**MINISTERUL EDUCAȚIEI ȘI CERCETĂRII AL REPUBLICII MOLDOVA**

**Universitatea Tehnică a Moldovei**

**Facultatea Calculatoare, Informatică şi Microelectronică**

**Departamentul Informatică şi Ingineria Sistemelor**

**RAPORT**

Lucrare de laborator nr 1

la cursul „***Matematica Discretă***”

**Tema: P*ă*strarea grafului în memoria calculatorului**

A efectuat : **St. gr. CR-221FR Serba Cristina**

A verificat: **Asis.univ. Orîndaş Victoria**

**Chișinău 2022**

**Cuprins**

[INTRODUCERE 2](#_Toc133315963)

[REALIZAREA PRACTICĂ A SARCINII DE LUCRU 3](#_Toc133315964)

[CONCLUZII 13](#_Toc133315965)

# 

# INTRODUCERE

Anul 1736 este considerat pe buna dreptate de inceput pentru teoria grafurilor. In acel an

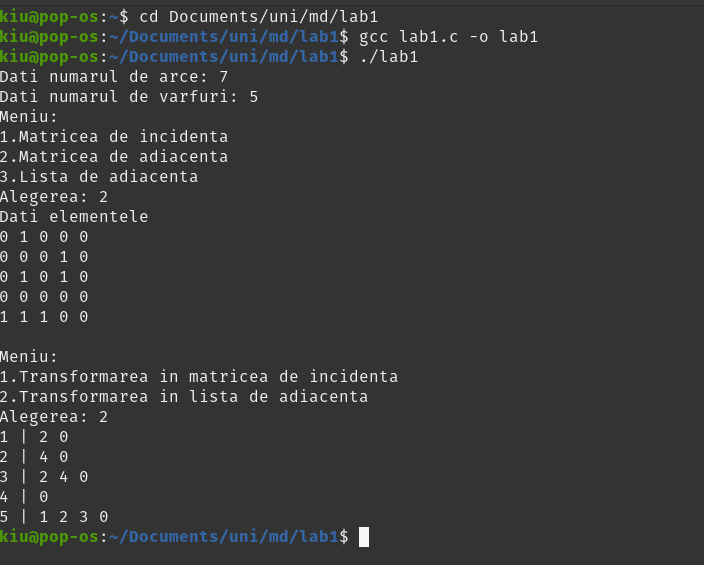
L. Euler a rezolvat problema despre podurile din Konigsberg,stabilind criteriul de existent in grafuri a unui circuit special,denumit astazi ciclu ciclu Euler.Avindusi inceputurile in rezolvarea unor jocuri distractive.astazi teoria grafurilor s-a transformat intr-un aparat simplu si accesibil, care permite rezolvarea unui cerc larg de probleme. Sub forma de grafuri pot fi reprezentate sisteme de drumuri si circuite electrice, harti geografice si molecule chimice, relatii dintre oameni si grupuri de oameni. Teoria grafurilor a devenit o parte componenta a aparatului matematic al ciberneticii,limbajul matematicii discrete.

**Def. Grafului** Se numeste graf ansamblu format dintr-o multime finite X si o aplicatie F a lui X in X. Se noteaza G=(X,F). Numarul elementelor multimilor X determina ordinal grafului finit. Daca card X=n, graful G=(X,F) se numeste graf finit de ordinul n. Elementele multimii X se numesc varfurile grafului. Geometric, varfurile unui graf le reprezentam prin puncte sau cerculete. Perechea de varfuri (x,y) se numeste arc varful x se numeste originea sau extremitatea initiala a arcului (x,y) iar varful y se numeste extremitatea finala sau terminal. Un arc (x,y) il reprezentam geometric printr-o sageata orientate de la varful x la varful y. Daca un varf nu este extremitatea nici unui arc el se numeste varf izolat, iar daca este extremitatea a mai mult de doua arce- nod. Un arc (x,y) pentru care extremitatea initiala coincide cu cea finala se numeste bucla.Arcele unui graf le mai notam si cu u1,u2,..., iar multimea arcelor grafului o noatam cu U. Doua arce se numesc adiacente daca sunt distncte si au o extremitate comuna.Doua varfuri se numesc adiacente daca sunt distinct si sunt unite prtr-un arc. Un arc (x,y) se spune ca este incident cu virful x spre exterior si este incident cu varful y spre interior.

