情報処理入門

5章「データ構造とアルゴリズム」

中島康彦

§5. 1 問題をコンピュータに解かせるということ

「未来の宝クジの当選番号はなにか」
- という問いには正解がない

「2×3はいくつか」
- 10進数を仮定している限り正解は一つ

「明日は何センチの雪が積もるか」
「読みたい本が近所の図書館にあるか」
- 簡単には解けないように思えるが．．．

§5. 2 プログラム＝データ構造＋アルゴリズム

基本データどうしを関連づける何かが必要
情報の本質的な構造を反映する仕掛けが「データ構造」
データ構造が決まれば，コンピュータは仕事をしてくれるか？
- 答えは否

仕事の手順が「アルゴリズム」
データ構造とアルゴリズムを伝えれば，コンピュータが働く．
- 伝えるための言葉がプログラミング言語
- プログラミング言語で記述したものが「プログラム」
- 同じ問題を解くプログラムであっても，データ構造とアルゴリズムが異なれば，プログラムも異なる．

§5. 3 時間的コストと空間的コスト

計算結果が同じであっても，プログラムは様々
- プログラミング言語の違いや見やすさなど外見上の差
- 同じ問題を解くのに必要なコストが異なる

コンピュータが問題を解くコスト
- 処理時間（時間的コスト）
- 必要なメモリ空間（空間的コスト）
- データ構造とアルゴリズムによって決まる
- 一般的に，一方を減らすと他方が増えるという困った関係

最良値，平均値，最悪値のそれぞれを考慮する必要
- 最良値と平均値が良くてもダメ．
- 最悪値が許容範囲を越えてもよいか？

§5. 4 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

著者（文字列）	
題名（文字列）	
出版社（文字列）	
ＩＳＢＮ（文字列）	
発行年（整数）	価格（整数）

(a)構造体

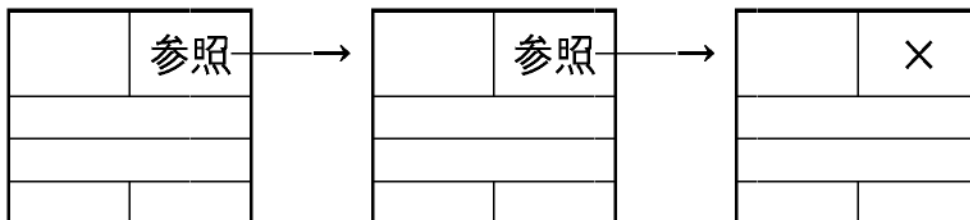
種別（整数）＝書籍（1）	
著者（文字列）	
題名（文字列）	
出版社（文字列）	
ＩＳＢＮ（文字列）	
発行年（整数）	価格（整数）

種別（整数）＝ＣＤ（2）	
作曲（文字列）	
曲名（文字列）	
製作（文字列）	
時間（整数）	枚数（整数）
製作年（整数）	価格（整数）

(b)共用体

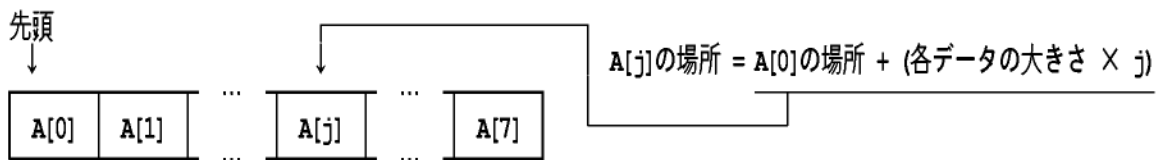
§5. 5 他のデータを参照するための型

参照/ポインタ reference/pointer



§5. 6 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.

