アルゴリズム論

ヒープで最小木発見

氏名:津波古正輝
学籍番号:075739A
所属:情報工学科

提出日:平成 21 年 8 月 3 日 (月曜日)

1.
最小木(ヒープ)

ソース：

```
#include <stdio.h>
#include <stdlib.h>

#define VMAX 100 //頂点の最大個数
#define EMAX 1000 //辺の最大個数

struct adjlist{
  int e_num;
  int e_weight;
  struct adjlist *next;
} *adjlist[VMAX];

int n; //頂点の個数
int m; //辺の個数
int visit[EMAX]; // 訪れた辺
int bfnum[VMAX]; //訪れた点
int edge[EMAX];//どこからきたのかを格納する配列
int out[EMAX], in[EMAX]; // 辺の両端
int heap_p[VMAX]; //点のヒープ
int heap_e[VMAX]; //辺のヒープ
int c, d; //訪問番号
int last; //ヒープの最後の要素を示す

void input_graph();
int input_start();
void addlist(int j, int v, int w, int e_num);
int input_heap(int v, int ew);
void mst(int v);
void initheap();
void swap(int x, int y);
```

```
void siftup(int s);
void siftdown(int s, int t);
int findmin();
void print_result();

void input_graph(){
  int i;
  int j, v, w; //j:辺の重み v;ノード w;ノード
  printf("Node amount: ");
  scanf("%d",&n); //n:頂点の個数
  printf("edge amount: ");
  scanf("%d",&m); //m:辺の個数
  printf("please input Egde's weight and both ends of a Egde\n");
  printf("example:weight EgdeNumber1 EgdeNumber2\n");
  for(i=1; i<=m; i++){
    printf("edge%d : ",i); //i=e_num
    scanf("%d %d %d", &j,&v,&w); //j:辺の重み v;ノード w;ノード
    out[i] = v; in[i] = w; //Edgeに対するノードの終点と始点を格納
    addlist(j, v, w, i); //i=e_num
  }
}

int input_start(){
  int w;
  printf("Please input start point. \n");
  printf("start : ");
  scanf("%d",&w);
  return(w);
}

// 入力した値から木を作成
// j = 辺の重み. v,w = 辺の両端,i = 辺の名前
void addlist(int j, int v, int w, int e_num){
  struct adjlist *new, *work;
  new = (struct adjlist *)malloc(sizeof(struct adjlist)); //リスト用の領域を確保
```

```
  new->e_num = e_num;
  new->e_weight = j;
  work = adjlist[v];
  adjlist[v] = new;
  new->next = work;
  new = (struct adjlist *)malloc(sizeof(struct adjlist));//リスト用の領域を確保
  new->e_num = e_num;
  new->e_weight = j;
  work = adjlist[w];
  adjlist[w] = new;
  new->next = work;
}

void mst(int v){
  int w,e;
  struct adjlist *p;
  heap_p[1] = v; //スタート地点のノード番号を格納
  last = 1;//last:ヒープ木の最後の要素をさす

  while(heap_p[1] > 0){ //すべて探索が完了したらfindminでheap_p[1]が-1になる
    v = findmin();      //木の中でノードからでている辺の重みが最小のものを見つけ、
                        //その辺に接しているノードを取り出す
    if(bfnum[v] == 0){ //v(ノード)を訪れていない
      bfnum[v] = c;//c;ノード用の訪問番号
        c = c+1;
        p = adjlist[v]; //ノードの情報取得

        while(p != NULL){
          e = p->e_num;
          if(visit[e] == 0){ //e(辺)を訪れていない
            visit[e] = d;//d;辺用の訪問番号
            d = d+1;

            if(out[e] == v){//vから繋がっているノードをwに格納
              w = in[e];
```

```
            }else{
              w = out[e];
            }
            if(bfnum[w] == 0){ //v(ノード)から繋がっているw(ノード)を訪れていない
              if(input_heap(w, p->e_weight)){//w;訪れているノード
                                              //e_weight;wに行く為に使用した辺の重さ
                                  //r=1と返ってきたらheap_eにinsertされたとういこと
