ハッシュ法

データの探索の基本的な手法として，教科書では線形探索，番兵法，２分探索が紹介されています．この中で最も効率よく探索を行える処理は２分探索ですが，探索に使用するデータはソート済みである必要があります．まずはソート済み配列の操作に伴う問題点について述べます．

ソート済み配列の操作

図１に示す要素数 13 の配列 x を考えます．先頭から 10 個の要素は昇順にソートされたデータです．

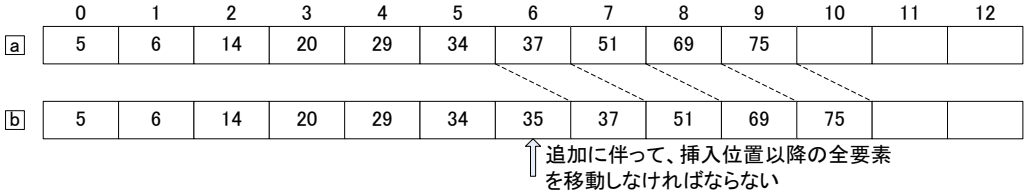

図１　ソート済み配列へのデータの追加

ソートを維持しつつ，データを新たに追加する場合には，適切な位置を探索によって見つけ，追加を行った後，挿入位置以降の全要素を移動する必要があります．データの追加や削除のたびに，このようなデータの移動を行う計算量は決して小さくありません．この問題に対応するハッシュ法と呼ばれる手法について，説明していきます．

ハッシュ法

探索だけではなく，データの追加・削除も効率よく行う手法として，ハッシュ法があります．ハッシュ法はデータにアクセスする際の目印としてハッシュ値を用います．以下の例では「キー値を配列の要素数 13 で割った際の剰余」をハッシュ値としています．

キー値	5	6	14	20	29	34	37	51	69	75
ハッシュ値(13で割った剰余)	5	6	1	7	3	8	11	12	4	10

図２　キー値とハッシュ値の対応例

ハッシュ値が添字となるように，キー値を格納した配列（表）がハッシュ表です．図３は図２のハッシュ値を用いたハッシュ表（データの格納結果）です． a の配列に 35 を追加した結果が b となっています．

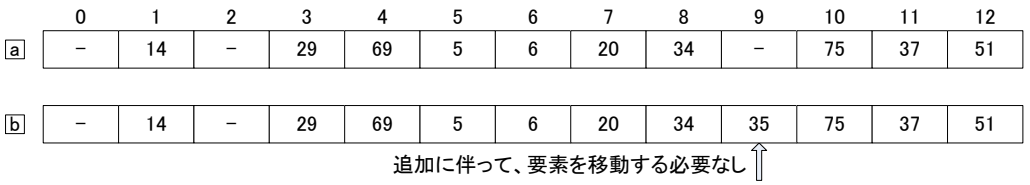

図３　ハッシュへの追加

この場合では，新しい要素の追加に伴って，図１に示した際のようなデータの移動がありません．

キー値からハッシュ値への変換を行う手続きをハッシュ関数と呼び，通常では配列の要素数を用いた剰余を求める演算，あるいは，それを応用した演算が使われます．ハッシュ表の各要素はバケットと呼ばれます．

衝突

図３のハッシュ表にさらに 18 を追加することを考えます．剰余は 5 ですが，格納先であるバケットの x[5]には既にデータがあります．キー値とハッシュ値の対応は１対１である保証はなく，通常は多対１です．格納すべきバケットが重複する現象は衝突と呼ばれます．

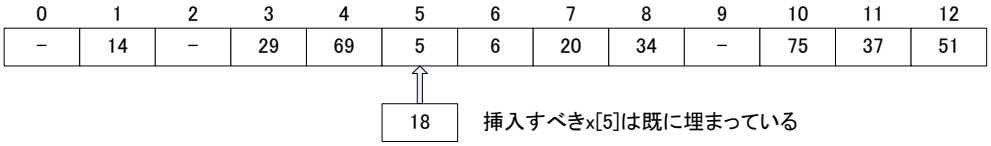

図４　衝突

衝突が発生した場合への対処には以下の２つがあります．

チェイン法　　：　同一のハッシュ値をもつ要素を線形リストによって管理する．
オープンアドレス法：　空きバケットを見つけるまで，ハッシュを繰り返す．

チェイン法（オープンハッシュ法）

チェイン法は図５に示すように同一のハッシュ値を持つデータを線形リストとして鎖状につなぎます．データの追加は各バケットのリストの先頭に追加し，削除はリストからのデータの削除として扱います（線形リストの項目を参照）．

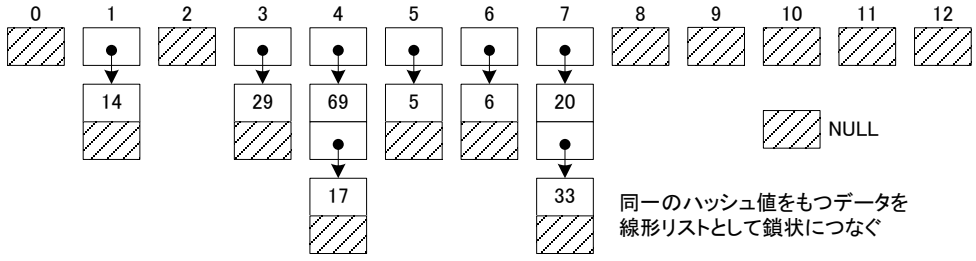

図５　チェイン法

オープンアドレス法（クローズドハッシュ法）

オープンアドレス法は衝突が発生した場合には再ハッシュを行い，格納先を求めなおします．再ハッシュの関数は自由に決めてよく，教科書ではハッシュ関数「キー値を配列の要素数 13 で割った際の剰余」

に対して，「キー値をインクリメントした値を配列の要素数 13 で割った際の剰余」を利用しています．衝突時には格納先が一つずつ後ろにずれることになります．

課題 7

ハッシュ法は【ハッシュ表が十分に大きい】場合，探索，挿入，削除の操作が O（1）で可能になります．チェイン法は衝突時に同一のハッシュ値を持つデータを線形リストとして連結します．そのため，線形リストが長くなった場合，探索やデータの削除の処理が重くなり，データ数はハッシュ表の数倍程度までが実用的とされています．オープンアドレス法についてはハッシュ表のサイズにより，衝突の発生頻度が変わり，衝突時には再ハッシュの処理（多くの場合はハッシュ表の配列の隣の領域に格納）を行います．ハッシュ表が小さい場合は，再ハッシュの処理が頻発し，性能が低下するため，データ数はハッシュ表の 80%～90%であれば実用的であるとされています．扱うプログラムの内容によっては，衝突を極力抑えるため，データ数をハッシュ表の 50%程度に抑える場合もあります．

以上の前提を踏まえ，定量的なデータを示しながら，考察を述べて下さい．チェイン法，オープンアドレス法のいずれかがあればよいです．可能な人は両方提出してください．