# REALIZAREA PRACTICĂ A SARCINII DE LUCRU

**Scopul lucrarii:** Studierea metodelor de definire a unui graf:matrice de incidenta, matrice de adiacenta,liste. Elaborarea unor procedure de introducere, extragere si transformare a diferitelor forme de reprezentare interna a grafurilor cu scoaterea rezultatelor la display si imprimanta.

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <stdio.h>  struct node  {  int val;  struct node\* head;  }\*list;  */\*MA = {{0, 1, 0, 0, 0},*  *{0, 0, 0, 1, 0},*  *{0, 1, 0, 1, 0},*  *{0, 0, 0, 0, 0},*  *{1, 1, 1, 0, 0}}\*/*  */\*MI = {{0, 1, 0, 0, -1},*  *{1, 0, 0, 0, -1},*  *{-1, 1, 0, 0, 0},*  *{0, 0, 1, 0, -1},*  *{0, 0, -1, 1, 0},*  *{0, 1, -1, 0, 0},*  *{0, -1, 0, 1, 0}}\*/*  */\*LA = {{1 | 2, 0},*  *{2 | 4, 0},*  *{3 | 2, 4, 0},*  *{4 | 0},*  *{5 | 1, 2, 3, 0}}\*/*  void MI(); *//matricea de incidenta*  void LA(); *//lista de adiacenta*  void MA(); *//matricea de adiacenta*  void MAtoMI(int \*\*MA, int \*\*MI, int varf, int arc); *//din matricea de adiacenta in matricea de incidenta*  void MItoLA(int \*\*MI, struct node \*\*LI, int varf, int arc); *//din matricea de incidenta in lista de adiacenta*  void LAtoMA(struct node \*\*LA, int \*\*MA, int varf, int arc); *//din lista de adiacenta in matricea de adiacenta*  int \*\*allocMatrix(int varf, int arc);  void insertMatrix(int \*\*\*a, int varf, int arc);  struct node\*\* allocArrayOfList(int arc);  void insertArrayOfList(struct node \*\*\*arrayOflist, int arc);  void insert\_node(struct node \*initial, int valoare);  void printList(struct node\* list);  void printMatrix(int \*\*a, int varf, int arc);  void freeMatrix(int \*\*a, int arc);  void freeList(struct node\* list);    int main()  {  int choice;  int varf, arc;  printf("Dati numarul de arce: "); scanf("%d", &arc);  printf("Dati numarul de varfuri: "); scanf("%d", &varf);  printf("Meniu:**\n**1.Matricea de incidenta**\n**2.Matricea de adiacenta**\n**3.Lista de adiacenta**\n**Alegerea: ");  scanf("%d", &choice);  switch(choice)  {  case 1: MI(varf, arc);  **break**;  case 2: MA(varf, arc);  **break**;  case 3: LA(varf, arc);  **break**;  default: printf("Alegere invalida**\n**");  **break**;  }  return 0;  }  void MI(int varf, int arc)  {  */\*INTRODUCERE DATE\*/*  int \*\*matrice;  insertMatrix(&matrice, varf, arc);    */\*TRANSFORMARI\*/*  printf("**\n**Meniu:**\n**1.Transformarea in matricea de adiacenta**\n**2.Transformarea in lista de adiacenta**\n**Alegerea :");  int choice;  scanf("%d", &choice);  switch(choice)  {  case 1:  {  int \*\*MA = allocMatrix(varf, varf);  struct node\*\* LA = allocArrayOfList(varf);  MItoLA(matrice, LA, varf, arc);  LAtoMA(LA, MA, varf, arc);  printMatrix(MA, varf, varf);  freeMatrix(MA, arc);  for(int i = 0; i < varf; i++)  {  freeList(LA[i]);  }  free(LA);  freeMatrix(MA, varf);  **break**;  }  case 2:  {  struct node\*\* LA = allocArrayOfList(varf);  MItoLA(matrice, LA, varf, arc);  printf("Lista este **\n**");  for(int i = 0; i < varf; i++)  {  printList(LA[i]);  }  for(int i = 0; i < varf; i++)  {  freeList(LA[i]);  }  free(LA);  **break**;  }  default:  printf("Alegere invalida**\n**");  **break**;  }    */\*ELIBERARE MEMORIE\*/*  freeMatrix(matrice, arc);  }  void MA(int varf, int arc)  {  */\*INTRODUCERE DATE\*/*  int \*\*matrice;  insertMatrix(&matrice, varf, varf);    */\*TRANSFORMARI\*/*  printf("**\n**Meniu:**\n**1.Transformarea in matricea de incidenta**\n**2.Transformarea in lista de adiacenta**\n**Alegerea: ");  int choice;  scanf("%d", &choice);  switch(choice)  {  case 1:  {  int \*\*MI = allocMatrix(varf, arc);  MAtoMI(matrice, MI, varf, arc);  printMatrix(MI, varf, arc);  freeMatrix(MI, arc);  **break**;  }    case 2:  {  struct node \*\*LA = allocArrayOfList(varf);  int \*\*MI = allocMatrix(varf, arc);  MAtoMI(matrice, MI, varf, arc);  MItoLA(MI, LA, varf, arc);  for(int i = 0; i < varf; i++)  {  printList(LA[i]);  }  for(int i = 0; i < varf; i++)  {  freeList(LA[i]);  }  free(LA);  freeMatrix(MI, arc);  **break**;  }  default:  printf("Alegere invalida**\n**");  **break**;  }    */\*ELIBERARE MEMORIE\*/*  freeMatrix(matrice, varf);  }  void LA(int varf, int arc)  {  */\*INTRODUCERE DATE\*/*  struct node \*\*arrayOflist = allocArrayOfList(varf);    insertArrayOfList(&arrayOflist, varf);    */\*AFISARE LISTA\*/*  printf("Lista de adiacenta:**\n**");  for(int i = 0; i < varf; i++)  {  printList(arrayOflist[i]);  }    */\*TRANSFORMARI\*/*  printf("Meniu: **\n**1. Transformarea in matricea de incidenta**\n**2. Transformarea in matricea de adiacenta**\n**Alegerea: ");  int choice;  scanf("%d", &choice);  switch(choice)  {  case 1:  {  int \*\*MI = allocMatrix(varf, arc);  int \*\*MA = allocMatrix(varf, varf);  LAtoMA(arrayOflist, MA, varf, arc);  MAtoMI(MA, MI, varf, arc);  printMatrix(MI, varf, arc+1);  freeMatrix(MI, arc);  freeMatrix(MA, varf);  **break**;  }  case 2:  {  int \*\*MA = allocMatrix(varf, varf);  LAtoMA(arrayOflist, MA, varf, arc);  printMatrix(MA, varf, varf);  freeMatrix(MA, varf);  **break**;  }  default: printf("Optiune invalida**\n**");  **break**;  }    */\*ELIBERARE MEMORIE\*/*  for(int i = 0; i < varf; i++)  {  freeList(arrayOflist[i]);  }  free(arrayOflist);    }  void MAtoMI(int \*\*MA, int \*\*MI, int varf, int arc)  {  int k = 0;  for(int i = 0; i < varf; i++)  {  for(int j = 0; j < varf; j++)  {  *//daca pe randul i este elementul 1*  *//atunci in matricea de incidenta coloana i va avea -1*  *//iar coloana j va avea 1*  if(MA[i][j])  {  MI[k][i] = -1;  MI[k][j] = 1;  k++;  }  }  }  }  void MItoLA(int \*\*MI, struct node \*\*LI, int varf, int arc)  {  for(int i = 0; i < arc; i++)  {  if(i < varf)  {  LI[i]->val = i+1; *//adauga indicii pentru fiecare rand*  LI[i]->head = NULL;  }  for(int j = 0; j < varf; j++)  {  if(MI[i][j] == -1) *//daca din varful i iese arc*  {  for(int k = 0; k < varf; k++)  {    if(MI[i][k] == 1) insert\_node(LI[j], k+1); *//va cauta varful in care intra arcul si va adauga la lista pe pozitia i*  }  **break**;  }  }  }  for(int i = 0 ; i < varf; i++) insert\_node(LI[i], 0); *//va adauga elementul 0 la sfarsitul fiecarul rand*  }    void LAtoMA(struct node \*\*LA, int \*\*MA, int varf, int arc)  {  struct node \*temp;  for(int i = 0; i < varf; i++)  {  int r = LA[i]->val; *//varfurile din care ies arce*  for(temp = LA[i]->head; temp->val != 0; temp = temp->head) *//varfurile in care intra arce*  {  MA[r-1][(temp->val)-1] = 1;  }  }  }  int \*\*allocMatrix(int varf, int arc)  {  int \*\*a = calloc(arc, sizeof(int\*));  if(!a)  {  printf("Matricea nu a fost alocata**\n**");  exit(1);  }  for(int i = 0; i < arc; i++)  {  a[i] = calloc(varf, sizeof(int));  if(!a[i])  {  printf("Matricea