アルゴリズムとデータ構造
第3回: 探索問題(1)

担当: 上原隆平 (uehara)

2014/04/15

探索問題

探索問題

- 問題: S をデータの集合とする．任意のデータ x が与えられたとき，x が S に含まれているかどうかを効率よく判定せよ

- 効率の良さ/悪さ: 最悪の場合の計算時間を集合 S のサイズ $n = |S|$ で評価
  - 今回は全ての要素を見ればよいから，
    最悪でも計算時間は $|S|$（に比例する時間）

解き方

- データ構造・配置方法を考える
  - 配列に適当に入れる
  - 配列に昇順に入れる
- 探索アルゴリズム: 計算の仕方を考える
  - 逐次探索法
  - m-ブロック法
  - 2重m-ブロック法

データ構造1
配列に適当に要素を入れる

- 集合Sの要素を1次元配列sの0番目からn-1番目までに適当な順序で蓄える.

s[ ]=	37	12	25	9	87	33	65	3	29

逐次探索法

- 入力: 任意の実数x
- 出力:
  - もし s[i] == x となる i があれば, i を出力
  - そうでないなら -1 を出力

```
for (i=0; i<n; ++i)
   if(x==s[i]) return i;
return -1;
```

最悪の場合 n 回の比較が必要. よって計算時間は n に比例. $\rightarrow \mathrm{O}(n)$

逐次探索法の計算量

- 高々 3n + 2 ステップ

```
for (i=0; i<n; ++i)
   if(x==s[i]) return i;
return -1;
```

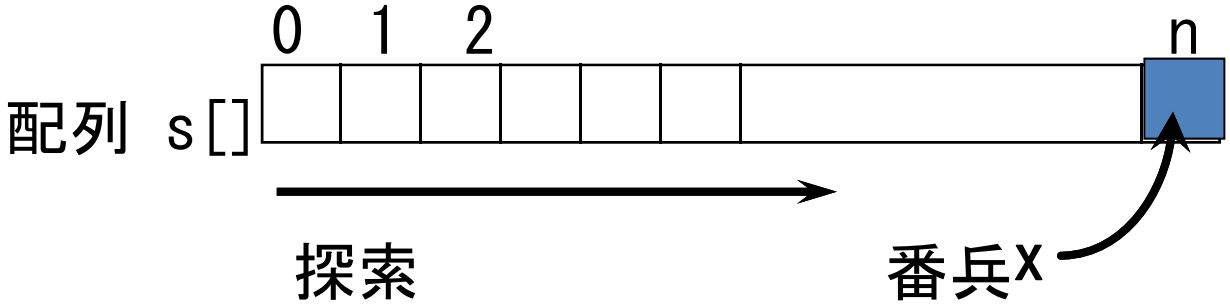
i の初期化 × 1回

ループ回数≦ nについて、
判定2回 (==, <)
インクリメント（++）1回

高々returnが1回

番兵を使ったアルゴリズムの整理・計算回数の削減

探索を始める前に配列の終りにx自身を番兵として置くと、
0<=i<=nで必ずx==s[i]が成り立つので、条件i<nが不要




```
s[n] = x;
i = 0;
while(x != s[i])
 i = i+1;
if(i < n) return i;
else      return -1;
```

番兵の設置

ループはこれだけ
➔ 2回の操作

高々2n+4回の操作
$=O(n)$

比較回数の解析

- 最良の場合: 1回
  - s[0] == x の場合
- 最悪の場合: n回
  - s[0]～s[n-1]が x でない場合
- 平均的な場合: $\sum_{i=1}^{n+1} \frac{i}{n} = \frac{n+2}{2}$
  - 比較回数の期待値を求める
  - i 番目の要素が比較の対象になる確率は1/n
  - i 番目で要素を発見したときの比較回数はi

```
s[n] = x;
i = 0;
while(x!=s[i])
 i = i+1;
if(i < n)
  return i;
else
  return -1;
```

