2Η ΕΡΓΑΣΙΑ ΣΤΗΣ ΔΟΜΕΣ ΔΕΔΟΜΕΝΩΝ

NIKOS GEORGIADIS 2043

THODWRIS FASOULAS 2096

Σε αυτην την εργασια υλοποιουμε ενα μη κατευθηνομενο γραφημα που περιεχει τις συνδεσεις μεταξυ κομβων.Υπαρχουν 3 κλασεις στον κωδικα.Η κλαση Hashtable ,η κλαση ID και η κλαση Node.

1.) Η Hashtable δημιουργει ενα πινακα κατακερματισμου ht[100.000] αφου 50.000 ειναι ολοι οι κομβοι του αρχειου input και ο πινακας θα πρεπει να εχει το διπλασιο μεγεθος ετσι ωστε ο παραγοντας φορτωσης να ειναι στο 0.5 .Επισης δημιουργουμε εναν πινακα empty οπου περιεχει τιμες bool.Αναλογα αν υπαρχει κορυφη για παραδειγμα σε μια θεση του ht τοτε η αντιστοιχη θεση του empty θα ειναι false αλλιως θα ειναι true. Ο πινακας ht περιεχει αντικειμενα τυπου ID .Τα πεδια αυτου του αντικειμενου ειναι το element δηλαδη το id της κορυφης και το list_head δηλαδη ο δεικτης ο οποιος δειχνει στο πρωτο γειτονα του id.Ετσι καθε θεση του πινακα ht περιεχει το id της κορυφης και ολους τους γειτονες της σε μια απλα συνδεδεμενη ταξινομημενη λιστα. Η κλαση hashtable Περιεχει διαφορες συναρτησεις οπως την insert_all_data() μεσα στην οποια διαβαζουμε ολες της κορυφες και τις ακμες απο το input.txt και της κορυφες της αποθηκευουμε στο hashtable ενω τις ακμες στην συνδεδεμενη λιστα της εκαστοτε κορυφης.Η insert() εισαγει μια ακμη και μια κορυφη.Η delete_() διαγραφει μια ακμη απο μια κορυφη.Η prim() βρισκει το ελαχιστο δεντρο του γραφηματος.Η merging() μας επιστρεφει το πληθος των κοινων γειτονων 2 κορυφων.Η search() αναζητει μια κορυφη στο hashtable και επιστρεφει την θεση αυτης της κορυφης εφοσον υπαρχει.

2.) **Η ID εχει συναρτησεις οπως την add_node_to_list() η οποια προσθετει μια ακμη δηλαδη εναν γειτονα στην συνδεδεμενη λιστα.Διπλες εμφανισεις δεν υποστηριζει.Μια αλλη ειναι η delete_node_to_list() η οποια διαγραΦει μια ακμη δηλαδη εναν γειτονα απο την συνδεδεμενη λιστα.Η sort_list() αυτο που κανει ειναι να ταξινομει me bubblesort την λιστα με βαση τα ID .**

3.) **Η Node υλοποιει την εννοια του κομβου στην συνδεδεμενη λιστα.Δηλαδα ο καθε γειτονας ενος ID του hashtable ειναι ενας κομβος που αποτελειται απο το data δηλαδη την τιμη του κομβου , το weight δηλαδη το βαρος της ακμης και το next δηλαδη τον δεικτη για την επομενη ακμη.**

