情報科学演習１_第１期課題

6321120
横溝　尚也

提出日：６月１３日（火）

1　課題

Ｊリーグの対戦成績データを元に，後述する実装の項目で指定されたデータを取り出すプログラムを作成せよ対戦データは対戦年 対戦月 対戦日 ホームチーム名 ホームチーム得点 アウェイチーム得点 アウェイチーム名
のデータが 1 列に記載されており，1992 年から 2016 年までの 14196 行 (試合) のデータがある．
実装
・対戦成績ファイルからデータを読み込み，リスト構造を用いてデータを作成する add() 関数を作成しなさい．ただし，match 構造体のデータは，ハッシュテーブルを用いて探索を効率化すること． その際，ハッシュ値は，ホームチーム名とアウェイチーム名の 2 つの文字列をキーとして算出すること．作成したハッシュテーブルの分布を調べる関数 dist() を作成しなさい．

・dist() の結果を元に，偏りの少ないハッシュ関数を自作しなさい．2 つの引数 (ホームチーム名，アウェイチーム名) を取り，これまでの対戦成績と試合数を表示する関数 record() を作成しなさい．この際，ホームとアウェイが入れ替わった対戦成績は加味しない．(別の対戦カード扱いとする) 上記に加え，指定された年を第 3 引数に取り，該当年の対戦成績と試合数を表示する関数 record_year() を作成しなさい．

・(任意課題) この対戦成績データを元に，他に面白いデータを検索する関数などを実装しても良い．その場合，検索しやすいように構造体やハッシュのキーを変更しても良い．(例:9 月の勝率, 相性の良い対戦相手，総得点の活用，ホームの勝率ランキング・・・など)

2　アルゴリズムの説明

add 関数についてのアルゴリズムを説明する。まず初めに受け取った引数をリストに追加するためにメモリの確保と構造体のメンバ変数へ値を代入する。構造体 match_score へデータを代入することは容易である。次に、対戦カードごとにリストを作成する。ここで対戦カードが一致する既存のリストが存在すればそこに新たなリスト要素として追加、存在しなければ新たなリストの先頭要素としてリストを作成する。ここで match 構造体のデータ探索を効率化するためのハッシュ関数を用いている。この関数では二つの文字列ポインタであるハッシュキーを引数として受け取り、初期化したハッシュ値にハッシュキーの値を足し続ける。この際、ハッシュキーをインクリメントしていき、ハッシュキーがヌル文字になるまでハッシュキーを計算する。計算した合計値をあらかじめ定められたハッシュサイズ（今回は１７とする）で割った余りを出力する。この関数で得られる値は０から１６の値である整数値であるため、配列要素数１７の配列の要素番号に適用する。

次に対戦カードを引数として受け取り、その対戦カードの戦績を返す関数 record のアルゴリズムを説明する。戦績として必要な情報は、試合数、勝利数、敗北数、引き分け数であるのでそれぞれの文字を宣言してカウントしていく。初めに受け取った対戦カードと一致するリストを構造体 match から探す必要がある。二つの対戦チームのアドレスと一致することを条件にリストの検索を行えばよい。一致する対戦カードが見つければそのリストに対して、対戦詳細が記録されているリストの要素を調べれば試合数がわかる。リストの先頭から次のリスト要素へと順にアクセスし、リストの末端までインクリメント変数を増やし続ける。リストの末端を判別するには次のリストポインタ変数の値が NULL かどうかを調べればよい。勝利数、敗北数、引き分け数も同じように考えられる。
record_year 関数は受け取った対戦カードと一致するリストを探索するところまでは record 関数と同じであ

る。見つかった対戦カードのうちすべての対戦詳細リストすべてを試合数のカウント対象とするのではなく。新たに引数として受け取った年度と対戦詳細に書かれている年度が一致するもののみカウント対象とすればよい。

