**Tema 30**

**Ex1.**

|  |  |
| --- | --- |
| **Descrierea exercitiului** | Programul din această căsuță ar trebui să șteargă un nod dintr-un graf. În cod este reprezentat graful inițial din imaginea de mai jos. Completează acest cod astfel încît el să șteargă nodul 5 din graf. (Variabila k este nodul care trebuie șters). Codul trebuie să fie adăugat strict în main fără a atinge alte funcții. Imaginea dată reprezintă graful înainte de ștergere și după. |
| **Ce trebuie sa fie initial în caseta compilatorului?** | #include <stdio.h>  int n = 8; // nr de virfuri  int k = -1; // virful ce va fi sters  int a[20][20] = {  { 0, 0, 0, 0, 0, 0, 0, 0, 0 }, // nodul 0 nu exista  { 0, 0, 0, 0, 1, 0, 0, 0, 1 },  { 0, 0, 0, 1, 1, 0, 1, 0, 1 },  { 0, 0, 1, 0, 1, 0, 1, 0, 0 },  { 0, 1, 1, 1, 0, 1, 0, 1, 0 },  { 0, 0, 0, 0, 1, 0, 0, 0, 1 },  { 0, 0, 1, 1, 0, 0, 0, 0, 0 },  { 0, 0, 0, 0, 1, 0, 0, 0, 1 },  { 0, 1, 1, 0, 0, 1, 0, 1, 0 },  };  void afisare()  {  for(int i = 1; i <= n; i++)  {  for(int j = 1; j <= n; j++)  printf("%d ", a[i][j]);  printf("<br>");  }  }  int main()  {  /\*  aici va fi codul de stergere  a nodului  \*/    afisare();  return 0;  } |
| **Ce trebuie să facă elevul?** | **Să înlocuiască comentariul cu cod**  /\*  aici va fi codul de stergere  a nodului  \*/  Cod corect:  #include <stdio.h>  int n = 8; // nr de virfuri  int k = 5; // virful ce va fi sters  int a[20][20] = {  { 0, 0, 0, 0, 0, 0, 0, 0, 0 }, // nodul 0 nu exista  { 0, 0, 0, 0, 1, 0, 0, 0, 1 },  { 0, 0, 0, 1, 1, 0, 1, 0, 1 },  { 0, 0, 1, 0, 1, 0, 1, 0, 0 },  { 0, 1, 1, 1, 0, 1, 0, 1, 0 },  { 0, 0, 0, 0, 1, 0, 0, 0, 1 },  { 0, 0, 1, 1, 0, 0, 0, 0, 0 },  { 0, 0, 0, 0, 1, 0, 0, 0, 1 },  { 0, 1, 1, 0, 0, 1, 0, 1, 0 },  };  void afisare()  {  for(int i = 1; i <= n; i++)  {  for(int j = 1; j <= n; j++)  printf("%d ", a[i][j]);  printf("<br>");  }  }  int main()  {    // excludere linie k  for (int z = k; z < n; z++)  for(int j = 0; j <= n; j++)  a[z][j] = a[z + 1][j];  // excludere coloana k  for (int z = k; z < n; z++)  for(int i = 0; i < n; i++)  a[i][z] = a[i][z+1];  // reducere nr de virfuri  n--;  afisare();  return 0;  } |
| **Care este rezultatul corect afisat de compilator dupa ce elevul introduce codul correct?** | 0 0 0 1 0 0 1 0 0 1 1 1 0 1 0 1 0 1 1 0 0 1 1 1 0 0 1 0 0 1 1 0 0 0 0 0 0 0 1 0 0 1 1 1 0 0 0 1 0 |

