Programiranje 2, letnji semestar 2023/4

Naredbe upravljanja tokom programa

Stefan Nikolić

Prirodno-matematički fakultet, Novi Sad
11.03.2024.

Na prošlom predavanju smo počeli da rešavamo sledeći problem

Neka je n proizvoljan broj iz $\mathbb{N}$

Neka su a i b nezavisno i uniformno odabrani iz $[1, n]$

Neka je p_n verovatnoća da su a i b uzajamno prosti

Da li postoji granična vrednost $\lim_{n\to\infty} p_n$ i ako postoji, koja joj je vrednost?

Ovo nam je bio osnovni pristup

1. Izlistamo sve parove brojeva $\{a, b\} \in [1, n]^2$
2. Prebrojimo koliko ih je uzajamno prostih
3. Podelimo taj broj sa n^2, da dobijemo p_n
4. Proverimo kako se p_n menja sa uvećanjem n i postavljamo hipotezu koju posle dokazujemo

I ustanovili smo da su nam sledeća znanja potrebna da bismo ga sproveli u delo

- Moramo nekako da predstavimo brojeve
- Moramo da proverimo da li su uzajamno prosti
  - Jedan način: proveravamo da li je NZD(a, b) > 1
  - Na primer, pitamo se da li $(\exists i > 1)(i \mid a \wedge i \mid b)$
- Moramo da prebrojimo sve koji jesu
- Moramo da nađemo odnos tog broja i n^2

Vratimo se na kratko na početak

1. Izlistamo sve parove brojeva $\{a, b\} \in [1, n]^2$
2. Prebrojimo koliko ih je uzajamno prostih
3. Podelimo taj broj sa n^2, da dobijemo p_n
4. Proverimo kako se p_n menja sa uvećanjem n i postavljamo hipotezu koju posle dokazujemo

Ove korake ponavljamo za različite vrednosti n

1. Izlistamo sve parove brojeva $\{a, b\} \in [1, n]^2$
2. Prebrojimo koliko ih je uzajamno prostih
3. Podelimo taj broj sa n^2, da dobijemo p_n
4. Proverimo kako se p_n menja sa uvećanjem n i postavljamo hipotezu koju posle dokazujemo

Do sad smo n učitavali sa tastature

n	p_n
100	0.6087
500	0.608924
1000	0.608383
5000	0.608037
10000	0.60795
50000	0.607939

No, u knjizi je vrednost $p(n)$ proverena za ove vrednosti n

Do sad smo n učitavali sa tastature

n	p_n
100	0.6087
500	0.608924
1000	0.608383
5000	0.608037
10000	0.60795
50000	0.607939

Uočavamo li neke pravilnosti u ovom nizu?

100, 500, 1000, 5000, 10000, 50000

$$n_1 = 100 \tag{1}$$

$$n_{i+1} = \begin{cases} 5 \times n_i, & \text{ako } 2 \nmid i \\ 2 \times n_i, & \text{u suprotnom} \end{cases}, \qquad \forall i > 1 \tag{2}$$

100, 500, 1000, 5000, 10000, 50000

$$n_1 = 100 \tag{1}$$

$$n_{i+1} = \begin{cases} 5 \times n_i, & \text{ako } 2 \nmid i \\ 2 \times n_i, & \text{u suprotnom} \end{cases}, \qquad \forall i > 1 \tag{2}$$

Da bi smo uočili isto što i autor knjige, nije potrebno da unosimo n sa tastature. Dovoljno je da znamo da izbrojimo od 1 do 6.

Šta nam je potrebno za brojanje od 1 do 6?

Šta nam je potrebno za brojanje od 1 do 6?

Petlje

Petlje

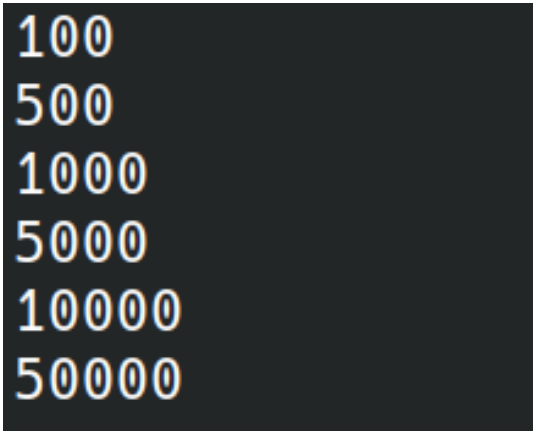
For petlja

```
for(inicijalizacija; uslov petlje; finalizacija)
        telo petlje
```

Naš primer

```
#include <iostream>
using namespace std;

int main()
{
        unsigned n = 100;
        for(unsigned i = 1; i <= 6; ++i)
        {
                cout << n << endl;

                if(i % 2)
                        n *= 5;
                else
                        n *= 2;
        }

        return 0;
}
```

Naš primer

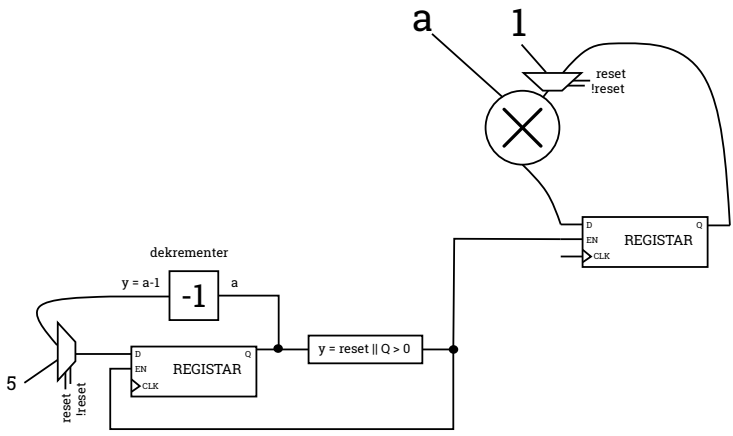


Svrha For petlje

For petlje se najčešće upotrebljavaju za implementaciju brojačkih petlji:

Kada je potrebno da određeni skup naredbi ponovimo tačno određen broj puta, pri čemu taj broj pratimo pomoću takozvanog *brojača petlje*

Setimo se stepenovanja



Svrha For petlje

Napomena: for petlje možemo koristiti i u druge svrhe, kao što ćemo uskoro videti, ali su u tim slučajevima drugi tipovi petlji najčešće prikladniji

Inicijalizacija

for(inicijalizacija; uslov petlje; finalizacija)
    telo petlje

Naredba koja se izvršava samo jednom, pre prvog izvršenja tela petlje

Obično služi za postavljanje brojača petlje na početnu vrednost

Ukoliko brojač nije ranije deklarisan, može sadržati i deklaraciju

Inicijalizacija sa deklaracijom brojača petlje

```
#include <iostream>
using namespace std;

int main()
{
        unsigned n = 100;
        for(unsigned i = 1; i <= 6; ++i)
        {
                cout << n << endl;

                if(i % 2)
                        n *= 5;
                else
                        n *= 2;
        }

        return 0;
}
```

Inicijalizacija bez deklaracije

```
#include <iostream>
using namespace std;

int main()
{
        unsigned n = 100;
        unsigned i = 7;

        cout << "Pre pocetka petlje, i = " << i << endl;
        for(i = 1; i <= 6; ++i)
        {
                cout << n << endl;

                if(i % 2)
                        n *= 5;
                else
                        n *= 2;
        }

        return 0;
}
```

Inicijalizacija bez deklaracije

```
Pre pocetka petlje, i = 7
100
500
1000
5000
10000
50000
```

Naravno, brojač mora negde biti deklarisan pre upotrebe

```
#include <iostream>
using namespace std;

int main()
{
        unsigned n = 100;

        for(i = 1; i <= 6; ++i)
        {
                cout << n << endl;

                if(i % 2)
                        n *= 5;
                else
                        n *= 2;
        }

        return 0;
}
```

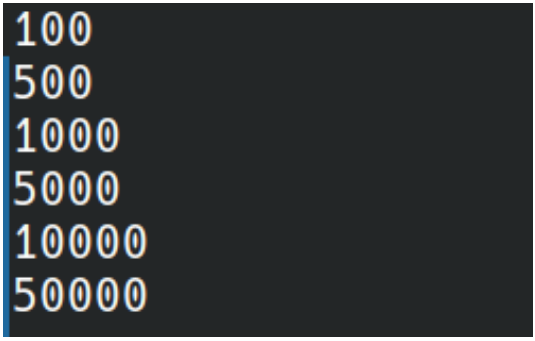
Naravno, brojač mora negde biti deklarisan pre upotrebe

```
six_count_for.cpp: In function ‘int main()’:
six_count_for.cpp:8:13: error: ‘i’ was not declared in this scope
    8 |         for(i = 1; i <= 6; ++i)
      |             ^

shell returned 1
```

Inicijalizacija nije obavezna: kada je nemamo, koristimo samo ;

```
#include <iostream>
using namespace std;

int main()
{
        unsigned n = 100;
        unsigned i = 1;

        for(; i <= 6; ++i)
        {
                cout << n << endl;

                if(i % 2)
                        n *= 5;
                else
                        n *= 2;
        }

        return 0;
}
```

Inicijalizacija nije obavezna: kada je nemamo, koristimo samo ;



Ali se gotovo uvek preporučuje

```cpp
#include <iostream>
using namespace std;

int main()
{
        unsigned n = 100;
        unsigned i = 5;
        //Zamislite da ispod ove linije postoji jos 200, pre petlje

        for(; i <= 6; ++i)
        {
                cout << n << endl;

                if(i % 2)
                        n *= 5;
                else
                        n *= 2;
        }

        return 0;
}
```

U suprotnom, može se dogoditi da ne znamo koja je početna vrednost brojača petlje

Ali se gotovo uvek preporučuje

U suprotnom, može se dogoditi da ne znamo koja je početna vrednost brojača petlje

Čak i u ovom slučaju

```
#include <iostream>
using namespace std;

int main()
{
        unsigned i;
        cout << "Unesite pocetnu vrednost:";
        cin >> i;

        cout << "Prirodni brojevi <= " << i << " u opadajucem poretku:\n";
        for(; i > 0; --i)
                cout << i << endl;

        return 0;
}
```

Ovo je ispravno

Čak i u ovom slučaju

```
Unesite pocetnu vrednost:10
Prirodni brojevi <= 10 u opadajucem poretku:
10
9
8
7
6
5
4
3
2
1
```

Ovo je ispravno

Čak i u ovom slučaju

```cpp
#include <iostream>
using namespace std;

int main()
{
        unsigned n;
        cout << "Unesite pocetnu vrednost:";
        cin >> n;

        cout << "Prirodni brojevi <= " << n << " u opadajucem poretku:\n";
        for(unsigned i = n; i > 0; --i)
                cout << i << endl;

        return 0;
}
```

Ali je ovo daleko jasnije[1] i lakše za održavanje

[1] Jasno je da smo želeli da i počne od vrednosti n a da to nije samo rezultat zaborava. Ponovo nije loše zamisliti daleko složeniji program.

Čak i u ovom slučaju

```
Unesite pocetnu vrednost:10
Prirodni brojevi <= 10 u opadajucem poretku:
10
9
8
7
6
5
4
3
2
1
```

Ali je ovo daleko jasnije[1] i lakše za održavanje

[1] Jasno je da smo želeli da i počne od vrednosti n a da to nije samo rezultat zaborava. Ponovo nije loše zamisliti daleko složeniji program.

