Primjena matrica u praksi (3) Algoritmi pretraživanja grafova

Tihana Strmečki[1], Hrvoje Renato Šokčić[2], Ivan Benčić[3], Domagoj Lepen[4]

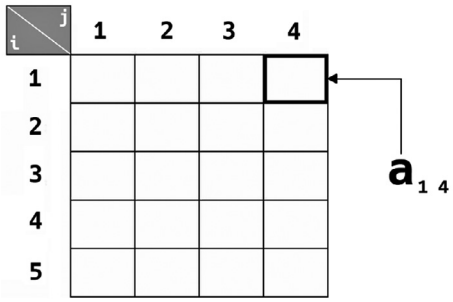
U trećem ćemo se dijelu članka na temu primjene matrica baviti njihovom ulogom u kontekstu teorije grafova, preciznije, kod algoritama pretraživanja grafova. Zapis grafa preko matrice omogućuje drugu vrstu vizualno-strukturiranog prikaza, zbog čega njegove podatke možemo na drugačiji način analizirati i manipulirati. U svrhu pronalaženja putova i otkrivanja ciklusa unutar grafa, proučit ćemo dva različita algoritma koji su ključni za istraživanje i navigaciju grafova: pretraživanje u dubinu i pretraživanje u širinu. Poznavanje i razumijevanje povezanosti matrica, grafova i algoritama pretraživanja omogućuje nam rješavanje i optimizaciju složenih računalnih problema i postupaka. Navest ćemo njihove definicije, osnovne ideje primjene, te usporediti njihove prednosti i nedostatke.

Matrice su vrlo značajan matematički koncept u području računarstva i programiranja. Njihova vrijednost u razvoju algoritama i optimizaciji performansi proizlazi iz sposobnosti da efikasno organiziraju podatke i njima manipuliraju prema potrebi korisnika. Cilj je ovog članka ilustrirati još jedan način primjene matrica tako da bude razumljiv i učenicima srednjih škola. Iz tog razloga pojam matrice i operacije nad matricama nećemo strogo matematički definirati, nego ćemo ih opisati. Matrica je pravokutna tablica od m redaka i n stupaca za koju tada kažemo da je tipa $m \times n$ (m i n su prirodni brojevi). Matrični element koji se nalazi u i -tom retku i j -tom stupcu matrice označavamo s a_{ij}, a samu matricu uglatim zagradama navodeći i njezin tip $\left[a_{ij}\right]_{m\times n}$. Kada je odgovarajuće, matrice označavamo i velikim tiskanim slovima (na primjer A, B, C, ...)

$$A = \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{bmatrix}.$$

[1]Tihana Strmečki, Tehničko veleučilište u Zagrebu
[2]Hrvoje Renato Šokčić, student, Tehničko veleučilište u Zagrebu
[3]Ivan Benčić, student, Tehničko veleučilište u Zagrebu
[4]Domagoj Lepen, student, Tehničko veleučilište u Zagrebu

Matrice mogu biti strogo pravokutnog oblika ili kvadratne (imaju jednak broj redaka i stupaca), a među često korištenim oblicima su i matrice sa samo jednim retkom ili sa samo jednim stupcem (vektori). Matrica tipa $m \times n$ sadrži $m \cdot n$ elemenata poput realnih ili često cijelih brojeva, no može sadržavati i funkcije, vektore, pa čak i same matrice.



Slika 1. Primjer matrice tipa $A_{5\times 4}$ i njezinog elementa a_{14}

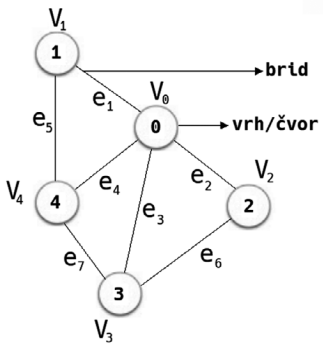
Među najčešćim operacijama na skupu matrica nalaze se zbrajanje, oduzimanje i množenje matrica, transponiranje i inverzija. One omogućavaju izvođenje kompleksnih izračuna poput rješavanja linearnih jednadžbi, analize podataka i modeliranja problema. Za detalje izvođenja ovih operacija čitatelj se upućuje na [1]. U području programiranja, matrice se koriste u različite svrhe, od kojih navodimo najpoznatije. U grafičkom programiranju nužne su za transformaciju i prikazivanje objekata u 3D prostoru, kao što je objašnjeno u [5]. U analizi podataka preko matrica se sortiraju i manipuliraju podatci, te primjenjuju različite statističke metode. Također, matrice se koriste u kriptografiji za šifriranje i dešifriranje podataka. Nama će od posebnog značaja biti matrice susjedstva, koje ćemo detaljnije proučiti i prikazati, te pojasniti njihovu primjenu kod algoritama pretraživanja grafova.

