動的な集合のためのデータ構造

アルゴリズム論

中野

Note 5

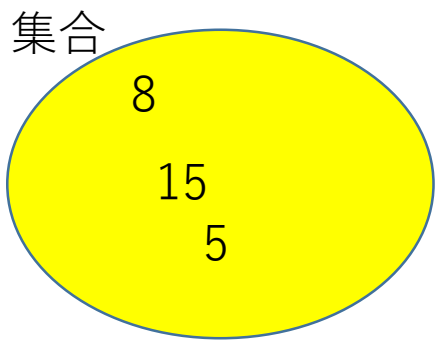
2020.4.13 作成

2020.5.23update　2021.5.10update

動的な集合

データの動的な集合Sがある。
各データは (1)固有のkeyと(2)key以外の部分(satellite date)からなるとする。
これを効率的に格納するデータ構造を考えよう。

動的 = データの追加や削除がある。

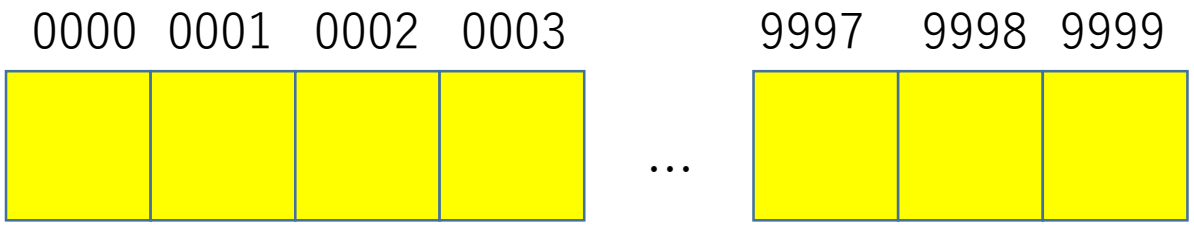


基本辞書操作

- Search(k)　keyがkであるデータの探索
- Insert(x)　新データxの追加
- Delete(x)　データxの削除(Delete(key)の場合もあり)
- Minimum　最小のkeyのデータの探索
- Maximum 最大のkeyのデータの探索
- Successor(x) データxの次に大きなkeyを持つデータの探索
- etc.

(注意 データとkeyの区別がない場合、つまりsatellite dataがない場合もある。)

Bit Vector



S ⊂ U = {0,1,2,...,m-1} (ただし m が比較的小さい整数)のとき使える。
(注意 今回はsatellite dataはありません。データ = key に注意)
整数の集合Sを、bitの配列 B[0..m-1] に格納する。
メモリの使用量がきわめて少ない。必要なメモリは |U| bit である。
i番目の要素が、集合Sに含まれていれば　B[i]=1 とし、
　　集合Sに含まれていなければ B[i]=0 とする。

基本辞書操作

Search(key) データの探索　O(1) time
Insert(key) 新データの追加　O(1) time
Delete(key) データの削除　O(1) time

(例)電話番号のソート

ソート済み配列(sorted array) + 2分探索(binary search)

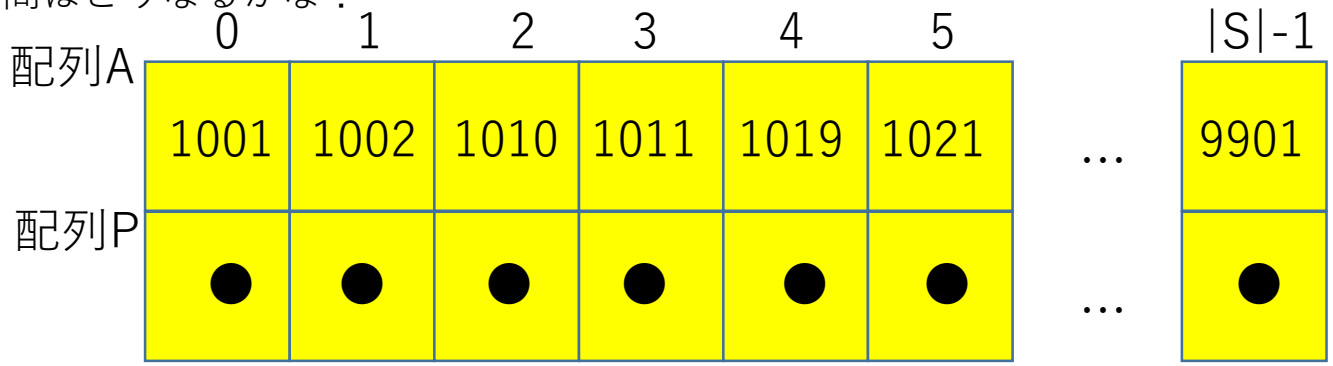
S ⊂ U = {0,1,2,...,m-1} (ただし m は 1 wordに格納できる整数の最大値)のとき使える。
(注意 今回はsatellite dataはありません。データ = key に注意)
整数の集合Sを、整数の配列 A[0..n-1] に格納する。 n ≧ |S| である。
メモリの使用量が少ない。|S| word + 余裕 = n word 使用する。