Čak i u ovom slučaju

```
#include <iostream>
using namespace std;

int main()
{
        unsigned n;
        cout << "Unesite pocetnu vrednost:";
        cin >> n;

        cout << "Prirodni brojevi <= " << n << " u opadajucem poretku:\n";
        for(unsigned i = n; i > 0; --i)
                cout << i << endl;

        cout << "Prirodni brojevi <= " << n << " u rastucem poretku:\n";
        for(unsigned i = 1; i <= n; ++i)
                cout << i << endl;

        return 0;
}
```

Uz to još i ne uništavamo vrednost učitanu sa tastature, koja može kasnije da nam zatreba

Čak i u ovom slučaju

```
Prirodni brojevi <= 10 u opadajucem poretku:
10
9
8
7
6
5
4
3
2
1
Prirodni brojevi <= 10 u rastucem poretku:
1
2
3
4
5
6
7
8
9
10
```

Uz to još i ne uništavamo vrednost učitanu sa tastature, koja može kasnije da nam zatreba

Uslov petlje

for(inicijalizacija; uslov petlje; finalizacija)
    telo petlje

Izraz koji čija je vrednost **TAČNA** ili **NETAČNA**

Dok je tačna, pristupa se izvršenju tela petlje

Čim postane netačna, telo petlje se preskače i program prelazi na sledeću naredbu nakon petlje

Uslovi

Šta je tačno a šta netačno?

```cpp
#include <iostream>
using namespace std;

int main()
{
        bool a = true;
        bool b = false;

        cout << "a = " << a << ", b = " << b << endl;

        return 0;
}
```

Šta je tačno a šta netačno?

```
a = 1, b = 0
```

bool tipovi su zapravo brojevi iz $\mathbb{Z}_2$

Zapravo, vrednost svakog izraza koju je moguće konvertovati u broj se smatra netačnom ako je jednaka nuli i tačnom u svim drugim slučajevima

Izrazi uslova se svode na broj

```
#include <iostream>
#define PI 3.14159
using namespace std;

int main()
{

        int r = 0;
        cout << "r = " << r << endl;
        if(2 * r * PI)
                cout << "Obim kruga je " << 2 * r * PI << endl;
        else
                cout << "Krug ne postoji.\n";

        r = 17;
        cout << "r = " << r << endl;
        if(2 * r * PI)
                cout << "Obim kruga je " << 2 * r * PI << endl;
        else
                cout << "Krug ne postoji.\n";

        r = -23;
        cout << "r = " << r << endl;
        if(2 * r * PI)
                cout << "Obim kruga je " << 2 * r * PI << endl;
        else
                cout << "Krug ne postoji.\n";

        return 0;
}
```

Izrazi uslova se svode na broj

```
r = 0
Krug ne postoji.
r = 17
Obim kruga je 106.814
r = -23
Obim kruga je -144.513
```

Relacioni operatori

<	manje
<=	manje ili jednako
==	jednako
>=	veće ili jednako
!=	različito

Relacioni operatori

```cpp
#include <iostream>
using namespace std;

int main()
{
        int a = 4;
        char b = -17;
        long c = -17;

        cout << "a = " << a << ", b = " << +b << ", c = " << c << endl;
        cout << "a == b? " << (a == b) << endl;
        cout << "a != b? " << (a != b) << endl;
        cout << "a > b? " << (a > b) << endl;
        cout << "a >= b? " << (a >= b) << endl;
        cout << "a < b? " << (a < b) << endl;
        cout << "a <= b? " << (a <= b) << endl;

        cout << "b == c? " << (b == c) << endl;

        return 0;
}
```

Relacioni operatori

```
a = 4, b = -17, c = -17
a == b? 0
a != b? 1
a > b? 1
a >= b? 1
a < b? 0
a <= b? 0
b == c? 1
```

Ne zaboravimo na probleme sa poređenjem brojeva sa pokretnom tačkom

```cpp
#include <iostream>
using namespace std;

int main()
{
        float a = 1e-2;
        float b = 1e-4;
        float c = 1e-5;

        if((a + b + c) == (c + b + a))
                cout << "Prekini trgovanje, gubitak je dosegao kriticnu tacku!.\n";
        else
                cout << "Sve je u redu, nastavi da trgujes.\n";

        return 0;
}
```

Ne zaboravimo na probleme sa poređenjem brojeva sa pokretnom tačkom

```cpp
#include <iostream>
#define TOL 1e-9
using namespace std;

int main()
{
        float a = 1e-2;
        float b = 1e-4;
        float c = 1e-5;

        if(abs((a + b + c) - (c + b + a)) < TOL)
                cout << "Prekini trgovanje, gubitak je dosegao kriticnu tacku!\n";
        else
                cout << "Sve je u redu, nastavi da trgujes.\n";

        return 0;
}
```

Ne zaboravimo ni to da je dodela izraz koji može biti uslov

```cpp
#include <iostream>
using namespace std;

int main()
{
        unsigned a = 0;
        unsigned b = 0;

        cout << "a pre provere = " << a << endl;
        if(a = b)
                cout << "a = b\n";
        else
                cout << "a != b\n";
        cout << "a posle provere = " << a << endl;

        a = 7;
        cout << "\na pre provere = " << a << endl;
        if(a = b)
                cout << "a = b\n";
        else
                cout << "a != b\n";
        cout << "a posle provere = " << a << endl;

        return 0;
}
```

Ne zaboravimo ni to da je dodela izraz koji može biti uslov

```
a pre provere = 0
a != b
a posle provere = 0

a pre provere = 7
a != b
a posle provere = 0
```

Kako kombinujemo uslovne izraze?

Logički operatori

!	ne
||	ili
&&	i

Logički operatori

```cpp
#include <iostream>
using namespace std;

int main()
{
        unsigned a = 7;
        unsigned b = 4;
        unsigned c = 11;
        unsigned d = 12;

        if((a > b) && (c > d))
                cout << "a je vece od b i c je vece od d\n";
        else if((a > b) && (c < d))
                cout << "a je vece od b ali je c manje od d\n";

        if((c == d) || ((c + 1) == d))
                cout << "d je u intervalu [c, c + 1]\n";

        if(a)
                cout << "a je tacno\n";
        else
                cout << "a je netacno\n";
        if(!a)
                cout << "a je netacno\n";
        else
                cout << "a je tacno\n";

        return 0;
}
```

Logički operatori

```
a je vece od b ali je c manje od d
d je u intervalu [c, c + 1]
a je tacno
a je tacno
```

Ne zaboravimo sledeće aksiome Bulovih algebri

$0 \cdot a = 0$

$1 + a = 1$

Moramo voditi računa o bočnim efektima

```
#include <iostream>
using namespace std;

int main()
{
        unsigned n = 100;
        unsigned m = 1000;

        cout << "1) n = " << n << endl;
        if(m || n++)
                cout << "n + 1 ili m su pozitivni\n";
        cout << "2) n = " << n << endl;

        m = 0;

        if(m && --n)
                cout << "m i n - 1 su pozitivni\n";
        cout << "3) n = " << n << endl;

        return 0;
}
```

Moramo voditi računa o bočnim efektima

```
1) n = 100
n + 1 ili m su pozitivni
2) n = 100
3) n = 100
```

Čim vrednost logičkog izraza postane poznata, instrukcije koje evaluiraju njegov preostali deo se preskaču

Ternarni operator

$$\begin{cases} 5 \times n_i, & \text{ako } 2 \nmid i \\ 2 \times n_i, & \text{u suprotnom} \end{cases} \tag{3}$$

Ponekad imamo potrebu za predstavljanjem ovakvih uslovnih izraza

Ternarni operator

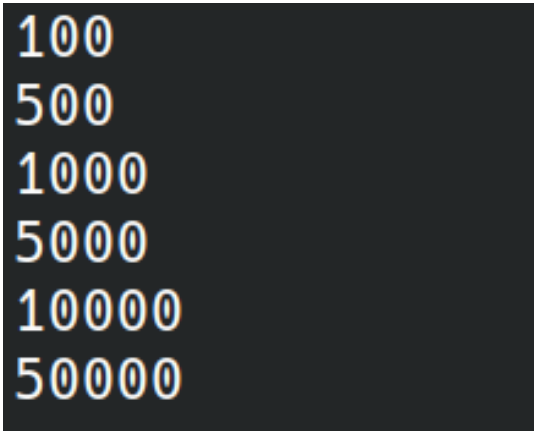
Za to možemo upotrebiti ternarni operator

uslov ? vrednost u slučaju tačnog uslova : vrednost u slučaju netačnog uslova

Vratimo se našem primeru

```
#include <iostream>
using namespace std;

int main()
{
        unsigned n = 100;
        for(unsigned i = 1; i <= 6; ++i)
        {
                cout << n << endl;

                if(i % 2)
                        n *= 5;
                else
                        n *= 2;
        }

        return 0;
}
```

Vratimo se našem primeru

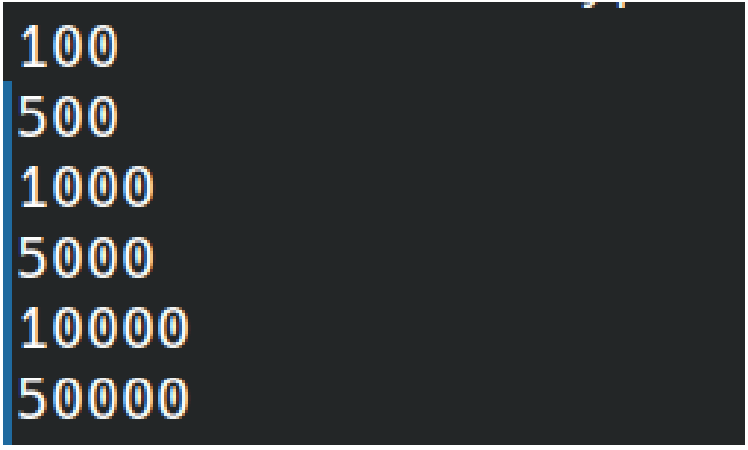


Vratimo se našem primeru

```cpp
#include <iostream>
using namespace std;

int main()
{
        unsigned n = 100;
        for(unsigned i = 1; i <= 6; ++i)
        {
                cout << n << endl;

                if(i % 2)
                        n *= 5;
                else
                        n *= 2;
        }

        return 0;
}
```

```cpp
#include <iostream>
using namespace std;

int main()
{
        unsigned n = 100;
        for(unsigned i = 1; i <= 6; ++i)
        {
                cout << n << endl;
                n *= i % 2 ? 5 : 2;
        }

        return 0;
}
```

Vratimo se našem primeru



Uslov u for petlji nije obavezan

Ukoliko ga izostavimo i napišemo samo ; umesto njega, dobićemo beskonačnu petlju

Finalizacija

for(inicijalizacija; uslov petlje; finalizacija)
    telo petlje

Naredba koja se izvršava nakon svakog prolaska kroz telo petlje

Obično služi za promenu brojača petlje

U našem primeru

```
#include <iostream>
using namespace std;

int main()
{
        unsigned n = 100;
        for(unsigned i = 1; i <= 6; ++i)
        {
                cout << n << endl;
                n *= i % 2 ? 5 : 2;
        }

        return 0;
}
```

Primetimo da kod finalizacije nema terminatora

```cpp
#include <iostream>
using namespace std;

int main()
{
        unsigned n = 100;
        for(unsigned i = 1; i <= 6; ++i;)
        {
                cout << n << endl;
                n *= i % 2 ? 5 : 2;
        }

        return 0;
}
```

Primetimo da kod finalizacije nema terminatora

```
six_count_ternary.cpp: In function ‘int main()’:
six_count_ternary.cpp:7:40: error: expected ‘)’ before ‘;’ token
    7 |         for(unsigned i = 1; i <= 6; ++i;)
      |            ~                           ^
      |                                        )
six_count_ternary.cpp:7:41: error: expected primary-expression before ‘)’ token
    7 |         for(unsigned i = 1; i <= 6; ++i;)
      |                                         ^

shell returned 1
```

Finalizacija nije obavezna

```cpp
#include <iostream>
using namespace std;

int main()
{
        unsigned n = 100;
        for(unsigned i = 1; i <= 6;)
        {
                cout << n << endl;
                n *= i % 2 ? 5 : 2;

                ++i;
        }

        return 0;
}
```

Finalizacija nije obavezna

Napomena

```
#include <iostream>
using namespace std;

int main()
{
        unsigned n = 100;
        for(unsigned i = 1; i <= 6; ++i)
        {
                cout << n << endl;
                n *= i % 2 ? 5 : 2;

                i *= 2;
        }

        return 0;
}
```

Napomena

Brojač petlje je promenljiva vidljiva u čitvom opsegu petlje, što znači da njome možemo manipulisati unutar tela petlje. To je vrlo retko korisno i potrebno je biti veoma pažljiv. Najčešće je bolje naprosto izbeći upotrebu te mogućnosti.