            edge[w] = v;//どこのノードからきたのかを格納
                        //(ノードwはノードvから来ましたっていうことを示す)
            }
          }
        }
      p = p->next; //次のノードに行き、同様の処理を行う
      }
    }
  }
}

void initheap(){
  int i;
  for(i=1; i<VMAX; i++){
    heap_p[i] = -1; //ノードのヒープ
    heap_e[i] = 1000; //辺のヒープ
  }
}



//ヒープのルートの値を返す
//ヒープのルートの値を消して、最後の要素をルートに持ってくる
//その後、再構築
int findmin(){
  int f;
  if(heap_p[1] != -1){
    f = heap_p[1]; //最初の要素を取り出す
  }
```

```
  if(last > 1){ //木が要素２以上で構成されている
    heap_p[1] = heap_p[last]; //ヒープの最後の要素を最初にもってきて再構築
    heap_e[1] = heap_e[last];

    heap_p[last] = -1;
    heap_e[last] = 1000;
    last--;

    siftdown(1,last);//最後の要素を最初に持ってきているのでシフトダウンの必要性あり
  }else{
    heap_p[1] = -1;
    heap_e[1] = 1000;
    last--;
  }
return(f);
}

int input_heap(int v, int ew){
  // v : 点の名前 , ew : 辺の重み
  int i,r;
  for(i=1; i<=n; i++){
    if(heap_p[i] == v){ //すでに一度そのノードの違う辺をヒープの要素として構築して
いる時
      if(heap_e[i] > ew){ //ヒープの値が辺の重みより大きい場合 (初期値 1000)
        heap_e[i] = ew; //値を代入する
        r=1;
        break;
      }else{
        r=0;
        break;
      } //ヒープの値が辺の重みより小さい場合
    } else if(heap_p[i] == -1){ //まだヒープに何も入っていない場合
        heap_p[i] = v;
        heap_e[i] = ew; //それぞれの辺と点の値を代入する。
        last++;//ヒープの最後を示す
```

```
        r=1;
        break;
      }
    }
  //最後にいれるので、シフトアップする
  siftup(last); //last は挿入した値のノードの位置を示している。
  return(r);
}

void siftup(int s){ //last=s が二分木のノード番号を表し、i が一つ上の木のノードを示
している
  int i;
  i = s/2;
  if(s>=1 && heap_e[s]<heap_e[i]){
    swap(s,i);//値が大きかったら交代
    siftup(i);//交代したノードに対して同様の処理を行う
  }
}

void siftdown(int s, int t){ //t=last
  int i;
  //左と右を比較して大きい値を交換していく
  for(i=1; i<=t; i++){
    if(heap_e[i*2+1] > heap_e[i*2]){//右が大きい
      if(heap_e[i] > heap_e[i*2]){ //右子ノードと親ノードを比べる
        swap(i,i*2);
      }
    }else{
      if(heap_e[i] > heap_e[i*2+1]){ //左子ノードと親ノードを比べる
        swap(i,i*2+1);
      }
    }
  }
}
```

```
void swap(int x, int y){ //x=子ノード,y=親ノード
  int temp1,temp2;
  temp1 = heap_e[x];
  heap_e[x] = heap_e[y];
  heap_e[y] = temp1;
  temp2 = heap_p[x];
  heap_p[x] = heap_p[y];
  heap_p[y] = temp2;
}

void print_result(){
  int i,j;
  for(i=1; i<=n; i++){
    for(j=1; j<=m; j++){
      if(bfnum[j] == i){//bfnum[VMAX];訪れた点
        if(edge[j]!=0){// int edge[EMAX];どこからきたのかを格納する配列
          printf("next:visit Nodenumber is  %d form %d\n",j,edge[j]);
        }
      }
    }
  }
}

int main(){
  int v;
  int e;
  c = 1; /ノード用の訪問番号
  d = 1; //辺用の訪問番号
  initheap(); //点と辺のヒープの初期化
  input_graph(); //木の作成
  for(v=1; v<=n; v++) bfnum[v] = 0;//点を訪れたかどうかの配列を初期化
  for(e=1; e<=m; e++) visit[e] = 0; //辺を訪れたかどうかの配列を初期化
  int s = input_start(); //出発ノードの決定
  printf("\n");
  printf("first visit Nodenumber is %d\n",s);
```

```
  mst(s);
  print_result();
  return (0);
}

//ノードと辺のヒープ木を二つ用意。
//辺のヒープ木をヒープによって構築し、辺の木に合わせてノードの木を構築する。
```

