならし計算量が $O(\log(n))$ であることを示します。

2　ポテンシャル解析

2a　償却計算量

ポテンシャル解析は計算量解析の中でもデータ構造の**償却計算量 (amortized complexity)** を求めるための手法です。償却計算量はならし計算量とも呼ばれます。ポテンシャル解析では償却計算量は**ポテンシャル関数**によって決められます。ポテンシャル関数はデータ構造の状態に対してある数値を返す関数です。ある操作をした時の償却計算量を $T_{\text{amortized}}$,

実際の計算量を T_{actual}, 操作前後の状態を $S_{\text{before}}, S_{\text{after}}$, ポテンシャル関数を $\Phi(S)$ とします。このとき以下の関係式が成り立ちます。

$$T_{\text{amortized}} = T_{\text{actual}} + C\dot{(}\Phi(S_{\text{after}}) - \Phi(S_{\text{before}})) \tag{II.1}$$

C は定数です。つまり償却計算量のオーダーは実計算量とポテンシャルの増加分の和のオーダーと等しいです。複数回の操作を行ったときの実計算量が償却計算量の総和と最初と最後の状態のポテンシャルだけで求めることができます。i 回目の操作の実計算量, 償却計算量を $T_{\text{actual},i}, T_{\text{amortized},i}$、$i$ 回目の操作が終わったあとの状態を S_i とすると、q 回の

処理を行ったあとの総実計算量は

$$\begin{aligned}\sum_{i=0}^{q} T_{\mathrm{actual},i} &= \sum_{i=1}^{q} \{T_{\mathrm{amortized},i} - C \cdot (\Phi(S_i) - \Phi(S_{i-1}))\} \\ &= \sum_{i=1}^{q} T_{\mathrm{amortized},i} - C \cdot (\Phi(S_1) - \Phi(S_0) + \Phi(S_2) - \Phi(S_1) + \cdots + \Phi(S_q) - \Phi(S_{q-1})) \\ &= \sum_{i=1}^{q} T_{\mathrm{amoritzed},i} - C \cdot (\Phi(S_q) - \Phi(S_0)) \qquad \text{(II.2)}\end{aligned}$$

つまりどのような順番で操作をしても、実計算量と償却計算量の誤差は最初と最後のポテンシャルの差に一致します。ここで、操作前と操作後のポテンシャルの増加量が操作の数に対してそこまで大きくならないようにポテンシャル関数をうまく定義してやると、各操作の計算量を償却計算量で評価してやっても問題無いということになります。このようにして、償却計算量をもとめるのがポテンシャル解析です。

2b　具体例

この節では具体的に、**動的配列 (dinamic array)** の計算量をポテンシャル解析で求めます。動的配列は、ランダムアクセス[1]ができ、末尾に値を追加するということができる配列です。常に 2 の累乗のサイズのメモリを確保しており、それが溢れたら更に今の 2 倍のサイズののメモリを動的に確保し、そこに現在持っている要素を全部コピーするというものです。できる操作と実計算量をまとめると以下のようになります。

操作	実計算量
ランダムアクセス	O(1)
通常の末尾への値の追加	O(1)
2 倍のメモリを確保してコピーして末尾へ追加	O(N)

まずはポテンシャル関数を決めましょう。この配列が確保しているサイズを N, 持っている値の数 (要素数) を n とします。ポテンシャル関数 Φ は

$$\Phi = 2n - N$$

です。それでは、各操作の償却計算量を求めていきましょう。先ほど求めたように償却計算量は実計算量とポテンシャル増加量の和です。

1 つ目のランダムアクセスは、操作前と操作後でポテンシャルは変わりません。なので償却計算量は $O(1)$ です。2 つ目の通常の末尾への値の追加はポテンシャルが $2n - N$ から

[1] 任意の要素を $O(1)$ で参照すること。C/C++ における配列の添字参照もランダムアクセス。

$2(n+1)-N$ にかわります。つまりポテンシャルの増加量は 2 です。実計算量は O(1) なので、償却計算量は $O(1)$ となります。3 つ目のメモリを確保してコピーして末尾へ値を追加する操作は、確保したサイズいっぱいに値が入っている時、つまり $n=N$ のときに行われます。よってポテンシャルは $2N-N$ から $2(N+1)-2N$ にかわります。よってポテンシャルの増加量は $2-N$ となります。実計算量は $O(N)$ です。$O(N)$ には定数倍がかかっているかもしれないので $2-N$ を足しても $O(N)$ のままのように見えますが、(II.1) の C で定数倍を調整することで和を $O(1)$ にすることができます。このとき C の調整が他の操作の償却計算量に影響しないことを確認しなければなりません。今回はこれ以外の操作は実計算量もポテンシャルも定数だったので C をいじくっても問題ありません。よって償却計算量は $O(1)$ です。

