nedeljainformatike v7.0

Kako znamo da ste varali na takmičenju

Elena Galjak

Matematička gimnazija

12. 04. 2022.

Agenda

1. Šta su plagijati?
2. Kako ih pronalazimo?
3. Možemo li ih prevariti?

Plagijati?

- **Plagijarizam** predstavlja neovlašćeno preuzimanje tuđih ideja, tekstova ili uopšteno rada i intelektualne svojine bez navođenja originalnog izvora.

- Tipovi plagijata:

  - Plagijati potpune identičnosti
  - Plagijati promenjenih identifikatora
  - Plagijati promenjenog redosleda
  - Plagijati promenjene sintakse

```
int NZD(int p,int q)
{
    int r;
    while (q != 0)
    { r = p % q;
      p = q;
      q = r;
    }

  return p;
}
```

```
int NZD(int a, int b)
{
    if (a == 0)
        return b;
    return NZD(b % a, a);
}
```

Problematike efektivnog algoritma

- Da bismo za neki algoritam, a kasnije i softver, mogli da kažemo da dobro detektuje sve vrste plagijata s prethodnog slajda, neophodna su mu sledeća 3 svojstva:

  (1) **Neosetljivost na beline** - Algoritam mora biti sposoban da ignoriše nepotrebne podatke i beznačajne sintakse: komentare, razmake, velika slova umesto malih, interpunkcijske znakove, promenu imena varijabli

  (2) **Suzbijanje "šumova"** - Svako podudaranje mora biti dovoljno veliko da implicira da je materijal kopiran i da nije samo uobičajena ili ključna reč u programskom jeziku na kom je dokument pisan.

  (3) **Nezavisnost segmenata** - Pozicija određenih segmenata u kodu ne treba da utiče na broj sličnosti koje detektujemo u dokumentu.

Agenda

1. Šta su plagijati?
2. Kako ih pronalazimo?
3. Možemo li ih prevariti?

Tehnike detekcije

- Tehnike tekstualne komparacije – Poređenje podataka se vrši na nivou sirovog teksta, uz eventualnu prethodnu normalizaciju teksta.
- Tehnike znakovne komparacije – Poređenje sledi nakon transformacije tekstualnog fajla u niz znakova
- Tehnike temeljene na stablima povezanosti – Unutar njih se program rastavlja na funkcionalne jedinice sve do najmanjih segmenata. Od navedenih elemenata se prave grafička stabla s čvorovima.
- Tehnike temeljene na graficima programskih zavisnosti – Ova metoda jeste naprednija verzija grafičkih stabala gde je moguće detaljnije poređenje nekih segmenata.
- Hibridne tehnike

Najpoznatiji softveri

Postoji pregršt različitih softvera koji pronalaze plagijate, ali ćemo mi pomenuti samo neke najpoznatije:

- *Moss*

  A System for Detecting Software Similarity

- I GPLAG (koji nema svoj logo :D)

MOSS

- Kreirao ga je Aleks Aiken 1994.
- Podržava 24 programska jezika.
- Nije open – source.

- Da bi izmerio sličnost između dokumenata, MOS kreira otiske prstiju dokumenata, a zatim te otiske upoređuje. Proces kreiranja otisaka možemo podeliti u sledeće faze:
  - Pretprocesiranje dokumenta u kome se uklanjaju nevažne karakteristike i dokument se sprema za obradu – (b)
  - Podela dokumenta na k-grame i heširanje – (v), (g)
  - Odabir podskupa hešova koji se koristi kao otisak prsta dokumenta (Winnowing algoritam) – (d), (đ)
  - Parovi dokumenata sa velikim brojem podudarajućih otisaka se označavaju za pregled.

