解　説

"B-tree" によるデータ管理†

溝　口　徹　夫††

1. まえがき

本文では木構造を持つデータ構成法の1つである，"B-tree" の特質を平易に解説してみたい．

"B-tree" は最初に Bayer, McCreight によって考案，発表されたデータ構造である[8]．すでに "B-tree" の考案から10年近く経ち，解説もいくつかなされている[1)~7)]．"B-tree"の名前の由来は，考案者の1人 Bayer の頭文字，考案した時の関係会社 Boeing の頭文字，または Balanced の頭文字等，諸説のうち第1説が有力だが，本人の釈明もないので，本文では単に B-tree と呼ぶことにする．

B-tree は一言で言えば，多重分岐平衡木 (multi-way balanced tree) である．平衡木とは木の根幹から葉までの高さがバランスしていることをここでは指し，多重とは木構造での分岐数が2つ以上を指す．多様なデータ利用形態に対して，できるだけ高い利用効率（時間および記憶容量上）を保つことができるのが理想のデータ構造であるとすれば，B-tree はこの性格を備えたものと言える．

最初に提案された B-tree と共通な特質を持つデータ構造，例えば 2-3 tree, B*-tree, B^+-tree, 等も提案されている．特に B^+-tree の性格を持つ VSAM が標準アクセス法として採用されたことが B-tree への関心を高めたのも事実である．また最近 B-tree 族の特質を理論的に解析する動きも盛んである．これは更に良質なデータ構造の考案を目指す活動の一部と言えよう．B-tree のページ分裂とハッシングの機能を組合せたもの[45)~47)]，TRIE 等と組合せたもの[48),49)]，等も提案されている．

本文ではまず，ファイルによるデータ利用にはどのような形態があるかを分類し，次いで B-tree の基本的機能を，分類されたデータ利用形態と関係づけて説明する．その後 B-tree の問題点や B-tree と類似したデータ構造の紹介および B-tree の性能等に関する解析結果をいくつか示す．

2. データの利用とファイル構造

前章で B-tree は多様なデータ利用形態に有効であると述べた．B-tree の特質を述べる前に，本章では "多様なデータ利用形態" とは具体的に何なのかについて，例を用いて示してみたい．

今仮りに××市で住民登録のデータを計算機ファイル化する場合を想定してみる．住民1人に1レコードを用意するとし，各レコードは氏名，住所，生年月日，性別，本籍地等からなっているとしよう．このファイルでは次のような取扱いが想定できる．

（1） 死亡や転出した住民のレコードを削除する．

（2） 誕生や転入した住民のレコードを追加する．

（3） 市内で転居した住民のレコードの住所を変更する．

（4） 時として全住民のレコードを五十音順にリストして配布する．

（5） 特定の住民レコード（例えば来年小学校入学予定者）のリストを作成する．

（6） 単独の住民レコードを出力する（例えば窓口での住民票作成依頼を受けた場合）．

以上のデータ利用をアクセス機能とアクセスパターンの二側面から眺めると，

① 格納されているデータの検索機能（例，五十音順リスト，住民票作成）．

② 新たなデータの追加機能（例，転入）．

③ 不要なデータの削除機能（例，転出）．

④ 不適当なデータの修正機能（例，転居）．

および，

（a） アクセスが単独でランダムに発生するもの（例，転入，住民票作成），

（b） アクセスがある決まった順序で一巡するように発生するもの（例，五十音順リスト），

（c） アクセスがある決まった順序で一部区間を走

† Data Management in "B-trees" by Tetsuo MIZOGUCHI (Mitsubishi Electric Corporation Computer Lab.).

†† 三菱電機(株)計算機研究部

査するもの（例，来年入学予定者）．
に分けて考えることができる．

このデータ利用の性格と従来のファイル構造との関係を概観してみよう．磁気テープを用いた順次ファイルは一巡検索には適するが，その他の利用にはこのままでは効率は低くなる．追加，削除，修正は一括して順次処理を行う等が必要である．磁気ディスクを用いたハッシングによるランダムファイルは“ある順序”での処理には不向きであり，ファイル全体の走査後のソートの必要がある．索引付順次ファイルは上記2ファイル方式の欠点はない．しかしこの索引付順次ファイルにも欠点はある．データレコードは索引によって分類，分割されているので，索引を利用すると高速で検索等が行える．しかしファイルに追加，削除が行われても索引の構造は動的に変化しない．そこでもしも追加，削除数が増大すると，性能上での低下を招くことになる．例えば，××市に大団地が出現したとすると，住所を索引とした住民ファイルでは，団地の住所近辺の住民レコードの検索は著しい性能低下を来たす．このような事態を防止するには必要に応じて（または周期的に）ファイルの再構成を行い，索引の構造全体を変える必要がある．しかし再構成は時間を要するし，いつ再構成すべきかを見守るのも大変である．もう1つの解決法は追加，削除時に索引の構造を動的に変化させることである．B-treeは後者の性格を持つデータ構造である．以下 B-tree の特質について述べるが，ここでいくつかの前提を示しておこう．B-treeでは索引の構造が問題となるので，データレコードのキーの扱いに主力が置かれ，データレコード本体をキーと一体として格納するか，キーのみを抜き出して索引として使うかいずれでも構わない．次章ではデータレコードはキーとは別個に格納され，索引としてのキーにはデータレコードへのポインタが陰に含まれているとする．またキーは固定長としておく．

