Vol. 34 No. 1　　情　報　処　理　　Jan. 1993

連載講座

キー検索技法—III

探索木法とその応用†

青江順一††　佐藤隆士†††

1. まえがき

探索木法はハッシュ法とともに非常に多くの分野で利用されており，多くの研究報告が存在する．前回までに説明したハッシュ法はキーから，キー（を含むレコード）を格納する番地を直接計算する方法であったが，探索木法はキー比較による大小関係から探索範囲を縮小しつつ，求めるキーを探す方法であり，検索コストはキーの総数の対数オーダとなる．また，探索木法は，キーの挿入・削除が頻繁な動的キー集合に向いており，キーの値順を反映した構造になっているため，キーの順検索や前後の検索，接頭辞を指定しての検索に対しては，ハッシュ法より優れている．なお，同じ探索木法で，キーを桁ごとに探索する「トライ法」は次回の講座で説明される．

1970年代を中心とした探索木法全般については，星ら[5]，Knuth[7]，弓場ら[10]にまとめられており，B木などの多分木については，Comer[3]，溝口[9]に，また2分探索木 (binary tree) 法については Nievergelt[8] に詳しく説明されている．近年の研究に関する文献の分類は，Gonnet[4] と Aoe[2] を参照されたい．したがって，本講座の文献は最近のものを主体的に取り上げてある．

本講座では，**2.** で探索木法の原理を概説し，**3.** で2分探索木法の基礎を例を用いて平易に説明し，そこで問題となる木の平衡（バランス）とそれを解決するための，種々の方法を紹介する．**4.** では多分木法として代表的なB木法を概説した後，B木の拡張（B^+木，B^*木，prefix B木など）の原理とそれらの特徴を説明する．また，**5.** では，探索木法の応用分野と特徴をまとめる．

2. 探索木法とは

2.1 探索木の構成

探索木法で用いる木構造[1),6),7)]は，節点と枝からなり，根 (root) と呼ばれる特別な節点を上方に書き，その他の節点を枝で接続したデータ構造である．枝で結ばれた2節点間には親子関係があり，上方が親である．下方に書かれた子のない節点を葉と呼ぶ．木には閉路がないので2点間を結ぶ経路は1通りに決まり，根からある節点までの路長（通る枝の本数）をその節点の深さと呼び，同じ深さにある節点は同一レベルにあると呼ぶ．ある節点から葉までの路長をその節点の高さと呼び，木の高さは根の高さと等しい．節点の子供間に左から右に順を付けた木を順序木と言う．

2.2 検索手法

探索木法では，節点に格納された一つまたは二つ以上のキーと，探すべきキーとの比較を繰り返して検索を行い，節点内に二つ以上のキーが格納されているときは，通常昇順に並べられる．キー比較で等しければ，キーは見つかったことになり，節点内のどのキーとも等しくない場合は，キーの値を範囲として含む枝を辿って下にある節点についてさらに比較を行う．この操作を続けて，探すべき枝がなくなれば，探索は失敗しキーは未登録となる．

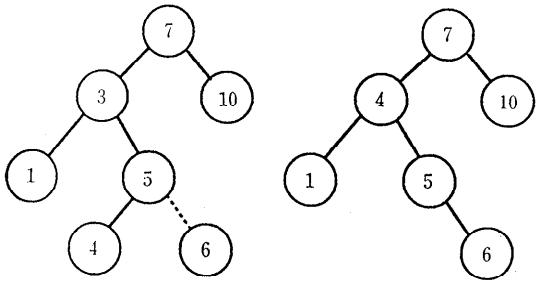
3. 2分探索木

3.1 2分探索木による検索

2分探索木は，**図-1** の（a）に示すように各節点が高々1組の左右の子をもつ順序木であり，各節点は1個のキーの値を格納する．たとえば，キー5の探索では根の値7に対する大小関係 5<

† Key Search Strategies—Tree Searching and Its Applications—
by Junichi AOE (Dept. of Information Science and Intelligent Systems, Faculty of Engineering, The University of Tokushima) and Takashi SATO (Dept. of Arts and Science, Osaka Kyouiku University).

