Algoritmi na grafovima i Python

Balen, Chiara

Sveučilište u Rijeci

Fakultet za matematiku

Sveučilišni prijediplomski studij Matematika

Chiara Balen

Algoritmi na grafovima i Python

Završni rad

Rijeka, lipanj 2024.

Sveučilište u Rijeci

Fakultet za matematiku

Sveučilišni prijediplomski studij Matematika

Chiara Balen

Algoritmi na grafovima i Python

Mentor: doc. dr. sc. Nina Mostarac

Završni rad

Rijeka, lipanj 2024.

Sažetak

U ovom radu bavit ćemo se algoritmima na grafovima te njihovom implementacijom u Pythonu. U uvodnom dijelu upoznat ćemo se s osnovama teorije grafova, između ostalog s definicijom usmjerenih i neusmjerenih grafova. Zatim će se u glavnom dijelu objasniti kako se usmjereni i neusmjereni grafovi mogu reprezentirati u Python modulu scipy.sparse.csgraph. Nakon toga će se navesti razni algoritmi na grafovima koji će se povezati s odgovarajućim naredbama za njihovu provedbu u spomenutom modulu. Spomenut će se i kako pomoću Python modula Pydot možemo formirati grafički prikaz, odnosno vizualizaciju grafova iz opisanih primjera.

Ključne riječi: graf, usmjereni i neusmjereni graf, algoritmi na grafovima

Sadržaj

Poglavlje 1

Uvod

Teorija grafova razvila se u 18.st. Grafovi su jedna od najosnovnijih matematičkih struktura koje se koriste za modeliranje problema u svakodnevnom životu tako da složeni problem modeliramo jednostavnijim. Osim u matematici, koriste se u računarstvu, arhitekturi, fizici, biologiji, ekonomiji, računalnoj znanosti. Prvim problemom u teoriji grafova smatra se problem sedam mostova, koji je postavio švicarski matematičar Leonhard Euler. Problem opisuje sedam mostova u Königsbergu koji povezuju kopno te se postavlja pitanje je li moguće pronaći rutu kojom ćemo proći kroz grad i proći svaki most samo jedanput.

Osim tog problema, mnoge se pojave modeliraju grafovima, koji se sastoje od točaka i njihovih spojnica (bridova). Na primjer, točke, tj. vrhovi, mogu predstavljati ljude neke skupine, dok spojnice, tj. bridovi, parove prijatelja. Graf može predstavljati električnu mrežu čiji su vrhovi električke komponente, a spojnice električne veze (cestovne, željezničke, zrakoplovne veze). U računarstvu se grafovi koriste kao dijagram toka nekog algoritma, koji se prikazuje grafom čiji su vrhovi naredbe (instrukcije), a lukovi iz jedne u drugu naredbu su usmjereni bridovi. Jednako tako, grafovima se prezentiraju i razne kompjuterske strukture podataka, umrežavanje i paralelizam računala i njihov sekvencijalni rad, evolucijska ili porodična stabla u biologiji itd.

1.1 Osnovni pojmovi

Definicija 1.1. *Graf G je uređena trojka $G = (V(G), E(G), \psi_G)$ gdje je:*

1. *$V(G) \neq \emptyset$ skup vrhova od G,*

2. *$E(G)$ skup bridova od G, disjunktan s $V(G)$,*

3. *ψ_G funkcija incidencije koja svakom bridu od G pridružuje neuređeni par, ne nužno različitih, vrhova od G.*

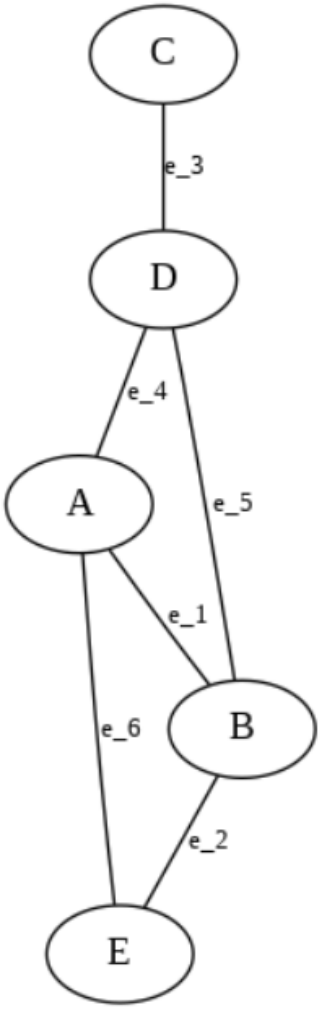
Definicija 1.2. *Ako je $\psi_G(e) = (u, v)$, za $e \in E(G)$,$u, v \in V(G)$, tada kažemo da je brid e incidentan s vrhovima u i v, tj. da ih brid e spaja ili povezuje. U tom slučaju kažemo i da su vrhovi u i v susjedni te da su oni krajevi brida e. Kraće pišemo $e = uv$.*