3. B-tree の基本的機能

前章で B-tree は多重分岐平衡木であると述べた．多重とここで言うのは2分木 (binary tree) と対照させるためであるが，B-treeの各節には b 個の分岐ポインタ，$(b-1)$ 個のキーが格納されている．分岐数 b の上限を m とすると，b の最小値は $\lceil m/2 \rceil$ である．ただし，葉の節での分岐ポインタは空であり，根幹節 (root) の最小分岐数は2である．格納キー数は必ず分岐数−1 である．このような B-tree は m 次の B-tree と呼ばれる．**図-1** に5次の B-tree ($m=5$) の例を示す．図-1 において枠内の数字はキー値を表わし，三角形は次下位レベルへのポインタを表わす．この例から B-tree の特質を列挙してみると，

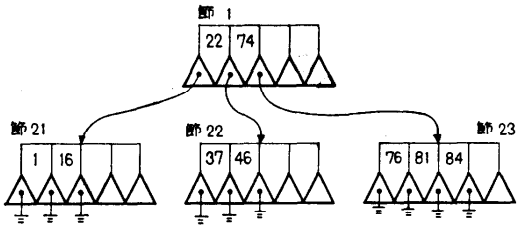

図-1 B-tree の構造例 ($m=5$)

（1） 各節はすべて同じ構造である（最大 $(m-1)$ 個のキーと最大 m 個のポインタを格納できる）．

（2） 木構造の葉の節は根幹の節から見て同一レベルにある．

（3） 葉の節のポインタは空である（ここで空と未使用とは区別が必要）．

（4） 葉の節以外の節ではキー $(b-1)$ 個に対してポインタ b 個が存在し，そのキーの値は左側のポインタで指している節内のキーより大で，右側のポインタで指している節内のキーより小である．

（5） (4)の逆を言えば，葉の節以外の節でのポインタはその左右にあるキーの値の中間のキーを持つ部分 B-tree をポイントしている．

根幹の節を除けばすべての節はキーは半分($\lceil m/2 \rceil - 1$)以上格納されている．以上のことからわかるように，目的のキーへ到着するのに必要な最悪アクセス時間は最小化される．勿論このような平衡状態はキーの追加，削除によって乱されることになる．上述したような B-tree の性格を保持するにはどうすればよいかを含めて以下では検索，追加，削除について順次示してみよう．

3.1 B-tree でのキーの検索

ここで検索とはキーを一個与えて，そのキーと関連するデータレコードを得ることであるが，それにはまずキーの B-tree 内での格納節を求める必要がある．

ステップ 1：根幹の節へのポインタを得る．

ステップ 2：ポインタが空ならば，B-tree 内にはそのキーは存在しない．検索不成功，停止．ポインタが空でないならば，ポイントされた節を読み込む．

ステップ 3：節内に検索キーが存在すれば，検索成功，停止．そうでない時，その検索キーが区間内に含まれる部分 B-tree へのポインタを得てステップ2へ．

B-tree 内に検索キーが存在すれば，ステップ3で見つかる．検索キーは根幹の節で見つかるかもしれないし，葉の節まで検索して見つかることもある．

検索キー発見後，そのキーに付随しているポインタを介してデータレコード本体への参照が行われる．検索キーが B-tree 内に存在しない場合は必ず葉の節に到着した後，ステップ2で検索不成功がわかる．そのような場合の検索の効率は葉の節へ至る木構造のレベル数に比例する．木構造のレベル数は勿論格納キー個数に依存する．レベル数，キーの個数等の関係については次章で触れよう．

