1 do while 文

1.1 学習のポイント

ループの終末で繰り返し条件を判断する do while 文を学びます。

while 文は、ループの先頭でループを繰り返すか否かを判断する繰り返し制御文でした。これに対し、do while 文はループの終末でループを繰り返すか否かを判断する繰り返し制御文です。

do while 文は、(文) を実行した後で (式) を評価し、(式) が真である間 (文) を繰り返します。

do while 文は、while 文に比べて使用頻度は少ないのですが、ある条件で繰り返す場合に、終了前にもう一度ループ内の文を実行してからループを抜けたいときに使うと便利です。

1.2 reidai29.c

```
/*
  reidai29.c
  a[] で示される文字列を大文字に変換して b[] に格納する関数 upr() を作りなさい。
  C 言語 125 頁
*/

#include <stdio.h>

void upr(char a[], char b[])
{
  int i = 0;

  do{
     if('a' <= a[i] && a[i] <= 'z'){
       b[i] = a[i] - ('a' - 'A');
     }else{
       b[i] = a[i];
     }
  }while(a[i++] != '\0');
}
```

```
int main()
{
  char dat[80];
  char result[80];

  printf("moji ? ");
  scanf("%s", dat);

  upr(dat, result);
  printf("%s\n", result);

  return 0;
}
```

2　AI による大規模データ処理入門

2.1　発見的な探索

最適経路探索では、探索の過程で得られた節点の開始節点からの評価値の積算値 v を用いて、最適な経路を探索します。節点を展開する過程で求まる入口からの距離を積算していき、積算距離の最も小さくなる経路を探索します。こうすれば、節点に至る経路が 2 つあっても、最終的には短い方が探索結果となります。先に述べたように、最良優先探索で用いる評価値 h は、ヒューリステック関数による推定値ですから、必ずしも正しい値ではありません。これに対して最適経路探索で用いる評価値 v は、探索開始後に得られた節点における探索開始時からの評価値の積算値です。したがってこの値は推定値ではなく、探索によって確定した値です。
AI による大規模データ処理入門　82 頁　小高　知宏著　オーム社

2.2　最良経路探索プログラム

```
/*
  最適経路探索プログラム
  (分岐限定法,branch and bound)
  bb.c
*/

#include <stdio.h>
```

```
#define LIMITL 256
#define TRUE 1
#define START 1
#define GOAL 999

struct node{
  int    nodeid;
  int    parentid;
  double value;
};

struct node openlist[LIMITL];
int         openlistep = 0;
struct node closedlist[LIMITL];
int         closedlistep = 0;

void initlist();
void printlist();
void expand(struct node cnode);
void movetofirst();
void removefirst();
int  ccheck(int id);
int  ocheck(struct node cnode, int id, int v);
void printroute(int id);
void sortopenlist();
int  cmp(const void *a, const void *b);

int main()
{
  struct node currentnode;

  initlist();
  printlist();

  while(TRUE){
    if(openlistep == 0){
      printf("ゴールは見つかりませんでした\n");
      break;
    }
```

```
    if(openlist[0].nodeid == GOAL){
      printf("\n ゴールを見つけました\n");
      printf("%d[%d]", openlist[0].nodeid,
                       openlist[0].parentid);
      printroute(openlist[0].parentid);
      break;
    }

    currentnode = openlist[0];

    removefirst();

    expand(currentnode);

    closedlist[closedlistep++] = currentnode;

    sortopenlist();

    printlist();
  }

  return 0;
}

void sortopenlist()
{
  qsort(openlist, openlistep, sizeof(struct node), cmp);
}

int cmp(const void *a, const void *b)
{
  struct node *x, *y;

  x = (struct node *)a;
  y = (struct node *)b;
  if((x->value) < (y->value)){
    return -1;
  }else{
    if((x->value) > (y->value)){
```

```
      return 1;
    }else{
     return 0;
    }
  }
}

void printroute(int id)
{
  int i;

  for(i = 0; i < closedlistep; ++i){
    if(closedlist[i].nodeid == id){
      printf("<-%d[%d]", closedlist[i].nodeid,
                         closedlist[i].parentid);
      break;
    }
  }

  if(closedlist[i].parentid != 0){
    printroute(closedlist[i].parentid);
  }

  printf("\n");
}

int ccheck(int id)
{
  int i;
  int res = 0;

  for(i = 0; i < closedlistep; ++i)
    if(closedlist[i].nodeid == id)
      res = TRUE;

  return res;
}

int ocheck(struct node cnode, int id, int v)
{
```

```
  int i,j;

  for(i = 0; i < openlistep; ++i)
    if(openlist[i].nodeid == id){
      if(openlist[i].value < cnode.value + v){
        return TRUE;
      }else{
        for(j = i; j < openlistep; ++j){
          openlist[j] = openlist[j + 1];
        }
        --openlistep;
        }
      }
  return 0;
}


void removefirst()
{
  int i;
  for(i = 0; i < openlistep; ++i){
    openlist[i] = openlist[i + 1];
  }
  --openlistep;
}

void movetofirst()
{
  closedlist[closedlistep++] = openlist[0];
  removefirst();
}

void expand(struct node cnode)
{
  double tree[][10] = {
    {1, 2, 2, 4, 3},
    {2, 3, 2, 6, 2},
    {3, 2, 2, 5, 2, 999, 2.8},
    {4, 3, 2.4},
    {5, 7, 2, 999, 2},
```

```
    {0}
  };

  int i = 0;
  int j;

  while(tree[i][0] != 0){
    if(tree[i][0] == cnode.nodeid){
      for(j = 1; tree[i][j] != 0; j+=2){

        if(ocheck(cnode, tree[i][j], tree[i][j+1]) != TRUE
          && (ccheck(tree[i][j]) != TRUE)){
            openlist[openlistep].nodeid = tree[i][j];
            openlist[openlistep].parentid = cnode.nodeid;
            openlist[openlistep++].value = cnode.value + tree[i][j + 1];
        }
      }
      break;
    }
    ++i;
  }
}

void initlist()
{
  openlist[0].nodeid = START;
  openlist[0].parentid = 0;
  ++openlistep;
}

void printlist()
{
  int i;

  printf("\nオープンリスト    ");
  for(i = 0; i < openlistep; ++i){
    printf("%d[%d, %.1lf],", openlist[i].nodeid,
                             openlist[i].parentid,
                             openlist[i].value);
  }
```

```
  printf("\n");

  printf("クロースドリスト  ");
  for(i = 0; i < closedlistep; ++i){
    printf("%d[%d],", closedlist[i].nodeid,
                      closedlist[i].parentid);
  }
  printf("\n");
}
```

2.3 使い方

```
./bb
```