8.1. DEFINIRANJE POJMOVA

Kod usmjerenog grafa, svaka veza povećava ulazni stupanj nekog čvora za jedan, te također izlazni stupanj tog istog ili nekog drugog čvora za jedan. Dakle, za usmjereni graf G=(V, E) vrijedi



Gdje je |E| kardinalni broj (broj elemenata u skupu E).

Kod neusmjerenog grafa, svaka veza doprinosi povećanju stupnja dva različita čvora pa slijedi



**Duljina puta** jednaka je broju veza na putu. Kažemo da je čvor **dohvatljiv** iz čvora u ako postoji put iz **u** prema **u'**. Put je **jednostavan** ako su svi čvorovi na putu različiti (osim eventualno prvog i zadnjeg).

**Ciklus** u usmjerenom grafu je put koji sadrži barem jednu vezu i za kojeg vrijedi v0 = vk. Ciklus je jednostavan ako su čvorovi v1, v2,......, vk različiti. Petlju koja se zatvara sama u sebe smatramo jednostavnim ciklusom duljine 1. Graf koji nema niti jedan ciklus zovemo **necikličan**.

Često nas zanimaju dvije posebne klase ciklusa. Jedan je **Hamilton-ov** ciklus kod kojeg treba posjetiti svaki čvor u grafu točno jednom. **Euler-ov** ciklus je ciklus kod kojeg treba posjetiti svaku vezu u grafu točno jednom.

8.3. PRETRAŽIVANJE PO ŠIRINI (breadth-first search, BFS)

BFS(G,s) //definiramo algoritam BFS na grafu G i

izvorišnom čvoru u s

int distanca[1.....size(V)] //distance čvora

int boja[1.....size(V)] //boje čvora

čvor\_prethodni[1......size(V)] //prethodni pointer

queue Q=empty //FIFO queue

za svaki u iz V

boja[u] = bijela

distanca[u] = INF

prethodni[u] = NULL

boja[s] = siva //definiranje izvora

distanca[s] = 0

enqueue(Q,s) //stavi izvor u queue

while(Q is nonempty)

u=dequeue(Q) //u je slijedeci svor kojeg cemo posjetiti

za svaki v iz Adj[u]

if(boja[v] == bijela) //ako susjed još nije otkriven

boja[v] = siva

distanca[v] = distanca[u] + 1

prethodni[v] = u

enqueue(Q,v)

boja[u] = crna

Za ovaj algoritam ako uzmemo i vrijeme inicijalizacije slijedi da je vrijeme izvršavanja BFS jednako O(|V| + |E|). To je vrijeme koje je proporcionalno s veličinom prikaza grafa preko liste susjedstva.

8.4. NAJKRAĆI PUTOVI SVIH PAROVA (all-pairs shortest paths) Vrijeme izvršavanja je Θ(|V|4):



**Dist(int m, int i, int j)**

if(m==1) return W[i,j] //slučaj jedne veze

najbolji = INF

for k = 1 to n do //n je ukupan broj čvorova

najbolji = min(najbolji, Dist(m-1, i, k) + w[k, j])

return najbolji

**NajkraciPut(int n, int W[1...n, 1...n])**

array D[1...n-1][1...n, 1...n]

kopiraj W u D[1] //inicijaliziranje D[1]

for m = 2 to n-1 do //računanje D[m] iz D[m-1]

D[m] = ProduzeniPut(n, D[m-1], W)

return D[n-1]

**ProduzeniPut(int n, int d[1...n, 1...n], int W[1...n, 1...n])**

//kopiraj d u privremenu matricu

matrix dd[1... .n, 1...n] = d[1... .n, 1....n]

for i=1 to n do

for j=1 to n do

for k=1 to n do

dd[i, j] = min(dd[i, j], d[i, k] + W[k, j])

return dd //vrati matricu cijena

Floyd-Warshall algoritam

Vrijeme izvršavanja je Θ(|V|3):

