(19)日本国特許庁(JP)　(12)特　許　公　報(B2)　(11)特許番号

特許第4498409号
(P4498409)

(45)発行日　平成22年7月7日(2010.7.7)　(24)登録日　平成22年4月23日(2010.4.23)

(51)Int.Cl.　F I
G06F 12/00 (2006.01)　G06F 12/00 520A
G06F 17/30 (2006.01)　G06F 17/30 240A

請求項の数 15　(全 36 頁)

(21)出願番号　特願2007-338413 (P2007-338413)
(22)出願日　平成19年12月28日 (2007.12.28)
(65)公開番号　特開2009-157853 (P2009-157853A)
(43)公開日　平成21年7月16日 (2009.7.16)
審査請求日　平成21年12月25日 (2009.12.25)

早期審査対象出願

(73)特許権者　506235616
株式会社エスグランツ
千葉県千葉市美浜区高洲三丁目５番３棟１２１０号
(74)代理人　100133570
弁理士　▲徳▼永　民雄
(72)発明者　新庄　敏男
千葉県千葉市美浜区高洲三丁目５番３棟１２１０号　株式会社エスグランツ内
(72)発明者　國分　光裕
千葉県千葉市美浜区高洲三丁目５番３棟１２１０号　株式会社エスグランツ内

審査官　工藤　嘉晃

最終頁に続く

(54)【発明の名称】データベースのインデックスキー更新方法及びプログラム

(57)【特許請求の範囲】
【請求項１】
コンピュータが実行するデータベースのインデックスキー更新方法であって、データベースのインデックスキーの旧データに対して該旧データと置き換えられる新しいインデックスキーの新データが供給されたときに、前記インデックスキーの更新を行うデータベースのインデックスキー更新方法において、
前記旧データあるいは新データのインデックスキーが、
ルートノードと、隣接した記憶領域に配置されるブランチノードとリーフノードまたはブランチノード同士またはリーフノード同士のノード対、からなるビット列検索に用いるツリーであって、前記ルートノードは、ツリーの始点を表すノードであって、該ツリーのノードが１つのときは前記リーフノード、ツリーのノードが２つ以上のときは前記ブランチノードであり、前記ブランチノードは、ビット列検索を行う検索キーの弁別ビット位置とリンク先のノード対の一方のノードである代表ノードの位置を示す位置情報を含み、前記リーフノードは検索対象のビット列からなるインデックスキーを含み、前記ツリーの任意のノードを検索開始ノードとして前記ブランチノードにおいて、該ブランチノードに含まれる弁別ビット位置の検索キーのビット値に応じてリンク先のノード対の代表ノードかあるいはそれと隣接した記憶領域に配置されたノードにリンクすることを順次前記リーフノードに至るまで繰り返すことにより、前記リーフノードに格納されたインデックスキーを、前記検索開始ノードをルートノードとする前記ツリーの任意の部分木の前記検索キーによる検索結果である検索結果キーとするように構成されたカップルドノードツリーのリ

ーフノードのインデックスキーとして格納されている場合は、該カップルドノードツリーを差分ツリーとし、登録されたルートノードの位置情報を取得し、前記旧データあるいは新データのインデックスキーが、カップルドノードツリーのリーフノードのインデックスキーとして格納されていない場合は、前記旧データあるいは新データのインデックスキーにより前記差分ツリーを生成する差分ツリー取得ステップと、

前記差分ツリーのルートノードを前記検索開始ノードとして前記新データあるいは旧データの全てのインデックスキーを検索キーとして検索を行い、検索キーと一致した前記検索結果キーであるインデックスキーは差分ツリーから削除し、前記検索結果キーであるインデックスキーと一致しない検索キーを差分データの挿入キーあるいは削除キーとし、前記新データあるいは旧データの全てのインデックスキーによる検索が終了した後に前記差分ツリーに削除されずに残ったインデックスキーを差分データの削除キーあるいは挿入キーとして差分データを作成する差分データ作成ステップと、

前記旧データのインデックスキーが、前記カップルドノードツリーのリーフノードのインデックスキーとして格納されている場合は、該カップルドノードツリーを更新ツリーとし、登録されたルートノードの位置情報を取得し、前記旧データのインデックスキーが、カップルドノードツリーのリーフノードのインデックスキーとして格納されていない場合は、前記旧データのインデックスキーにより前記更新ツリーを生成する更新ツリー取得ステップと、

前記差分データから取り出した削除キーを前記更新ツリーから削除し、前記差分データから取り出した挿入キーを前記更新ツリーに挿入することにより新データのインデックスキーをリーフノードのインデックスキーとして格納した新データカップルドノードツリーを作成し、該新データカップルドノードツリーに基づいて、前記旧データを前記新データに更新する新旧インデックスキー更新ステップと、

を備えることを特徴とするインデックスキー更新方法。

【請求項２】

コンピュータが実行する差分データ作成方法であって、データベースのインデックスキーの旧データに対して該旧データと置き換えられる新しいインデックスキーの新データが供給されたときに、前記インデックスキーの更新を行うデータベースのインデックスキー更新に用いる前記旧データと前記新データの差分データ作成方法において、

前記旧データあるいは新データのインデックスキーが、

ルートノードと、隣接した記憶領域に配置されるブランチノードとリーフノードまたはブランチノード同士またはリーフノード同士のノード対、からなるビット列検索に用いるツリーであって、前記ルートノードは、ツリーの始点を表すノードであって、該ツリーのノードが１つのときは前記リーフノード、ツリーのノードが２つ以上のときは前記ブランチノードであり、前記ブランチノードは、ビット列検索を行う検索キーの弁別ビット位置とリンク先のノード対の一方のノードである代表ノードの位置を示す位置情報を含み、前記リーフノードは検索対象のビット列からなるインデックスキーを含み、前記ツリーの任意のノードを検索開始ノードとして前記ブランチノードにおいて、該ブランチノードに含まれる弁別ビット位置の検索キーのビット値に応じてリンク先のノード対の代表ノードかあるいはそれと隣接した記憶領域に配置されたノードにリンクすることを順次前記リーフノードに至るまで繰り返すことにより、前記リーフノードに格納されたインデックスキーを、前記検索開始ノードをルートノードとする前記ツリーの任意の部分木の前記検索キーによる検索結果である検索結果キーとするように構成されたカップルドノードツリーのリーフノードのインデックスキーとして格納されている場合は、該カップルドノードツリーを差分ツリーとし、登録されたルートノードの位置情報を取得し、前記旧データあるいは新データのインデックスキーが、カップルドノードツリーのリーフノードのインデックスキーとして格納されていない場合は、前記旧データあるいは新データのインデックスキーにより前記差分ツリーを生成する差分ツリー取得ステップと、

前記差分ツリーのルートノードを前記検索開始ノードとして前記新データあるいは旧データの全てのインデックスキーを検索キーとして検索を行い、検索キーと一致した前記検

索結果キーであるインデックスキーは差分ツリーから削除し、前記検索結果キーであるインデックスキーと一致しない検索キーを差分データの挿入キーあるいは削除キーとし、前記新データあるいは旧データの全てのインデックスキーによる検索が終了した後に前記差分ツリーに削除されずに残ったインデックスキーを差分データの削除キーあるいは挿入キーとして差分データを作成する差分データ作成ステップと、

を備えることを特徴とする差分データ作成方法。

【請求項３】

請求項２記載の差分データ作成方法において、

前記カップルドノードツリーは、配列に記憶され、前記リンク先のノード対の代表ノードの位置を示す情報は、そのノードが格納された前記配列の配列要素の配列番号であることを特徴とする差分データ作成方法。

【請求項４】

請求項３記載の差分データ作成方法において、

前記差分ツリー取得ステップは、前記差分ツリーを生成する際に、前記旧データあるいは新データから最初に取り出したインデックスキーを含むリーフノードを該差分ツリーのルートノードとした後、それ以後前記旧データあるいは新データから取り出した前記インデックスキーを前記検索キーとして該差分ツリーから該当するリーフノードを検索するとともに、該リーフノードに至るまでたどったリンク経路のブランチノード及び該リーフノードが格納された配列要素の配列番号をスタックに順次格納し、前記検索キーと前記該当するリーフノードに含まれるインデックスキーの間で大小比較とビット列比較を行い、ビット列比較で異なるビット値となる先頭のビット位置と前記スタックに格納されているブランチノードの弁別ビット位置との相対的位置関係により挿入されるインデックスキーを含むリーフノードともう一方のノードからなるノード対の挿入位置を決定し、前記大小関係により挿入するインデックスキーを含むリーフノードを前記挿入されるノード対のどちらのノードとするかを決定するインデックスキーの挿入処理を繰り返すものであることを特徴とする差分データ作成方法。

【請求項５】

コンピュータが実行する差分データ更新方法であって、データベースのインデックスキーの旧データと該旧データと置き換えられる新しいインデックスキーの新データの差分データを用いて前記インデックスキーの更新を行うインデックスキーの差分データ更新方法において、

前記旧データのインデックスキーが、

ルートノードと、隣接した記憶領域に配置されるブランチノードとリーフノードまたはブランチノード同士またはリーフノード同士のノード対、からなるビット列検索に用いるツリーであって、前記ルートノードは、ツリーの始点を表すノードであって、該ツリーのノードが１つのときは前記リーフノード、ツリーのノードが２つ以上のときは前記ブランチノードであり、前記ブランチノードは、ビット列検索を行う検索キーの弁別ビット位置とリンク先のノード対の一方のノードである代表ノードの位置を示す位置情報を含み、前記リーフノードは検索対象のビット列からなるインデックスキーを含み、前記ツリーの任意のノードを検索開始ノードとして前記ブランチノードにおいて、該ブランチノードに含まれる弁別ビット位置の検索キーのビット値に応じてリンク先のノード対の代表ノードかあるいはそれと隣接した記憶領域に配置されたノードにリンクすることを順次前記リーフノードに至るまで繰り返すことにより、前記リーフノードに格納されたインデックスキーを、前記検索開始ノードをルートノードとする前記ツリーの任意の部分木の前記検索キーによる検索結果である検索結果キーとするように構成されたカップルドノードツリーのリーフノードのインデックスキーとして格納されている場合は、該カップルドノードツリーを更新ツリーとし、登録されたルートノードの位置情報を取得し、前記旧データのインデックスキーが、カップルドノードツリーのリーフノードのインデックスキーとして格納されていない場合は、前記旧データのインデックスキーにより前記更新ツリーを生成する更新ツリー取得ステップと、

前記旧データあるいは新データのインデックスキーを前記カップルドノードツリーのリーフノードのインデックスキーとして格納した差分ツリーのルートノードを前記検索開始ノードとして前記新データあるいは旧データの全てのインデックスキーを検索キーとして検索を行い、検索キーと一致した前記検索結果キーであるインデックスキーは差分ツリーから削除し、前記検索結果キーであるインデックスキーと一致しない検索キーを差分データの挿入キーあるいは削除キーとし、前記新データあるいは旧データの全てのインデックスキーによる検索が終了した後に前記差分ツリーに削除されずに残ったインデックスキーを前記差分データの削除キーあるいは挿入キーとして作成した差分データから取り出した削除キーを前記更新ツリーから削除し、前記差分データから取り出した挿入キーを前記更新ツリーに挿入することにより新データのインデックスキーをリーフノードのインデックスキーとして格納した新データカップルドノードツリーを作成し、該新データカップルドノードツリーに基づいて、前記旧データを前記新データに更新する新旧インデックスキー更新ステップと、

を備えることを特徴とするインデックスキーの差分データ更新方法。

【請求項６】

請求項５記載のインデックスキーの差分データ更新方法において、

前記カップルドノードツリーは、配列に記憶され、前記リンク先のノード対の代表ノードの位置を示す情報は、そのノードが格納された前記配列の配列要素の配列番号であることを特徴とするインデックスキーの差分データ更新方法。

【請求項７】

請求項６記載のインデックスキーの差分データ更新方法において、

前記更新ツリー取得ステップは、前記更新ツリーを生成する際に、前記旧データから最初に取り出したインデックスキーを含むリーフノードを該更新ツリーのルートノードとした後、それ以後前記旧データから取り出した前記インデックスキーを前記検索キーとして該更新ツリーから該当するリーフノードを検索するとともに、該リーフノードに至るまでたどったリンク経路のブランチノード及び該リーフノードが格納された配列要素の配列番号をスタックに順次格納し、前記検索キーと前記該当するリーフノードに含まれるインデックスキーの間で大小比較とビット列比較を行い、ビット列比較で異なるビット値となる先頭のビット位置と前記スタックに格納されているブランチノードの弁別ビット位置との相対的位置関係により挿入されるインデックスキーを含むリーフノードともう一方のノードからなるノード対の挿入位置を決定し、前記大小関係により挿入するインデックスキーを含むリーフノードを前記挿入されるノード対のどちらのノードとするかを決定するインデックスキーの挿入処理を繰り返すものであることを特徴とするインデックスキーの差分データ更新方法。

【請求項８】

請求項７記載のインデックスキーの差分データ更新方法において、

前記更新ツリー取得ステップは、前記更新ツリーを生成する際に、前記旧データのインデックスキーに加えて、前記差分データの挿入キーを取り出して前記インデックスキーの挿入処理を繰り返すものであり、

前記新旧インデックスキー更新ステップは、前記差分データから取り出した削除キーを前記更新ツリーから削除することにより新データのインデックスキーをリーフノードのインデックスキーとして格納した新データカップルドノードツリーを作成し、該新データカップルドノードツリーに基づいて、前記旧データを前記新データに更新するものであることを特徴とするインデックスキーの差分データ更新方法。