操作	実計算量	償却計算量
ランダムアクセス	O(1)	O(1)
通常の末尾への値の追加	O(1)	O(1)
2 倍のメモリを確保してコピーして末尾へ追加	O(N)	O(1)

結果的に全ての操作の償却計算量が O(1) でした。はじめ空だった動的配列に q 回の操作を行った時のことを考えます。全部が末尾への追加だったとしても最後の配列の要素数は q です。サイズ N が要素数 n の 2 倍を超えないことは容易に示せるので q 回の操作を終えたあとのポテンシャル Φ は多くとも q です。よって q 回の操作の実計算量は途中で実計算量が $O(N)$ の操作があるにもかかわらず、(II.2) より $O(q)$ であることが示せます。もし最悪計算量で評価していたら途中で発生する $O(N)$ の操作のために、q 回目までのその操作の回数の上限などを評価する必要があります。また、動的配列は操作による遷移が一本道なので各操作の回数の評価が容易いですが、状態の移り方が無数に存在して場合分けでは解けないアルゴリズムもあります。そのようなときでもポテンシャル解析ならやるだけかもしれません。

3 フィボナッチヒープ

それでは、もう少し複雑なデータ構造としてフィボナッチヒープの償却計算量を解析してみましょう。

3a フィボナッチヒープの機能と計算量

フィボナッチヒープはヒープの一種で、値の挿入、削除、最小値検索、値の減算、2 つのヒープのマージ、ができるヒープです。とくに挿入、最小値検索、値の減算、マージの償

却計算量は定数となっており非常に高速です。

3b　フィボナッチヒープの操作の実装

計算量を解析する前に実装を決めておかないといけません。フィボナッチヒープは木のリスト（森）の形でヒープを表します。どのノードも自分の親より大きいか等しい値を持っています。森に含まれる木の根たちは環状の双方向リストとして管理します。ある頂点の子達も同様に管理します。各フィボナッチヒープは森に含まれる木の中で最も頂点の値が小さいものがどれかを覚えておきます。これが、ヒープ内の最小値になります。そして、どの頂点の次数（子の数）も $logN(N$ は頂点) になるようにします。また、各ノードは自分がマークされているかどうかという情報を持ちます。このマークをどのように使うかは後ほど説明します。

マージはやるだけです。2 つの木のリストを結合するだけです。

挿入は挿入したい値だけからなる大きさ 1 のヒープをつくってマージするだけです。

最小値検索もやるだけです。最小値の情報は常に保持しているので参照するだけです。

最小値削除は少し複雑です。3 つの段階に分けられます。まず最小要素の子を全て根に持って行き、元の木のリストに結合します。次に、根が増えすぎるのを防ぐために、根の次数（子の数）が同じ 2 つの木を 1 つにまとめる操作を同じ次数の根がなくなるまで続けます。この操作をすると根が減るだけでなく、全ての頂点の次数が $O(log(N))$ に抑えられます[2]。最後に根を全てチェックしてどれがヒープの中で最小値かを検索します。

値の減算は、減算を実行した後にヒープの形が崩れたら（親の値より小さくなったら）、その頂点を親から切り離し根として森に追加します。もしマークがついていたらマークは消します。このとき、減算した頂点の親が根ではなかったらその親にマークを付けます。もし既にマークがついていたら、それも切り離し森に追加し、マークを消します。そしてそれの親をマークします。というのを親が根になるかマークされてない頂点になるまで根の方向に向かって行い続けます。

値の削除はある値をどの値よりも小さい値まで減算して最小値にした後に最小値削除をするだけです。

これで一応すべての操作の実装を述べました。

[2] なぜ次数が $O(log(N))$ に抑えられるかの証明は割愛させていただきますが、頂点の数について適当な帰納法を利用します。この過程でフィボナッチ数 (指数的に増加する) が出てきます。これがフィボナッチヒープの名前の由来でもあります