基本辞書操作

Search(key) データの探索 O(log n) time
Insert(key) 新データの追加 O(n) time
Delete(key) データの削除 O(n) time

(例 データがない = -1 とする)(配列の後の方の余りに -1 を書いておく。)

(クイズ 1) ソートせずに格納すると、基本辞書操作の時間はどうなるかな？
(クイズ 2) satelliteデータを扱うには
どうすればいいかな？
ヒント: ポインタの配列を使用。



動的(dynamic)ソート済み配列(sorted array) + 2分探索

S ⊂ U = {0,1,2,...,m-1} (ただし m は 1 wordに格納できる整数の最大値)のとき使える。
(注意 今回はsatellite dataはありません。データ = key に注意)
整数の集合Sを、整数の配列 A[0..n-1] に格納する。
(ここで n は |S|より大きい最小の2のべき乗とする。)
この配列を

長さ1の整数の配列 A_0
長さ2の整数の配列 A_1
長さ4の整数の配列 A_2
長さ8の整数の配列 A_3
...
長さ2^{k-1}の整数の配列 A_{k-1}

に分割する。k = log (n+1) の切り上げ　とする。

動的(dynamic)ソート済み配列(sorted array)＋2分探索

整数の集合Sを、整数の配列 A[0..n-1] に格納する。この配列を

長さ1の整数の配列　A_0

…

長さ2^{k-1}の整数の配列　A_{k-1}

に分割する。k = log (n+1) の切り上げ　とする。

|S| の2進数表現を (b_{k-1} b_{k-2} … b_0)とする。

Sの要素を分割して、各A_i に格納する。

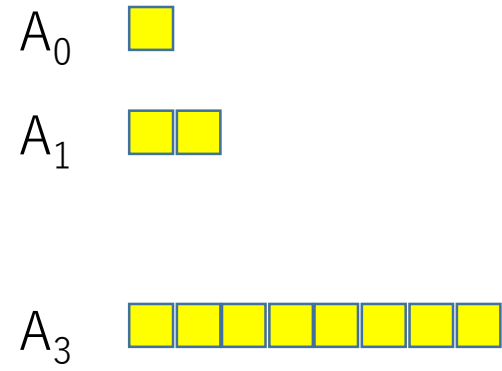
ただし、$b_i = 0$ である 各 A_i は使用しない。

$b_i = 1$ である 各 A_i のみ使用する。

$b_i = 1$ である 各 A_i には、2^i個の要素を(すなわちmax個)格納する。

(n の値によって、どの b_i を使用するのかが、変化することに注意しよう。)

例　$n=11=1011_{(2)}$



動的(dynamic)ソート済み配列(sorted array)＋2分探索

各A_i はソートされている。

しかし、A_iと A_j中のデータには特に関係ない。

メモリの使用量は、高々 2|S| word である。

基本辞書操作(今回は Search,Insertのみ)