Telo petlje

for(inicijalizacija; uslov petlje; finalizacija)
    telo petlje

Naredba ili blok naredbi koje se ponavljaju sve dok je uslov petlje zadovoljen

U našem primeru

```
#include <iostream>
using namespace std;

int main()
{
        unsigned n = 100;
        for(unsigned i = 1; i <= 6; ++i)
        {
                cout << n << endl;
                n *= i % 2 ? 5 : 2;
        }

        return 0;
}
```

Primetimo da je moguće da uslov petlje nikada ne bude zadovoljen

```
#include <iostream>
using namespace std;

int main()
{
        unsigned n = 100;
        for(unsigned i = n; i <= 6; ++i)
        {
                cout << n << endl;
                n *= i % 2 ? 5 : 2;
        }
        cout << "Nakon petlje\n";

        return 0;
}
```

Primetimo da je moguće da uslov petlje nikada ne bude zadovoljen

```
Nakon petlje
```

Stavljanje terminatora iza zaglavlja petlje menja telo praznom naredbom

```
#include <iostream>
using namespace std;

int main()
{
        unsigned n = 100;
        for(unsigned i = 1; i <= 6; ++i);
        {
                cout << n << endl;
                n *= i % 2 ? 5 : 2;
        }

        return 0;
}
```

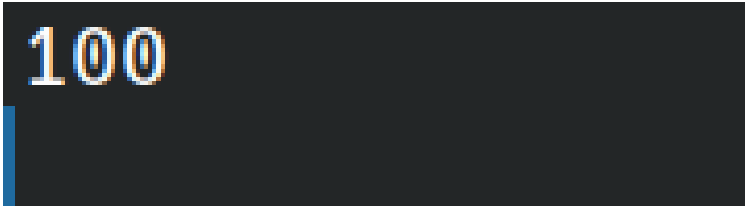
Stavljanje terminatora iza zaglavlja petlje menja telo praznom naredbom

```
six_count_ternary.cpp: In function ‘int main()’:
six_count_ternary.cpp:10:22: error: ‘i’ was not declared in this scope
   10 |                 n *= i % 2 ? 5 : 2;
      |                      ^

shell returned 1
```

Stavljanje terminatora iza zaglavlja petlje menja telo praznom naredbom

```
#include <iostream>
using namespace std;

int main()
{
        unsigned n = 100;
        unsigned i;
        for(i = 1; i <= 6; ++i);
        {
                cout << n << endl;
                n *= i % 2 ? 5 : 2;
        }

        return 0;
}
```

Stavljanje terminatora iza zaglavlja petlje menja telo praznom naredbom



Bezuslovni skokovi

Prekidanje petlje: naredba BREAK

```
#include <iostream>
using namespace std;

int main()
{
        unsigned n = 100;
        unsigned i;

        cout << n << endl;
        for(i = 1; i <= 6; ++i)
        {
                cout << "i = " << i << endl;
                n *= i % 2 ? 5 : 2;

                if(n > 3000)
                        break;

                cout << n << endl;
        }

        cout << "Prva naredba nakon petlje.\n"; //break bezuslovno skace ovde

        return 0;
}
```

Prekidanje petlje: naredba BREAK

```
100
i = 1
500
i = 2
1000
i = 3
Prva naredba nakon petlje.
```

Preskakanje ostatka iteracije: naredba CONTINUE

```cpp
#include <iostream>
using namespace std;

int main()
{
        unsigned n = 100;
        unsigned i;

        cout << n << endl;
        for(i = 1; i <= 6; ++i)
        {
                cout << "i = " << i << endl;
                n *= i % 2 ? 5 : 2;

                if(n / 1000 == 5)
                        continue;

                cout << n << endl;
                cout << "Telo petlje je zavrseno\n";

                //continue bezuslovno skace ovde
        }

        return 0;
}
```

Preskakanje ostatka iteracije: naredba CONTINUE

```
100
i = 1
500
Telo petlje je zavrseno
i = 2
1000
Telo petlje je zavrseno
i = 3
i = 4
10000
Telo petlje je zavrseno
i = 5
50000
Telo petlje je zavrseno
i = 6
100000
Telo petlje je zavrseno
```

Vratimo se na početni problem

Ceo program

```cpp
#include <iostream>
using namespace std;

int main()
{
	unsigned n = 100;
	for(unsigned i = 1; i <= 6; ++i)
	{
		unsigned uzajamno_prostih = 0;
		for(unsigned a = 1; a <= n; ++a)
		{
			for(unsigned b = 1; b <= n; ++b)
			{
				unsigned nzd = 1;
				for(unsigned d = 1; d <= a; ++d)
				{
					if(!(a % d) && !(b % d))
						nzd = d;
				}
				if (nzd == 1)
				{
					uzajamno_prostih++;
				}
			}
		}
		cout << "p(" << n << ") = " <<
			double(uzajamno_prostih) / (n * n) << endl;

		n *= i % 2 ? 5 : 2;
	}

	return 0;
}
```

Ceo program

```
p(100) = 0.6087
p(500) = 0.608924
p(1000) = 0.608383

real    0m1.780s
user    0m1.780s
sys     0m0.000s
```

Ceo program

```cpp
#include <iostream>
using namespace std;

int main()
{
        unsigned n = 100;
        for(unsigned i = 1; i <= 6; ++i)
        {
                unsigned uzajamno_prostih = 0;
                for(unsigned a = 1; a <= n; ++a)
                {
                        for(unsigned b = 1; b <= n; ++b)
                        {
                                unsigned nzd = 1;
                                for(unsigned d = 1; d <= a; ++d)
                                {
                                        if(!(a % d) && !(b % d))
                                                nzd = d;
                                }
                                if (nzd == 1)
                                {
                                        uzajamno_prostih++;
                                }
                        }
                }
                cout << "p(" << n << ") = " <<
                        double(uzajamno_prostih) / (n * n) << endl;

                n *= i % 2 ? 5 : 2;
        }

        return 0;
}
```

Da li možemo nešto da poboljšamo?

Druga verzija

```cpp
#include <iostream>
using namespace std;

int main()
{
	unsigned n = 100;
	for(unsigned i = 1; i <= 6; ++i)
	{
		unsigned uzajamno_prostih = 0;
		for(unsigned a = 1; a <= n; ++a)
		{
			for(unsigned b = 1; b <= n; ++b)
			{
				unsigned nzd = 1;
				for(unsigned d = 2; d <= a; ++d)
				{
					if(!(a % d) && !(b % d))
					{
						nzd = d;
						//Posto nam je bitno samo da
						//nadjemo neki delilac > 1,
						//mozemo da prekinemo petlju
						//cim se to desi.

						break;
					}
				}
				if (nzd == 1)
				{
					uzajamno_prostih++;
				}
```

Druga verzija

```
p(100) = 0.6087
p(500) = 0.608924
p(1000) = 0.608383

real    0m1.207s
user    0m1.205s
sys     0m0.000s
```

Druga verzija

```cpp
#include <iostream>
using namespace std;

int main()
{
	unsigned n = 100;
	for(unsigned i = 1; i <= 6; ++i)
	{
		unsigned uzajamno_prostih = 0;
		for(unsigned a = 1; a <= n; ++a)
		{
			for(unsigned b = 1; b <= n; ++b)
			{
				unsigned nzd = 1;
				for(unsigned d = 2; d <= a; ++d)
				{
					if(!(a % d) && !(b % d))
					{
						nzd = d;
						//Posto nam je bitno samo da
						//nadjemo neki delilac > 1,
						//mozemo da prekinemo petlju
						//cim se to desi.

						break;
					}
				}
				if (nzd == 1)
				{
					uzajamno_prostih++;
				}
```

Da li možemo još nešto da poboljšamo?

Treća verzija

```cpp
#include <iostream>
using namespace std;

int main()
{
	unsigned n = 100;
	for(unsigned i = 1; i <= 6; ++i)
	{
		unsigned uzajamno_prostih = 0;
		for(unsigned a = 1; a <= n; ++a)
		{
			for(unsigned b = 1; b <= n; ++b)
			{
				unsigned nzd = 1;
				for(unsigned d = 2; d <= a; ++d)
				{
					if(d > b)
						//Ako je delilac > b, sigurno ga ne deli,
						//pa mozemo odmah da zaustavimo petlju.
						break;

					if(!(a % d) && !(b % d))
					{
						nzd = d;
						//Posto nam je bitno samo da
						//nadjemo neki delilac > 1,
						//mozemo da prekinemo petlju
						//cim se to desi.

						break;
					}
				}
```

Treća verzija

```
p(100) = 0.6087
p(500) = 0.608924
p(1000) = 0.608383

real    0m0.766s
user    0m0.766s
sys     0m0.000s
```

Treća verzija

```cpp
#include <iostream>
using namespace std;

int main()
{
	unsigned n = 100;
	for(unsigned i = 1; i <= 6; ++i)
	{
		unsigned uzajamno_prostih = 0;
		for(unsigned a = 1; a <= n; ++a)
		{
			for(unsigned b = 1; b <= n; ++b)
			{
				unsigned nzd = 1;
				for(unsigned d = 2; d <= a; ++d)
				{
					if(d > b)
						//Ako je delilac > b, sigurno ga ne deli,
						//pa mozemo odmah da zaustavimo petlju.
						break;

					if(!(a % d) && !(b % d))
					{
						nzd = d;
						//Posto nam je bitno samo da
						//nadjemo neki delilac > 1,
						//mozemo da prekinemo petlju
						//cim se to desi.

						break;
					}
				}
```

Da li možemo još nešto da poboljšamo?