【請求項９】

請求項５記載のインデックスキーの差分データ更新方法において、

前記旧データのインデックスキーはカップルドノードツリーに格納されており、

前記更新ツリー取得ステップは、該カップルドノードツリーのルートノードの位置を示す位置情報を取得するものであり、

前記新旧インデックスキー更新ステップは、前記新データカップルドノードツリーを作

成することで前記旧データを前記新データに更新する、
ことを特徴とするインデックスキーの差分データ更新方法。
【請求項１０】
データベースのインデックスキーの旧データに対して該旧データと置き換えられる新しいインデックスキーの新データが供給されたときに、前記インデックスキーの更新を行うデータベースのインデックスキー更新に用いる前記旧データと前記新データの差分データ作成装置において、
前記旧データあるいは新データのインデックスキーが、
ルートノードと、隣接した記憶領域に配置されるブランチノードとリーフノードまたはブランチノード同士またはリーフノード同士のノード対、からなるビット列検索に用いるツリーであって、前記ルートノードは、ツリーの始点を表すノードであって、該ツリーのノードが１つのときは前記リーフノード、ツリーのノードが２つ以上のときは前記ブランチノードであり、前記ブランチノードは、ビット列検索を行う検索キーの弁別ビット位置とリンク先のノード対の一方のノードである代表ノードの位置を示す位置情報を含み、前記リーフノードは検索対象のビット列からなるインデックスキーを含み、前記ツリーの任意のノードを検索開始ノードとして前記ブランチノードにおいて、該ブランチノードに含まれる弁別ビット位置の検索キーのビット値に応じてリンク先のノード対の代表ノードかあるいはそれと隣接した記憶領域に配置されたノードにリンクすることを順次前記リーフノードに至るまで繰り返すことにより、前記リーフノードに格納されたインデックスキーを、前記検索開始ノードをルートノードとする前記ツリーの任意の部分木の前記検索キーによる検索結果である検索結果キーとするように構成されたカップルドノードツリーのリーフノードのインデックスキーとして格納されている場合は、該カップルドノードツリーを差分ツリーとし、登録されたルートノードの位置情報を取得し、前記旧データあるいは新データのインデックスキーが、カップルドノードツリーのリーフノードのインデックスキーとして格納されていない場合は、前記旧データあるいは新データのインデックスキーにより前記差分ツリーを生成する差分ツリー取得手段と、
前記差分ツリーのルートノードを前記検索開始ノードとして前記新データあるいは旧データの全てのインデックスキーを検索キーとして検索を行い、検索キーと一致した前記検索結果キーであるインデックスキーは差分ツリーから削除し、前記検索結果キーであるインデックスキーと一致しない検索キーを差分データの挿入キーあるいは削除キーとし、前記新データあるいは旧データの全てのインデックスキーによる検索が終了した後に前記差分ツリーに削除されずに残ったインデックスキーを差分データの削除キーあるいは挿入キーとして差分データを作成する差分データ作成手段と、
を備えることを特徴とする差分データ作成装置。
【請求項１１】
データベースのインデックスキーの旧データと該旧データと置き換えられる新しいインデックスキーの新データの差分データを用いて前記インデックスキーの更新を行うインデックスキーの差分データ更新装置において、
前記旧データのインデックスキーが、
ルートノードと、隣接した記憶領域に配置されるブランチノードとリーフノードまたはブランチノード同士またはリーフノード同士のノード対、からなるビット列検索に用いるツリーであって、前記ルートノードは、ツリーの始点を表すノードであって、該ツリーのノードが１つのときは前記リーフノード、ツリーのノードが２つ以上のときは前記ブランチノードであり、前記ブランチノードは、ビット列検索を行う検索キーの弁別ビット位置とリンク先のノード対の一方のノードである代表ノードの位置を示す位置情報を含み、前記リーフノードは検索対象のビット列からなるインデックスキーを含み、前記ツリーの任意のノードを検索開始ノードとして前記ブランチノードにおいて、該ブランチノードに含まれる弁別ビット位置の検索キーのビット値に応じてリンク先のノード対の代表ノードかあるいはそれと隣接した記憶領域に配置されたノードにリンクすることを順次前記リーフノードに至るまで繰り返すことにより、前記リーフノードに格納されたインデックスキーを

、前記検索開始ノードをルートノードとする前記ツリーの任意の部分木の前記検索キーによる検索結果である検索結果キーとするように構成されたカップルドノードツリーのリーフノードのインデックスキーとして格納されている場合は、該カップルドノードツリーを更新ツリーとし、登録されたルートノードの位置情報を取得し、前記旧データのインデックスキーが、カップルドノードツリーのリーフノードのインデックスキーとして格納されていない場合は、前記旧データのインデックスキーにより前記更新ツリーを生成する更新ツリー取得手段と、

前記旧データあるいは新データのインデックスキーを前記カップルドノードツリーのリーフノードのインデックスキーとして格納した差分ツリーのルートノードを前記検索開始ノードとして前記新データあるいは旧データの全てのインデックスキーを検索キーとして検索を行い、検索キーと一致した前記検索結果キーであるインデックスキーは差分ツリーから削除し、前記検索結果キーであるインデックスキーと一致しない検索キーを差分データの挿入キーあるいは削除キーとし、前記新データあるいは旧データの全てのインデックスキーによる検索が終了した後に前記差分ツリーに削除されずに残ったインデックスキーを差分データの削除キーあるいは挿入キーとして作成した差分データから取り出した削除キーを前記更新ツリーから削除し、前記差分データから取り出した挿入キーを前記更新ツリーに挿入することにより新データのインデックスキーをリーフノードのインデックスキーとして格納した新データカップルドノードツリーを作成し、該新データカップルドノードツリーに基づいて、前記旧データを前記新データに更新する新旧インデックスキー更新手段と、

を備えることを特徴とするインデックスキーの差分データ更新装置。

【請求項１２】

請求項１１記載のインデックスキーの差分データ更新装置において、

前記旧データのインデックスキーはカップルドノードツリーに格納されており、

前記更新ツリー取得手段は、該カップルドノードツリーのルートノードの位置を示す位置情報を取得するものであり、

前記新旧インデックスキー更新手段は、前記新データカップルドノードツリーを作成することで前記旧データを前記新データに更新する、

ことを特徴とするインデックスキーの差分データ更新装置。

【請求項１３】

データベースのインデックスキーの旧データに対して該旧データと置き換えられる新しいインデックスキーの新データが供給されたときに、前記インデックスキーの更新を行うデータベースのインデックスキー更新に用いられるコンピュータ上で動作するプログラムにおいて、

前記旧データあるいは新データのインデックスキーが、

ルートノードと、隣接した記憶領域に配置されるブランチノードとリーフノードまたはブランチノード同士またはリーフノード同士のノード対、からなるビット列検索に用いるツリーであって、前記ルートノードは、ツリーの始点を表すノードであって、該ツリーのノードが１つのときは前記リーフノード、ツリーのノードが２つ以上のときは前記ブランチノードであり、前記ブランチノードは、ビット列検索を行う検索キーの弁別ビット位置とリンク先のノード対の一方のノードである代表ノードの位置を示す位置情報を含み、前記リーフノードは検索対象のビット列からなるインデックスキーを含み、前記ツリーの任意のノードを検索開始ノードとして前記ブランチノードにおいて、該ブランチノードに含まれる弁別ビット位置の検索キーのビット値に応じてリンク先のノード対の代表ノードかあるいはそれと隣接した記憶領域に配置されたノードにリンクすることを順次前記リーフノードに至るまで繰り返すことにより、前記リーフノードに格納されたインデックスキーを、前記検索開始ノードをルートノードとする前記ツリーの任意の部分木の前記検索キーによる検索結果である検索結果キーとするように構成されたカップルドノードツリーのリーフノードのインデックスキーとして格納されている場合は、該カップルドノードツリーを差分ツリーとし、登録されたルートノードの位置情報を取得し、前記旧データあるいは

新データのインデックスキーが、カップルドノードツリーのリーフノードのインデックスキーとして格納されていない場合は、前記旧データあるいは新データのインデックスキーにより前記差分ツリーを生成する差分ツリー取得ステップと、

前記差分ツリーのルートノードを前記検索開始ノードとして前記新データあるいは旧データの全てのインデックスキーを検索キーとして検索を行い、検索キーと一致した前記検索結果キーであるインデックスキーは差分ツリーから削除し、前記検索結果キーであるインデックスキーと一致しない検索キーを差分データの挿入キーあるいは削除キーとし、前記新データあるいは旧データの全てのインデックスキーによる検索が終了した後に前記差分ツリーに削除されずに残ったインデックスキーを差分データの削除キーあるいは挿入キーとして差分データを作成する差分データ作成ステップと、

をコンピュータに実行させることを特徴とするプログラム。

【請求項１４】

データベースのインデックスキーの旧データに対して該旧データと置き換えられる新しいインデックスキーの新データが供給されたときに、前記インデックスキーの更新を行うデータベースのインデックスキー更新に用いられるコンピュータ上で動作するプログラムにおいて、

前記旧データのインデックスキーが、

ルートノードと、隣接した記憶領域に配置されるブランチノードとリーフノードまたはブランチノード同士またはリーフノード同士のノード対、からなるビット列検索に用いるツリーであって、前記ルートノードは、ツリーの始点を表すノードであって、該ツリーのノードが１つのときは前記リーフノード、ツリーのノードが２つ以上のときは前記ブランチノードであり、前記ブランチノードは、ビット列検索を行う検索キーの弁別ビット位置とリンク先のノード対の一方のノードである代表ノードの位置を示す位置情報を含み、前記リーフノードは検索対象のビット列からなるインデックスキーを含み、前記ツリーの任意のノードを検索開始ノードとして前記ブランチノードにおいて、該ブランチノードに含まれる弁別ビット位置の検索キーのビット値に応じてリンク先のノード対の代表ノードかあるいはそれと隣接した記憶領域に配置されたノードにリンクすることを順次前記リーフノードに至るまで繰り返すことにより、前記リーフノードに格納されたインデックスキーを、前記検索開始ノードをルートノードとする前記ツリーの任意の部分木の前記検索キーによる検索結果である検索結果キーとするように構成されたカップルドノードツリーのリーフノードのインデックスキーとして格納されている場合は、該カップルドノードツリーを更新ツリーとし、登録されたルートノードの位置情報を取得し、前記旧データのインデックスキーが、カップルドノードツリーのリーフノードのインデックスキーとして格納されていない場合は、前記旧データのインデックスキーにより前記更新ツリーを生成する更新ツリー取得ステップと、

請求項１３記載の前記差分データ作成ステップで作成された差分データから取り出した削除キーを前記更新ツリーから削除し、前記差分データから取り出した挿入キーを前記更新ツリーに挿入することにより新データのインデックスキーをリーフノードのインデックスキーとして格納した新データカップルドノードツリーを作成し、該新データカップルドノードツリーに基づいて、前記旧データを前記新データに更新する新旧インデックスキー更新ステップと、

をコンピュータに実行させることを特徴とするプログラム。

【請求項１５】

請求項１３又は請求項１４記載のプログラムを記憶したことを特徴とするコンピュータ読み取り可能な記憶媒体。

【発明の詳細な説明】

【技術分野】

【０００１】

本発明は、データベースのインデックスキーの更新方法及びプログラムに関する。

【背景技術】

【０００２】
近年、社会の情報化が進展し、大規模なデータベースが各所で利用されるようになってきている。このような大規模なデータベースからレコードを検索するには、各レコードの記憶されたアドレスと対応づけられたレコード内の項目をインデックスキーとして検索をし、所望のレコードを探し出すことが通例である。また、全文検索における文字列も、文書のインデックスキーと見なすことができる。
【０００３】
そして、それらのインデックスキーはビット列で表現されることから、データベースの検索はビット列の検索に帰着されるということができる。
本発明の背景技術の１つとして、まず上述のビット列の検索処理について説明する。
【０００４】
ビット列の検索処理手法については、種々のものが知られている。それらのなかでも、ビット列の検索を高速に行うために、ビット列を記憶するデータ構造を種々に工夫することが従来から行われている。このようなものの一つとして、パトリシアツリーという木構造が知られている。
【０００５】
図１は、上述の従来の検索処理に用いられているパトリシアツリーの一例を示すものである。パトリシアツリーのノードは、インデックスキー、検索キーの検査ビット位置、左右のリンクポインタを含んで構成される。明示はされていないが、ノードにはインデックスキーに対応するレコードにアクセスするための情報が含まれていることは勿論である。
【０００６】
図１の例では、インデックスキー“１０００１０”を保持するノード１７５０ａがルートノードとなっており、その検査ビット位置は０である。ノード１７５０ａの左リンク１７４０ａにはノード１７５０ｂが接続され、右リンク１７４１ａにはノード１７５０ｆが接続されている。
【０００７】
ノード１７５０ｂの保持するインデックスキーは“０１００１１”であり、検査ビット位置２０３０ｂは１である。ノード１７５０ｂの左リンク１７４０ｂにはノード１７５０ｃが、右リンク１７４１ｂにはノード１７５０ｄが接続されている。ノード１７５０ｃが保持するインデックスキーは“０００１１１”、検査ビット位置は３である。ノード１７５０ｄが保持するインデックスキーは“０１１０１０”、検査ビット位置は２である。
【０００８】
ノード１７５０ｃから実線で接続された部分はノード１７５０ｃの左右のリンクポインタを示すものであり、点線の接続されていない左ポインタ１７４０ｃは、その欄が空欄であることを示している。点線の接続された右ポインタ１７４１ｃの点線の接続先は、ポインタの示すアドレスを表しており、今の場合ノード１７５０ｃを右ポインタが指定していることを表している。
【０００９】
ノード１７５０ｄの右ポインタ１７４１ｄはノード１７５０ｄ自身を指しており、左リンク１７４０ｄにはノード１７５０ｅが接続されている。ノード１７５０ｅの保持するインデックスキーは“０１００１０”、検査ビット位置は５である。ノード１７５０ｅの左ポインタ１７４０ｅはノード１７５０ｂを、右ポインタ１７４１ｅはノード１７５０ｅを指している。
【００１０】
また、ノード１７５０ｆの保持するインデックスキーは“１０１０１１”であり、検査ビット位置１７３０ｆは２である。ノード１７５０ｆの左リンク１７４０ｆにはノード１７５０ｇが、右リンク１７４１ｆにはノード１７５０ｈが接続されている。
【００１１】
ノード１７５０ｇの保持するインデックスキーは“１０００１１”であり、検査ビット位置１７３０ｇは５である。ノード１７５０ｇの左ポインタ１７４０ｇはノード１７５０

ａを、右ポインタ１７４１ｇはノード１７５０ｇを指している。
【００１２】
ノード１７５０ｈの保持するインデックスキーは“１０１１００”であり、検査ビット位置１７３０ｈは3である。ノード１７５０ｈの左ポインタ１７４０ｈはノード１７５０ｆを、右ポインタ１７４１ｈはノード１７５０ｈを指している。
【００１３】
図１の例では、ルートノード１７５０ａからツリーを降りるにしたがって、各ノードの検査ビット位置が大きくなるように構成されている。
ある検索キーで検索を行うとき、ルートノードから順次各ノードに保持される検索キーの検査ビット位置を検査していき、検査ビット位置のビット値が１であるか０であるか判定を行い、１であれば右リンクをたどり、０であれば左リンクをたどる。そして、リンク先のノードの検査ビット位置がリンク元のノードの検査ビット位置より大きくなければ、すなわち、リンク先が下方でなく上方に戻れば（図１において点線で示されたこの逆戻りのリンクをバックリンクという）、リンク先のノードのインデックスキーと検索キーの比較を行う。比較の結果、等しければ検索成功であり、等しくなければ検索失敗であることが保証されている。
【００１４】
上記のように、パトリシアツリーを用いた検索処理では、必要なビットの検査だけで検索できること、キー全体の比較は１回ですむことなどのメリットがあるが、各ノードからの２つのリンクが必ずあることにより記憶容量が増大することや、バックリンクの存在による判定処理の複雑化、バックリンクにより戻ることで初めてインデックスキーと比較することによる検索処理の遅延及び追加削除等データメンテナンスの困難性などの欠点がある。
【００１５】
これらのパトリシアツリーの欠点を解消しようとするものとして、例えば下記特許文献１に開示された技術がある。下記特許文献１に記載されたパトリシアツリーにおいては、下位の左右のノードは連続した領域に記憶することによりポインタの記憶容量を削減するとともに、次のリンクがバックリンクであるか否かを示すビットを各ノードに設けることにより、バックリンクの判定処理を軽減している。
【００１６】
しかしながら、下記特許文献１に開示されたものにおいても、１つのノードは必ずインデックスキーの領域とポインタの領域を占めること、下位の左右のノードを連続した領域に記憶するようにしてポインタを１つとしたため、例えば図１に示したパトリシアツリーの最下段の部分である左ポインタ１７４０ｃ、右ポインタ１７４１ｈ等の部分にもノードと同じ容量の記憶領域を割り当てる必要があるなど、記憶容量の削減効果はあまり大きいものではない。また、バックリンクによる検索処理の遅延の問題や追加削除等の処理が困難であることも改善されていない。
【００１７】
次に、本発明の背景技術の別の１つとして、データベースの更新処理について説明する。
データベースシステムの機能として、データの更新機能は必須である。データベースに保管されるデータ量が増大するのに伴って、例えばバッチ処理により既存のデータベースに対して大量のデータの追加や削除を行おうとすると、その作業に要する時間が長くなるという不都合が生じている。
【００１８】
データベースのバッチ更新の態様としては、既存のデータベースに対する追加、変更あるいは削除データによりデータベースを更新する態様と、既存のデータベースを新しいデータベースでそっくり置き換える態様が存在する。
【００１９】
後者のデータベース更新の態様は、例えばデータベースのデータの供給者が、データベ

