データベース構築論1

3章「データ構造」

(配列・つなぎ・抽象データ型・木・ハッシュ)

中島康彦

§3. 1 データの構造

データの本質的な構造を反映できる構造.

計算機が処理しやすい構造.

§3. 2 構造のあるデータ型

構造体 structure

- 複数の型の組合せ (解釈は1通り)

共用体 union

- 複数の型の組合せ (解釈は複数通り)
- 特定の領域の値により解釈が決まる

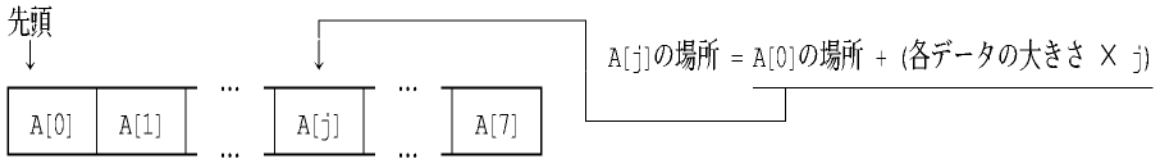
(a)構造体　　(b)共用体

§3. 3 他のデータを参照するための型

参照/ポインタ reference/pointer

§3. 4 ならび(リスト) … 配列 … 一次元配列

一次元配列 … 要素数が有限の一次元構造



要素番号から要素の位置を簡単に求められる.
予め決められた総数を越えるデータを格納できない.
途中の要素に対して追加/削除を行う場合, 後続の全要素に影響がおよぶ.

§3. 4 (続き)

- C言語の場合

```
/* 宣言時に総数が決まる */
int i;
int A[1000];

/* 使える要素は A[0]～A[999] */
for (i=0; i<1000; i++) {
  A[i] = xxxx;
}
```

§3. 4（続き）

- FORTRAN言語の場合

```
C 宣言時に総数が決まる
      INTEGER I;
      INTEGER A(1000);

C 使える要素は A(1)～A(1000)
      DO 10 I=1,1000
        A(I) = xxxx;
 10 CONTINUE
```

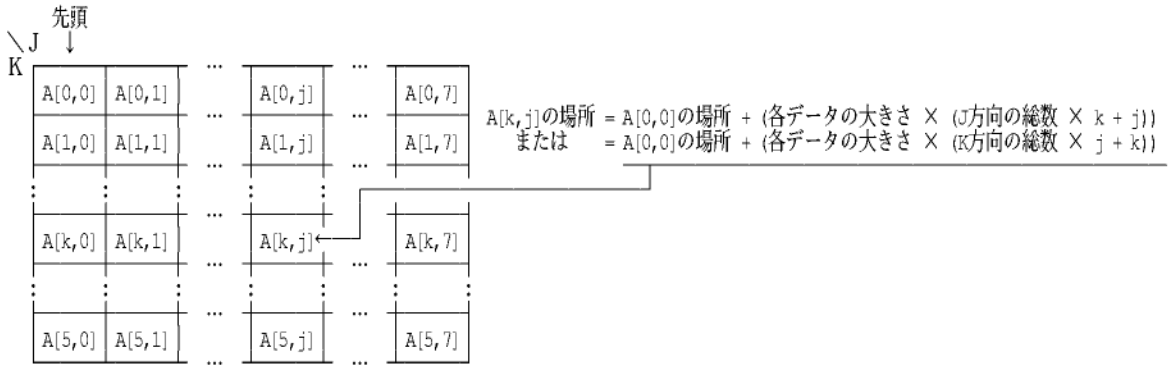
先頭要素番号が0ではなく1であることに注意

A(番号)の位置

＝ 先頭位置 ＋ 要素サイズ ＊（番号−1）

§3. 5 ならび(リスト) … 配列 … 多次元配列

二次元配列/多次元配列 … 要素数が有限の多次元構造



主記憶上でのA[X][Y]からの距離は，A[X][Y+1]とA[X+1][Y]とで大きく異なる.