コイン投げに基づく探索

- コインを投げて表が出れば配列の先頭から順に探索
- 裏が出れば配列の末尾から逆方向に探索

今までの方法では最悪の場合が続くことがあるが，今度はコインを投げているので，最悪の場合が連続する確率は低い．したがって，n/2回程度の比較回数で探索を終える可能性が高くなる．

乱択アルゴリズム

乱数に依存して動作が決まる．
最悪の場合が起こりにくいのが特徴．

データ構造2
配列に昇順に要素を入れる

- s[]=

3	9	12	25	29	33	37	65	87

- Q: 逐次探索法のアルゴリズムに
  改善の余地はある?

```
s[n]=x;
i = 0;
while(x!=s[i])
 i = i+1;
if(i < n) return i;
else      return -1;
```

xより大きい値になったら
探索は打ち切ってよい
x!=s[i] ➔ x>s[i]

データ構造2
配列に昇順に要素を入れる

- s[]=

3	9	12	25	29	33	37	65	87

- Q: 逐次探索法のアルゴリズムに
  改善の余地はある?

```
s[n]=x;
i = 0;
while(s[i]<x)
 i = i+1;
if(i < n) return i;
else      return -1;
```

xより大きい値になったら
探索は打ち切ってよい
x!=s[i] ➔ x>s[i]

データ構造2
配列に昇順に要素を入れる

- s[]=

3	9	12	25	29	33	37	65	87

- Q: 逐次探索法のアルゴリズムに
  改善の余地はある?

```
s[n]=x;
i = 0;
while(s[i]<x)
 i = i+1;
if(i < n) return i;
else      return -1;
```

xより大きい値になったら
探索は打ち切ってよい
x!=s[i] ➔ x>s[i]

i<nでも探索を打ち切っ
ている場合がある
i<n ➔ s[i]==x

データ構造2
配列に昇順に要素を入れる

- s[]=

3	9	12	25	29	33	37	65	87

- Q: 逐次探索法のアルゴリズムに
  改善の余地はある?

```
s[n]=x;
i = 0;
while(s[i]<x)
 i = i+1;
if(s[i]==x) return i;
else        return -1;
```

xより大きい値になったら
探索は打ち切ってよい
x!=s[i] ➔ x>s[i]

i<nでも探索を打ち切っ
ている場合がある
i<n ➔ s[i]==x

データ構造2
配列に昇順に要素を入れる

- s[]=

3	9	12	25	29	33	37	65	87

- Q: 逐[illegible]リズムに
  改[illegible]

要素が見つからなかったときに，nを返してしまう
s[n]=x ➔ s[n]=x+1

```
s[n]=x;
i = 0;
while(s[i]<x)
 i = i+1;
if(s[i]==x) return i;
else        return -1;
```

xより大きい値になったら探索は打ち切ってよい
x!=s[i] ➔ x>s[i]

i<nでも探索を打ち切っている場合がある
i<n ➔ s[i]==x

データ構造2
配列に昇順に要素を入れる

- s[]=

3	9	12	25	29	33	37	65	87

- Q: 逐 リズムに
  改

要素が見つからなかったときに，nを返してしまう
s[n]=x ➔ s[n]=x+1

```
s[n]=x+1;
i = 0;
while(s[i]<x)
 i = i+1;
if(s[i]==x) return i;
else        return -1;
```

xより大きい値になったら探索は打ち切ってよい
x!=s[i] ➔ x>s[i]

i<nでも探索を打ち切っている場合がある
i<n ➔ s[i]==x

データ構造2
配列に昇順に要素を入れる

- s[]=

3	9	12	25	29	33	37	65	87

  - ループ脱出条件: s[i]≧x
  - ループ脱出後の判定: s[i]==x
  - 番兵: xより大きい値、たとえばx+1

```
s[n]=x+1;
i = 0;
while(s[i]<x)
 i = i+1;
if(s[i]==x) return i;
else        return -1;
```