ΤΟ ΠΡΟΓΡΑΜΜΑ : **Το προγραμα ξεκιναει ανοιγωντας το αρχειο commands.txt .Δημιουργουμε εναν πινακα ht τυπου hashtable και καλουμε την συναρτηση instert_all_data (); Οπου εκχωρουμε ολες τις κορυφες απο το αρχειο αφου πρωτα ομως καλεσουμε την search() για να ελενξουμε αν ηδη υπαρχει η καθε κορυφη στο hashtable.Αν δεν υπαρχει τοτε την αποθηκευουμε στον πινακα και στην αντιστοιχη θεση του empty βαζουμε false και την ακμη της δηλαδη τον γειτονα της τον αποθηκευουμε στην λιστα καλωντας την add_node_to_list(). Αν υπαρχει η κορυφη στον πινακα τοτε κατευθειαν πηγαινω στην θεση οπου υπαρχει και εκχωρω στην λιστα την ακμη καλωντας παλι την add_node_to_list().Την παραπανω διαδικασια την κανω και για το αναποδο ζευγαρι κορυφης-κομβου γιατι το γραφημα ειναι μη κατευθηνομενο.Για παραδειγμα στο ζευγαρι 1 3 10 θα πρεπει να υπολογισω και το ζευγαρι 3 1 10 με την ιδια ακριβως μεθοδο.Στην συνεχεια κανουμε ταξινομηση της καθε λιστας με την ταξινομηση φυσαλιδας.**

```
while (!myfile1.eof( ))
{
    int key; //to prwto id
    int key2; // to diplano id pou tha kanei akmi me to prwto
    int key3;  // to varos tis akmis
    myfile1>>key;   //diavazei to id apo to arxeio
    int b;
    b=search(key);
    if (empty[b])     //otan to stoixeio key den uparxei ston pinaka me ta ID tote to ekxwrw
    {
        empty[b]=false;
        ht[b].element=key;
        myfile1>>key2;  //diavazoume twra to stoixeio me to opoio tha dimiourgisei akmi to key
        myfile1>>key3;
        ht[b].add_node_to_list(key2,key3); // kai to ekxwroume stin lista tou sugkekrimenou id
    }
     else if (ht[b].element==key)
    {
    myfile1>>key2;
    myfile1>>key3;
    ht[b].add_node_to_list(key2,key3);
    }
```

```
int  Hashtable::search(int k)
{

    int tmp;
    tmp=((2*k)+3)%(size/2);    //home bucket
    int j=tmp;  //start at home bucket
    do{
    if (empty[j] || ht[j].element==k ) return j;
    j=(j+1)%size; //next bucket
    }while (j!=tmp); //returned to home ?
    return j; //table full
}
```

```
void ID::add_node_to_list(int d,int w)
{
    Node *temp,*neos;
    temp=list_head;
    bool flag;
    flag=false;
    while (list_head!=NULL)    //elegxw an uparxei o geitonas mesa stin lista
    {

        if (list_head->data==d)
        {

            flag=true;
            break;
        }
        list_head=list_head->next;
        }
    list_head=temp;
    if (flag==false)     //an den uparxei tote ton ekxwrw
    {
    neos=new Node;
    neos->data=d;
    neos->weight=w;
    neos->next=list_head;
    list_head=neos;
    }
}
```

```
for (int i=0; i<size; i++)  //taksinomisi kathe listas tou hashtable
{
     Node *temp;
     temp=ht[i].list_head;
     if (empty[i]==false)
     {
     ht[i].sort_list(); //taksinomisi bubblesort

     while (ht[i].list_head!=NULL)
     {


     ht[i].list_head=ht[i].list_head->next;
     }

     ht[i].list_head=temp;
     }
}

  myfile1.close();
```

```
void ID::sort_list()  //bubble sort se lista
{
     Node *tmpPtr = list_head;  //boithitikoi deiktes
     Node *tmpNxt = list_head->next;
     Node *tmpPtr1 = list_head;
     Node *tmpNxt1 = list_head->next;
    int tmp,tmp1;
    while(tmpNxt != NULL){
           while(tmpNxt != tmpPtr){  //auksousa taksinomisi
                    if(tmpNxt->data < tmpPtr->data){ //antimetathesi twn geitonwn se perip
                            tmp = tmpPtr->data;   //vriskontai se lathos seira
                            tmpPtr->data = tmpNxt->data;
                            tmpNxt->data = tmp;

                             tmp1 = tmpPtr1->weight; //kanw to idio kai gia ta vari
                            tmpPtr1->weight = tmpNxt1->weight;
                            tmpNxt1->weight = tmp1;
                    }
                    tmpPtr = tmpPtr->next;
                    tmpPtr1 = tmpPtr1->next;
            }
            tmpPtr = list_head;
            tmpNxt = tmpNxt->next;
            tmpPtr1 = list_head;
            tmpNxt1 = tmpNxt1->next;
    }
}
```