ースの更新が必要になったときにデータベースの更新データを供給するのではなく、データを更新済みの新しい版のデータベースを改めて供給する場合に採用される。しかし、このようなデータベースの更新の態様では、更新作業中はデータベースの利用ができないことから、この更新作業に長時間を要するのでは不便である。

【００２０】

また、データベースは、データベース本体のデータ部分とデータベース本体からデータを検索するためのインデックスから構成されるのが通例である。そこで、データベースの更新の態様には、インデックス部分を更新する態様があり、この部分の更新においても、更新後のインデックス全体が供給され、更新前のインデックスとそっくり入れ替える場合がある。

【００２１】

例えばカーナビゲーションシステムの地図データのインデックスを更新する場合には、地図データの販売業者から購入した新しい地図データのインデックスデータを、カーナビゲーションシステムの販売業者あるいは車のディーラーのセンターから、カーナビゲーションシステムを搭載した各車に配布し、搭載されたカーナビゲーションシステムごとに、インデックスデータを更新する。このように、新しいインデックスデータ全体を配布するため、配布するデータ量が多く、また各カーナビゲーションシステムにおける地図データの更新時間も長いものとなる。

【特許文献１】特開２００１－３５７０７０号公報

【発明の開示】

【発明が解決しようとする課題】

【００２２】

そこで、本発明の課題は、既存のデータベースのインデックスキーに対してそれと置き換えられる新しいデータベース用のインデックスキーが供給されたときに、効率よくデータベースのインデックスキーの更新を行うことができるデータベース更新処理の手法を提供することである。

【００２３】

本出願人は、特願２００６－１８７８２７において、ルートノードと、隣接した記憶領域に配置されるブランチノードとリーフノードまたはブランチノード同士またはリーフノード同士のノード対からなるビット列検索に用いるツリーであって、ルートノードはツリーの始点を表すノードであって、該ツリーのノードが１つのときはリーフノード、ツリーのノードが２つ以上のときは前記ブランチノードであり、前記ブランチノードは、ビット列検索を行う検索キーの弁別ビット位置とリンク先のノード対の一方のノードの位置を示す位置情報を含み、前記リーフノードは検索対象のビット列からなるインデックスキーを含むカップルドノードツリーを用いたビット列検索を提案した。

【００２４】

上記出願においては、与えられたインデックスキーの集合からカップルドノードツリーを生成する方法と、カップルドノードツリーから単一のインデックスキーを検索する手法等の、カップルドノードツリーを用いた基本的な検索手法が示されている。

【００２５】

また、ビット列の検索には、最小値、最大値を求める、ある範囲の値のものを求める等の各種の検索要求が存在する。そこで、本出願人は、特願２００６－２９３６１９において、カップルドノードツリーの任意の部分木に含まれるインデックスキーの最大値／最小値を求める手法等を提案した。

【００２６】

本発明は、このカップルドノードツリーを応用した高速なデータベースのインデックスキー更新手法を実現することを目的とする。

【課題を解決するための手段】

【００２７】

本発明によれば、インデックスキーの旧データと新データの差分データを作成し、差分

データにより旧データを更新して新データを作成する。
その際、旧データあるいは新データのインデックスキーをリーフノードのインデックスキーとして格納したカップルドノードツリーである差分ツリーを取得し、差分ツリーから新データあるいは旧データのインデックスキーを検索キーとして検索を行い、検索キーと一致したインデックスキーは差分ツリーから削除し、旧データ中のインデックスキーと一致しない検索キーは差分データの挿入キーあるいは削除キーとし、新データあるいは旧データの全てのインデックスキーによる検索が終了した後に差分ツリーに残った旧データのインデックスキーを差分データの削除キーあるいは挿入キーとして差分データを作成する。

【００２８】
差分データによる旧データの新データへの更新においては、旧データのインデックスキーをリーフノードのインデックスキーとして格納したカップルドノードツリーである更新ツリーを取得し、差分データから取り出した削除キーを更新ツリーから削除し、差分データから取り出した挿入キーを更新ツリーに挿入することにより新データを作成する。

【発明の効果】

【００２９】
本発明によれば、カップルドノードツリーを用いることにより、高速で差分データを作成することができる。そして、インデックスキーを更新するサイトに配布するのは、新データ全体ではなく差分データなので、配布するデータ量を削減することができる。また、インデックスキーの更新にもカップルドノードツリーを用いることにより、高速なデータ更新が可能となる。

【発明を実施するための最良の形態】

【００３０】
最初に、本出願人により先の上記出願において提案された、本発明で利用するカップルドノードツリーについて、カップルドノードツリーを配列に格納する例を説明する。ブランチノードが保持するリンク先の位置を示すデータとして、記憶装置のアドレス情報とすることもできるが、ブランチノードあるいはリーフノードのうち占有する領域の記憶容量の大きい方を格納可能な配列要素からなる配列を用いることにより、ノードの位置を配列番号で表すことができ、位置情報の情報量を削減することができる。

【００３１】
図２Ａは、配列に格納されたカップルドノードツリーの構成例を説明する図である。
図２Ａを参照すると、ノード１０１が配列１００の配列番号１０の配列要素に配置されている。ノード１０１はノード種別１０２、弁別ビット位置１０３及び代表ノード番号１０４で構成されている。ノード種別１０２は０であり、ノード１０１がブランチノードであることを示している。弁別ビット位置１０３には１が格納されている。代表ノード番号１０４にはリンク先のノード対の代表ノードの配列番号２０が格納されている。なお、以下では表記の簡略化のため、代表ノード番号に格納された配列番号を代表ノード番号ということもある。また、代表ノード番号に格納された配列番号をそのノードに付した符号あるいはノード対に付した符号で表すこともある。

【００３２】
配列番号２０の配列要素には、ノード対１１１の代表ノードであるノード［０］１１２が格納されている。そして隣接する次の配列要素（配列番号２０＋１）に代表ノードと対になるノード［１］１１３が格納されている。ノード［０］１１２のノード種別１１４には０が、弁別ビット位置１１５には３が、代表ノード番号１１６には３０が格納されている。またノード［１］１１３のノード種別１１７には１が格納されており、ノード［１］１１３がリーフノードであることを示している。インデックスキー１１８には、“０００１”が格納されている。パトリシアツリーについて先に述べたと同様に、リーフノードにインデックスキーと対応するレコードにアクセスする情報が含まれることは当然であるが、表記は省略している。

【００３３】

なお、代表ノードをノード［０］で表し、それと対になるノードをノード［１］で表すことがある。また、ある配列番号の配列要素に格納されたノードを、その配列番号のノードということがあり、ノードの格納された配列要素の配列番号を、ノードの配列番号ということもある。

【００３４】

配列番号３０及び３１の配列要素に格納されたノード１２２とノード１２３からなるノード対１２１の内容は省略されている。

ノード［０］１１２、ノード［１］１１３、ノード１２２、及びノード１２３の格納された配列要素にそれぞれ付された０あるいは１は、検索キーで検索を行う場合にノード対のどちらのノードにリンクするかを示すものである。前段のブランチノードの弁別ビット位置にある検索キーのビット値である０か１を代表ノード番号に加えた配列番号のノードにリンクする。

【００３５】

したがって、前段のブランチノードの代表ノード番号に、検索キーの弁別ビット位置のビット値を加えることにより、リンク先のノードが格納された配列要素の配列番号を求めることができる。

【００３６】

なお、上記の例では代表ノード番号をノード対の配置された配列番号のうち小さい方を採用しているが、大きいほうを採用することも可能であることは明らかである。

図２Ｂは、カップルドノードツリーのツリー構造を概念的に示す図である。図示の６ビットのインデックスキーは、図１に例示されたパトリシアツリーのものと同じである。

【００３７】

符号２１０ａで示すのがルートノードである。図示の例では、ルートノード２１０ａは配列番号２２０に配置されたノード対２０１ａの代表ノードとしている。

ツリー構造としては、ルートノード２１０ａの下にノート対２０１ｂが、その下層にノード対２０１ｃとノード対２０１ｆが配置され、ノード対２０１ｆの下層にはノード対２０１ｈとノード対２０１ｇが配置されている。ノード対２０１ｃの下にはノード対２０１ｄが、さらにその下にはノード対２０１ｅが配置されている。

【００３８】

各ノードの前に付された０あるいは１の符号は、図２Ａにおいて説明した配列要素の前に付された符号と同じである。検索キーの弁別ビット位置のビット値に応じてツリーをたどり、検索対象のリーフノードを見つけることになる。

【００３９】

図示された例では、ルートノード２１０ａのノード種別２６０ａは０でブランチノードであることを示し、弁別ビット位置２３０ａは０を示している。代表ノード番号は２２０ａであり、それはノード対２０１ｂの代表ノード２１０ｂの格納された配列要素の配列番号である。

【００４０】

ノード対２０１ｂはノード２１０ｂと２１１ｂで構成され、それらのノード種別２６０ｂ、２６１ｂはともに０であり、ブランチノードであることを示している。ノード２１０ｂの弁別ビット位置２３０ｂには１が格納され、リンク先の代表ノード番号にはノード対２０１ｃの代表ノード２１０ｃの格納された配列要素の配列番号２２０ｂが格納されている。

【００４１】

ノード２１０ｃのノード種別２６０ｃには１が格納されているので、このノードはリーフノードであり、したがって、インデックスキーを含んでいる。インデックスキー２５０ｃには“０００１１１”が格納されている。一方ノード２１１ｃのノード種別２６１ｃは０、弁別ビット位置２３１ｃは２であり、代表ノード番号にはノード対２０１ｄの代表ノード２１０ｄの格納された配列要素の配列番号２２１ｃが格納されている。

【００４２】

ノード２１０ｄのノード種別２６０ｄは０、弁別ビット位置２３０ｄは５であり、代表ノード番号にはノード対２０１ｅの代表ノード２１０ｅの格納された配列要素の配列番号２２０ｄが格納されている。ノード２１０ｄと対になるノード２１１ｄのノード種別２６１ｄは１であり、インデックスキー２５１ｄには“０１１０１０”が格納されている。

【００４３】

ノード対２０１ｅのノード２１０ｅ、２１１ｅのノード種別２６０ｅ、２６１ｅはともに１であり双方ともリーフノードであることを示し、それぞれのインデックスキー２５０ｅ、２５１ｅにはインデックスキーとして“０１００１０”と“０１００１１”が格納されている。

【００４４】

ノード対２０１ｂのもう一方のノードであるノード２１１ｂの弁別ビット位置２３１ｂには２が格納され、リンク先の代表ノード番号にはノード対２０１ｆの代表ノード２１０ｆの格納された配列要素の配列番号２２１ｂが格納されている。

【００４５】

ノード対２０１ｆのノード２１０ｆ、２１１ｆのノード種別２６０ｆ、２６１ｆはともに０であり双方ともブランチノードである。それぞれの弁別ビット位置２３０ｆ、２３１ｆには５、３が格納されている。ノード２１０ｆの代表ノード番号にはノード対２０１ｇの代表ノード２１０ｇの格納された配列要素の配列番号２２０ｆが格納され、ノード２１１ｆの代表ノード番号にはノード対２０１ｈの代表ノードであるノード［０］２１０ｈの格納された配列要素の配列番号２２１ｆが格納されている。

【００４６】

ノード対２０１ｇのノード２１０ｇ、２１１ｇのノード種別２６０ｇ、２６１ｇはともに１であり双方ともリーフノードであることを示し、それぞれのインデックスキー２５０ｇ、２５１ｇには“１０００１０”と“１０００１１”が格納されている。

【００４７】

また同じくノード対２０１ｈの代表ノードであるノード［０］２１０ｈとそれと対をなすノード［１］２１１ｈのノード種別２６０ｈ、２６１ｈはともに１であり双方ともリーフノードであることを示し、それぞれのインデックスキー２５０ｈ、２５１ｈには“１０１０１１”と“１０１１００“が格納されている。

【００４８】

以下、上述のツリーからインデックスキー“１０００１０”を検索する処理の流れを簡単に説明する。弁別ビット位置は、左から０、１、２、・・・とする。

まず、ビット列“１０００１０”を検索キーとしてルートノード２１０ａから処理をスタートする。ルートノード２１０ａの弁別ビット位置２３０ａは０であるので、検索キー“１０００１０”の弁別ビット位置が０のビット値をみると１である。そこで代表ノード番号の格納された配列番号２２０ａに１を加えた配列番号の配列要素に格納されたノード２１１ｂにリンクする。ノード２１１ｂの弁別ビット位置２３１ｂには２が格納されているので、検索キー“１０００１０”の弁別ビット位置が２のビット値をみると０であるから、代表ノード番号の格納された配列番号２２１ｂの配列要素に格納されたノード２１０ｆにリンクする。

【００４９】

ノード２１０ｇのノード種別２６０ｇは１でありリーフノードであることを示しているので、インデックスキー２５０ｇを読み出して検索キーと比較すると両方とも“１０００１０”であって一致している。このようにしてカップルドノードツリーを用いた検索が行われる。

【００５０】

次に、図２Ｂを参照してカップルドノードツリーの構成の意味について説明する。

カップルドノードツリーの構成はインデックスキーの集合により規定される。図２Ｂの例で、ルートノード２１０ａの弁別ビット位置が０であるのは、図２Ｂに例示されたインデックスキーに０ビット目が０のものと１のものがあるからである。０ビット目が０のイ

ンデックスキーのグループはノード２１０ｂの下に分類され、０ビット目が１のインデックスキーのグループはノード２１１ｂの下に分類されている。

【００５１】

ノード２１１ｂの弁別ビット位置が２であるのは、ノード２１１ｈ、２１０ｈ、２１１ｇ、２１０ｇに格納された０ビット目が１のインデックスキーの１ビット目がすべて０で等しく、２ビット目で初めて異なるものがあるという、インデックスキーの集合の性質を反映している。

