STRUTTURE DATI e LABORATORIO II
Esercitazione n° 15

Grafi rappresentati mediante liste di adiacenza

Scrivere un programma in C che implementi la struttura dati grafo mediante le liste di adiacenza. Si richiede che

- l'input sia costituito dal numero dei vertici e per ciascun vertice siano assegnati i vertici adiacenti;
- si possa visitare il grafo in profondità ed in ampiezza.

Suggerimento. Le dichiarazioni dell'algoritmo possono essere

```
typedef struct nod1 {
        int vertice;
        struct nod1 *link;
        }nodo;
typedef nodo *nodo_pointer;
nodo_pointer grafo[MAX_VERTICI];
```

La funzione main dell'algoritmo può essere basata sulla seguente funzione menu.

```
int menu(void)
{
      int buf;
      printf("\nIndica l'operazione da eseguire:\n\n");
      printf("crea il grafo                        --> 1\n");
      printf("visita in profondità                 --> 2\n");
      printf("visita in ampiezza                   --> 3\n");
      printf("termina                              --> 0\n");
      scanf("%d",&buf);
      return buf;
}
```

in cui ad ogni scelta corrisponderà una funzione.
La funzione di creazione del grafo può essere

```
void insert(void)
{
nodo_pointer temp, current;
int i, buf;

printf("\nDai il numero di vertici \n");
scanf("%d",&n);
for (i = 0; i < n; i++) {
   grafo[i] = (nodo_pointer) malloc(sizeof(nodo));
   grafo[i]->vertice = i;     grafo[i]->link = NULL;
   current = grafo[i];
   while(1) {
      printf("\ndai i vertici connessi a %3d \n",i);
      printf("(valori tra 0 e %3d escluso %3d; -1 termina)", n-1,i);
       scanf("%d",&buf);
      if (buf == -1)    break;
          else if (buf <=-1 || buf > n-1 || buf==i)   continue;
          temp = (nodo_pointer) malloc(sizeof(nodo));
          temp->link = NULL;         temp->vertice = buf;
          current->link = temp;      current = current->link;
```

```
        }
     }
}
```

La ricerca in profondità può essere implementata mediante

```
void rpr(int v)   /*ricerca in profondità di un grafo*/
{
  nodo_pointer w;

  printf("%5d  ",v);    visitato[v]=TRUE;
  for(w = grafo[v]; w; w = w->link)
      if(!visitato[w->vertice]) rpr(w->vertice);
}
```

La ricerca in ampiezza può essere implementata mediante

```
void ram(int v)   /*ricerca in ampiezza di un grafo */
{
  coda_pointer davanti, dietro;
  nodo_pointer w;

  davanti=dietro=NULL;              /*inizializza la coda*/
  printf("%5d",v);      visitato[v] = TRUE;
  addc(&davanti,&dietro, v);
  while (davanti) {
      v = deletec(&davanti);
      for(w = grafo[v]; w; w = w->link)
            if (!visitato[w->vertice]) {
                  printf("%5d", w->vertice);
                  addc(&davanti, &dietro, w->vertice);
                  visitato[w->vertice] = TRUE;
                  }
      }
}

void addc(coda_pointer *davanti, coda_pointer *dietro , int item)
{                       /*aggiunge un elemento alla coda*/
  coda_pointer temp;
  temp=(coda_pointer) malloc(sizeof(coda));
  if(temp == NULL) {
      printf("\nla memoria è piena");     exit(1);    }
  temp->vertice = item;       temp->link = NULL;
  if (*davanti)   (*dietro)->link = temp;
  else            *davanti = temp;
  *dietro = temp;
}

int deletec(coda_pointer  *davanti)
{                       /*cancella il primo elemento di una coda*/
  coda_pointer temp = *davanti;
  int item;

  if (!*davanti)  { printf("\nla coda è vuota\n");   exit(1);    }
  item = temp->vertice;       *davanti = temp->link;
  free(temp);
  return item;
}
```

Da notare che prima di una visita è necessario “azzerare” l’array visitato[ ].
Buon lavoro!