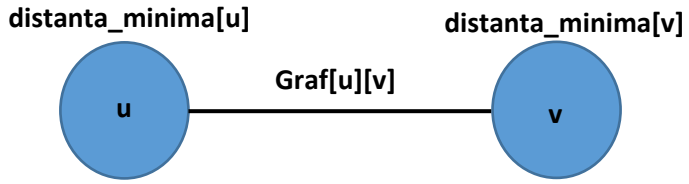
Algoritmul lui Dijkstra

Pseudocod

Pasi	**functia** Dijkstra(Graf, sursa)
1	Lista distantelor minime din Graf ale nodurilor
2	Lista nodurilor marcate ca vizitate
3	Lista nodurilor parinte
4	Pentru i <- 0 pana la | Numar_noduri_graf| - 1
5	lista_noduri_parinte [0] <- -1
6	distanta_minima[i]<-infinit
7	nod_vizitat[i] <- false
8	distanta_minima[sursa] <- 0
9	Pentru i <- 0 pana la | Numar_noduri_graf| - 2
10	u<-Nod_Distanta_Minima(distanta_minima, nod_vizitat)
11	nod_vizitat[u]<-true
12	Pentru v <- 0 pana la | Numar_noduri_graf| - 1
13	daca !nod_vizitat[v] si Graf[Numar_noduri_graf][Numar_noduri_graf]!=0 si distanta_minima[u] + Graf[u][v]< distanta_minima[v]
14	lista_noduri_parinte [v]<-u
15	distanta_minima[v]<- distanta_minima[u] + Graf[u][v]
16	Afisare_Solutie(distanta_minima, Numar_noduri_graf, lista_noduri_parinte)

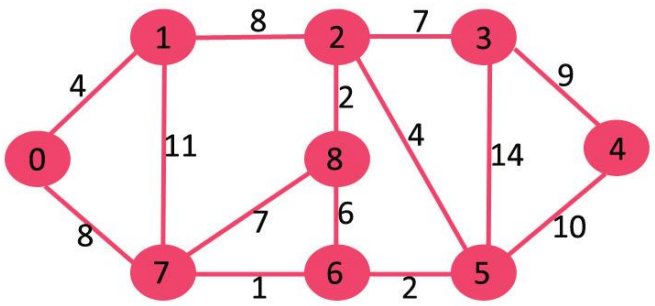


Pasi	**functia** Nod_Distanta_Minima(distanta_minima, nod_vizitat)
1	minim <- infinit
2	Pentru v<-0 pana la |Numar_noduri_graf |-1
3	daca nod_vizitat nu este vizitat si distanta_minima[v]<= minim
4	minim <- distanta_minima[v]
5	indice_minim<-v
6	**Returneaza** indice_minim

Pasi	**functia** Afisare_Cale(lista_noduri_parinte, nod_tinta)
1	daca lista_noduri_parinte[nod_tinta] = -1
2	Nu se returneaza nimic
3	Afisare_Cale(lista_noduri_parinte,lista_noduri_parinte[nod_tinta])
4	Afisare(nod_tinta)

Pasi	**functie** Afisare_Solutie(distanta_minima, Numar_noduri_graf, lista_noduri_parinte)
1	nod_sursa<-0
2	Pentru i<-1 pana la |Numar_noduri_graf| -1
3	Afisare(nod_sursa, i, distanta_minima[i], nod_sursa)
4	Afisare_Cale(lista_noduri_parinte, i)

