Splay tree の解析

(STOC '83)で Sleator, Tarjanらが発表

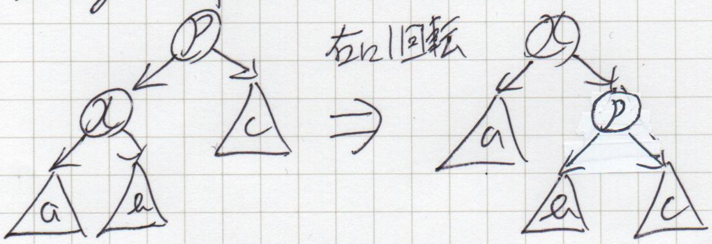
Splay tree は平衡二分探索木(BST)の一種で, look up, insert, delete, merge, split などの操作をならし $O(\log n)$ で行うデータ構造

(特長)
- 各ノードkeyと値のみ保持すれば十分なのでメモリの意味で優れている。
- "self-adjusting"… よくアクセスされるノードは根に近い, つまりアクセスしやすい位置になるように変形されるため, 偏ったデータ(クエリ)などに対しては特に有効
- ならしなので最悪の場合 $O(n)$ time かかりうる。
- "self-adjusting"の恩恵で情報理論的な意味で全体の計算量が漸近的に最適

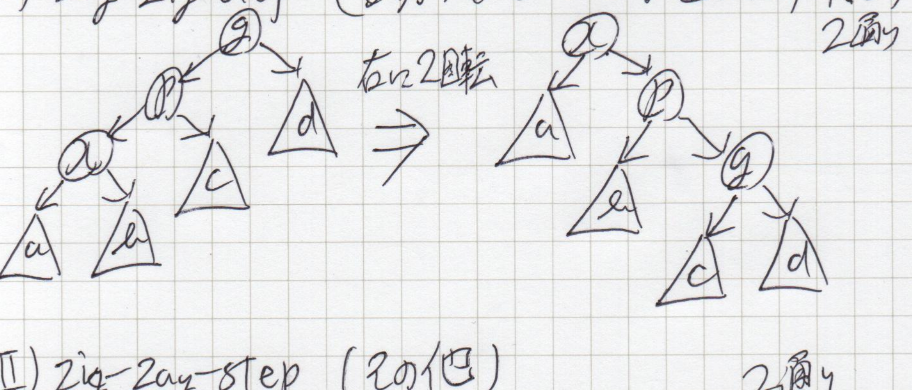
メインの操作 Splay(x)…xを根にする。

Splay(x)では x, xの親, xの親の親の位置関係に応じて次の3つの操作を繰り返してxを根にする。

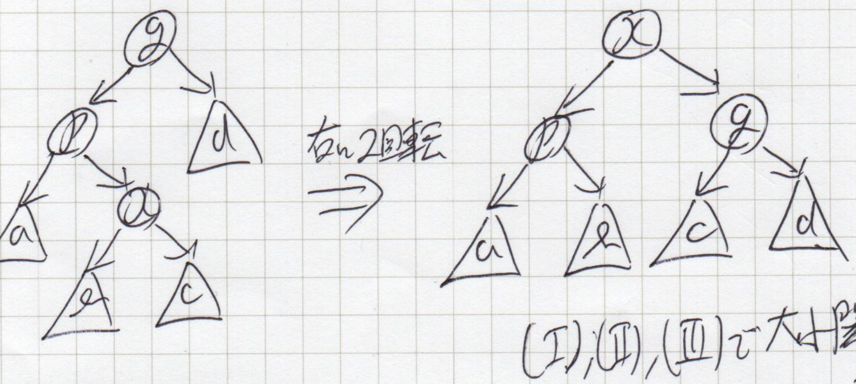
(I) zig-step (xの親の親がいない場合) 2通り



(II) zig-zig-step ((x,p)と(p,g)の位置関係が同じ) 2通り