3　プログラムの説明

　以下ではソースコードを分割して一つ一つ説明していく。
ただし課題内容に書かれていた構造体やメイン関数の説明は今回省略する。

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define HASHSIZE 17


struct match_score{ /*各試合の詳細*/
  int year;
  int month;
  int day;
  int home_score;
  int away_score;
  struct match_score *next;
};

struct match{  /*試合一つ一つの構造体*/
  char *home;
  char *away;
  struct match_score *r;
  struct match *next;
};

struct match *hashtable[HASHSIZE];


/*ハッシュ関数*/
int hash(char *hashkey1, char *hashkey2){
  int hashval = 0;
  while(*hashkey1 != '\0'){
    hashval = hashval + *hashkey1;
```

```
    hashkey1++;
  }
  while(*hashkey2 != '\0'){
    hashval = hashval + *hashkey2;
    hashkey2++;
  }

  return hashval % HASHSIZE;
}
```

ここまではハッシュ関数の記述をしている。今回採用したハッシュ関数はハッシュキーがヌル文字になるまでハッシュ値に足し続けるアルゴリズムであるので while 文を使用して hashval を求めている。最終的には0から１６までの値を返したいので HASHSIZE の剰余を関数の返戻地としている。

```
void add(int year,int month,int day,char* home_team,int home_score,
  int away_score,char* away_team){
  struct match_score *match_data;
  match_data = malloc(sizeof(struct match_score));

  //メモリの確保と、メンバ変数への代入
  match_data -> year = year;
  match_data -> month = month;
  match_data -> day = day;
  match_data -> home_score = home_score;
  match_data -> away_score = away_score;
  match_data -> next = NULL;

  //対戦カードの探索
  int matchpair = 0;
  int hashval = hash(home_team, away_team);
  struct match *ptr = hashtable[hashval];

  while(ptr != NULL){
    if(strcmp(ptr->home, home_team) == 0 && strcmp(ptr->away,away_team) ==0){
      matchpair = 1; //対戦カードの組が見つかったため真
      match_data -> next = ptr -> r;
      ptr -> r = match_data;
      break;
    }
```

```
    ptr = ptr -> next;
  }

  /*対戦カードが見つからなかった場合、新たに対戦カードを追加*/
  if(matchpair == 0){
    struct match *new_matchcard;
    new_matchcard = malloc(sizeof(struct match));
    new_matchcard -> home = strdup(home_team);
    new_matchcard -> away = strdup(away_team);
    new_matchcard -> r = NULL;
    new_matchcard -> next = NULL;

    new_matchcard -> r = match_data;
    new_matchcard -> next = hashtable[hashval];
    hashtable[hashval] = new_matchcard;
  }


}
```

ここでは add 関数の記述である。前半では変数などの宣言と、メンバ変数へ値を代入してデータを格納している。後半でリスト構造を作成している。対戦カードが既存であるかどうかを判別したいので変数 matchpair を宣言し、1 になればリストが存在、0 ならば存在しないとしている。この時 strcmp を使用し、対戦カードに一致するものがあるか while 文で繰り返し探索している。while 文によってすべて探索し終えた段階で matchpair が 0 のまま、つまり一致する対戦カードが存在しなかった場合、新たなリストを作成するために、次の要素へのポインタが NULL である要素を作成している。

```
void dist(){
  printf("--HASHSIZE: %d--\n" , HASHSIZE);
  for(int i = 0;i < HASHSIZE; i++){
    int count = 0;
    struct match *ptr = hashtable[i];
    while(ptr != NULL){
      count++;
      ptr = ptr -> next;
    }

    printf("TABLE[%d]:%d\n" , i, count);
  }
```

```
}


void record(char* home, char* away){
  int hashval = hash(home, away);
  int count = 0;
  int win = 0;
  int lose = 0;
  int draw = 0;

  struct match *ptr;
  struct match_score *ptr2;
  ptr = hashtable[hashval];