Primjer 1.3. *Graf $G = (V(G), E(G), \psi_G)$ zadan je skupom vrhova $V(G) = \{A, B, C, D, E\}$ i skupom bridova $E(G) = \{e_1, e_2, e_3, e_4, e_5, e_6\}$ te funkcijom incidencije $\psi = \psi_G$ za koju je: $\psi(e_1) = \{A, B\}$, $\psi(e_2) = \{B, E\}$, $\psi(e_3) = \{C, D\}$, $\psi(e_4) = \{D, A\}$, $\psi(e_5) = \{D, B\}$, $\psi(e_6) = \{A, E\}$.*

Vizualni prikaz jednostavnog neusmjerenog grafa G iz primjera 1.3 možemo dobiti u Pythonu pomoću modula Pydot, korištenjem sljedećeg koda.

```
import pydot
#Stvaranje instance neusmjerenog grafa (graph)
graph = pydot.Dot(graph_type='graph', strict=True)
#Dodavanje vrhova u graf
nodes = ["A", "B", "C", "D", "E"]
for node in nodes:
    graph.add_node(pydot.Node(node))
#Dodavanje vrhova i bridova u graf
graph.add_edge(pydot.Edge("A", "B", label="e_1"))
graph.add_edge(pydot.Edge("B", "E", label="e_2"))
graph.add_edge(pydot.Edge("C", "D", label="e_3"))
graph.add_edge(pydot.Edge("D", "A", label="e_4"))
graph.add_edge(pydot.Edge("D", "B", label="e_5"))
graph.add_edge(pydot.Edge("A", "E", label="e_6"))
#Spremanje slike grafa u png datoteku
graph.write_ png("undirected_graph_pydot1.png")
```

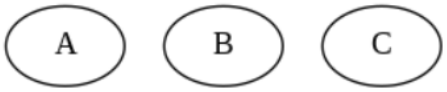
Dobiveni prikaz grafa G iz primjera 1.3. dan je na slici 1.1.



Slika 1.1: Graf G iz primjera 1.3.

Napomena 1.4. *Kako je skup vrhova V po definiciji neprazan skup, onda graf sadrži najmanje jedan vrh. Isto tako, graf ne mora sadržavati bridove, tj. moguće je da bude $E(G) = \emptyset$.*

Graf koji ne sadrži bridove nazivamo prazan graf. Za graf koji sadrži samo jedan vrh kažemo da je trivijalan graf.

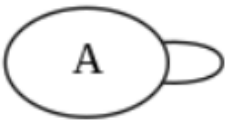


Slika 1.2: Prazan graf G

Slika 1.3: Trivijalan graf G

Funkcija incidencije ψ ne mora biti injekcija pa je moguće da više bridova bude incidentno s istim parom vrhova. Takvi se bridovi nazivaju višestruki bridovi.

Nadalje, za $e \in E(G)$ i $v \in V(G)$ moguće je da $\psi(e) = (v, v)$. Brid koji spaja vrh sa samim sobom nazivamo petlja.



Slika 1.4: Petlja

Ako graf nema petlji ni višestrukih bridova, kažemo da je to jednostavan graf.

Definicija 1.5. *Red grafa je broj elemenata skupa vrhova grafa.*

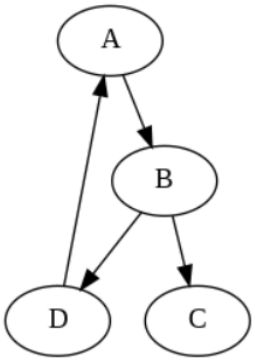
Definicija 1.6. *Veličina grafa je broj bridova danog grafa.*

Definicija 1.7. *Stupanj vrha $v \in V(G)$, u oznaci $d_G(v)$, je broj bridova incidentnih s v, pri čemu svaku petlju u v računamo kao dva brida incidentna s v.*

Napomena 1.8. *Kod jednostavnih grafova, stupanj vrha v možemo promatrati kao kardinalni broj skupa svih vrhova susjednih vrhu v u danom grafu.*

Definicija 1.9. *Vrh stupnja 1, tj. vrh v za koji je $d_G(v) = 1$, nazivamo list, a vrh stupnja 0, tj. vrh v za koji je $d_G(v) = 0$, nazivamo izolirani vrh.*

Definicija 1.10. *Usmjereni graf ili digraf je uređena trojka $G = (V(G), E(G), \psi_G)$, gdje je $V \neq \emptyset$ skup vrhova, $E = E(G)$, gdje je $E \cap V = \emptyset$, skup bridova, a $\psi_G : E \to V \times V$ funkcija incidencije koja svakom bridu $e \in E(G)$ pridružuje uređeni par ne nužno različitih vrhova (u, v), $u, v \in V(G)$. Vrh u tada zovemo početni vrh, a vrh v krajnji vrh brida e, dok brid e nazivamo lukom od vrha u do vrha v.*