MOSS

Miš uz pušku, miš niz pušku
a) Neki tekst

Mišuzpuškumišnizpušku
b) Tekst bez irelevantnih karakteristika

Mišuz išuzp šuzpu uzpuš zpušk pušku uškum
škumi kumiš umišn mišni išniz šnizp nizpu
izpuš zpušk pušku
v) Niz 5-grama generisanih iz teksta

74 58 60 52 19 30 24 20 35 93 68 7 92 41 80 19 30
g) Hipotetički niz heševa svih 5-grama

(74, 58, 60, 52) (58, 60, 52, 19)
(60, 52, 19, 30) (52, 19, 30, 24)
(19, 30, 24, 20) (30, 24, 20, 35)
(24, 20, 35, 93) (20, 35, 93, 68)
(35, 93, 68, 7) (93, 68, 7, 92)
(68, 7, 92, 41) (7, 92, 41, 80)
(92, 41, 80, 19) (41, 80, 19, 30)
d) Svi okviri heševa dužine 4

52 19 20 7 19
đ) Otisci prstiju odabrani algoritmom vejanja

Normalizacija i tokenizacija

- **Normalizacija-** eliminišu se karakteristike koje ne bi trebalo da razlikuju dokumente: semantički nepotrebne beline (razmake), komentari, deklarisanje paketa i import iskazi. Pored toga, uključuje i preimenovanje svih identifikatora istog tipa u istu vrednost. Kombinovane deklaracije promenljivih podeljuju se u niz pojedinačnih deklaracija.

Операција	Елемент	Пример елемента	Функционална замена
1	Текст	„Прекид“	„....“
2	Знак	's'	‘.’
3	Број	9	1
4	Децимални број	0.314159	1.0
5	Променљива	x, counter	var
6	Име функције	NZD()	func()

- **Tokenizacija-** proces konvertovanja izvornog koda u tokene.Tokeni ili lekseme se biraju na takav način da karakterišu suštinu programa, koju će teško izmeniti plagijator.

Име лексеме (токена)	Пример вредности лексеме
identifier	x,y,z,p,q,counter
keyword	if, while, return
separator	}, (, ;
operator	+, <, =
literal	true, 6.02e23, "music"

Heširanje, heš funkcije, k-gram

Kada smo od početnog dokumenta dobili tokene, sledeći korak jeste da te tokene podelimo u k-grame.

```
int hello = 5;
return hello;
```

```
TYP_INT ID EQ NUM SEMI
RET ID SEMI
```

Generisanje k-grama

```
(TYP_INT ID EQ), (ID EQ NUM), (EQ NUM SEMI),
(NUM SEMI RET), (RET ID SEMI)
```

Heširanje

```
30 15 56 83 71 10
```

Winnowing

- Kao što smo rekli, MOS radi po principu generisanja otisaka prstiju. Taj skup otisaka jeste zapravo neki podskup svih heševa koje smo dobili iz fajla.
- Ne biramo ceo skup jer bi to bilo veoma računski skupo, pogotovo sa velikim fajlovima.
- Na koji način biramo heševe za najveću efikasnost?

<u>Vejanje</u>

Neka je dat skup dokumenata. Želimo da nađena podudaranja između njih zadovoljavaju sledeća 2 svojstva:

1. Ako postoji neko podudaranje koje je jednako ili duže od garantnog praga **t** , onda će to podudaranje sigurno biti detektovano;
2. Sva podudaranja kraća od praga šuma **k** neće biti detektovana.

Prvi korak vejanja jeste grupisanje heševa u preklapajuće okvire dužine w, slično kao i kod k-grama, gde je w=t-k+1.

Winnowing

```
(30,15,56) (15,56,83) (56,83,71) (83,71,10)
```

- Budući da je veličina okvira definisana kao t-k+1, za svaki podniz dužine t (tj. dužine t tokena) postoji najmanje jedan okvir takav da bi svaki heš u njemu bio detektovan kao podudaranje, u slučaju da druga ulazna datoteka sadrži taj podniz.
- Kao rezultat toga, za smanjivanje skupa otisaka koje biramo, sledeći korak jeste odabir **po jednog heša iz svakog okvira**, koji će se nalaziti među otiscima prstiju.
- MOS bira **minimalnu heš vrednost u svakom okviru.** Ako je heš već odabran kao otisak u prethodnom okviru, tada neće biti ponovo izabran u sledećem. Ako se minimalna vrednost pojavi više puta u prozoru, MOS bira krajnju desnu.

```
(30,15,56) (15,56,83) (56,83,71) (83,71,10)
```

```
15  56  10
```

```
[15, 1], [56, 2], [10, 5]
```

GPLAG

- Podržava sve programske jezike
- Jeste open-source

- GPlag pronalazi plagijate upoređivanjem **PDG**-ova (**Program Dependence Graph**).
- Budući da ne dolazi do značajnijih promena u PDG prilikom plagiranja, ovakav način pretrage je izuzetno efikasan.
- Svaki PDG se sastoji od **čvorova**, koji predstavljaju naredbe programa, i **grana** koje predstavljaju podatke i kontrolne zavisnosti.