Teorija grafova dio je diskretne matematike koji se bavi proučavanjem grafova. U okviru područja računarstva i programiranja, grafovi imaju ključnu ulogu u prikazivanju i u analizi odnosa između entiteta (entiteti u grafovima su pojedinačni članovi ili objekti koji se povezuju s drugim entitetima putem bridova ili veza). Oni pružaju snažan alat za prikazivanje i usvajanje složenih sustava i njihovih povezanih dijelova. Razumijevanje i učinkovito pretraživanje grafova ključni su za razvoj korisnih algoritama i za uspješno rješavanje praktičnih problema. Društvene mreže, transportni sustavi, analiza podataka i rutiranje interneta (proces preusmjeravanja podataka između različitih mrežnih čvorova ili rutera kako bi se omogućila ispravna dostava informacija putem interneta) samo su neka od područja u kojima se grafovi često koriste.

Graf G definiran je pomoću dva skupa i funkcije koja opisuje odnose među njima. Neprazan konačni skup *V(G)* skup je čije elemente zovemo *vrhovi* ili *čvorovi* i označavamo prirodnim brojevima i nulom. Konačni skup *E(G)* skup je različitih

parova elemenata skupa *V(G),* koje zovemo *bridovi*. Skup bridova mora biti disjunktan sa skupom vrhova (disjunktni skupovi su oni čiji je presjek prazan, odnosno nemaju zajedničkih elemenata). Skup *V(G)* kraće ćemo označavati s *V*, a skup *E(G)* s *E*. Za bridove *e* i *f* kažemo da su *susjedni bridovi* ako postoji vrh u tome grafu koji im je zajednički. Dakle, graf u oznaci *G* jest uređena trojka $G=\left(V,E,\varphi_G\right)$, pri čemu se φ_G naziva *funkcija incidencije*. Ona svakom bridu grafa iz skupa *E* pridružuje par ne nužno različitih vrhova iz *V* koje taj brid spaja. Za vrhove *i* i *j* kažemo da su *susjedni vrhovi* ako postoji brid takav da vrijedi $\varphi_G(e)=\{i,j\}$ (u tom grafu postoji brid koji ih spaja). Ponekad ćemo koristiti i kraći zapis za graf *G* = (*V*, *E*).

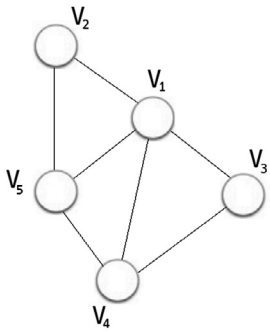
Za razumijevanje će nam biti potrebne neke definicije, stoga ih navodimo. Graf je *povezan* ako i samo ako postoji put između svaka dva vrha toga grafa. *Put u grafu* definira se kao konačni niz bridova $e_1,e_2,e_3,...,e_{n-1},e_n$, pri čemu su svaka dva brida susjedna, a vrhovi u putu različiti, što možemo označavati s $e_1 \to e_2 \to e_3 \to \cdots \to e_n$. *Ciklus* u grafu jest put koji počinje u jednom čvoru, te završava u tom istom čvoru. Da bi graf bio *cikličan*, mora sadržavati barem jedan ciklus. Grafovi mogu biti *usmjereni* ili *neusmjereni*. Usmjereni graf ili digraf ima vrhove čiji su bridovi usmjereni iz jednog vrha prema drugom, u suprotnom kažemo da je graf neusmjeren odnosno pravi. U pravom grafu, brid od vrha *i* do vrha *j* identificira se linijom označavajući na taj način da su dva vrha pridružena jednom bridu međusobno ravnopravna. U digrafu su između svaka dva vrha moguća dva različita smjera. U *težinskim grafovima* svaki brid ima svoju vrijednost, koju definira funkcija *w*. Ona bridu *e* pridružuje realan broj *w*(*e*), njegovu težinu. U *bestežinskim grafovima* vrijednost svakog brida prema dogovoru iznosi 1.



