アルゴリズムとデータ構造

- 第10回講義トピック：探索 (searching)
  - 逐次探索
  - 2分探索法
  - 内挿探索法

探索 (searching)

- 探索とは与えられたデータ配列の中から、特定の条件に合うデータを探し出すことを指す。
- 具体的には与えられた探索キーと一致するキーをもつレコードを見つけ出し、レコード内の必要なデータを引き出す。

探索に使われるデータ構造の例

- 辞書 (dictionary)
  - 検索のkeyは単語である。
  - データは単語に付随する項目で、単語の定義、発音など他の情報を含む。
- コンパイラによって生じる記号表 (symbol table)
  - プログラムに現れる名前を見出しとする辞書のこと
  - 検索のkeyはプログラムに現れる変数名や関数名である。
  - データは名前の表す対象を記述する情報を含む項目となる。

探索に関連する基本操作

- 探索に関連のあるいくつかの基本操作
  - initialize:データ構造を初期設定する。
  - search:与えられたキーをもつレコードを探索する。
  - insert:新しいレコードを挿入する。
  - delete:指定されたレコードを削除する。
  - join:2つの辞書を合併して1つの辞書を作る。
  - sort:辞書を整列する。
- これからはこの中でのsearchとinsertを中心に分析を進めていく。

逐次探索 (sequential search)

- 整列されていないデータ列を対象とする。
- レコードを探索する時にはレコードを順番にチェックし、希望のレコードを探していく。
- 整列する必要がないので、新しいレコードを挿入する時に配列の最後尾に挿入するだけでよい。
- 実現法:
  - 配列による方法
  - 連結リストによる方法

配列によるプログラム例

```
static struct node {int key; int info; };
static struct node a[maxN+1];
static int N;
seqinitialize() {N = 0;}
int search(int v) {
  int x = N+1;
  a[0].key = v; a[0].info = -1;
  while (v != a[--x].key) ;
  return a[x].info;
}
seqinsert(int v, int info) {
  a[++N].key = v; a[N].info = info;
}
```

連結リストによる実現

- 逐次探索は必ずしもデータ列に対して整列することを要求しないが、データ列が整列されている場合は探しているキー以上の値のキーが見つかった時で探索を終了できるので、リストの最後まで見る必要がない利点を持つ。
- リスト構造を使えば、整列状態を保ったままデータの挿入や削除が容易にできるようになる。

リストによる逐次探索プログラム

```
static struct node {int key, info; struct node *next; };
static struct node *head, *z;
listinitialize() {
  head = (struct node *) malloc(sizeof *head);
  z = (struct node *) malloc(sizeof *z);
  head->next = z; z->next = z; z->info =-1;
}
int listsearch(int v) {
  struct node *t = head; z->key = v;
  while (v > t->key) t = t->next;
  if (v != t->key) return z->info;
  return t->info;
}
```

リストによる逐次探索プログラム

```
listinsert(int v, int info) {
  struct node *x, *t = head;
  z->key = v;
  while (v > t->next->key) t = t->next;
  x = (struct node *) malloc(sizeof *x);
  x->next = t->next; t->next =x;
  x->key = v; x->info = info;
}
```

逐次探索の計算量

- 配列による実現
  - 不成功探索
    - N+1回の比較
  - 成功探索
    - 平均約N/2回の比較
- 整列されたリストによる実現
  - 成功、不成功とも
    - 平均約N/2回の比較

2分探索法 (binary search)

- 2分探索法は整列されたデータ列を対象とする。
- 探索データ列のkeyとの比較を逐次に行わないで、いきなり中間位置にあるレコードと比較する。
- 比較の結果によって分割統治法を適用する。
  - 探す対象のkeyと一致する場合は探索を終了する。
  - 中間レコードのkeyが探索対象keyの値よりも小さい場合は右の部分列をさらに2分探索する。
  - 中間レコードのkeyが探索対象keyの値よりも大きい場合は左の部分列をさらに2分探索する。

2分探索のプログラム（非再帰）

```
int binarysearch(int v) {
  int l = 1, r = N, m;
  while (r >= l) {
    m = (l+r)/2;
    if (v < a[m].key) r = m-1;
    else              l = m+1;
      if (v == a[m].key) return a[m].info;
  }
  return -1;
}
```

2分探索の例

- キーMを探す様子

A A A C E E E G H I L M N P R S X

I L M N P R S X

I L M

M

内挿探索 (interpolation search)