**Ex2.**

|  |  |
| --- | --- |
| **Descrierea exercitiului** | Acest program va efectua parcurgerea în adîncime pentru graful din imaginea de mai jos. Modfică programul astfel încît funcția DFS să parcurgă corect în adîncime graful propus. Vîrful de start va fi considerat vîrful 8. Toate nodurile vizitate sunt afișate prin spațiu. (Tot codul trebuie adăugat în interiorul funcției DFS() ).  Exemplu de output (pentru virful 1 ca start):  **Parcurgerea: 6 3 5 7 8 2 4 1** |
| **Ce trebuie sa fie initial în caseta compilatorului?** | #include <stdio.h>  int n = 8; // nr de virfuri  int start = 8; // nod start  int vizitati[100] = { 0 }; // tot tabloul e 0  int a[20][20] = {  { 0, 0, 0, 0, 0, 0, 0, 0, 0 }, // nodul 0 nu exista  { 0, 0, 0, 0, 1, 0, 0, 0, 1 },  { 0, 0, 0, 1, 1, 0, 1, 0, 1 },  { 0, 0, 1, 0, 1, 0, 1, 0, 0 },  { 0, 1, 1, 1, 0, 1, 0, 1, 0 },  { 0, 0, 0, 0, 1, 0, 0, 0, 1 },  { 0, 0, 1, 1, 0, 0, 0, 0, 0 },  { 0, 0, 0, 0, 1, 0, 0, 0, 1 },  { 0, 1, 1, 0, 0, 1, 0, 1, 0 },  };  void DFS(int v)  {  // aici lipseste codul  }  int main()  {  printf("Parcurgerea: ");  DFS(start);  return 0;  } |
| Ce trebuie să facă elevul? | **Să adauge codul marcat cu galben**  Cod corect:  #include <stdio.h>  int n = 8; // nr de virfuri  int start = 8; // nod start  int vizitati[100] = { 0 }; // tot tabloul e 0  int a[20][20] = {  { 0, 0, 0, 0, 0, 0, 0, 0, 0 }, // nodul 0 nu exista  { 0, 0, 0, 0, 1, 0, 0, 0, 1 },  { 0, 0, 0, 1, 1, 0, 1, 0, 1 },  { 0, 0, 1, 0, 1, 0, 1, 0, 0 },  { 0, 1, 1, 1, 0, 1, 0, 1, 0 },  { 0, 0, 0, 0, 1, 0, 0, 0, 1 },  { 0, 0, 1, 1, 0, 0, 0, 0, 0 },  { 0, 0, 0, 0, 1, 0, 0, 0, 1 },  { 0, 1, 1, 0, 0, 1, 0, 1, 0 },  };  void DFS(int v)  {  int i;  vizitati[v] = 1;  for(i = 1; i <= n; i++)  if (vizitati[i] == 0 && a[v][i] != 0)  DFS(i);  printf("%d ", v);  }  int main()  {  printf("Parcurgerea: ");  DFS(start);  return 0;  } |
| **Care este rezultatul corect afisat de compilator dupa ce elevul introduce codul correct?** | Parcurgerea: 6 3 2 5 7 4 1 8 |

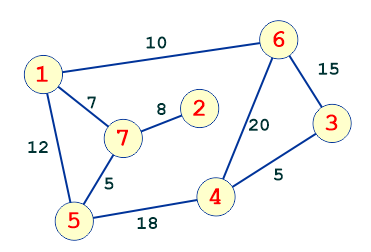
Sarcini practice:

1. Folosiți algoritmul de propagare a undei, pentru a implementa algoritmul de parcurgere a grafului în lățime.
2. Creați un program care va citi un graf neorientat de la tastatură, iar apoi la dorința utilizatorului va adăuga n arce noi de la tastatură.

**Tema 31**

**Ex1.**

Graful din imagine este un arbore ?



1. Da
2. Nu

Ex2.

Care sunt criteriile ca un graf să fie un arbore ?

1. Să fie conex, să aibă structură arborescentă.
2. Să aibă doar o structură arborescentă.
3. Arborele este un graf conex fără cicluri.

Ex3.

Care sunt pașii corecți a algoritmului Kruskal ?

1. Pas1: Sortăm toate muchiile în ordine crescătoare.   
   Pas2: Luăm muchia cu cea mai mică pondere și o adăugăm în arborele de cost minim, dacă după adăugare se formează cicluri - eliminăm muchia dată din lista de candidați, în caz contrar o păstrăm.  
   Pas3: Repetăm pasul 2 atît timp cît nu am inclus (n – 1) muchii. (Unde n este numărul de vîrfuri)
2. Pas1: Luăm muchia cu cea mai mare pondere și o adăugăm în arborele de cost minim, dacă după adăugare se formează cicluri - eliminăm muchia dată din lista de candidați, în caz contrar o păstrăm.