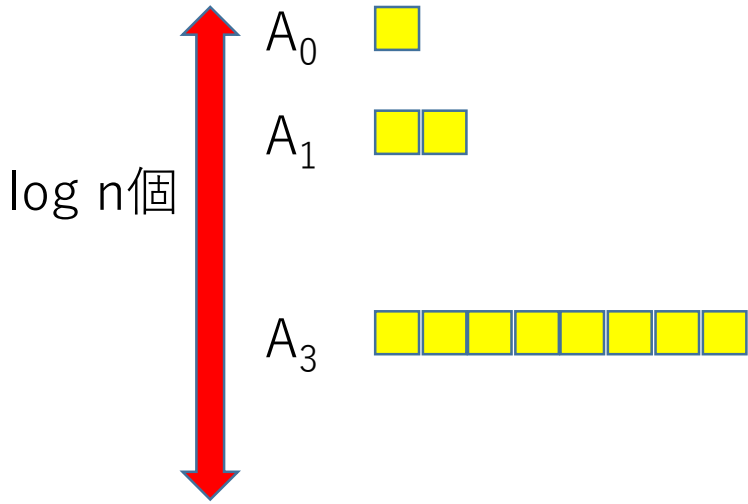
Search(key) データの探索　$O(\log^2 n)$ time

Insert(key) 新データの追加

平均 $O(\log n)$ time

(各要素は 高々log n回 コピーされる)

例　$n=11=1011_{(2)}$



(クイズ 3)

Search(key) はどのようにすればいいかな？

(クイズ 4)

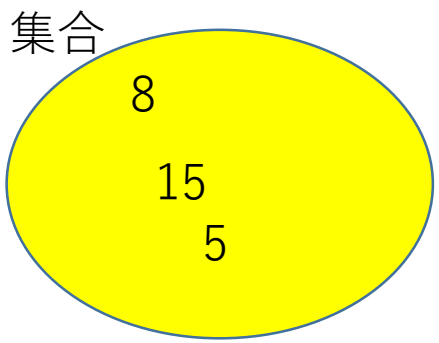
Delete(key) データの削除も効率的に実現できるかな？

(クイズ 5) satelliteデータを扱うにはどうすればいいかな？

Linked lists

NULL©
None(Python)
NIL(Pascal)

keyの集合は任意である。(配列では整数でないとダメでした!)
集合Sを、線形リストとして格納する。



ポンタとデータから構成されるものをノードという。
ここで, ポインタはノードを指す。
ノードの集合を直線的に繋いだものを線形リストという.

線形リストの他にもいろいろなリストがある。

単方向 --- 双方向　　sorted --- unsorted　　circular ---

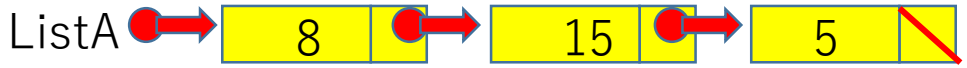
基本辞書操作

Search(key) データの探索　O(n) time
Insert(x)　新データの追加　O(1) time(unsortedの場合)
Delete(x)　データの削除　O(n) time(探索してから削除)