†† 徳島大学工学部知能情報工学科
††† 大阪教育大学教養学科



（a） 6を挿入　　（b） 4を削除

図-1 2分探索木の例

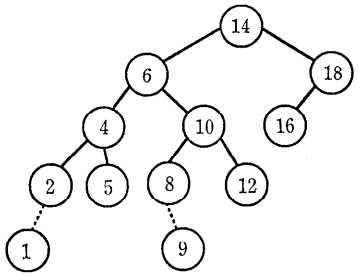
7より値3である左の枝へ，次に 5>3 より値5である右の枝へと辿り，キー5を発見する．n 個のキーが木にうまく平衡（各節点の部分木の高さが等しい状態）して格納されているならば，1回の比較で探索領域は 1/2 に狭められ，検索時間計算量[1]は $O(\log_2 n)$ となる．このような木を完全平衡木と呼ぶ．

キーの挿入は，探索が失敗した場所に新しい葉を作ればよい．たとえば，図-1の（a）に対してキー6は，キー5の節の右の枝の葉として挿入される．削除は葉であるか一つの子しかもたない節点の場合は簡単であるが，二つの子をもつ場合は，削除する節点の右部分木の最小要素（左部分木の最大要素でも同じ）で置き換える．たとえば，図-1の（a）から3を削除する場合は右部分木の最小要素4で置き換えられ，図-1の(b)が得られる[6]．

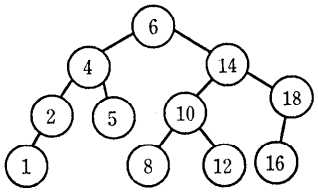
3.2 平衡木 (balanced tree)

2分探索木での最大比較回数は，根から最も遠い葉まで辿るときの節点数（木の高さ＋1）になるので，検索効率の向上のためには，平衡木を作る必要がある．ただし，ここでの平衡木は各節点の部分木の高さの差が一定値以下である場合を含む．

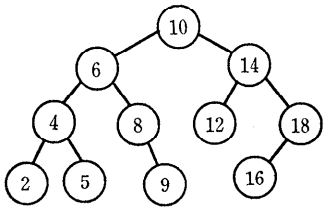
キー集合が静的でかつ検索頻度が一様な場合は，完全平衡木[18]，経路平衡木[28]（内部路長最小の木）を作ることが可能である．また，動的キー集合に対して空の木に挿入・削除を繰り返し，特別の工夫なしにできる木をランダム木と呼ぶ．ランダム木の平均的な木の高さは $1.386\log_2 n$ になることが知られているが，最悪の場合は $n-1$ になるので，よい検索効率を保証するためには次のような工夫が必要である[9),10)]．



（a） 1または9を挿入



（b） 1を挿入後の再バランス



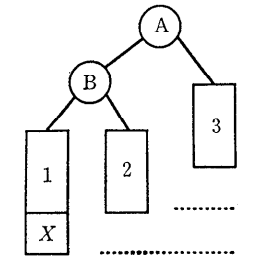
（c） 9を挿入の場合

図-2 AVL 木の例

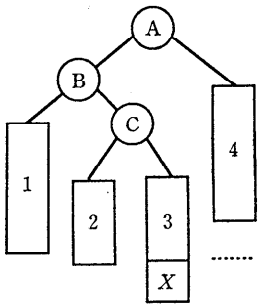
AVL 木[12]（提案者の頭文字より命名）は，左右の部分木の高さの差を1以下（+1, 0, −1）に維持する木である．たとえば，**図-2** の（a）の木に 1 を挿入すると，14 から左の部分木の高さが4，右の部分木の高さは 2 のままなのでその差が2になる．そこで6と14の部分木の接続を変え，**図-2** の（b）のように再バランスさせる．図-2の（a）の木に，9を挿入する場合も，アンバランスを生じる．この場合は，バランスをとるために**図-2** の（c）のように 14，6，10 の部分木の接続を変える必要がある．

一般的に，キー X が挿入されてアンバランスになる場合は**図-3** の3通り考えられるが，(b)と(c)は対称な場合であるので，(b)でまとめて説明する．この再バランス法である**図-4** の（a）の解消を1重回転，**図-4** の（b）の解消法を2重回転という．

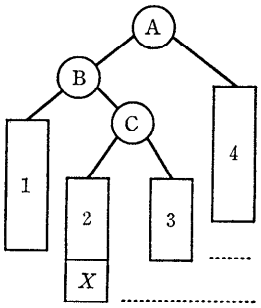
アンバランスの解消は，2または3節点についての部分木の接続を変えるだけなので，$O(1)$ の