Q. 比較回数は改善?

A. 平均は改善。
だが最悪の場合は同じ

昇順に入れたときの
比較回数の改善@逐次探索法

（微調整１）

データの最小値は配列の先頭にあり、データの最大値は配列の最後尾にある。よって、xが最小値と最大値のどちらに近いかによって探索の方向を決めるのもよい。

➔ 依然として最悪の場合には n-1 の比較が必要

（微調整２）

配列の中央の値 s[n/2] と最初に比較を行ない、これより大きい場合は右へ、そうでなければ左へ探索する。

➔ 比較回数は高々 n/2 回となるが$O(n)$には変わりない

アルゴリズム2: mブロック法

mブロック法の考え方

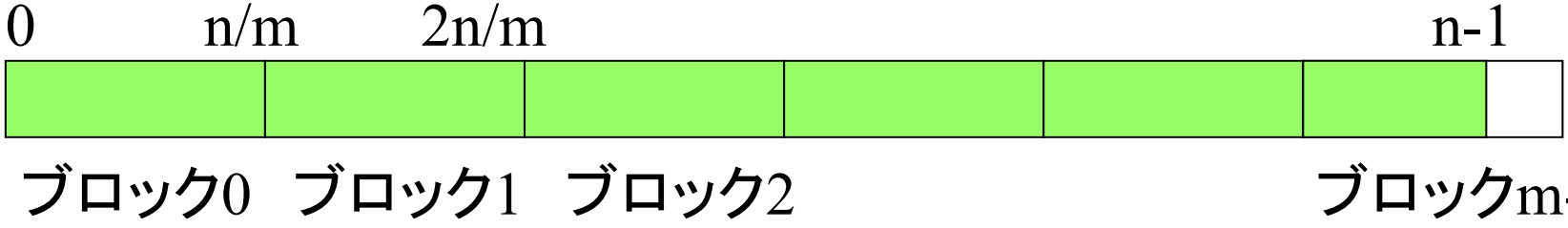
(0)ソートされた配列全体をm個のブロック$B_0, B_1, \ldots, B_{m-1}$に分割.
(1)各ブロックの最大値と比較することにより，質問xを含みうる
　　ブロックB_jを求める.
(2)ブロックB_jの中を逐次探索する.

アルゴリズム2: mブロック法

mブロック法の考え方

(0)ソートされた配列全体をm個のブロック$B_0, B_1, \ldots, B_{m-1}$に分割.
(1)各ブロックの最大値と比較することにより，質問xを含みうる
　ブロックB_jを求める.
(2)ブロックB_jの中を逐次探索する.

最も簡単な実現方法は，各ブロックの長さを同じにすること.
ただし，最後のブロックだけは長さが異なる.



- ブロック長を k とすると k = roundup(n/m)
- ブロック B_j は配列の jk 番目から (j+1)k-1番目: $B_j = [jk, (j+1)k-1]$

アルゴリズム2: mブロック法

mブロック法の考え方

(0)ソートされた配列全体をm個のブロック$B_0, B_1, \ldots, B_{m-1}$に分割.
(1)各ブロックの最大値と比較することにより，質問xを含みうる
　ブロックB_jを求める.
(2)ブロックB_jの中を逐次探索する.

```
j=0;
while(j<=m-2)
  if x<=s[(j+1)*k-1] then ループから出る
  else j=j+1;
```