Slika 2. Prikaz bestežinskog neusmjerenog grafa i njegovih čvorova i bridova

Kao poveznica između matrica i grafova koristi se matrica susjedstva, pogotovo u algoritmima pretraživanja. *Matrica susjedstva* kvadratna je matrica u oznaci *M* tipa $V \times V$ (pri čemu je *V* skup čvorova grafa), koja predstavlja grafičku strukturu povezanosti čvorova unutar grafa. Element matrice a_{ij} definira se kao 1 u slučaju kada postoji brid između čvora *i* i *j* takav da je *i* početni, a *j* završni čvor brida. Kada takav brid

ne postoji, element a_{ij} definira se kao 0. Matrica susjeda simetrična je, što znači da se elementi zrcale po glavnoj dijagonali. Uz pomoć matrice susjedstva lako se može pratiti posjećenost čvorova (koliko bridova ima taj čvor) i izvršiti pretraživanje susjeda.



i \ j	1	2	3	4	5
1	0	1	1	1	1
2	1	0	0	0	1
3	1	0	0	1	0
4	1	0	1	0	1
5	1	1	0	1	0

Slika 3. Prikaz grafa i njegove matrice susjedstva

Odavno znamo da su temelj računalnog i matematičkog svijeta algoritmi. Bez njih bi bilo nemoguće održavati računalne procese i rješavati različite probleme na efikasan način. Algoritam je konačan niz koraka koji pretvaraju ulazne podatke u izlazne, pri čemu je svaki korak precizno definiran. Algoritmi mogu biti iskazani različitim metodama poput dijagrama toka ili pseudo koda, te trebaju biti prilagođeni tako da se efikasno koriste ograničene količine memorije i vremena. Kratko navodimo koncepte vremenske složenosti i prostorne složenosti kao važne pojmove za procjenu korisnosti algoritama. *Vremenska složenost* (engl. *time complexity*) opisuje broj elementarnih operacija koje algoritam obavlja u odnosu na veličinu ulaznih podataka. Uglavnom se za operacije kod računanja ove složenosti uzimaju iteracije petlji i poziva funkcija koje su najkompleksnije, te najviše vremenski utječu na brzinu algoritma. *Prostorna složenost* (engl. *space complexity*) količinski opisuje memorijski prostor koji algoritam koristi za pomoćne varijable i strukture podataka u odnosu na veličinu ulaznih podataka. Ova se memorija ne odnosi na ulazne podatke i prostor za spremanje izlaznih vrijednosti algoritma. Vremenska i prostorna složenost najčešće se zapisuju u velikoj O notaciji[5], koje daje prvotni pregled efikasnosti algoritma bez potrebe detaljnijeg testiranja i mjerenja.

Postoje algoritmi kod kojih je potrebno ponavljanje uz uvjet za obavljanje određene radnje. Pod te algoritme spadaju algoritmi pretraživanja koje ćemo mi proučavati. Ovakav se tip algoritma može implementirati na iterativni i rekurzivni način. Iterativni način obuhvaća korištenje petlji za višestruko ponavljanje dijela algoritma, poput za (engl. *for*), dok (engl. *while*) i činiti-dok (engl. *do-while*). Iterativni algoritmi izvršavaju se korak po korak, pri čemu svaki korak označava promjenu stanja ili manipulaciju podatcima. Uvjet pod kojim se vrši ponavljanje, odnosno broj koraka petlje, određuje programer. Rekurzivan način obuhvaća korištenje mogućnosti pozivanja

[5]Matematička notacija koja daje teoretsku gornju granicu zauzeća memorije ili vremena izvršavanja algoritma

funkcije unutar same sebe, čime se omogućuje uvjetno ponavljanje daljnjim rekurzivnim pozivima sve dok se ne dostigne bazni slučaj koji onemogućuje daljnje pozive. Svaki rekurzivno implementirani algoritam ima tri osnovna dijela: bazni slučaj, rekurzivni poziv funkcije, te dio koda koji omogućuje napredak prema baznom slučaju.