(a) その1



(b) その2

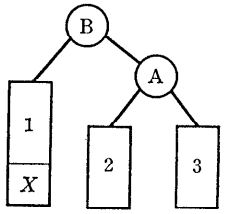


(c) その3

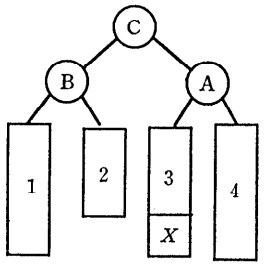
図-3 挿入による3種類のアンバランス

計算量で実行できる．削除によるアンバランスも挿入と同様に解消できるが，回転後もさらに親方向に遡って解消処理をしなければならないことがあるので，コストは $O(\log_2 n)$ となる．AVL 木の最大比較回数は，完全2分木に比べ最大 1.44 倍であり，挿入の際回転が必要となる確率は約47%，削除の際の回転回数は約 21% となる実験結果が得られている[6]．

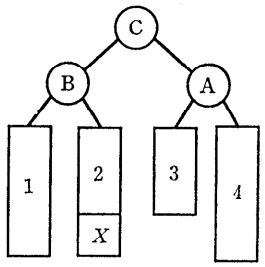
そのほか，AVL木の拡張で左右の部分木の高さ差を1から一定値以下に拡張した高さ平衡木[10),20)]や節平衡木 (bounded balance tree)[8),10),23)] では，左右の部分木に含まれる節点数の偏りをパラメータにとり，検索コストと挿入削除コストのバランス調整をすることができる．節点からの枝の数 m



(a) その1の解消



(b) その2の解消

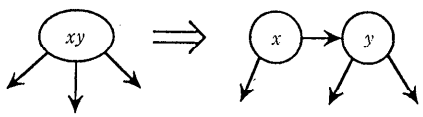


(c) その3の解消

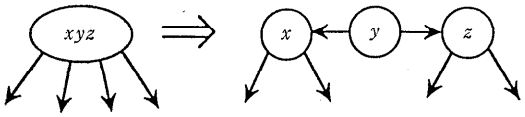
図-4 再バランス

が2のB木（B木については **4.** 参照）である 2-3木[1),10)]は，一つの節点に1ないし2個のキーをもっており，各節点から2分岐ないし3分岐する．2-3 木における二つのキーをもつ節点を二つの節点に置き換えた2分B木[14)]，論理節点に3個までのキーを入れる $m=3$ のB木で，3個のキーをもつ節点を2分木構造で実現した SBB 木 (symmetric binaryB-trees)[15)] の節点をそれぞれ**図-5** の（a）と（b）に示す．2分B木と SBB 木の最大路長は $2\log_2 n$ となり，AVL 木の $1.44\log_2 n$ に比べ大きくなるが，キーの挿入削除時の再バランスの頻度は低いので，挿入削除の多い目的に使用するとよい．

赤黒木 (red-black tree)[21)] は，SBB 木のように論理節点に3個までキーを入れる 2-3-4 木である．論理節点は，キー1個をもつ2分木構造で実現され，論理節点内でのリンクを赤，論理節点間のリンクを黒の枝と呼ぶ．SBB 木とは挿入削除による



（a） 2個のキーをもつ場合



（b） 3個のキーをもつ場合

図-5 論理節点の実現法

再バランスの方法が異なり，SBB 木が $O(\log_2 n)$ かかるのに対し，赤黒木は $O(1)$ で高速である[27]．SBB(k) 木[11]は SBB 木の拡張で仮想節点の大きさを $m=2^{k+1}-1$ としたものであり，木の最大路長は $\lceil(1+1/k)\log_2(n+1)\rceil$ に改善される．半平衡木 (half balanced tree)[25] は，各節点から葉までの経長の最大値と最小値の比が2以内の木で，$O(1)$ のコストで再バランスが可能である．そのほかの $O(1)$ コストの再バランス法は，Tarjan[29]，Levcopoulos ら[22]，Ottmann ら[26]に述べられている．また，平衡木の解析については，Yao[30]，Brown[17]，Mehlhorn[24]，Baeza[13] の研究を参照されたい．Yao[30] の解析は 2-3 木のものであるが，その拡張であるB木の解析でもある．

検索確率が一様でない場合，頻繁に検索されるキーは根に近い節点に置くと検索コストを下げることができる．このような木は重み平衡木[11]と呼ばれ，検索頻度により構成を動的に変化させる方法が Bent ら[16]，Sleator ら[27]により議論されている．