途中でループを出たときはそのときのjの値がブロックを指す。
そうでないときは最後のブロックが対象となる。

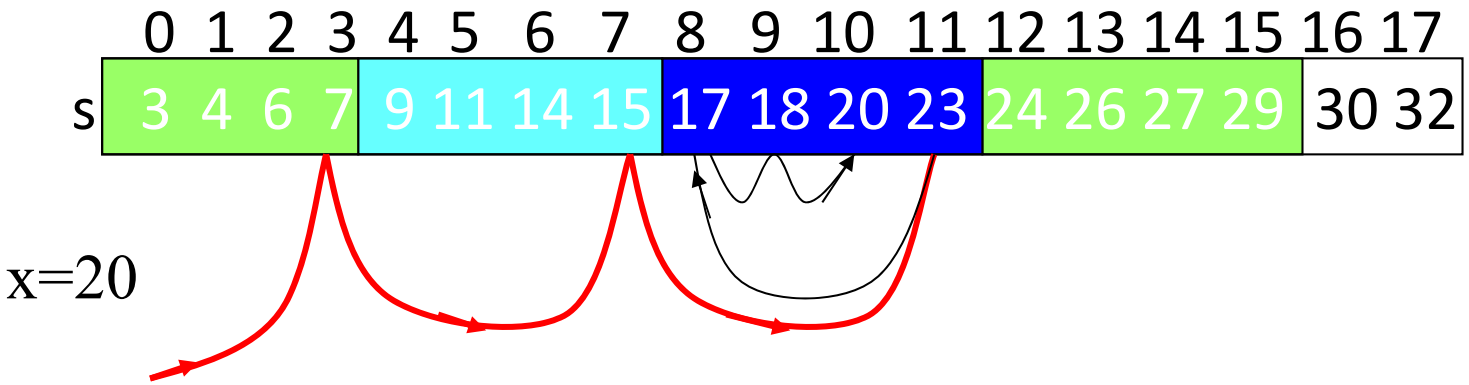
アルゴリズム2: mブロック法

mブロック法の考え方

(0)ソートされた配列全体をm個のブロック$B_0, B_1, \ldots, B_{m-1}$に分割.
(1)各ブロックの最大値と比較することにより，質問xを含みうる
　ブロックB_jを求める.
(2)ブロックB_jの中を逐次探索する.

```
i=j*k; t = min{ (j+1)*k-1, n-1 };
while( i < t )
  if x≧s[i] then ループから出る;
  else i=i+1;  //ブロック内の次の要素へ
if x == s[i] then iを返して終了;
else  -1を返して終了;
```

探索例と計算時間



- 比較回数≦ブロック数＋ブロック長 = m + n/m
- m + n/m を最小にする m の値は？
  - f(m) = m + n/m とおいて，mに関して偏微分する
  - $f'(m) = 1 - n/m^2 = 0 \rightarrow m = \sqrt{n}$
  - m = √n のとき，比較回数≦ √n + n/√n = 2 √n
- 計算時間: O(√n)