Mi ćemo se fokusirati na dva algoritma za prelaženje i pretraživanje struktura podataka unutar grafova. *Pretraživanje u dubinu*, koje kraće označavamo DFS (engl. *depth first search*), obično se implementira rekurzivno, no može se implementirati i iterativno (s petljama). Kod prikaza implementacije DFS algoritma mi ćemo koristiti rekurzivnu verziju zbog jednostavnije izvedbe. DFS algoritam funkcionira tako da dubinskim pretraživanjem nastoji analizirati naznačene grane susjednih čvorova[6] unutar strukture podataka. Ako duž trenutne grane ne postoji više čvorova za pretragu, algoritam se vraća unatrag i nastavlja pretraživanje u sljedećoj dostupnoj grani susjednih čvorova.

Prvo ćemo pojasniti implementaciju DFS algoritma za matrice, pri čemu je potrebna provjera legitimnosti pozicije svakog elementa unutar matrice. Stoga za element a_{ij} matrice tipa $m \times n$ treba uvjetovati da vrijedi $1 \leq i \leq m$ i $1 \leq j \leq n$. Prvo se zadaje početni element matrice nad kojim se započinje pretraga, potom se funkcija DFS algoritma poziva za svaki susjedni element koji nije do tada posjećen. Kod pretraživanja se koristi jednodimenzijsko polje[7] koje predstavlja pomak po retku ili pomak po stupcu. Pomak se zatim nadodaje na koordinate elementa na kojem se trenutačno nalazimo kako bismo pristupili jednom od osam mogućih susjednih elementa. Kod polja za određivanje pomaka poredak posjećivanja susjeda određuje sam programer. Oznaka posjećenosti čvora *u* može se izraziti oznakom TRUE, a za rekurzivno pretraživanje svakog njegovog do tada neposjećenog susjeda *v* ponovno se

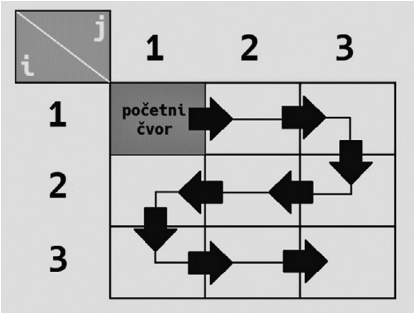
```
funkcija DFS_algoritam(matrica, i, j): // definicija funkcije
       p_j = [1,-1,0,0,1,-1,-1,1] // polje za horizontalni pomak
       p_i = [0,0,1,-1,1,-1,1,-1] // polje za vertikalni pomak
       ako matrica[i][j] nije označena: // provjera posjećenosti
              označiti matrica[i][j] = TRUE  // potvrđivanje posjećivanja
// petlja za prolaz susjednih elemenata
              za b = 1 do 8 činiti:
// provjera ispravnosti koordinata
                     ako (i + p_i[b]) i (j + p_j[b]) unutar matrice:
                     // rekurzivno pozivanje funkcije sa susj. elementom
                            DFS_algoritam(matrica, i + p_i[b], j + p_j[b])
```

[6] u kontekstu podatkovnih struktura čvor predstavlja ćeliju s određenim podatcima koja može biti povezana sa ostalim ćelijama čineći mrežu povezanih podataka

[7] linearni redoslijed elemenata istog tipa koji su spremljeni u kontinuiranom memorijskom prostoru, pri čemu svaki element ima indeks koji označava njegovu poziciju unutar polja

poziva DFS algoritam. Slijedi prikaz izvedbe DFS algoritma za pretraživanje matrica u pseudo kodu.

Za primjer možemo uzeti kvadratnu matricu tipa 3×3 i zadati početni element na sjecištu prvog retka i prvog stupca a_{11}. Slika 4. prikazuje jednu verziju primjene DFS algoritma.