- 内挿探索は2分探索と似ているが、中間位置のレコードと比較する変わりに、内挿を行う。
- つまり、中間位置は以下の式で与えられるが

  m = (r+l)/2 = l + (r-l)/2

  データの値によって探索keyの値に一番近いレコードを線形予測し、そのレコードの位置を中間位置の代わりに用いる。

  m = l + (v-a[l].key)(r-l)/(a[r].key-a[l].key)
- 内挿探索の条件
  - 上の式の分母が0ではないこと、
  - mの値がデータ列の範囲を超えないこと

内挿探索の例

- キーMを探す様子

A A A C E E E G H I L M N P R S X

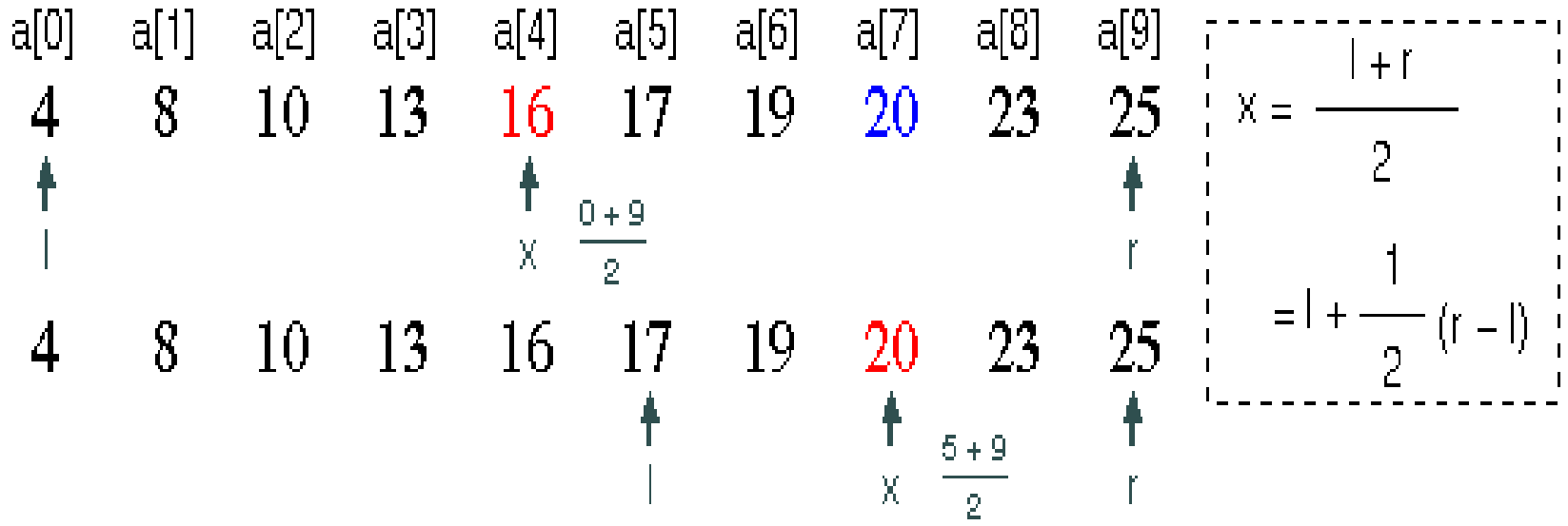
I L M N P R S X

M N P R S X

2分探索と内挿探索の比較

- 両方とも整列されているデータ配列に対して行う探索方法である。
- 2分探索は中間位置のデータが探したいデータかどうか調べ、探したいデータより小さければ左側に対して、大きければ右側に対して、同じことを繰り返す。
- 内挿探索は中間位置のデータのかわりに、内挿によるデータ予測を行う。2分探索よりも速く探したいデータを見つける可能性がある。

2分探索の例


a[0] a[1] a[2] a[3] a[4] a[5] a[6] a[7] a[8] a[9]
4 8 10 13 16 17 19 20 23 25
l
x
0+9
2
r
4 8 10 13 16 17 19 20 23 25
l
x
5+9
2
r
x = l + r / 2
= l + 1 / 2 (r − l)

内挿探索の例

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]
4	8	10	13	16	17	19	20	23	25
↑						↑			↑
l						x			r
4	8	10	13	16	17	19	20	23	25
							↑↑		↑
							lx		r

$$0 + \frac{20 - 4}{25 - 4}(9 - 0)$$

$$7 + \frac{20 - 20}{25 - 20}(9 - 7)$$

$$x = l + \frac{v - a[l]}{a[r] - a[l]}(r - l)$$

$v = 20$

2分探索と内挿探索の計算量

- 2分探索
  - どんなデータセットに対しても
  - 成功、不成功とも、比較はlnN+1回以下
- 内挿探索
  - ランダムに選ばれるkeyにより構成されたデータセットに対して
  - 成功、不成功とも、比較は平均でlglgN+1回以下