高速化のためには，なるべく参照順に要素が並ぶように次元を選ぶ.

プログラミング言語によっても並び方が異なる.

§3. 5（続き）

- C言語の場合

```
int i,j;
int A[1000][1000];

/* 使える要素は A[0][0]～A[999][999] */
for (i=0; i<1000; i++) {
  for (j=0; j<1000; j++) {
    A[i][j] = xxxx;
  }
}
```

最も右の添字を最内ループで変化させると高速
A[0][0] A[0][1] … A[999][998] A[999][999]

§3. 5（続き）

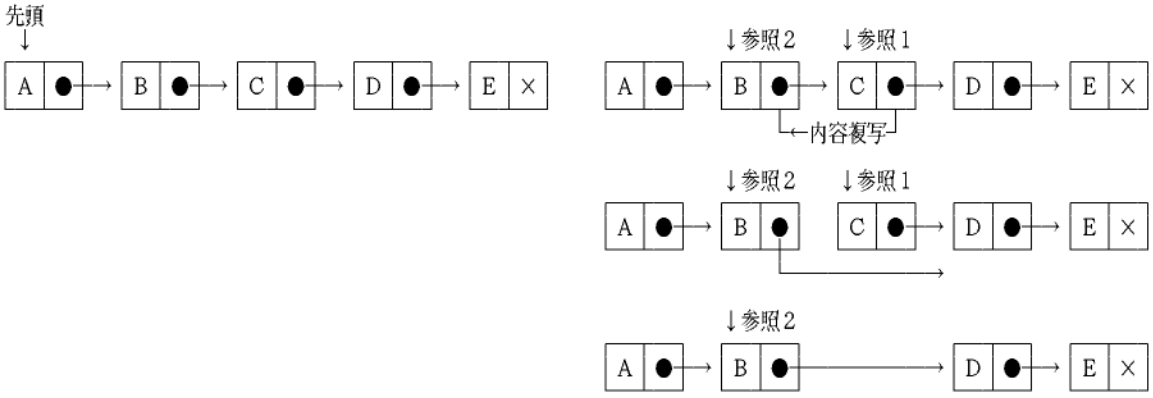
- FORTRAN言語の場合

```
      INTEGER I,J;
      INTEGER A(1000,1000);

C 使える要素は A(1,1)～A(1000,1000)
      DO 10 I=1,1000
        DO 10 J=1,1000
          A(J,I) = xxxx;
 10 CONTINUE
```

最も左の添字を最内ループで変化させると高速
A(1,1) A(2,1) … A(999,1000) A(1000,1000)

§3. 6 ならび(リスト) … つなぎ … 一方向つなぎ

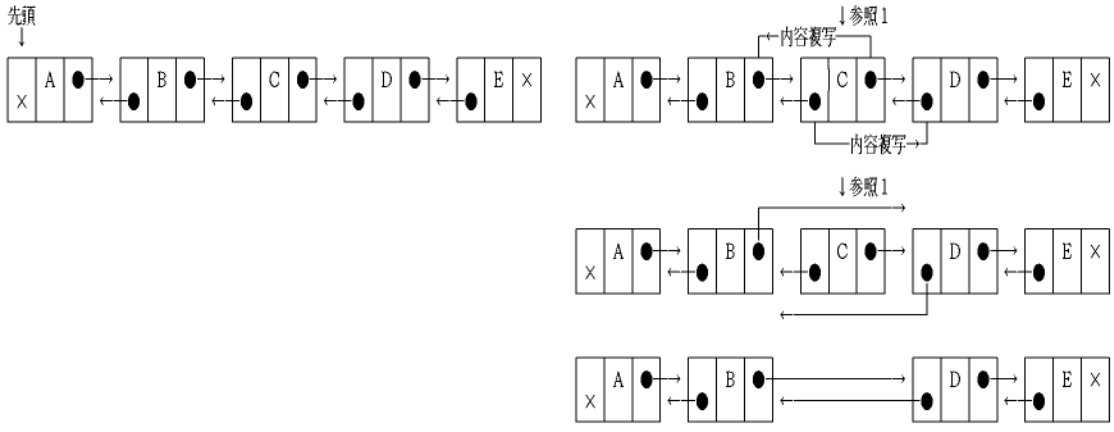
一方向つなぎ … 要素数が変化する一次元構造



(a) 一方向つなぎの例　　(b) 削除手順 (追加はこの逆)

