教科書輪講

第5章

2017/5/22

秋山研 M1　林 孝紀

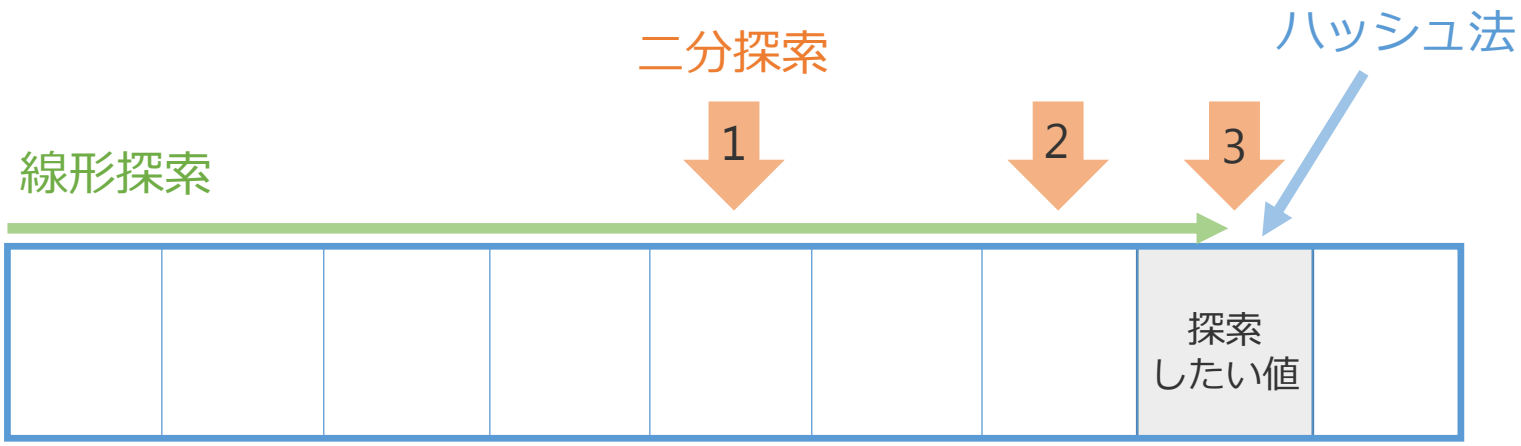
目次

1. 概要
2. 線形探索
3. 二分探索
4. ハッシュ
5. 最適解の計算

概要

1. 線形探索：
   右から順番に探索していく方法 ：$O(n)$
2. 二分探索：
   ソートされた配列に対して，半分ずつ探索：$O(\log n)$
   なお，ソートは平均 $O(n \log n)$
3. ハッシュ法：
   キーとデータを1対1で対応させて格納：$O(1)$

n回探索を行う場合，
線形探索は$O(n^2)$だが
二分探索は$O(nlogn)$



線形探索

- 前から順に探索を行う
- 番兵を設置することにより，比較回数が1ループあたり2回から1回に削減できる

長さlengthの配列aに対してkeyを線形探索するコード

1. 素朴な実装

```
int serch_liner(const int* a,
const int length, const int key){
  int i;
  for(i = 0; i < length; i++){
    if(a[i] == key) return 1;
  }
  return 0;
}
```

2. 番兵を用いた実装

```
int search(int* a, const int length, const int key){
  //番兵の配置は引数に破壊的操作を行う
  int i = 0;
  a[length] = key; //番兵の設置
  while(a[i] != key) i++;
  return (i != length);
}
```

二分探索

例題：

n 個の整数を含む数列 S と、q 個の異なる整数を含む数列 T を読み込み、T に含まれる整数の中で S に含まれるものの個数 C を出力するプログラムを作成してください。

入力 1行目に n、2行目に S を表す n 個の整数、3行目に q、4行目に T を表す q 個の整数が与えられます。

出力 C を1行に出力してください。

制約
- S の要素は昇順に整列されている
- $n \leq 100{,}000$
- $q \leq 50{,}000$
- $0 \leq S$ の要素 $\leq 10^9$
- $0 \leq T$ の要素 $\leq 10^9$
- T の要素は互いに異なる