Četvrta verzija

```
unsigned uzajamno_prostih = 0;
for(unsigned a = 1; a <= n; ++a)
{
        for(unsigned b = 1; b < a; ++b)
        {
                //Dovoljno je da posmatramo samo neuredjene parove

                unsigned nzd = 1;
                for(unsigned d = 2; d <= b; ++d)
                {
                        if(!(a % d) && !(b % d))
                        {
                                nzd = d;
                                //Posto nam je bitno samo da
                                //nadjemo neki delilac > 1,
                                //mozemo da prekinemo petlju
                                //cim se to desi.

                                break;
                        }
                }
```

```
                        if (nzd == 1)
                        {
                                uzajamno_prostih++;
                        }
                }
        }
        cout << "p(" << n << ") = " <<
                double(uzajamno_prostih * 2 + 1) / (n * n) << endl;

        //Za svaki prost par (a, b), imamo i prost par (b, a) i dodajemo jos (1, 1)

        n *= i % 2 ? 5 : 2;
    }

    return 0;
}
```

Četvrta verzija

```
p(100) = 0.6087
p(500) = 0.608924
p(1000) = 0.608383

real    0m0.399s
user    0m0.399s
sys     0m0.000s
```

Optimizacicijom koda smo ubrzali program $> 4\times$

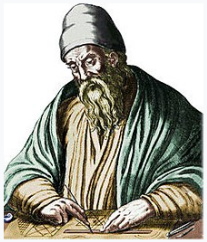
Četvrta verzija

```
    unsigned uzajamno_prostih = 0;
    for(unsigned a = 1; a <= n; ++a)
    {
            for(unsigned b = 1; b < a; ++b)
            {
                    //Dovoljno je da posmatramo samo neuredjene parove

                    unsigned nzd = 1;
                    for(unsigned d = 2; d <= b; ++d)
                    {
                            if(!(a % d) && !(b % d))
                            {
                                    nzd = d;
                                    //Posto nam je bitno samo da
                                    //nadjemo neki delilac > 1,
                                    //mozemo da prekinemo petlju
                                    //cim se to desi.

                                    break;
                            }
                    }
```

```
                            if (nzd == 1)
                            {
                                    uzajamno_prostih++;
                            }
                    }
            }
            cout << "p(" << n << ") = " <<
                    double(uzajamno_prostih * 2 + 1) / (n * n) << endl;

            //Za svaki prost par (a, b), imamo i prost par (b, a) i dodajemo jos (1, 1)

            n *= i % 2 ? 5 : 2;
    }

    return 0;
}
```

Ponovo tražimo odgovor u matematici

Еуклид



Лични подаци	
Датум рођења	око 400. п. н. е.
Датум смрти	око 3. век п. н. е.
Научни рад	
Поље	математика, геометрија
Познат по	Елементима - уџбенику геометрије

Ljudi su proučavali mehanizaciju računanja mnogo pre pojave automatskih računara

Peta verzija

```cpp
#include <iostream>
using namespace std;

int main()
{
        unsigned n = 100;
        for(unsigned i = 1; i <= 6; ++i)
        {
                unsigned uzajamno_prostih = 0;
                for(unsigned a = 1; a <= n; ++a)
                {
                        for(unsigned b = 1; b < a; ++b)
                        {
                                unsigned a_kopija = a;
                                unsigned b_kopija = b;

                                for(; b_kopija > 0;)
                                {
                                        //Ne znamo koliko puta ce petlja biti izvrsena
                                        //a nemamo ni bilo kakvu drugu svrhu od brojaca
                                        //pa nam je dovoljan samo uslov

                                        unsigned zamena = b_kopija;
                                        b_kopija = a_kopija % b_kopija; // b <- a mod b
                                        a_kopija = zamena;              // a <- b
                                }
                                if (a_kopija == 1)
                                {
                                        uzajamno_prostih++;
                                }
                        }
```

Peta verzija

```
p(100) = 0.608722
p(500) = 0.608925
p(1000) = 0.608383

real    0m0.040s
user    0m0.040s
sys     0m0.000s
```

Poboljšanje algoritma je ubrzalo program $\sim 40\times$

Peta verzija

```cpp
#include <iostream>
using namespace std;

int main()
{
	unsigned n = 100;
	for(unsigned i = 1; i <= 6; ++i)
	{
		unsigned uzajamno_prostih = 0;
		for(unsigned a = 1; a <= n; ++a)
		{
			for(unsigned b = 1; b < a; ++b)
			{
				unsigned a_kopija = a;
				unsigned b_kopija = b;

				for(; b_kopija > 0;)
				{
					//Ne znamo koliko puta ce petlja biti izvrsena
					//a nemamo ni bilo kakvu drugu svrhu od brojaca
					//pa nam je dovoljan samo uslov

					unsigned zamena = b_kopija;
					b_kopija = a_kopija % b_kopija; // b <- a mod b
					a_kopija = zamena;              // a <- b
				}
				if (a_kopija == 1)
				{
					uzajamno_prostih++;
				}
			}
```

Da li možemo još nešto da poboljšamo?

Poboljšanje čitljivosti

```cpp
#include <iostream>
using namespace std;

int main()
{
        unsigned n = 100;
        for(unsigned i = 1; i <= 6; ++i)
        {
                unsigned uzajamno_prostih = 0;
                for(unsigned a = 1; a <= n; ++a)
                {
                        for(unsigned b = 1; b < a; ++b)
                        {
                                unsigned a_kopija = a;
                                unsigned b_kopija = b;

                                while(b_kopija > 0)
                                {
                                        //Ne znamo koliko puta ce petlja biti izvrsena
                                        //a nemamo ni bilo kakvu drugu svrhu od brojaca
                                        //pa nam je dovoljan samo uslov

                                        //U tim situacijama je prikladnija while petlja

                                        unsigned zamena = b_kopija;
                                        b_kopija = a_kopija % b_kopija; // b <- a mod b
                                        a_kopija = zamena;              // a <- b
                                }
                                if (a_kopija == 1)
                                {
                                        uzajamno_prostih++;
```

While petlja

While petlja: sintaksa

```
while(uslov)
        telo petlje
```

While petlja: uobičajena upotreba

While petlju obično koristimo kada ne znamo unapred tačan broj izvršenja petlje

Naravno, kao što while petlju možemo svesti na for, možemo i obrnuto

```cpp
#include <iostream>
using namespace std;

int main()
{
        unsigned n = 100;
        for(unsigned i = 1; i <= 6; ++i)
        {
                cout << n << endl;
                n *= i % 2 ? 5 : 2;
        }

        return 0;
}
```

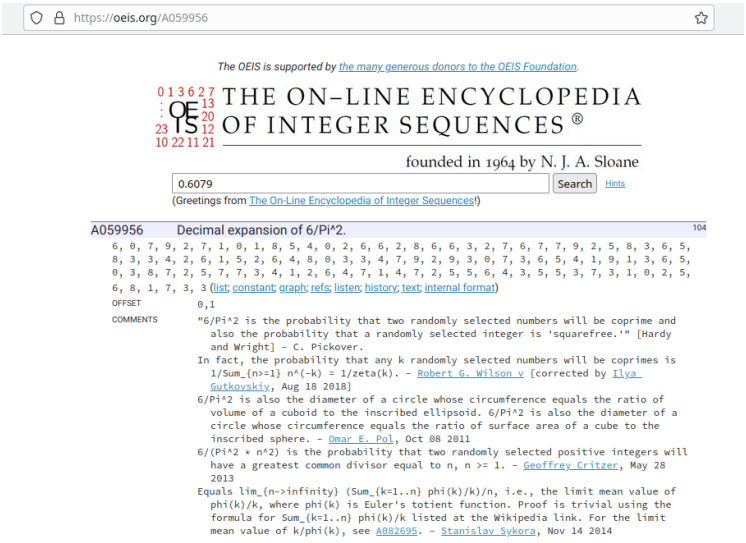
```cpp
#include <iostream>
using namespace std;

int main()
{
        unsigned n = 100;

        unsigned i = 1;
        while(i <= 6)
        {
                cout << n << endl;
                n *= i % 2 ? 5 : 2;

                ++i;
        }

        return 0;
}
```

Šta je odgovor na naše početno pitanje?

Pomoću Euklidovog algoritma, nalazimo sledeći niz vrednosti za p

```
p(100) = 0.6087
p(500) = 0.608924
p(1000) = 0.608383
p(5000) = 0.608037
p(10000) = 0.60795
p(50000) = 0.607939

real    2m8.822s
user    2m8.541s
sys     0m0.044s
```

Hipoteza: Tražena granična vrednost je $\frac{6}{\pi^2}$



Skica dokaza se nalazi na stranici 87 u knjizi

We close this chapter with a sketch of the proof.

Sweeping all worries about convergence under the rug, consider two large integers. What is the probability they are not both even (a necessary condition for the numbers to be relatively prime)? Each has a $\frac{1}{2}$ chance of being even, so the probability neither has a factor of 2 is $\left(1-\frac{1}{4}\right)$. More generally, the probability neither has a prime p as a common factor is $\left(1-1/p^2\right)$. So the limit of p_n is

$$\prod_p \left(1-\frac{1}{p^2}\right)$$

where the product is over all primes.

Recall that $\zeta(2)$ is given by

$$\zeta(2)=\sum_{n=1}^{\infty}\frac{1}{n^2}.$$

This can be expressed as a product. The idea is to factor n^2 into even powers of its prime divisors. The product representation is

$$\zeta(2)=\prod_p \left(1+\frac{1}{p^2}+\frac{1}{p^4}+\frac{1}{p^6}+\cdots\right)$$

Do-while petlja

Ponekad imamo potrebu da telo petlje izvršimo barem jednom, bez obzira na uslov

```cpp
#include <iostream>
using namespace std;

int main()
{
        cout << "Za izlaz unesite 'S'.\n";

        char a = 'A'; //Inicijalizacijom osiguravamo da ce uslov na pocetku biti zadovoljen
        while(a != 'S')
        {
                cout << "Unesite karakter: ";
                cin >> a;

                cout << "ASCII vrednost unetog karaktera (" << a << ") je " << +a << endl;
        }

        return 0;
}
```

Ponekad imamo potrebu da telo petlje izvršimo barem jednom, bez obzira na uslov

```
Za izlaz unesite 'S'.
Unesite karakter: a
ASCII vrednost unetog karaktera (a) je 97
Unesite karakter: b
ASCII vrednost unetog karaktera (b) je 98
Unesite karakter: c
ASCII vrednost unetog karaktera (c) je 99
Unesite karakter: S
ASCII vrednost unetog karaktera (S) je 83
```

Ponekad imamo potrebu da telo petlje izvršimo barem jednom, bez obzira na uslov

```cpp
#include <iostream>
using namespace std;

int main()
{
        cout << "Za izlaz unesite 'S'.\n";
        while(1)        //Beskonacna petlja
        {
                char a;
                cout << "Unesite karakter: ";
                cin >> a;

                cout << "ASCII vrednost unetog karaktera (" << a << ") je " << +a << endl;

                if(a == 'S')  //Proveravamo inverziju uslova (!(a != 'S')) i ako je tacna, izlazimo iz petlje
                        break;
        }

        return 0;
}
```

Ponekad imamo potrebu da telo petlje izvršimo barem jednom, bez obzira na uslov

```
Za izlaz unesite 'S'.
Unesite karakter: a
ASCII vrednost unetog karaktera (a) je 97
Unesite karakter: b
ASCII vrednost unetog karaktera (b) je 98
Unesite karakter: c
ASCII vrednost unetog karaktera (c) je 99
Unesite karakter: S
ASCII vrednost unetog karaktera (S) je 83
```

Ponekad imamo potrebu da telo petlje izvršimo barem jednom, bez obzira na uslov

```cpp
#include <iostream>
using namespace std;

int main()
{
        cout << "Za izlaz unesite 'S'.\n";
        char a;
        do
        {
                cout << "Unesite karakter: ";
                cin >> a;

                cout << "ASCII vrednost unetog karaktera (" << a << ") je " << +a << endl;

        } while(a != 'S');

        return 0;
}
```