3.2 B-tree へのキーの追加

B-tree の特徴はキーの追加，削除時の索引構造の動的再構成にあることはすでに述べた．本節ではキーの追加時の索引構造変化法について示す．

格納されているキーはその大小順に従って格納位置が定まっているため，キーを一個追加したいとすると，そのキーを挿入すべき B-tree の葉の節は一意に決まる．B-tree の節には最大 $(m-1)$ 個までのキーが格納可能であり，節内に $(m-1)$ 個未満のキーしか格納されていない場合は問題なく挿入ができるが，すでに $(m-1)$ 個のキーが格納されていた場合はキーの追加は物理的に不可能である．このような場合には B-tree の性格を守って次の方法でキーの追加を行う．

B-tree は多重分岐木であり，兄弟節数は自分を含めて m 個まで許せるので，自分の荷が多すぎたなら，親に救いを求めて，すぐ次の弟を用意してもらい，半分ずつ荷を負担すればよい．

このように満杯の節に更にキーの追加が要求されたとき，兄弟節を隣りに新設し，格納キーを2節で2等分することを分裂 (split) と呼ぶ．分裂に関与したこの2個の節でのキーを以後区別するために，親の節には分裂したキーの真中の値（中央値）を格納する必要が出てくる．分裂時の親の節へのキーの追加は更に親の節での分裂を招く可能性があり，場合によっては根幹の節まで分裂は波及し得る．根幹節の分裂の場合には1レベル上に根幹節が新設される．

キー追加の手順をまとめてみると，

ステップ 1：与えられた追加キーの B-tree 内検索，(キーを追加すべき葉の節の発見)．

ステップ 2：その節内の格納キーの個数が $(m-1)$ 未満の場合はキーを節へソートされた順に追加，処理終了．

節内の格納キーの個数が $(m-1)$ の場合は追加キーも含めて節内キーの分裂．

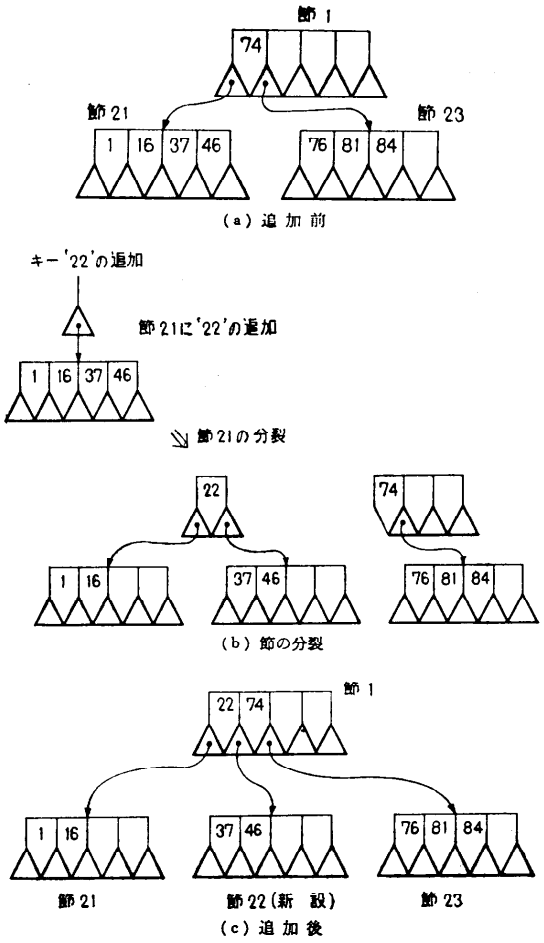

図-2 B-tree へのキー追加，分裂

ステップ 3：親の節に追加するキーを得る．分裂した節が根幹であった場合はステップ4へ，さもなければ，親の節へのキー追加を行うためにステップ2へ．

ステップ 4：根幹を新たに一段上に設け，ステップ2へ．

根幹の分裂直後の新根幹では格納キー1個，2分岐である．また追加されるキーは B-tree には以前存在しないキーであるので，分裂は必ず葉の節で最初に発生することがわかる．

図-2 に分裂の例を示す．満杯の節 21 へのキーの追加 ('22' の追加) 要求が発生したとしよう．節 21 は分裂を生じ，分裂キーの中央値 '22' を中心として2個の節にキーは分割される．親の節1に挿入されるキー '22' には余分のスペースが存在したので，分裂の再発はない．

