Computing Science 204 (B1)

Connected Components of a Graph

Friday, August 11

Algorithms for Finding the Connected Components of a Graph

Let $G = (V, E)$ be a graph which is represented by an adjacency list. In this note we implement procedures to find the connected components of the graph G. The input to the procedures is the set of vertices V and the set of edges E. The output is the set of connected components of G, each component is labelled and contains the adjacency lists for all vertices in that component.

We give two algorithms for finding the connected components of an undirected graph $G = (V, E)$, the first is a stack-based implementation of the depth-first search algorithm, while the second is a recusive version of depth-first search.

Nonrecursive Connected Component Algorithm

```
#include<stdio.h>
#define maxV = 100

struct node
{
      int key;
      struct node *next;
};

struct snode
{
      int key;
      struct snode *next;
};

struct node *z,*adj[maxV];

struct snode *sz,*head;

int V,E,componentnumber=0,dfsnumber=0;

int mark[maxV],component[maxV];

initarrays()
{
      for (j = 0; j < maxV; j++)
      {
           mark[j] = 0;
           component[j] = 0;
      }
}
```

```
adjlist()
{
      int j,x,y;
      struct node *t;

      printf("Enter the number of Vertices and Edges:\n");
      scanf("%d %d",&V,&E);

      z = (struct node *) malloc(sizeof *z);
      z->key = 0;
      z->next = z;

      for (j = 0; j <= V; j++)
           adj[j] = z;

      printf("Enter the edges x y:\n");

      for (j = 1; j <= E; j++)
      {
           scanf("%d %d",&x,&y);

           t = (struct node *) malloc(sizeof *t);
           t->key = x;
           t->next = adj[y];
           adj[y] = t;

           t = (struct node *) malloc(sizeof *t);
           t->key = y;
           t->next = adj[x];
           adj[x] = t;
      }
}

stackinit()
{
      sz = (struct snode *) malloc(sizeof *sz);
      head = (struct snode *) malloc(sizeof *head);
      sz->next = sz;
      head->next = sz;
      head->key = 0;
}

int stackempty()
{
      return (head->next == sz);
}
```

```
push(v)
int v;
{
      struct snode *t;

      t = (struct snode *)malloc(sizeof *t);
      t->key = v ;
      t->next = head->next;
      head->next = t;
}

int pop()
{
      int x;
      struct snode *t;

           t = head->next;
           head->next = t->next;
           x = t->key;
           free(t);
           return x;
}

visit(k)
int k;
{
      struct node *t;

      push(k);

      while (!stackempty())
      {
           k = pop();
           dfsnumber++;;
           mark[k] = dfsnumber;
           component[k] = componentnumber;

           t = adj[k];

           while (t != z)
           {
               if (mark[t->key] == 0)
               {
                   push(t->key);
                   mark[t->key] = -1;
               }
               t = t->next;
           }
      }
}
```

```
dfs()
{
      int k;

      stackinit();

      for (k = 1; k <= V; k++)
      {
           if (mark[k] == 0)
           {
               componentnumber++:
               visit(k);
           }
      }
}

printoutput()
{
      int i,k;
      struct node *p;

      for (k = 1; k <= componentnumber; k++)
      {
           printf("\nCOMPONENT # %d :  ",k);
           for (i = 1; i <= V; i++)
           {
               if (component[i] == k)
               {
                   printf("\nadj[%d] :  ",i);
                   p = adj[i];
                   while (p != z)
                   {
                       printf("%4d",p->key);
                       p = p->next;
                   }
               }
           }
           printf("\n");
      }
}

main()
{
      adjlist();
      initarrays();
      stackinit();
      dfs();
      printoutput();
}
```

Recursive Connected Component Algorithm

```
#include<stdio.h>
#define maxV = 100

struct node
{
      int key;
      struct node *next;
} *z,*adj[maxV];

int V,E,componentnumber=0;
int mark[maxV];

initarrays()
{
      for (j = 0; j < maxV; j++)
           mark[j] = 0;
}

adjlist()
{
      int j,x,y;
      struct node *t;

      printf("Enter the number of Vertices and Edges:\n");
      scanf("%d %d",&V,&E);

      z = (struct node *) malloc(sizeof *z);
      z->v = 0;
      z->next = z;

      for (j = 0; j <= V; j++)
           adj[j] = z;

      printf("Enter the edges x y:\n");

      for (j = 1; j <= E; j++)
      {
           scanf("%d %d",&x,&y);

           t = (struct node *) malloc(sizeof *t);
           t->v = x;
           t->next = adj[y];
           adj[y] = t;

           t = (struct node *) malloc(sizeof *t);
           t->v = y;
           t->next = adj[x];
           adj[x] = t;
      }
}
```

```
connectedcomponents()
{
      int k;

      for (k = 1; k <= V; k++)
      {
           if (mark[k] == 0);
           {
               componentnumber++;
               dfs(k);
           }
      }
}

dfs(v)
int v;
{
      int w;
      struct node *p;

      mark[v] = componentnumber;
      p= adj[v];

      while (p != z)
      {
           w = p->v;
           if (mark[w] == 0)
               dfs(w);
           p = p->next;
      }
}
printoutput()
{
      int i,k;
      struct node *p;

      for (k = 1; k <= componentnumber; k++)
      {
           printf("\nCOMPONENT # %d :  ",k);
           for (i = 1; i <= V; i++)
           {
               if (mark[i] == k)
               {
                   printf("\nadj[%d] :  ",i);
                   p = adj[i];
                   while (p != z)
                   {
                       printf("%4d",p->key);
                       p = p->next;
                   }
               }
           }
           printf("\n");
      }
}
```

```
main()
{
      adjlist();
      initarrays();
      connectedcomponents();
      printoutput();
}
```