  while(ptr != NULL){ /*リスト構造の末端まで探索*/
    if(strcmp(ptr->home,home) == 0&&strcmp(ptr->away,away) ==0){
      ptr2 = ptr -> r;
      while(ptr2 != NULL){
/*探索した対戦カードのリストを末端までアクセスし数をカウント*/
count++;
if(ptr2 ->home_score > ptr2 -> away_score){
  win++;
}else if(ptr2 -> home_score < ptr2 -> away_score){
  lose++;
}else{
  draw++;
}
ptr2 = ptr2 -> next;
      }
      break;
    }
    ptr = ptr -> next;
  }
  printf("%s (HOME) vs %s (AWAY) : %d games.(win:%d, lose:%d, draw:%d)\n"
 , home, away, count, win, lose, draw);
}


void record_year(char* home, char* away, int year){
  int hashval= hash(home,away);
```

```
  int count = 0;
  int win = 0;
  int lose = 0;
  int draw = 0;

  struct match *ptr;
  struct match_score *ptr2;
  ptr = hashtable[hashval];

  while(ptr != NULL){
    if(strcmp(ptr -> home,home) ==0 && strcmp(ptr -> away,away) ==0){
      ptr2 = ptr -> r;
      while(ptr2 != NULL){
if(ptr2->year ==year){ /*年度が一致しているか条件式を追加*/
  count++;
  if(ptr2->home_score > ptr2 -> away_score){
    win++;
  }else if(ptr2 -> home_score < ptr2 -> away_score){
    lose++;
  }else{
    draw++;
  }
}
ptr2 = ptr2 -> next;
      }
      break;
    }
    ptr = ptr -> next;
  }
  printf("[year:%d]: %s (HOME) vs %s (AWAY) : %d games.(win:%d, lose:%d, draw: %d)\n"
  , year, home, away, count, win, lose, draw);
}



int main(){
  char home_team[256], away_team[256];
  int home_score, away_score;
  int year, month, day, i, f;
  char *filename = "jleague.txt";
```

```
  FILE *fp;

  for(i=0; i < HASHSIZE; i++){
    hashtable[i]=NULL;
  }


  fp = fopen(filename,"r");
  while((f = fscanf(fp,"%d %d %d %s %d %d %s",
  &year,&month,&day,home_team,&home_score,&away_score,away_team))!=EOF){
    add(year,month,day,home_team,home_score,away_score,away_team);
  }