Glavna ideja jeste:

Neka originalni program P i program koji sumnjičimo za plagijarizaciju P' imaju po n i m naredbi, respektivno. I neka je program P predstavljen sa grafom G i G'.
Problem nalaženja podudarnosti onda delimo na 2:

-Ako imamo čvor g iz G i g' iz G', kako možemo da znamo da li je g' plagijat g?

-I kako ćemo efikasnije pretraživati grafove, umesto originalnih n*m parova?

PDG

- **Graf programske zavisnosti (PDG)** jeste grafička reprezentacija izvornog koda nekog programa.
- Osnovne naredbe, poput deklarisanja promenljivih, dodeljivanja vrednosti, poziva funkcija itd. su predstavljene pomoću čvorova u PDG-u.
- Svaki čvor ima samo jedan tip.
- Grane predstavljaju ili kontrolnu zavisnost ili zavisnost putem podataka.

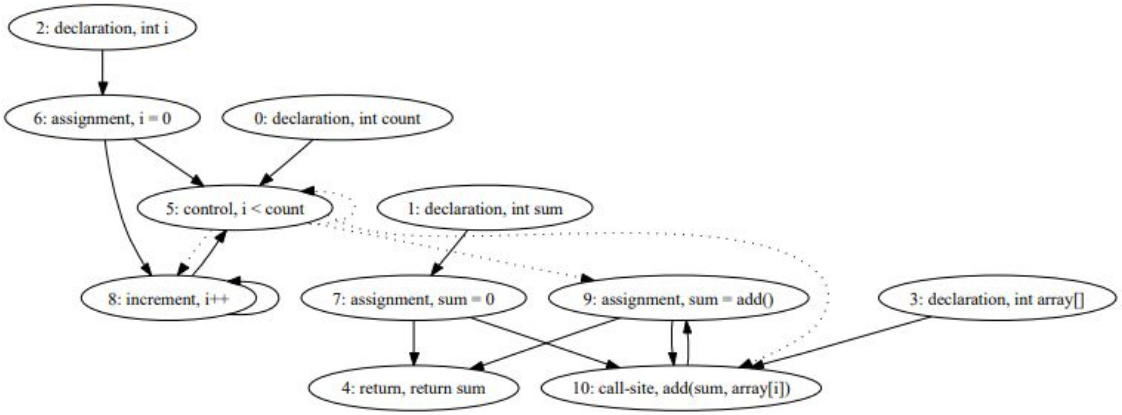
- **Kontrolna zavisnost** - Postoji grana kontrolne zavisnosti između "glavnog" čvora do sledećeg ako odlazak na drugi čvor zavisi od ispunjenosti uslova u prvom čvoru.

- **Zavisnost putem podataka** - Postoji grana zavisnosti od čvora V1 do čvora V2 ako postoji neka promenljiva var tako da:
  - V1 može biti postavljen na var, direktno ili pomoću pokazivača
  - V2 može da koristi vrednot var, direktno ili pomoću pokazivača
  - Postoji putanja izvršavanja programa od V1 do V2 na kojoj nema promene promenljive var

PDG

Type	Description
call-site	Call to procedures.
control	If, switch, while, do-while, or for.
declaration	Declaration for a variable or formal parameter.
assignment	Assignment expression.
increment	++ or −− expression
return	Function return expression.
expression	General expression except the above three, like one with ? operator
jump	Goto, break, or continue
label	Program labels
switch-case	Case or Default

Tabela vrednosti koje uzimaju čvorovi




```
int sum(int array[], int count)
{
  int i, sum;
  sum = 0;
  for(i = 0; i < count; i++){
    sum = add(sum, array[i]);
  }
  return sum;
}

int add(int a, int b)
{
  return a + b;
}
```