Anderson ら[11]は，各節点での比較 (<, =, >) で必要な二つの if 文を一つにして高速化する方法を提案している．また，通常の2分探索木では一つのキーに2個のポインタを必要とするが，Cunto ら[19]は2分木の1節点に複数個のキーを格納し，ポインタの記憶領域を節約した構造を提案している．

4. 多分木法

4.1 B木法による検索

2分探索木で2次記憶上に格納されたキーとレコードを検索する場合，木の高さ＋1が最大探索回数になるので，2次記憶から主記憶へのデータ転送回数が多くなる．これに対して，主記憶と2次記憶間のデータ転送のブロックに複数個のキーを格納し，転送回数を軽減する構造が多分木であり，2分木と比べ木の高さは低くなる．ただし，検索性能を保証するためには，2分木と同様に平衡木にする必要がある．

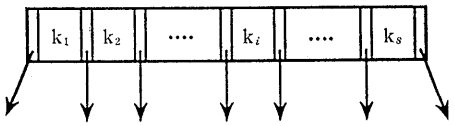
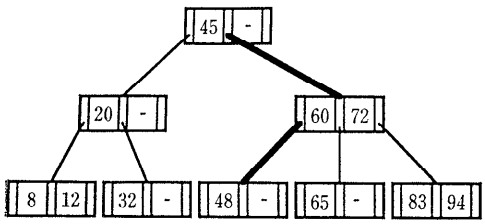


図-6 B木の節点

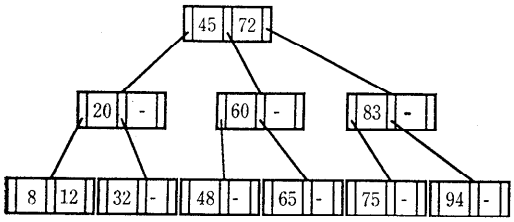
多分木の代表的なものが，Bayer ら[32]によるB木であり，多くの解説が存在する[3),5),9),10]．分岐数 m のB木の各節点は，最大 m 個，最小 $\lceil m/2\rceil$ 個（節点が根のときは1個）のキーとキー数＋1のポインタからなる．挿入と削除に対し，節点の同一レベル内での分割，併合操作により，平衡木の性質を維持している．ある節点に $s(\lceil m/2\rceil \leqq s \leqq m)$ 個のキー $k_1, k_2, \cdots k_s$ が順に入っているとすると，$k_1<k_2<\cdots<k_s$ である（**図-6**）．この節点でキー x を検索する場合，k_1 から順に比較し，$x=k_i\ (1\leqq i\leqq s)$ が見つかれば成功し，そうでなければ，$x<k_i$ となる最初のキーの左にあるポインタを辿り，1段下の節点について検索を行う．ただし，$x>k_s$ のときは右端のポインタを辿る．葉レベルまで降りてもキー x が見つからなかった場合，x は未登録となる．

図-7 は節点に二つのキーをもつ簡単な B 木の例であるが，図-7 の（a）には 48 を検索する経路を太線で示してあるので，確認されたい．同図（a）に 75 を挿入する場合，根から検索で 75 が入るべき場所を見つけると，83 と 94 が入っている節点に辿り着くが，満杯で入らない．そこで 75, 83, 94 の中央値 83 を上のレベルに移動し，残りのキーを二つの節点に分割する．ところが，上のレベルの節点も満杯で入らない．ここで，さらにこの節点を分割し，60, 72, 83 のうちの中央値 72 をさらに上の節（根）に挿入し，図-7 の（b）が得られる．この根も満杯ならば，もう1レベル上に新しい根を作ることになる．

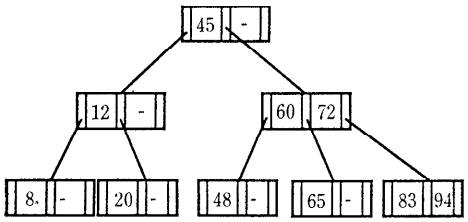
削除方法を説明するために，まず**図-7** の（a）から 8 を削除する場合を考える．この場合，8 は葉にあり，しかも 8 を削除後も葉内に $\lceil m/2\rceil$ 以上のキーが残るので，単に 8 を削除するだけでよ