入力例

```
5
1 2 3 4 5 →S
3
3 4 1 →T
```

出力例

```
3
```

二分探索

```
int binary_search(const int* a,
const int length, const int key){
  int left = 0;
  int right = length - 1;
  int mid;
  while(right > left){
    mid = (left + right) / 2;
    if(a[mid] == key) return 1;
    if(key > a[mid]){
      left = mid + 1;
    }else{
      right = mid - 1;
    }
  }
  return 0;
}
```

閉区間を使用した探索
できる人はここを半開区間にしてみてください

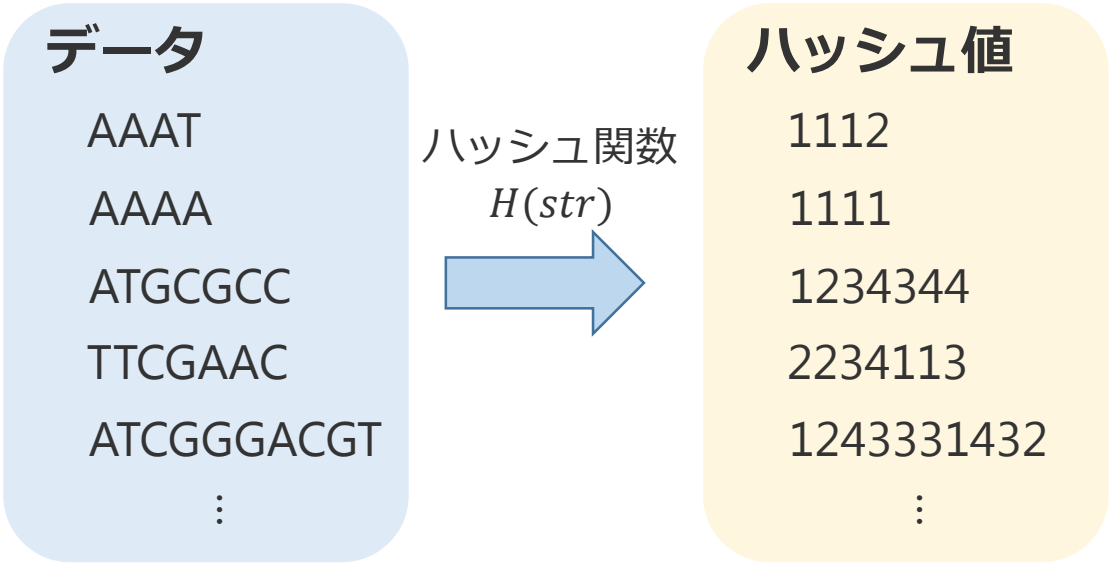
再帰のbase

再帰を使った探索

```
int binary_search_rev(const int* a, const int left,
            const int right, const int key){
  if(left == right) return (a[left] == key);
  int mid = (left + right) / 2;
  if(a[mid] == key) return 1;
  if(key > a[mid]){
    return binary_search_rev(a, mid+1, right, key);
  }else{
    return binary_search_rev(a, left, mid-1, key);
  }
}
```

ハッシュの実装

ハッシュの考え方：



ハッシュテーブルのサイズを m として

$$0 \leq H(str) < m$$

そのため，ハッシュ関数$H(str)$の設計次第では，ハッシュ値が衝突してしまうことも…

衝突を考慮したハッシュ関数

kを整数とした場合のハッシュ関数の設計：

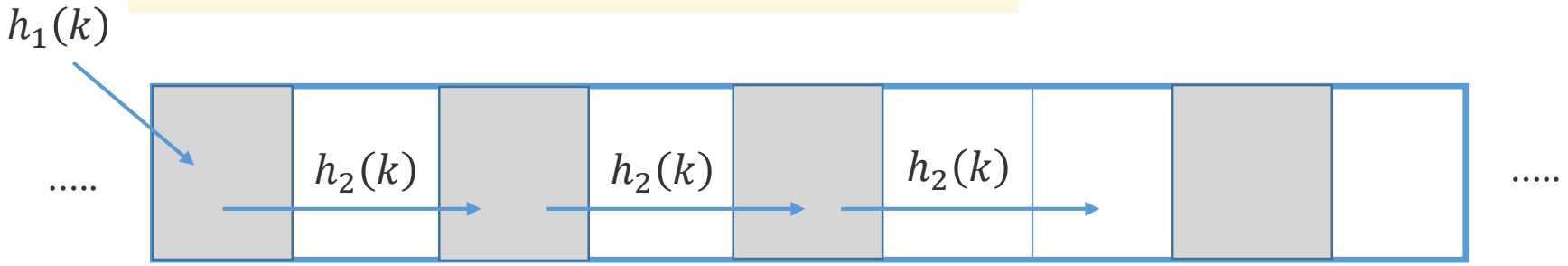
$$H(k) = h(k,i) = \left(h_1(k) + i * h_2(k)\right) mod\ m$$

