* 1. Zadatak 3

Na slici se nalazi pojednostavljeni prikaz središnje prometne mreže grada Ivanca. Također, prikazan je i pripadni graf prometne mreže. Potrebno je napraviti analizu ranjivosti ulica grada Ivanca.

1. Za rješavanje problema ove veličine potrebno je napraviti programsko rješenje koje će na osnovu unesenih podataka o prometnoj mreži ispisati najkritičniju ulicu. Programsko rješenje mora rješavati cijelu domenu problema, a ne samo ovaj graf!
2. Izračunajte vremensku složenost izrađenog algoritma (Pomoćna literatura: Lovrenčić, 2018)
3. Razmislite o mogućem poboljšanju prometne mreže grada na osnovu dobivenih rezultata. Navedite ulicu ili ulice koje bi bile dovoljne da se smanji kritičnost prometne mreže.

A picture containing sky, indoor, table

Description automatically generated

* 1. Rješenje zadatka 3

1. Za rješavanje problema ove veličine potrebno je napraviti programsko rješenje koje će na osnovu unesenih podataka o prometnoj mreži ispisati najkritičniju ulicu. Programsko rješenje mora rješavati cijelu domenu problema, a ne samo ovaj graf!

Programsko rješenje izrađeno je u Pythonu [3], s bibliotekama NetworkX [4] i MatPlotLib [5].

Funkcija sum\_of\_shortest\_paths() izračunava sumu najkraćih puteva u grafu (ili podgrafu) G. U funkciji se iterativno izračunava najkraći put između svih parova vrhova grafa G te se duljine puteva zbrajaju.

def sum\_of\_shortest\_paths(G):

    referent\_val = 0

    for s in G.nodes:

        for t in G.nodes:

                if(nx.has\_path(G,s,t)):

                    referent\_val = referent\_val + nx.shortest\_path\_length(G,

source=s, target=t, weight='weight')

    return referent\_val

Funkcija calculate\_criticallity\_for\_given\_edge() izračunava kritičnost danog brida ee u početnom grafu G. U funkciji se izračunava suma najkraćih puteva u podgrafu H koji ne sadrži brid (ee – edge element) koji je proslijeđen u parametru. Parametar m definira način izvršavanja. Način izvršavanja 0 - Grafovi u kojemu su lukovi između para vrhova (v1, v2) jednakih težina. Način izvršavanja 1 - Grafovi u kojemu lukovi između para vrhova (v1, v2) različitih težina.

def calculate\_criticallity\_for\_given\_edge(G, ee, m):

    try:

        H = G.copy()

        if m==0:

            H.remove\_edge(ee[0],ee[1])

            if(H.has\_edge(ee[1],ee[0])):

                H.remove\_edge(ee[1],ee[0])

        elif m==1:

            H.remove\_edge(ee[0],ee[1])

        edge\_val = sum\_of\_shortest\_paths(H)

        H.clear()

        return edge\_val

    except Exception as e:

        print('', end = '')

Funkcija calculate\_referent\_node\_value() izračunava referentnu vrijednost za dani vrh u početnom grafu G tako da stvori podgraf H u kojemu se uklanja vrh (ne – node element) proslijeđen u parametru i bridovi koji ulaze/izlaze iz njega, zatim se lukovi povezuju s obzirom na susjedne vrhove vrha ne i dodjeljuje im se odgovarajuća težina. Parametar m definira način izvršavanja. Način izvršavanja 0 - Grafovi u kojemu su lukovi između para vrhova (v1, v2) jednakih težina. Način izvršavanja 1 - Grafovi u kojemu lukovi između para vrhova (v1, v2) različitih težina.

def calculate\_referent\_node\_value(G, ne, m):

    H = G.copy()

    edge\_list\_for\_ne = []

    if m==0:

        for ee in G.edges:

            if ee[1]==ne:

                edge\_list\_for\_ne.append(ee)

    elif m==1:

        for ee in G.edges:

            if ee[1]==ne or ee[0]==ne:

                edge\_list\_for\_ne.append(ee)

    for ee in edge\_list\_for\_ne:

        try:

            H.remove\_edge(ee[0],ee[1])

        except Exception as e:

            print('', end = '')

        try:

            H.remove\_edge(ee[1],ee[0])

        except Exception as e:

            print('', end = '')

    H.remove\_node(ne)

    if m==0:

        for sv in edge\_list\_for\_ne:

            for dv in edge\_list\_for\_ne:

                if sv!=dv:

                    s=sv[0]

                    d=dv[0]

                    w = G[sv[0]][sv[1]][0]['weight'] +

G[dv[0]][dv[1]][0]['weight']

