マップとハッシュ

山本昌志*

2006 年 1 月 30 日

1　本日の学習内容

1.1　前回の復習

ツリー構造を C 言語で実装する方法を学習した．ツリー構造のノードを示すポインターは，次のようになっていた．

```
typedef struct _tag_tree_node
{
    int value;
    struct _tag_tree_node *left;
    struct _tag_tree_node *right;
} tree_node;
```

このノードを用いて，ツリー構造を取り扱うためには，次の関数が必要であった．

- ノードの作成
- ノードの追加
- ノードのサーチ (探索)
- ノードの削除
- メモリーの解放

1.2　本日の学習内容

本日は教科書 [1] の第 7 章である．以下のことを学ぶ．

- 2 分木を用いたマップ
- ハッシュ法

*独立行政法人　秋田工業高等専門学校　電気情報工学科

– ハッシュ法の基本的な考え方
– ハッシュ表の意味
– ハッシュ関数の作り方
– ハッシュ値の衝突 (重複) が生じた場合の回避の仕方

2 解決したい問題

英単語のデータベースを作ることを考える．表 1 のようにスペルと和訳の組があり，探索 (サーチ) と削除，追加ができるようにする．もちろん，これはコンピューターのメモリー上で実現させるのである．

表 1: 単語データベース

スペル	和訳
orange	みかん
apple	リンゴ
peach	桃
pineapple	パイナップル
pear	梨
grape	ぶどう
⋮	⋮

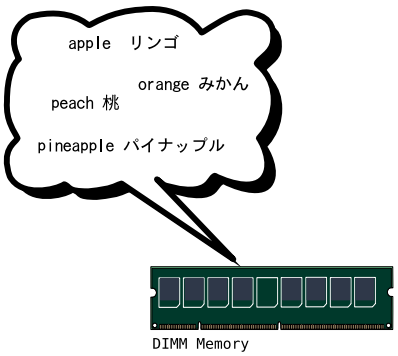

図 1: データはメモリーに入れる

単語は，スペルと和訳が組になっている．このような場合，構造体を使うのが常套手段である．複数の単語を表す単純なデータ構造として，教科書 [1] では以下の 2 つの方法が示されている．

- リスト
- 配列

リストを探索する場合，必ず先頭から順番に調べる．そのため，予めソートしても探索スピードは変わらない．したがって，ソートする必要がなく，データの追加はリストの末尾に接続することになる．このような場合，探索には $O(N)$，削除には $O(N)$, 追加には $O(N)$ の計算量が必要となる．実際，ソートすると実行スピードは変わるが，オーダーは変わらないことに注意しよう．

配列の場合，バイナリーサーチが使えるので，予めソートしておくと探索スピードを向上させることができる．しかし，データの追加と削除を行う場合，配列の入れ替えが必要となり，そこでの計算量が増加する．この場合，探索には $O(\log_2 N)$，削除には $O(N)$，追加には $O(N)$ の計算量が必要となる．ソートしない配列の場合，計算量のオーダーはリストと変わらない．

まとめると，表 2 のようになる．いずれにしても，データ数 N が大きくなると，かなりの計算量が必要となる．

表 2: 計算量のオーダー

方法	探索	削除	追加
リスト	$O(N)$	$O(N)$	$O(N)$
ソートされた配列	$\log_2 N$	$O(N)$	$O(N)$

リストや配列を使うと $O(N)$ のオーダーの計算量が必要になってくる．もう少し，スピードアップしたい場合，マップというものが使われる．通常は，

- 「2 分木」をつかった「ツリーマップ」
- 「ハッシュ表」をつかった「ハッシュマップ」

が使われる．

3　2 分木を使ったツリーマップ

以前学習した 2 分木を使ったツリーであれば，探索と追加，削除が $O(\log_2 N)$ 程度となり，リストやソートされた配列よりも高速である．そのためには，英単語のスペルの大小関係の比較が出来ることが条件で，C 言語では `strcmp()` という関数が用意されている．この関数は，次のようになっている．

```
#include <string.h>
int strcmp(char *s1, char *s2);
```

ヘッダーファイル `string.h` が必要で，ポインター `s1` と `s2` が示す文字列を比較する．文字列の大小は辞書順となる．比較の結果は，戻り値により表され，以下のようになる．

表 3: `strcomp` の戻り値

`s1 > s2`	正
`s1 = s2`	0
`s1 < s2`	負

後は今まで学習したとおりであるが，もう一度，C 言語のプログラムの書き方の復習をしておく．まずは，構造体であるが，次のように定義する．

```
typedef struct tag_word{
  char *english;
  char *japanese;
  struct tag_word *left;
  struct tag_word *right;
}word_node;
```

メモリーの確保とデータの代入とデータの取り出しは，次のようにする．

```
word_node *cc;

cc=(word_node *)malloc(sizeof(word_node));

cc->english="apple";
cc->japanese="リンゴ";
cc->left=NULL;
cc->right=NULL;

printf("%s\t%s\n",cc->english,cc->japanese);
```