3c　フィボナッチヒープのポテンシャル関数

ポテンシャル解析をするためにフィボナッチヒープのポテンシャル関数を決めましょう。フィボナッチヒープが持つ木の数を t、マークされた頂点の数を m とすると

$$\mathrm{Potential} = t + 2m$$

となります。それでは各操作の償却計算量を求めていきましょう。

3d　各操作の償却計算量

マージ、挿入、最小値検索、の償却計算量は簡単です。実計算量は明らかに定数ですし、ポテンシャルも定数しか増えません。よって償却計算量は定数です。
最小値削除では操作が 3 段階にわかれていたので償却計算量をそれぞれ別々に計算しましょう。まず最小値の子をすべて頂点に持っていきます。このとき最小値だった頂点の次数の数だけ根が増えます。よって t は $O(logN)$ だけ増えます。実計算量はリストに頂点を追加していくだけなので $O(logN)$ です。よって償却計算量は $O(logN)$ です。次は同じ次数の根を一つにまとめていく作業です。効率よく同じ次数の根を見つけるために、長さ $O(logN)$ を持つ配列を用意し、それぞれの深さのある根を持つようにします。森の根を次々に配列に追加していき、もし既に他の根で埋まっていたら同じ次数の頂点があるということなので、その 2 つの木をまとめて配列を更新します。このようにすると実計算量 $O(logN + M)$ となります (M は 2 番目の操作を始める前の根の個数)。そして、この操作が終わった後には値の数はたかだか $O(logN)$ 個になっています。よって t は少なくとも $M - O(logN)$ 減ります。よって償却計算量は $O(logN)$ です。最後に根を全てチェックして最小値を更新する作業ですが、特にヒープの構造は変わらないのでポテンシャルは変化しません。実計算量は $O(logN)$ なので償却計算量も $O(logN)$ です。よってこの 3 つの操作をあわせて最小値削除は償却計算量 $O(logN)$ で実行することができます。
値の減算は、減算するノードから根の方に向かってマークされていた頂点がすべて切り離され、根になります。このときいくつか根が増えますが、増えた量を k とします。k この頂点は少なくとも最初の頂点以外はもともとマークされていて、根になることによってマークが消されました。よって t は k 増えて m は少なくとも $k - 1$ 減ります。よってポテンシャル関数は少なくとも $k - 2$ 減ります。実計算量は $O(k)$ なので償却計算量は一定です。
値の削除は値の減算 $O(1)$ のあとに最小値の削除 $O(logN)$ をするので償却計算量は $O(logN)$ です。

これでフィボナッチヒープの各操作の償却計算量が求まりました。やったね。

4　平衡 2 分探索木　スプレー木

4a　スプレー木

スプレー木というのは平衡 2 分探索木の一種です。平衡 2 分探索木については 2013 年度部誌に載っている「木のはなし」に詳しく書かれておりますので、そちらを参照ください。スプレー木は頂点に参照するたびにそれを回転させて根に持っていく (スプレー) というものです。しかもその回転のさせ方は少し特殊で、普通の回転は頂点を 1 つ浅いところに持っていくものですが、スプレー木の回転は深さを 2 つごとに変化させます。

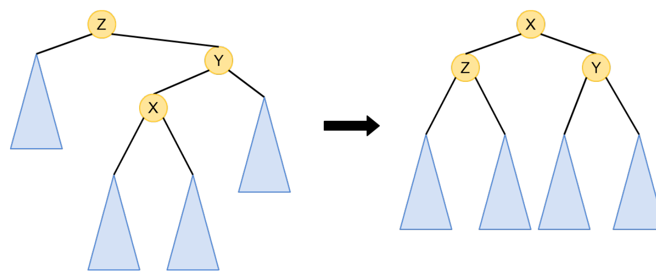

図 II.1　ZigZag

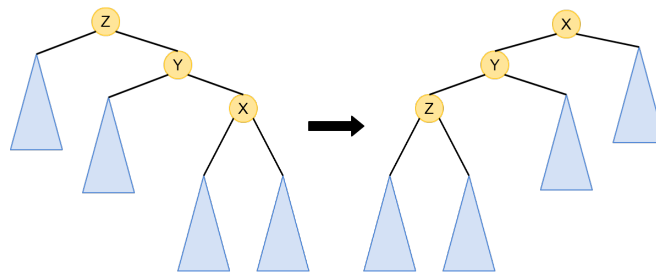

図 II.2　ZigZig

4b　補助関数

スプレーの計算量を求めるために、いくつか頂点を引数に取る補助関数を定義します。

$$\mathrm{size}(x) = x \text{ を根とする部分木の頂点数}$$
$$\mathrm{rank}(x) = \lfloor \log_2(\mathrm{size}(x)) \rfloor$$