時刻	追加入力キーの値	追加後のファイル内でのキー順序 (数値の昇順＊印は新規入力)	キーの節（ページ）への物理的分割 (最大4キー／ページ)
1	76	76*	[76]
2	74	74*, 76	[74, 76]
3	84	74, 76, 84*	[74, 76, 84]
4	37	37*, 74, 76, 84	[37, 74, 76, 84]
5	16	16*, 37, 74, 76, 84	74 / [16, 37] — [76, 84]
6	1	1*, 16, 37, 74, 76, 84	74 / [1, 16, 37] — [76, 84]
7	46	1, 16, 37, 46*, 74, 76, 84	74 / [1, 16, 37, 46] — [76, 84]
8	81	1, 16, 37, 46, 74, 76, 81*, 84	74 / [1, 16, 37, 46] — [76, 81, 84]
9	22	1, 16, 22*, 37, 46, 74, 76, 81, 84	22 / 74 / [1, 16] — [37, 46] — [76, 81, 84]
10	48	1, 16, 22, 37, 46, 48*, 74, 76, 81, 84	22 / 74 / [1, 16] — [37, 46, 48] — [76, 81, 84]
11	80	1, 16, 22, 37, 46, 48, 74, 76, 80*, 81, 84	22 / 74 / [1, 16] — [37, 46, 48] — [76, 80, 81, 84]
12	77	1, 16, 22, 37, 46, 48, 74, 76, 77*, 80, 81, 84	22 / 74 / 80 / [1, 16] — [37, 46, 48] — [76, 77] — [81, 84]

図-3 データ入力と B-tree の構造 ($m=5$, 入力数 12)

ここでキーを順次追加した時の B-tree の構造の変化を図-3 に別の形で表わしてみる．今仮りに 12 個のキー (76, 74, 84, 37, 16, 1, 46, 81, 22, 48, 80, 77) が順次入力されたとする．図の第1欄は時刻，第2欄は追加キー，第3欄は各時刻でのキーの順序，第4欄は各時刻での B-tree ($m=5$) 内のキーの配列を示し，キーの間に引かれた線の上下の区切りは木構造のレベル，左右の区切りは節の境界を表わす．時刻5の追加時に分裂によって2レベルになり，時刻 9, 12 で更に分裂が繰返されている．図-1 は時刻9後を，図-2 は時刻9の分裂経過を示す．

根幹の節を除けば分裂時に節内のキーは 1/2 満たされていて，以後の追加では必ず 1/2 以上満たされている．

3.3 追加時のオーバフローについて

B-tree での記憶利用効率(節内の格納キーの割合)は最悪時で 50% である．このことは B-tree の考案者たちも気にして，分裂を減らす方法が検討された[8]．分裂を発生する条件が成立した時，左右に隣接する節に格納スペースに余裕があれば分裂を行わずに，あぶれたキーの流出(オーバフロー)を行う方法がそれである．

図-2 での分裂の例でオーバフロー技法を用いれば実は分裂は不要となる（図-4 参照）．時刻9直後の記憶利用効率はオーバフローにより 19% 改善する．

オーバフローを隣接の節以遠に波及させることも可能である．

3.4 B-tree からのキーの削除

本節では続いてキーの削除について述べよう．

キーの削除はキーの追加の逆になるが，追加と削除では次の2点が異なる．追加では未格納のキーを追加するので，キーの属すべき葉の節をまず見つけること，および追加による節内の容量超過を防ぐことが必要であった．削除では削除されるべきキーはどこかの節に既格納であり，削除に伴う節内の容量低下を防ぐことが必要である．