Ponekad imamo potrebu da telo petlje izvršimo barem jednom, bez obzira na uslov

```
Za izlaz unesite 'S'.
Unesite karakter: a
ASCII vrednost unetog karaktera (a) je 97
Unesite karakter: b
ASCII vrednost unetog karaktera (b) je 98
Unesite karakter: c
ASCII vrednost unetog karaktera (c) je 99
Unesite karakter: S
ASCII vrednost unetog karaktera (S) je 83
```

Do-while petlja: sintaksa

```
do
        telo petje
while(uslov);
```

Naredba switch

Naredba switch

```cpp
#include <iostream>

#define CRVENA 0
#define ZELENA 1
#define PLAVA 2

using namespace std;

int main()
{
	unsigned short boja = 0;

	cout << "Unesite sifru boje (0 za crvenu, 1 za zelenu, 2 za plavu): ";
	cin >> boja;

	switch(boja)
	{
		case CRVENA:
			cout << "Izabrana boja je crvena\n";
			break;

		case ZELENA:
			cout << "Izabrana boja je zelena\n";
			break;

		case PLAVA:
			cout << "Izabrana boja je plava\n";
			break;

		default:
			cout << "Izabrana je nepoznata boja\n";
			break;
	}

	return 0;
}
```

Ponekad je potrebno da vrednost jedne celobrojne promenljive uporedimo sa više konstanti i donesemo odluku na osnovu tog poređenja

Naredba switch

```
Unesite sifru boje (0 za crvenu, 1 za zelenu, 2 za plavu): 1
Izabrana boja je zelena
```

Naredba switch

```
Unesite sifru boje (0 za crvenu, 1 za zelenu, 2 za plavu): 4
Izabrana je nepoznata boja
```

Naredba switch: sintaksa

```
switch(promenljiva)
{
        case konstantna_vrednost_1: naredbe za vrednost 1
        case konstantna_vrednost_2: naredbe za vrednost 2
       ......
        default: naredbe za nenavedene vrednosti
}
```

„Propadanje” switch-a

```cpp
#include <iostream>

#define CRVENA 0
#define ZELENA 1
#define PLAVA 2

using namespace std;

int main()
{
        unsigned short boja = 0;

        cout << "Unesite sifru boje (0 za crvenu, 1 za zelenu, 2 za plavu): ";
        cin >> boja;

        switch(boja)
        {
                case CRVENA:
                        cout << "Izabrana boja je crvena\n";
                        break;

                case ZELENA:
                        cout << "Izabrana boja je zelena\n";

                case PLAVA:
                        cout << "Izabrana boja je plava\n";

                default:
                        cout << "Izabrana je nepoznata boja\n";
        }

        return 0;
}
```

„Propadanje” switch-a

```
Unesite sifru boje (0 za crvenu, 1 za zelenu, 2 za plavu): 1
Izabrana boja je zelena
Izabrana boja je plava
Izabrana je nepoznata boja
```

Izvršavaju se sve naredbe unutar switch-a nakon prvog zadovoljenog case-a. Zato kada želimo da razdvojimo slučajeve, koristimo naredbu break.

„Propadanje” switch-a nekad može da bude i korisno

```cpp
#include <iostream>
using namespace std;

int main()
{
	char c;
	cout << "Unesite karakter: ";
	cin >> c;

	switch(c)
	{
		case 'a':
		case 'b':
		case 'c':
		case 'd':
			cout << "Uneto je malo slovo izmedju a i d\n";
			break;
		case 'A':
		case 'B':
		case 'C':
		case 'D':
			cout << "Uneto je veliko slovo izmedju A i D\n";
			break;

		case '1':
		case '2':
		case '3':
		case '4':
			cout << "Unet je prirodni broj < 5\n";
			break;

		default:
			cout << "Unet je nepoznat karakter\n";
	}

	return 0;
}
```

„Propadanje” switch-a nekad može da bude i korisno

```
Unesite karakter: a
Uneto je malo slovo izmedju a i d
```

„Propadanje” switch-a nekad može da bude i korisno

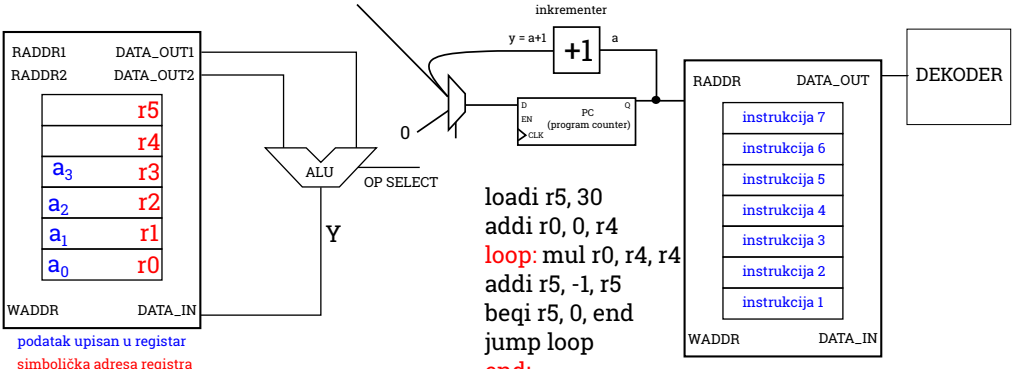
```
Unesite karakter: 1
Unet je prirodni broj < 5
```

„Propadanje” switch-a nekad može da bude i korisno

```
Unesite karakter: B
Uneto je veliko slovo izmedju A i D
```

Naredba goto

Setimo se petlji u asembliju, realizovanih upotrebom bezuslovnog skoka



Naredba goto

```cpp
#include <iostream>
using namespace std;

int main()
{
        unsigned n = 100;
        unsigned i;

        cout << n << endl;
        for(i = 1; i <= 6; ++i)
        {
                cout << "i = " << i << endl;
                n *= i % 2 ? 5 : 2;

                if(n > 3000)
                        break;

                cout << n << endl;
        }

        cout << "Prva naredba nakon petlje.\n"; //break bezuslovno skace ovde

        return 0;
}
```

```cpp
#include <iostream>
using namespace std;

int main()
{
        unsigned n = 100;
        unsigned i = 1;

        cout << n << endl;

LOOP_INIT:
        cout << "i = " << i << endl;
        n *= i % 2 ? 5 : 2;

        if(n > 3000)
                goto BREAK_EQUIVALENT;

        cout << n << endl;
        ++i;

        if(i <= 6)
                goto LOOP_INIT;

BREAK_EQUIVALENT:
        cout << "Prva naredba nakon petlje.\n"; //break bezuslovno skace ovde

        return 0;
}
```

C/C++ zadržava tu mogućnost

Naredba goto: sintaksa

```
goto labela;

labela:
```

Naredba goto čini kod teško čitljivim i otežava kompajleru optimizacije

Jedan od retkih slučajeva gde je ponekad ima smisla koristiti je izlazak iz duboko ugnježdenih petlji:

```
petlja 1
{
        petlja 2
       {
              petlja 3
             {
                    if(uslov konacnog izlaza)
                          goto IZLAZ;
             }
       }
}
IZLAZ:
```

Nekada su ljudi zloupotrebljavali goto, ali je 1968. Edsger Dajkstra ukazao na taj problem

Edgar Dijkstra: Go To Statement Considered Harmful

Go To Statement Considered Harmful

Key Words and Phrases: go to statement, jump instruction, branch instruction, conditional clause, alternative clause, repetitive clause, program intelligibility, program sequencing
CR Categories: 4.22, 5.23, 5.24

EDITOR:

For a number of years I have been familiar with the observation that the quality of programmers is a decreasing function of the density of **go to** statements in the programs they produce. More recently I discovered why the use of the **go to** statement has such disastrous effects, and I became convinced that the **go to** statement should be abolished from all "higher level" programming languages (i.e. everything except, perhaps, plain machine code). At that time I did not attach too much importance to this discovery; I now submit my considerations for publication because in very recent discussions in which the subject turned up, I have been urged to do so.

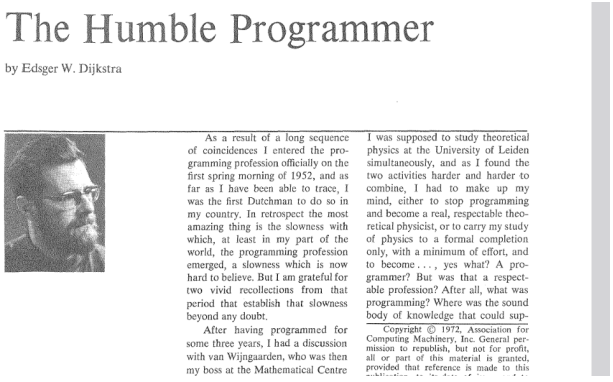
dynamic progress is only characterized when we also give to which call of the procedure we refer. With the inclusion of procedures we can characterize the progress of the process via a sequence of textual indices, the length of this sequence being equal to the dynamic depth of procedure calling.

Let us now consider repetition clauses (like, **while** *B* **repeat** *A* or **repeat** *A* **until** *B*). Logically speaking, such clauses are now superfluous, because we can express repetition with the aid of recursive procedures. For reasons of realism I don't wish to exclude them: on the one hand, repetition clauses can be implemented quite comfortably with present day finite equipment; on the other hand, the reasoning pattern known as "induction" makes us well equipped to retain our intellectual grasp on the processes generated by repetition clauses. With the inclusion of the repetition clauses textual indices are no longer sufficient to describe the dynamic progress of the process. With each entry into a repetition clause, however, we can associate a so-called "dynamic index," inexorably counting the ordinal number of the

Edsger Dajkstra (Edsger Dijkstra)

Nakon što sam programirao oko tri godine, imao sam razgovor sa A. van Vijngardenom, mojim tadašnjim šefom u Matematičkom centru u Amsterdamu. Bio je to razgovor na kom ću mu biti zahvalan dok sam živ. Suština je bila u tome da je trebalo da istovremeno studiram teorijsku fiziku na Univerzitetu u Lajdenu, a kako mi je bivalo sve teže i teže da kombinujem te dve aktivnosti, trebalo je da donesem odluku: ili da prestanem da programiram i postanem pravi, ozbiljan teorijski fizičar, ili da nastavim svoje studije fizike do tek formalnog završetka, uz minimalni trud, i da postanem... Da, šta? Programer? Ali je li to bilo zanimanje vredno poštovanja? Na kraju krajeva, šta je to bilo programiranje? Gde je bilo smislen korpus znanja koji bi programiranje podržao kao intelektualno ozbiljnu disciplinu? Sećam se prilično živo koliko sam zavideo svojim kolegama iz oblasti hardvera koji su, kada bi ih neko upitao za stručna znanja, barem mogli da kažu da znaju sve o vakumskim cevima, pojačavačima i sličnom. S druge strane, ja sam se pri susretu sa tim pitanjem nalazio bez odgovora. Pun nedoumica, pokucao sam na vrata van Vijngardenove kancelarije i upitao ga da li mogu „na trenutak" da razgovaram sa njim. Kada sam, posle nekoliko sati, napustio njegov kabinet, bio sam druga osoba. Nakon što je pažljivo saslušao moje probleme, složio se da do tada nije bilo nečeg poput programerske discipline. Zatim je dodao da su automatski računari tu da ostanu, da smo mi tek na početku, i zar ne bih ja mogao da budem jedna od tih osoba pozvanih da programiranje učine ozbiljnom disciplinom u godinama koje slede? To je bila prekretnica u mom životu i završio sam svoje studije fizike samo formalno, što sam brže mogao. Jedna pouka ove priče je, naravno, da moramo da budemo veoma pažljivi kada mlađim ljudima dajemo savete; ponekad ih i slede!