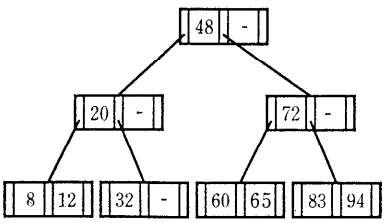
（a） 48 を探索



（b） （a）に 75 を挿入



（c） （a）から 32 を削除



（d） （a）から 45 を削除

図-7　B木の例

い．ただし，8 ではなく 32 を削除する場合は，削除後葉内に「$m/2$」以上のキーは残らないので，隣の葉にある 8，12 と上のレベルにある 20 を分配し，葉のキーの数が「$m/2$」以上になるように変更する（**図-7** の（c））．次に，より複雑な例として，図-7 の（a）から 45 を削除する場合を考える．45 は根のただ一つのキーであるので，45 の次に大きいキーを根に移動する．このキーは，45 の右隣のポインタを辿り，以下のレベルの節点では一番左のポインタを辿ることで辿り着く葉の中で一番左に位置する値（すなわち 48）を見つけ，48 を根のキーとして移動する．ここで，48 が入っていた節点に「$m/2$」個以上のキーが残っていれば終了するが，この場合はキーは残らないので，隣の節点のキーを分配して，「$m/2$」以上にすることを試みる．しかし，この例では隣の節点には 65 の 1 個しかないので分配も不可能となり，二つの節点を併合する．併合の結果，共通の親の節点にある 60 は下のレベルに降ろされる（**図-7** の（d））．ただし，併合のため親の節点数が「$m/2$」未満になれば，さらに同様な分配あるいは併合を繰り返し，併合が根まで遡ればB木の高さが 1 だけ減少する．

根以外の節点の分岐数は「$m/2$」$+1$ $(=d)$ であるので，検索のために節点を訪問する回数は，キー数の m を底とする対数オーダである．根以外の節点の記憶利用率は，50 から 100% の間である．ランダムに挿入を繰り返してできる木の記憶利用率が 69% となることは第 2 回の動的ハッシュでも説明したので，参照されたい．

記憶利用率のより分かりやすい導出と一般化は Leung[41]，Gupta ら[36]にあり，Arnow ら[31]はB木を含むシミュレーションによる比較をしている．また Langenhop ら[40]は B 木を家系木（lineage tree）でモデル化し，マルコフ過程を用いた解析，Driscoll ら[35]は B 木の根の近傍の解析を行っている．

4.2 B^+ 木による検索

B^+ 木[3),7)]はキーの直接アクセスだけでなく，順アクセスも高速に行えるようにB木を拡張したものであって，データベースのインデックスなどに広く使用されている（**図-8**）．

B^+ 木では，全てのキー（とその関連情報）を葉にのみ置いて，葉より上位レベルはランダム探索のための探索索引部と考える．そして，葉同士は順検索を可能にするためにリンクされる．したがって，検索時には探索路を葉まで辿り，その葉でキーの存在を確認すればよい．ただし，検索途

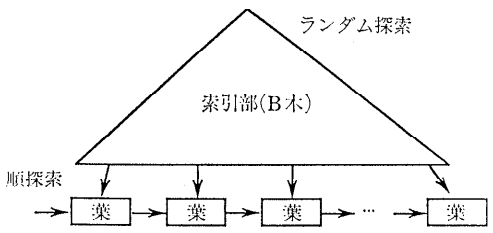


図-8　B^+ 木の構成

中の節でキーが等しくなっても，最終確認は葉まで辿ってから行う必要がある．

挿入操作はB木と同様であるが，満杯の葉の分割では上のレベルに中央キーを移動するのでなく，そのコピーを挿入することで行われる．また，削除されるレコードは常に葉にあるので，B木に比べ削除操作は簡単である．すなわち，索引部にそのレコードのキーが置かれていても，以降も正しく分離値として働くので，削除せずそのまま残しておいてもよい．削除により葉の使用率が半分未満になれば，B木と同様の分配または併合の処理を行う．

B木では，あるキーの次の値のキーを探すのに $\log_m n$ 回のアクセスを要するかもしれないが，B^+ 木では次のキー探索には高々1回の2次記憶アクセスしか要しない．このように B^+ 木は，B木のランダム検索効率を維持したうえで効率的な順検索を可能にした技法である．

