**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. 2, 3

la cursul „**Matematica discretă**”

**Tema: Drumul de valoare minimă şi maximă. Algoritmii Ford şi Bellman-Kallaba**

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

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

**Chișinău 2022**

**Cuprins**

[INTRODUCERE 2](#_Toc2064279830)

[REALIZAREA PRACTICĂ A SARCINII DE LUCRU 4](#_Toc508494632)

[CONCLUZII 12](#_Toc1556059285)

# 

# INTRODUCERE

Pentru un graf orientat *G = (X,U)* se va numi *drum* un şir de vârfuri *D* = *(x0, x1,..., xr)* cu proprietatea că *(x0, x1)*, *(x1, x2)*,..., *(xr-1, xr)* aparţin lui *U*, deci sunt arce ale grafului şi extremitatea finală a arcului precedent coincide cu extremitatea iniţială a arcului următor.

Adeseori, fiecărui arc (muchii) i se pune în corespondenţă un număr real care se numeşte *ponderea* (lungimea) arcului. Lungimea arcului *(xi, xj)* se va nota *w(i,j)*, iar în cazul în care un arc este lipsă ponderea lui va fi considerată foarte mare (pentru calculator cel mai mare număr pozitiv posibil). În cazul grafurilor cu arce ponderate (grafuri ponderate) se va considera lungime a unui drum suma ponderilor arcelor care formează acest drum. Drumul care uneşte două vârfuri concrete şi are lungimea cea mai mică se va numi *drum minim* iar lungimea drumului minim vom numi *distanţă*.

**Algoritmul lui Ford** pentru detrminarea drumului minim permite determinarea drumului minim care începe cu un vârf iniţial xi până la oricare vârf al grafului G. Dacă prin Lij se va nota ponderea arcului (xi, xj) atunci algoritmul conţine următorii paşi:

1. Fiecărui vârf *xj* al grafului *G* se va ataşa un număr foarte mare *Hj(∞)*. Vârfului iniţial i se va ataşa *Ho = 0*;
2. Se vor calcula diferenţele H*j- Hi* pentru fiecare arc *(xi, xj)*. Sunt posibile trei cazuri:
3. *Hj - Hi*  *< Lij,*
4. *Hj - Hi*  *= Lij,*
5. *Hj - Hi*  *> Lij.*

Cazul "*c*" permite micşorarea distanţei dintre vârful iniţial şi *xj* din care cauză se va realiza *Hj = Hi + Lij*.

Pasul 2 se va repeta atâta timp cât vor mai exista arce pentru care are loc inegalitatea “c”. La terminare, etichetele *Hi* vor defini distanţa de la vârful iniţial până la vârful dat *xi*.

1. Acest pas presupune stabilirea secvenţei de vârfuri care va forma drumul minim. Se va pleca de la vârful final *xj* spre cel iniţial. Predecesorul lui *xj* va fi considerat vârful *xi* pentru care va avea loc *Hj - Hi = Lij*. Dacă vor exista câteva arce pentru care are loc această relaţie se va alege la opţiune.

**Algoritmul Bellman – Kalaba** permite determinarea drumului minim dintre oricare vârf al grafului până la un vârf, numit vârf final.

Etapa iniţială presupune ataşarea grafului dat *G* a unei matrice ponderate de adiacenţă, care se va forma în conformitate cu următoarele:

1. *M(i,j)* = *Lij*, dacă există arcul *(xi, xj)* de pondere *Lij*;
2. *M(i,j)* = ∞, unde ∞ este un număr foarte mare (de tip întreg maximal pentru calculatorul dat), dacă arcul *(xi, xj)* este lipsă;
3. M(i,j) = 0, dacă i = j.

La etapa a doua se va elabora un vector V0 în felul următor:

1. *V0(i) = Lin*, dacă există arcul *(xi, xn)*, unde *xn* este vârful final pentru care se caută drumul minim, *Lin* este ponderea acestui arc;
2. *V0(i) =* ∞, dacă arcul (x*i, xn)* este lipsă;
3. *V0(i) = 0*, dacă *i= j*.

