アルゴリズム論 4,5,6

探索

- 線形探索
- ２分探索
- ハッシュ探索

ハッシュ探索の考え方

線形探索の場合：探索を行うデータの並びに制約はない

２分探索：探索を行うデータを事前に昇順または降順に並べる必要がある

ハッシュ探索：探索を行うデータをメモリに格納する際に、さらに工夫する。

データの格納場所を決める関数：ハッシュ関数

ハッシュ探索の実現法

ハッシュ探索：

ハッシュ関数:データを格納する場所を決める規則

データを格納する場所をプログラム上で用意する

例: 通常　データ数の1.5倍～2倍の領域を確保

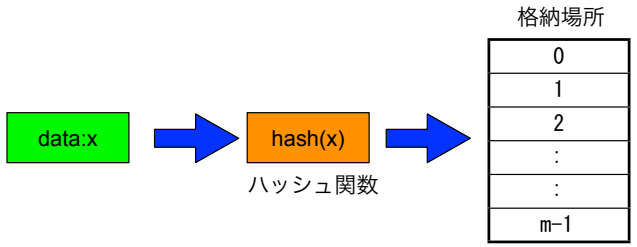
ハッシュ関数

望ましいハッシュ関数の条件：

・計算が容易なこと

・格納する場所にデータがランダムに配置されること

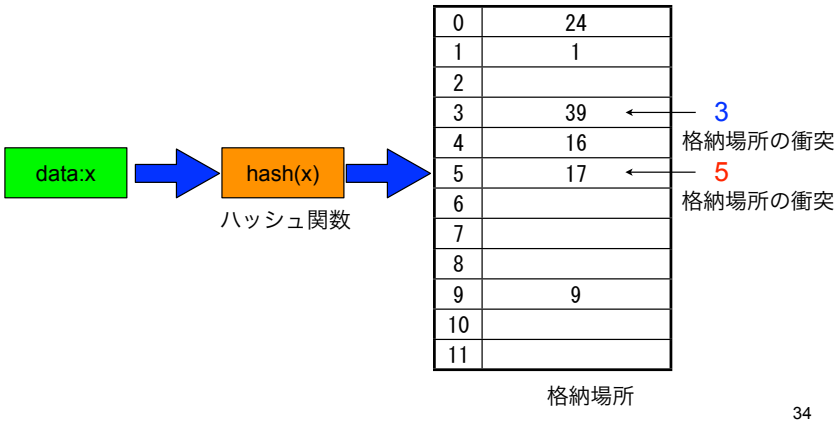
ハッシュ関数例：剰余算



ハッシュ探索のデータ格納例

入力データ: 17, 39, 1, 9, 24, 16, 5, 3

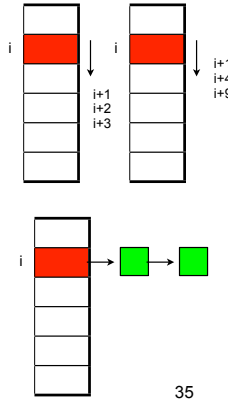
ハッシュ関数:　hash(x)= x % 12



データ格納場所の衝突処理

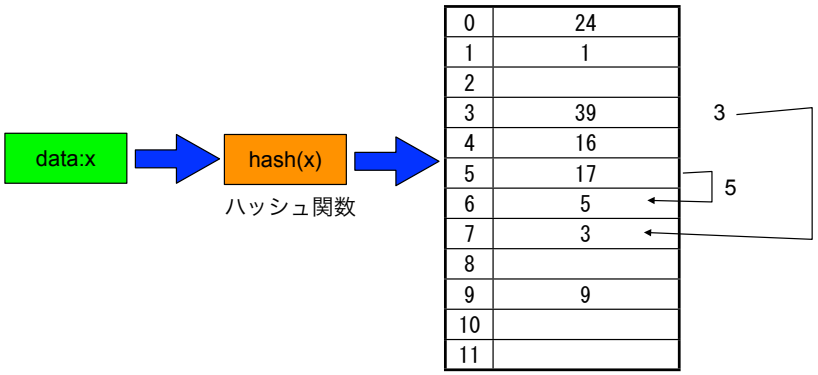
データ格納場所が衝突した場合の処理

- オープンアドレス法(クローズドハッシュ法)
  - データ数nより多めの格納場所を用意し、衝突が発生したら格納場所の空いている場所を探して格納する方法
    - -線形探査法
    - -平方探査法
- チェイン法(オープンハッシュ法)
  - -衝突が発生したらリストで格納場所を確保する方法