4.3 多分木の種々な変形

（1） B* 木

満杯になった節点にキー挿入を行う場合，左（または右）の節点の格納スペースに余裕があるときは，その節点との間で分配し，節点の分割を避けることができる．余裕がないときは，これら2節点を3節点に分割する．このような修正を施したB木または B^+ 木を Knuth[7] は B* 木と名付けている． B* 木では記憶の利用効率が増すだけでなく，節点からの平均分岐数が増加するので，木の高さが低くなり検索が速くなる．Kusper[39] は分配を行う節点数を2以上の場合に一般化し，記憶利用率を解析している．

（2） 接頭辞 B^+ 木 (prefix B^+ tree)

B^+ 木の索引部に格納されたキーは，分離値としての役割を果たしているだけなので，実際に存在するキーそのものを格納する必要はない[8),33)]．キーが文字列である場合，分離の役割をするのに必要なだけその接頭部分を利用すればよい．英語の月名に対する簡単な接頭辞 B^+ 木の例を**図-9**に示す．ただし，キーは固定長であってもこのようにして作られた接頭辞は可変長であるため，可変長のデータ管理が必要となる．

（3） そのほかの変形

可変長キーへの対応や葉を大きくして補間で探索するなどの変形については，Comer[3], Knuth[7], 溝口[9]に議論されている．また，Held ら[37]は準静的なB木について，静的なディレクトリ構造の利点を述べている．静的な場合，キー順にデータを挿入することによりコンパクトなB木を作ることができる[34]．Sherk[46] と Martel[43] は，2分木と同様重み付きの多分木について議論している．そのほか，Hsio ら[38], Ouksel ら[44], Lomet[42] は，ハッシュ法と組み合わせたハイブリッド法を提案している．

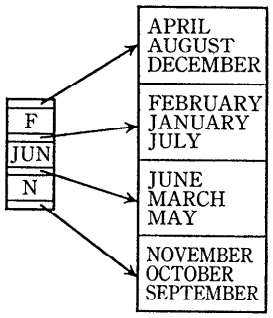


図-9 接頭辞B木の例（左が根，右が葉）

5. 応用分野とまとめ

今回の講座で扱った探索木法は，キーの検索更新の最悪コストをキーの総数の対数オーダで処理することが可能であるので，キーの性質の予測できない分野では非常に有効な技法となる．特に，B^+ 木は次のキー値が $O(1)$ で取り出せる優れた特徴を有している．前回までの講座のハッシュ法で述べたように，ハッシュ法でも順検索のための手法はいくつか提案されているが，キー分布の仮定やキー更新の頻度を考慮する必要がある．したがって， B^+ 木の順検索は最も自然で効率的な技法であるといえる．

探索木法は検索を目的とする応用分野では，ほとんど何の制約もなく安定して動作する手法である．主記憶上での検索では2分探索木が，2次記憶上に格納されたデータの検索には多分木が用いられる．特にデータベースの検索手段としては，実用システムのほとんどに B^+ 木あるいはその変形が用いられている．また，日高ら[47]はB木を自然言語辞書検索へ応用している．

謝辞 本稿に対して種々の有益な指摘をいただいた査読者に感謝いたします．

参 考 文 献

【探索木法の入門解説書】

1) Aho, A. V., Hopcroft, J. E. and Ullman, J. D.: Data Structures and Algorithms, Addison-Wesley, Reading Mass., Chs. 4, 5 and 11 (1983). 大野義夫訳, 培風館 (1987).
2) Aoe, J.: Computer Algorithms—Key Search Strategies—, IEEE Comput. Society Press (1991).
3) Comer, D.: The Ubiquitous B-Tree, Comput. Surveys, Vol. 11, No. 2, pp. 121-137 (1979). 上田訳: bit 別冊 (11月), pp. 21-39 (1980).
4) Gonnet, G. H.: Handbook of Algorithms and Data Structures, Addison-Wesley, Reading Mass. Ch. 3 (Searching Algorithms) pp. 25-147 (1984). 玄 光男 (共訳) 啓学出版 (1987).
5) 星, 弓場: 2分探索木, B木, k-d 木による見出し探索, 情報処理, Vol. 24, No. 4, pp. 396-400 (Apr. 1983).
6) 石畑 清: データ構造とアルゴリズム, 岩波書店, pp. 73-123 (1989).
7) Knuth, D. E.: The Art of Computer Programming, Addison-Wesley, Reading Mass., Vol. Ⅲ, Sorting and Searching, pp. 422-480 (1973).
8) Nievergelt, J.: Binary Search Trees and File Organization, ACM Comput. Surveys, Vol. 6, No. 3, pp. 195-207 (1974).
9) 溝口徹夫: "B-tree" によるデータ管理, 情報処理, Vol. 21, No. 7, pp. 769-776 (July 1980).
10) 弓場, 星: 木構造を用いた見出し探索の技法, 情報処理, Vol. 21, No. 1, pp. 28-49 (Jan. 1980).