§5. 6（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;
}
```

§5. 6（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
```

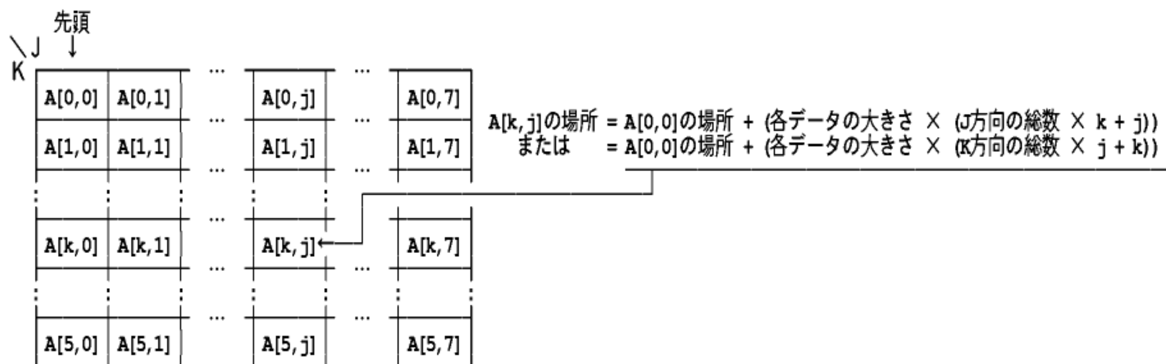
先頭要素番号が0ではなく1であることに注意

A(番号)の位置

＝ 先頭位置 ＋ 要素サイズ ＊（番号−1）

§5. 7 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.

§5. 7（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]