削除すべきキーの含まれている節の発見は検索手順で行う．そしてそのキーを節から削除する．もしも節

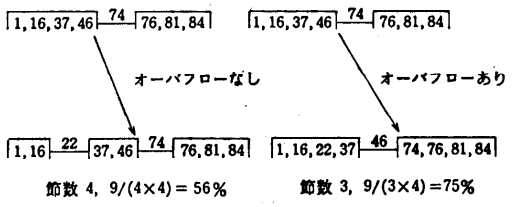

図-4 キー '22' 追加時のオーバフローの効果

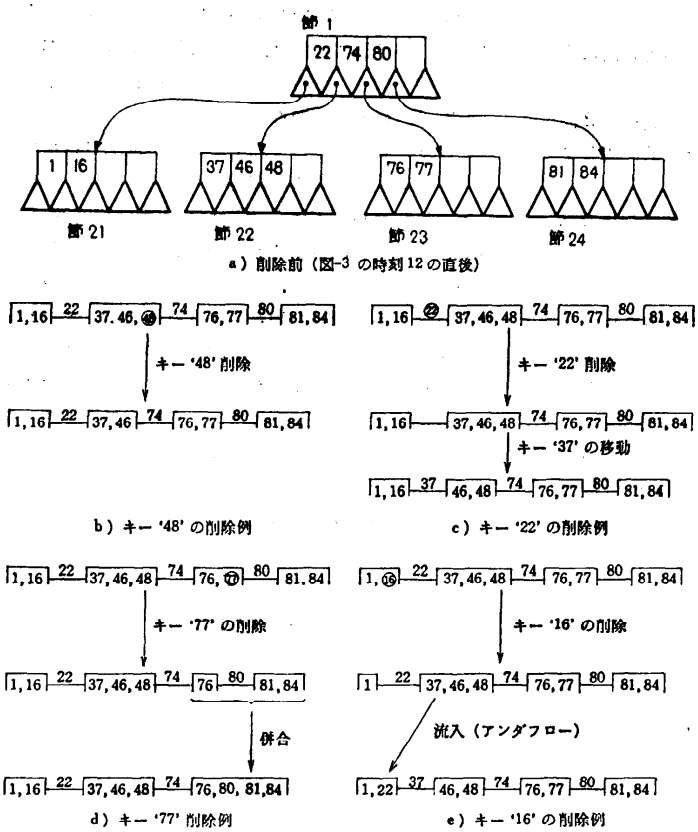

図-5 B-tree からのキーの削除

が葉である場合はそのような問題はないが，葉でない節のキーを削除しただけでは木の途中に空白ができ以後の B-tree 利用に支障を来たす．この抜きとられたキーの埋め合せに，次に大きなキーである右部分 B-tree の左端の葉のキーを移動する．次に大きなキーは必ず葉に存在することは理解できよう．

図-5 に削除の例を示そう．**図-5**(b) では削除キー '48' は葉の節に存在するので，検索後削除される．**図-5**(c) では削除キー '22' は葉に存在しなかったので '22' の削除後，次に大きなキー '37' が葉から移動して埋め合される．この例では削除によって節内のキー個数は 1/2 未満にならなかったが，(d),(e) の例ではキーの削除で節内の格納キーは 1/2 未満となる．(d) では隣接節と親のキーで 1 個の節に併合が出来るが，(e)では併合は不可能でありキーの流入（アンダフロー）が行われる．(d)の例は分裂の逆であり，(e)の例は流出（オーバフロー）の逆である．

3.5 B-tree のその他の特質

B-tree への修正は削除と追加を組合せて行えるし，順次検索については根幹から開始して，左部分木，親，右部分木の順序 (inorder) で検索を行えばよい．ただし木構造であるため，根幹から現在検索中の節へ至る全節をバッファ内に確保してあるかまたはそのアドレスを持って置く必要[25),26)]がある．本章では B-tree でのキーの管理のみを述べ，データレコード本体の格納法については触れなかった．単独のランダムアクセスにおいては特に問題とはならないデータレコード格納法は順次アクセスでは見過せない点である．順次アクセスの効率向上には連続したキーの値を持つデータレコードは互いに近接して格納されている必要がある．一方節内の格納キー数が大である程，B-tree のレベル数は少なく，アクセス効率は向上する．(したがって索引のキーはデータレコードと別の節内に格納する方がよい．)本章では節からの最大分岐数 m は与えられるものとして来たが，m を適切に選ぶことによって効率の向上が望める．m はキーの長さにもよるが，m の上限はバッファサイズ等の物理的条件，節内のキーの検索時間により，下限は木構造のレベル数増に伴う入出力回数によって制約をうける[2)]．

アクセス効率等に関しては次章で再び触れる．

4. B-tree の改善と解析

3 章では B-tree の基本的特質について述べた．次いで本章では B-tree と同族である B*-tree, B+-tree, (VSAM) について概説し，更に B-tree に対する 3 つの問題点を示し，その後この最重要問題点と言われた効率の解析成果について概説してみよう．

4.1 B-tree の改善

(1) B*-tree[3)]

キーの追加時でのオーバフロー技法の利用は記憶利用効率向上に役立つことはすでに 3.3 節で述べた．Knuth によってオーバフローを使った B-tree は B*-tree (B star-tree) と名づけられ，以下の性格を持つ

ものとして整理された.

① 満杯の節にキーの追加がある時，隣接の節に余裕があればオーバフローする.

② ①において隣接の節も満杯ならば，満杯の2節のキーと追加キーを新設の節を加えた3節に等分する.

この方法では各節内のキー数は追加のみの状態では2/3を下回ることはない.

(2) B⁺-tree (M-tree[20])

B⁺-tree ではすべてのキーは葉の節に格納され，上位の節内の索引となるキーは各葉の節の（最大）キーを重複して使う．各節は B-tree 構造を採り，追加で分裂を行う．削除では上位節の重複キーは削除しなくても支障はない．葉の節同志は順次方向にポインタで接続され，順次アクセス用に横方向の動きが可能である．次に B⁺-tree の構造を持つ VSAM (Virtual Storage Access Method) について触れる.