データ格納場所の衝突処理

オープンアドレス法(クローズドハッシュ法)+線形探査

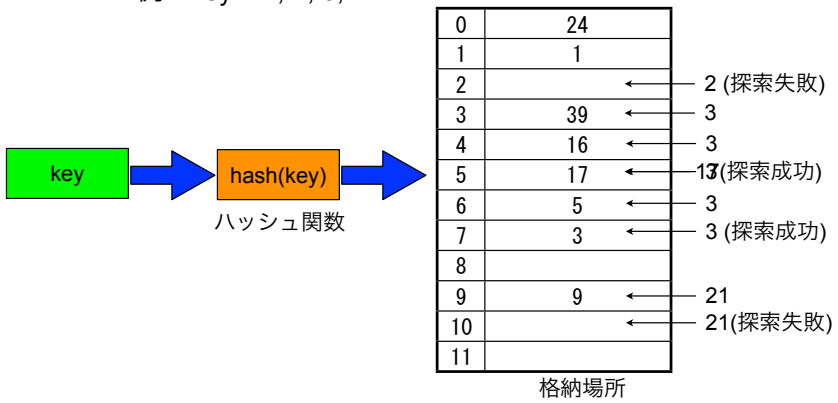


ハッシュ探索処理

データ格納と同じ手順で行う

ハッシュ関数:　hash(key)= key % 12

例　key: 17, 2, 3, 21



ハッシュ探索プログラム（メイン）

```
#include <stdio.h>
#include <stdlib.h>

#define NUM 8

struct harea {    /* データを格納する構造体  */
    int flag;     /* データ格納の状態 0:無 1:有  */
    int data;     /* データ  */

};

void hash_add(struct harea a[], int hsize, int key);
int hash_search(struct harea a[], int hsize, int
key);

int main(void)
{
    int    i,ky,idx;
    int    x[NUM];
    int    hsize=1.5*NUM;
    struct harea ha[hsize];   /* データ格納場所 */

    for (i=0;i<hsize;i++) ha[i].flag=ha[i].data=0;

    printf(" Input integer number %d times \n",NUM);

    for (i=0;i<NUM;i++)  {
        printf("x[%d]:",i);
        scanf("%d",&x[i]);
    }
```

```
for (i=0;i<NUM;i++)  {
        hash_add(ha,hsize,x[i]);
    }

    printf("Number to search:");
    scanf("%d",&ky);

    idx=hash_search(ha,hsize,ky);
    if (idx==-1)
        printf("Searching was failed!\n");
    else
        printf("%d is located at %d \n",ky,idx);

    return(0);
}
```


ハッシュ探索プログラム（関数）

データ格納を行う関数

```
void hash_add(struct harea a[], int hsize, int key)
{
   int pos;

   pos=key % hsize;        /* ハッシュ関数  */

   while(a[pos].flag!=0) {
      if (a[pos].flag>0) {
         if (a[pos].data==key)
            return;
         }
      pos=(pos+1) % hsize;    /* オープンアドレス  */
      }

   a[pos].flag=1;     /* データ格納の状態  */
   a[pos].data=key;   /* データ格納  */

}
```


ハッシュ探索プログラム（関数）

データ探索を行う関数

```
int hash_search(struct harea a[], int hsize, int key)
{
   int   pos, ret;

   pos=key % hsize;        /* ハッシュ関数  */
   ret=-1;                 /* 探索結果　初期値  */

   while(a[pos].flag!=0) {
      if (a[pos].flag>0) {
         if (a[pos].data==key)  {
            ret=pos;
            break;
         }
      }
      pos=(pos+1) % hsize;  /* オープンアドレス　線形探索  */
   }
   return ret;    /* 探索結果　-1: 失敗  */
}
```


実行結果

```
Input integer number 8 times
x[0]:17
x[1]:39
x[2]:1
x[3]:9
x[4]:24
x[5]:16
x[6]:5
x[7]:3
Number to search:17
17 is located at 5
```

実行結果

```
 Input integer number 8 times
x[0]:17
x[1]:39
x[2]:1
x[3]:9
x[4]:24
x[5]:16
x[6]:5
x[7]:3
Number to search:2
Searching was failed!
```


演習問題6-1(講義時間内で実施)

- ハッシュ探索を行うプログラムのソースコードを入力し実行形式ファイルを作成する
  - メイン（線形探索のメインを流用）
  - データ格納および探索を行う関数
- データを入力し、実行結果を確認する

ハッシュ探索における比較回数

探索における比較回数:

データ数nでハッシュ関数による格納場所がmの場合

- 最小：ハッシュ関数によって均等に配置 (1回)
- 最大：ハッシュ関数によって1箇所に集中(n回)
- 平均:　O(m/(m-n))

∵

オーダ：最小・平均 O(1),　最大O(n)
時間計算量はハッシュ関数に依存するが、一般的には非常に高速でありデータサイズに依存しない

まとめ（探索）

	最良の場合	最悪の場合	平均的の場合
線形探索	O(1)	O(n)	O(n)
２分探索	$O(\log_2 n)$	$O(\log_2 n)$	$O(\log_2 n)$
ハッシュ法	O(1)	O(n)	O(1)

課題問題２（レポート）

課題の目的：線形探索と2分探索の計算量の違いを理解する

(1)課題4-2同様にファイルからデータを読み込むように課題5-１のプログラムを修正する。

(2)テキストファイルからデータを入力する。（各ファイルにはそれぞれ1000個のデータが入っている）
test1.txt,test2.txt,test3.txt,test4.txt

(3)2分探索を適用する場合はデータを昇順に整列処理を行う。整列処理はどの手法を使用してもかまわない。

(4)線形探索と２分探索で500を探索するのに必要な比較の回数(データとキー)を求める。

(5)線形探索と2分探索についてオーダの確認および考察を実施する。

- レポートを作成し次週(5/28)に提出すること。
- レポートのフォーマットは自由とするが作成したプログラムのソースコードと実行結果(比較の回数を表示)を添付すること。

小テストの実施について

- 来週(5/28)の授業で小テストを実施
- 内容:アルゴリズム論1〜6の範囲
- 教科書や授業資料は参照できない