                    H.add\_edge(s, d, weight=w)

        try:

            return sum\_of\_shortest\_paths(H)

        except:

            return 0

    elif m==1:

        for sv in edge\_list\_for\_ne:

            for dv in edge\_list\_for\_ne:

                if  sv!=dv and sv[1]==dv[0]:

                    s=sv[0]

                    d=dv[1]

                    w = G[sv[0]][sv[1]][0]['weight'] +

G[dv[0]][dv[1]][0]['weight']

                    if s!=ne and d!=ne:

                        H.add\_edge(s, d, weight=w)

        try:

            return sum\_of\_shortest\_paths(H)

        except:

            return 0

Funkcija calculate\_sum\_of\_shortest\_paths\_without\_given\_node() izračunava sumu najkraćih puteva između svakog para vrhova u pografu H koji ne sadrži vrh koji je proslijeđen u parametru i lukova koji ulaze/izlaze iz njega. Nakon što se provedu promijene (brisanje lukova i vrha) u podgrafu H poziva se prethodno opisana funkcija sum\_of\_shortest\_paths(). Parametar G predstavlja početni graf (ne – node element) u kojemu se nalazi vrh (ne – node element) kojeg želimo ukloniti iz podgrafa. Funkcija se temelji na stvaranju podgrafa H na kojem se provode prethodno opisane promijene te poziva funkciju sum\_of\_shortest\_paths().

def calculate\_sum\_of\_shortest\_paths\_without\_given\_node(G, ne):

    H = G.copy()

    edge\_list\_for\_ne = []

    for ee in G.edges:

        if ee[1]==ne:

            edge\_list\_for\_ne.append(ee)

    for ee in edge\_list\_for\_ne:

        try:

            H.remove\_edge(ee[0],ee[1])

        except Exception as e:

            print('', end = '')

        try:

            H.remove\_edge(ee[1],ee[0])

        except Exception as e:

            print('', end = '')

    H.remove\_node(ne)

    try:

        return sum\_of\_shortest\_paths(H)

    except:

        return 0

1. Izračunajte vremensku složenost izrađenog algoritma (Pomoćna literatura: Lovrenčić, 2018)

Složenost ovog algoritma je gdje predstavlja broj vrhova grafa.

Naime, za izračun najkraćih udaljenosti između pojedinih vrhova koristili smo poboljšani Dijkstrin algoritam kojem je složenost . Za određivanje najkritičnije ulice potrebno je ovaj algoritam izvesti jednom za izračun referentne vrijednosti te ga ponoviti za svaki brid. U najgorem slučaju, taj se algoritam treba ponoviti + 1 puta, odnosno kada bi u usmjerenom grafu svaki vrh bio povezan sa svakim bridom u jednom i drugom smjeru, bilo bi bridova, što je – . Očito je ( – ) + 1 broj manji od što znači da je vremenska složenost . Dakle, zaključujemo da je ukupna vremenska složenost ovog dijela algoritma .

Nadalje, potrebno je izračunati vremensku složenost za drugi dio algoritma, odnosno za određivanje najkritičnijeg raskrižja. Za to je također potrebno izračunati najkraće udaljenosti za što koristimo poboljšani Dijkstrin algoritam kojem je složenost . Najprije za svaki vrh treba dobiti referentnu vrijednost (), a zatim za svaki vrh treba dobiti zbroj svih najkraćih puteva kada maknemo vrh i pripadne bridove za svaki vrh (). Dakle, poboljšani Dijkstrin algoritam trebamo izvesti ukupno puta. Stoga dobivamo da je ukupna vremenska složenost tog dijela algoritma .

Budući da se ta dva dijela algoritma izvode slijedno, uzimamo da je vremenska složenost cijelog algoritma . [6]

1. Razmislite o mogućem poboljšanju prometne mreže grada na osnovu dobivenih rezultata. Navedite ulicu ili ulice koje bi bile dovoljne da se smanji kritičnost prometne mreže.