Pas2: Repetăm pasul 1 atît timp cît nu am inclus (n – 1) muchii. (Unde n este numărul de vîrfuri)

Pas3: Sortăm toate muchiile în ordine crescătoare.

1. Pas1: Sortăm toate muchiile în ordine descrescătoare.   
   Pas2: Luăm muchia cu cea mai mare pondere și o adăugăm în arborele de cost minim, dacă după adăugare se formează cicluri - eliminăm muchia dată din lista de candidați, în caz contrar o păstrăm.  
   Pas3: Repetăm pasul 2 atît timp cît nu am inclus (n – 1) muchii. (Unde n este numărul de vîrfuri)

|  |  |
| --- | --- |
| **Descrierea exercitiului** | În imagine este arătat un graf conex. Programul de mai jos este o variantă a algoritmului Kruskal. Această implementare permite să combine etapa de sortare cu etapa de selectare a muchiei candidat. Doar că acest program calculează arborele de cost maxim. Se cere de a modifica acest program astfel încît el să calculeze corect arborele de cost minim. Folosește comentariile ca indicii și încearcă să găsești greșeala. Succese ! |
| **Ce trebuie sa fie initial în caseta compilatorului?** | #include <stdio.h>  #include <limits.h> // Pentru INT\_MAX = + 2 miliarde...  int destinatii[100] = {0};  int virfuri = 7;  int g[100][100] = {  {0, 0, 0, 0, 0, 0, 0, 0},  {0, 0, 0, 0, 0, 12, 10, 7},  {0, 0, 0, 0, 0, 0, 0, 8},  {0, 0, 0, 0, 5, 0, 15, 0},  {0, 0, 0, 5, 0, 18, 20, 0},  {0, 12, 0, 0, 18, 0, 0, 5},  {0, 10, 0, 15, 20, 0, 0, 0},  {0, 7, 8, 0, 0, 5, 0, 0}  };  void init\_adiacenta()  {  int i, j;  for (i = 1; i <= virfuri; i++)  for (j = 1; j <= virfuri; j++)  if (g[i][j] == 0)  g[i][j] = INT\_MAX;  }  int cauta\_destinatie(int virf)  {  while (destinatii[virf] != virf)  virf = destinatii[virf];  return virf;  }  void reuniune(int v1, int v2)  {  int a = cauta\_destinatie(v1);  int b = cauta\_destinatie(v2);  destinatii[a] = b;  }  void kruskal()  {  int cost\_min = 0;  // din virful i eu pot ajunge in i  // din 1 -> 1 , 2 -> 2, 3 -> 3.  for (int i = 1; i <= virfuri; i++)  destinatii[i] = i;  int muchii\_incluse = 0;  while (muchii\_incluse < virfuri - 1)  {  int min;  min = 0;  // poate min = INT\_MAX ? de ce ?  int a = -1, b = -1;  // parcurgerea tabloului si cautarea muchiei cu cost minim  // FARA A FORMA CICLURI !!!  for (int i = 1; i <= virfuri; i++)  for (int j = 1; j <= virfuri; j++)  {  // Verificam daca nu formam ciclu  // si daca muchia este cea mai mica gasita pina acum  // dar oare corect este pus semnul (>) ? Totusi cautam minim  if (g[i][j] > min &&cauta\_destinatie(i) != cauta\_destinatie(j))  {  min = g[i][j];  a = i;  b = j;  }  }  //Conectam destinatiile  reuniune(a, b);  printf("(%d, %d, %d) ", a, b, min);  muchii\_incluse++;  cost\_min += min;  }  printf("Cost minim = %d", cost\_min);  }  int main()  {  // caile inexistente le marcam cu INT\_MAX  // poate rindul de mai jos trebuie decomentat ?  // de ce ?  //init\_adiacenta();  printf("rezultat: ");  kruskal();  return 0;  } |
| Ce trebuie să facă elevul? | **Sa corecteze greselile**  #include <stdio.h>  #include <limits.h> // Pentru INT\_MAX = + 2 miliarde...  int destinatii[100] = {0};  int virfuri = 7;  int g[100][100] = {  {0, 0, 0, 0, 0, 0, 0, 0},  {0, 0, 0, 0, 0, 12, 10, 7},  {0, 0, 0, 0, 0, 0, 0, 8},  {0, 0, 0, 0, 5, 0, 15, 0},  {0, 0, 0, 5, 0, 18, 20, 0},  {0, 12, 0, 0, 18, 0, 0, 5},  {0, 10, 0, 15, 20, 0, 0, 0},  {0, 7, 8, 0, 0, 5, 0, 0}  };  void init\_adiacenta()  {  int i, j;  for (i = 1; i <= virfuri; i++)  for (j = 1; j <= virfuri; j++)  if (g[i][j] == 0)  g[i][j] = INT\_MAX;  }  int cauta\_destinatie(int virf)  {  while (destinatii[virf] != virf)  virf = destinatii[virf];  return virf;  }  void reuniune(int v1, int v2)  {  int a = cauta\_destinatie(v1);  int b = cauta\_destinatie(v2);  destinatii[a] = b;  }  void kruskal()  {  int cost\_min = 0;  // din virful i eu pot ajunge in i  // din 1 -> 1 , 2 -> 2, 3 -> 3.  for (int i = 1; i <= virfuri; i++)  destinatii[i] = i;  int muchii\_incluse = 0;  while (muchii\_incluse < virfuri - 1)  {  int min;  min = INT\_MAX;  // poate min = INT\_MAX ? de ce ?  int a = -1, b = -1;  // parcurgerea tabloului si cautarea muchiei cu cost minim  // FARA A FORMA CICLURI !!!  for (int i = 1; i <= virfuri; i++)  for (int j = 1; j <= virfuri; j++)  {  // Verificam daca nu formam ciclu  // si daca muchia este cea mai mica gasita pina acum  if (g[i][j] < min &&cauta\_destinatie(i) != cauta\_destinatie(j))  {  min = g[i][j];  a = i;  b = j;  }  }  //Conectam destinatiile  reuniune(a, b);  printf("(%d, %d, %d) ", a, b, min);  muchii\_incluse++;  cost\_min += min;  }  printf("Cost minim = %d", cost\_min);  }  int main()  {  // caile inexistente le marcam cu INT\_MAX  // poate rindul de mai jos trebuie decomentat ?  // de ce ?  init\_adiacenta();  printf("rezultat: ");  kruskal();  return 0;  } |
| **Care este rezultatul corect afisat de compilator dupa ce elevul introduce codul correct?** | rezultat: (3, 4, 5) (5, 7, 5) (1, 7, 7) (2, 7, 8) (1, 6, 10) (3, 6, 15) Cost minim = 50 |