Αφου εκτελεσαμε την εντολη READ_DATA απο το αρχειο commands σειρα ειναι να εκτελεσουμε και τις υπολοιπες εντολες , οι οποιες ειναι μια απο τις INSERT , DELETE , MST , NCN οπου βρισκονται με τυχαια σειρα .

INSERT:

Εστω οτι θελουμε να εκχωρησουμε την ακμη 1 3 5 . Θα ψαξουμε στον empty[i] αν υπαρχει το 1 .Αν δεν υπαρχει τοτε το εκχωρουμε και στην λιστα του 1 θα βαλουμε το 5 .Αν υπαρχει θα παμε στην θεση του και θα βαλουμε στην λιστα του το 5.Στην συνεχεια κανουμε ταξινομηση σε ολες τις λιστες που υπαρχουν ετσι ωστε η αναζητηση στην συνεχεια των κοινων γειτονων δυο κομβων να γινεται με λιγοτερο κοστος. Ακολουθουμε ακριβως την ιδια διαδικασια και για το 3 1 5.

```
    int b;
    b=search(key);
    if (empty[b])     //otan to stoixeio key den uparxei ston pinaka me ta ID tote to ekxwrw
    {
        empty[b]=false;
        ht[b].element=key;
        ht[b].add_node_to_list(key2,key3); // kai to ekxwroume stin lista tou sugkekrimenou id
    }
    if (ht[b].element==key)
    {
    ht[b].add_node_to_list(key2,key3);
    }
    //KANW AKRIVWS TIN IDIA DIADIKASIA KAI ME TO ALLO ID  EPEIDI EINAI MH KATEUTHINOMENO GRAFIMA
    //DILADI OPOU EIXA TO KEY BAZW TO KEY2 KAI TO ANTITHETO
    ht[b].sort_list();
     b=search(key2);
    if (empty[b])     //otan to stoixeio key den uparxei ston pinaka me ta ID tote to ekxwrw
    {
        empty[b]=false;
        ht[b].element=key2;
        ht[b].add_node_to_list(key,key3); // kai to ekxwroume stin lista tou sugkekrimenou id
    }
    if (ht[b].element==key2)
    {
    ht[b].add_node_to_list(key,key3);
    }
    ht[b].sort_list();
}
```

```
void ID::add_node_to_list(int d,int w)
{
     Node *temp,*neos;
     temp=list_head;
     bool flag;
     flag=false;
     while (list_head!=NULL)    //elegxw an uparxei o geitonas mesa stin lista
     {

          if (list_head->data==d)
          {

              flag=true;
              break;
          }
          list_head=list_head->next;
          }
     list_head=temp;
     if (flag==false)     //an den uparxei tote ton ekxwrw
     {
     neos=new Node;
     neos->data=d;
     neos->weight=w;
     neos->next=list_head;
     list_head=neos;
     }
}
```

DELETE:

Εστω οτι θελουμε να διαγραψουμε την ακμη 1 3. Θα ψαξουμε στον empty[i] αν υπαρχει το 1 .Αν δεν υπαρχει τοτε δεν κανουμε τιποτα.Αν υπαρχει τοτε στην λιστα του κανουμε αναζητηση το 3 και αν το βρουμε τοτε το διαγραφουμε. Ακολουθουμε ακριβως την ιδια

```
void Hashtable::delete_(int key,int key2)
{


    int b;
    b=search(key); //anazitw tin thesi tou kleidiou
    if (!empty[b])     //otan to stoixeio key uparxei ston pinaka me ta ID tote  diagrafw ton gei
    {
        empty[b]=true;
     ht[b].delete_node_from_list(key2);
    }

    //akolouthw tin idia diadikasia gia to antitheto zeugari
    b=search(key2);
    if (!empty[b])
     diagrafw ton geitona tou
    {
        empty[b]=true;
     ht[b].delete_node_from_list(key);
    }


}
```