ここ以降では、操作の前と後での値を比較することがあります。操作後の値は $\mathrm{size}'(x)$ や $\mathrm{rank}'(x)$ のようにプライムをつけた形で表すことにします。

スプレー木のポテンシャル関数を以下のように定義します。

$$\text{Potential} = \text{全頂点の rank の総和} \tag{II.3}$$

4c　スプレーの償却計算量

頂点を根まで持って行く時に行う操作は Zigzig,、Zigzag、または Zig です。まず、それぞれの償却計算量を求めてみましょう。**Zig** は簡単です。操作によって rank が変わるの

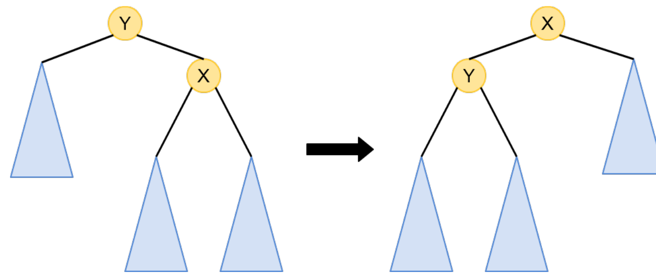

図 II.3　Zig

は、色を付けた 2 つの頂点だけです。それ以外の頂点は自分の子孫の数が変わっていません。回転は 1 回なので実計算量は 1 なので償却計算量は

$$\begin{aligned}
&1 + \text{rank}'(x) + \text{rank}'(y) - \text{rank}(x) - \text{rank}(y) \\
\leq &1 + \text{rank}'(x) - \text{rank}(x) && (\because \text{rank}'(y) \leq \text{rank}(y)) \\
\leq &1 + 3(\text{rank}'(x) - \text{rank}(x)) && (\because \text{rank}'(x) \geq \text{rank}(x))
\end{aligned}$$

評価を $1 + 3(\text{rank}'(x) - \text{rank}(x))$ までゆるくしたのは後々、Zigzig や Zigzag の結果と足し合わせる時に都合が良いからです。

つづいて **Zigzig** の償却計算量も評価してみましょう。Zig と同様に 3 つの頂点の rank しか変わらないのでそれだけに注目すればよいです。今回は 2 回、回転しているので実計算量は 2 です。

$$\begin{aligned}
&2 + \text{rank}'(x) + \text{rank}'(y) + \text{rank}'(z) - \text{rank}(x) - \text{rank}(y) - \text{rank}(z) \\
= &2 + \text{rank}'(y) + \text{rank}'(z) - \text{rank}(x) - \text{rank}(y) && (\because \text{rank}'(x) = \text{rank}(z)) \\
\leq &2 + \text{rank}'(x) + \text{rank}'(z) - 2\text{rank}(x) && (\because \text{rank}'(x) \geq \text{rank}'(y), \text{rank}(y) \geq \text{rank}(x)) \\
\leq &3(\text{rank}'(x) - \text{rank}(x))
\end{aligned} \tag{II.4}$$

最後の式変形をするためには $\text{rank}(x) + \text{rank}'(z) - 2\text{rank}'(x) \leq -2$ を証明します。その

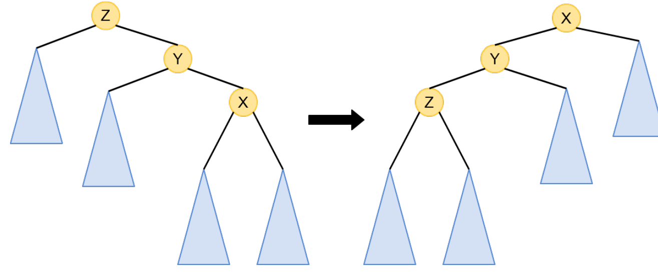

図 II.4　Zigzig

前に一つ重要な不等式を導きます。

$$
\begin{aligned}
&a \geq 0, b \geq 0, a+b \leq 1 \text{ のとき} \\
&\quad \log(a) + \log(b) \\
&\leq \log(a) + \log(1-a) \\
&= \log(-(a-1/2)^2 + 1/4) \\
&\leq -2
\end{aligned}
\tag{II.5}
$$