Algoritmul constă în calcularea iterativă a vectorului V în conformitate cu următorul procedeu:

1. *Vk(i) = min{Vk-1; Lij+Vk-1(j)}*, unde *i = 1, 2,…, n - 1, j = 1, 2,..., n*; *i<>j;*
2. *Vk(n) = 0*.

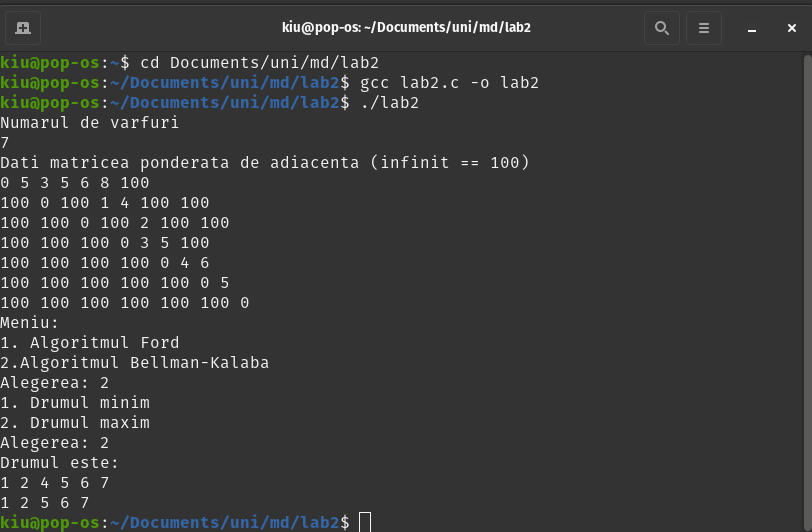
Când se va ajunge la Vk = Vk-1 - STOP.

Componenta cu numărul *i* a vectorului *Vk* cu valoarea diferită de zero ne va da valoarea minimă a drumului care leagă vârful *i* cu vârful *n*.

# REALIZAREA PRACTICĂ A SARCINII DE LUCRU

**Scopul lucrării:** Studierea algoritmilor de determinare a drumurilor minime într-un graf. Elaborarea programelor de determinare a drumului minim într-un graf ponderat.