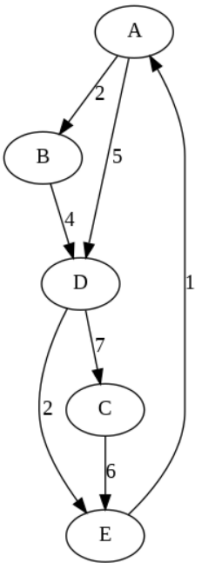
Slika 1.5: Usmjereni graf

Napomena 1.11. *Kažemo da je usmjereni graf G jednostavan ukoliko nema petlji ni paralelnih bridova, pri čemu je kod usmjerenih grafova petlja luk iz vrha u isti taj vrh, dok su paralelni bridovi 2 ili više lukova sa zajedničkim početnim vrhom i zajedničkim krajnjim vrhom.*

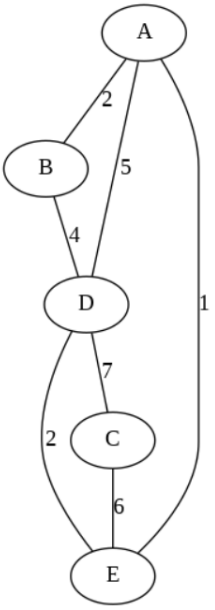
Napomena 1.12. *Svaki neusmjereni graf možemo promatrati kao usmjereni, pri čemu brid u G incidentan s vrhovima v_i i v_j promatramo kao par bridova od v_i do v_j te v_j do v_i.*

Definicija 1.13. *Neka je $G = (V, E)$ graf i $w(e) : E(G) \to \mathbb{R}$ funkcija koja svakom bridu grafa G pridružuje realan broj. Takvu funkciju zovemo težinska funckija, a graf za koji je definirana takva funkcija težinski graf. Realan broj $w(e)$ pridružen bridu $e \in E(G)$ zovemo težina brida e. Težinski graf može biti usmjeren ili neusmjeren, ovisno o tome jesu li bridovi usmjereni ili ne.*

Na slici 1.6. dan je primjer usmjerenog težinskog grafa, dok je na slici 1.7. primjer neusmjerenog težinskog grafa.



Slika 1.6: Usmjereni težinski graf G



Slika 1.7: Neusmjereni težinski graf G

Napomena 1.14. *Matrica susjedstva daje informacije o tome koji su vrhovi povezani bridovima i može se koristiti za razne operacije i analize nad grafovima.*

Definicija 1.15. *Neka je* $V(G) = \{v_1, v_2, ..., v_n\}$. *Matrica susjedstva neusmjerenog grafa* $G = (V(G), E(G), \psi_G)$ *je kvadratna matrica* $A = [a_{ij}]$ *reda* $n = |V(G)|$ *takva da je* a_{ij} *jednak broju bridova koji spajaju vrhove* v_i *i* v_j, *za* $i, j \in \{1, ..., n\}$.

Napomena 1.16. *Graf je jednoznačno određen svojom matricom susjedstva te se može zadati pomoću nje.*

Definicija 1.17. *Neka je* $V(G) = \{v_1, v_2, ..., v_n\}$. *Matrica susjedstva usmjerenog grafa* $G = (V(G), E(G), \psi_G)$ *je kvadratna matrica* $A = [a_{ij}]$ *reda* $n = |V(G)|$ *takva da je* a_{ij} *jednak broju lukova iz vrha* v_i *u* v_j.

Definicija 1.18. *Simetrična matrica je kvadratna matrica* $A = [A_{ij}]$ *dimenzije* $n \times n$ *koja zadovoljava uvjet* $A_{ij} = A_{ji}$, *za sve* $i, j \in \{1, 2, ..., n\}$.

Primjer 1.19. *Primjer simetrične matrice:*

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 4 & 5 \\ 3 & 5 & 6 \end{pmatrix}$$

.

Napomena 1.20. *Za neusmjereni graf* G, *matrica susjedstva je simetrična, dok za usmjereni graf to ne mora biti slučaj.*

Napomena 1.21. *Za jednostavan graf* G, *matrica susjedstva* A *grafa* G *je matrica s elementima iz skupa* $\{0, 1\}$, *budući da u tom slučaju dva vrha mogu biti povezana najviše jednim bridom.*

Primjer 1.22. *Pogledajmo matricu susjedstva usmjerenog grafa sa slike 1.5.:*

$$A = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

Matrica susjedstva neusmjerenog grafa sa slike 1.1. :

$$A' = \begin{pmatrix} 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \end{pmatrix}$$

Napomena 1.23. *Uočimo razliku između matrice susjedstva usmjerenog i neusmjerenog grafa. Matrica susjedstva neusmjerenog grafa odražava prisutnost bridova između vrhova bez obzira na smjer, dok matrica susjedstva usmjerenog grafa odražava prisutnost bridova koji imaju određeni smjer od jednog vrha do drugog vrha.*