**Floyd-Warshall(int n, int W[1...n,1...n])**

array d[1...n,1...n]

for i=1 to n do

for j=1 to n do

d[i,j]=W[i,j]

pred[i,j]=NULL

for k=1 to n do

for i = 1 to n do

for j= 1 to n do

if (d[i,k] + d[k,j]) < d[i,j])

d[i,j] = d[i,k] + d[k,j]

pred[i,j] = k

return d

**NajkraćiPut(i,j)**

if pred[i,j] = NULL

ispisi(i,j)

else

NajkraćiPut(i,pred[i,j])

NajkraćiPut(pred[i,j],j)

Najduža zajednička podsekvenca - LCS

Vrijeme izvršavanja je O(mn):

**LCS(char x[1..m], char y[1..n])**

int c[0..m, 0..n]

for i = 0 to m do

c[i,0] = 0

b[i,0] = 0

for j = 0 to n do

c[0,j] = 0

b[0,j] = 0

for i = 1 to m do

for j = 1 to n do

if (x[i] == y[j])

c[i,j] = c[i-1, j-1] + 1

b[i,j] = GOREiLIJEVO

else if (c[i-1, j] >= c[i,j-1]

c[i,j] = c[i-1, j]

b[i,j] = GORE

else

c[i,j] = c[i, j-1]

b[i,j] = LIJEVO

return c[m,n]

**IzvlačenjeLCS(char x[1..m], char y[1..n], int b[0…m, 0..n])** LCS = prazan niz

i = m j = n

while (i!= 0 && j != 0)

switch b[i,j]

case GOREiLIJEVO

dodaj x[i] u LCS

i-- j-- break

case GORE

i-- break

case LIJEVO

i-- break

return LCS

Algoritam računa stupanj svakog čvora:

a) kada je graf dan u obliku matrice susjedstva

int Stupanj [1...|V|] ; polje u koje spremamo stupnjeve čvorova

for i = 1 to |V| ; za svaki čvor zbraja sumu po retku

suma = 0 ; suma se za svaki čvor inicijalizira na 0

for j = 1 to |V|

suma = suma + A[i,j] + A[j,i] -> za usmjeren graf za naći ukupni stupanj

if (A[i,i]==1)

suma=suma+1 -> dodamo samo za NEusmjeren graf

stupanj[i] = suma ; stupanj se sprema na odgovarajuće

mjesto u polju.

Vrijeme izvršavanja O(|V|2)

b) kada je graf dan u obliku liste susjedstva

int Stupanj[1...|V|]

for i = 1 to |V|

Stupanj[i]=0 ; sve stupnjeve postavljamo na 0

for i = 1 to |V| ; idemo po čvorovima

za svaki u iz Adj(i) ; idemo po listi za svaki čvor

Stupanj[i] = Stupanj[i]+1 ; za svakog susjeda nadodamo 1

Stupanj[u]=Stupanj[u]+1 -> za usmjeren graf di triba naći ukupni stupanj

if(čvor u == čvor i) ; ako je petlja to dodaje 2

Stupanj[i]=Stupanj[i]+1 -> za NEusmjeren graf

Vrijeme izvršavanja O(|V| + |E|)

Algoritam određuje dali je dani povezani neusmjereni graf dvodijelan.

Dvodijelan (G, s) ; kao argument dobiva graf

int Pripadnost[1...size(V)] ; za svaki element ispitamo kojoj grupi

queue Q ; kojoj grupi pripada

for i = 1 to size(V)

pripadnost[i] = 0 ; u početku inicijaliziramo na 0

pripadnost [s] = 1 ;idemo na pretraživanje po susjedima   
enqueue (Q,s) ;S je početni čvor

while (Q is nonempty) ; dok ima čvorova za obradit

u = dequeue (Q)

za svaki v iz Adj(u) ; hajde po svim susjedima

if (pripadnost[v] == 0) ; nije obrađen

if (pripadnost(u) == 1)

pripadnost(v) = 2 ; jedna grupa 1 druga 2

else

pripadnost (v) = 1 ; ako je od u 2

enqueue (Q,V)

if (pripadnost(v) == pripadnost (u)) ; iz u ide u v

print „Nije dvodjelan“ ; veza, pa bi

trebali pripadati različitim skupovima

return

print „Dvodijelan je“

Neka je dano stablo G=(V, E)

a) Ako dodamo vezu u G tada novi graf sadrži ciklus.