§3. 7 ならび(リスト) … つなぎ … 双方向つなぎ

双方向つなぎ … 要素数が変化する一次元構造

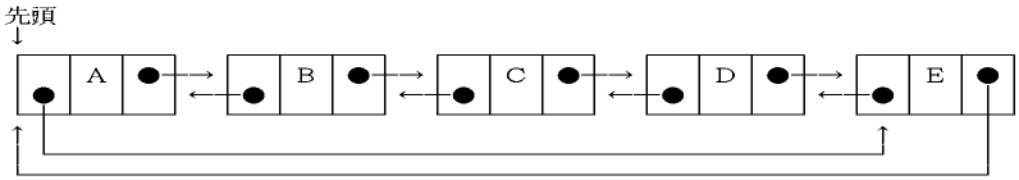


(a) 双方向つなぎの例　　(b) 削除手順 (追加はこの逆)

§3. 8 ならび(リスト) … つなぎ … 循環つなぎ

循環つなぎ … 要素数が変化する一次元構造

- 要素削除/追加の手順は双方向つなぎに同じ
- ただし，両端に関して特別扱いの必要がない



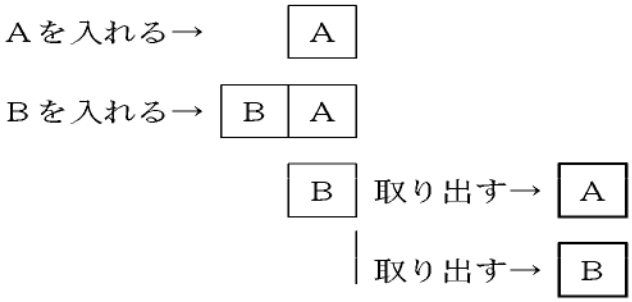
§3. 9 キューとスタック

スタック

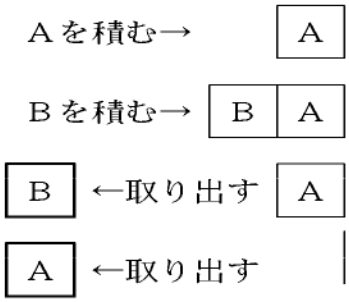
- 入口と出口が同じ側

キュー

- 入口と出口が異なる側



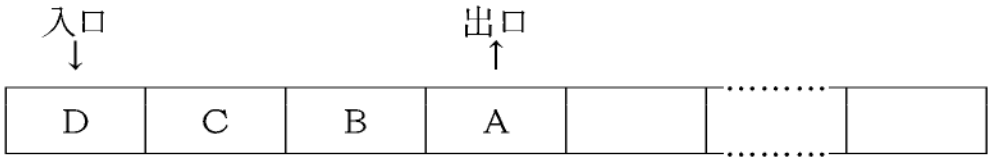
(a) キュー



(b) スタック

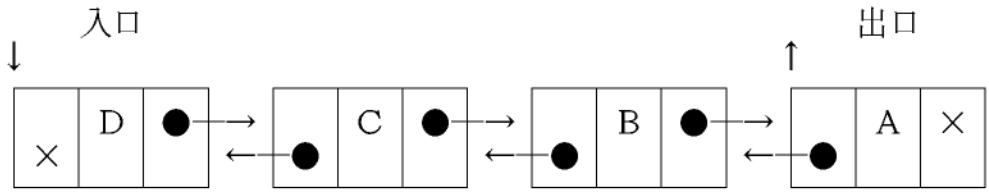
§3. 10 キューの表現

配列による表現



(a)配列による表現

つなぎによる表現



(b)つなぎによる表現

§3. 11 データ構造の信頼性を高める方法

「データ」と「データに対して許される操作」を一体のものとして考える.