Primer

Još neke formalne definicije

- PDG - Programska zavisnost grafa G za program P jeste uredjena cetvorka G = (V, E, μ, δ) gde je:
  - V skup svih čvorova u P
  - $E \subseteq V \times V$ je skup svih grana
  - $\mu : V \rightarrow S$ funkcija koja dodeljuje tipove svim čvorovima
  - $\delta : E \rightarrow T$ funkcija koja dodeljuje tip granama, ili kontrolni ili od podataka

Problemu pretrage i poređenja grafova pristupamo pomoću **izomorfizma grafova, odnosno podgrafova**. Formalna definicija izomorfizma grafova bi bila:

- Bijektivna funkcija $f : V \rightarrow V'$ za $G = (V, E, \mu, \delta)$ i $G' = (V', E', \mu', \delta')$ tako da važi
  - $\mu(v) = \mu'(f(v))$
  - $\forall e = (v1, v2) \in E, \exists e' = (f(v1), f(v2)) \in E'$ such that $\delta(e) = \delta(e')$
  - $\forall e' = (v'1, v'2) \in E', \exists e = (f^{-1}(v'1), f^{-1}(v'2)) \in E$ such that $\delta(e') = \delta(e)$

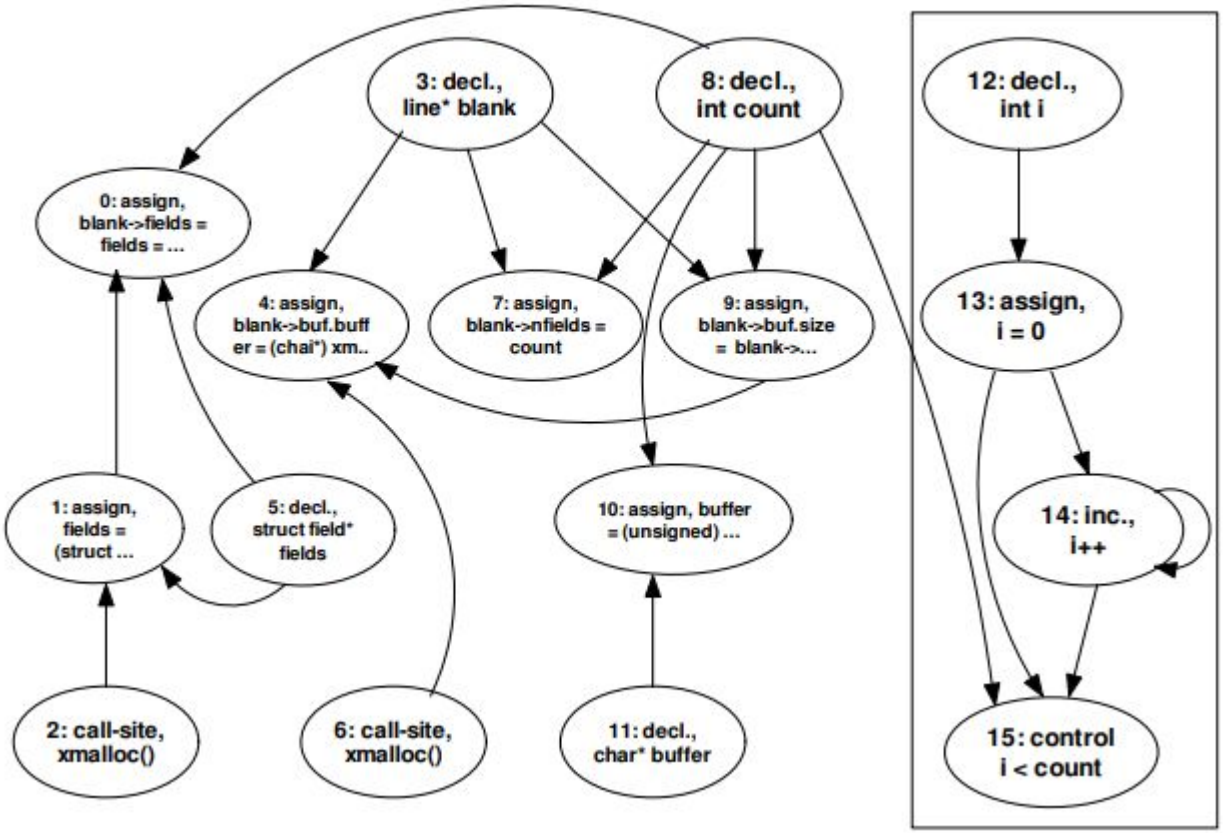
Ali u suštini **izomorfizam grafova** G i H jeste bijektivna funkcija f, koja slika G u H, tako da su neka 2 čvora susedna u G ako i samo ako su susedna u H.