Graf je povezan ako se svaki čvor može doseći iz svakog drugog

čvora. Necikličan povezan graf - stablo

Budući da višestruke veze nisu dozvoljene možemo povezati samo

one čvorove koji već nisu povezani (isprekidano na slici), a time

dobivamo ciklus od najmanje tri člana.

b) Ako izbrišemo vezu u G, tada nam graf nije povezan.

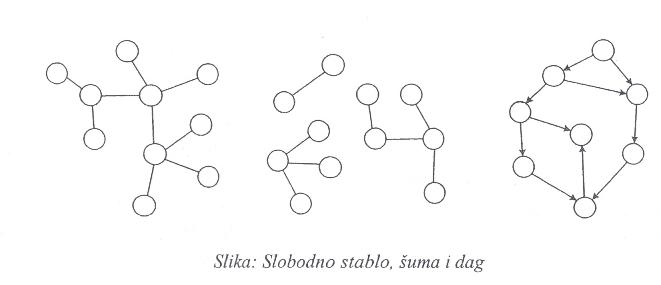
Kod stabla svaki čvor je povezan sa najmanje jednim čvorom. Dakle,

ako izbrišemo tu vezu onda to više nije stablo.

c) Postoji točno jedan jednostavan put između svaka dva čvora u G.

Mora postojati barem jedan put jer inače graf ne bi bio povezan.

Ako ima više veza tada postoji ciklus.



Algoritam za dani graf određuje dali postoji put između čvora j i k.

a) kada je graf dan u obliku liste susjedstva

Povezanost(G, j, k)

int dist[1...|V|]

int boja[1...|V|]

cvor pred[1...|V|]

queue Q = empty ; red za obradu čvorova

za svakog u iz |V|

dist[u] = INF ; na početku su sve distance INF

boja[u] = bijela ; boja bijela - neobrađeni su

pred[u] = NULL ; pokazivači na prethodnoga su NULL

dist[j] = 0 ; j uzimamo kao početni čvor

boja[j] = siva

enqueue(Q,j)

while(Q is nonempty)

u = dequeue(Q)

za svakog v iz Adj(u) ; obradujemo po susjedima

if(boja[v] == bijela) ; ako nije obrađen

boja[v] = siva

dist[v] = dist[u] + 1

pred[V] = u

enqueue(Q,v)

boja[u] = crna

if(dist[k] == INF) ; kada smo sve obradili trebala bi postojati distanca od k

print „put ne postoji“

else

print „put postoji“

Vrijeme izvršavanja O(|V| + |E|)

b) kada je graf dan u obliku matrice susjedstva

Povezanost(G, j, k)

int dist[1...|V|]

int boja[1...|V|]

cvor pred[1...|V|]

queue Q = empty ; red za obradu čvorova

za svakog u iz |V|

dist[u] = INF ; na početku su sve distance INF

boja[u] = bijela ; boja bijela - neobrađeni su

pred[u] = NULL ; pokazivači na prethodnoga su NULL

dist[j] = 0 ; j uzimamo kao početni čvor

boja[j] = siva

enqueue(Q,j)

while(Q is nonempty)

u = dequeue(Q)

for v = 1 to |V| do ; idemo po retku matrice

if(A[u,v] == 1) ; ako su susjedi

if(boja[v] == bijela) ; ako čvor još nije obrađen

boja[v] = siva

dist[v] = dist[u] + 1

pred[V] = u

enqueue(Q,V)

boja[u] = crna ; kada mu obradimo sve susjede

if(dist[k] == INF)

print „nema puta“

else

print „put postoji“

Vrijeme izvršavanja O(|V|2)

Za dva niza X i Y definiramo najkraću zajedničku supersekvencu kao niz

najkraće duljine takav da su i X i Y podsekvence od Z. Npr. ako su X   
=(A,B) i Y=(B,C) tada je Z=(A,B,C). Nađite algoritam koji će izračunati   
dužinu najkraće zajedničke supersekvence /dakle ne niz nego samo

dužinu). O(mn)

