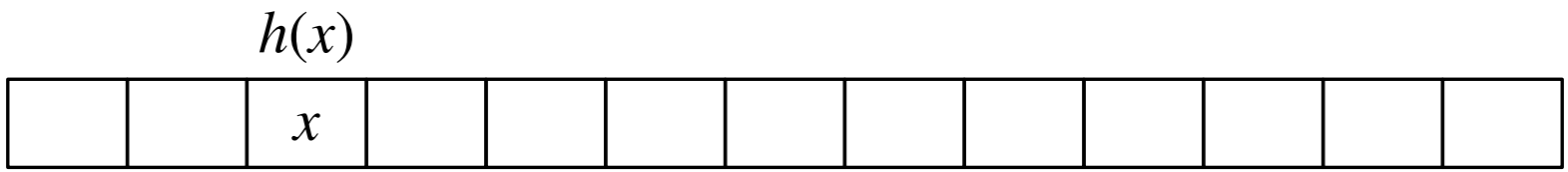
「アルゴリズムとデータ構造」資料 12
ハッシュ表

奈良女子大学
鴨浩靖

2012 年 1 月 17 日 初版
2020 年 12 月 24 日第二版

基本的な考え方

長さ N の配列 a[] と、N 未満の非負整数を返す関数 h を、用意しておく。
$h(x)$ を x のハッシュ値と呼ぶ。
データ x は a[$h(x)$] に格納する。



良いハッシュ関数の条件

- よく散らばること
- 速く計算できること

ハッシュ関数の実例

```
inline size_t
__stl_hash_string(const char* __s)
{
  unsigned long __h = 0;
  for ( ; *__s; ++__s)
    __h = 5 * __h + *__s;
  return (size_t)__h;
}
```