Primer

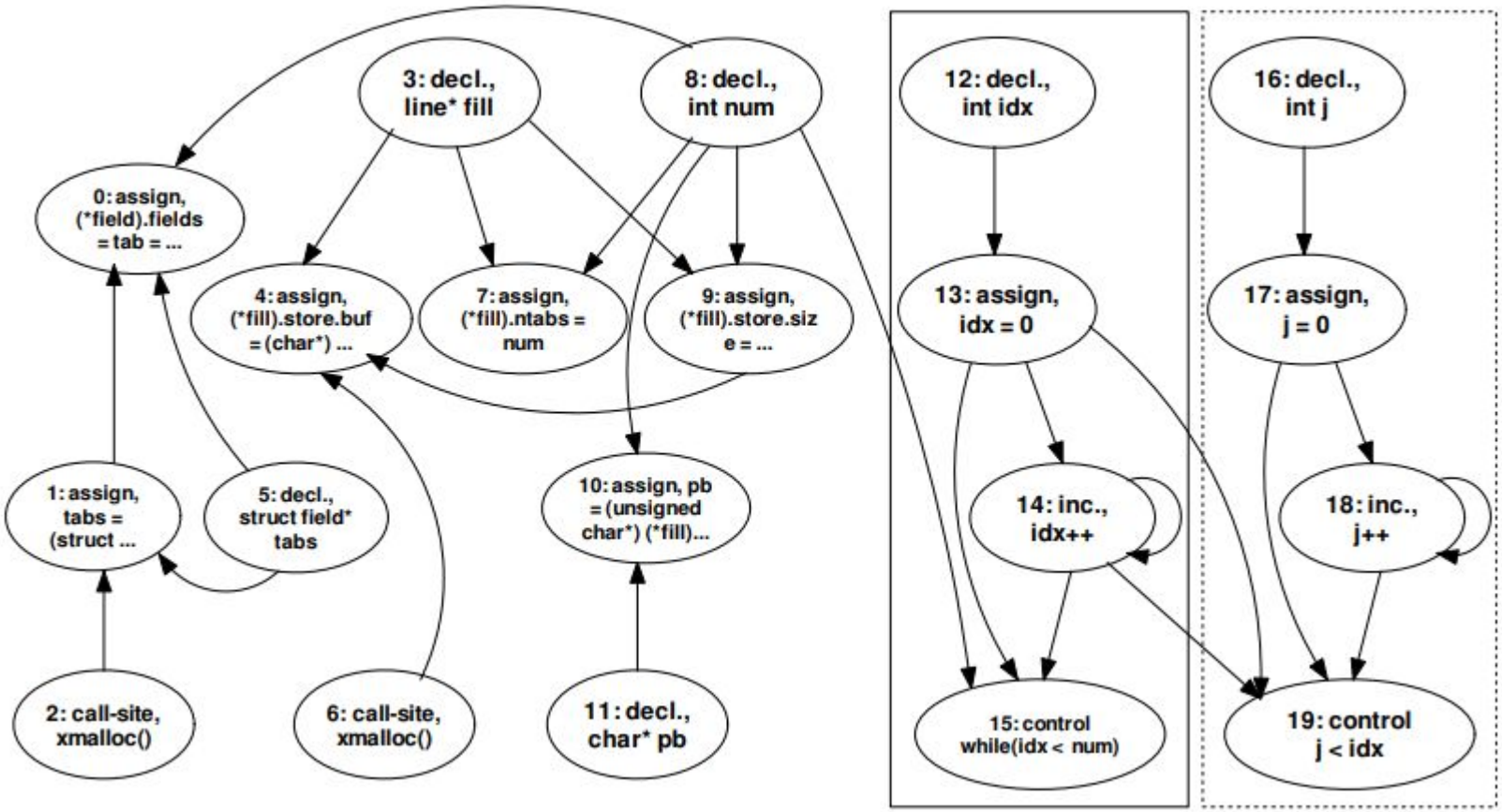
```
01  static void
02  make_blank (struct line *blank, int count)
03  {
04    int i;
05    unsigned char *buffer;
06    struct field *fields;

07    blank->nfields = count;
08    blank->buf.size = blank->buf.length = count + 1;
09    blank->buf.buffer = (char*) xmalloc (blank->buf.size);
10    buffer = (unsigned char *) blank->buf.buffer;
11    blank->fields = fields =
        (struct field *) xmalloc (sizeof (struct field) * count);

12    for (i = 0; i < count; i++){
13        ...
14    }
15  }
```

```
01 static void
02 fill_content(int num, struct line* fill)
03 {
04   (*fill).store.size = fill->store.length = num + 1;
05   struct field *tabs;
06   (*fill).fields = tabs = (struct field *)
                xmalloc (sizeof (struct field) * num);
07   (*fill).store.buffer = (char*) xmalloc (fill->store.size);
08   (*fill).ntabs = num;
09   unsigned char *pb;
10   pb = (unsigned char *) (*fill).store.buffer;

11   int idx = 0;
12   while(idx < num){ // fill in the storage
13     ...
14     for(int j = 0; j < idx; j++)
15       ...
16     idx++;
17   }
18 }
```