明らかに不正な操作からデータを保護する.

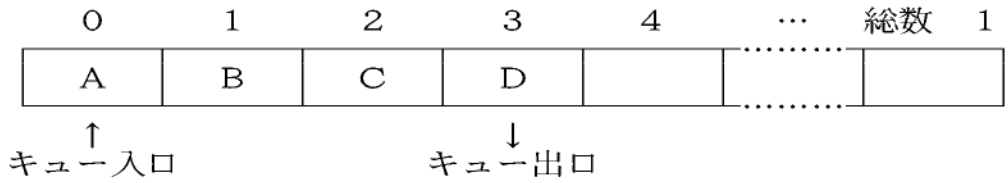
データの型に，許される操作を加えた，新しい型を「抽象データ型」と呼ぶ.

§3. 12 キューの構築

抽象データ型の適用前

→ 任意の要素を操作可能

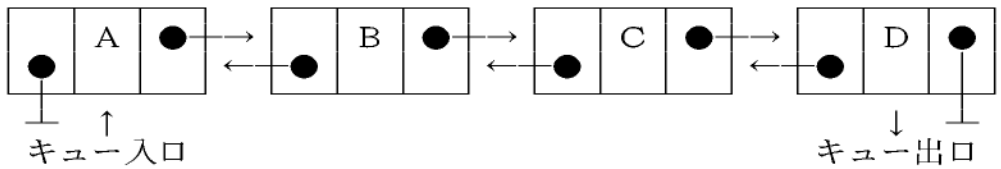
- 整数を要素とする構造体の配列




```
int queue[256];
int head = 0, tail = 0;

head = (head + 255) % 256; // enqueue
if (head == tail) overflow();
queue[head] = in;

if (head == tail) underflow(); // dequeue
tail = (tail + 255) % 256;
out = queue[tail];
```

§3. 12 キューの構築(続き)

- 整数とポインタを要素とする構造体の双方向リスト

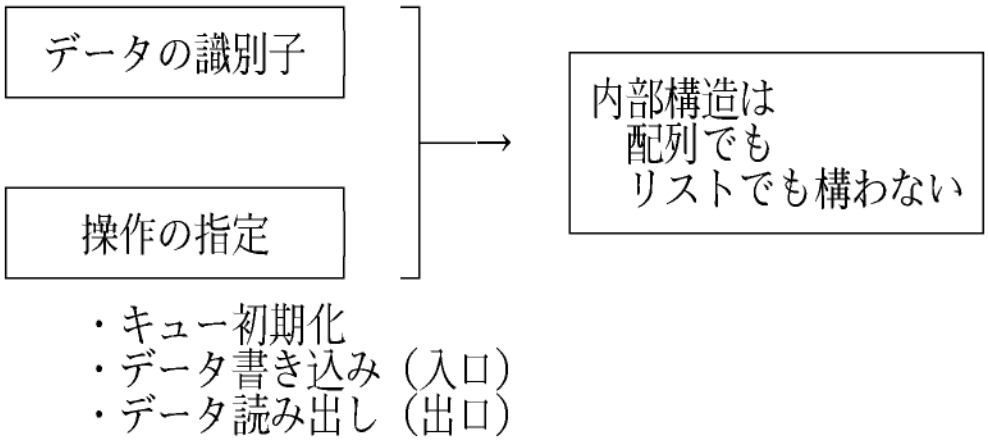



```
struct cell {
  int element;
  cell *llink;
  cell *rlink;
};
struct cell *head = NULL, *tail = NULL, *new;

if (!(new = malloc(sizeof(struct cell)))) overflow(); // enqueue
if (new->rlink = head) head->llink = new;
head = new;
head->element = in;

if (!tail) underflow(); // dequeue
out = tail->element;
if (tail = tail->llink) tail->rlink = NULL;
```