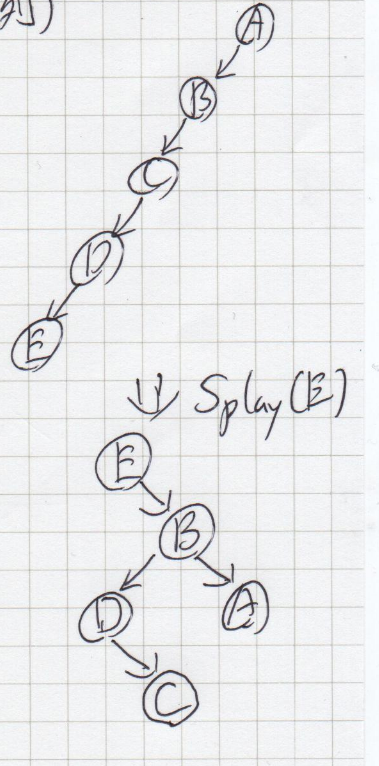


(III) zig-zag-step (その他) 2通り



(I),(II),(III)で大小関係がちゃんと保たれていることに注意!

(例)



Splay tree の解析

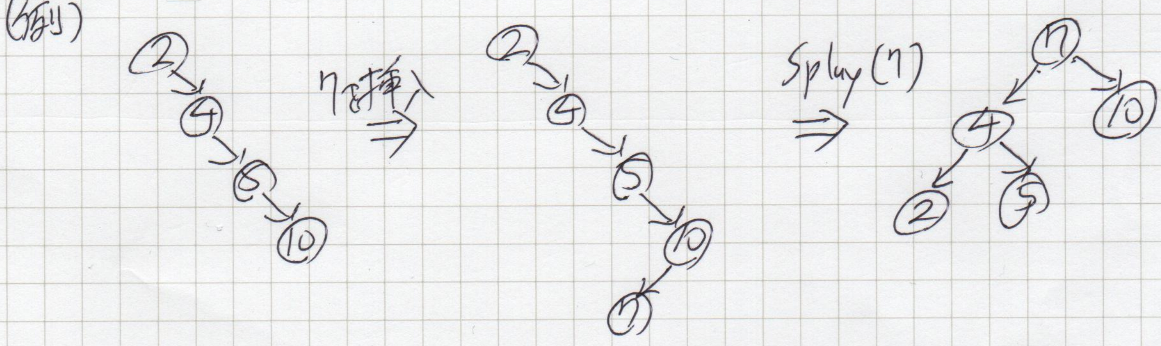
Splay(x)を行うことで頂点xだけでなくその周りの部分木も深さが小さくなっているのがポイント。(部分木の深さが1減るとその部分木内全ての頂点の深さが1減ることになる。)

(操作パート)

・Lookup(x)…上から木上を探索し、最終的に着いた頂点をyとすると、Splay(y)を行う。
(存在するなら x=y)

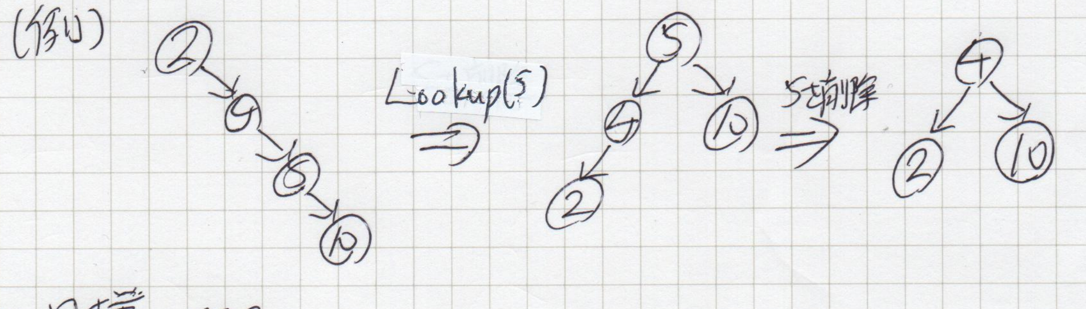
・Insert(x)…上から木上を探索し、適切な場所に挿入し、Splay(x)を行う。

(例)



・Delete(x)…Lookup(x)を行い、根である頂点xを削除し、残りの2つの部分木をマージする。

(例)



merge, splitも同様にできる。

(解析パート) 結局のところどの操作もO(1)回のSplay操作を行っていることと等しいので Splay操作のみ考える。(木上の探索はSplay操作1回の計算量に明らかにバウンドされる。)