Skromni programer



The Humble Programmer

by Edsger W. Dijkstra

As a result of a long sequence of coincidences I entered the programming profession officially on the first spring morning of 1952, and as far as I have been able to trace, I was the first Dutchman to do so in my country. In retrospect the most amazing thing is the slowness with which, at least in my part of the world, the programming profession emerged, a slowness which is now hard to believe. But I am grateful for two vivid recollections from that period that establish that slowness beyond any doubt.

After having programmed for some three years, I had a discussion with van Wijngaarden, who was then my boss at the Mathematical Centre I was supposed to study theoretical physics at the University of Leiden simultaneously, and as I found the two activities harder and harder to combine, I had to make up my mind, either to stop programming and become a real, respectable theoretical physicist, or to carry my study of physics to a formal completion only, with a minimum of effort, and to become . . . , yes what? A programmer? But was that a respectable profession? After all, what was programming? Where was the sound body of knowledge that could sup-

Prošireni tekst Dajkstrinog saopštenja sa dodele Tjuringove nagrade, pod naslovom „Skromni programer” je dostupan na sledećoj adresi:
`https://dl.acm.org/doi/pdf/10.1145/355604.361591`

Osnivač Instituta za veštačku inteligenciju Srbije

U utorak 5. marta 2024. godine u 17:15 u Amfiteatru 7, u zgradi DF/DMI, dr Branislav Kisačanin će održati predavanje na temu:

Kako najbolje iskoristiti "veliki prasak" primena veštačke inteligencije?

Sažetak:

Svedoci smo ogromnog napretka u mnogim naučnim i tehničkim disciplinama zahvaljujući upotrebi novih tehnika i alata na bazi mašinskog učenja, tj. veštačke inteligencije. Neverovatna je i brzina kojom se nova otkrića komercijalizuju. Na ovom predavanju pričaćemo o tome i skiciraćemo šta treba da znate da bi ste se uključili u sve to.

O predavaču:

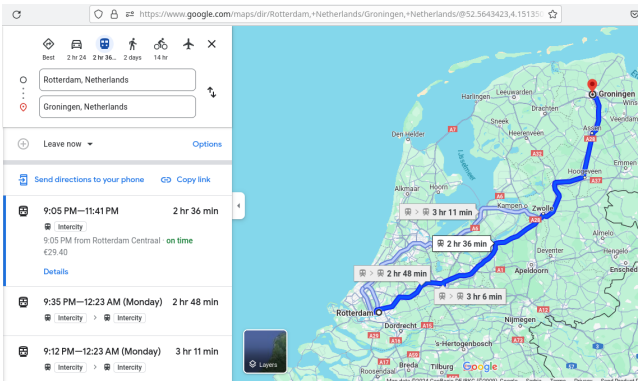
Dr Branislav Kisačanin je završio redovne studije na Univerzitetu u Novom Sadu (FTN, Elektro), doktorirao je računarske nauke na University of Illinois i od tada je radio u nekoliko industrijskih R&D laboratorija u SAD (Delphi, Texas Instruments, a od 2015. Nvidia). Angažovan je na Katedri za elektroniku na FTN-u i jedan je od osnivača Instituta za veštačku inteligenciju Srbije. Napisao je ili uredio 9 knjiga, ima 10 patenata u SAD i EU i ima veliko iskustvo u radu sa mladim talentima, najviše kroz organizaciju za pripreme za srednjoškolska takmičenja iz matematike i fizike, AwesomeMath. Ambicija mu je da pomogne da Srbija osvoji 1% svetskog tržišta proizvoda na bazi veštačke inteligencije i dodaje "ko ume 1, može da cilja i na 2 i na 3 procenta"!

Mašinsko učenje je > 90% matematika, a pošto mi u Srbiji imamo jako dobre matematičare, to je naša šansa da napravimo proboj u ovoj oblasti

Novi problem: najkraće putanje u grafovima

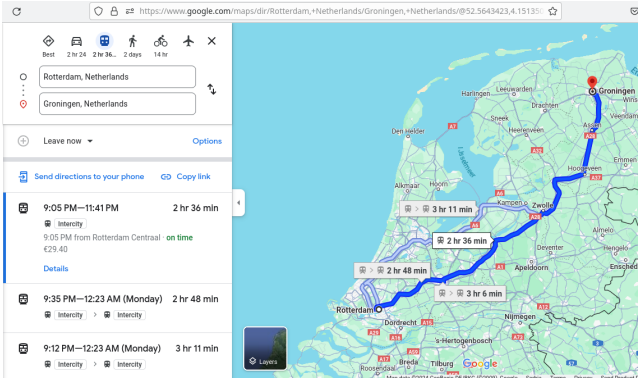
Vratimo se u 1956.

Vratimo se u 1956.



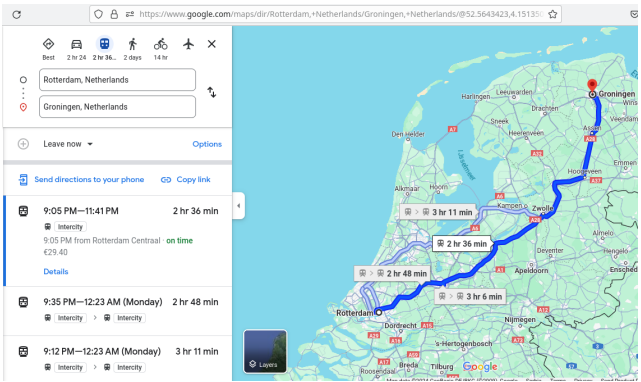
Dajkstru je mučio sledeći problem: kako da nađe najkraću putanju vozom između Roterama i Groningena?

Vratimo se u 1956.



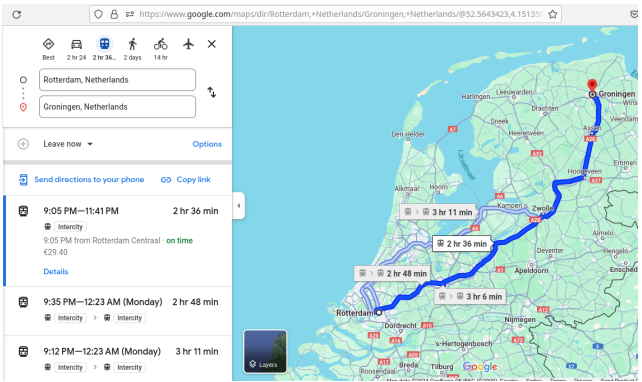
A onda i uopšteno, između bilo koja dva grada

Vratimo se u 1956.



Problem je rešio na licu mesta, za 20ak minuta, bez korišćenja papira i olovke

Vratimo se u 1956.



Nije smatrao da je to nešto značajno, pa je rezultat objavio tek tri godine kasnije

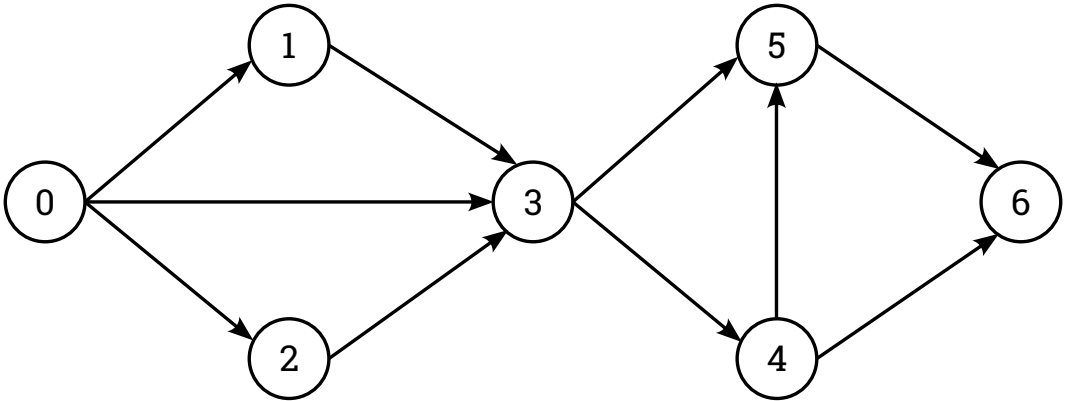
Dajkstrin algoritam

Počnimo od jednostavnijeg problema

Umesto traženja najkraće putanje u smislu ukupnog trajanja vožnje, tražimo najkraću putanju po broju presedanja

Kako možemo predstaviti železničku mrežu?

Kako možemo predstaviti železničku mrežu?



najkraća putanja od 0 do 6: (0, 3, 4, 6)

Kako tražimo najkraću putanju od početka do cilja?

Kako tražimo najkraću putanju od početka do cilja?

Jedan pristup: optimizacioni problem pretvorimo u problem odluke

~~Koja je najkraća putanja od s do t?~~

Da li postoji putanja od s do t, dužine n, $n \in (1, 2, 3, 4, 5, ...)$?

Da li od s do t postoji putanja dužine n?

Kako možemo naći odgovor na ovo pitanje?

Da li od s do t postoji putanja dužine n?

Naivan pristup: izlistamo sve čvorove na razdaljini n od s i proverimo da li je t među njima

Koji su čvorovi na razdaljini n od s?

To su čvorovi na razdaljini 1 od nekog čvora koji je na razdaljini $n - 1$ od s

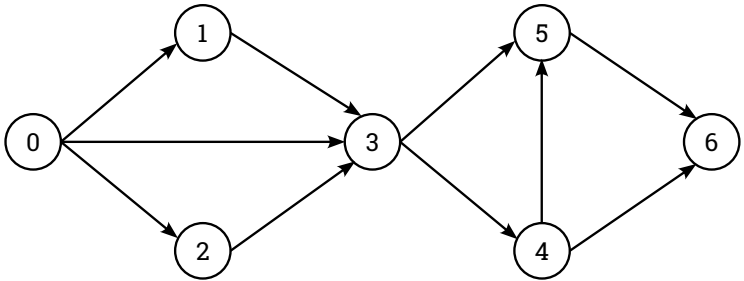
Rekurzivno

To su pak čvorovi na razdaljini 1 od nekog čvora koji je na razdaljini $n - 2$ od s

Pretraga po širini (eng. *Breadth-First Search (BFS)*)

1. $S_0 = \{s\}$
2. Za svako u iz S_i, dodaj u S_{i+1} sve v za koje je $(u, v) \in E$ i koji nisu već dodati u neki S_j, $\quad j \leqslant i$
3. Ako je t u S_i, dužina najkraće putanje između s i t je i

Na našem primeru



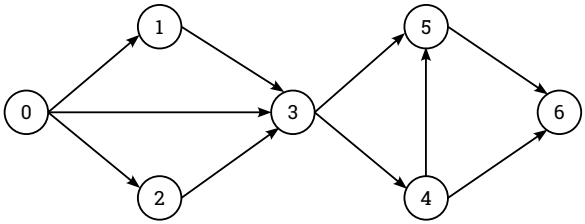
$S_0 = \{0\}$

$S_1 = \{1, 2, 3\}$

$S_2 = \{4, 5\}$ 3 je već u S_1 pa ga ne ubacujemo u S_2

$S_3 = \{6\}$ 5 je već u S_2 pa ga ne ubacujemo u S_3

Rekonstrukcija putanje



$S_0 = \{0\}$

$S_1 = \{1, 2, 3\}$

$S_2 = \{4, 5\}$ 3 je već u S_1 pa ga ne ubacujemo u S_2

$S_3 = \{6\}$ 5 je već u S_2 pa ga ne ubacujemo u S_3

0 1 2 3 4 5 6

$A = (0, 0, 0, 0, 3, 3, 4)$

Da bismo mogli da rekonstruišemo putanju iz skupova *S*, potrebno je da zapamtimo prilikom obrade kog roditelja je dati čvor *v* uvršten u svoj skup *S*