διαδικασια και για το 3 1.

```
void ID::delete_node_from_list(int d)
{
    Node *temp,*temp1;
    temp=list_head;

    while (list_head!=NULL)          //elegxw an uparxei o geitonas stin lista kai ton diagrafw
    {
        if (list_head->data==d)
        {
            temp1=list_head->next;
            delete list_head;
            list_head=temp1;
             cout<<"delete node "<<d<<"  done"<<endl;
            break;
        }
        list_head=list_head->next;
    }

    list_head=temp;


}
```

```
int Hashtable::prim()
{
    bool flag=false; // simaia
    int k=1;
    int m=0;
    int min_cost=0; // elaxisto dentro
    int current=0;
    int visited[50000]; //komboi pou exoume episkeftei
    for (int i=0 ; i<size; i++) //psaxnoume na vroume enan komvo gia  na arxisoume na ftiaxnoune
    {                           //to elaxisto dentro . molis vroume enan tuxaio kombo tote ton ek
                                //ston pinaka visited kai vazoume -1 oles tis alles theseis
        if (empty[i]==false)
        {
            current=i;
         visited[0]=ht[current].element;
         break;
        }
    }

    for (int i=1; i<50000; i++)
    {
        visited[i]=-1;
    }
     struct N n;

    priority_queue<N> q;         //oura proteraiotitas gia na dialegoume kathe fora tin akmi me to
    for (int i=0; i<size; i++)
    {
```

MST : **Καλουμε την συναρτηση prim() για να βρουμε το ελαχιστο κοστος .Ξεκιναμε ψαχνωντας να βρουμε μια θεση στον πινακα hashtable που να περιεχει εναν κομβο δηλαδη να μην ειναι κενη, για να ξεκινησουμε να φτιαχνουμε το δεντρο απο καπου. Αφου βρουμε εναν τον αποθηκευουμε σε εναν πινακα visited[i] αφου τον επισκευθηκαμε.Παμε στην λιστα του ,παιρνουμε ολους τους γειτονες και τους αποθηκεουμε μεσα σε εναν σωρο μεγιστων αφου πρωτα κανουμε ελεγχο για το αν εχουμε επισκεφθει καποιον απο τους κομβους που εκχωρουμε.Αφου κανουμε push() τους καταλληλους στην συνεχεια παιρνουμε το πρωτο στοιχειο δλδ αυτου που εχει το μικροτερο βαρος και ελεγχω αν το εχω επισκεφθει ξανα, αν ναι τοτε το κανω pop() και παιρνω το επομενο αλλιως παιρνω το βαρος του και το προσθετω στο συνολικο βαρος που εχω βρει μεχρι εκεινη την στιγμη.Στην συνεχεια βρισκω την θεση του στον πινακα και ξεκιναω παλι την ιδια διαδικασια αφου το κανω pop().Ο αλγοριθμος εχει ως τερματικη συνθηκη το κενο του σωρου δλδ αν αδειασει τοτε σταματαει και επιστρεφει το συνολικο ελαχιστο κοστος.Με την συναρτηση clock() μετραω ποσο χρονο κανει μεχρι να μου βρει το ελαχιστο κοστος .**

```
int Hashtable::prim()
{
    bool flag=false; // simaia
    int k=1;
    int m=0;
    int min_cost=0; // elaxisto dentro
    int current=0;
    int visited[50000]; //komboi pou exoume episkeftei
    for (int i=0 ; i<size; i++) //psaxnoume na vroume enan komvo gia  na arxisoume na ftiaxnoune
    {                           //to elaxisto dentro . molis vroume enan tuxaio kombo tote ton ek
                                //ston pinaka visited kai vazoume -1 oles tis alles theseis
        if (empty[i]==false)
        {
            current=i;
         visited[0]=ht[current].element;
         break;
        }
    }

    for (int i=1; i<50000; i++)
    {
        visited[i]=-1;
    }
     struct N n;

    priority_queue<N> q;        //oura proteraiotitas gia na dialegoume kathe fora tin akmi me to
    for (int i=0; i<size; i++)
    {
```


```
int Hashtable::prim()
{
    bool flag=false; // simaia
    int k=1;
    int m=0;
    int min_cost=0; // elaxisto dentro
    int current=0;
    int visited[50000]; //komboi pou exoume episkeftei
    for (int i=0 ; i<size; i++) //psaxnoume na vroume enan komvo gia  na arxisoume na ftiaxnoune
    {                           //to elaxisto dentro . molis vroume enan tuxaio kombo tote ton ek
                                //ston pinaka visited kai vazoume -1 oles tis alles theseis
        if (empty[i]==false)
        {
            current=i;
         visited[0]=ht[current].element;
         break;
        }
    }

    for (int i=1; i<50000; i++)
    {
        visited[i]=-1;
    }
     struct N n;

    priority_queue<N> q;        //oura proteraiotitas gia na dialegoume kathe fora tin akmi me to
    for (int i=0; i<size; i++)
    {
```