§3. 13 抽象データ型 ... Abstruct Data Type

抽象データ型の適用後

- キューとしての機能のみ使用可能
- データの内部構造は隠蔽される
- 内部構造変更の影響が少ない



§3. 13 抽象データ型(続き)

- c言語によるインプリメント

```
enum command { INIT, ENQ, DEQ };
int Queue(enum command cmd, int *data) {
  switch (cmd) {
  case INIT: // initializeの場合
     :
    return(0);
  case ENQ: // enqueueの場合
     :
    xxx = *data;
    return(0);
  case DEQ: // dequeueの場合
     :
    *data = xxx;
    return(0);
  default:  // その他はerror
     :
    return(-1);
  }
}

Queue( INIT, NULL ); // initialize
Queue( ENQ,  &in );  // enqueue
Queue( DEQ,  &out ); // dequeue
```

§3. 13 抽象データ型(続き)

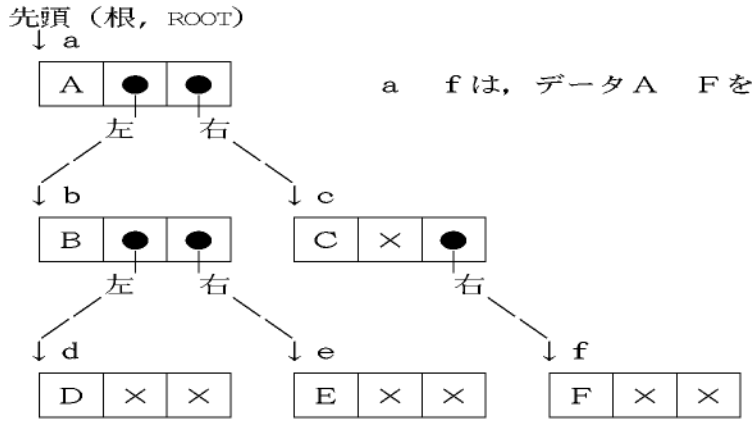
▶ JAVA言語によるインプリメント

```
class Queue {
  int [] queue;
  int head, tail;

  int init () { // initializeの場合
    queue = new int[256];
    head = 0; tail = 0;
     :
  }
  int enq (int data) { // enqueueの場合
     :
  }
  int deq () { // dequeueの場合
     :
  }
}

Queue Q = new Queue(); // オブジェクトの生成
Q.init();              // initialize
Q.enq(in);             // enqueue
out = Q.deq();         // dequeue
```

§3. 14 要素数が変化する多次元構造(二分木)

木構造

▶ 二分岐の木(二分木)



a　fは，データA　Fを指す参照とする．

木の走査

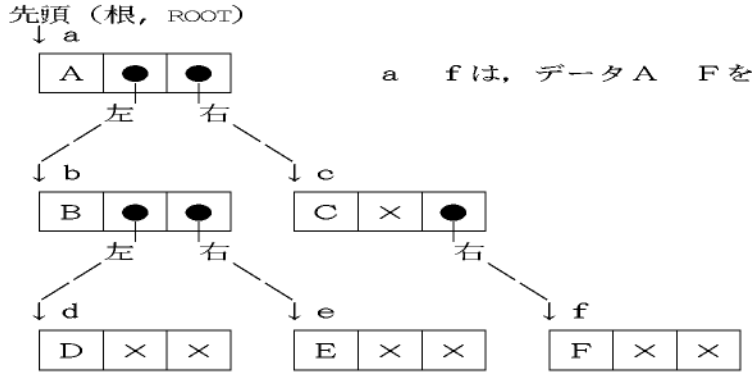
1．左手が終端でなければ左手へ進む
2．左手が終端ならば右手へ進む
3．右手が終端ならば親へ戻る

§3. 14 要素数が変化する多次元構造(二分木続き)

