

# Reševanje problemov

1. Predpogoji za reševanje problemov
2. Kateri problemi so sploh rešljivi?
3. Osnovni principi reševanja
4. Reševanje problemov z algoritmi
5. Primer razvoja algoritma

# Algoritmi

*Algoritem je **končno zaporedje** natančno določenih ukazov, ki opravijo neko nalogo v **končnem številu korakov**.*

*Algoritem sprejme vhodne podatke in vrne rezultat.*

*Program je algoritem, ki ga lahko izvajamo na računalniku.*

*Program je algoritem, ki je zapisan v **programskem jeziku**.*

# Muḥammad ibn Mūsā al-Khwārizmī

(cca.780 – cca. 850)



Perzijski matematik, astronom, geograf in modrec

- imel močan vpliv na uveljavitev decimalnega sistema na zahodu
- prvi sistematično opisal rešitve linearnih in kvadratnih enačb v arabščini
- dolgo veljal za avtorja algebre
- njegovo ime v latinščini: **Algoritmi**

# Razvoj algoritma

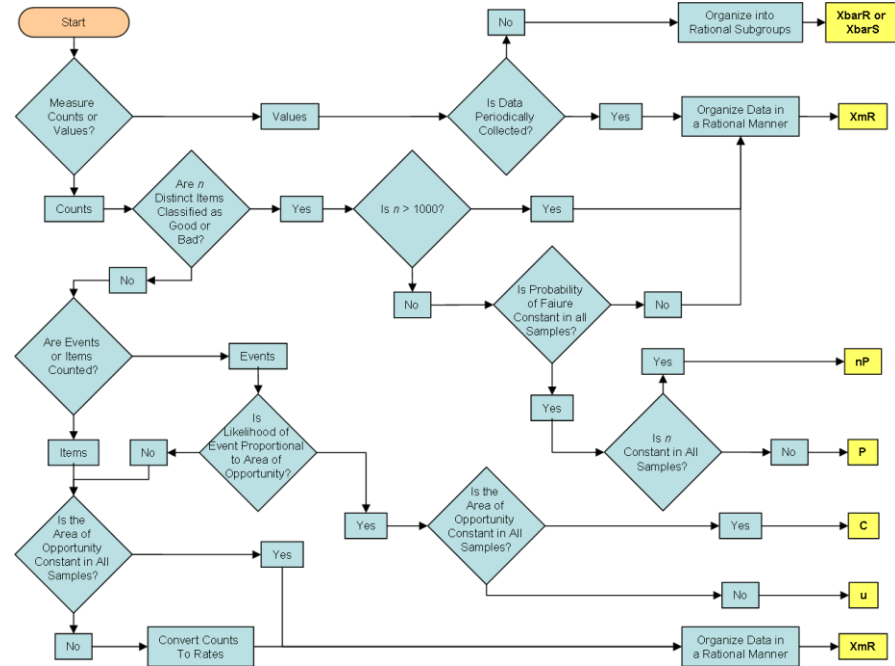


---

1. Kako izraziti algoritem?
2. Kako zasnovati algoritem?
3. Kako analizirati algoritem?
4. Kako preveriti algoritem?

# 1. Kako izraziti algoritem?

- Diagram poteka



- Psevdokoda

**ponavljaljaj**

naključno izberi stanje (naključna rešitev);

**ponavljaljaj**

naredi najboljšo od možnih sprememb;