NCN :

Για να βρουμε τους κοινους γειτονες 2 κομβων καλουμε την συναρτηση merging() .Κανουμε ελεγχο πρωτα για να δουμε αν υπαρχουν αυτοι οι κομβοι στον πινακα , αν δεν υπαρχουν τοτε δεν εχουν κοινους γειτονες .Αν υπαρχουν τοτε παμε στις λιστες τους και συγκρινουμε ενα ενα τα στοιχεια μεταξυ τους. Αν τα στοιχεια ειναι ιδια τοτε προχωραμα και στις δυο λιστες μαζι στο επομενο στοιχειο .Αν ειναι διαφορετικα τοτε μετακινουμε την λιστα που εχει μικροτερο id σε συγκριση με την αλλη.Αυτο το κανουμε επειδη οι λιστες ειναι ταξινομημενες και θα χουμε καλυτερη αποδοτικοτητα.

```
int Hashtable::merging(int key,int key2)
{
    int k,k2;
    Node *temp,*temp2;
    int mutuals=0;
    k=search(key); // psaxnw tin thesi tou prwtou kleidiou
    k2=search(key2); // psaxnw tin thesi tou deuterou kleidiou
    temp=ht[k].list_head;
    temp2=ht[k2].list_head;
    while(ht[k].list_head!=NULL && ht[k2].list_head!=NULL)
    {

        if (ht[k].list_head->data == ht[k2].list_head->data) // an exoun ton idio kombo tote
        {    //paw sto epomeno stoixeio kai apo tis 2 alusides
        ht[k].list_head=ht[k].list_head->next;
        ht[k2].list_head=ht[k2].list_head->next;
        mutuals++;  //oi mutuals auksanontai kata 1
        }
        else if (ht[k].list_head->data < ht[k2].list_head->data)
        { //metakinoumaste kathe fora tin alusida pou periexei megalutero stoixeio se sugkrisi met
            ht[k].list_head=ht[k].list_head->next;
        }
        else
         ht[k2].list_head=ht[k2].list_head->next;
    }
    ht[k].list_head=temp;
    ht[k2].list_head=temp2;
    return mutuals;
```

Το ελαχιστο κοστος , ο χρονος του αλγοριθμου prim καθως και τους κοινους γειτονες τα αποθηκευουμε σε ενα αρχειο με ονομα output.txt

Τελος προγραμματος .