後は，学習したとおりで，2 分木で表すと図 2 のようになる．

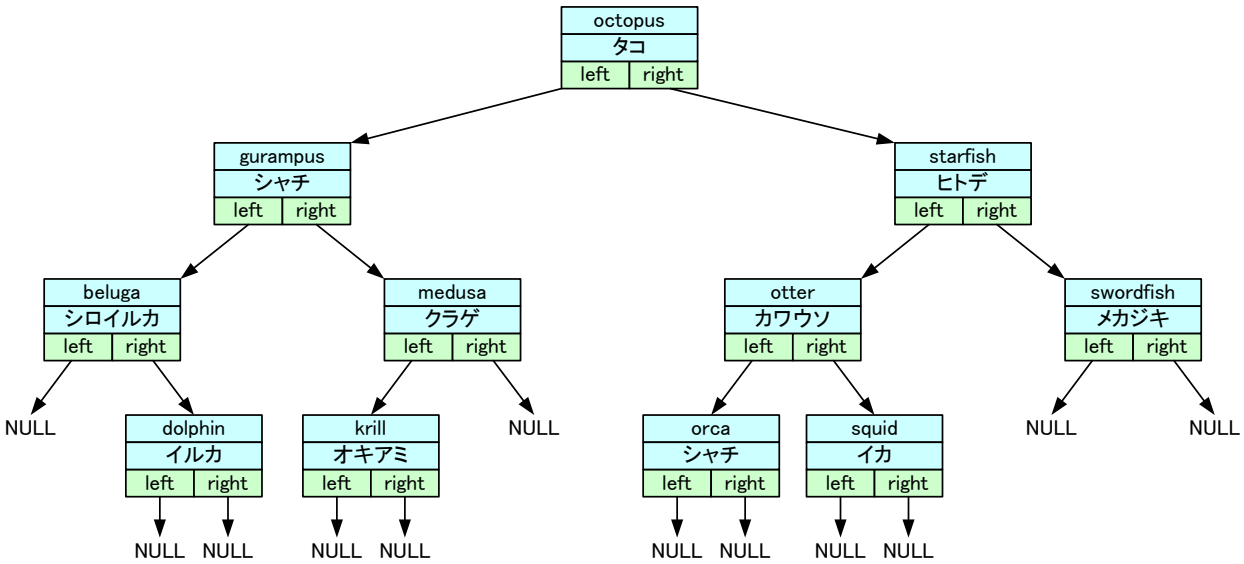

図 2: 2 分木を用いたマップ．アルファベット順にソートしている．

4 ハッシュマップ

4.1 基本的な考え方

単語のスペルから直に配列の添え字の値が導くことが出来れば，$O(1)$ の計算量で済む．これはとても高速である．たとえば，表 1 に示した単語データベースのようなものを配列に格納することを考える．ただし，以下の条件を課す．

- スペルが同じで意味の異なる単語は無いものとする．
- スペルは最大，アルファベットで 9 文字とする．

次のようにすれば，スペルから配列の値を求めることができる．

- アルファベットに対して，整数を割り当てる．aが 1，bが 2，cが 3,$\cdots$,zが 26 のようにである．もうひとつアルファベット長い場合 (空白) が必要で，それは 0 を割り当てる．このようにすれば，英単語は整数 $(a_1a_2a_3a_4a_5a_6a_7a_8a_9)$ で表現できる．

  apple→(1,16,16,12,5,0,0,0,0)　　pineapple→(16,9,14,5,1,16,16,12,5)

- これを 9 次元配列に格納することもできるが，少し工夫して 1 次元配列 a[i] に格納することを考える．つぎの演算により，添え字 i を求めれば 1 次元配列とすることができる．

$$i = a_1 \times 27^0 + a_2 \times 27^1 + a_3 \times 27^2 + a_4 \times 27^3 + a_5 \times 27^4 + a_6 \times 27^5 + a_7 \times 27^6 + a_8 \times 27^7 + a_9 \times 27^8$$

このようにスペルから直接添え字の値が計算できる配列に英単語を格納すると，とても高速に探索，削除，追加ができるデータ構造となる．すべて，計算量のオーダーは $O(1)$ である．しかし，この方法をよく考えると大きな問題がある．巨大なメモリー空間が必要なのである．配列の最大の添え字は，$1\times27^9-1 = 7625597484986$ となりとても現在のコンピューターで取り扱うことができない．

このような配列に単語を格納すると，ほとんどその中は空になる．単語の数はそれほど多くない．諸君の英和辞典の単語数を調べてもよく分かる．そこで，この配列の添え字を 0〜99999 に圧縮することを考える．新たな添え字を j とすると，

```
j=i%100000;
```

とすれば良い．100000 で割った余りとすれば，0〜99999 の値が得られる．このようにして，データにアクセスする方法をハッシュ法と言う．得られた値 (0〜99999) をハッシュ値と言い，キーを変数として，それを求める関数をハッシュ関数と言う．

このようにすると，明らかに以下の問題点がある．

- 異なるキーで同じハッシュ値となる場合がある．

これについて，以降，教科書に沿って説明する．

4.2　ハッシュ値の決め方

ハッシュを使う場合，配列にそのままデータを入れないで，データを表すポインターを入れる方が便利である．ポインターが表すものを構造体とすれば，かなり柔軟に対応できる．後で述べるがハッシュ値の重複 (衝突) 対策でリストやツリー構造を使う場合も対応が容易である．