(3) VSAM[31)~33)]

VSAM は IBM 社による巨大なソフトウェア製品であるが，そのデータ構造に関する部分のみをまとめてみる．ファイル構成にはキー順，エントリ順，レコード番号順があるが，B⁺-tree と対応するのはキー順である.

1個のファイルは何個かの CA (Control Area) からなり，1個の CA は何個かの CI (Control Interval) からなり，1個の CI に何個かのデータレコードが格納される．ファイルには索引が付いていて，1 CA 当り 1 Sequence set が用意され，各 CI 内の最大キーが Sequence set 内に格納される．索引は Index set を含めて B⁺-tree 構造を採る.

アクセス法はランダム，順次の両法を許し，CIへのデータ追加で CI サイズ超過の場合は同一 CA 内の他の空き CI を用いて分裂を行う．CA 内に空き CIが拭底した時は CA の分裂を行う．CI 分裂，CA 分裂の2段階分裂法によって，順次アクセスでの次アクセス

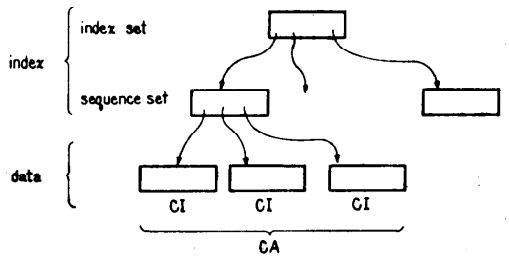

図-6 VSAM のデータ構造

CI が同一シリンダに格納されている可能性を高くしている．削除時には併合，流入は行わず索引内部ではキー圧縮（前方，後方の両圧縮）が行われている.

4.2 B-tree の問題点

B-tree の問題点として以下の3つが掲示された[42)].

(1) 2次キーの管理問題

データレコード本体を B⁺-tree (VSAM) の例のように索引の構造に合せて分裂を行わせることが順次アクセス上有効であるが，分裂(レコードの物理的移動)に伴う2次キー管理の複雑さをさけるには，インバーティドファイルには主キーのアドレスではなく，主キー自体を用いざるを得ない．2次キーからのアクセスには必ず主キーの検索を要し，効率が低下する．1つの代替法はデータレコードを索引構造から独立させる方法がある.

(2) 同時アクセス問題

同時に複数ユーザが同じ B-tree へのアクセスを行うとすると，分裂や併合のように索引構造の動的変化の下での安全なアクセスを保証しなければならない．Lock/Unlock 等の導入で複雑化するし，B*-treeのオーバフローはデッドロックの元になる.

(3) 索引のレベル数

索引付順次ファイルが B-tree より性能が悪くなるのは，検索と追加の比率等にもよるが，それほど一般的でない.

B-tree の問題点は性能効率上にあると結んでいる．その後(2)の同時アクセス問題についてはいくつかの成果が示されているので参照されたい[27)~30)].

以上3点以外に，キーの可変長化[22)]，等も解決しつつある．次に問題とされた効率について述べてみる.

4.3 B-tree の効率解析

B-tree の欠点は節内の不充分な活用から来るレベル数の増大と性能低下がその1つであるとされた．本節では効率上の問題をアクセス時間効率，記憶容量利用効率に分けてまとめてみる.

(1) 記憶アクセス効率

B-tree の記憶アクセス効率の評価には ① 節単位での主記憶と二次記憶間のデータの転送回数，② 特定のキーの検索，追加，削除に必要な比較回数の2つがある．前者の転送回数についてまず述べ，次いでキー比較回数に触れる.

キーの検索等での節単位のデータの転送回数を知るには，B-tree のレベル数とキーの個数の関係を求めればよい．今仮りに n 個のキーが格納されているとする

と，格納されていないキーを見つける位置は葉の節の空ポインタであり，$(n+1)$ 個所ある（図-1参照）．今レベル数を d とすると，

$$m^d \geqslant n+1 \geqslant 2(\lceil \frac{m}{2} \rceil)^{d-1}$$

が成立つ．最悪ケースでは根幹が2分岐，各節が $\lceil m/2 \rceil$ 分岐で葉に至り，最良ケースはすべて m 分岐による．すなわち $\lceil \log_m(n+1) \rceil \geqslant d \geqslant 1+\lceil \log_{\lceil m/2 \rceil}(n+1/2) \rceil$ である．