  dist();
  record("Urawa_Red_Diamonds","Kashiwa_Reysol");
  record("Kashiwa_Reysol","Urawa_Red_Diamonds");
  record("Kashima_Antlers","Gamba_Osaka");
  record_year("Kashima_Antlers","Gamba_Osaka",2007);
}
```

ここまでで、関数 dist,record,reccord_year 関数の記述をしている。dist 関数では課題内容に書かれている実行結果に合うよう出力するために、hash 関数で計算したハッシュ値を出力している。

record 関数では add 関数と同様に strcmp を使用して一致する対戦カードを探索している。見つかった場合に、そのリストの要素数、が試合数に一致するので count を宣言し、リストをたどるごとにインクリメントしていく。勝利数、敗北数、引き分け数も同様に if 文で試合内容を点数の大小比較によって勝ち負け、引き分けを判別し数えている。

record_year 関数は record 関数とほとんど同じ記述である。異なる点は、一致する対戦カードを見つけた後、試合数などをカウントする対象をさらに if 文で絞っている。今回絞る条件が引数として受け取った年度に一致するものであるので $ptr2->year == year$ としている。

4　実行結果

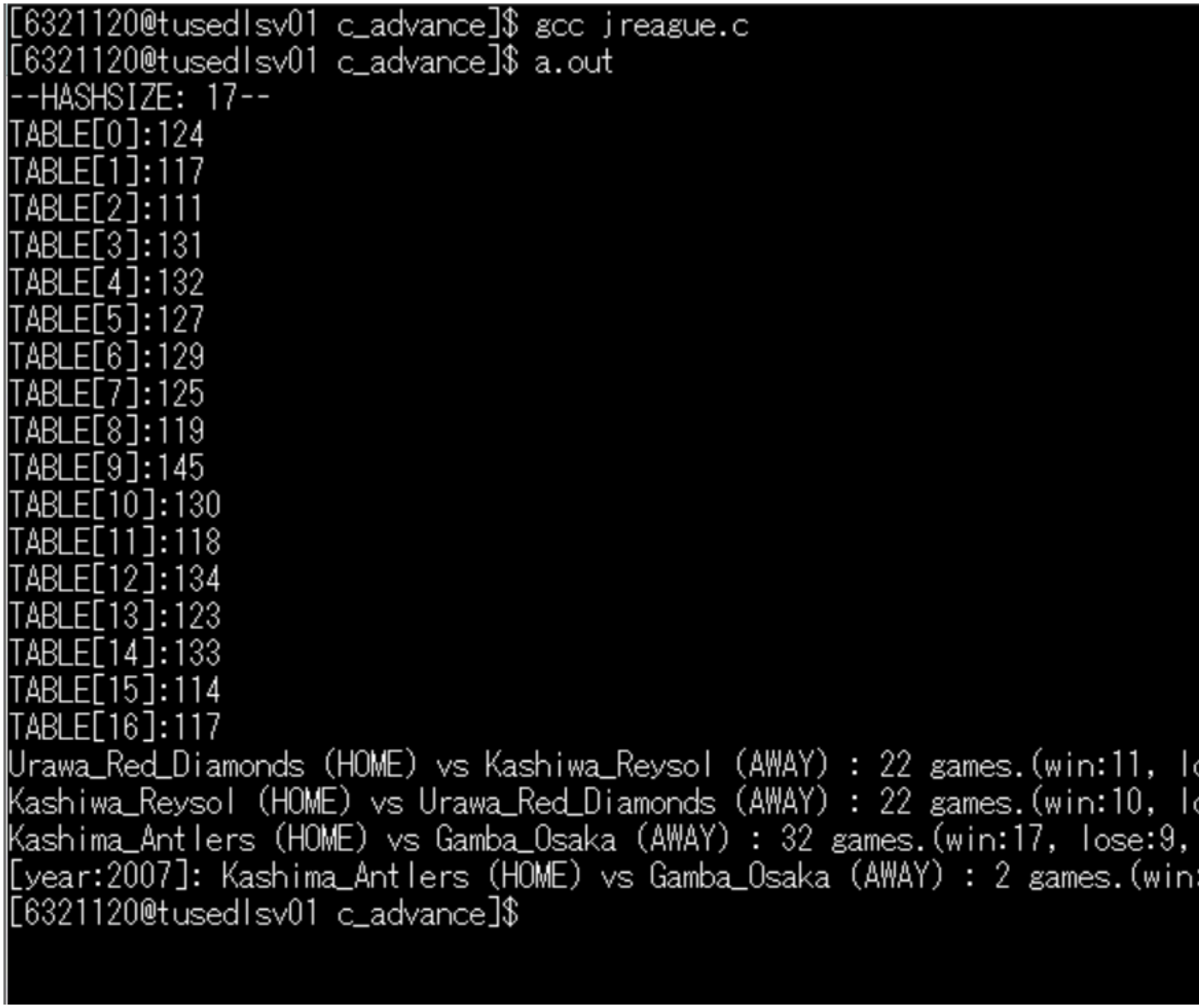

図 1　出力結果

5　考察

5.1　ハッシュ関数の構成について

今回のプログラムでは HASHSIZE を１７とした。今回の関数では受け取ったアドレスに値を足していった合計値を HASHSIZE で割った余りをハッシュ値とした。この時 HASHSIZE が素数でなかった場合、ハッシュ値が同じ値になってしまうシノニムの発生確率が高くなるのであ h 内科と考えた。

5.2　データ構造について

今回のプログラムではファイルから入力されたデータを定義した構造体を一つの要素としたリスト構造に格納していった。今回の j リーグ戦績データはデータの長さが不明であったのでリストを使用する利点があったと思う。データの長さが決められていれば、要素の追加や削除などがリストよりも容易にできる配列構造を使用してもデータ構造として成立するのではないかと考えた。