これを使って $\mathrm{rank}(x) + \mathrm{rank}'(z) - 2\mathrm{rank}'(x)$ を評価します。

$$
\begin{aligned}
&\mathrm{rank}(x) + \mathrm{rank}'(z) - 2\mathrm{rank}'(x) \\
&= \log(\mathrm{size}(x)/\mathrm{size}'(x)) + \log(\mathrm{size}'(z)/\mathrm{size}'(x))(\because \text{定義}) \\
&\leq -2(\because \mathrm{size}(x) + \mathrm{size}'(z) \leq \mathrm{size}'(x), (\text{II.5}))
\end{aligned}
$$

よってこれと (II.4) を使って Zigzig の償却計算量 $3(\mathrm{rank}'(x) - \mathrm{rank}(x))$ が得られます。

同様に **Zigzag** の償却計算量も求めてみましょう。

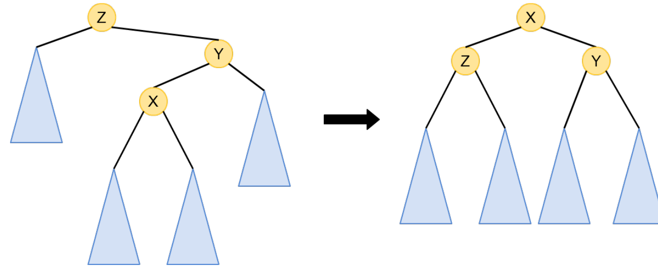

図 II.5　Zigzag

$$
\begin{aligned}
&2 + \mathrm{rank}'(x) + \mathrm{rank}'(y) + \mathrm{rank}'(z) - \mathrm{rank}(x) - \mathrm{rank}(y) - \mathrm{rank}(z) \\
\leq& 2 + \mathrm{rank}'(y) + \mathrm{rank}'(z) - 2\mathrm{rank}(x) \qquad (\because \mathrm{rank}'(x) = \mathrm{rank}(z), \mathrm{rank}(x) \leq \mathrm{rank}(y)) \\
\leq& 3(\mathrm{rank}'(x) - \mathrm{rank}(x))
\end{aligned}
$$

これも最後の式変形に Zigzig と同様に $\mathrm{rank}'(y) + \mathrm{rank}'(z) - 2\mathrm{rank}'(x) \leq -2$ を利用します。こちらの証明は先程と同様なので割愛させていただきます。

これで、Zig、ZigZig、Zigzag の償却計算量がわかりました。ある頂点 x がスプレーされて頂点に行く時の償却計算量は、t をスプレー完了後の x とすると

$$\begin{aligned}
&3(\mathrm{rank}'(x) - \mathrm{rank}(x)) + 3(\mathrm{rank}''(x) - \mathrm{rank}'(x)) + \cdots \\
\leq &3(\mathrm{rank}(t) - \mathrm{rank}(x)) + 1 \\
= &O(\log(N))
\end{aligned}$$

となります。最後に 1 を足しているのはたかだか 1 回しかない Zig の分です。スプレー木でなぜ、深さを 2 つごとに変化させるかというと、Zig で足される定数を 1 回に済ませるためです。

これでスプレー木におけるスプレーの償却計算量が $O(\log(N)))$ ということがわかりました。やったね。

5　まとめ

この章ではポテンシャルという概念を使った償却計算量の解析 (**Potential method**) を紹介しました。これ以外にも貯金という概念をつかった方法 (**Accounting method**) という物があります。前者は状態にポテンシャルを持たせることによって計算量を均しますが、後者は操作によって貯金や借金をして計算量を均します。どちらにせよポテンシャル関数や貯金や借金をあらかじめうまく設定してあげないといけません。これは思いつくものではなく、どちらかというと目指したい償却計算量から逆算して設定するものです。その目指したい償却計算量というのは、特殊な場合の最悪計算量や、平均計算量を使ったりします。

ここで紹介しなかった **UnionFind** 木の償却計算量解析にもポテンシャル解析が使われます。こちらはどこからともなくアッカーマン関数があらわれて華麗に計算量を上から押さえつけてくれます。読者の皆さんも文献などを漁って、証明をやってみてください。とはいえ、やるのは一回だけで充分！

参考文献

[1] Thomas H.Cormen, Charles E.Leiserson, Roland L.Rivest, and Clifford Stein. *Amortized Analysis*, pp. 451–480. The MIT Press, 2009.

[2] Thomas H.Cormen, Charles E.Leiserson, Roland L.Rivest, and Clifford Stein. *Fibonacci Heaps*, pp. 505–530. The MIT Press, 2009.