SCS(char X[1..m], char Y[1..n])

int C[0..m][0..n]

for i=0 to m do

C[i,0]=i

for j=0 to n do

C[0,j]=j

for i=1 to m

for j=1 to n

if (x[i]==y[j])

C[i,j]=C[i-1, j-1] +1

else if (C[i,j-1] < C[i-1,j])

C[i,j]=C[i,j-1]+1

else

C[i,j]=C[i-1,j]+1

return C[m,n]

Množenje lanca matrica

Vrijeme izvršavanja je Θ(n3)

NizMatrica(array p[i..n], int n)

array s[1..n-1, 2..n]

for i = 1 to n do m[i,i] = 0

for L = 2 to n do

for i = 1 to n-L+1 do

j = i + L - 1

m[i,j] = INFINITY

for k = i to j-1 do

q= m[i,k] + m[k+1,j] + p[i-1]\*p[k]\*p[j]

if (q < m[i,j])

m[i,j] = q

s[i,j] = k

return m[1,n]

return s

Mnozenje(i, j)

if(i<j)

k = s[i,j]

X = Mnozenje(i,k)

Y = Mnozenje(k+1,j)

return X\*Y

else

return A[i]

Dajte algoritam kojim se, za niz prirodnih brojeva, računa najduža rastuća podsekvenca

LIS (int A[1..n])

int B[1..n]

for i=1 to n

B[i]=A[i]

sortiraj B ; algortimom teta(nlogn)

LCS (A,B) ; algoritam teta (n^2)

Floyd-Warshall algoritam – rekurzivno

Dist(int i, int j, int k)

if k==0

return W[i,j]

return min(Dist(i,j,k-1), Dist(i,k,k-1)+Dist(k,j,k-1))

T(k, |V|) = 3T(k-1, |V|) +1

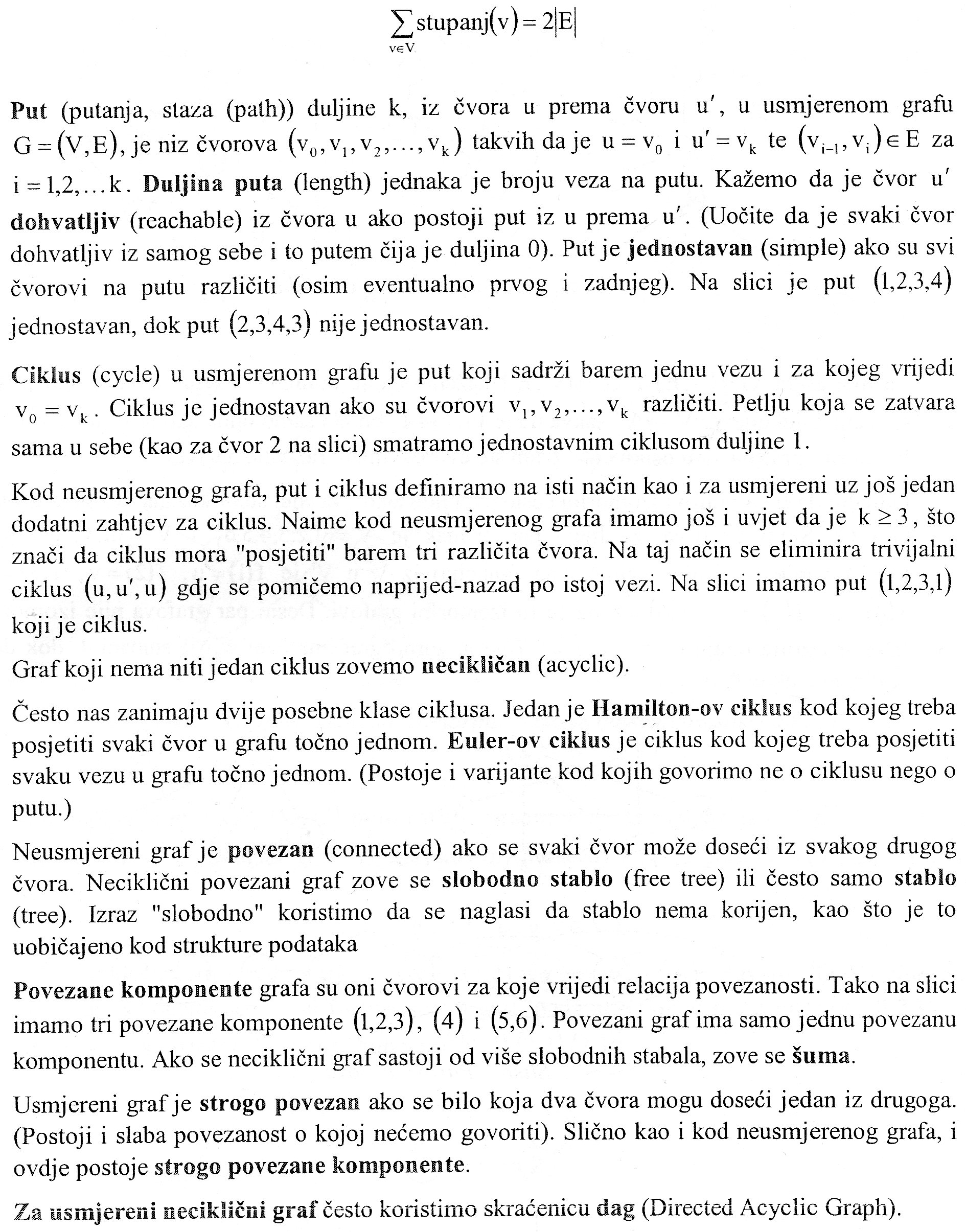
Ako postoje ciklusi sa negativnom tezinom, svakom iteracijom algoritma ce se smanjivati cijena ciklusa. U F-W algoritmo oni se očituju kroz negativne vrijednosti na dijagonali, a ne nula što bi značilo da je npr. udaljenost čvora 1 od samog sebe neka negativna vrijednost, a ne nula što je apsurdno.