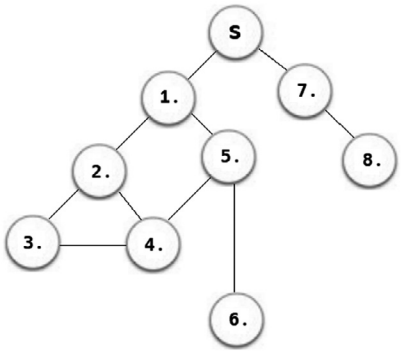


Slika 4. Prikaz postupka potpunog pretraživanja matrice preko DFS algoritma

Implementacija DFS algoritma za grafove slična je implementaciji za matrice. Za razliku od matričnih elemenata kojih je konačno, u grafu jedan čvor u teoriji može imati beskonačan broj susjednih čvorova. Ova mogućnost nameće potrebu za strukturom u kojoj je čvor reprezentiran kao objekt koji se sastoji od vrijednosti i liste susjednih čvorova. Zbog moguće pojave cikličkih grafova, što dovodi do beskonačnih petlji i do prekoračenja maksimalne rekurzivne dubine pretrage, kod pretraživanja grafova dodatno se implementira proizvoljna struktura podataka koja služi za praćenje posjećenih čvorova. Realizacija relacija između čvorova unutar grafa u ovom primjeru DFS algoritma postignuta je kroz matricu susjedstva, u pseudo kodu označena s mat_sud. Slijedi prikaz izvedbe DFS algoritma za pretraživanje grafova u pseudo kodu.

```
funkcija DFS_algoritam(graf, p_čvor, pos_čvor, mat_sud): // definicija funkcije
       ako pos_čvor[p_čvor] nije posjećen: // provjera posjećenosti
              označiti pos_čvor[p_čvor] = TRUE // potvrđivanje posjećivanja
// uzima se svaki povezani susjedni čvor iz matrice susjedstva
              za svaki s_čvor  mat_sud[p_čvor] tako da
       mat_sud[p_čvor][s_čvor] nije 0 činiti:
// ponovno pozivanje DFS-a nad odabranim čvorom
                     DFS_algoritam(graf, graf[s_čvor], pos_čvor, mat_sud)
```

Za primjer ćemo uzeti graf s 9 čvorova, pri čemu je početni čvor označen slovom *s*. Slika 5. prikazuje jednu verziju primjene DFS algoritma na ovaj graf.



Slika 5. Prikaz redoslijeda potpunog pretraživanja grafa preko DFS algoritma

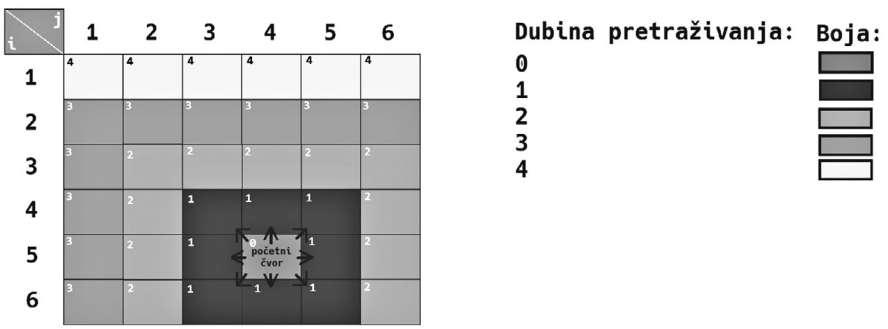
Za razliku od DFS-a, *pretraživanje u širinu*, kraće BFS (engl. *breadth first search*), implementira se iterativno zbog jedinstvene strukture podataka za pamćenje potencijalnih čvorova čija bi je rekurzivna implementacija činila nepotrebno kompleksnom. BFS algoritam započinje od zadanog početnog čvora grafa i istražuje sve susjedne čvorove na trenutnoj dubini prije prelaska na susjedne čvorove na sljedećoj razini dubine. Za praćenje susjednih čvorova koje je potrebno istražiti koristi se dodatna memorija u obliku strukture podataka red[8]. Ovakav način pretraživanja također se naziva slojevita obrada, s obzirom na to da je prije prelaska na sljedeću dubinu potrebno istražiti sve čvorove prethodne dubine. Zbog toga se BFS algoritam najčešće koristi za pronalaženje najkraćeg puta između dvaju čvorova kod grafova i matrica.