Napomena 1.24. *Ukoliko je graf G jednostavan težinski graf s težinskom funkcijom w, tada ga možemo opisati pomoću težinske matrice susjedstva $A = [a_{ij}]$ u kojoj je $a_{ij} = w(e)$ ukoliko postoji brid $e \in E(G)$ incidentan s vrhovima v_i i v_j, a $a_{ij} = \infty$ ukoliko vrhovi v_i i v_j nisu susjedni. Analogno se definira težinska matrica susjedstva $A = [a_{ij}]$ jednostavnog usmjerenog težinskog grafa G. U tom slučaju je $a_{ij} = w(e)$ ukoliko postoji luk $e \in E(G)$ od vrha v_i do vrha v_j te $a_{ij} = \infty$ inače.*

Poglavlje 2

Algoritmi na grafovima

U ovom poglavlju govorit ćemo o algoritmima na grafovima te ćemo opisati Python modul **scipy.sparse.csgraph** i neke naredbe iz tog modula koje služe za pronalaženje rješenja pojedinih problema vezanih uz grafove. Grafovi se u Python modulu **scipy.sparse.csgraph** zadaju odabranom matričnom reprezentacijom odgovarajuće matrice susjedstva grafa.
U Pythonu postoji više načina za reprezentaciju matrice. Klasični način je prikazati matricu kao dvodimenzionalnu listu u kojoj je eksplicitno naveden svaki njezin element pomoću naredbe array iz modula Numpy, što ćemo nazivati gusti matrični prikaz. Drugi način je koristiti rijetki (sparse) matrični prikaz, u kojemu se navode samo ne-nul elementi matrice te odgovarajuće pozicije. Rijetki matrični prikaz koristan je za rijetke matrice (one u kojima je više od polovice elemenata jednako nuli), ali i općenito za matrice s puno nul-elemenata.
Neka je dana matrica:

$$A = \begin{pmatrix} 1 & 0 & 2 & 0 & 0 \\ 0 & 0 & 3 & 0 & 4 \end{pmatrix}.$$

Gusti matrični prikaz matrice A dobivamo pomoću sljedećih naredbi.

```
import numpy as np
A=np.array([[1,0,2,0,0],[0,0,3,0,4]])
```

Rijetka matrična reprezentacija, koja štedi memoriju zapisivanjem samo ne-nul (odnosno nepraznih) elemenata, dobiva se na sljedeći način.

```
from scipy.sparse import csr_matrix
#zadavanje niza podataka
podaci=np.array([1,2,3,4])
#zadavanje indeksa redaka i stupaca redom za svaki podatak
retci=np.array([0,0,1,1])
stupci=np.array([0,2,2,4])

#stvaranje rijetkog matričnog prikaza
rijetki_prikaz=csr_matrix((podaci,(retci,stupci)),shape =(2,5))
##oblik (shape) određuje tip matrice

##prelazak na gusti matrični prikaz
gusti_prikaz= rijetki_prikaz.toarray()

print("Rijetki matrični prikaz:\n", rijetki_prikaz)
print("Gusti matrični prikaz;\n", gusti_prikaz)
```

```
Rijetki matrični prikaz:
  (0, 0)        1
  (0, 2)        2
  (1, 2)        3
  (1, 4)        4
Gusti matrični prikaz;
 [[1 0 2 0 0]
 [0 0 3 0 4]]
```

Slika 2.1: Prikaz rijetke i guste matrične reprezentacije

2.1 Python modul scipy.sparse.csgraph

Python modul scipy.sparse.csgraph pruža brze algoritme za grafove koji se temelje na njihovoj reprezentaciji pomoću rijetkog matričnog prikaza.

U ovom modulu grafovi se spremaju u matričnom obliku. Graf sa n vrhova reprezentira se pomoću kvadratne matrice A reda n koja predstavlja težinsku matricu susjedstva grafa

G. Ako postoji luk od $i-$tog do j-tog vrha, tada je $A[i,j] = w$, gdje je w težina danog luka. Ako ne postoji luk iz i-tog u j-ti vrh, tada vrijednost $A[i,j]$ ovisi o izabranoj vrsti matrične reprezentacije. Tako razlikujemo gustu matričnu reprezentaciju od rijetke matrične reprezentacije.

Napomena 2.1. *Rijetka matrična reprezentacija koristi se kada graf ima malo bridova u odnosu na broj vrhova, radi uštede memorije.*

Kod guste matrične reprezentacije, ako ne postoji luk iz i-tog u j-ti vrh, tada postavljamo $A[i,j]$ na 0 (ako bridovi težine 0 nisu dopustivi), beskonačno ili NaN.
Kod rijetke matrične reprezentacije, nepostojanje luka iz i-tog u j-ti vrh očituje se izostavljanjem unosa odgovarajućeg elementa u rijetku matricu A. Ova vrsta rijetke reprezentacije također dopušta bridove težine nula.