【００５２】

以下０ビット目の場合と同様に、２ビット目が１であるものはノード２１１ｆ側に分類され、２ビット目が０であるものはノード２１０ｆ側に分類される。

そして２ビット目が１であるインデックスキーは３ビット目の異なるものがあるのでノード２１１ｆの弁別ビット位置には３が格納され、２ビット目が０であるインデックスキーでは３ビット目も４ビット目も等しく５ビット目で異なるのでノード２１０ｆの弁別ビット位置には５が格納される。

【００５３】

ノード２１１ｆのリンク先においては、３ビット目が１のものと０のものがそれぞれ１つしかないことから、ノード２１０ｈ、２１１ｈはリーフノードとなり、それぞれインデックスキー２５０ｈと２５１ｈに“１０１０１１”と“１０１１００”が格納されている。

【００５４】

仮にインデックスキーの集合に“１０１１００”の代わりに“１０１１０１”か“１０１１１０”が含まれていたとしても、３ビット目までは“１０１１００”と等しいので、ノード２１１ｈに格納されるインデックスキーが変わるだけで、ツリー構造自体は変わることはない。しかし、“１０１１００”に加えて“１０１１０１”が含まれていると、ノード２１１ｈはブランチノードとなり、その弁別ビット位置は５になる。追加されるインデックスキーが“１０１１１０”であれば、弁別ビット位置は４となる。

【００５５】

以上説明したように、カップルドノードツリーの構造は、インデックスキーの集合に含まれる各インデックスキーの各ビット位置のビット値により決定される。

そしてさらにいえば、異なるビット値となるビット位置ごとにビット値が“１”のノードとビット値が“０”のノードとに分岐していることから、ノード［１］側とツリーの深さ方向を優先させてリーフノードをたどると、それらに格納されたインデックスキーは、ノード２１１ｈのインデックスキー２５１ｈの“１０１１００”、ノード２１０ｈのインデックスキー２５０ｈの“１０１０１１”、・・・、ノード２１０ｃのインデックスキー２５０ｃの“０００１１１”となり降順にソートされている。

【００５６】

すなわち、カップルドノードツリーにおいては、インデックスキーはソートされてツリー上に配置されている。

検索キーで検索するときはインデックスキーがカップルドノードツリー上に配置されたルートをたどることになり、例えば検索キーが“１０１１００”であればノード２１１ｈに到達することができる。また、上記説明からも想像がつくように、“１０１１０１”か“１０１１１０”を検索キーとした場合でもノード２１１ｈにたどり着き、インデックスキー２５１ｈと比較することにより検索が失敗したことが分かる。

【００５７】

また、例えば“１００１００”で検索した場合でも、ノード２１０ａ、２１１ｂ、２１０ｆのリンク経路では検索キーの３ビット目と４ビット目は使われることがなく、“１００１００”の５ビット目が０なので、“１０００１０”で検索した場合と同様にノード２１０ｇに到達することになる。このように、カップルドノードツリーに格納されたインデックスキーのビット構成に応じた弁別ビット位置を用いて分岐が行われる。

【００５８】

次に、本発明の原理を説明する。
図３Ａは、本発明の原理を模式的に示す図である。図３Ａの（１）に示すのは、旧データと新データから差分データを作成する原理を説明する図であり、図３Ａの（２）に示すのは、差分データにより旧データを新データに更新する原理を説明する図である。
【００５９】
図３Ａの（１）に示すように、差分データを作成するには、新データに含まれる全てのキーで旧データに含まれるキーを検索する。旧データと一致したキー、すなわち旧データと新データに存在するキーについては、更新の対象外であるから、旧データから削除する。
【００６０】
旧データとは一致しなかった検索キー、すなわち新データに追加されたキーは、挿入キーとして差分データに取り込まれる。新データにより検索されず旧データに残存したキー、すなわち新データに存在しないキーは、削除キーとして差分データに取り込まれる。
なお、上述の説明では、新データに含まれる全てのキーで旧データに含まれるキーを検索する、としたが、差分データを作成するのであるから、旧データに含まれる全てのキーで新データに含まれるキーを検索する、として差分データを作成してもよいことは明らかである。その場合は、新データに残存したキーが挿入キーとして差分データに取り込まれ、新データと一致しなかった旧データが削除キーとして差分データに取り込まれる。
【００６１】
図３Ａの（２）に示すように、差分データにより旧データを新データに更新するには、挿入キーを新データに追加されたキーとして旧データに挿入し、削除キーにより新データに存在しないキーを旧データから削除する。
【００６２】
図３Ｂは、本発明を実施するためのハードウェア構成例を説明する図である。図に示すように、本発明を実施するためのハードウェアは、差分データ作成装置３００と差分データ更新装置４００ａ～４００ｘで構成される。差分データ更新装置４００ａ～４００ｘの数は任意である。
【００６３】
本発明の差分データ作成装置３００による差分データ作成は中央処理装置３０２及びキャッシュメモリ３０３を少なくとも備えたデータ処理装置３０１によりデータ格納装置３０８を用いて実施される。カップルドノードツリーが配置される配列３０９と検索中にたどるノードが格納された配列要素の配列番号を記憶する探索経路スタック３１０、旧データを格納する旧データ格納領域３２１、新データを格納する新データ格納領域３２２及び作成済みの差分データを格納する差分データ格納領域３２０を有するデータ格納装置３０８は、主記憶装置３０５または外部記憶装置３０６で実現することができ、あるいは通信装置３０７を介して接続された遠方に配置された装置を用いることも可能である。
【００６４】
図３Ｂの例示では、主記憶装置３０５、外部記憶装置３０６及び通信装置３０７が一本のバス３０４によりデータ処理装置３０１に接続されているが、接続方法はこれに限るものではない。また、主記憶装置３０５をデータ処理装置３０１内のものとすることもできるし、探索経路スタック３１０を中央処理装置３０２内のハードウェアとして実現することも可能である。あるいは、旧データ格納領域３２１、新データ格納領域３２２及び差分データ格納領域３２０は外部記憶装置３０６に、探索経路スタック３１０を主記憶装置３０５に持つなど、使用可能なハードウェア環境、インデックスキー集合の大きさ等に応じて適宜ハードウェア構成を選択できることは明らかである。
【００６５】
また、特に図示されてはいないが、処理の途中で得られた各種の値を後の処理で用いるためにそれぞれの処理に応じた一時記憶装置が用いられることは当然である。
差分データ作成装置３００において作成された差分データは、差分データ更新装置４０

０ａ～４００ｘに送られ、それぞれの差分データ格納領域４２０ａ～４２０ｘに格納され、それぞれのインデックス格納領域４２１ａ～４２１ｘに格納された旧データを新データに更新するために用いられる。特に図示はしていないが、差分データ更新装置４００ａ～４００ｘにおいても、データ処理装置や、差分データ格納領域、インデックス格納領域及びその他の記憶領域を有するデータ格納装置が備えられている。

なお、以下においては、上述の例えば差分データ格納領域３２０に格納された差分データを差分データ３２０と表記するように、あるデータ格納領域に格納されるデータ自体にデータ格納領域の符号を付して説明する場合がある。

【００６６】

図３Ｃは、差分データ作成装置３００の機能ブロック構成を説明する図である。

図に示すように、差分データ作成装置３００は、差分ツリー取得手段３３０と差分データ作成手段３３１の機能ブロックを含む。これらの機能ブロックは、図３Ｂに例示するハードウェアと以下において説明する処理フローを実行するソフトウェアにより実現される。

差分ツリー取得手段３３０は、旧データあるいは新データのインデックスキーをカップルドノードツリーのリーフノードのインデックスキーとして格納した、差分ツリーを取得する。

差分データ作成手段３３１は、差分ツリーのルートノードを検索開始ノードとして新データあるいは旧データの全てのインデックスキーを検索キーとして検索を行い、検索キーと一致した検索結果キーであるインデックスキーは差分ツリーから削除し、検索結果キーであるインデックスキーと一致しない検索キーを差分データの挿入キーあるいは削除キーとする。そして、新データあるいは旧データの全てのインデックスキーによる検索が終了した後に差分ツリーに削除されずに残ったインデックスキーを差分データの削除キーあるいは挿入キーとして差分データを作成する。

【００６７】

図３Ｄは、差分データ更新装置の機能ブロック構成を説明する図である。

図に示すように、差分データ更新装置４００は、更新ツリー取得手段４２２と新旧インデックスキー更新手段４２３の機能ブロックを含む。これらの機能ブロックは、図３Ｂに示す差分データ更新装置４００ａ～４００ｘについて説明したハードウェアと以下において説明する処理フローを実行するソフトウェアにより実現される。

更新ツリー取得手段４２２は、旧データのインデックスキーをカップルドノードツリーのリーフノードのインデックスキーとして格納した更新ツリーを取得する。

新旧インデックスキー更新手段４２３は、差分データ作成手段３３１が作成した差分データから取り出した削除キーを更新ツリーから削除し、差分データから取り出した挿入キーを更新ツリーに挿入することにより新データのインデックスキーをリーフノードのインデックスキーとして格納した新データカップルドノードツリーを作成し、新データカップルドノードツリーに基づいて、旧データを新データに更新する。

【００６８】

次に、上述の出願において、本出願人により提案されたカップルドノードツリーを用いた基本的な検索処理、カップルドノードツリーにおける挿入削除処理、及びカップルドノードツリーに含まれるインデックスキーの最小値を求める処理等の応用処理の一部について、本発明を理解するために必要な範囲で紹介する。

【００６９】

図４は、本出願人による出願である上記特願２００６－２９３６１９で提案されたビット列検索の基本動作を示したフローチャートである。

まず、ステップＳ４０１で、検索開始ノードの配列番号を取得する。取得された配列番号に対応する配列は、カップルドノードツリーを構成する任意のノードを格納したものである。検索開始ノードの指定は、後に説明する各種応用検索において行われる。

取得された検索開始ノードの配列番号は、図示しない検索開始ノード設定エリアに設定されるが、この検索開始ノード設定エリアは、先に述べた「処理の途中で得られた各種の値を後の処理で用いるためにそれぞれの処理に応じた一時記憶装置」の一つである。以下の説明では、「図示しない検索開始ノード設定エリアに設定する」のような表現に変えて、「検索開始ノードの配列番号を得る。」、「検索開始ノードとして設定する」あるいは単に「検索開始ノードに設定する」のように記述することもある。

【００７０】

次に、ステップＳ４０２で、探索経路スタック３１０に取得された配列番号を格納し、ステップＳ４０３で、その配列番号に対応する配列要素を参照すべきノードとして読み出す。そして、ステップＳ４０４で、読み出したノードから、ノード種別を取り出し、ステップＳ４０５で、ノード種別がブランチノードであるか否かを判定する。

【００７１】

ステップＳ４０５の判定において、読み出したノードがブランチノードである場合は、ステップＳ４０６に進み、ノードから弁別ビット位置についての情報を取り出し、更に、ステップＳ４０７で、取り出した弁別ビット位置に対応するビット値を検索キーから取り出す。そして、ステップＳ４０８で、ノードから代表ノード番号を取り出して、ステップＳ４０９で、検索キーから取り出したビット値と代表ノード番号とを加算し、新たな配列番号として、ステップＳ４０２に戻る。

【００７２】

以降、ステップＳ４０５の判定においてリーフノードと判定されてステップＳ４１０に進むまで、ステップＳ４０２からステップＳ４０９までの処理を繰り返す。ステップＳ４１０で、リーフノードからインデックスキーを取り出して、処理を終了する。

【００７３】

図５は、本出願人による出願である上記特願２００６－２９３６１９で提案されたカップルドノードツリー（部分木を含む）に格納されたインデックスキーの最小値を求める処理を示したフローチャートである。先に述べたようなインデックスキーのツリー上の配置から、インデックスキーの最小値を求める処理は検索開始ノードからノード［０］をリーフノードに至るまでツリー上でたどることに相当する。

【００７４】

まず、ステップＳ５０１の検索開始ノードの配列番号の取得からステップＳ５０５のノード種別の判定までは、それぞれ上述の図４のステップＳ４０１からステップＳ４０５の処理と同様である。

【００７５】

ステップＳ５０５のノード種別の判定においてノード種別がブランチノードと判定されると、ステップＳ５０６に進み、ノードから配列の代表ノード番号を取り出し、ステップＳ５０７で、取り出した代表ノード番号に値“０”を加算してその結果を新たな配列番号とし、ステップＳ５０２に戻る。以降、ステップＳ５０５においてそのノードがリーフノードと判定されるまでステップＳ５０２からステップＳ５０７までの処理を繰り返し、ステップＳ５０８で、リーフノードからインデックスキーを取り出し、処理を終了する。

【００７６】

上記の図５に示す処理においては、ノード［０］をたどるため、代表ノード番号に一律“０”を加算している。すなわち、図５の処理によれば、リンク先のノードは、ノード対のうち必ずノード［０］とし、より小さい値のインデックスキーが格納されているノードのほうに分岐している。これにより、ツリー構造が先に述べたように順列構成であるカップルドノードツリーの最小のインデックスキーを取り出すことができる。

【００７７】

なお、図示はしていないが、図５のステップＳ５０７において、代表ノード番号に値“０”を加算する代わりに値“１”を加算すればノード［１］をたどることになり、カップルドノードツリーの最大のインデックスキーを取り出すことができる。

【００７８】

図６は、カップルドノードツリーに格納されたインデックスキーを昇順に取り出す処理を示したフローチャートである。インデックスキーを昇順に取り出す処理は、ノード対をなすノードのうちノード［０］側及びツリーの深さを優先させてノードから順次リーフノードをたどり、各リーフノードからインデックスキーを取り出していくことに相当する。

【００７９】

まず、ステップＳ６０１で、検索開始ノードの配列番号にルートノードの配列番号を設定し、ステップＳ６０２で、上記図５を参照して説明した最小のインデックスキーを求める処理を実行して、カップルドノードツリーに含まれるインデックスキーの中で最小のインデックスキーを取得する。そして、ステップＳ６０３で、取得したインデックスキーを取り出し、ステップＳ６０４に進む。

【００８０】

ステップＳ６０４で、探索経路スタック３１０のスタックポインタ（以下、ポインタということもある。）がルートノードの配列番号を指しているか否かを判定する。スタックポインタが指す配列番号がルートノード以外である場合は、ステップＳ６０５に進む。そして、ステップＳ６０５で、探索経路スタック３１０からスタックポインタの指す配列番号を取り出してから、スタックポインタの値を１減らす。

【００８１】

ステップＳ６０６で、ステップＳ６０５で取り出した配列番号から、その配列番号のノードがノード対のどちらの配列要素に格納されているかのノード位置を得る。例えばノード［０］については偶数番号の配列の配列要素に格納する等により、配列番号からノード位置を求めることができる。そして、ステップＳ６０７で、ステップＳ６０６で得たノード位置がノード［１］側であるか否かを判定する。ステップＳ６０７においてノード［１］側と判定された場合は、ステップＳ６０４に戻り、スタックポインタの指す配列番号のノードがノード［０］あるいはルートノードとなるまで、ステップＳ６０４からステップＳ６０７までの処理を繰り返す。