**dokler** so še izboljšave;

**dokler** ni potekel dodeljen čas;

- Programski jezik

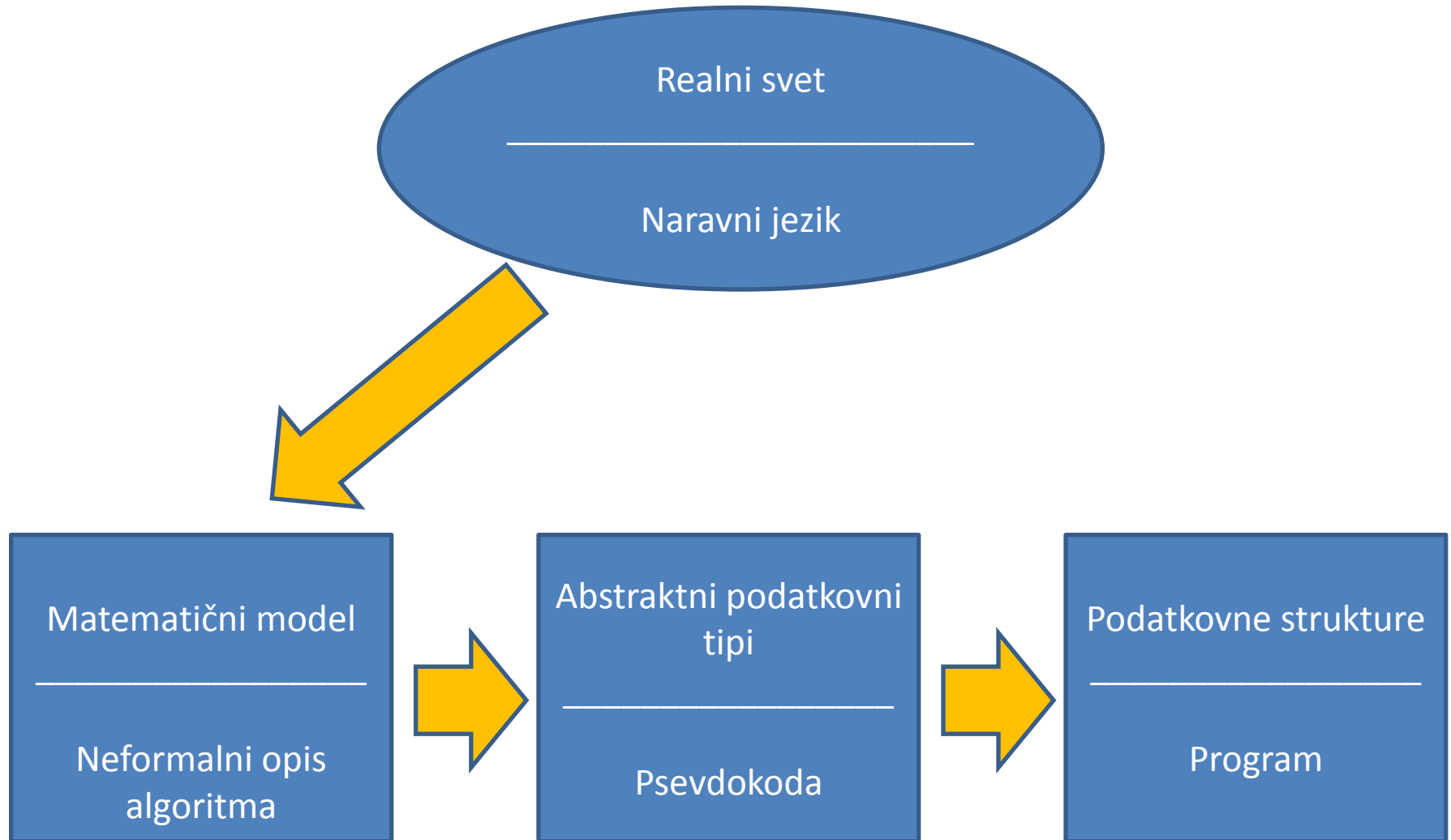
## 2. Kako zasnovati algoritem?



---

1. Razumevanje problema
2. Abstrakcija problema
3. Izbira notacije
4. Razbitje na podprobleme
5. Podobnost med problemi
6. Izbira preiskovalne strategije

## 2. Kako zasnovati algoritem?



# 3. Kako analizirati algoritem?



Analizirati je potrebno:

1. Časovno zahtevnost
2. Prostorsko zahtevnost

Analiza je potrebna že pri zasnovi algoritma!




# 4. Kako preveriti algoritem?

---

## 1. Branje

- Nezanosljivo
- Vidimo tisto, kar naj bi program delal, in ne, kar dejansko dela.