$where$:

i: ハッシュが衝突した回数
m: ハッシュテーブルのサイズ
$h_1(k) = k\ mod\ m$
$h_2(k) = 1 + (k\ mod\ (m-1))$

これを用いて$h(k,i)$としたキーが衝突した場合は$h(k,i+1)$ を計算すれば良い

ここで$h_2(k)$ と m は互いに素
（m：素数，$0 \leq h_2(k) < m$）



前のスライドのように文字列をキーとするためには，
文字列から整数を作成する関数が別途必要

$F(str) = H(f(str))$ とするような $f(str)$ を考えてみよう！

ハッシュ

課題１：

以下の命令を実行する簡易的な「辞書」を実装してください。

- insert *str*: 辞書に文字列 *str* を追加する。
- find *str*: その時点で辞書に *str* が含まれる場合 'yes' と、含まれない場合 'no' と出力する。

入力 最初の行に命令の数 *n* が与えられます。続く *n* 行に *n* 件の命令が順番に与えられます。命令の形式は上記のとおりです。

出力 各 find 命令について、yes または no を 1 行に出力してください。

制約 与えられる文字列は、'A', 'C', 'G', 'T' の 4 種類の文字から構成される。
$1 \leq$ 文字列の長さ ≤ 12
$n \leq 1{,}000{,}000$

入力例

```
6
insert AAA
insert AAC
find AAA
find CCC
insert CCC
find CCC
```

出力例

```
yes
no
yes
```

＊回答はC言語を使用

ハッシュの実装

（趣味もかねて）プレーンなC言語で実装してみた

ハッシュの定義

```
typedef char* string;

typedef struct {
  string dict[HASH_SIZE];
} hash_map;
```

ハッシュの初期化

```
#define INIT ""

for(i = 0; i < HASH_SIZE; i++){
    map.dict[i] = (string)
              malloc(sizeof(char) * MAX_LEN);
    strcpy(map.dict[i], INIT);
  }
```

A, T, G, Cに1, 2, 3, 4
を対応させて5進数表記

```
long get_key(const string target){
  int i;
  long result = 0;
  int length = (int)strlen(target);
  if(length > MAX_LEN) return -1;
  for(i = 0; i < length; i++){
    char c = target[i];
    result = result << 3;
    if(c == 'A'){
      result += 1;
    }else if(c == 'C'){
      result += 2;
    }else if(c == 'G'){
      result += 3;
    }else if(c == 'T'){
      result += 4;
    }
  }
  return result; }
```

文字列 -> 整数の変換

ハッシュの実装

```
int insert(const string target, hash_map* map){
  long key = get_key(target);
  if(key == -1) return 0;
  int point, i = 0;
  while(1){
    point = ((h1(key) + i * h2(key))) % HASH_SIZE;
    string tmp = map -> dict[point];
    //already exist
    if(!strcmp(tmp, target)) return 0;
    if(!strcmp(tmp, INIT)){
      //insert target
      strcpy((map -> dict[point]), target);
      return 1;
    }
    //hash map is full
    if((++i) > HASH_SIZE) return 0;
  }
}
```