(Tips)
実はSplay treeの解析はかなり難しく、未だに示されていないconjectureもいくつか存在する。

ならし計算量ということで、ポテンシャルを定めたい。

w_iは後のOptimalityの証明において各値の出現確率に対応するもので特にこの証明には必要ないが導入しておく。

各ノードx_iの根からの距離をl_i、各ノードに対応する重みをw_iとするとき、$\sum_{i=1}^{n} w_i l_i$に注目する。(nはノードの数)

正確には後述のrank(x)の和をポテンシャルとする。

$S(x) = \sum_{x_i \in (x\text{を根とする部分木})} w_i$ と定めると、$\sum_{x} S(x) = \sum_{i=1}^{n} w_i l_i + \sum_{i=1}^{n} w_i$ となる。

Splay treeの解析

また xのrank, r(x)を $r(x) \overset{def}{=} \log_2 S(x)$ と定め, ポテンシャルΦを $\Phi = \sum_{i=1}^{n} r(x_i)$ と定める。

ここで zig, zig-zig, zig-zag の各操作を繰り返して行われる Splay(x) の操作にかかるならしコストは $3(r(t)-r(x))+1$ 以下であることを示す。(tは木の根とする。)

実際にかかる計算量 + ポテンシャルの変分

r'を操作後のrankとしたとき

(特に zig は $3(r'(x)-r(x))+1$ 以下, zig-zig, zig-zag は $3(r'(x)-r(x))$ 以下を示す。)

(注) ならし解析(ポテンシャルを用いた場合)は m回の操作について

$$\underbrace{\sum_{i=1}^{m} a(op_i)}_{\text{ならしコストの和}} = \underbrace{\sum_{i=1}^{m} t(op_i)}_{\text{実際にかかる総計算量}} + \underbrace{O(1)\cdot(\Phi_m - \Phi_0)}_{\text{最初と最後でのポテンシャルの差}}$$

(Proof)

(i) zig の場合

以下の議論は2通りのzigの両方に対応している。

$r'(x)=r(p)$, $r'(p) \le r'(x)$ かつ "x, p以外のrankは変化しない" より

$$(\text{ならしコスト}) = \underbrace{1}_{\text{1回転する計算量}} + \Delta\Phi = 1 + r'(x) + r'(p) - r(x) - r(p)$$
$$= 1 + r'(p) - r(x)$$
$$\le 1 + r'(x) - r(x)$$
$$\le 3(r'(x) - r(x)) + 1$$

(ii) zig-zig の場合 以下の議論は2通りのzig-zigの両方に対応している。

$$(\text{ならしコスト}) = 2 + \Delta\Phi = 2 + r'(x) + r'(p) + r'(g) - r(g) - r(p) - r(x)$$
$$= 2 + r'(p) + r'(g) - r(p) - r(x) \underset{?}{\le} 3(r'(x) - r(x))$$

結局 $\underbrace{r'(p) + r'(g) - r(p) - r(x) - 3r'(x) + 3r(x)}_{①} \underset{?}{\le} -2$ を示したい。

$$① = r'(p) + r'(g) - r(p) - 3r'(x) + 2r(x)$$
$$\le r'(x) + r'(g) - r(p) - 3r'(x) + 2r(x)$$
$$= r'(g) - r(p) - 2r'(x) + 2r(x)$$
$$\le r'(g) - r(x) - 2r'(x) + 2r(x)$$
$$= r(x) - r'(x) + r'(g) - r'(x)$$
$$= \log \frac{S(x)}{S'(x)} + \log \frac{S'(g)}{S'(x)}$$

ここで $x>0, y>0, x+y \le 1$ なら $\log_2 x + \log_2 y \le -2$ より, 示せた。

$\because \log_2 x + \log_2 y = \log_2 xy$
$\le 2\log_2 \frac{x+y}{2}$
$= -2$

Splay treeの解析

(iii) zig-zagの場合　zig-zigと同様にできる。

(i)~(iii)より xが根になるまで zig, zig-zig, zig-zagを繰り返したときにかかる総ならしコストは高々 $3(r(t)-r(x))+1$ となる。

(zigは最後に高々1回行われる。) ∎

最後に全体でのポテンシャルの減少分をboundする。 $W=\sum_{i=1}^{n} w(x_i)$ とすると、

$$\log w_i \le r(x_i) \le \log W.$$

よってノードx_iの減少分は高々 $\log W - \log w_i$.