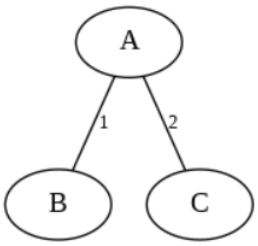
Napomena 2.2. *Kada radimo s grafom koji ima mnogo vrhova, ali malo bridova korisno je napraviti prelazak iz guste matrične reprezentacije u rijetku matričnu reprezentaciju. Za izvršavanje prelaska, u Pythonu koristimo naredbu csr_matrix.*
Kod u Pythonu:

```
import numpy as np
from scipy.sparse import csr_matrix

# Gusti matrični prikaz
dense_matrix = np.array([
    [0, 1, 0, 0],
    [1, 0, 1, 0],
    [0, 1, 0, 1],
    [0, 0, 1, 0]
]).

# Pretvorba u rijetki matrični prikaz
sparse_matrix = csr_matrix(dense_matrix)
```

Primjer 2.3. *Neka je G neusmjereni težinski graf sa skupom vrhova $V(G) = \{v_1, v_2, v_3\}$ te skupom bridova $E(G) = \{e_1, e_2\}$, pri čemu je $e_1 = v_1v_2$, $e_2 = v_1v_3$, $w(e_1) = 1$ te $w(e_2) = 2$.*

Kod guste matrične reprezentacije, dani graf G možemo reprezentirati simetričnom matricom čiji su svi elementi eksplicitno navedeni, koja se u Pythonu definira pomoću naredbe np.array(). Kod koji prikazuje gusti matrični prikaz danog grafa G:

```
import numpy as np
G_dense = np.array([[0, 1, 2],
                    [1, 0, 0],
                    [2, 0, 0]])
```



Slika 2.2: Neusmjereni težinski graf iz primjera 2.3.

2.2 Problem najkraćeg puta i Dijkstrin algoritam

U raznim primjenama, kako u matematici tako i u drugim znanostima, često su potrebne i dodatne strukture na grafovima. Jedna od najprirodnijih je da svakom bridu $e \in E(G)$ pridružimo njegovu *težinu* $w(e) \in \mathbb{R}$, tj. realan broj veći od 0.

Tako se problem pronalaska najkraćeg puta svodi na to da se u težinskom grafu nađe podgraf određenog tipa s najmanjom težinom. Pronaći najkraći put između dva vrha znači pronaći put s najmanjom težinom. Ovaj problem je od velike važnosti u mnogim područjima kao što su putovanja, telekomunikacija, određivanje rute i slično.

Dijkstrin algoritam, koji je dobio naziv po nizozemskom znanstveniku Edsger Wybe Dijkstra, koji ga je prvi put objavio 1959. godine, je jedan od najpoznatijih algoritama za rješavanje problema najkraćeg puta u grafu. Opis Dijkstrinog algoritma:
Neka je zadan graf G sa skupom vrhova $V(G)$ te skupom bridova $E(G)$, gdje svaki brid $e \in E(G)$ ima ne-negativnu težinu $w(e)$. Neka je s početni vrh iz kojeg želimo pronaći

najkraće puteve do svih ostalih vrhova u grafu G. Neka je $dist(v)$ funkcija udaljenosti koja predstavlja trenutnu udaljenost od vrha s do vrha v. Neka je Q niz koji sadrži sve vrhove grafa $V(G)$ sortirane prema trenutnim udaljenostima od vrha s. Za početak postavimo $dist(s) = 0$ i $dist(v) = \infty$ za sve ostale vrhove $v \neq s$. Dodajemo sve vrhove u niz Q. Dok god niz Q nije prazan, ponavljamo sljedeće korake. Izaberemo vrh u s najmanjom trenutnom udaljenosti $dist(u)$ iz niza Q. Zatim, uklonimo vrh u. Za svaki susjedni vrh v vrha u, tj. za svaki vrh povezan bridom s u, ažuriramo udaljenost $dist(v)$ na $min(dist(v), dist(u) + w(e))$, gdje je $e = uv$. Nakon što završe sve iteracije, $dist(v)$ će vraćati udaljenosti od početnog vrha s do svih ostalih vrhova v u grafu G.

Naredba u scipy.sparse.csgraph-u koja automatski izvršava Dijkstrin algoritam je:
$scipy.sparse.csgraph.dijkstra(csgraph, directed = True, indices = None,$
$return\_predecessors = False, unweighted = False, limit = np.inf, min\_only = False)$

Objasnimo sada što pojedini parametar u ovoj naredbi za gotov Dijsktrin algoritam predstavlja.

Parametar **csgraph**: matrica susjedstva ili udaljenosti koja predstavlja graf. Može biti gusta ($np.array()$) ili rijetka ($scipy.sparse()$) matrica.

Parametar **directed**: logička vrijednost koja označava da li je graf usmjeren (True) ili ne (False). Ako je usmjeren, Dijsktrin algoritam uzima u obzir smjerove bridova.

Parametar **indices**: Indeksi početnih vrhova za koje se izvodi algoritam. Ako je 'None', algoritam će se izvršiti za sve vrhove.