【００８２】

ステップＳ６０７においてノード［０］側と判定されると、ステップＳ６０８に進み、配列番号に１を加算し、そのノードと対をなすノード［１］の配列番号を得る。そして、ステップＳ６０９で、検索開始ノードにステップＳ６０８で得たノード［１］の配列番号を設定し、ステップＳ６１０で、検索開始ノードをルートノードとする部分木の最小のインデックスキーを求める処理を実行する。ステップＳ６１０の処理は、ステップＳ６０２と同様であり、図５に示す最小値検索処理が用いられる。

【００８３】

ステップＳ６１０で最小のインデックスキーを求めると、ステップＳ６０３に戻り、求めたインデックスキーを取り出し、以降ステップＳ６０４においてポインタがルートノードの配列番号を指していると判定されるまで、同様の処理を繰り返す。

【００８４】

このように、探索経路スタック３１０のポインタが指す配列番号を参照し、探索経路スタック３１０に格納されている配列番号のノード［０］と対をなすノード［１］について、そのノードを検索開始ノードとしてそのノード配下で最小のインデックスキーを求める。ステップＳ６０２で最小値が求められた段階においては、探索経路スタック３１０のポインタは、カップルドノードツリーの最小のインデックスキーを含むノードの配列番号を指している。続いて探索経路スタック３１０のポップ動作を行い、取り出した配列番号のノードのうちノード［１］を検索開始ノードとして該検索開始ノードをルートノードとする部分木のインデックスキーの最小値を求める処理を行い、探索経路スタック３１０のポップ動作の結果カップルドノードツリーのルートノードの配列番号が取り出されるまでポ

ップ動作と最小値検索処理を繰り返す。

【００８５】

探索経路スタック３１０には、先にステップＳ６０２においてルートノード配下のノードについて最小値を求める処理を行っていることにより、リンク経路についての配列番号が順次格納されている。このため、探索経路スタック３１０のポインタを１減らして新たにポインタが指す配列番号のノードのうち、ノード［０］について、そのノードと対をなすノード［１］を求め、ノード［１］のノード配下について順次最小値検索の処理を行っていくと、インデックスキーは昇順に取り出されることとなる。

【００８６】

先に最小値検索を変形して最大値検索が可能であることを述べたのと同様に、最小値検索に代えて最大値検索を行い、ノード［１］の判定をノード［０］の判定に置き換える等により、インデックスキーを降順に取り出すことが可能であることは、当業者に明らかである。

【００８７】

次に、図７Ａ～図７Ｄにより本出願人による出願である上記特願２００６－１８７８２７で提案されたカップルドノードツリーにおけるノード挿入処理を説明する。図７Ａ～図７Ｃが通常の挿入処理を説明するものであり、図７Ｄはルートノードの挿入処理を説明するものである。ルートノードの挿入処理と通常の挿入処理により、カップルドノードツリーが生成されることから、ノード挿入処理の説明はカップルドノードツリーの生成処理の説明でもある。

【００８８】

図７Ａは挿入処理の前段である検索処理の処理フローを示す図であり、図４に示した検索処理において、ルートノードを検索開始ノードとし、挿入キーを検索キーとしたものに相当する。

【００８９】

ステップＳ７０１において、検索開始ノードの配列番号を設定するエリアにルートノードの配列番号を設定し、ステップＳ７０２において、検索キーに挿入キーを設定する。

次にステップＳ７１０において、図４に示す検索処理を実行し検索結果のインデックスキーを得て、ステップＳ７１１に進む。

【００９０】

ステップＳ７１１において挿入キーとインデックスキーを比較し、等しければ挿入キーは既にカップルドノードツリーに存在するのであるから、挿入は失敗となり、処理を終了する。等しくなければ次の処理、図７ＢのステップＳ７１２以下の処理に進む。

【００９１】

図７Ｂは、挿入するノード対のための配列要素を準備する処理を説明する処理フロー図である。

ステップＳ７１２において、配列から空きのノード対を求め、そのノード対のうち代表ノードとなるべき配列要素の配列番号を取得する。

【００９２】

ステップＳ７１３に進み、挿入キーとステップＳ７１０で得たインデックスキーの大小を比較し、挿入キーが大きいときは値１を小さいときは値０のブール値を得る。

ステップＳ７１４に進み、ステップＳ７１２で得た代表ノードの配列番号にステップＳ７１３で得たブール値を加算した配列番号を得る。

【００９３】

ステップＳ７１５に進み、ステップＳ７１２で得た代表ノードの配列番号にステップＳ７１３で得たブール値の論理否定値を加算した配列番号を得る。

ステップＳ７１４で得た配列番号は、挿入キーをインデックスキーとして持つリーフノードが格納される配列要素の配列番号であり、ステップＳ７１５で得た配列番号は、そのリーフノードと対を成すブランチノードあるいはリーフノードが格納される配列要素のものである。

【００９４】
つまり、前段の検索処理で得られたリーフノードに格納されたインデックスキーと挿入キーの大小により、挿入されるノード対のうちどちらのノードに挿入キーを保持するリーフノードが格納されるかが決定される。
【００９５】
例えば図２Ｂのカップルドノードツリーに“０１１０１１”を挿入する場合、検索結果のインデックスキーはノード２１１ｄに格納された“０１１０１０”になる。挿入キー“０１１０１１”とノード２１１ｄに格納されたインデックスキー“０１１０１０”の大小比較によりブール値が求められ、今の例では挿入キーが大きいのでブール値１が得られ、挿入されるノード対の代表ノード番号に１を加えた配列要素に挿入キーを保持するリーフノードが格納される。一方、インデックスキー“０１１０１０”は、大小比較で得られたブール値を論理反転した値を代表ノード番号に加算した配列番号の配列要素に格納される。
【００９６】
その際、インデックスキー“０１１０１０”と挿入キー“０１１０１１”とは５ビット目で異なることから、ノード２１１ｄは、弁別ビット位置を５とし、代表ノード番号を挿入されたノード対の代表ノードの配列番号とするブランチノードとなる。
【００９７】
また図２Ｂのカップルドノードツリーに“０１１００１”を挿入しようとする場合も、検索結果のインデックスキーはノード２１１ｄに格納された“０１１０１０”になる。この場合には挿入キーが小さいのでブール値０が得られ、挿入されるノード対の代表ノード番号に０を加えた配列要素に挿入キーを保持するリーフノードが格納される。そして、インデックスキー“０１１０１０”と挿入キー“０１１００１”とは４ビット目で異なることから、ノード２１１ｄは、弁別ビット位置を４とし、代表ノード番号を挿入されたノード対の代表ノードの配列番号とするブランチノードとなる。次に図７Ｃのステップ７１６以下の処理に進む。
【００９８】
図７Ｃは図７Ｂで準備された配列にノードを格納するとともにその挿入位置を求め、既存のノードの内容を変更して挿入処理を完成させる処理フローを示す図である。
ステップＳ７１６～ステップＳ７２３までの処理は、挿入するノード対のカップルドノードツリー上の位置を求める処理であり、ステップＳ７２４以下の処理は各ノードにデータを設定して挿入処理を完成させる処理である。
【００９９】
ステップＳ７１６で、挿入キーとステップＳ７１０で得たインデックスキーのビット列比較を例えば排他的論理和で行い、差分ビット列を得る。
ステップＳ７１７に進み、ステップＳ７１６で得た差分ビット列から、上位０ビット目から見た最初の不一致ビットのビット位置を得る。この処理は、例えばプライオリティエンコーダを有するＣＰＵではそこに差分ビット列を入力し、不一致のビット位置を得ることができる。また、ソフト的にプライオリティエンコーダと同等の処理を行い最初の不一致ビットのビット位置を得ることも可能である。
【０１００】
次にステップＳ７１８に進み、探索経路スタックのスタックポインタがルートノードの配列番号を指しているか判定する。指していればステップＳ７２４に移行し、指していなければステップＳ７１９に進む。
【０１０１】
ステップＳ７１９において、探索経路スタックのスタックポインタを１つ戻してそこにスタックされている配列番号を取り出す。
ステップＳ７２０に進み、ステップＳ７１９で取り出した配列番号の配列要素を配列からノードとして読み出す。
【０１０２】

ステップＳ７２１に進み、ステップＳ７２０で読み出したノードから、弁別ビット位置を取り出す。

次にステップＳ７２２に進み、ステップＳ７２１で取り出した弁別ビット位置がステップＳ７１７で得たビット位置より上位の位置関係か判定する。ここで上位の位置関係とは、ビット列のより左側の位置、すなわちビット位置の値が小さい位置であることとする。

【０１０３】

ステップＳ７２２の判定結果が否定であれば、ステップＳ７１８に戻り、ステップＳ７１８での判定が肯定になるかステップＳ７２２での判定が肯定になるまで繰り返す。ステップＳ７２２での判定が肯定になると、ステップＳ７２３で経路探索スタックのスタックポインタを1つ進め、ステップＳ７２４以下の処理に移行する。

【０１０４】

上記ステップＳ７１６～ステップＳ７２３で説明した処理は、挿入するノード対の挿入位置を決定するために、挿入するインデックスキーと検索により取得されたインデックスキーの間でビット列比較を行い、ビット列比較で異なるビット値となる先頭の（最上位の）ビット位置と探索経路スタックに格納されているブランチノードの弁別ビット位置との相対的位置関係を調べ、弁別ビット位置が上位となるブランチノードの次のブランチノードのリンク先を挿入するノード対の挿入位置とするものである。

【０１０５】

例えば図２Ｂのカップルドノードツリーに“１１１０００”を挿入するとき、検索結果のインデックスキーはノード２１０ｈに格納された“１０１０１１”になる。挿入キー“１１１０００”とノード２１０ｈに格納されたインデックスキー“１０１０１１”のビット列比較により異なるビット値となる最上位のビット位置１が得られる。得られたビット位置１と経路探索スタックに積まれた配列番号の配列要素に格納されたブランチノードの弁別ビット位置との位置関係を弁別ビット位置が上位になるまでを順次経路探索スタック逆にたどると、ルートノード２１０ａに至る。そこで探索経路スタックのポインタを１つ進め、ノード２１１ｂの配列番号を得る。挿入キー“１１１０００”はノード２１１ｂのリンク先に挿入される。

【０１０６】

また、経路探索スタック逆にたどりルートノードに至っても、ルートノードの弁別ビット位置が、先に求めたビット列比較で異なるビット値となる最上位のビット位置より上位のビット位置でないということは、そのカップルドノードツリーのインデックスキーの上位ビットで、ルートノードの弁別ビット位置より上位のビットの値は全て等しい場合である。そして、挿入するインデックスキーにおいて、初めてルートノードの弁別ビット位置より上位のビットの値に異なるビット値のものがあるということである。したがって、挿入するノード対はルートノードの直接のリンク先となり、ルートノードの弁別ビット位置は、既存のインデックスキーと異なる値である挿入キーの最上位ビットの位置に変わる。

【０１０７】

次に、ステップＳ７２４以下の各ノードにデータを設定して挿入処理を完成させる処理について説明する。

ステップＳ７２４では探索経路スタックからスタックポインタの指す配列番号を取り出す。

【０１０８】

ステップＳ７２５において、ステップＳ７１４で得た配列番号の指す配列要素のノード種別に１（リーフノード)を、インデックスキーに挿入キーを書き込む。

ステップＳ７２６に進み、配列からステップＳ７２４で得た配列番号の配列要素を読み出す。

【０１０９】

次にステップＳ７２７において、ステップＳ７１５で得た配列番号の配列要素にステップＳ７２６で読み出した内容を書き込む。

最後にステップＳ７２８において、ステップＳ７２４で得た配列番号の指す配列要素の

ノード種別に０(ブランチノード）を、弁別ビット位置にステップＳ７１７で得たビット位置を、代表ノード番号にステップＳ７１２で得た配列番号を書き込み、処理を終了する。

【０１１０】

上述の図２Ｂのカップルドノードツリーに“１１１０００”を挿入する例では、取得された空ノード対のノード[０]にノード２１１ｂの内容を書き込み（ステップＳ７２７）、ノード[１]を挿入キー“１１１０００”を保持するリーフノードとする（ステップＳ７２５）。そして、ノード２１１ｂの弁別ビット位置にビット列比較により異なるビット値となる最上位のビット位置１を格納し、代表ノード番号には取得されたノード対の代表ノードが格納される配列要素の配列番号が格納される（ステップＳ７２８）。

【０１１１】

図７Ｄは、本出願人による出願である上記特願２００６－１８７８２７で提案されたルートノードの挿入処理を含むインデックスキーを追加する場合のノード挿入処理全体を説明する処理フロー図である。

【０１１２】

ステップＳ１０１において、取得することを求められたカップルドノードツリーのルートノードの配列番号が登録済みであるか判定される。登録済みであれば、図７Ａ～図７Ｃを用いて説明した通常の挿入処理が行われる。

【０１１３】

ステップＳ１０１での判定が登録済みでなければ、まったく新しいカップルドノードツリーの登録、生成が始まることになる。

まず、ステップＳ１０２において、配列から空きのノード対を求め、そのノード対のうち代表ノードとなるべき配列要素の配列番号を取得する。次にステップＳ１０３において、ステップＳ１０２で得た配列番号に０を加えた配列番号を求める。（実際には、ステップＳ１０２で取得した配列番号に等しい。)。さらにステップＳ１０４において、ステップＳ１０３で得た配列番号の配列要素に、挿入するルートノードのノード種別に１（リーフノード)とインデックスキーに挿入キーを書き込み、ステップＳ１０５で、ステップＳ１０２で取得したルートノードの配列番号を登録して処理を終了する。

【０１１４】

先にも述べたように、インデックスキーの集合があるとき、そこから順次インデックスキーを取り出し、図７Ｄ及び図７Ａ～図７Ｃの処理を繰り返すことにより、インデックスキーの集合に対応した本発明のカップルドノードツリーを構築することができることは明らかである。

【０１１５】

次に図８Ａ、図８Ｂを参照して、本出願人による出願である上記特願２００６－１８７８２７で提案されたカップルドノードツリーに係るインデックスキーの集合から、特定のインデックスキーを削除する処理フローを説明する。

【０１１６】

図８Ａは、削除処理の前段である検索処理の処理フローを示す図であり、図４に示した検索処理において、ルートノードを検索開始ノードとし、挿入キーを検索キーとしたものに相当する。

【０１１７】

ステップＳ８０１において、検索開始ノードの配列番号を設定するエリアにルートノードの配列番号を設定し、ステップＳ８０２において、検索キーに削除キーを設定する。

次にステップＳ８１０において、図４に示す検索処理を実行し検索結果のインデックスキーを得て、ステップＳ８１１に進む。

【０１１８】

図８ＡのステップＳ８１１において削除キーとインデックスキーを比較し、等しくなければければ削除するインデックスキーはカップルドノードツリーに存在しないのであるから、削除は失敗となり、処理を終了する。等しければ次の処理、図８Ｂのステップ８１