コンパイルと実行結果：

```
% gcc -o mst mst.c
% ./mst
Node
Node amount: 8
Edge
edge amount: 10
please input Egde's weight and both ends of a Egde
example:weight EgdeNumber1 EgdeNumber2
edge1 : 1 1 2
edge2 : 3 1 3
edge3 : 5 1 4
edge4 : 3 1 5
edge5 : 3 2 8
edge6 : 2 2 6
edge7 : 5 2 3
edge8 : 1 4 6
edge9 : 1 6 8
edge10 : 4 6 7
Please input start point.
start : 1

first visit Nodenumber is 1
next:visit Nodenumber is  2 form 1
next:visit Nodenumber is  6 form 2
next:visit Nodenumber is  8 form 6
next:visit Nodenumber is  4 form 6
next:visit Nodenumber is  5 form 1
next:visit Nodenumber is  3 form 1
next:visit Nodenumber is  7 form 6
```

以下の問題木を使用した。

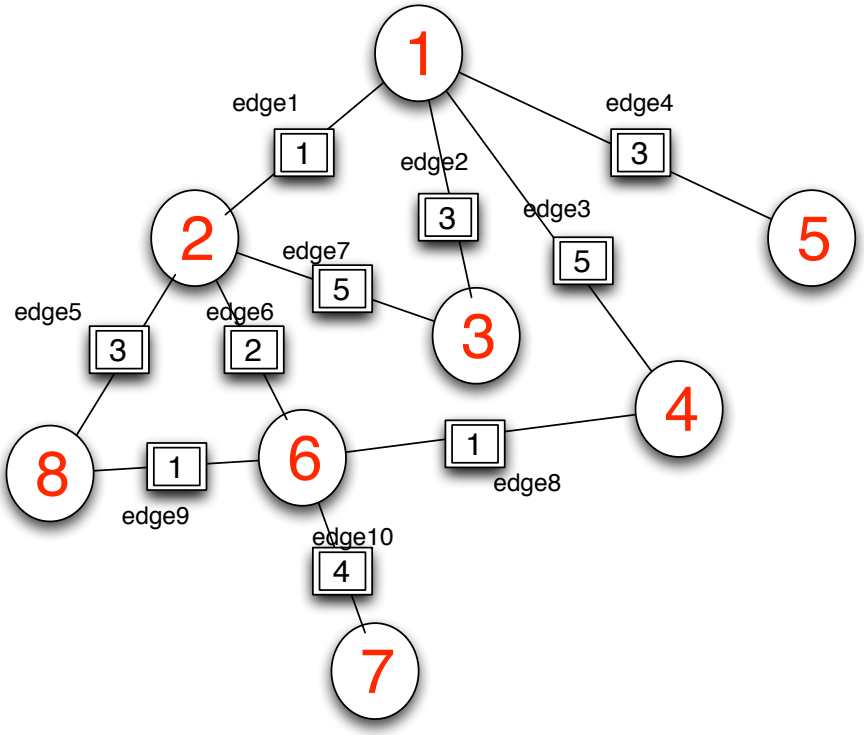

図 1: 問題木

四角で囲まれている数字が辺の重さである。それぞれの辺には名前がついており、edgeX となっている。

- 問題木について：
  構造体で表した。

- 関数について：
  - void input_graph()
    問題木のデータを入力するノードの個数と辺の個数を入力する。その後、辺の重み, その辺の両端のノード 1, その辺の両端のノード 1 を入力する。
  - int input_start()
    最小木のスタート地点を入力する
  - void addlist(int j,int v,int w,int e_num)
    入力したデータを元に構造体で木を表現する。
  - int input_heap(int v,int ew)
    辺と辺の重さを二分木で表し、ヒープ構造にする
  - void mst(int v)
    辺とノードをたどり、最小木を作る
  - void initheap()
    ヒープを初期化する
  - void swap(int x,int y)
    値を交換する
  - void siftup(int s) と void siftdown(int s,int t)
    木をヒープで再構築する時に必要な
  - int findmin()
    木から最小値を取り出し、木の再構築を行う
  - void print_result()
    結果を出力する

探索スタートの頂点を 1 に設定すると以下の様な結果を得られる。

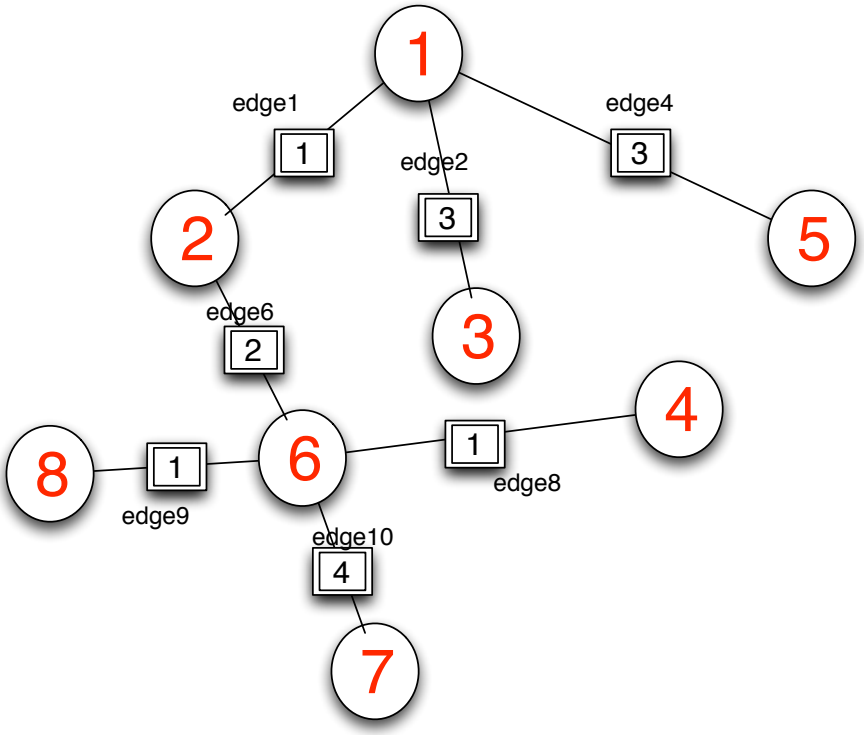

図 2: 問題木

参考文献:
075756 のレポートおよび 065758 のレポート