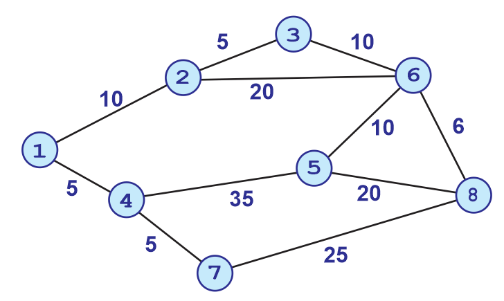
**Sarcini Practice:**

1. Caută informații adiționale despre algoritmul lui Prim. Acest algoritm rezolvă aceeași problemă ca și algoritmul Kruskal doar că are o logică diferită. Analizează acest algoritm și găsește diferențele dintre algoritmul Kruskal și Prim.

**Tema 32**

**Ex1.**

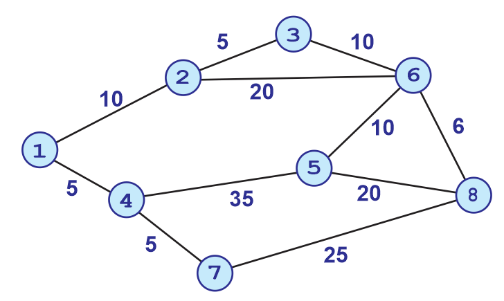
Avînd ca exemplu graful de mai jos, calculează care ar fi lungimea drumului minim din nodul 1 în nodul 5. Scrie în căsuță distanța. (Trebuie să fie o căsuță text dacă e posibil).