２以下の処理に進む。

【０１１９】

図８Ｂは、削除処理の後段の処理フローを説明する図である。

まず、ステップＳ８１２で探索経路スタックに２つ以上の配列番号が格納されているか判定する。２つ以上の配列番号が格納されていないということは、言い換えれば１つだけで、その配列番号はルートノードの格納された配列要素のものである。その場合はステップＳ８１８に移行し、ステップＳ８０１で得たルートノードの配列番号に係るノード対を削除する。次にステップＳ８１９に進み、登録されていたルートノードの配列番号を削除して処理を終了する。

【０１２０】

ステップＳ８１２において探索経路スタックに２つ以上の配列番号が格納されていると判定されたときはステップＳ８１３に進み、ステップＳ８０８で得た代表ノード番号にステップＳ８０７で得たビット値を反転した値を加算した配列番号を得る。この処理は、削除対象のインデックスキーが格納されたリーフノードと対をなすノードの配置された配列番号を求めるものである。

【０１２１】

次にステップＳ８１４において、ステップＳ８１３で得た配列番号の配列要素の内容を読み出し、ステップＳ８１５において探索経路スタックのスタックポインタを１つ戻して配列番号を取り出す。

【０１２２】

次にステップＳ８１６に進み、ステップＳ８１４で読み出した配列要素の内容をステップＳ８１５で得た配列番号の配列要素に上書きする。この処理は、削除対象のインデックスキーが格納されたリーフノードへのリンク元であるブランチノードを上記リーフノードと対をなすノードに置き換えるものである。

【０１２３】

最後にステップＳ８１７においてステップＳ８０８で得た代表ノード番号に係るノード対を削除して処理を終了する。

以上、カップルドノードツリーに関する本発明の前提となる技術を説明したが、必要とあれば、上述の特許出願の明細書及び図面の記載を参照されたい。

【０１２４】

次に、図９～図１３を参照して、本発明に係る差分データを作成する処理について説明する。

図９は、本発明の一実施形態における旧データと新データにより差分データを作成する処理の概要を説明する図である。図９に示す例では、旧データ３２１はキー３２１ａ“０１００１０”、３２１ｂ“０１００１１”、３２１ｃ“１００１１”、３２１ｄ“１０１０１１”及び３２１ｅ“１０１１００”で構成され、新データ３２２はキー３２２ａ“０１００１０”、３２２ｂ“０１０１１０”、３２２ｃ“１０００１０”、３２２ｄ“１０００１１”及び３２２ｅ“１０１１００”で構成されている。

【０１２５】

また、差分データ３２０の各更新データ３８０ａ～３８０ｄは、更新種別３８１と更新キー３８２の項目で構成されている。更新データ３８０ａ～３８０ｄの各更新キー３８２は、図に示すようにキー３２１ｄ、３２１ｂ、３２２ｃ、３２２ｂである。

【０１２６】

旧データであるキー３２１ａ～３２１ｅは、図９の（Ａ）に示す挿入処理により、更新前の差分ツリー３０９－１に挿入される。なお、図９の（Ａ）に示すＳ１００１は、後記図１０に示す差分データ作成の処理フローのステップＳ１００１での処理に対応することを示すものである。以下の図９の（Ｂ）～（Ｅ）についても同様である。

【０１２７】

次に、図９の（Ｂ）に示すように、新データのキー３２２ａ～３２２ｅによる削除処理が図１０に示すフローのステップＳ１００５で行われる。削除処理の結果、更新後の差分

ツリー３０９－２には旧データのキー３２１ｄと３２１ｂが削除されずに残っている。

【０１２８】

旧データに同一のデータが存在せず、図９の（Ｃ）に示すように図１０のフローのステップＳ１００６で削除に失敗したと判定された新データのキー３２２ｃと３２２ｂは、図９の（Ｄ）に示すように、図１０のステップＳ１００７で挿入キーとして差分データ格納領域３２０に格納される。図９に示すように、挿入キーの値は更新キー３８２に格納され、更新種別３８１には“ｉ”が格納され、更新キーに格納された３２２ｃ、３２２ｂが挿入キーであることが示されている。

【０１２９】

一方、更新後の差分ツリー３０９－２に残った旧データのキー３２１ｄと３２１ｂは、図９の（Ｅ）に示すように、図１０のステップＳ１００９で差分ツリー３０９－２から取り出され、削除キーとして差分データ格納領域３２０に格納される。図９に示すように、削除キーの値は更新キー３８２に格納され、更新種別３８１には“ｄ”が格納され、更新キーに格納された３２１ｄ、３２１ｂが削除キーであることが示されている。

【０１３０】

なお、上述の説明では、旧データから差分ツリー３０９－１を生成し、新データにより削除処理を行うものとした。しかし、旧データと新データの差分を作成するのであるから、逆に新データから差分ツリーを生成し、旧データにより削除処理を行って差分データを作成できることは明らかである。その場合には、更新後の差分ツリーに残った新データが挿入キーとなり、削除に失敗した旧データが削除キーとなることも当業者には明らかである。

【０１３１】

図１０は旧データと新データにより差分データを作成する処理フローを説明する図である。以下、図１０を参照して新旧データによる差分データ作成の処理フローを説明する。

図１０に示すように、ステップＳ１００１において、旧データの全てのキーを差分ツリーに挿入する。この処理の詳細は、後に図１１を参照して行う。なお、旧データがすでにカップルドノードツリーに格納されている、すなわちデータベースのインデックスがカップルドノードツリーのデータ構造を有しているものであれば、ステップ１００１に代えてその登録されたルートノードの配列番号を取得するだけでよい。また、カップルドノードツリーを配列に格納しない場合であれば、ルートノードの位置を示す位置情報を取得すればよい。

【０１３２】

次に、ステップＳ１００２において、新データを設定する。この新データの設定は、新データの供給元から供給された新データを、例えば図３Ｂ及び図９に示す新データ格納領域３２２に格納することによって行われるが、供給元から供給された新データを格納した記憶媒体から直接キーを読み出すことも可能である。

【０１３３】

次にステップＳ１００３に進み、全ての新データが処理済であるか判定する。全ての新データが処理済であればステップＳ１００８に移行し、処理済でなければステップＳ１００４に進む。

【０１３４】

ステップＳ１００４では、ステップＳ１００２で設定された新データからキーを取り出す。次のステップＳ１００５で、該取り出したキーを削除キーとして差分ツリーから削除する。この削除処理は、図８Ａ及び図８Ｂに示すものである。

【０１３５】

次にステップＳ１００６において、ステップＳ１００５での削除が成功したかを判定する。削除に成功すれば、ステップＳ１００３に戻り、失敗すればステップＳ１００７において、削除キーを挿入キーとして差分データを作成し、ステップＳ１００３に戻る。ステップＳ１００７の処理の詳細は、後に図１２を参照して説明する。

【０１３６】

全ての新データが処理済となってステップＳ１００３から分岐したステップＳ１００８においては、差分ツリーが登録されているかを判定する。登録されていなければ処理を終了し、登録されていればステップＳ１００９で差分ツリーの全てのキーを取り出し、削除キーとして差分データを作成して処理を終了する。ステップＳ１００９の詳細は、後に図１３を参照して説明する。

【０１３７】

なお、図１０で説明してフローはあくまで１つの例であり、先に図９による説明で述べたように新データにより差分ツリーを生成する方法や処理の順番を一部替える等の種々の変形が可能であることは当業者にとって自明である。

【０１３８】

図１１は、図１０に示す旧データと新データにより差分データを作成する処理フローのステップＳ１００１で実行される、旧データによりカップルドノードツリー（差分ツリー）を作成する処理フローを説明する図である。

【０１３９】

最初にステップＳ１１０１において、旧データを設定する。この旧データの設定は、旧データの供給元から供給された旧データを保管しておき、例えば図３Ｂ及び図９に示す旧データ格納領域３２１に格納することによって行われるが、供給元から供給された旧データを格納した記憶媒体から直接キーを読み出すことも可能である。

【０１４０】

次にステップＳ１１０２で旧データから全てのキーを取り出し済みか判定し、取り出し済みであれば処理を終了する。

旧データから全てのキーを取り出し済みでなければ、ステップＳ１１０３において旧データからキーを取り出し、ステップＳ１１０４に進んで取り出したキーを挿入キーとして、図７Ａ～図７Ｄに示す処理により、差分ツリーにキーを挿入する。

【０１４１】

ステップＳ１１０４の挿入処理の後、ステップＳ１１０２の判定処理に戻るループ処理を、旧データから全てのキーが取り出し済みとなるまで繰り返す。

図１２は、差分データに更新データを格納する処理フローを説明する図である。図１２に示す処理フローは、図１０に示す旧データと新データにより差分データを作成する処理フローのステップＳ１００７で実行される、削除キーを挿入キーとして差分データを作成する処理に適用可能なものである。

【０１４２】

また、図１２に示す処理フローは、図１０に示す処理フローのステップＳ１００９で実行される差分データ作成処理にも適用可能なものとなっており、データの更新が挿入であるか削除であるかの情報と、差分データの更新キーとなるキーの値が与えられるものとしている。また、ワークエリアとして、図示しない更新種別と更新キーからなる更新データ設定エリアを使用するものとする。

【０１４３】

図に示すようにステップＳ１２０１において、更新データ設定エリアの更新種別に、データの更新が挿入であるか削除であるかの情報に基づいて、更新種別を設定する。図９に示す例では、呼出元がステップＳ１００７であり、データ更新が挿入であれば、更新種別に“ｉ”が設定される。

【０１４４】

次にステップＳ１２０２において、更新データ設定エリアの更新キーに、更新キーとなるキーの値を設定する。

次にステップＳ１２０３において、ステップＳ１２０１とステップＳ１２０２で更新データ設定エリアに設定された更新データを差分データ格納領域に格納して処理を終了する。

【０１４５】

図１３は、図１０に示す旧データと新データにより差分データを作成する処理フローの

ステップＳ１００９で実行される、差分ツリーの全てのキーを取り出し、削除キーとして差分データを作成する処理フローを説明する図である。

【０１４６】

ステップＳ１３０１において、検索開始ノードの配列番号として、差分ツリーのルートノードの配列番号を設定する。次に検索開始ノードより図５に示す最小値検索を行い、インデックスキーの最小値を求める。

【０１４７】

次にステップＳ１３０３に進み、ステップＳ１３０２で求めた最小値を削除キーとして差分データを作成する。このステップ１３０３の処理は、更新種別を削除とし、更新キーの値を直前の最小値検索で求めた最小値として図１２に示す差分データに更新データを格納する処理により実現される。

【０１４８】

以下、ステップＳ１３０３、ステップＳ１３０４～ステップＳ１３１０に示すように、差分ツリーから最小値検索を繰り返して昇順でインデックスキーを取り出し、取り出した最小値を削除キーとして差分データを作成するステップＳ１３０３を繰り返す。

【０１４９】

したがって、ステップＳ１３０４～ステップＳ１３１０の処理は、図６に示すインデックスキーを昇順に取り出す処理フローのステップＳ６０４～ステップＳ６１０の処理と同様であるので、説明を省略する。

【０１５０】

なお、図１０に示すステップＳ１００９で実行される処理では、差分ツリーに残された全てのキーを取り出せればよいのであるから、図１３に示す昇順取り出しに限らず、先に述べた降順取り出しを採用することも可能である。

【０１５１】

以上の図１０～図１３を参照して説明した処理により、旧データと新データにより差分データが作成される。

次に、図１４、図１５を参照して、本発明に係る差分データにより旧データを新データに更新する処理について説明する。

【０１５２】

図１４は、本発明の一実施形態における差分データにより旧データを新データに更新する処理の概要を説明する図である。図１４に示す例は、図９において例示した旧データと新データにより作成された差分データにより、図３Ｂに例示する差分データ更新装置４００ａにおいて旧データを新データに更新するものとしている。

【０１５３】

したがって、インデックス格納領域４２１ａには、図９に示す旧データと同様に、旧データのキー３２１ａ“０１００１０”、３２１ｂ“０１００１１”、３２１ｃ“１０００１１”、３２１ｄ“１０１０１１”及び３２１ｅ“１０１１００”が格納されている。また同様に、差分データ格納領域４２０ａにも、図９に示す差分データ格納領域に格納された更新データと同一のデータが格納されている。

【０１５４】

旧データであるキー３２１ａ～３２１ｅは、図１４の（Ａ）に示す挿入処理により、更新前の更新ツリー４０９ａ－１に挿入される。なお、図１４の（Ａ）に示すＳ１５０１は、後記図１５に示す差分データにより旧データを新データに更新する処理フローのステップＳ１５０１での処理に対応することを示すものである。以下の図１４の（Ｂ）～（Ｄ）についても同様である。

【０１５５】

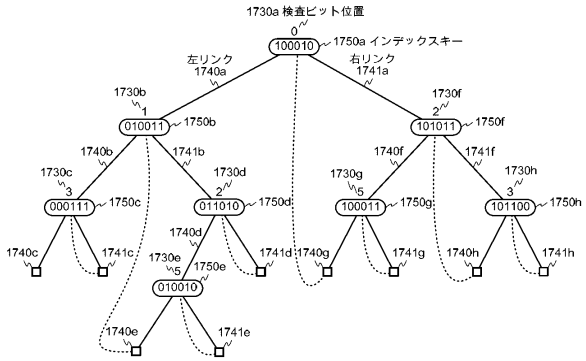
次に、図１４の（Ｂ）に示すように、差分データ格納領域４２０ａに格納された更新データのうち、更新種別２８１の値が“ｄ”である更新データ３８０ａ、３８０ｂの更新キー２８２であるキー３２１ｂとキー３２１ｄによる削除処理が図１５に示すフローのステップＳ１５０８で行われる。