|  |
| --- |
| #include <stdio.h>  #include <stdio.h>  #include <stdlib.h>  int prec[7], curent[7], varf, choice;  */\*int matrice[7][7] = {{0, 5, 3, 5, 6, 8, 100},*  *{100, 0, 100, 1, 4, 100, 100},*  *{100, 100, 0, 100, 2, 100, 100},*  *{100, 100, 100, 0, 3, 5, 100},*  *{100, 100, 100, 100, 0, 4, 6},*  *{100, 100, 100, 100, 100, 0, 5},*  *{100, 100, 100, 100, 100, 100, 0}};\*/*  int \*\*matrice; *//matricea ponderata de adiacenta*  #define min(a, b) (a<b) ? a : b;  #define max(a, b) (a>b) ? a : b;  int \*\*allocMatrix();  void createMatrix();  void input();  void Ford();  void BellmanKalaba();  void PathFord(int H2[7],int vcurent, int indice);  void PathBellman(int H2[7],int vcurent, int indice);  int cmpArr(int \*A, int \*B, int varf);  void cpyArr(int \*A, int \*B, int varf);  void freeArr();    int main()  {    printf("Numarul de varfuri**\n**");  scanf("%d", &varf);  createMatrix(varf);  printf("Meniu:**\n**1. Algoritmul Ford**\n**2.Algoritmul Bellman-Kalaba**\n**Alegerea: ");  scanf("%d", &choice);  switch(choice)  {  case 1:  {  Ford(matrice);  **break**;  }  case 2:  {  BellmanKalaba(matrice);  **break**;  }  }  freeArr();  return 0;  }  void Ford()  {  printf("1. Drumul minim**\n**2. Drumul maxim**\n**Alegerea: ");  int c;  scanf("%d", &c);  int \*H = calloc(varf, sizeof(int));  int \*H2 = calloc(varf, sizeof(int));  *//initializeaza varfurile cu etichete*  switch(c)  {  case 1:  {  for(int i = 0; i < varf; i++)  {  H2[i] = 100;  }  **break**;  }  case 2:  {  for(int i = 0; i < varf; i++)  {  H2[i] = -100;  }  **break**;  }  }  H2[0] = 0; *//varful initial este 0*  do{  for(int i = 0; i < varf; i++)  {  cpyArr(H, H2, varf);  cpyArr(H2, H, varf);  for(int j = 0; j < varf; j++)  {  *//trece peste elementele de sub diagonala principala*  *//si celelalte unde nu este varf*  if(i != j && matrice[i][j] != 100)  {  switch(c)  {  case 1:  {  if(H2[j] - H[i] > matrice[i][j])  {  H2[j] = H[i] + matrice[i][j];  }  **break**;  }  case 2:  {  if(H2[j] - H[i] < matrice[i][j])  {  H2[j] = H[i] + matrice[i][j];  }  **break**;  }  }  }  }  }  }while(cmpArr(H, H2, varf)!= 0); *//se va opri cand niciun varf nu isi va mai schimba eticheta*  printf("Drumul este **\n**");  PathFord(H2, varf-1, 101);  }  void BellmanKalaba()  {  printf("1. Drumul minim**\n**2. Drumul maxim**\n**Alegerea: ");  int H2[varf], H[varf];  int c;  scanf("%d", &c);  switch(c)  {  case 1:  {  int minim;  for(int i = 0; i < varf; i++)  {  H2[i] = matrice[i][varf-1]; *//initializam vectorul 0 cu elementele de pe ultima coloana*  }  do{  cpyArr(H, H2, varf);  cpyArr(H2, H, varf);  for(int i = 0; i < varf; i++)  {  minim = matrice[i][i+1] + H[i+1];  for(int j = 0; j < varf-1; j++)  {  int k = j+1;  if(i != j)  {  if(i == k) k = j+2; *//daca urmatorul element al matricii este 0, il ignora*  minim = min(minim, matrice[i][k] + H[k]); *// min dintre elementul curent si urmatorul*  H2[i] = minim;  }  }  H2[varf-1] = 0;  }  }while(cmpArr(H2, H, varf) != 0); *//se va opri cand nicio eticheta nu se va mai schimba*  **break**;  }  case 2:  {  for(int i = 0; i < varf; i++)  {  for(int j = 0; j < varf; j++)  {  if(matrice[i][j] == 100) matrice[i][j] = -100; *//pentru drumul maxim folosim -inf*  }  H2[i] = matrice[i][varf-1];  }  int maxim;  do{  cpyArr(H, H2, varf);  cpyArr(H2, H, varf);  for(int i = 0; i < varf; i++)  {  maxim = matrice[i][i+1] + H[i+1];  for(int j = 0; j < varf-1; j++)  {  int k = j+1;;  if(i != j)  {  if(i == k) k = j+2;  maxim = max(maxim, matrice[i][k] + H[k]);  H2[i] = maxim;  }  }  H2[varf-1] = 0;  }  }while(cmpArr(H2, H, varf) != 0);  **break**;  }  }  printf("Drumul este: **\n**");  PathBellman(H2, 0, varf+1);  }  int vprec = 0, k = 0;  void PathFord(int H2[varf],int vcurent, int indice)  {  *//vcurent = varful curent*  *//k = indicele drumului curent*  *//indice = indicele drumului pana unde acesta nu se imparte in altul*  *//prec[] = drumul precedent (daca drumul a fost impartit in mai multe)*  *//curent[] = drumul curent*  *//vprec = varful precedent unde s-a impartit drumul*    if(indice < 100) *//testeaza daca incepe un nou drum*  {  int i;  for(i = 0; i < vprec; i++)  {  if(indice == prec[i])  {  **break**;  }  }  for(int j = vprec - 1; j >= i; j--)  curent[k++] = prec[j];*//copie toate valorile de pana la varful nou*  indice = 101; *//indicele nu poate fi mai mare decat nr de varfuri*  }  curent[k++] = vcurent; *//adauga varful curent la drum*  if (vcurent == 0)  {  for (int i = k-1; i >= 0; i--)  printf("%d ", curent[i]+1);  printf("**\n**");  vprec = k; *//salveaza pozitia din drumul curent*  for (int i = k-1; i >= 0; i--)  {  prec[k-i-1] = curent[i];  }  k = 0; *//reseteaza indicele pentru un drum nou*  }  for(int i = varf-1; i >= 0; i--)  {  if(matrice[i][vcurent] > 0)  {  if(H2[vcurent] - H2[i] == matrice[i][vcurent])  {  PathFord(H2, i, indice);  indice = vcurent; *//cand ajunge la capatul unui drum se inntoarce aici si reia procesul din punctul dat*  }  }  }  }  void PathBellman(int H2[varf],int vcurent, int indice)  {  if(indice < varf+1)  {  int i;  int pozitie;  for(i = 0; i < vprec; i++)  {  if(indice == prec[i])  {  **break**;  }  }  for(int j = 0; j <= i; j++)  curent[k++] = prec[j];  indice = varf+1;  }  curent[k++] = vcurent;  if (vcurent == varf-1)  {  for (int i = 0; i < k; i++)  printf("%d ", curent[i]+1);  printf("**\n**");  vprec = k;  for (int i = 0; i < k; i++)  {  prec[i] = curent[i];  }  k = 0;  }  for(int i = vcurent; i <= varf-1; i++)  {  if(matrice[vcurent][i] > 0)  {  if(H2[vcurent] - H2[i] == matrice[vcurent][i])  {  PathBellman(H2, i, indice);  indice = vcurent;  }  }  }  }  void createMatrix()  {  matrice = allocMatrix();  input(matrice, varf);  }  int \*\*allocMatrix()  {  int \*\*temp = malloc(varf\*sizeof(int\*));  if(!temp)  {  printf("Alocare esuata");  exit(1);  }  for(int i = 0; i < varf; i++)  {  temp[i] = malloc(varf\*sizeof(int));  if(!temp[i])  {  printf("Alocare esuata");  exit(1);  }  }  return temp;  }  void input(int \*\*matrice, int varf)  {  printf("Dati matricea ponderata de adiacenta (infinit == 100)**\n**");  for(int i = 0; i < varf; i++)  {  for(int j = 0; j < varf; j++)  {  scanf("%d", &matrice[i][j]);  }  }  }  int cmpArr(int \*A, int \*B, int varf)  {  for(int i = 0; i < varf; i++)  if(A[i] != B[i]) return 1;  return 0;  }  void cpyArr(int \*A, int \*B, int varf)  {  for(int i = 0; i < varf; i++)  {  A[i] = B[i];  }  }  void freeArr()  {  for(int i = 0; i < varf; i++)  {  free(matrice[i]);  }  free(matrice);  } |



# CONCLUZII

În concluzie, algoritmii Ford și Bellman-Kalaba sunt două metode eficiente pentru a găsi cel mai scurt sau lung drum într-un graf ponderat.

Algoritmul lui Ford este simplu și intuitiv, folosind o strategie de relaxare iterativă pentru a obține distanțele minime/maxime de la un nod sursă la toate celelalte noduri din graf. Cu toate acestea, acest algoritm are o complexitate de timp mai mare decât algoritmul lui Bellman-Kalaba și poate fi mai puțin eficient pentru grafuri mari.

În comparație, algoritmul Bellman-Kalaba este mai eficient în cazul grafurilor mai mari, deoarece utilizează o strategie de relaxare globală, iar costurile sunt actualizate de mai multe ori până când se ajunge la soluție.

În general, alegerea între acești doi algoritmi depinde de dimensiunea grafului și de necesitățile aplicației. În timp ce algoritmul lui Ford este mai simplu și poate fi mai eficient pentru grafuri mai mici, algoritmul Bellman-Kalaba poate fi mai potrivit pentru grafuri mai mari.