キーの検索では最大 d 回の節検索，キーの追加では最悪 d 回の分裂，キーの削除では最悪 $(d-1)$ 回の併合が生じる．

d の現実的な値を求めるには節内の平均格納キー数あるいは記憶利用効率が有効である．

（2） 節内のキーの検索法

B-tree 内のキーが二次記憶から主記憶内に転送された後での節内でのキーの検索法についての解析もなされている．

節内検索法として考えられるものには，2分検索法，補間法，平方根法，順次検索法等が考えられる．またこれらの組合せも可能である．一般に2分検索法がよいとされるが[24]，キーが可変長（たとえばキー圧縮を行った場合等）では利用できず，VSAM では平方根法を採用している[21]．

（3） 記憶利用効率

B-tree での記憶利用効率の低さは B-tree の欠点とされているが，キーの追加がランダムに繰返された時の記憶利用効率が解析され，ページ内のキー数 m が充分大である時，$\ln 2 = .69$ であること[41]，削除時に併合，横流しを用いない時のキーのランダム追加と削除後の記憶利用効率が解析され[37]，また併合，流入の効果のシミュレーション比較も行われている[35]．

横流し（オーバフロー，アンダフロー），併合を含んだ正確な解析は困難である．その理由は，横流し，併合は共に隣接の節内のキー数によって動きが異なり，分裂のように追加のあった節内のキー数のみによる単純な解析ができないためである．

5. むすび

以上限られた紙面での B-tree の解説を試みた．

B-tree が今後のデータ管理法で他をすべて凌駕するとは言えないが，その特徴は種々の形で具体化，活用されると考えられる．B-tree の提案の重要さはその有効性の議論もさることながら，データ管理の基となる基本構造になお改善の可能性と課題が存在することを示したことではないだろうか．

本文では今後の方向を示す余裕はなかった．筆者の力不足による誤解を生じなかったことを願う次第である．終りに資料提供いただいた各位に感謝したい．

参考文献

1) Comer, D.: The ubiquitous B-tree, ACM Computing Surveys, Vol. 11, No. 2, pp. 121-137 (Jun. 1979).
2) Horowitz, E. and Sahni, S.: Fundamentals of data structures, Pitman (1976).
3) Knuth, D. E.: The art of computer programming, Vol. 3, sorting and searching, Addison-Wesley, pp. 471-480 (1973).
4) 植村俊亮：データベースシステムの基礎，オーム社，pp. 210-215 (1979).
5) Wiederhold, G.: Data base design, McGraw-Hill (1977).
6) Wirth, N. 片山卓也訳： Algorithms + data structures=programs, Prentice Hall pp. 246-257 (1975).
7) 弓場敏嗣，星 守：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 (1980).
8) Bayer, R. and McCreight, E.: Organization and maintenance of large ordered indexes, Acta Informatica 1, pp. 173-189 (Jan. 1972).
9) Giordano, N. J. and Schwartz, M. S.: Data base recovery at CMIC, Proc. of ACM-SIGMOD Int'l Conf. on Management of Data (Jun. 1976).
10) Bayer, R. and Unterauer, K.: Prefix B-trees ACM Trans. on Database Systems Vol. 2, No. 1, pp. 11-26 (Mar. 1977).
11) Bayer, R. and Metzger, J. K.: On the encipherment of search trees and random access files, ACM Trans. on Database Systems, Vol. 1, No. 1, pp. 37-52 (Mar. 1976).
12) Culik, K. II, Ottmann, Th. and Wood, D.: Dense multiway trees, Computer Science Technical Report 79-CS-5, McMaster University (Ontario).
13) Guibas, L. and Sedgewick, R.: A dichromatic framework for balanced trees, Proc. of 19th IEEE Symp. on Foundation of Computer Science (1978).
14) Schneiderman, B. and Goodman, V.: Batched searching of sequential and tree structured files, ACM Trans. on Database Systems, Vol. 1, No. 3, pp. 268-275 (Mar. 1976).
15) Aho, A., Hopcroft, J. E. and Ullman, J. D.: The design and analysis of computer algorithms, Addison-Wesley (1976).
16) Bayer, R.: Symmetric binary B-trees: data