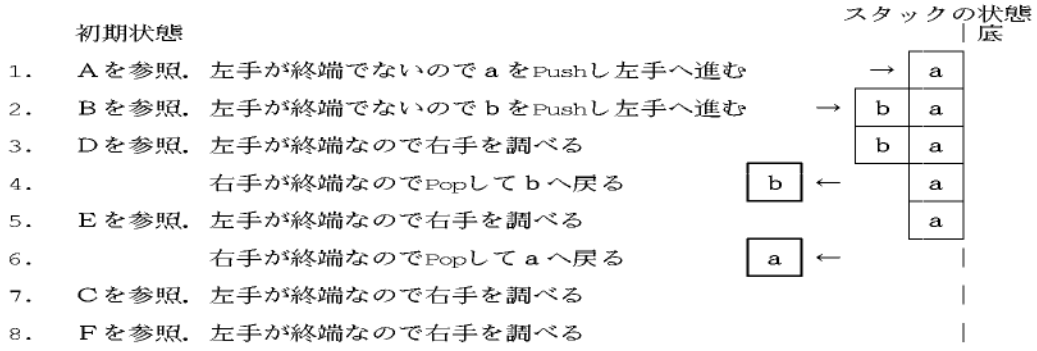
【行きがけ順，Pre-Order，深さ優先走査】

llink走査前にデータを参照する … スタックに適合

スタックの代わりにキューを用いると，幅優先走査



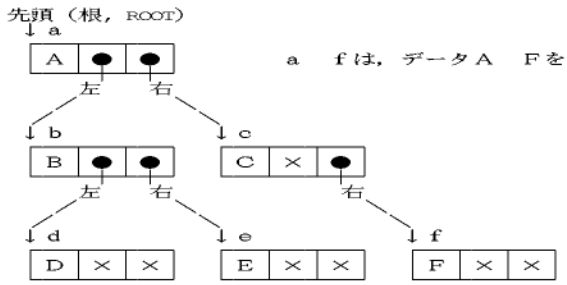
a f は，データA F を指す参照とする．



§3. 14 要素数が変化する多次元構造(二分木続き)

【通りがけ順，In-Order】

llink走査後rlink走査前にデータを参照する



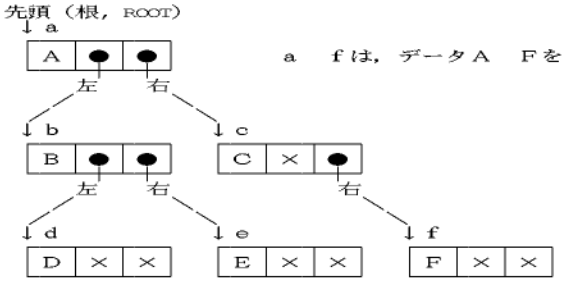
a f は，データA F を指す参照とする．

aのllink≠空		aのllinkへ
bのllink≠空		bのllinkへ
dのllink=空	Dを参照	dのrlink=空
bへ戻り	Bを参照	bのrlinkへ
eのllink=空	Eを参照	eのrlink=空
bへ戻りaへ戻り	Aを参照	aのrlinkへ
cのllink=空	Cを参照	cのrlinkへ
fのllink=空	Fを参照	fのrlink=空
cへ戻りaへ戻り	終了	

データの参照順はDBEACF

§3. 14 要素数が変化する多次元構造(二分木続き)