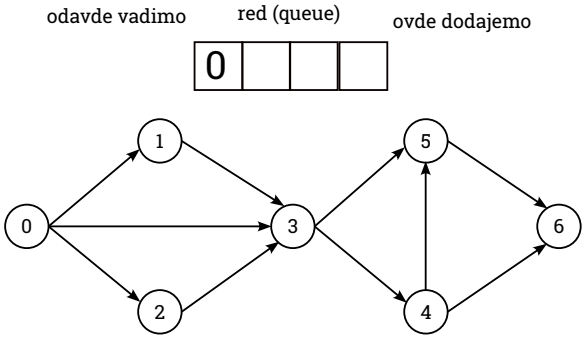
Jedna korisna struktura podataka: red

Red (eng. Queue): FIFO (first-in, first-out) struktura

Niz na čiji kraj dodajemo elemente, a sa čijeg početka ih uzimamo

Čuva poredak kada u njega pakujemo čvorove umesto u skupove S

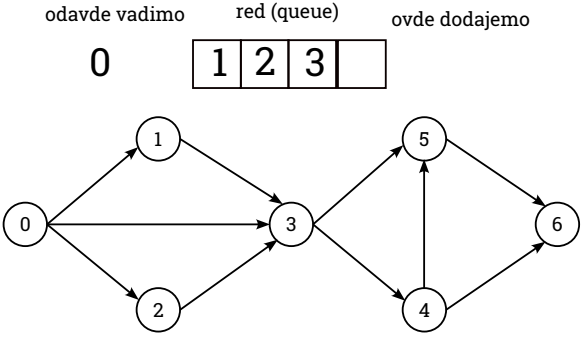
Naš primer



0 1 2 3 4 5 6

A = (0, -1, -1, -1, -1, -1, -1)

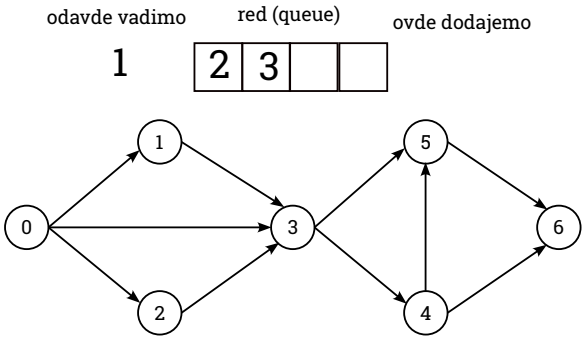
Naš primer



0 1 2 3 4 5 6

A = (0, 0, 0, 0, -1, -1, -1)

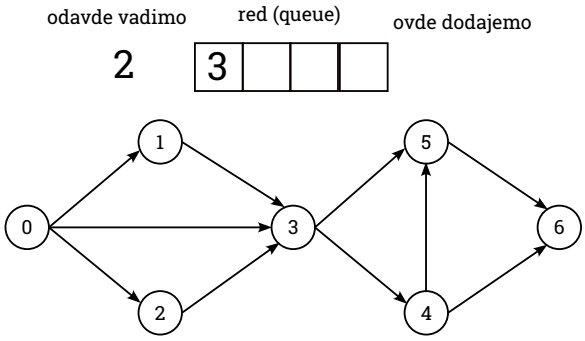
Naš primer



0 1 2 3 4 5 6

A = (0, 0, 0, 0, -1, -1, -1)

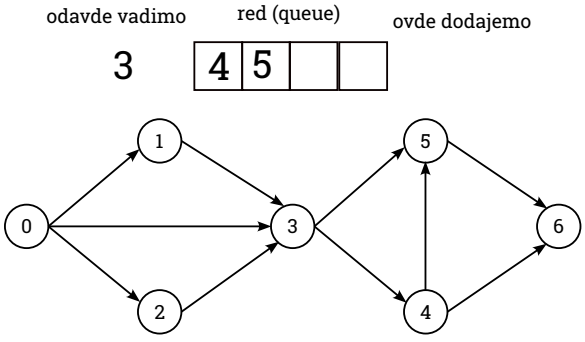
Naš primer



0 1 2 3 4 5 6

A = (0, 0, 0, 0, -1, -1, -1)

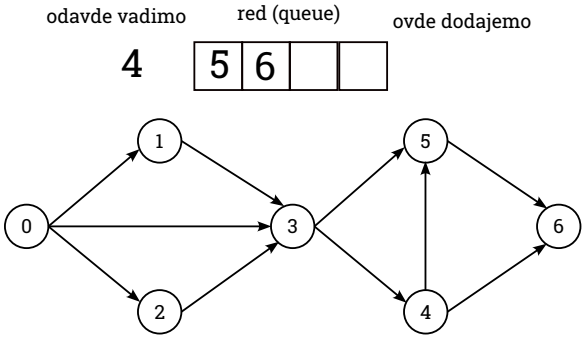
Naš primer



0 1 2 3 4 5 6

A = (0, 0, 0, 0, 3, 3, -1)

Naš primer



0 1 2 3 4 5 6

A = (0, 0, 0, 0, 3, 3, 4)

Jedna implementacija BFS-a u C++-u

```cpp
#include <iostream>
#define V 7
#define NOT_REACHED -1

using namespace std;

int main()
{
        unsigned char adj[V][V] = {
                                    {1, 1, 1, 1, 0, 0, 0},
                                    {0, 1, 0, 1, 0, 0, 0},
                                    {0, 0, 1, 1, 0, 0, 0},
                                    {0, 0, 0, 1, 1, 1, 0},
                                    {0, 0, 0, 0, 1, 1, 1},
                                    {0, 0, 0, 0, 0, 1, 1},
                                    {0, 0, 0, 0, 0, 0, 1}
                                   };

        const unsigned char s = 0;
        const unsigned char t = 6;

```

Najpre predstavimo graf matricom susednosti

Jedna implementacija BFS-a u C++-u

```
22          unsigned char queue[V];
23          unsigned char head_ptr = 0; //lokacija uzimanja
24          unsigned char tail_ptr = 0; //lokacija dodavanja
25          queue[tail_ptr++] = s;
26
```

Inicijalizujemo red

Jedna implementacija BFS-a u C++-u

```
27          char reached_from[V];
28          for(unsigned char i = 0; i < V; ++i)
29                  reached_from[i] = NOT_REACHED;
30          reached_from[s] = s;
31
```

Inicijalizujemo niz prethodnika

Jedna implementacija BFS-a u C++-u

```
32          bool end_reached = false;
33          while(!end_reached && (tail_ptr > head_ptr))
34          {
35                  unsigned char u = queue[head_ptr++];
36
37                  for(unsigned char v = 0; v < V; ++v)
38                  {
39                          if((reached_from[v] == NOT_REACHED) && adj[u][v])
40                          {
41                                  queue[tail_ptr++] = v;
42                                  reached_from[v] = u;
43
44                                  if(v == t)
45                                  {
46                                          end_reached = true;
47                                          break;
48                                  }
49                          }
50                          if(end_reached)
51                                  break;
52                  }
53          }
54
```

Redom obrađujemo čvorove

Jedna implementacija BFS-a u C++-u

```
55          if(!end_reached)
56                  cout << "Putanja od " << +s << " do " << +t << " nije pronadjena.\n";
57          else
58          {
59                  unsigned char trace = t;
60                  unsigned char path[V];
61                  unsigned char path_ptr = 0;
62
63                  while(trace != s)
64                  {
65                          path[path_ptr++] = trace;
66                          trace = reached_from[trace];
67                  }
68                  path[path_ptr] = s;
69
70                  cout << "Najkraca putanja od " << +s << " do " << +t << ":\n";
71                  while(path_ptr > 0)
72                          cout << +path[path_ptr--] << " -> ";
73                  cout << +path[0] << endl;
74          }
75
76          return 0;
77  }
```

Rekonstruišemo putanju

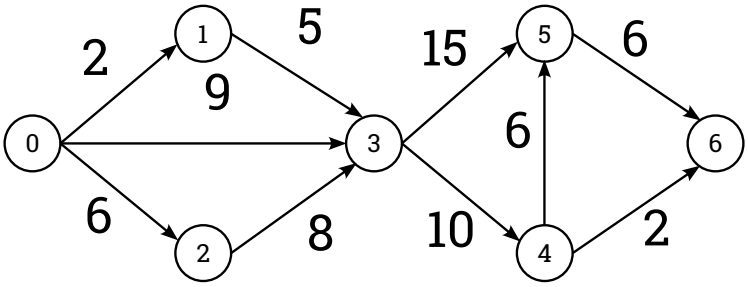
Jedna implementacija BFS-a u C++-u

```
Najkraca putanja od 0 do 6:
0 -> 3 -> 4 -> 6
```

Nizove ćemo detaljno raditi na sledećem predavanju

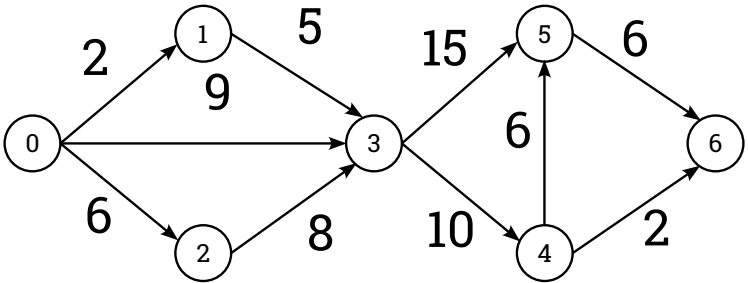
Vratimo se na Dajkstrin inicijalni problem

Vremena putovanja možemo predstaviti težinama na granama



Najkraća putanja od 0 do 6: (0, 1, 3, 4, 6)

Kako sada možemo da nađemo najkraću putanju?



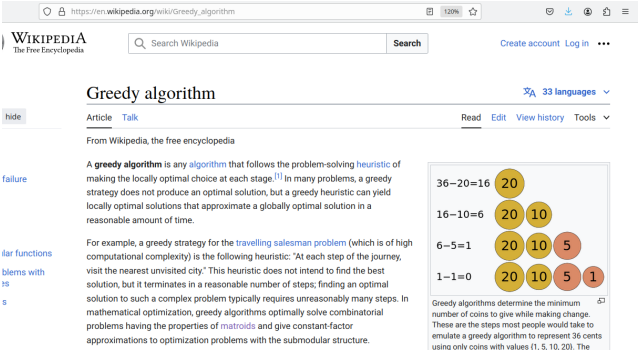
Najkraća putanja od 0 do 6: (0, 1, 3, 4, 6)

Ponovo obrađujemo čvorove po udaljenosti od početka, ali sada otežanoj

Bitne ideje:

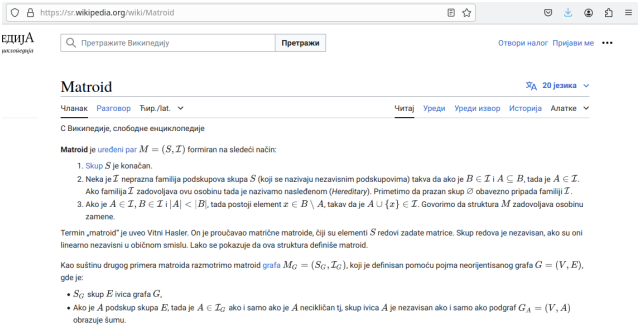
1. Za svaki čvor v čuvamo najkraću do sada viđenu putanju od s, $d'(v)$ (na početku ∞, osim za s, gde je 0)
2. Kada obrađujemo čvor u, na udaljenosti $d(u)$ od s, $d(u) + w(u, v)$ može biti manje od $d'(v)$; ako jeste, postavimo $d'(v)$ na tu vrednost
3. Ako u svakom trenutku obrađujemo čvor u sa najmanjim $d'(u) = d(u)$, i ako je $\forall w \geqslant 0$, ne može se dogoditi da naiđemo na u' tako da je $d'(u) > d'(u') + w(u', u)$, jer onda $d'(u)$ ne bi bilo najmanje
4. Drugim rečima, u trenutku kada čvor u sa najmanjim $d'(u)$ dođe na red za obradu, najkraća do tada viđena putanja od početka do njega je ujedno i globalno najkraća putanja od početka do njega

Opet nam pomaže matematika



Pošto u svakom trenutku obrađuje čvor u sa najmanjim d', Dajkstrin algoritam pripada klasi takozvanih „pohlepnih” (eng. *greedy*) algoritama

Dokazivanje optimalnosti pohlepnih algoritama



https://sr.wikipedia.org/wiki/Matroid

ЕДИЈА

Претражите Википедију

Претражи

Отвори налог Пријави ме

Matroid

20 језика

Чланак Разговор Тип./lat.