Codul C++



Graf	0	1	2	3	4	5	6	7	8
0	0	4	0	0	0	0	0	8	0
1	4	0	8	0	0	0	0	11	0
2	0	8	0	7	0	4	0	0	2
3	0	0	7	0	9	14	0	0	0
4	0	0	0	9	0	10	0	0	0
5	0	0	4	0	10	0	2	0	0
6	0	0	0	14	0	2	0	1	6
7	8	11	0	0	0	0	1	0	7
8	0	0	2	0	0	0	6	7	0

```
#include <iostream>
#include <limits.h>
#define Numar_noduri_graf 9
#define infinit INT_MAX
using namespace std;
// returneaza indicele nodului cu distanta minima
int Nod_Distanta_Minima(int distanta_minima[], bool nod_vizitat[]){
   int minim=infinit, indice_minim;
   for(int i=0;i<Numar_noduri_graf;i++){
      if(nod_vizitat[i]==false && distanta_minima[i]<= minim){
         minim=distanta_minima[i];
         indice_minim=i;
      }
   }
   return indice_minim;
}
```

```
// afiseaza calea minima, fara nodul sursa
void Afisare_Cale(int lista_noduri_parinte[],int nod_tinta){
   if(lista_noduri_parinte[nod_tinta]==-1){
      return;
   }
   Afisare_Cale(lista_noduri_parinte,lista_noduri_parinte[nod_tinta]);
   cout<<nod_tinta<<" ";
}
// afiseaza nodurile, distanta minima si calea minima cu nod sursa
void Afisare_Solutie(int distanta_minima[], int NumarNoduriGraf,int lista_noduri_parinte[]){
   int nod_sursa=0;
   cout<<"Nod\t\tDistanta\tCale";
   for(int i=0;i<NumarNoduriGraf;i++){
      cout<<"\n"<<nod_sursa<<"->"<<i<<"\t\t"<<distanta_minima[i]<<"\t\t"<<nod_sursa<<" ";
      Afisare_Cale(lista_noduri_parinte,i);
   }
}
// Algoritmul Dijkstra care ne ofera calea cea mai scurta dintre un nod sursa si unul destinatie
void Dijkstra(int Graf[Numar_noduri_graf][Numar_noduri_graf], int nod_sursa){
   int distanta_minima[Numar_noduri_graf];
   bool nod_vizitat[Numar_noduri_graf];
   int lista_noduri_parinte[Numar_noduri_graf];
   for (int i=0;i<Numar_noduri_graf;i++){
      lista_noduri_parinte[0]=-1;
      distanta_minima[i]=infinit;
      nod_vizitat[i]=false;
   }
   distanta_minima[nod_sursa]=0;
   for(int i=0;i<Numar_noduri_graf-1;i++){
```

```
      int u = Nod_Distanta_Minima(distanta_minima, nod_vizitat);
      nod_vizitat[u]= true;
      for(int v=0;v<Numar_noduri_graf;v++){
         if(nod_vizitat[v]==false && Graf[u][v]!=0 &&
distanta_minima[u]+Graf[u][v]<distanta_minima[v]){
            lista_noduri_parinte[v]=u;
            distanta_minima[v]=distanta_minima[u]+Graf[u][v];
         }
      }
   }
   Afisare_Solutie(distanta_minima,Numar_noduri_graf,lista_noduri_parinte);
}

int main(){

   int nod_sursa=0;
   int Graf[Numar_noduri_graf][Numar_noduri_graf]={
                        {0, 4, 0, 0, 0, 0, 0, 8, 0},
                        {4, 0, 8, 0, 0, 0, 0, 11, 0},
                        {0, 8, 0, 7, 0, 4, 0, 0, 2},
                        {0, 0, 7, 0, 9, 14, 0, 0, 0},
                        {0, 0, 0, 9, 0, 10, 0, 0, 0},
                        {0, 0, 4, 0, 10, 0, 2, 0, 0},
                        {0, 0, 0, 14, 0, 2, 0, 1, 6},
                        {8, 11, 0, 0, 0, 0, 1, 0, 7},
                        {0, 0, 2, 0, 0, 0, 6, 7, 0}

};
   Dijkstra(Graf,nod_sursa);
   return 0;
}
```

Rezultat

```
Nod                 Distanta        Cale
0 -> 1                 4            0 1
0 -> 2                 12           0 1 2
0 -> 3                 19           0 1 2 3
0 -> 4                 21           0 7 6 5 4
0 -> 5                 11           0 7 6 5
0 -> 6                 9            0 7 6
0 -> 7                 8            0 7
0 -> 8                 14           0 1 2 8
```