【帰りがけ順，Post-Order】
rlink走査後にデータを参照する

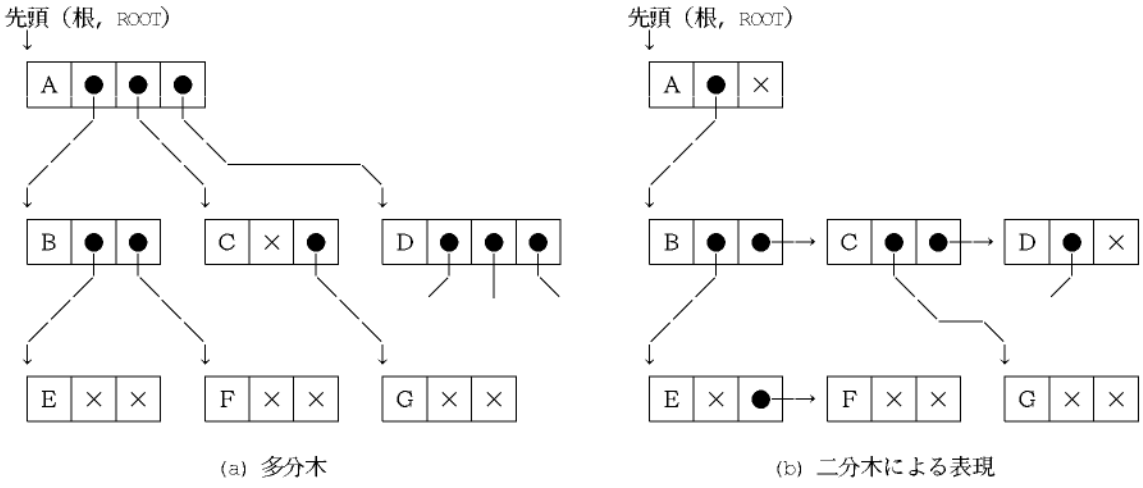


aのllink≠空	aのllinkへ	
bのllink≠空	bのllinkへ	
dのl/rlink=空		Dを参照
bへ戻り	bのrlinkへ	
eのl/rlink=空		Eを参照
bへ戻り		Bを参照
aへ戻り	aのrlinkへ	
cのllink=空	cのrlinkへ	
fのl/rlink=空		Fを参照
cへ戻り		Cを参照
aへ戻り		Aを参照

データの参照順はDEBFCA

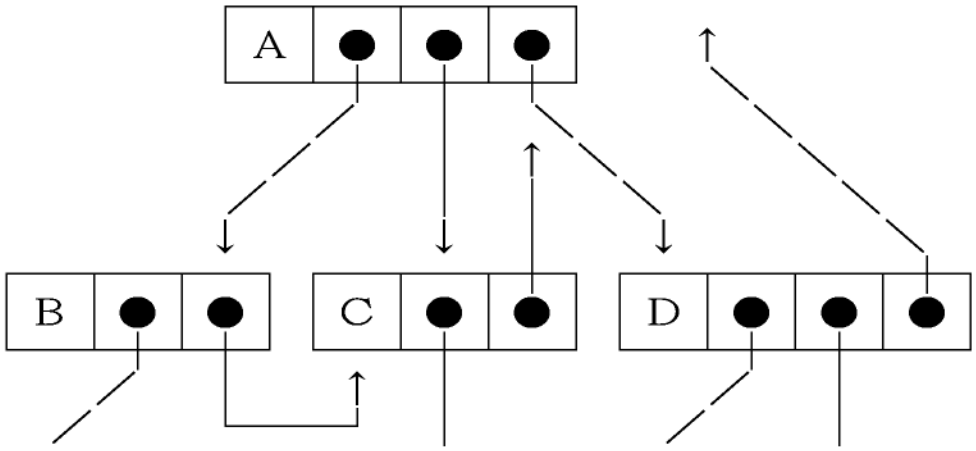
§3. 15 要素数が変化する多次元構造(多分岐の木)

多分岐の木
- 二分木による表現



§3. 16 要素数が変化する多次元構造(グラフ構造)