* Ispis rezultata:

Kriticnost brida (ulice) (1, 2, 0)=21

Kriticnost brida (ulice) (1, 4, 0)=21

Kriticnost brida (ulice) (1, 7, 0)=11

Kriticnost brida (ulice) (2, 1, 0)=21

Kriticnost brida (ulice) (2, 3, 0)=26

Kriticnost brida (ulice) (2, 4, 0)=7

Kriticnost brida (ulice) (3, 2, 0)=26

Kriticnost brida (ulice) (3, 6, 0)=61

Kriticnost brida (ulice) (3, 9, 0)=9

Kriticnost brida (ulice) (4, 1, 0)=21

Kriticnost brida (ulice) (4, 5, 0)=106

Kriticnost brida (ulice) (5, 4, 0)=106

Kriticnost brida (ulice) (5, 6, 0)=194

Kriticnost brida (ulice) (5, 7, 0)=40

Kriticnost brida (ulice) (6, 3, 0)=61

Kriticnost brida (ulice) (6, 5, 0)=194

Kriticnost brida (ulice) (6, 8, 0)=400

Kriticnost brida (ulice) (7, 1, 0)=11

Kriticnost brida (ulice) (7, 5, 0)=40

Kriticnost brida (ulice) (7, 11, 0)=167

Kriticnost brida (ulice) (8, 6, 0)=400

Kriticnost brida (ulice) (8, 9, 0)=123

Kriticnost brida (ulice) (9, 3, 0)=9

Kriticnost brida (ulice) (9, 8, 0)=123

Kriticnost brida (ulice) (9, 14, 0)=371

Kriticnost brida (ulice) (10, 8, 0)=340

Kriticnost brida (ulice) (10, 11, 0)=110

Kriticnost brida (ulice) (10, 12, 0)=327

Kriticnost brida (ulice) (11, 7, 0)=167

Kriticnost brida (ulice) (11, 10, 0)=110

Kriticnost brida (ulice) (11, 16, 0)=18

Kriticnost brida (ulice) (12, 10, 0)=327

Kriticnost brida (ulice) (12, 13, 0)=249

Kriticnost brida (ulice) (12, 16, 0)=55

Kriticnost brida (ulice) (13, 12, 0)=249

Kriticnost brida (ulice) (13, 14, 0)=106

Kriticnost brida (ulice) (13, 15, 0)=13

Kriticnost brida (ulice) (14, 9, 0)=371

Kriticnost brida (ulice) (14, 13, 0)=106

Kriticnost brida (ulice) (14, 15, 0)=39

Kriticnost brida (ulice) (15, 13, 0)=13

Kriticnost brida (ulice) (15, 14, 0)=39

Kriticnost brida (ulice) (15, 16, 0)=14

Kriticnost brida (ulice) (16, 11, 0)=18

Kriticnost brida (ulice) (16, 12, 0)=55

Kriticnost brida (ulice) (16, 15, 0)=14

Rezultati pokazuju da je ulica između vrha 6 i vrha 8 najkritičnija, zato što ima najveću vrijednost, točnije jer bi njezino zatvaranje stvaralo najveću „gužvu“. Postoje mnoga rješenja ovog problema u realnom svijetu, pa ćemo promotriti nekolicinu njih.

Prvo rješenje je stvaranje još jedne dvosmjerne relacije između vrha 6 i 8 koja bi mogla raditi u slučaju zatvaranje prve.

Drugo rješenje je poboljšanje postojećih obilaznica, točnije smanjenje težine na bridovima obilaznice, trenutna obilaznica ide putem v6 ⇒ v3 ⇒ v9 ⇒ v8.

Posljednja mogućnost je dodavanje dvosmjernih ulica susjedima vrhova 6 i 8, dakle mogli bi dodati ulicu iz vrha 5 ili 3 u vrh 8, ili pak ulicu iz vrha 9 ili 10 u vrh 6, te se s time dobiva bolja obilaznica.

# Literatura

|  |  |
| --- | --- |
| [1] | G. Online, »Graph Online,« Graph Online, 2015. [Mrežno]. Available: https://graphonline.ru/en/. [Pokušaj pristupa 25 1 2020]. |
| [2] | p. d. s. B. Divjak, »Elf - Šetnje u grafu. Težinski grafovi,« [Mrežno]. Available: https://elf.foi.hr/pluginfile.php/17708/mod\_page/content/28/grafovi2.pdf. [Pokušaj pristupa 25 1 2020]. |
| [3] | G. v. Rossum, »Python,« Python Software Foundation, 1990. [Mrežno]. Available: https://www.python.org/. [Pokušaj pristupa 25 1 2020]. |
| [4] | N. developers, »NetworkX,« 2014. [Mrežno]. Available: https://networkx.github.io/. [Pokušaj pristupa 25 1 2020]. |
| [5] | J. Hunter, »matplotlib,« 2007. [Mrežno]. Available: https://matplotlib.org/. [Pokušaj pristupa 25 1 2020]. |
| [6] | A. Lovrenčić, Uvod u složenost algoritama, 2018. |