故に全体で $n\log W - \sum_{i=1}^{n} \log w_i$ となる。

Theorem. n頂点のスプレー木にm回の操作を行ったときの計算量は $O(m\log n + n\log n)$

Proof. $w_i = \frac{1}{n}$ $(i=1,\dots,n)$ と定める。(このとき $r(x) \ge \log\frac{1}{n} = -\log n$ に注意)

各操作は前述のように $O(1)$回のSplay操作にかかる計算量によってboundされるので、

結局 $O(m)$回のSplay操作にかかる計算量を考えればよい。

すると
$$\sum_{i=1}^{m} t(op_i) = \sum_{i=1}^{m} a(op_i) + O(1)\cdot(\Phi_0 - \Phi_m)$$
$$\le \sum_{i=1}^{m} \left(3(r(t)-r(x))+1\right) + \left(n\log W - \sum_{i=1}^{n} \log w_i\right)$$
$$= O(m\log n) + O(n\log n) \quad (\because r(t)=0,\ r(x) \ge -\log n)$$
∎

Static Optimalityの話

Fixされた二分探索木(BST)の構築について考える。

それぞれのKeyがLookupされる確率の分布が予め分かっているときに最適なBST
(期待計算量が最小な)

を求めたりというモチベーション(よくアクセスするノードは根の近くにするみたいな)

($O(n^2)$ timeのアルゴリズムが存在する(Knuth'71)。

$O(n^3)$のDPを巷でKnuth Optimizationと呼ばれているアレで$O(n^2)$にする。

Splay tree の解析

ここで期待計算量の下限とそれの有名な定理について説明する。

Theorem. Static Optimality

要素 x_i に確率 p_i でアクセスするようなクエリに対する最適の $\underbrace{\text{BST}}_{\text{staticな}}$ の $\underbrace{\text{期待計算量}}_{\text{Lookupにかかる}}$ は

$\Omega\left(\sum_{i=1}^{n} -p_i \log p_i + 1\right)$ である。(つまり、$\Omega(1+H)$)

$\sum_{i=1}^{n} -p_i \log p_i$ は Shannon entropy と呼ばれるもの。基本 H と表すことが多い。

(上記の定理は各要素が少なくとも一度はサンプルされること、全ての Lookup は成功することを仮定している。)

ここで Splay木は事前に出現確率を知らなくてもならしで $O(1+H)$ の計算量を達成する。

言い方を変えると、Splay木は 与えられた (x_i, p_i) に対して最適に動く staticな BST に対して定数倍より大きく遅くなることはない。(ならしの意味で)

← 赤黒木とかだと偏りある分布でも $\Omega(\log n)$ time かかりうるので劣る。

(Proof.) 先ほどのならしコストが $3(r(t) - r(x_i)) + 1 \leq -3\log p_i + 1$ となり、

$$\sum_{i=1}^{m} t(op_i) = \underbrace{\sum_{i=1}^{n} m p_i(-3\log p_i + 1)}_{\sum_{i=1}^{m} a(op_i)} + \underbrace{n \log W - \sum_{i=1}^{n} \log w_i}_{O(1)\cdot(\Phi_m - \Phi_0)}$$

$$\leq (m + 3mH) + mH$$

$$= O(m + mH)$$

つまりならしで $O(1+H)$ ∎

- Dynamic Optimality Conjecture の話

与えられた Lookup のクエリ列 $X = \{op_1, \cdots op_m\}$ に対して最適に動作する BST を $T(X)$ とする。

このとき、Splay木は $T(X)$ に対して定数倍より大きく遅くならないという仮説。

ちゃんというと、competitive ratio が $O(1)$ であるという仮説。

($O(1)$ の competitive ratio を達成する BST が存在するかどうかは分かっていない。)

$O(\log\log n)$ factor のものは知られている。

(tango tree や multisplay tree)
('04) ('06)