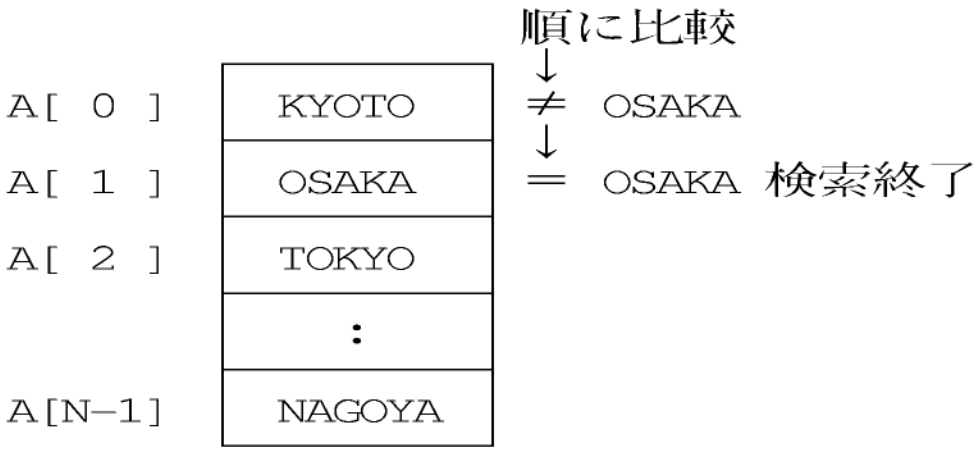
グラフ構造



§3. 17 応用 … データの検索(N個のデータから1つを探す)

配列やつなぎを用いる方法

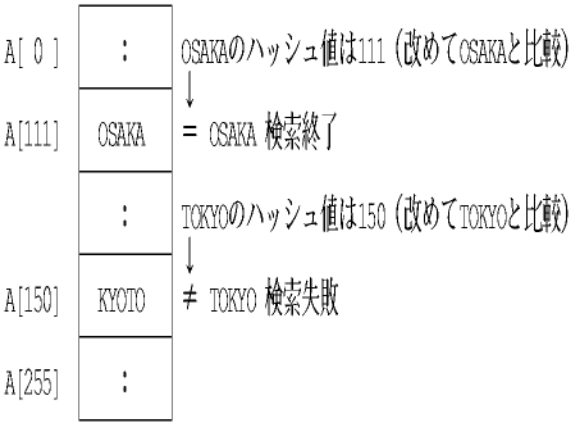
- 先頭から順に比較していく
- 最良で1回，最悪でN回の検索



§3. 17 応用(続き)

ハッシュ

- ハッシュ値から添字の見当をつけ，比較する.

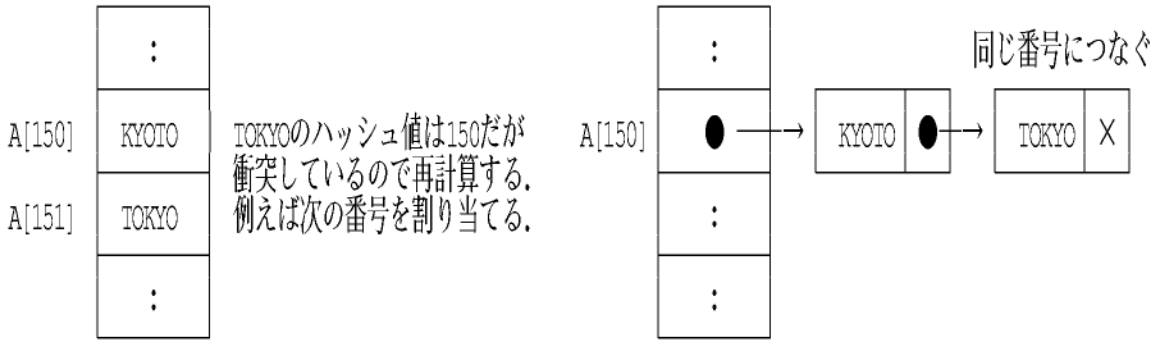


文字列	ASCIIコード	合計	下位8bit	10進数
KYOTO	4B 59 4F 54 4F	196(16)	96(16)	150(10)
OSASA	4F 53 41 4B 41	16F(16)	6F(16)	111(10)
TOKYO	54 4F 4B 59 4F	196(16)	96(16)	150(10)
NAGOYA	4E 41 47 4F 59 41	1BF(16)	BF(16)	191(10)

§3. 17 応用(続き)

衝突の回避

- 不一致の場合，再ハッシュにより別の添字を求める.（1を加えるなど）
- ハッシュ表に対して要素が疎であれば1回の検索.



(a) クローズドハッシュ　　(b) オープンハッシュ

今日はここまで