【平衡木】

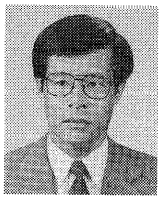
11) Andersson, A., Icking, C., Klein, R. and Ottmann, T.: Binary Search Trees of Almost Optimal Height, Acta Inf., Vol. 28, No. 2, pp. 165-178 (1990).
12) Adel'son-Vel'skii, G. M. and Landins, E. M.: An Algorithm for the Organization of Information, Doklady Akademia Nauk USSR, Vol. 146, pp. 263-266 (1962). English translation in Sov. Math, Dokl., Vol. 3, pp. 1259-1262.
13) Baeza-Yates, R., Gonnet, G. H. and Ziviani, N.: Expected Behaviour Analysis of AVL Trees, Proc. of Workshop on Algorithm Theory, pp. 143-159 (1990).
14) Bayer, R., MaCreight, E.: Binary B-trees for Virtual Memory, Proc. of ACM SIGFIDET Work shop, pp. 219-235 (1971).
15) Bayer, R. and MaCreight, E.: Symmetric Binary B-trees: Data Structure and Maintenance Algorithms, Acta Inf., Vol. 1, No. 4, pp. 290-306 (1972).
16) Bent, S. W., Sleator, D. D. and Tarjan, R. E.: Biased Search Trees, SIAM J. Comput., Vol. 14, No. 3, pp. 545-568 (1985).
17) Brown, M. R.: A Partial Analysis of Random Height-Balanced Trees, SIAM J. Comput. Vol. 8, No. 1, pp. 33-41 (1979).
18) Chang, H. and Iyengar, S. S.: Efficient Algorithms to Globally Balance a Binary Search Tree, Comm. ACM, Vol. 27, No. 7, pp. 695-702 (1984).
19) Cunto, W. and Gascon, J. L.: Improving Time and Space Efficiency in Generalized Binary Search Trees, Acta Inf., Vol. 24, No. 5, pp. 583-594 (1987).
20) Foster, C. C.: A Generalization of AVL Trees, Comm. ACM, Vol. 16, No. 8, pp. 513-517 (1973).
21) Guibas, L. J. and Sedgewick, R. A.: A Dichromatic Framework for Balanced Trees, Proc. of 19 th FOCS, pp. 8-21 (1978).
22) Levcopoulos, C. and Overmars, M. H.: A Balanced Search Tree with $O(1)$ Worst-Case Update Time, Acta Inf. Vol. 26, No. 3, pp. 269-277 (1988).
23) Nievergelt, J., Reingold, E. M.: Binary Search Trees of Bounded Balance, SIAM J. Comput., Vol. 2, No. 1, pp. 33-43 (1973).
24) Mehlhorn, K.: A Partial Analysis of Height-Balanced Trees Under Random Insertions and Deletions, SIAM J. Comput, Vol. 11, No. 4, pp. 748-760 (1982).
25) Olivie, H. J.: A New Class of Balanced Search Trees: Half-Balanced Search Trees, RAIRO Informatique Theorique, Vol. 16, pp. 51-71 (1982).
26) Ottmann, T. and Wood, D.: How to Update a Balanced Binary Tree with a Constant Number of Rotations, Proc. of Workshop on Algorithm Theory, pp. 122-131 (1990).
27) Sleator, D. D. and Tarjan, R. E.: Self-Adjusting Binary Search Trees, J. ACM, Vol. 32, pp. 652-686 (1985).
28) Stout, Q. F. and Warren, B. L.: Tree Rebalancing in Optimal Time and Space, Comm. ACM, Vol. 29, No. 9, pp. 902-908 (1986).
29) Tarjan, R. E.: Updating a Balanced Search Tree in O(1) Rotations, Inf. Proc. Lett., Vol. 16, No. 5, pp. 253-257 (1983).
30) Yao, A.: On Random 2-3 Trees, Acta Inf., Vol. 9, pp. 159-170 (1970).