Inicijalna implementacija BFS algoritma za primjenu na matrice, koja uključuje inicijalizaciju matrice, jednodimenzijskih polja za pomak i određivanja početnog elementa, ista je kao i kod implementacije DFS algoritma. BFS algoritam kod matrica prolazi kroz sve susjede određenog člana matrice i dodaje ih na red. Pri dodavanju i pretrazi, provjerava se posjećenost elementa i ispravnost njegovih koordinata unutar matrice. Ispravan novootkriveni susjedni element dodaje se na začelje reda, dok se za pretragu uzimaju elementi koji se nalaze na čelu reda. Ovime se postiže slojevito pretraživanje kod kojeg se slojevi šire od centra, što je prikazano na Slici 6. Prije prelaska na sljedeći element, trenutni se obilježava kao posjećen i uklanja s reda. Algoritam završava prilikom nailaska na zadani cilj ili kada više ne postoji elemenata za pretragu unutar matrice. Kod BFS se također koristi matrica susjedstva, ali s ulogom praćenja redoslijeda posjećivanja elemenata. To se može izraziti formulom *redoslijed_posjeta[u]*, koja bilježi redoslijed posjećenosti elemenata kako bi se kasnije analiziralo ili koristilo u drugim dijelovima algoritma. Slijedi prikaz izvedbe BFS algoritma za pretraživanje matrica u pseudo kodu.

8[eng. *queue*] – FIFO (First In First Out) podatkovna struktura kod koje se podatci nadodaju na začelje strukture, a kod brisanja ili dohvaćanja podataka uzimaju se s čela strukture

```
funkcija BFS_algoritam(matrica, i, j): // definicija funkcije
       p_j = [1,-1,0,0,1,-1,-1,1] // polje za horizontalni pomak
       p_i = [0,0,1,-1,1,-1,1,-1] // polje za vertikalni pomak
       red = [] // inicijalizacija praznog reda susjeda
       redoslijed_posjeta = [] // inicijalizacija praznog niza
red.dodaj_natrag([i, j]) // dodavanje početnog elementa na red
dok red nije prazan: // postoji li još elemenata za pretragu
       trenutni_element = red.uzmi_prednji() // uzimanje koordinate elemen-
ta za pretragu
       red.izbriši_prednji() // brisanje trenutnog elementa sa reda čekanja
       ako matrica[i][j] nije označena: // provjera posjećenosti
       označiti matrica[i][j] = TRUE // potvrđivanje posjećivanja
      redoslijed_posjeta.dodaj_natrag(trenutni_element) // dodavanje
koordinate u redoslijed posjećivanja
       za b = 1 do 8 činiti: // prolazak svih susjeda trenutnom elementu
              // susjed se dodaje u red
              red.dodaj_natrag([i + p_i[b], j + p_j[b]])
```

Za primjer možemo uzeti kvadratnu matricu tipa 6×6 i zadati početni element na sjecištu petog retka i četvrtog stupca a_{54}. Slika 6. prikazuje jednu verziju primjene BFS algoritma na zadanu matricu.

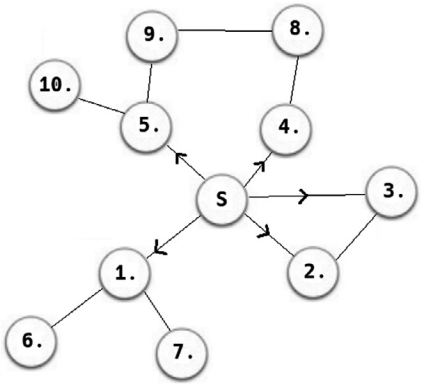


Slika 6. Prikaz postupka potpunog pretraživanja matrice preko BFS algoritma

Kao i kod DFS algoritma, osnovne razlike kod implementacije BFS algoritma za graf su korištenje dodatne strukture podataka za praćenje posjećenih čvorova i drugačiji način prolaženja kroz elemente grafa. Realizacija relacija između čvorova u ovom primjeru BFS algoritma izvedena je kroz listu susjednih čvorova koja se posebno stvara za svaku instancu čvora u grafu.

```
funkcija BFS_algoritam(graf, p_čvor): // definicija funkcije
red = [] // inicijalizacija praznog reda susjeda
red.dodaj_natrag(p_čvor) // dodavanje početnog čvora na red
       pos_čvor = {} // hash mapa za praćenje posjećenih čvorova
dok red nije prazan: // postoji li još čvorova za pretragu
       p_čvor = red.uzmi_prednji() // uzimanje koordinate čvora za pretragu
       red.izbriši_prednji() // brisanje čvora sa reda čekanja
       ako pos_čvor[p_čvor] nije posjećen: // provjera posjećenosti
            označiti pos_čvor[p_čvor] = TRUE // potvrđivanje posjećivanja
// petlja za prolaz susjednih čvorova
              za svaki s_čvor  graf[p_čvor].susjedi() činiti:
                     // dodavanje čvora na red čekanja
              red.dodaj_natrag(s_čvor)
```

Za primjer ćemo uzeti graf s 11 čvorova, pri čemu je početni čvor označen slovom *s*. Slika 7. prikazuje jednu verziju primjene BFS algoritma na ovaj graf.