【０１５６】
さらに、図１４の（Ｃ）に示すように、差分データ格納領域４２０ａに格納された差分データのうち、更新種別２８１の値が“ｉ”である更新データ３８０ｃ、３８０ｄの更新キー２８２であるキー３２２ｂとキー３２２ｃよる挿入処理が図１５に示すフローのステップＳ１５０７で行われる。
【０１５７】
その結果、更新前の更新ツリー４０９ａ－１は更新後の更新ツリー４０９ａ－２に更新される。更新ツリー４０９ａ－２には、旧データに含まれていたキー３２１ａ、３２１ｃ及び３２１ｅと挿入されたキー３２２ｂと３２２ｃが含まれている。
【０１５８】
次に、図１４の（Ｄ）に示すように、更新後の更新ツリー４０９ａ－２からキーを取り出して新データ格納領域４２２ａに格納する処理が図１５に示すステップＳ１５０９で行われ、旧データが新データに更新される。新データ格納領域４２２ａには、キー３２１ａ“０１００１０”、３２２ｂ“０１０１１０”、３２２ｃ“１０００１０”、３２１ｃ“１０００１１”及び３２１ｅ“１０１１００”が格納されている。なお、新データ格納領域４２２ａはインデックス格納領域４２１ａと同一の領域とし、旧データに新データを上書きすることができる。
【０１５９】
なお、上述の説明では、更新ツリー４０９ａ－１に対して削除処理を実行した後に挿入処理を行うものとしたが、どのような順番で削除挿入を実行しても結果は同一である。
また、差分データ４２０ａのうち、挿入キーのみ取り出して旧データ４２１ａとともに更新前の更新ツリー４０９ａ－１を作成し、その後削除処理のみによって更新後の更新ツリー４０９ａ－２を作成する等種々の変形が可能である。
【０１６０】
図１５は、差分データにより旧データを新データに更新する処理フローを説明する図である。
図１５に示すように、ステップＳ１５０１において、旧データの全てのキーを更新ツリーに挿入する。この処理は、図１１を参照して説明した旧データによりカップルドノードツリーを作成する処理により、実現される。先に述べたのと同様に、旧データがすでにカップルドノードツリーに格納されている、すなわちデータベースのインデックスがカップルドノードツリーのデータ構造を有しているものであれば、ステップ１５０１に代えてその登録されたルートノードの配列番号あるいはルートノードの位置を示す位置情報を取得するだけでよい。さらにその場合、データベースのインデックスがカップルドノードツリーのデータ構造を有しているのであるから、後記ステップＳ１５０９の処理が不要になることも明らかである。
【０１６１】
次に、ステップＳ１５０２において、差分データを設定する。この差分データの設定は、差分データ作成装置により差分データを作成した供給元から供給された差分データを、例えば図３Ｂ及び図１４に示す差分データ格納領域４２０ａ～４２０ｘに格納することによって行われるが、供給元から供給された新データを格納した記憶媒体から直接キーを読み出すことも可能である。
【０１６２】
次にステップＳ１５０３に進み、全ての更新データが処理済であるか判定する。全ての更新データが処理済であればステップＳ１５０９に移行し、処理済でなければステップＳ１５０４に進む。
【０１６３】
ステップＳ１５０４では、ステップＳ１５０２で設定された差分データの更新データを取り出し、そこから更新キーを取り出す。
次のステップＳ１５０５で、更新データから更新種別を取り出し、次にステップＳ１５０６において、ステップＳ１５０５で取り出した更新種別を判定する。

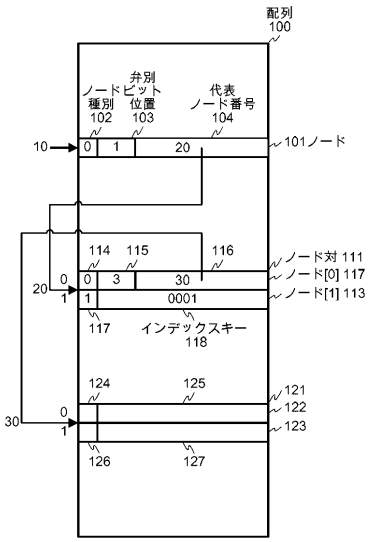
【０１６４】
更新種別が削除であればステップＳ１５０７に進み、挿入であればステップＳ１５０８に進む。
ステップＳ１５０７では、図８Ａ、図８Ｂに示す削除処理により、更新キーを削除キーとして更新ツリーより削除して、ステップＳ１５０３に戻る。
【０１６５】
ステップＳ１５０８では、図７Ａ～図７Ｄに示す挿入処理により、更新キーを挿入キーとして更新ツリーに挿入して、ステップＳ１５０３に戻る。
全ての更新データが処理済となってステップＳ１５０３から分岐したステップＳ１５０９においては、図６に示す昇順取り出し処理により、更新ツリーの全てのキーを新データとして取り出し、処理を終了する。
【０１６６】
上述の説明では、更新データの更新種別により削除処理と挿入処理を切り分けたが、図１０に示す差分データ作成処理において、差分データファイルを削除キーのファイルと挿入キーのファイルに分割して作成すれば、それぞれのファイルごとに削除処理と挿入処理を実行することになる。また、図１４の説明で述べたように、旧データと挿入キーのファイルの全てのキーを更新ツリーに挿入する処理を行い、削除キーのファイルに格納された削除キーにより更新ツリーの削除処理を実行して旧データを新データに更新することも可能である。
【０１６７】
以上詳細に説明したところから明らかなとおり、本発明によればカップルドノードツリーを活用することにより、データベースのインデックスを旧データのものから新データに効率よく更新することができる。
【０１６８】
また、本発明の実施の形態は上記に限ることなく種々の変形が可能であることは当業者に明らかである。さらに、本発明のインデックスキーの更新方法を、コンピュータに実行させるプログラムによりコンピュータで実行可能なことは明らかである。
【０１６９】
したがって、上記プログラム、及びプログラムを記憶したコンピュータ読み取り可能な記憶媒体は、本発明の実施の形態に含まれる。
【図面の簡単な説明】
【０１７０】
【図１】従来の検索で用いられるパトリシアツリーの一例を示す図である。
【図２Ａ】配列に格納されたカップルドノードツリーの構成例を説明する図である。
【図２Ｂ】カップルドノードツリーのツリー構造を概念的に示す図である。
【図３Ａ】本発明の原理を説明する図である。
【図３Ｂ】本発明を実施するためのハードウェア構成例を説明する図である。
【図３Ｃ】差分データ作成装置の機能ブロック構成を説明する図である。
【図３Ｄ】差分データ更新装置の機能ブロック構成を説明する図である。
【図４】ビット列検索の基本動作を示したフローチャートである。
【図５】カップルドノードツリーに格納されたインデックスキーの最小値を求める処理を示したフローチャートである。
【図６】カップルドノードツリーに格納されたインデックスキーを昇順に取り出す処理を示したフローチャートである。
【図７Ａ】挿入処理の前段である検索処理の処理フローを示す図である。
【図７Ｂ】挿入するノード対のための配列要素を準備する処理を説明する処理フロー図である。
【図７Ｃ】ノード対を挿入する位置を求め、ノード対の各ノードの内容を書き込んで挿入処理を完成させる処理フローを示す図である。
【図７Ｄ】ルートノードの挿入処理を含むインデックスキーを追加する場合のノード挿入

処理全体を説明する処理フロー図である。
【図８Ａ】削除処理の前段である検索処理の処理フローを示す図である。
【図８Ｂ】削除処理の後段の処理フローを説明する図である。
【図９】新旧データによる差分データ作成処理の概要を説明する図である。
【図１０】新旧データによる差分データ作成の処理フローを説明する図である。
【図１１】旧データによりカップルドノードツリーを作成する処理フローを説明する図である。
【図１２】差分データに更新データを格納する処理フローを説明する図である。
【図１３】差分ツリーに残ったキーを削除キーとして差分データを作成する処理フローを説明する図である。
【図１４】差分データにより旧データを新データに更新する処理の概要を説明する図である。
【図１５】差分データにより旧データを新データに更新する処理フローを説明する図である。
【符号の説明】
【０１７１】
１０、２０、３０　配列番号
１００　配列
１０１　ノード
１０２　ノード種別
１０３　弁別ビット位置
１０４　代表ノード番号
１１１　ノード対
１１２　ノード［０］、代表ノード
１１３　ノード［１］、代表ノードと対をなすノード
１１８　インデックスキー
３００　差分データ作成装置
３０１　データ処理装置
３０２　中央処理装置
３０３　キャッシュメモリ
３０４　バス
３０５　主記憶装置
３０６　外部記憶装置
３０７　通信装置
３０９　配列
３１０　探索経路スタック
３２０　差分データ格納領域
３２１　旧データ格納領域
３２２　新データ格納領域
３３０　差分ツリー取得手段
３３１　差分データ作成手段
４００　差分データ更新装置
４２０　差分データ格納領域
４２１　インデックス格納領域
４２２　更新ツリー取得手段
４２３　新旧インデックスキー更新手段