§5. 7（続き）

▶ FORTRAN言語の場合

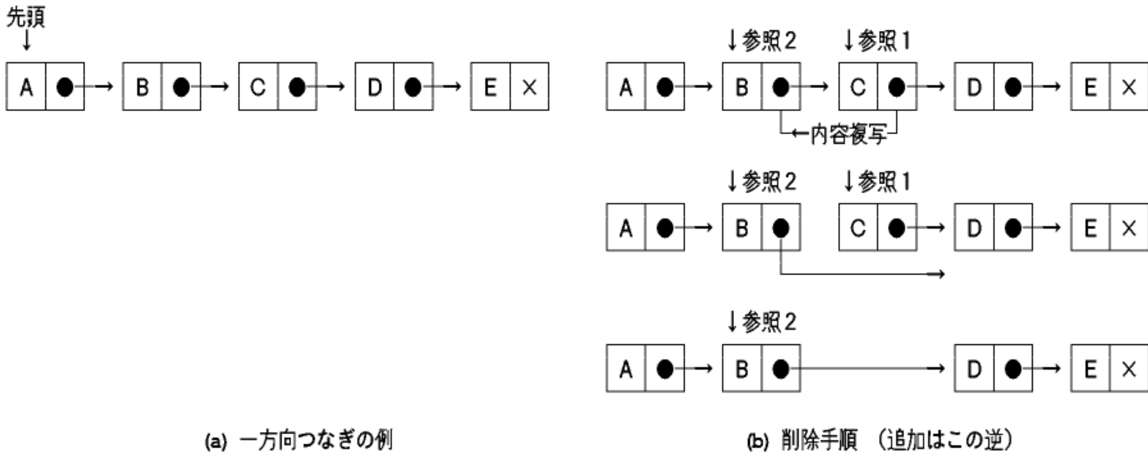
```
    INTEGER I,J;
    INTEGER A(1000,1000);

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)

§5. 8 ならび(リスト) … つなぎ … 一方向つなぎ

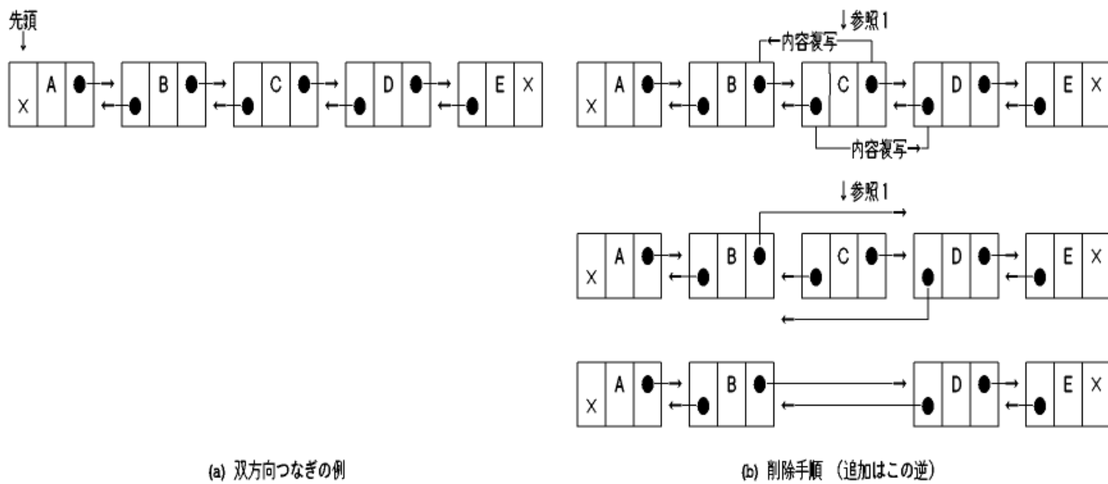
一方向つなぎ … 要素数が変化する一次元構造



(a) 一方向つなぎの例　　(b) 削除手順 （追加はこの逆）

§5. 8 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

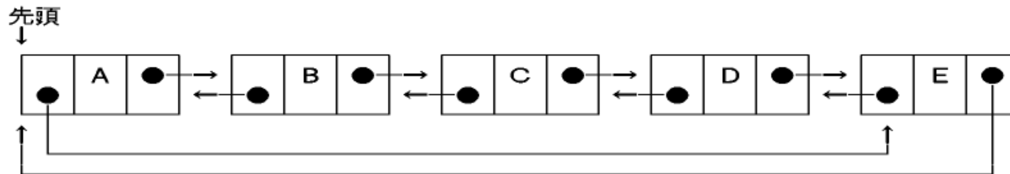


(a) 双方向つなぎの例　　(b) 削除手順 （追加はこの逆）

§5. 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