mブロック法の検討

- ブロックの長さは同じでないといけないか？

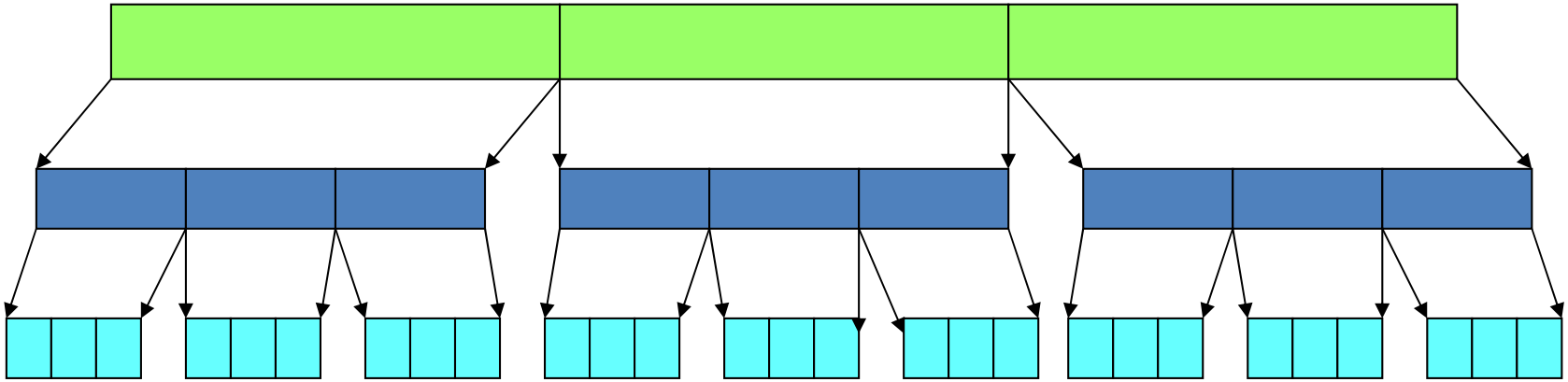
（**観察**）
比較回数 ＝ 調べたブロックの数 ＋ ブロック内での比較

最初の方のブロックで見つかれば、ブロック内での探索に時間がかかっても大丈夫
➔ブロックの長さを徐々に減らす
・たとえば、1ずつ減らしていく
・ブロック番号＋ブロックの長さ＝一定、とする

アルゴリズム3: 2重mブロック法

mブロック法では、ブロック内の探索に逐次探索を利用

⇒ ブロック内も同じ方法で探索



二重mブロック法の原理

探索区間をm個のブロックに分割し、
xを含むブロックに対して、同じ探索を再帰的に適用。
これを、ブロック長が定数N以下になるまで行なう。

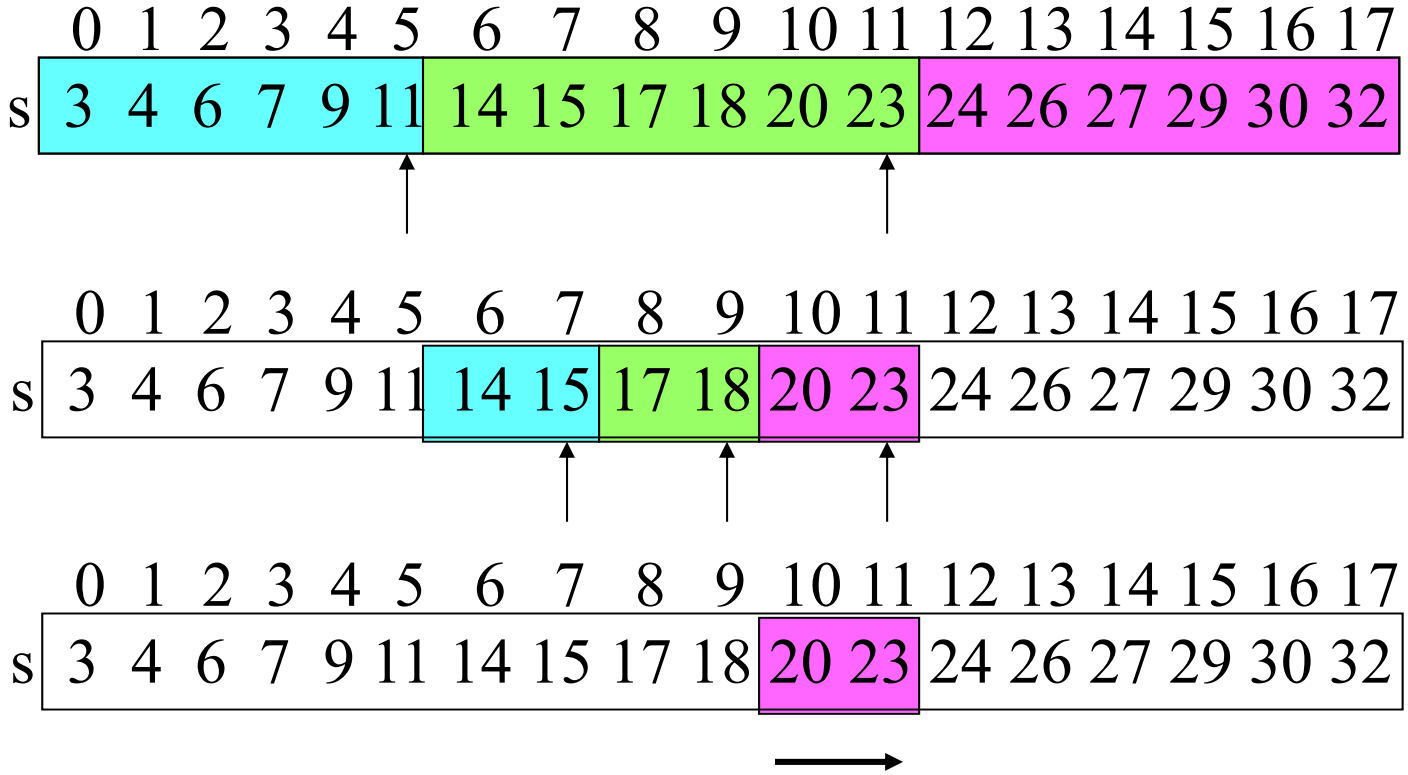
```
double-m-block-search(int left, int right) {
    区間長 L = right − left + 1
    if  L > 区間長の最小値 Lmin  then
        k = L/m;
        for  j = 0  to  m−2  do
            if  x ≦ s[left + (j+1)k − 1]  then  ループ脱出;
        endfor
        f = left + jk;  t = min{left + (j+1)k − 1, n−1};
        double-m-block-search(f, t);
    else
        i = left;
        while (i < right)
            if  x ≦ s[i]  then  ループから出る  else i = i + 1;
        if  x  ==  s[i]  then  return i;
        else  return −1;
    endif
}
```