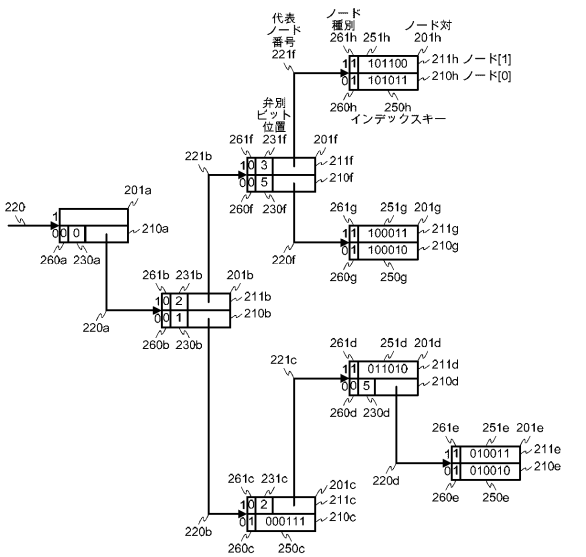
【図１】



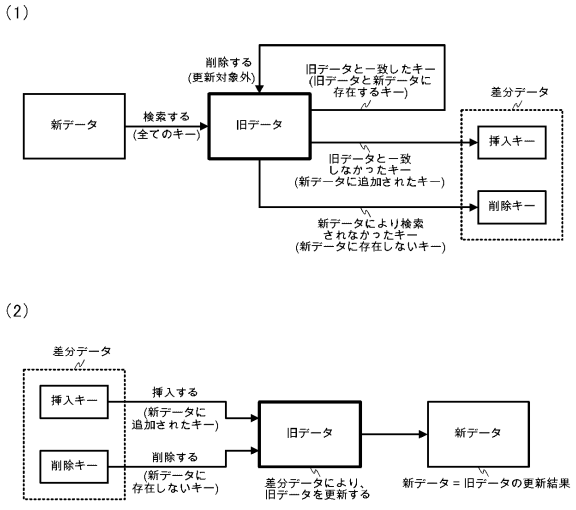
【図２Ａ】



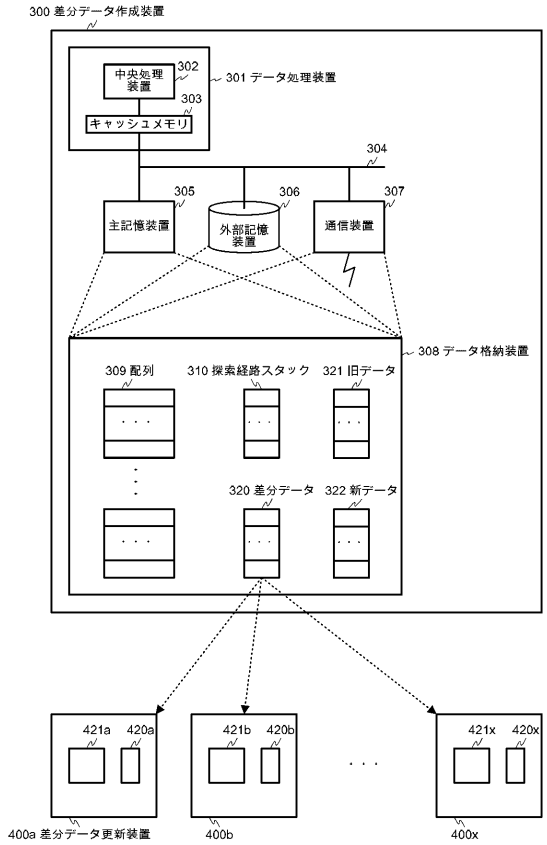
【図２Ｂ】



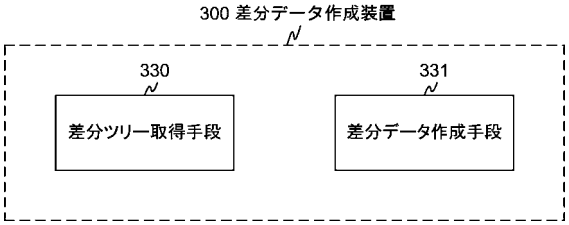
【図３Ａ】



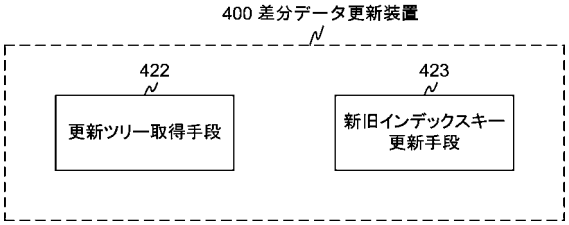
【図３Ｂ】



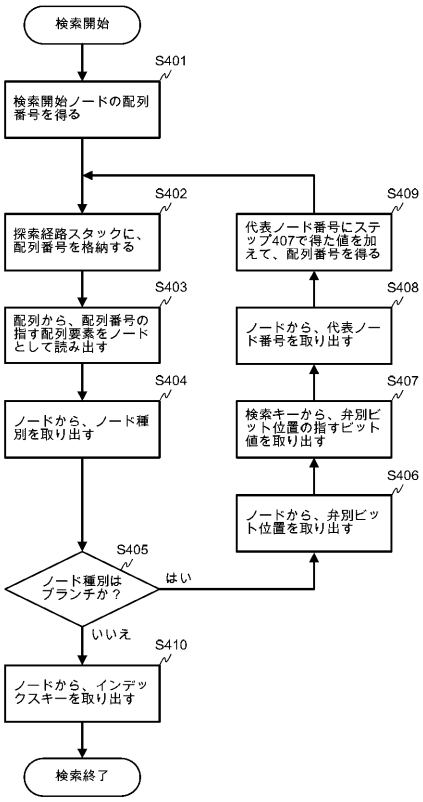
【図３Ｃ】



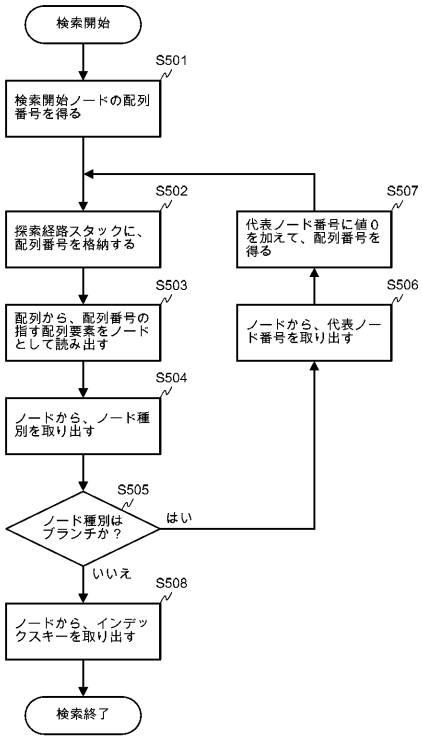
【図３Ｄ】



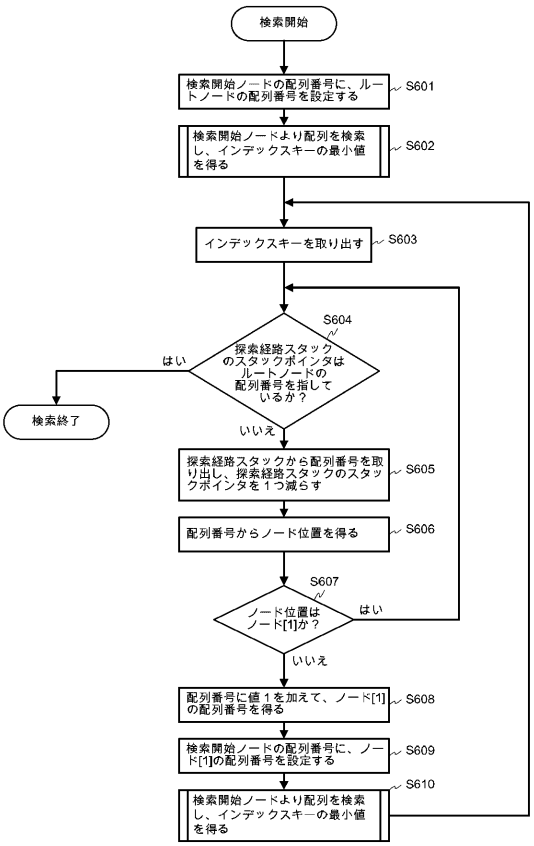
【図４】



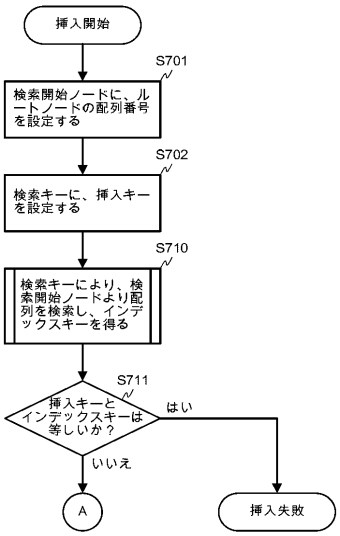
【図５】



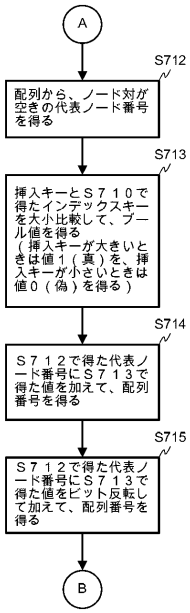
【図６】



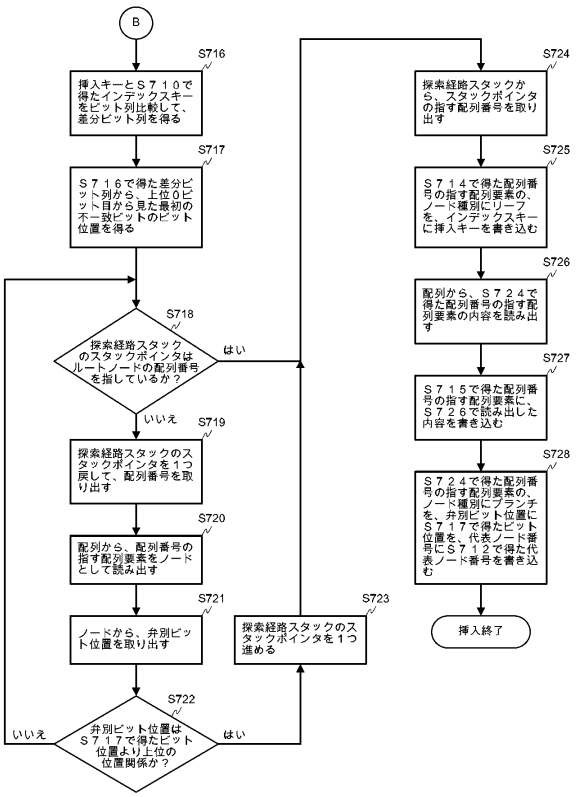
【図７Ａ】



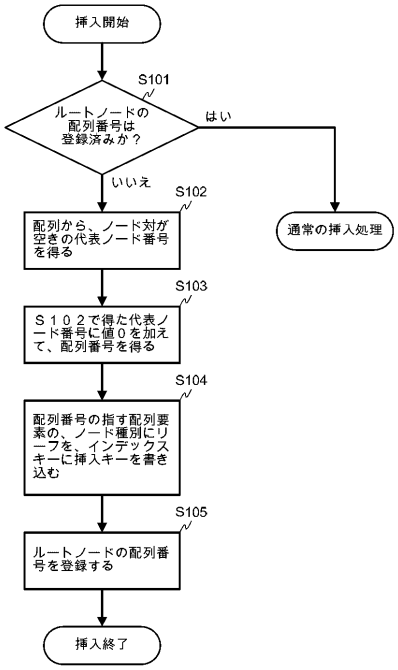
【図７Ｂ】



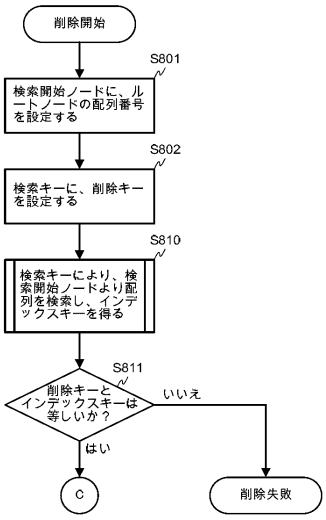
【図７Ｃ】



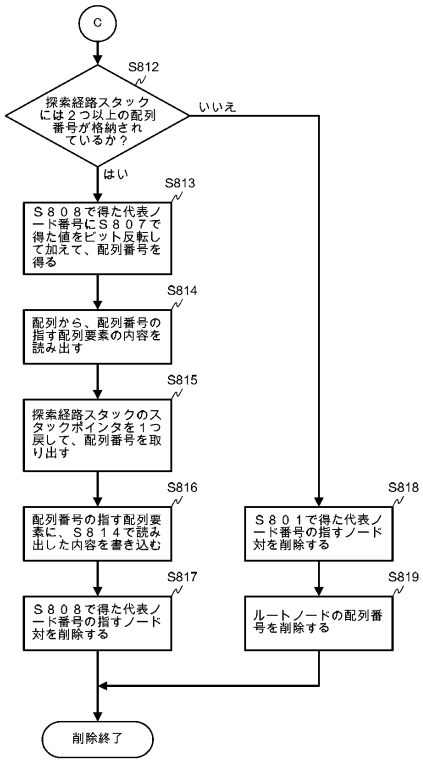
【図７Ｄ】



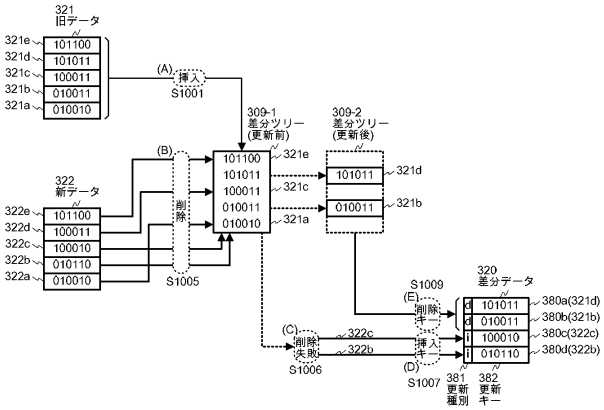
【図８Ａ】



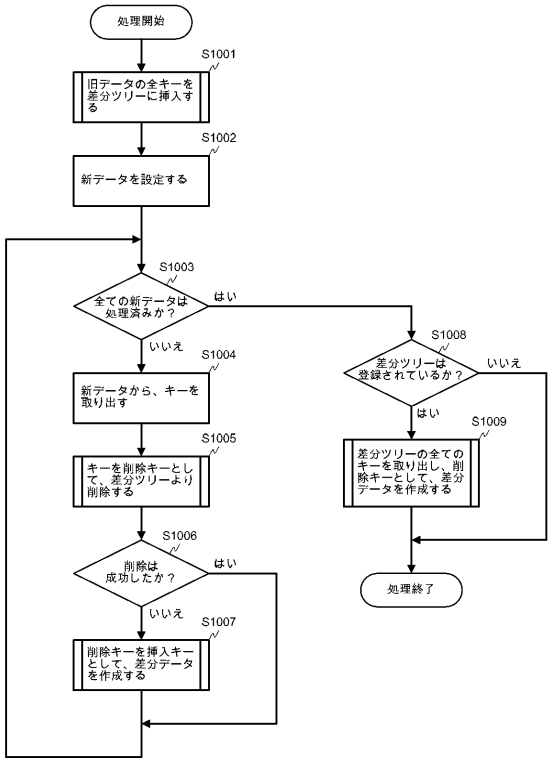
【図８Ｂ】



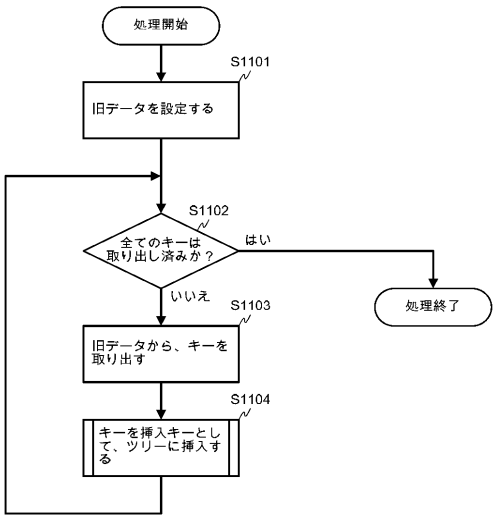
【図９】



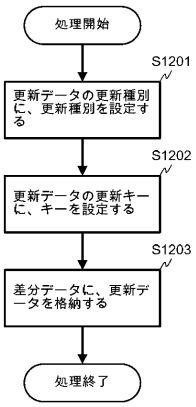
【図１０】



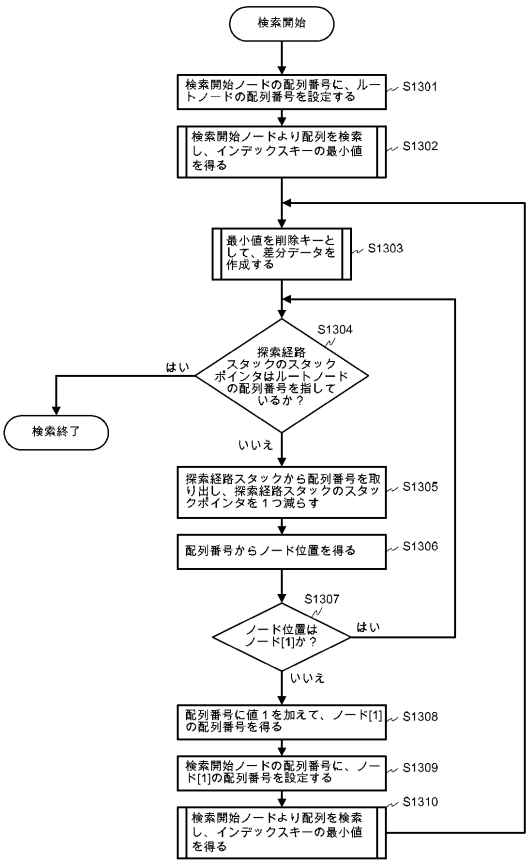
【図１１】



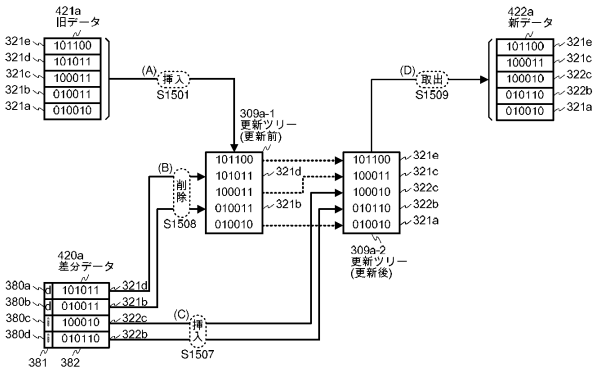
【図１２】



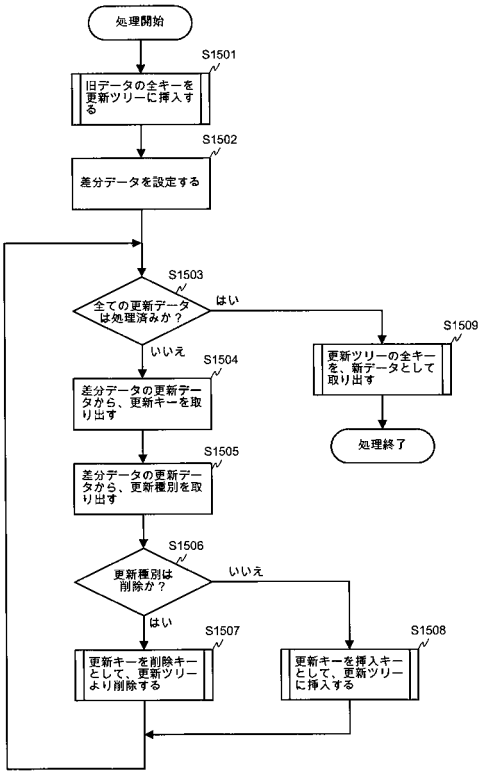
【図１３】



【図１４】


421a
旧データ
101100
101011
100011
010011
010010
321e
321d
321c
321b
321a
(A)
挿入
S1501
309a-1
更新ツリー
(更新前)
(B)
削除
S1508
420a
差分データ
380a
380b
380c
380d
101011
010011
100010
010110
321d
321b
322c
322b
381
382
(C)
挿入
S1507
309a-2
更新ツリー
(更新後)
(D)
取出
S1509
422a
新データ

【図１５】


処理開始
S1501
旧データの全キーを更新ツリーに挿入する
S1502
差分データを設定する
S1503
全ての更新データは処理済みか？
はい
いいえ
S1504
差分データの更新データから、更新キーを取り出す
S1505
差分データの更新データから、更新種別を取り出す
S1506
更新種別は削除か？
いいえ
はい
S1507
更新キーを削除キーとして、更新ツリーより削除する
S1508
更新キーを挿入キーとして、更新ツリーに挿入する
S1509
更新ツリーの全キーを、新データとして取り出す
処理終了

フロントページの続き

(56)参考文献 特開２００１－３５７０７０（ＪＰ，Ａ）
特開２００８－０１５８７２（ＪＰ，Ａ）
特開２００８－１１２２４０（ＪＰ，Ａ）
特開２００８－１５９０２５（ＪＰ，Ａ）
特開２００８－１８１２６０（ＪＰ，Ａ）
特開２００８－２６９１９７（ＪＰ，Ａ）
矢崎 茂明，図でわかる！プログラミングの１０大基礎知識，日経ソフトウエア ，日本，日経ＢＰ社，２００３年 1月２４日，第6巻第2号，p.44～p.45
後藤 大地，さわって学ぶデータ構造、つかって身につくアルゴリズム こちらＪａｖａ ＡＰＩ研究所！，ＪＡＶＡ ＰＲＥＳＳ，日本，（株）技術評論社，２００４年 2月１５日，第34巻，p.202～p.210

(58)調査した分野(Int.Cl.，ＤＢ名)
Ｇ０６Ｆ １２／００
Ｇ０６Ｆ １７／３０
ＪＳＴＰｌｕｓ（ＪＤｒｅａｍＩＩ）