ハッシュ法を使うためには，教科書に書いてあるとおり，以下のような動作が必要である．

- ハッシュ表を用意する．これは，ポインターを格納する配列である．
- ハッシュ関数 (ハッシュ値) を決める．

ハッシュ表は，データを示すポインターを入れる配列である．キーを決めるとハッシュ関数を計算してハッシュ値が決まる．このハッシュ値がこの配列の添え字の値となる．そして，そこに格納されている配列の値 (ポインター) により，データにアクセスする．

ハッシュ表を作るときにはデータ数よりも大きいことが望ましい．そうしないと，異なったキーでも同じハッシュ値となり，衝突が発生し，データの探索スピードが落ちてしまう．衝突の問題は後で述べるので，ここではハッシュ関数の作り方を説明する．ハッシュ表の大きさは素数とすることが望ましい．こうすることにより，キーのどのビットが変化してもハッシュ値の値を変化させることができる．

ハッシュ関数は，一般に，次の 2 つの手順により成り立っている．

- キーからなにがしかの演算を行い整数値を導き出す.
- 導き出された整数をなにがしかの演算により，ハッシュ表 (配列) の大きさの整数に直す.

最初のキーから整数値を導き出す方法は，いろいろある．これについては，教科書の方法を説明する．一方，これをハッシュ表の大きさの整数に直す方法は，大体決まっている．キーから導き出された整数をハッシュ表の大きさで割って，その余りとするのである．

4.3 ハッシュ値の重複対策

異なるキーであっても，同じハッシュ値となることがある．これが多いとハッシュ法の効果が薄いのであるが，どうしても避けることができない．プログラマーはその対策を講じなくてはならない．教科書に書かれている方法は，次の通りである．

- リストを用いる方法 (教科書 p.222 Fig.7-7)
- 2 分木を用いる方法 (教科書 p.224 Fig.7-9)
- 再ハッシュ(教科書 p.222 Fig.7-8　 p.224 Fig.7-10)

これらについては，教科書を用いて説明する．

5 練習問題

[問 1] 6 桁の整数 $(a_1a_2a_3a_4a_5a_6)$ をハッシュを用いて配列に格納することを考える．ハッシュ関数を $mod(a_1+a_2+a_3+a_4+a_5+a_6, 17)$ とする．$mod(x, y)$ は x を y で割った余りとする．C 言語では，`x%y` である．以下の問いに答えよ．

- ハッシュ値の取りうる値の範囲を述べよ．
- 以下の場合のハッシュ値を述べよ．
  (ア) 246801　 (イ) 986532　 (ウ) 123456　 (エ) 654321

[問 2] ハッシュ関数を $mod(a_1+a_2+a_3+a_4+a_5+a_6, 17)$ とした場合，123456 と 654321 は同じハッシュ値になる．これを防ぐためには，ハッシュ関数をどのようにすれば良いかのべよ．

5.1 レポート提出要領

提出方法は，次の通りとする．

期限　2 月 6 日 (月) AM 10:40
用紙　A4
提出場所　山本研究室の入口のポスト
表紙　表紙を 1 枚つけて，以下の項目を分かりやすく記述すること．
　授業科目名「情報工学」
　課題名「課題　ハッシュ」
　2E　学籍番号　氏名
　提出日
内容　2 ページ以降に問いに対する答えを分かりやすく記述すること．

6　付録

6.1　アスキーコード

教科書にアスキーコード表が載っていないので，表 4 に載せておく．この表から数字 (10 進数) を計算する方法は，次のようにする．

1. 対応する文字の上位 3 ビットの値を探す.
2. 次に下位 4 ビットを探す.
3. 対応する整数の計算を行う．計算方法は 16×(上位 3 ビット)+(下位 4 ビット) である．ただし，A=10, B=11,$\cdots$,F=15 である[1].

具体的な例で表すと，大文字の'd' は，64 なので

$$16 \times 6 + 4 = 100 \tag{1}$$

となる．

表 4: アスキーコード表．表の行 (0〜7) は上位 3 ビットで，列 (0〜F) は下位 4 ビットを表す．表中の 2 文字以上のものは文字ではなく，制御コードと呼ばれる特殊文字である．

	0	1	2	3	4	5	6	7
0	NUL	DLE	SP	0	@	P	‘	p
1	SOH	DC1	!	1	A	Q	a	q
2	STX	DC2	”	2	B	R	b	r
3	ETX	DC3	#	3	C	S	c	s
4	EOT	DC4	$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	’	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	HT	EM	)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[	k	{
C	FF	FS	,	<	L	\	l	|
D	CR	GS	-	=	M	]	m	}
E	SO	RS	.	>	N	^	n	~
F	SI	US	/	?	O	_	o	DEL

[1] ようするに 16 進数

参考文献

[1] 紀平拓男, 春日伸弥. プログラミングの宝箱　アルゴリズムとデータ構造. ソフトバンクパブリッシング(株), 2004 年.