ブロック長(切り捨て)

再帰

区間が十分短いときは
ブロック内を逐次探索

探索例: 20を探す (x=20)
ブロック数を3とした場合



計算時間の解析

- 探索区間の長さの変化

$$n \to \left\lceil \frac{n}{m} \right\rceil \to \left\lceil \frac{\left\lceil \frac{n}{m} \right\rceil}{m} \right\rceil \to \left\lceil \frac{\left\lceil \frac{\left\lceil \frac{n}{m} \right\rceil}{m} \right\rceil}{m} \right\rceil \to \cdots$$

- i 回目の呼出後の探索区間長を n_i とする.

$$n_1 = \left\lceil \frac{n}{m} \right\rceil \leq \frac{n}{m} + 1$$

$$n_2 = \left\lceil \frac{n_1}{m} \right\rceil \leq \frac{n}{m^2} + \frac{1}{m} + 1$$

$$\cdots$$

$$n_i \leq \frac{n}{m^i} + \sum_{j=0}^{i-1} \frac{1}{m^j} \leq \frac{n}{m^i} + 2$$

計算時間の解析

- i 回目の呼出後の探索区間長 $n_i \leqq n/m^i + 2$
- 何段階再帰すればよい？

$$n_i \leq \mathrm{Lmin} \impliedby \mathrm{Lmin} \geq \frac{n}{m^i} + 2 \iff i \geq \log_m \frac{n}{\mathrm{Lmin} - 2}$$

- 1回の呼出で最大m-1回の比較を行うから
  全比較回数 $\leq (m-1)\log_m \frac{n}{\mathrm{Lmin}-2} + \mathrm{Lmin}$
- 計算時間はO(log *n*)

計算時間の解析: 最適なmの値

- $T(n,m) = (m-1)\log_m \frac{n}{Lmin-2} + Lmin$

$$= \frac{m-1}{\log_2 m}\log_2 \frac{n}{Lmin-2} + Lmin$$

- T(n,m)を最小にするには m が小さいほどよい
  - m-1 の方が $\log_2 m$ より増え方が大きい
- よって m=2 が最適

今日はオフィスアワーに補講します.

ミニ演習

- 配列 a: a[i] = 3i−1 (0 <= i < 487)
  - a[0] = −1, a[1] = 2, a[2] = 5, …, a[486] = 1457

1) mブロック法を使う場合，ブロックサイズは？
2) ブロックサイズを (1) で決めた値にしたとき，「749」を探査するときの比較回数は？