Linked lists

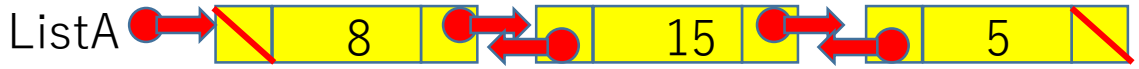
必要なメモリは、

単方向の場合



|S| + (1つのポインタを格納するのに必要なword数)|S| word

双方向の場合

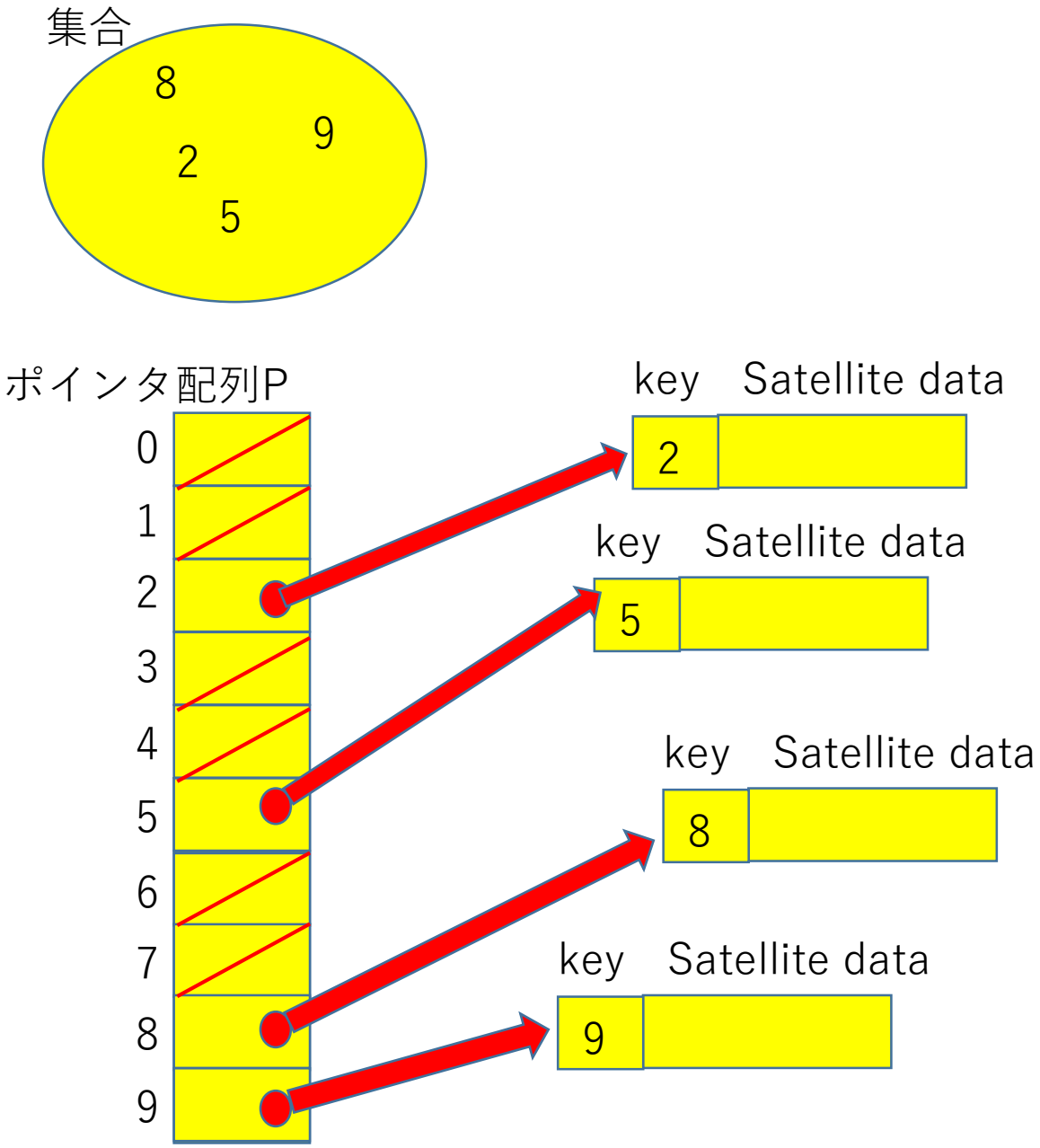


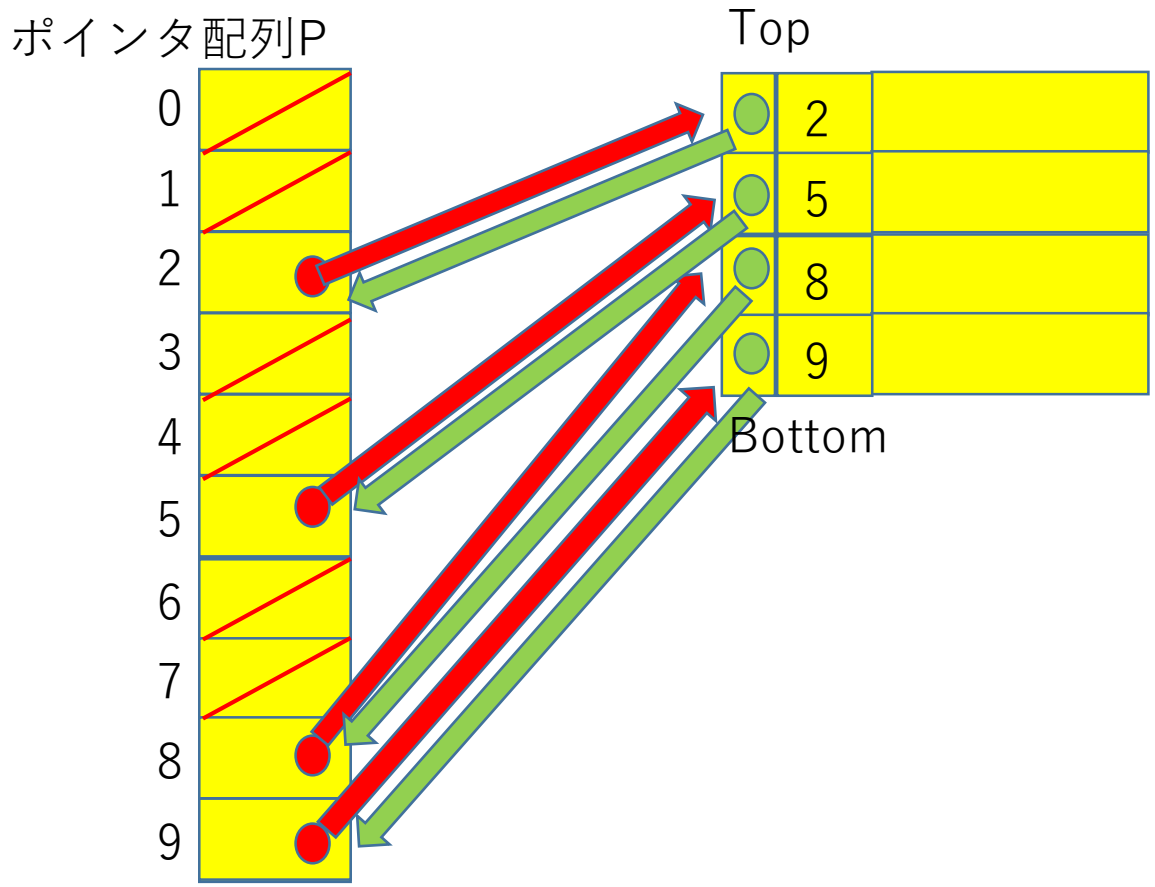
|S| + 2(1つのポインタを格納するのに必要なword数)|S| word

Direct-address tables

keyの集合 ⊂ {0,1,2,...,m-1}
(ただし m は 1 word に格納できる整数の最大値)
のときに使える。
(注意 今回はsatellite dataがあります)
集合Sを、
ポインタの配列 P[0..m-1] を用いて、
次のようなデータ構造に格納する。

もしSが、
key=i なるデータを含めば、
　　P[i]はそのデータへのポインタである。
key=i なるデータを含まなければ、
　　P[i] = NIL である。





基本辞書操作 (Bit Vectorと同様)

Search(key) データの探索 O(1) time

Insert(x) 新データの追加 O(1) time

Delete(x) データの削除 O(1) time

(クイズ 6)

Maximum keyが最大のデータの探索の計算時間はどのくらいかな？

(クイズ 7)

巨大な配列 P を使用する際には、最初に、初期化が必要である。(Garbageをクリアする。) P が非常に巨大なとき、この初期化をしないでもよい方法はないかな？

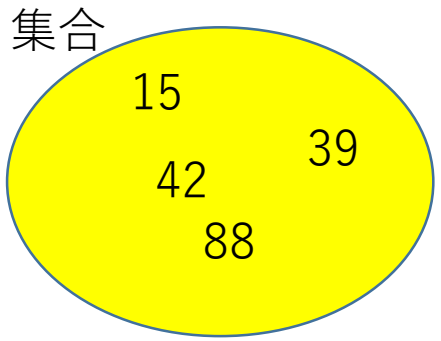
(クイズ 8)ポインタのテーブルではなく、(サテライト)データのテーブルを使用するとどうなるかな？