structure and maintenance algorithms, Acta Informatica, Vol. 1, No. 4, pp. 290-306 (1972).
17) Kriegel, H. P., Vaishnavi, V. K. and Wood, D.: 2-3 brother trees, BIT, 18 pp. 425-435 (1978).
18) Vaishnavi, V. K., Kriegel, H. P. and Wood, D.: Height balanced 2-3 trees, Computing, 21, pp. 195-211, Springer-Verlag (1979).
19) Kwong, Y. S. and Wood, D.: T-trees: a variant of B-trees, Computer Science Technical Report, 78-CS-18 McMaster University (Ontario).
20) Maruyama, K.: Index structures for virtual memory-comparison between B-trees and M-trees, IBM Technical Report RC 5258 (Feb. 1975).
21) Maruyama, K. and Smith, S.: Analysis of design alternatives for virtual memory indexes, CACM Vol. 20, No. 4, pp. 245-254 (Apr. 1977).
22) McCreight, E.: Pagination of B*-trees with variable-length records, CACM Vol. 20, No. 9, pp. 670-674 (Sep. 1977).
23) Rosenberg, A. and Snyder, L.: Compact B-trees, Proc. of ACM-SIGMOD Int'l Conf. on Management of Data (1979).
24) Strong, H., Markowsky, G. and Chandra, A.: Search within a page, JACM Vol. 26, No. 2, pp. 457-482 (Jun. 1979).
25) Brown, M. and Tarjan, R.: A representation of linear lists with movable fingers, Proc. of 10th ACM Symp. on Theory of Computing (1978).
26) Guibas, L., McCreight, E., Plass, M. and Roberts, J.: A new representation of linear lists, Proc. of 9th ACM Symp. on TOC (1977).
27) Bayer, R. and Schkolnick, M.: Concurrency of operations on B-trees, Acta Informatica, Vol. 9, No. 1, pp. 1-21 (1977).
28) Ellis, C. S.: Concurrent search and insertion in 2-3 trees, Tech. Report 78-05-01 Dept. of Computer Science, University of Washington.
29) Kwong, Y. S. and Wood, D.: Concurrency in B-trees, S-trees and T-trees, Computer Science Tech. Report 79-CS-17 McMaster Univ. (Ontario).
30) Samadi, B.: B-trees in a system with multiple users, Information Processing Letters Vol. 5, No. 4, pp. 107-112 (Oct. 1976).
31) Keehn, D. and Lacy, J.: VSAM data set design parameters, IBM Systems J., 3 pp. 186-202 (1974).
32) OS/VS virtual storage access method (VSAM) planning guide, GC 26-3799 IBM Corporation.
33) OS/VS virtual storage access method (VSAM) logic, SY 26-3841 IBM Corporation.
34) Frederickson, G. N.: Improving storage utilization in balanced trees, Proc. of 17th Allerton Conf., Univ. of Ill. (1979).
35) 牧野武則: B-tree 型インデックスにおけるページ内使用率, 情報処理学会論文誌, Vol. 20, No. 5, pp. 375-381 (1979).
36) Miller, R., Rosenberg, A. and Pippenger, N.: Optimal 2, 3 trees, SIAM J. on Computing, Vol. 8, No. 1, pp. 42-59 (Feb. 1979).
37) Mizoguchi, T.: On required space for random split files, Proc. of 17th Allerton Conf., University of Illinois (1979).
38) Reiter, A.: Some experiments in directory organization-a simulation study, Proc. of Int'l Symp. on Computer Performance Modelling, Measurement and Evaluation (1976).
39) Rosenberg, A. and Snyder, L.: Minimal-comparison 2, 3-trees, SIAM J. on Comput. Vol. 7, No. 4, pp. 465-480 (Nov. 1978).
40) Wong, C. K. and Yue, P. C.: Free-space utilization of a disk file organization method, Proc. of 7th Princeton Conf. on Information Sciences and Systems (1973).
41) Yao, A. C-C.: On random 3-2 trees, Acta Informatica Vol. 9, No. 2, pp. 159-170 (1978).
42) Held, G. and Stonebraker, M.: B-trees re-examined, CACM, Vol. 21, No. 2, pp. 139-143 (Feb. 1978).
43) Held, G. and Stonebraker, M.: B-trees re-examined-reply, CACM, Vol. 21, No. 7, p. 594 (July 1978).
44) Snyder, L.: B-trees, re-examined, CACM, Vol. 21, No. 7, p. 594 (July 1978).
45) Fagin, R., Nievergelt. J., Pippenger, N. and Strong, H. R.: Extendible hashing-a fast access method for dynamic files, ACM TODS, Vol. 4, No. 3, pp. 315-344 (Sep. 1979).
46) Larson, P.: Dynamic hashing, BIT, 18 pp. 184-201 (1978).
47) Litwin, W.: Virtual hashing: a dynamically changing hashing, Proc. of 4th VLDB, Berlin (1978).
48) Howard, P. H. and Borgendale, K. W.: System/38 machine indexing support, IBM System/38 Technical Developments, IBM GSD pp. 67-69.
49) 松本秀一, 野口正一: PATRICIA tree を用いた情報検索システムの構成, 信学会・研資 EC 77-22 (1978 3月).

(昭和55年1月14日受付)