GNU ISO C++ Library のものを C に書き換え。
実際には、この関数の返値を配列の大きさで割った余りをハッシュ値として使う。

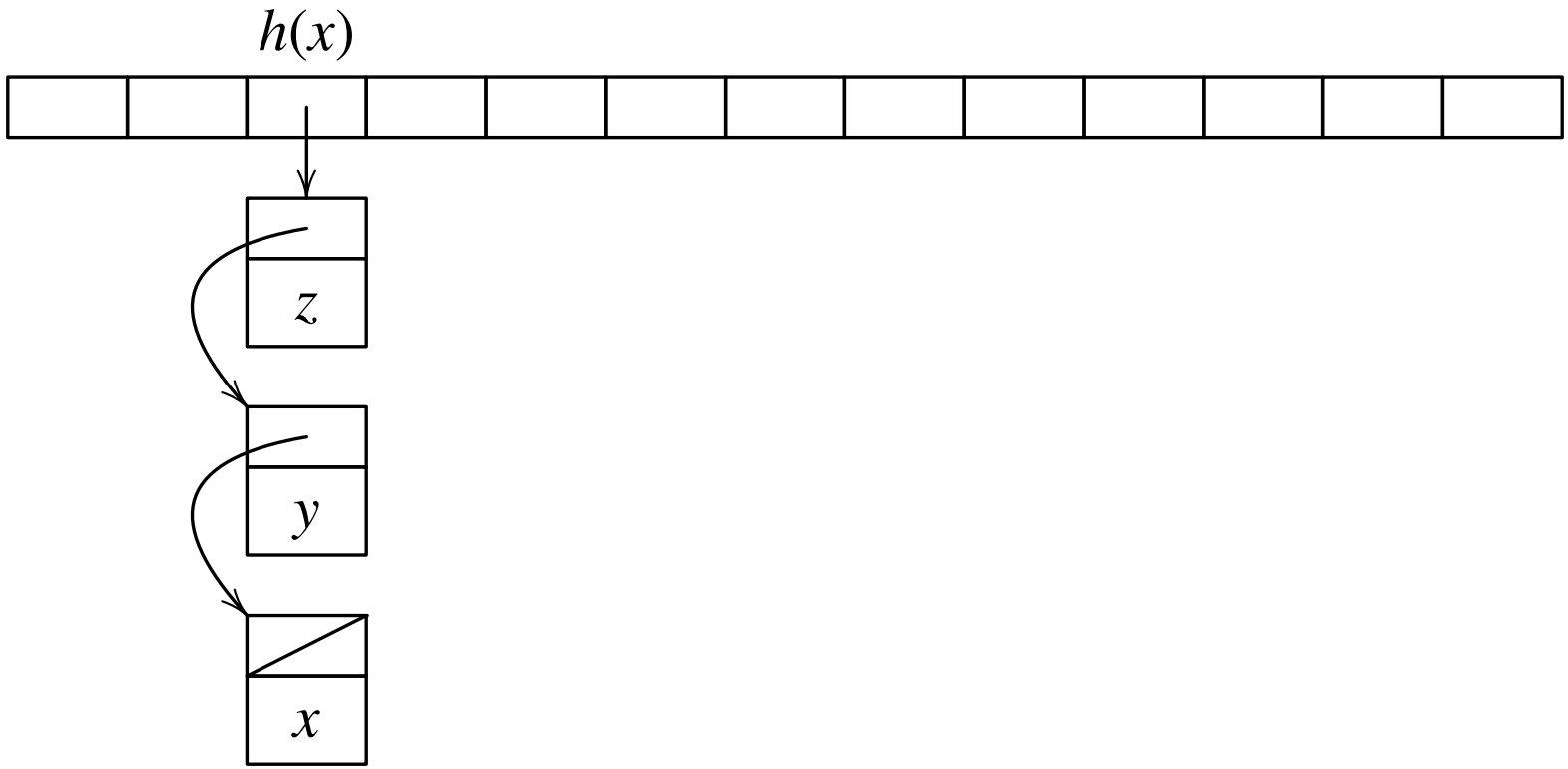
衝突とその回避

実際には、$x \neq x'$ なのに $h(x) = h(x')$ となることもある。
これをハッシュ値の衝突という。
対策が必要。

- 分離連鎖法
- 開番地法

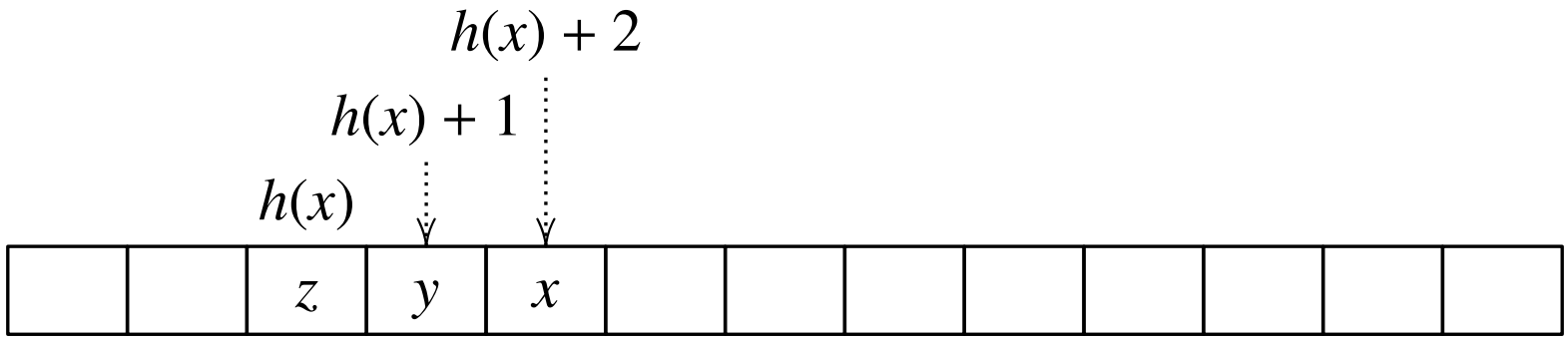
分離連鎖法

データの配列ではなく、データの線形リストの配列を使う。



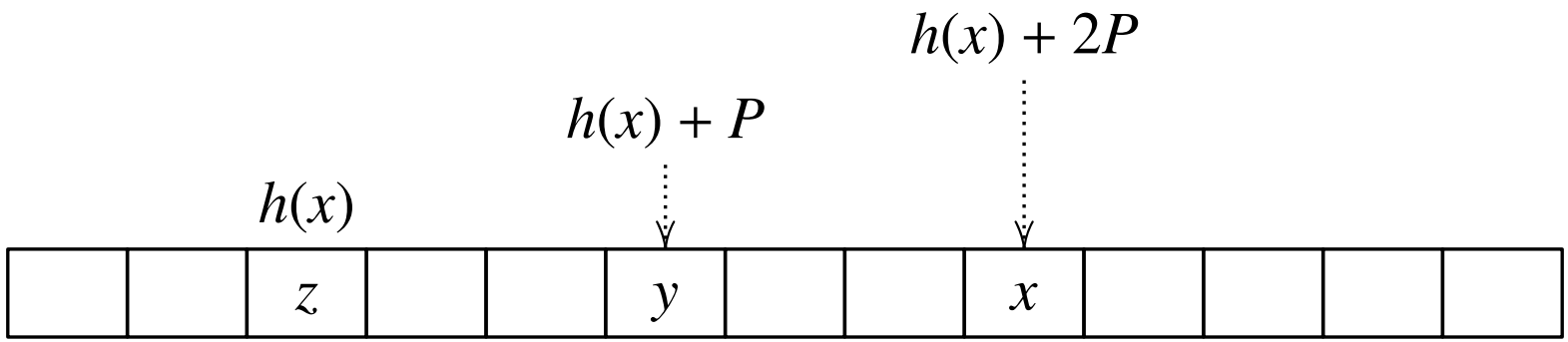
開番地法—線形探査法

a[$h(x)$] が使用済みならばその次を試す。次も使用済みなら、さらにその次を試す。これを、空きが見つかるまで繰り返す。
ただし、配列の末尾の次は先頭とする。



開番地法—線形探査法の変種

N と互いに素な整数 P を決めておく。
a[$h(x)$] が使用済みならば a[$(h(x)+P) \bmod N$] を試す。
a[$(h(x)+P) \bmod N$] も使用済みならば a[$(h(x)+2P) \bmod N$] を試す。これを、空きが見つかるまで繰り返す。



開番地法—二次ハッシュ法

もうひとつの関数 h_1 を用意しておく。$h_1(x)$ は常に配列の大きさ N と互いに素な整数値を取るものである必要がある。