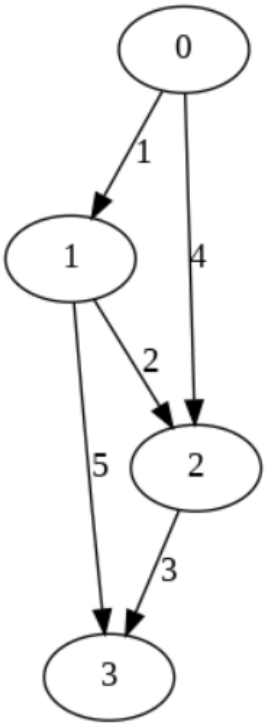
Parametar **return_predecessors**: logička vrijednost koja označava da li će funkcija vratiti informacije o prethodnicima za svaki vrh. Ako je True, funkcija će vratiti matricu prethodnika koja sadrži indekse prethodnika za svaki vrh.

Parametar **unweigthed**: logička vrijednost koja označava jesu li svi bridovi u grafu jedinične težine (True) ili ne (False). Ako su svi bridovi jedinične težine, algoritam može biti optimiziran za brži rad.

Parametar **limit**: broj koji označava maksimalnu vrijednost udaljenosti koja će biti uzeta u obzir tijekom izvođenja algoritma. Ako je neka udaljenost veća od ovog ograničenja, bit će ignorirana.

Parametar **min_only**: logička vrijednost koja označava da li će algoritam pronaći samo najmanje udaljenosti do svih drugih vrhova (True) ili će pronaći i prethodnike

(False).



Slika 2.3: Dijkstrin algoritam na grafu

Primjer 2.4. *U ovom primjeru, pogledat ćemo kod za izvršavanje Dijkstrinog algoritma za graf sa slike 2.3.*

```
import numpy as np
from scipy.sparse import csr_matrix
from scipy.sparse.csgraph import dijkstra

# Definirajmo težinsku matricu susjedstva za graf
graph = np.array([
    [0, 1, 4, 0],
    [0, 0, 2, 5],
    [0, 0, 0, 3],
    [0, 0, 0, 0]
])

# Pretvorba u rijetki matrični prikaz
```

```
sparse_graph = csr_matrix(graph)

# Izvršavanje Dijkstrinog algoritma
distances, predecessors = dijkstra(sparse_graph, directed=True,\
return_predecessors=True)

print("Udaljenosti od vrha 0 do ostalih vrhova:")
print(distances[0])  # Ispisuje udaljenosti od vrha 0 do svih ostalih vrhova
print("\nPrethodnici na najkraćim putevima:")
print(predecessors)  # Ispisuje matricu prethodnika
```

Nakon izvršavanja koda dobivaju se sljedeći izlazni podaci:
Udaljenosti od vrha 0 do ostalih vrhova:

$$\begin{pmatrix} 0 & 1 & 3 & 6 \end{pmatrix}.$$

Prethodnici na najkraćim putevima:

$$\begin{pmatrix} -1 & 0 & 1 & 1 \\ -1 & -1 & 1 & 1 \\ -1 & -1 & -1 & 2 \\ -1 & -1 & -1 & -1 \end{pmatrix}.$$

Napomena 2.5. *Iako je Dijkstrin algoritam izuzetno koristan i često primjenjiv algoritam za pronalaženje najkraćih puteva u grafu, postoji nekoliko nedostataka koje treba uzeti u obzir. Kao prvo, Dijkstrin algoritam radi samo s pozitivnim težinama bridova, tj. radi samo s grafom koji nema negativne težine bridova. U suprotnom, ako graf sadrži negativne težine, algoritam ne pronalazi ispravno najkraće puteve te može doći do beskonačne petlje kada se radi o grafu s ciklusima negativne težine.*

Kako bi takve nedostatke izbjegnuli, potrebno je koristiti alternativne algoritme poput Floyd-Warshall algoritma i Bellman-Ford algoritma koje ću opisati u nastavku.

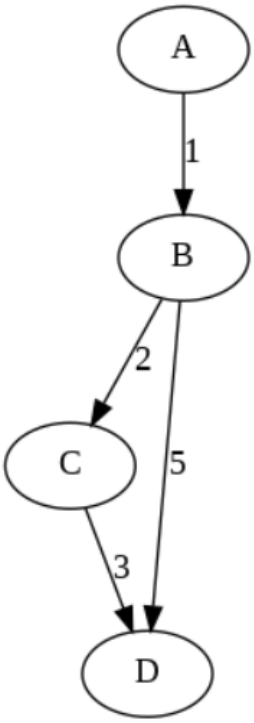
2.3 Floyd-Warshall algoritam

Floyd-Warshall algoritam je algoritam za pronalaženje najkraćih puteva između svih parova vrhova u težinskom grafu, čak i ako graf sadrži negativne težine. Objavio ga je Robert Floyd 1962. godine. To je u biti isti algoritam kao i onaj kojeg su prethodno objavili Bernard Roy 1959. i Stephen Warshall 1962. godine za pronalaženje tranzitivnog zatvaranja grafa.