§5. 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を入れる→ A

Bを入れる→ B A

B 取り出す→ A

取り出す→ B

(a) キュー

Aを積む→ A

Bを積む→ B A

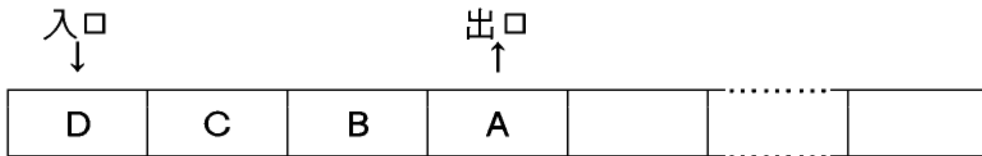
B ←取り出す A

A ←取り出す

(b) スタック

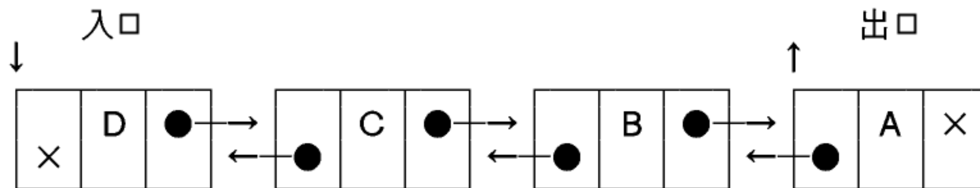
§5. 10 キューの表現

配列による表現



(a)配列による表現

つなぎによる表現

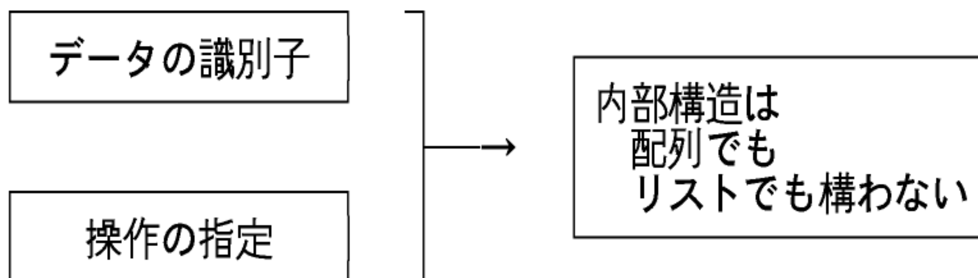


(b)つなぎによる表現

§5. 11 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



・キュー初期化
・データ書き込み（入口）
・データ読み出し（出口）

§5. 11 抽象データ型(続き)

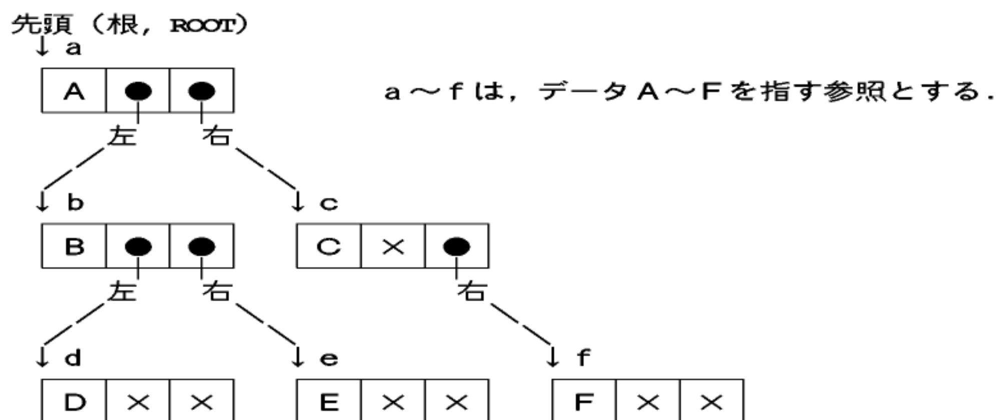
例えばJAVA言語の場合