Prvi graf


3: decl., line* blank
8: decl., int count
12: decl., int i
0: assign, blank->fields = fields = ...
4: assign, blank->buf.buff er = (chai*) xm..
7: assign, blank->nfields = count
9: assign, blank->buf.size = blank->...
13: assign, i = 0
1: assign, fields = (struct ...
5: decl., struct field* fields
10: assign, buffer = (unsigned) ...
14: inc., i++
2: call-site, xmalloc()
6: call-site, xmalloc()
11: decl., char* buffer
15: control i < count

Drugi graf


3: decl., line* fill
8: decl., int num
12: decl., int idx
16: decl., int j
0: assign, (*field).fields = tab = ...
4: assign, (*fill).store.buf = (char*) ...
7: assign, (*fill).ntabs = num
9: assign, (*fill).store.siz e = ...
13: assign, idx = 0
17: assign, j = 0
1: assign, tabs = (struct ...
5: decl., struct field* tabs
10: assign, pb = (unsigned char*) (*fill)...
14: inc., idx++
18: inc., j++
2: call-site, xmalloc()
6: call-site, xmalloc()
11: decl., char* pb
15: control while(idx < num)
19: control j < idx

Pseudo kod

Algorithm 1 GPLAG($\mathcal{P}$, $\mathcal{P}'$, K, γ, α)

Input: $\mathcal{P}$: The original program
$\mathcal{P}'$: A plagiarism suspect
K: Minimum size of nontrivial PDGs, default 10
γ: Mature rate in isomorphism testing, default 0.9
α: Significance level in lossy filter, default 0.05
Output: $\mathcal{F}$: PDG pairs regarded to involve plagiarism

1: $\mathcal{G}$ = The set of PDGs from $\mathcal{P}$
2: $\mathcal{G}'$ = The set of PDGs from $\mathcal{P}'$
3: $\mathcal{G}_K = \{g | g \in \mathcal{G} \text{ and } |g| > K\}$
4: $\mathcal{G}'_K = \{g' | g' \in \mathcal{G} \text{ and } |g'| > K\}$
5: **for each** $g \in \mathcal{G}_K$
6: **let** $\mathcal{G}'_{K,g} = \{g' | g' \in \mathcal{G}'_K, |g'| \geq \gamma|g|, (g, g') \text{ passes filter}\}$
7: **for each** $g' \in \mathcal{G}'_{K,g}$
8: **if** g is γ-isomorphic to g'
9: $\mathcal{F} = \mathcal{F} \cup (g, g')$
10: **return** $\mathcal{F}$;

Agenda

1. Šta su plagijati?
2. Kako ih pronalazimo?
3. Možemo li ih prevariti?

Možemo li ih prevariti?

Odgovor je: **DA**

Hvala na pažnji!

Pitanja?