Slika 7. Prikaz postupka potpunog pretraživanja grafa preko BFS algoritma

Za postizanje najboljih rezultata izvođenja BFS i DFS algoritama potrebno je odabrati odgovarajuću strukturu za pohranjivanje podataka relacija. Već spomenute matrice susjedstva i grafovi najčešće se koriste za ove algoritme pretrage. Odabir između te dvije opcije određen je sa složenosti podataka. Matrice susjedstva koriste se kada je graf sastavljen od više bridova naspram broja čvorova. Za takav graf $G = (V, E)$ vrijedi $|E| >> |V|$. On se još naziva i *gusto povezani graf*. Tada će elementi matrice biti ispunjeni većinski jedinicama, što će omogućiti brži pristup podatcima o susjedima svakog čvora. Graf kao struktura podataka koristi se kada graf ima puno manje bridova u odnosu na broj čvorova. Za takav graf $G = (V, E)$ vrijedi

$|E| << |V|$. On se još naziva *rijetko povezani graf*. U ovom slučaju elementi matrice susjedstva većinski su ispunjeni nulama, što bi dovelo do neefikasnosti korištenja te strukture, stoga se koristi graf kao struktura za pohranjivanje podataka relacija.

Vremenska složenost BFS i DFS algoritma predočena O notacijom je $O(|V|+|E|)$, pri čemu $|V|$ predstavlja broj čvorova, a $|E|$ broj bridova. Ovaj nam zapis govori da će oba algoritma u najgorem slučaju posjetiti sve čvorove i proći kroz sve moguće bridove kod pretrage. Memorijska složenost BFS i DFS algoritma iznosi $O(|V|)$, što znači da će oba algoritma maksimalno pohraniti sve čvorove u svoju strukturu podataka u svrhu daljnje pretrage. Zanimljiva je činjenica da će memorijska složenost obaju algoritama biti najveća kada se koriste za pretragu grafova koji svojom strukturom podsjećaju na način pretrage danog algoritma. Dakle, BFS će davati najgoru memorijsku složenost za grafove koji se prostiru u širinu u što više grana, dok će DFS davati najgoru memorijsku složenost za grafove koji se prostiru duž jedne grane.

U ovom smo se članku fokusirali na primjenu matrica unutar algoritama pretraživanja grafova. Uveli smo osnovne pojmove, te se posebno osvrnuli na matrice susjedstva kao alat za reprezentaciju grafova. Kod algoritama za pretraživanje grafova detaljno smo objasnili princip rada implementacija posebno za matrice i posebno za grafove s primjerima u pseudo kodu. Dodatno smo se uvjerili da su matrice snažan alat u kontekstu računarstva, pružajući jednostavnu implementaciju i efikasno pretraživanje grafičkih podataka, dok su grafovi snažan alat za prikazivanje i analizu složenih sustava. Razumijevanje i vještina implementacija matrica predstavljaju važan dio znanja svakog programera koji se bavi grafovima i optimizacijom algoritama.

Literatura:

1. Strmečki, T.: *Matematika 1,* Tehničko veleučilište u Zagrebu, Zagreb, 2021.
2. Bakić, D.: *Linearna algebra*, Školska knjiga, Zagreb, 2008.
3. Kovačić B., Marohnić L., Strmečki T.: *Repetitorij matematike za studente elektrotehnike, informatike i računarstva*, Tehničko veleučilište u Zagrebu, 2016.
4. Strmečki, T., Mičković, V., Rak, R., Pajković, L.: *Primjena matrica u praksi 1 – Digitalna obrada fotografija*, Poučak, 23., 92.; 68-79, 2022.
5. Strmečki, T., Sirovatka, G., Rak, R., Pajković, L.: *Primjena matrica u praksi 2 – Geometrijske transformacije,* Poučak 24., 94.; 11-21, 2023.
6. https://www.geeksforgeeks.org/
7. https://hrcak.srce.hr/file/3246
8. http://www.zemris.fer.hr/predmeti/ra/LabRadovi/dokumentacija/2021AmalijaRamljak.pdf
9. https://www.javatpoint.com/