```
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
```

§5. 12 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)

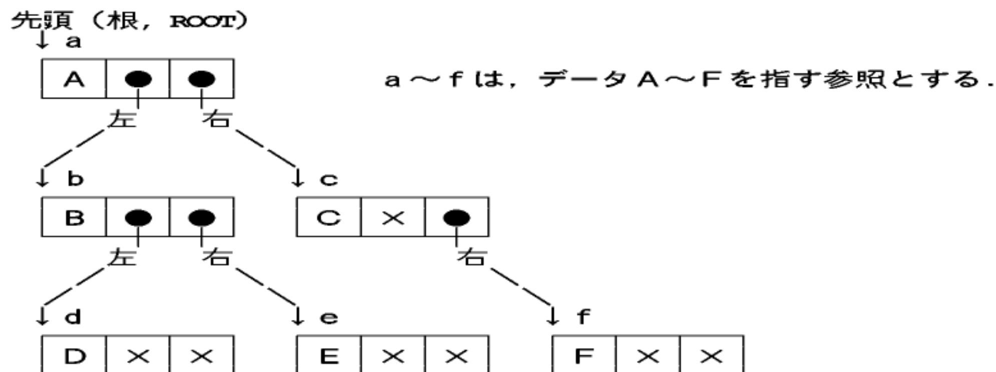


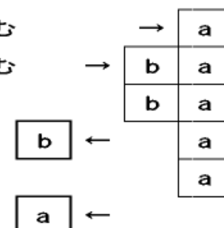
木の走査

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

§5．12 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





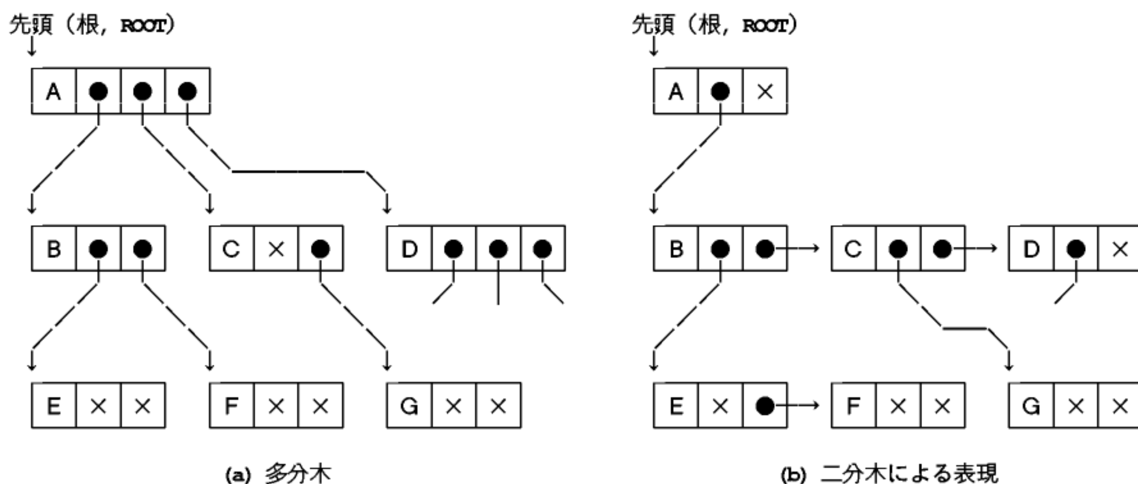
初期状態

1. Aを参照．左手が終端でないのでaをPushし左手へ進む
2. Bを参照．左手が終端でないのでbをPushし左手へ進む
3. Dを参照．左手が終端なので右手を調べる
4. 右手が終端なのでPopしてbへ戻る
5. Eを参照．左手が終端なので右手を調べる
6. 右手が終端なのでPopしてaへ戻る
7. Cを参照．左手が終端なので右手を調べる
8. Fを参照．左手が終端なので右手を調べる
9. 右手が終端かつスタックが空なので探索終了

§5．12 要素数が変化する多次元構造(多分岐の木)

多分岐の木

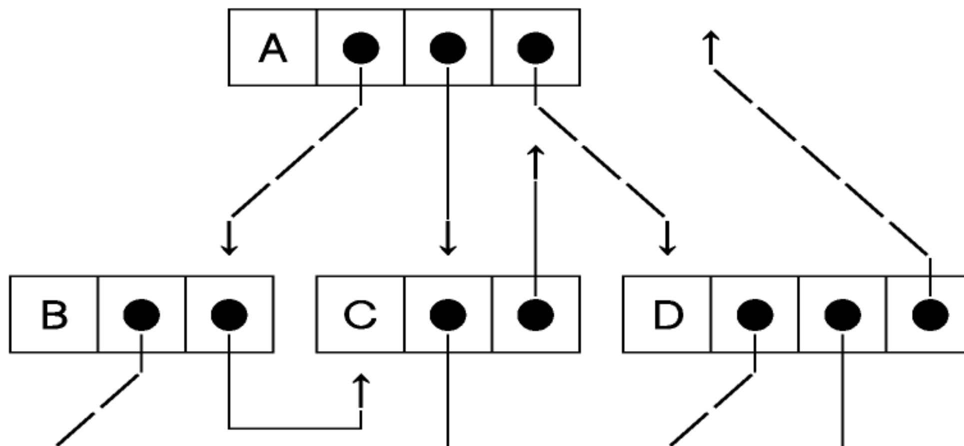
▶ 二分木による表現



(a) 多分木　　(b) 二分木による表現

§5. 12 要素数が変化する多次元構造(グラフ構造)

グラフ構造



§5. 13 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

		順に比較 ↓
A[0]	KYOTO	≠ OSAKA ↓
A[1]	OSAKA	＝ OSAKA 検索終了
A[2]	TOKYO	
	:	
A[N−1]	NAGOYA	

§5. 13 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.

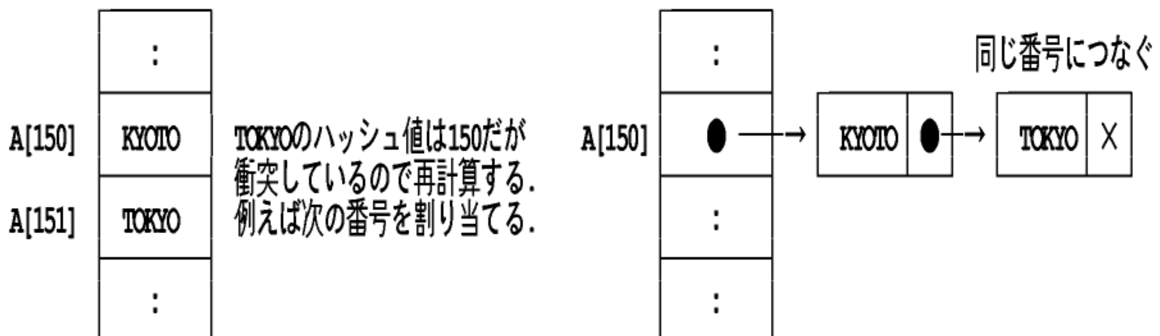
A[0]	:	OSAKAのハッシュ値は111 (改めてOSAKAと比較) ↓
A[111]	OSAKA	= OSAKA 検索終了
	:	TOKYOのハッシュ値は150 (改めてTOKYOと比較) ↓
A[150]	KYOTO	≠ TOKYO 検索失敗
A[255]	:	

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)

§5. 13 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