Читај Уреди Уреди извор Историја Алатке

С Википедије, слободне енциклопедије

Matroid је уређени пар $M = (S, \mathcal{I})$ формиран на следећи начин:

1. Скуп S је коначан.
2. Neka je $\mathcal{I}$ neprazna familija podskupova skupa S (koji se nazivaju nezavisnim podskupovima) takva da ako je $B \subset \mathcal{I}$ i $A \subseteq B$, tada je $A \in \mathcal{I}$. Ako familija $\mathcal{I}$ zadovoljava ovu osobinu tada je nazivamo nasleđenom (*Hereditary*). Primetimo da prazan skup $\varnothing$ obavezno pripada familiji $\mathcal{I}$.
3. Ako je $A \in \mathcal{I}, B \in \mathcal{I}$ i $|A| < |B|$, tada postoji element $x \in B \setminus A$, takav da je $A \cup \{x\} \in \mathcal{I}$. Govorimo da struktura M zadovoljava osobinu zamene.

Termin „matroid" je uveo Vitni Hasler. On je proučavao matrične matroide, čiji su elementi S redovi zadate matrice. Skup redova je nezavisan, ako su oni linearno nezavisni u običnom smislu. Lako se pokazuje da ova struktura definiše matroid.

Kao suštinu drugog primera matroida razmotrimo matroid grafa $M_G = (S_G, \mathcal{I}_G)$, koji je definisan pomoću pojma neorijentisanog grafa $G = (V, E)$, gde je:

- S_G skup E ivica grafa G,
- Ako je A podskup skupa E, tada je $A \in \mathcal{I}_G$ ako i samo ako je A neciklični tj, skup ivica A je nezavisan ako i samo ako podgraf $G_A = (V, A)$ obrazuje šumu.

U opštem slučaju, dokazivanje optimalnosti pohlepnih algoritama se sprovodi dokazivanjem da je dati problem matroid

Jedna implementacija Dajkstrinog algoritma u C++-u

```cpp
#include <iostream>
#include <climits>

#define V 7
#define NOT_REACHED -1
#define INF UCHAR_MAX

using namespace std;

int main()
{
        unsigned char adj[V][V] = {
                                    {0, 2, 6, 9, INF, INF, INF},
                                    {INF, 0, INF, 5, INF, INF, INF},
                                    {INF, INF, 0, 8, INF, INF, INF},
                                    {INF, INF, INF, 0, 10, INF, INF},
                                    {INF, INF, INF, INF, 0, 6, 2},
                                    {INF, INF, INF, INF, INF, 0, 6},
                                    {INF, INF, INF, INF, INF, INF, 0}
                                   };

        const unsigned char s = 0;
        const unsigned char t = 6;
```

Najpre predstavimo graf otežanom matricom susednosti

Jedna implementacija Dajkstrinog algoritma u C++-u

```
25          unsigned char dist_from_s[V]; //najkrace do sada vidjene putanje do svakog covra
26          char reached_from[V]; //niz predaka
27          bool visited[V]; //evidencija obradjenih cvorova (svaki obradjujemo samo jednom)
28          for(unsigned char i = 0; i < V; ++i)
29          {
30                  dist_from_s[i] = INF;
31                  reached_from[i] = NOT_REACHED;
32                  visited[i] = false;
33          }
34          dist_from_s[s] = 0;
35          reached_from[s] = s;
36
```

Inicijalizujemo niz trenutnih najkraćih putanja do svakog čvora, niz prethodnika i indikaciju obrađenih čvorova

Jedna implementacija Dajkstrinog algoritma u C++-u

```
37          bool end_reached = false;
38          while(!end_reached)
39          {
40                  unsigned char min_dist = INF;
41                  unsigned char u = 0;
42                  for(unsigned i = 0; i < V; ++i)
43                  {
44                          if(visited[i])
45                                  continue;
46                          if(dist_from_s[i] < min_dist)
47                          {
48                                  min_dist = dist_from_s[i];
49                                  u = i;
50                          }
51                  }
52
53                  if(min_dist == INF)
54                  {
55                          cout << "Node " << +t << " is not reachable from " << +s << endl;
56                          break;
57                  }
58
```

U svakoj iteraciji tražimo neposećen čvor na najmanjoj trenutnoj distanci od s. Ako ne nađemo ni jedan neposećen čvor, onda ne postoji putanja od s do t.

Jedna implementacija Dajkstrinog algoritma u C++-u

```
59                  visited[u] = true;
60                  if(u == t)
61                  {
62                          end_reached = true;
63                          break;
64                  }
65
66
```

Zabeležimo da smo obradili taj čvor. Ako smo stigli do *t*, našli smo najkraću putanju.

Jedna implementacija Dajkstrinog algoritma u C++-u

```
67                  for(unsigned char v = 0; v < V; ++v)
68                  {
69                          if(adj[u][v] < INF)
70                          {
71                                  if(dist_from_s[v] > dist_from_s[u] + adj[u][v])
72                                  {
73                                          reached_from[v] = u;
74                                          dist_from_s[v] = dist_from_s[u] + adj[u][v];
75                                  }
76                          }
77                  }
78          }
79
```

Osvežimo najkraću do sada viđenu putanju od s do svakog od sledbenika čovra koji obrađujemo

Jedna implementacija Dajkstrinog algoritma u C++-u

```
80          if(!end_reached)
81                  cout << "Putanja od " << +s << " do " << +t << " nije pronadjena.\n";
82          else
83          {
84                  unsigned char trace = t;
85                  unsigned char path[V];
86                  unsigned char path_ptr = 0;
87
88                  while(trace != s)
89                  {
90                          path[path_ptr++] = trace;
91                          trace = reached_from[trace];
92                  }
93                  path[path_ptr] = s;
94
95                  cout << "Najkraca putanja od " << +s << " do " << +t << ":\n";
96                  while(path_ptr > 0)
97                          cout << +path[path_ptr--] << " -> ";
98                  cout << +path[0] << endl;
99          }
100
101         return 0;
102 }
```

Rekonstruišemo putanju, kao i ranije

Jedna implementacija Dajkstrinog algoritma u C++-u

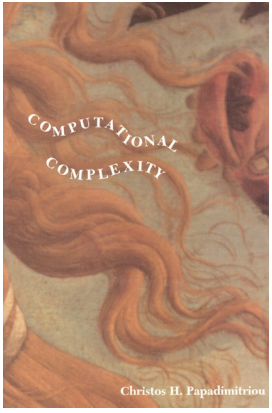
```
Najkraca putanja od 0 do 6:
0 -> 1 -> 3 -> 4 -> 6
```

Napomena

U datoj implementaciji, u najgorem slučaju moramo da obradimo svih $|V|$ čvorova i da za svaki od njih pronađemo čvor sa najmanjim $d'(u)$; opet moramo proći kroz svih $|V|$ čvorova. Stoga kažemo da je **RAČUNSKA SLOŽENOST** Dajkstrinog algoritma u ovom (izvornom) obliku $\mathcal{O}(|V|^2)$

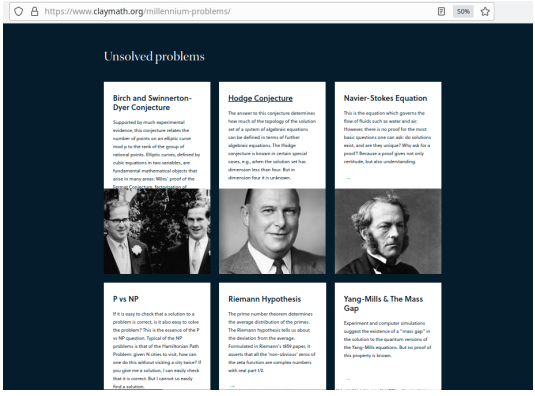
Kasniji autori su niz dužina do sada viđenih putanja do svakog čvora zamenili različitim oblicima **PRIORITETNOG REDA** (eng. *priority queue*), čime su izbegli prolazak kroz sve čvorove tokom traženja $d'(u)$ i tako sveli računsku složenost na $\Theta(|E| + |V| \log_2 |V|)$

Za one koji žele da znaju više



Ova knjiga je odličan uvod u računsku složenost

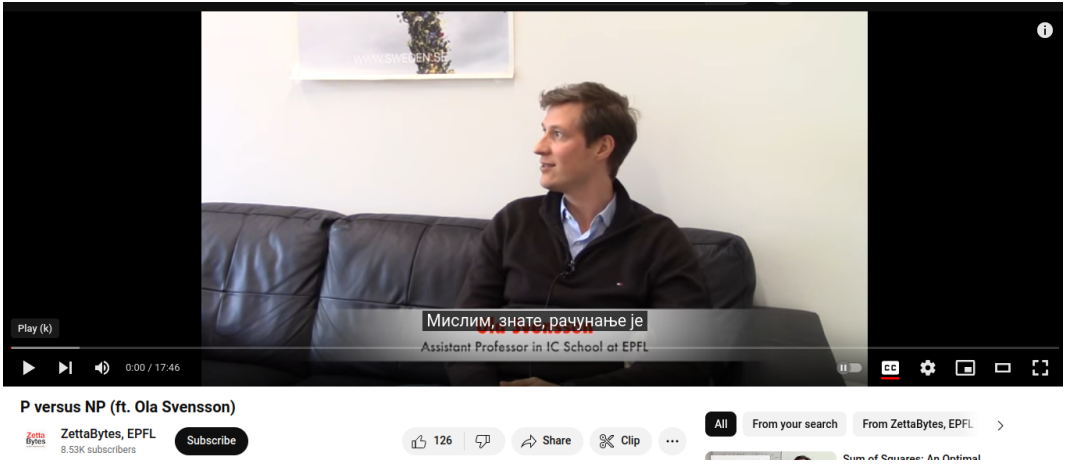
Teorija računske složenosti je zapravo matematička oblast



Jedan od šest nerešenih *milenijumskih problema* u matematici je pitanje da li je klasa složenosti P jednaka klasi složenosti NP

Osoba koja ga reši će od Klej matematičkog instituta dobiti milion dolara

A na sledećoj adresi je dostupan jedan zanimljiv video zapis o svemu tome



https://www.youtube.com/watch?v=hyAH9Tv8EKg

Hvala na pažnji