Za razliku od Dijkstrinog algoritma koji pronalazi najkraći put od početnog vrha do svih ostalih vrhova, Floyd-Warshall algoritam pronalazi najmanje udaljenosti između svih parova vrhova u grafu. Stoga je algoritam poznat i pod nazivom "All-Pairs Shortest Paths", tj. APSP algoritam. Opis Floyd-Warshall algoritma:
Neka je zadan graf G sa skupom vrhova $V(G)$ tako da vrijedi $|V(G)| = n$ te skupom bridova $E(G)$, gdje svaki brid $e \in E(G)$ ima svoju težinu $w(e)$. Neka je d_{ij} težina najkraćeg puta između vrhova i i j. Stvaramo matricu udaljenosti D tipa $n \times n$, gdje je $D[i][j]$ početno postavljen na težinu brida između vrhova i i j. Udaljenost $D[i][j] = \infty$, ako brid između vrhova ne postoji, a ako je specijalno $i = j$ onda je $D[i][j] = 0$, za svaki vrh i. Algoritam se izvodi u n iteracija. Za svaki vrh $k \in \{1, ..., n\}$, algoritam provjerava svaki par vrhova i i j te ažurira udaljenost $D[i][j]$ po sljedećoj formuli:
$D[i][j] = min(D[i][j], D[i][k]+D[k][j])$. Ova formula znači da se udaljenost između vrhova i i j ažurira tako da se uspoređuje trenutna udaljenost $D[i][j]$ s udaljenostima koje se mogu postići prolaskom kroz vrh k. Nakon svih n iteracija, matrica udaljenosti D sadržavat će udaljenosti između svih parova vrhova.

Naredba u scipy.sparse.csgraph-u koja automatski izvršava Floyd-Warshall algoritam je:
$scipy.sparse.csgraph.floyd\_warshall(csgraph, directed = True, return\_predecessors = False, unweighted = False, overwrite = False)$

Parametri csgraph, directed, return_predecessors i unweigthed definirani su analogno kao kod Dijkstrinog algoritma.
Parametar **overwrite**: logička vrijednost koja označava je li rezultat udaljenosti pohranjen u ulaznu matricu grafa. Ako je postavljena na True, rezultat će se pohraniti u ulaznu matricu grafa. Ako je postavljen na False, rezultat će biti vraćen kao zasebna matrica.



Slika 2.4: Floyd-Warshall algoritam na grafu

Primjer 2.6. *U ovom primjeru, pogledat ćemo kod za izvršavanje Floyd-Warshall algoritma za graf sa slike 2.4.:*

```
import numpy as np
from scipy.sparse.csgraph import floyd_warshall

# Težinska matrica grafa
graph_weights = np.array([
    [0, 1, np.inf, np.inf],
    [np.inf, 0, 2, 5],
    [np.inf, np.inf, 0, 3],
    [np.inf, np.inf, np.inf, 0]
])
# Izračunavanje najkraćih puteva pomoću Floyd-Warshall algoritma
shortest_paths = floyd_warshall(csgraph=graph_weights, directed=True,\
return_predecessors=True, unweighted=False, overwrite=False)

print("Matrica najkraćih puteva:")
```

```
print(shortest_paths[0])
```

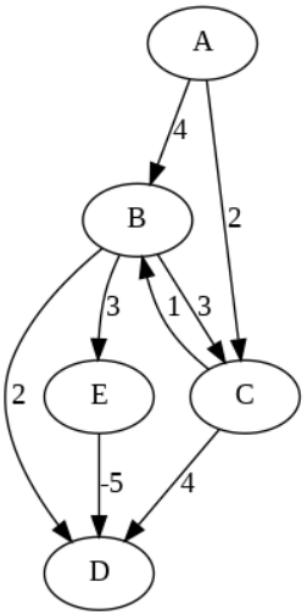
Dobivaju se sljedeći izlazni podaci:
Matrica najkraćih puteva A:

$$A = \begin{pmatrix} 0 & 1 & 3 & 6 \\ inf & 0 & 2 & 5 \\ inf & inf & 0 & 3 \\ inf & inf & inf & 0 \end{pmatrix}.$$