Hash table

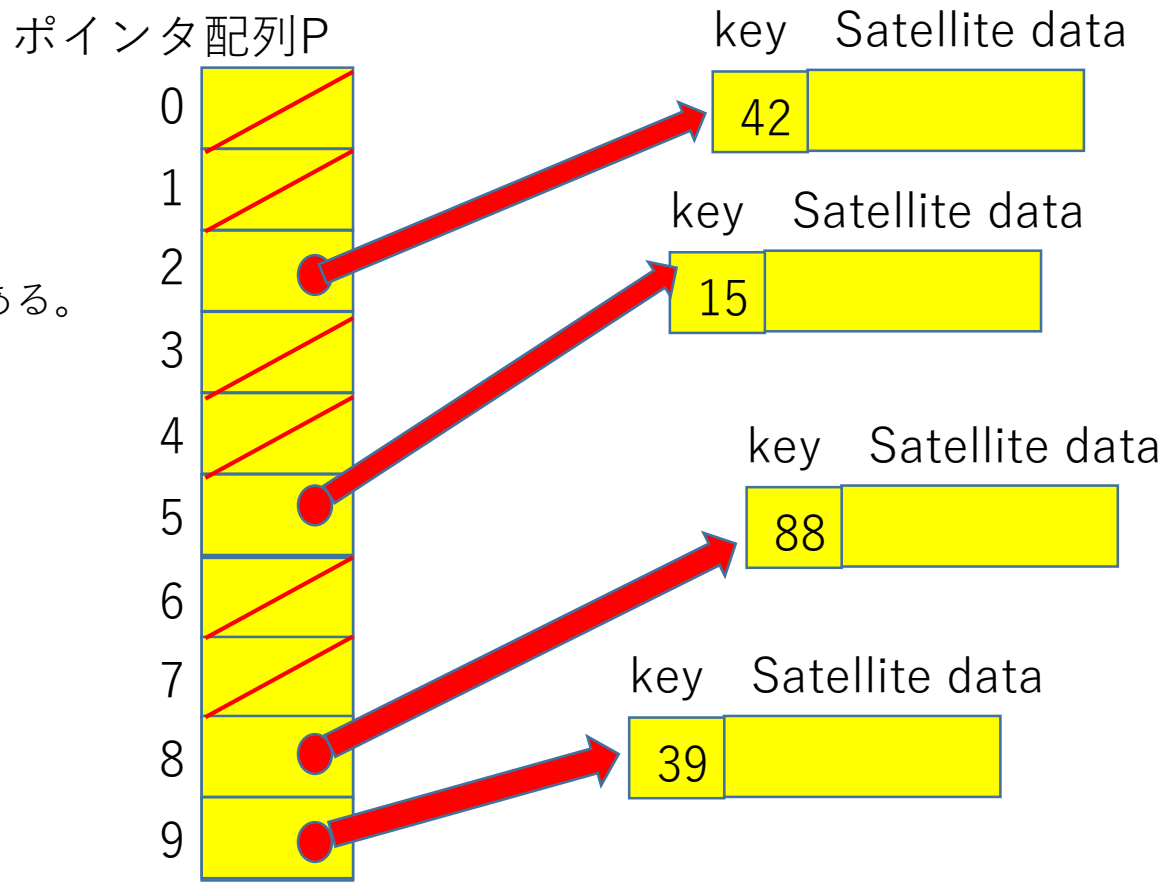
Direct-address tableの配列Pはほどんど NIL かもしれない。
(メモリのムダ。)

keyの集合 ⊂ {0,1,2,...,u-1} (ただし u は 巨大な整数でも大丈夫)

(整数とは限らない)集合Sを、ハッシュ関数hと
ポインタの配列 P[0..m-1] を用いて、
次のようなデータ構造に格納する。
　hは{0,1,2,...,u-1}から{0,1,2,...,m-1}への関数である。u>>mである。
もしSが、
key=i なるデータを含めば、　P[h(i)]はそのデータへのポインタ
key=i なるデータを含まなければ、P[h(i)] = NIL
つまり、メモリのムダを少なくしている。
シカシ、 h(i) = h(j) のようなkeyがあると問題が生じる。
衝突(collision)という。
chaining(リスト構造)、もしくは、open addressing
(第2,3,4,..希望の格納場所)によって解決する。



ハッシュ関数の例
1 の位を返す



基本辞書操作

Search(key)　データの探索　　平均O(1) time

Insert(x)　新データの追加　O(1) time

Delete(x)　データの削除　　平均O(1) time