**Răspuns: 35**

**Ex2.**

Avînd ca exemplu graful de mai jos, calculează care ar fi lungimea drumului minim din nodul 1 în nodul 8. Scrie în căsuță distanța. (Trebuie să fie o căsuță text dacă e posibil).



**Răspuns: 31**

**Ex3.**

**Aici este prezentată o implementare pentru Algoritmul Dijkstra. Tema este destul de complicată pentru elevi. Din acest motiv am decis să le prezint o versiune de implementare algoritmului, aici elevii mai mult trebuie să analizeze codul și să vadă principiul de lucru.**

|  |  |
| --- | --- |
| **Descrierea exercitiului** | Aici este prezent o variantă alternativă a codului pentru căutarea drumului minim într-un graf. Mai concret spus, este implementarea algoritmului lui Dijsktra. Analizează principiul de lucrul a codului dat și modifică punctul de start astfel încît programul să găsească costul drumurilor minime din nodul 3. |
| **Ce trebuie sa fie initial în caseta compilatorului?** | #include <stdio.h>  #include <limits.h>  int vizitati[100] = {0}, cost[100][100] = {0};  int n = 7;  int start = 2;  int a[100][100] = {  {0, 0, 0, 0, 0, 0, 0, 0},  {0, 0, 0, 0, 0, 12, 10, 7},  {0, 0, 0, 0, 0, 0, 0, 8},  {0, 0, 0, 0, 5, 0, 15, 0},  {0, 0, 0, 5, 0, 18, 20, 0},  {0, 12, 0, 0, 18, 0, 0, 5},  {0, 10, 0, 15, 20, 0, 0, 0},  {0, 7, 8, 0, 0, 5, 0, 0}  };  int get\_min\_index(int rind)  {  int min\_index, min = INT\_MAX;  int i;  for (i = 1; i <= n; i++)  {  if (cost[rind][i] < min && vizitati[i] == 0)  {  min\_index = i;  min = cost[rind][i];  }  }  return min\_index;  }  void afisare\_distante()  {  int j;  printf("Distantele: ");  for (j = 1; j <= n; j++)  printf("%d->%d=%d. ", start, j, cost[n - 1][j]);  }  void copie\_vizitati(int rind)  {  int i;  for (i = 1; i <= n; i++)  cost[rind][i] = cost[rind - 1][i];  }  void dijkstra(int start)  {  int i, j, vc;  // initializam primul rind  vizitati[start] = 1;  for (i = 1; i <= n; i++)  {  if (a[start][i] == 0)  cost[0][i] = INT\_MAX;  else  cost[0][i] = a[start][i];  }  // drumul din start pina in start este 0  cost[0][start] = 0;  for (i = 1; i <= n - 1; i++)  {  vc = get\_min\_index(i - 1);  vizitati[vc] = 1;  copie\_vizitati(i);  for (j = 1; j <= n; j++)  {  // daca virful nu este vizitat si exista cale  // si distanta noua formata este mai mica ca cea veche  if (vizitati[j] == 0 && a[vc][j] != 0 &&  cost[i][vc] + a[vc][j] < cost[i - 1][j])  {  cost[i][j] = cost[i][vc] + a[vc][j];  }  }  }  }  int main()  {  dijkstra(start);  afisare\_distante();  return 0;  } |
| Ce trebuie să facă elevul? | **Este un program complex. Elevul trebuie să îl analizeze, să înțeleagă mecanismul de lucru iar apoi să modifice punctul de start.**  **Să înlocuiască**  int start = 2;  **cu**  int start = 3; |
| **Care este rezultatul corect afisat de compilator dupa ce elevul introduce codul correct?** | Distantele: 3->1=25. 3->2=36. 3->3=0. 3->4=5. 3->5=23. 3->6=15. 3->7=28. |