【多分木, B木】

31) Arnow, D. M. and Tenenbaum, A. M.: An Empirical Comparison of B-Trees, Compact B-Trees and Multiway Trees, Proc. of ACM SIGMOD Conf., pp. 33-46 (1984).
32) Bayer, R. and McCreight, E.: Organization and Maintenance of Large Order Indexes, Acta Inf. Vol. 1, pp. 173-189 (1972).
33) Bayer, R. and Unterrauer, K.: Prefix B-Trees, ACM Trans. Database Syst., Vol. 1, No. 1, pp. 11-26 (1977).
34) Cesarini, F. and Soda, G.: An Algorithm to Construct a Compact B-Tree in Case of Ordered Keys, Inf. Proc. Lett., Vol. 17, No. 1, pp. 13-16 (1983).
35) Driscoll, J. R., Lang, S. D. and Franklin, L. A.: Modeling B-Tree Insertion Activity, Inf. Proc.

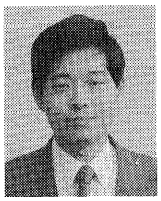
Lett., Vol. 26, No. 1, pp. 5-18 (1987).
36) Gupta, G. K. and Srivivasan, B.: Approximate Storage Utilization of B-Trees, Inf. Proc. Lett., Vol. 22, No. 5, pp. 243-246 (1986).
37) Held, G. and Stonebraker, M.: B-Tree Re-Examined, Comm. ACM, Vol. 21, No. 2, pp. 139-143 (1978).
38) Hsiao, Y. and Tharp, A. L.: Adaptive Hashing, Inf. Syst., Vol. 13, No. 1, pp. 111-127 (1988).
39) Kuspert, K.: Storage Utilization in B*-Trees with a Generalized Overflow Technique, Acta Inf., Vol. 19, pp. 35-55 (1983).
40) Langenhop, A. E. and Wright, W. E.: A Model of Dynamic Behavior of B-Trees, Acta Inf., Vol. 27, pp. 41-59 (1989).
41) Leung, C. H. C.: Approximate Storage Utilization of B-Trees: A Simple Derivation and Generalizations, Inf. Proc. Lett., Vol. 19, No. 4, pp. 199-201 (1984).
42) Lomet, D. B.: A Simple Bounded Disorder File Organization with Good Performance, ACM Trans. Database Syst., Vol. 13, No. 4, pp. 525-551 (1988).
43) Martel, C.: Self-Adjusting Multi-Way Search Trees, Inf. Proc. Lett., Vol. 38, No. 3, pp. 135-141 (1991).
44) Ouksel, M. and Scheuermann, P.: Implicit Data Structures for Linear Hashing Schemes, Inf. Proc. Lett., Vol. 29, No. 4, pp. 183-189 (1988).
45) Quitzow, K. and Klopproge, M.: Space Utilization and Access Path Length in B-Trees, ACT Trans. Database Syst., Vol. 6, pp. 174-183 (1981).
46) Sherk, M.: Self-Adjusting k-Ary Search Trees, Proc. of Algorithms and Data Structures Workshop, pp. 381-392 (1989).
47) 日高，稲永，吉田：拡張 B-Tree と日本語単語辞書への応用，信学論 (D), Vol. J 67-D, No. 4, pp. 399-404 (1984).

(平成4年8月5日受付)



青江 順一 (正会員)

昭和49年徳島大学工学部電子工学科卒業．昭和51年同大学院修士課程修了．同年同大学工学部助手(情報工学科)，現在知能情報工学科助教授．工学博士．この間，コンパイラ自動生成系，自然言語処理と理解，知識工学に関する研究に従事し，パーサ，情報検索，ストリングパターンマッチング，データ圧縮に関するアルゴリズムの効率化に興味をもつ．著書「Computer Algorithms-Key Search Strategies」IEEE CS Press など．電子情報通信学会，日本人工知能学会，日本ソフトウェア科学会，日本機械翻訳協会，IEEE, ACM, AAA, ACL 各会員．



佐藤 隆士 (正会員)

昭和53年岡山大学大学院修士課程工学研究科電子工学専攻修了．同年，詫間電波高専助手．平成2年，大阪教育大学教養学科助教授，現在に至る．データベース，記憶階層のアルゴリズムの研究に従事．工学博士．電子情報通信学会，CAI 学会，IEEE Computer Society, ACM 各会員．