Kod Floyd-Warshall-ovog algoritma koristili smo slijedeću rekurziju za računanje dij(k)



Međutim, u realizaciji koju smo dali na predavanju, nismo koristili referencu na (k) (dakle nismo koristili superskript (k) i (k-1) iz prethodnog izraza). Objasnite zašto je algoritam točan bez obzira što nismo koristili referencu na (k).

Odgovor na ovo pitanje leži u činjenici da za racunanje D (k) potrebni su k-ti redak i k-ti stupac iz D(k-1). Vrijednosti k-tog retka i k-tog stupca za D(k) ostaju nepromijenjene u odnosu na D (k-1), pa stoga nije potrebno koristiti referencu na k. (zatamnjeni redak i stupac se prepisu iz gornje tablice i ne mjenjaju se u trenutnoj).



Za dva grafa G=(V, E) i G'=(V', E') kažemo da su **izomorfna** ako postoji funkcija bijekcije f:V🡪V' takva da je (u, v) element od E onda i samo onda ako je (f(u), f(v)) element od E'. Izomorfni grafovi su u osnovi isti, osim što čvorovi imaju različite nazive.

Neusmjereni graf koji ima max broj veza naziva se **potpuni graf**.

Za dani graf G kažemo da je podskup čvorova V'⊆V formira **kliku** ako je podgraf uzorkovan s V' potpun. Podskup čvorova V' formira **neovisni skup** ako podgraf uzorkovan s V' nema veza.

**Dvodijeljan** graf je neusmjereni graf G=(V, E) kod kojeg se skup čvorova V može podijeliti na dva skupa V1 i V2 takva da (u, v)∈E implicira ili u∈V1 i v∈V2 ili u∈V2 i v∈V1. Drugim riječima, sve veze idu izmedu dva skupa V1 i V2.

**Komplement** grafa G=(V, E) je graf na istom skupu čvorova, ali kod kojeg su veze komplementi veza na grafu G. (kada od potpunog grafa oduzmemo veze grafa G).

**Inverz** usmjerenog grafa je graf na istom skupu čvorova, ali s vezama čiji je smjer invertiran.