nu a fost alocata**\n**");  exit(1);  }  }  return a;  }  void insertMatrix(int \*\*\*a, int varf, int arc)  {  \*a = allocMatrix(varf, arc);  printf("Dati elementele**\n**");  for(int i = 0; i < arc; i++)  for(int j = 0; j < varf; j++)  {  scanf("%d", &(\*a)[i][j]);  }  }    struct node\*\* allocArrayOfList(int varf)  {  struct node \*\*arrayOflist = malloc(varf\*sizeof(struct node\*));  if(!arrayOflist)  {  printf("Alocare esuata**\n**");  exit(1);  }  for(int i = 0; i < varf; i++)  {  arrayOflist[i] = malloc(sizeof(struct node));  if(!arrayOflist[i])  {  printf("Alocare esuata**\n**");  exit(1);  }  }  return arrayOflist;  }  void insertArrayOfList(struct node \*\*\*arrayOflist, int varf)  {  for(int i = 0; i < varf; i++)  {  printf("Dati numarul coloanei:");  scanf("%d", &((\*arrayOflist)[i]->val));  int node;  (\*arrayOflist)[i]->head=NULL;  printf("Dati arcele:");  do  {  scanf("%d", &node);  insert\_node((\*arrayOflist)[i], node);  }while(node);  }  }  void insert\_node(struct node \*initial, int valoare)  {  struct node \*temp = NULL;  struct node \*curent = (struct node\*) malloc(sizeof(struct node));  if(!curent)  {  printf("Alocare esuata**\n**");  exit(1);  }  curent->val = valoare;  curent->head = NULL;  *//temp si initial vor avea aceeasi adresa in memorie*  temp = initial;  while(temp->head != NULL)  {  temp = temp->head; *//trece prin toate nodurile pana ajunge la ultimul*  }  temp->head = curent; *//adauga noul nod la capat*  }  void printList(struct node\* list)  {  printf("%d | ", list->val);  list = list->head;  while(list)  {  printf("%d ", list->val);  list=list->head;  }  printf("**\n**");  }  void printMatrix(int \*\*a, int varf, int arc)  {  printf("Matricea este **\n** ");  for(int j = 1; j <= varf; j++)  {  printf(" x%d", j);  }  for(int i = 0; i < arc; i++)  {  printf("**\n**x%d ", i+1);  for(int j = 0; j < varf; j++)  {  printf("%d ", a[i][j]);  }  }  }  void freeMatrix(int \*\*a, int arc)  {  for(int i = 0; i < arc; i++) free(a[i]);  free(a);  }  void freeList(struct node\* list)  {  struct node\* tmp;  while (list)  {  tmp = list;  list = list->head;  free(tmp);  }  } |

Fig 1. Exemplu de executie a programului

# CONCLUZII

În concluzie, alegerea unei structuri de date pentru reprezentarea unui graf în memoria unui calculator depinde de cerințele specifice ale aplicației.

Matricele de adiacență sunt ideale pentru grafuri dense, în care numărul de muchii este aproape de maximul posibil. Acestea oferă acces în timp constant la existența unei muchii între două vârfuri, dar necesită o cantitate mare de memorie proporțională cu pătratul numărului de vârfuri.

Listele de adiacență permit traversarea eficientă a grafului, dar accesul la existența unei muchii între două vârfuri poate dura timp liniar în cel mai rău caz.

Matricele de incidență reprezintă un compromis între matricele și listele de adiacență. Acestea oferă acces în timp constant la existența unei muchii și utilizează memorie proporțională cu numărul de vârfuri și muchii, dar traversarea grafului necesită operații mai complexe și poate dura mai mult decât în cazul listelor de adiacență.

În general, alegerea unei structuri de date pentru un graf în memoria unui calculator depinde de factori precum dimensiunea și densitatea grafului, operațiile necesare și memoria disponibilă.