a[$h(x)$] が使用済みならば a[$(h(x) + h_1(x)) \bmod N$] を試す。
a[$(h(x) + h_1(x)) \bmod N$] も使用済みならば
a[$(h(x) + 2h_1(x)) \bmod N$] を試す。これを、空きが見つかるまで繰り返す。

計算量

ハッシュ表の操作の時間計算量は、通常は以下の通り。

	平均	最悪
探索	$O(1)$	$O(n)$
追加	$O(1)$	$O(n)$
削除	$O(1)$	$O(n)$

ただし、配列が小さすぎたり、ハッシュ関数として非効率なものを選んだりすると、この限りでない。

削除について

分離連鎖法 ハッシュ表からの削除は線形リストからの削除で実現できるので、特に困難はない。

開番地法 削除した後を整合的に埋めるのは、著しく困難。
配列の各要素に、未使用/使用中/削除済み の三状態のマークをつけるのが簡単。
削除が多いと、「削除済み」が増えてメモリ効率が悪化する。さらに、探索時間も長くなる。

ハッシュ表の長所と短所

長所 早い。

- 探索・挿入・削除ともに、平均的には定数時間で処理できる。

短所 項目の個数に応じて、表の大きさを適切に設定しないと、不都合が生じる。

- 表が小さすぎると、ハッシュ値の衝突の頻度が上がり、極端に遅くなる。
- 表が大きすぎると、メモリの浪費となる。

まとめ

- ハッシュ法
  - 分離連鎖法
  - 開番地法