キーの計算，関数化すべき

```
int find(const string target,
                const hash_map* map){
  long key = get_key(target);
  if(key == -1) return 0;
  int point, i = 0;
  while(1){
    point = ((h1(key) + i * h2(key))) % HASH_SIZE;
    string tmp = map -> dict[point];
    if(!strcmp(tmp, target)) return 1;
    if((!strcmp(tmp, INIT)) || ((++i) > HASH_SIZE))
      return 0;
  }
}
```

pointに文字が格納されていて
targetではない場合次のループ

応用問題

課題２：

重さがそれぞれ$w_i(i = 0, 1, ...n - 1)$のn個の荷物が、ベルトコンベアから順番に流れてきます。これらの荷物をk台のトラックに積みます。各トラックには連続する0個以上の荷物を積むことができますが、それらの重さの和がトラックの最大積載量Pを超えてはなりません。最大積載量Pはすべてのトラックで共通です。

n、k、w_iが与えられるので、すべての荷物を積むために必要な最大積載量Pの最小値を求めるプログラムを作成してください。

入力 最初の行に整数nと整数kが空白区切りで与えられます。続くn行にn個の整数w_iがそれぞれ1行に与えられます。

出力 Pの最小値を1行に出力してください。

制約 $1 \leq n \leq 100,000$
$1 \leq k \leq 100,000$
$1 \leq w_i \leq 10,000$

＊回答はC++を利用

入力例

```
5 3
8
1
7
3
9
```

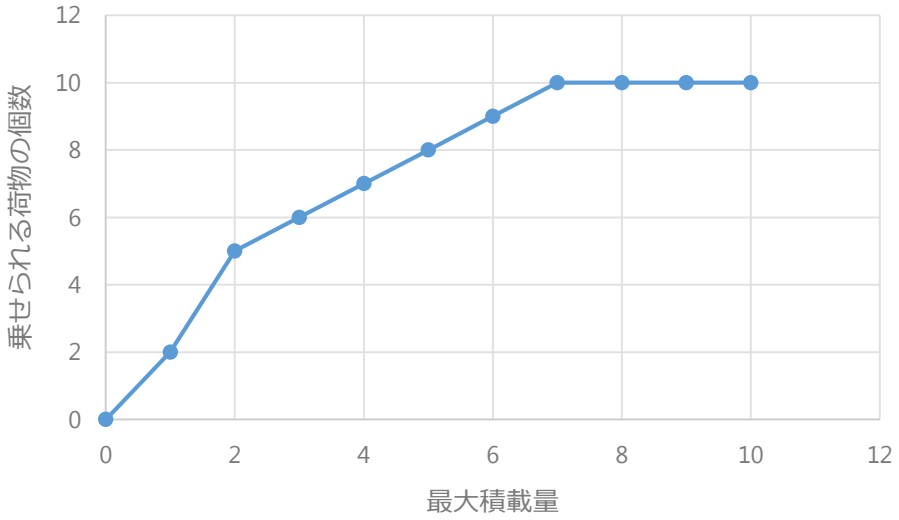
出力例

```
10
```

1台目のトラックに2つの荷物{8, 1}, 2台目のトラックに2つの荷物{7, 3}、3台目のトラックに1つの荷物{9}を積んで、最大積載量の最小値が10となります。

二分探索の応用

載せることができる荷物の個数と
最大積載量を考えるとこれは単調増加
かつ最大積載量は離散値



二分探索を利用できる

最大積載量から荷物の個数を計算

```
typedef long long ll;

int check(ll p){
  int i = 0;
  for (int j = 0; j < track_num; j++){
    ll s = 0;
    while(s + weight[i] <= p){
      s += weight[i++];
      if(i == weight_num) return weight_num;
    }
  }
  return i;
}
```

i個目の荷物を扱う

トラックに入れられるだけ入れる

二分探索の応用

```
int solve(){
  ll left = 0;
  ll right = (ll) MAX * (ll) MAX;
  ll mid;
  while(right - left > 1){
    mid = (right + left) / 2;
    int v = check(mid);
    if(v >= weight_num) right = mid;
    else left = mid;
  }
  return right;
}
```

(right = left+1)まで二分探索

今回はweight_num以下の数は返ってこないことに注意

二分探索：

1. 条件を満たす最小の要素の探索
2. 条件を満たす最大の要素の探索
3. 条件を満たす要素があるかないかの探索

解の区間設定はここに詳しく書いてある
http://topcoder.g.hatena.ne.jp/agw/20151215