2.4 Bellman-Ford algoritam

Još jedan algoritam na grafovima je Bellman-Ford algoritam koji služi za pronalaženje najkraćeg puta na težinskom grafu, čak i ako graf sadrži negativne težine bridova, osim ako graf ne sadrži ciklus s negativnom težinom. Algoritam je prvi izveo Alfonso Shimbel, 1955. godine, ali je umjesto toga nazvan po Richard Bellmanu i Lester Fordu, koji su ga objavili 1958. godine. Opis Bellman-Ford algoritma:
Pretpostavimo da imamo težinski graf G sa skupom vrhova $V(G)$ i skupom bridova $E(G)$ te funkcijom težine bridova $w(u, v)$ koja daje težinu brida između vrhova u i v. Neka je s početni vrh za koji želimo pronaći najkraće puteve do svih ostalih vrhova u grafu G. U prvom koraku, postavljamo početnu udaljenost od početnog vrha s do svih preostalih vrhova na ∞, tj. $d[v] = \infty, \forall v \in V(G) \setminus \{s\}$, osim udaljenosti od s do samog sebe koju postavljamo na vrijednost 0, tj. $d[s] = 0$.
Sljedeći postupak ponavljamo $|V(G)| - 1$ puta. Za svaki brid $e = (u, v) \in E(G)$ provjeravamo mogućnost ažuriranja udaljenosti do vrha v kroz vrh u. Ako je trenutna udaljenost od početnog vrha s do vrha u plus težina brida (u, v) manja od trenutne udaljenosti do vrha v, ažuriramo udaljenost $d[v]$. Ako $d[u] + w(u, v) < d[v]$ onda $d[v] = d[u] + w(u, v)$. Nakon $|V(G)| - 1$ iteracija, provjeravamo postoji li negativni ciklus u grafu. Negativni ciklus je bilo koji ciklus čiji je zbroj težina negativan. Ako pronađemo takav ciklus, to znači da algoritam ne može pronaći najkraće puteve jer možemo kružiti tim ciklusom kako bismo smanjili ukupnu udaljenost. Nakon izvođenja algoritma, udaljenost $d[v]$ od početnog vrha s do svakog od vrhova v predstavlja duljinu najkraćeg puta od s do v.

Naredba u scipy.sparse.csgraph-u koja automatski izvršava Bellman-Ford algoritam je:
$scipy.sparse.csgraph.bellman\_ford(csgraph, directed = True, indices = None,$
$return\_predecessors = False, unweighted = False)$
Svi parametri definirani su analogno kao kod prethodnih algoritama.



Slika 2.5: Bellman-Ford algoritam na grafu

Primjer 2.7. *U ovom primjeru, pogledat ćemo kod za izvršavanje Bellman-Ford algoritma za graf sa slike 2.5.*

```
import numpy as np
from scipy.sparse.csgraph import bellman_ford

#Težinska matrica susjedstva usmjerenog grafa

csgraph=np.array([
    [0, 4, 2, 0, 0],
    [0, 0, 3, 2, 3],
    [0, 1, 0, 4, 0],
    [0, 0, 0, 0, 0],
    [0, 0, 0, -5, 0]
])

# Poziv funkcije za Bellman-Ford algoritam
distances, predecessors = bellman_ford(csgraph, directed=True,\
 return_predecessors=True)

# Ispis rezultata
print("Udaljenosti od početnog vrha:")
print(distances)
print("Matrica prethodnika:")
print(predecessors)
```

Dobivaju se sljedeći izlazni podaci:

Udaljenosti od početnog vrha:

$$\begin{pmatrix} 0 & 3 & 2 & 1 & 6 \\ np.inf & 0 & 3 & -2 & 3 \\ np.inf & 1 & 0 & -1 & 4 \\ np.inf & np.inf & np.inf & 0 & np.inf \\ np.inf & np.inf & np.inf & -5 & 0 \end{pmatrix}.$$

Matrica prethodnika:

$$\begin{pmatrix} -9999 & 2 & 0 & 4 & 1 \\ -9999 & -9999 & 1 & 4 & 1 \\ -9999 & 2 & -9999 & 4 & 1 \\ -9999 & -9999 & -9999 & -9999 & -9999 \\ -9999 & -9999 & -9999 & 4 & -9999 \end{pmatrix}.$$

Poglavlje 3

Zaključak

Cilj je rada bio pomoću naredbi iz Python modula *scipy.sparse.csgraph* provesti neke algoritme na grafovima koji daju rješenje određenih problema, kao što je problem pronalaska najkraćeg puta na grafu. Upoznali smo se s Dijkstrinim algoritmom, koji pronalazi najkraći put u grafu od početnog vrha do svih preostalih vrhova, bez bridova negativnih težina. Zatim, upoznali smo se s Floyd-Warshall algoritmom koji pronalazi najmanje udaljenosti između svih parova vrhova u grafu. Za kraj, dotaknuli smo se Bellman-Ford algoritma koji služi za pronalaženje najkraćeg puta na težinskom grafu, čak i ako graf sadrži negativne težine bridova, ali samo ukoliko nema negativnih ciklusa.
Svi ovi algoritmi, kao i njima slični, izuzetno su korisni i primjenjivi, kako u matematici tako i u drugim znanostima.

Popis slika

Bibliografija

[1] J. A. Bondy, U. S. R. Murty, *Graph Theory, Springer*, New York, 2008.

[2] SciPy User Guide (`https://docs.scipy.org/doc/scipy/tutorial/index.html#user-guide`), 24.6.2024.

[3] Pydot project description (`https://pypi.org/project/pydot/`), 24.6.2024.