## 2. Testiranje

- Zlobno! 
- Testiraj vse poti, ekstremne vrednosti (max/min/0)
- Testiranje lahko dokaže nepravilnost, ne pa pravilnosti

## 3. Formalno dokazovanje pravilnosti

- Začetni, vmesni in ciljni pogoji, zančne invariante
- Formalno zahtevno
- Se uporablja za kritične aplikacije

```
#include <stdio.h>
int main(void)
{
    int count;
    for (count = 1; count <= 500; count++)
        printf("I will not throw paper airplanes in class.");
    return 0;
}
```

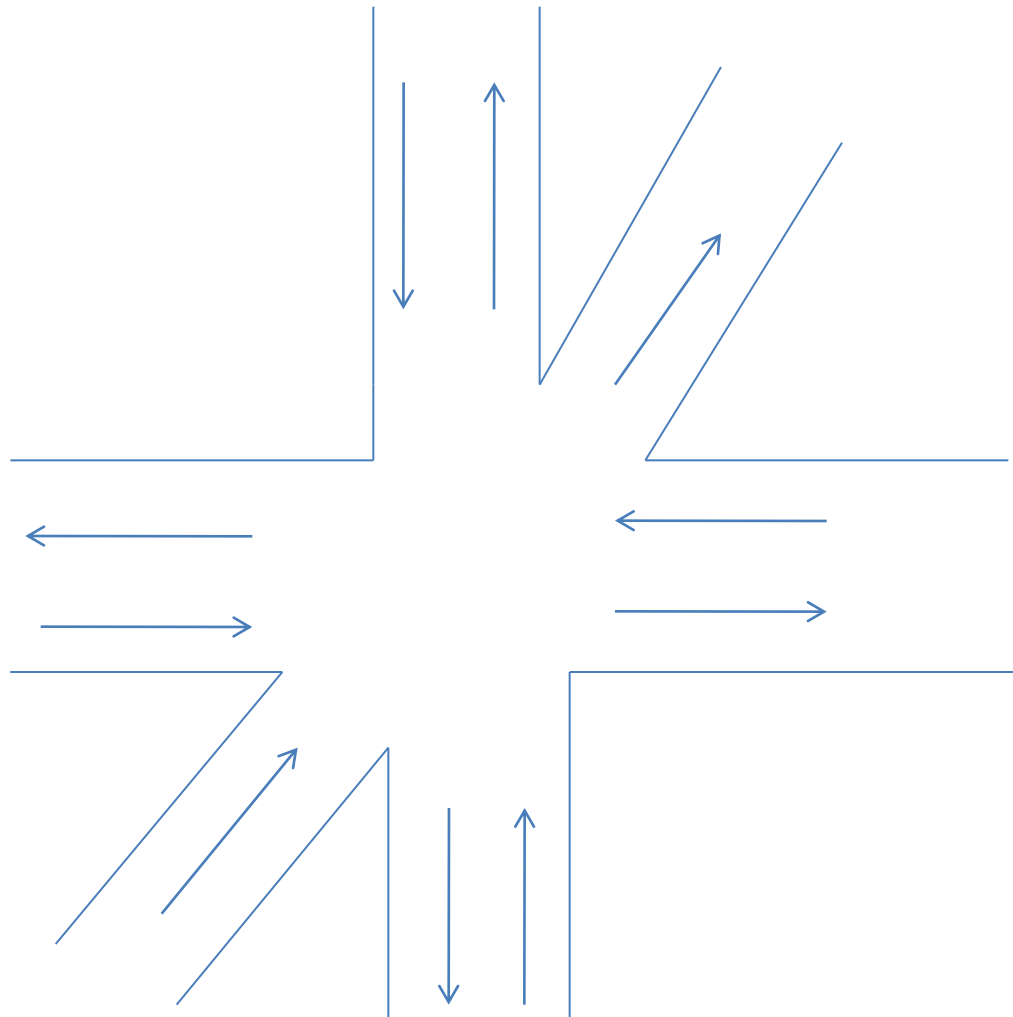
AMEIO 10-3

NICE TRY.



reactor.cc

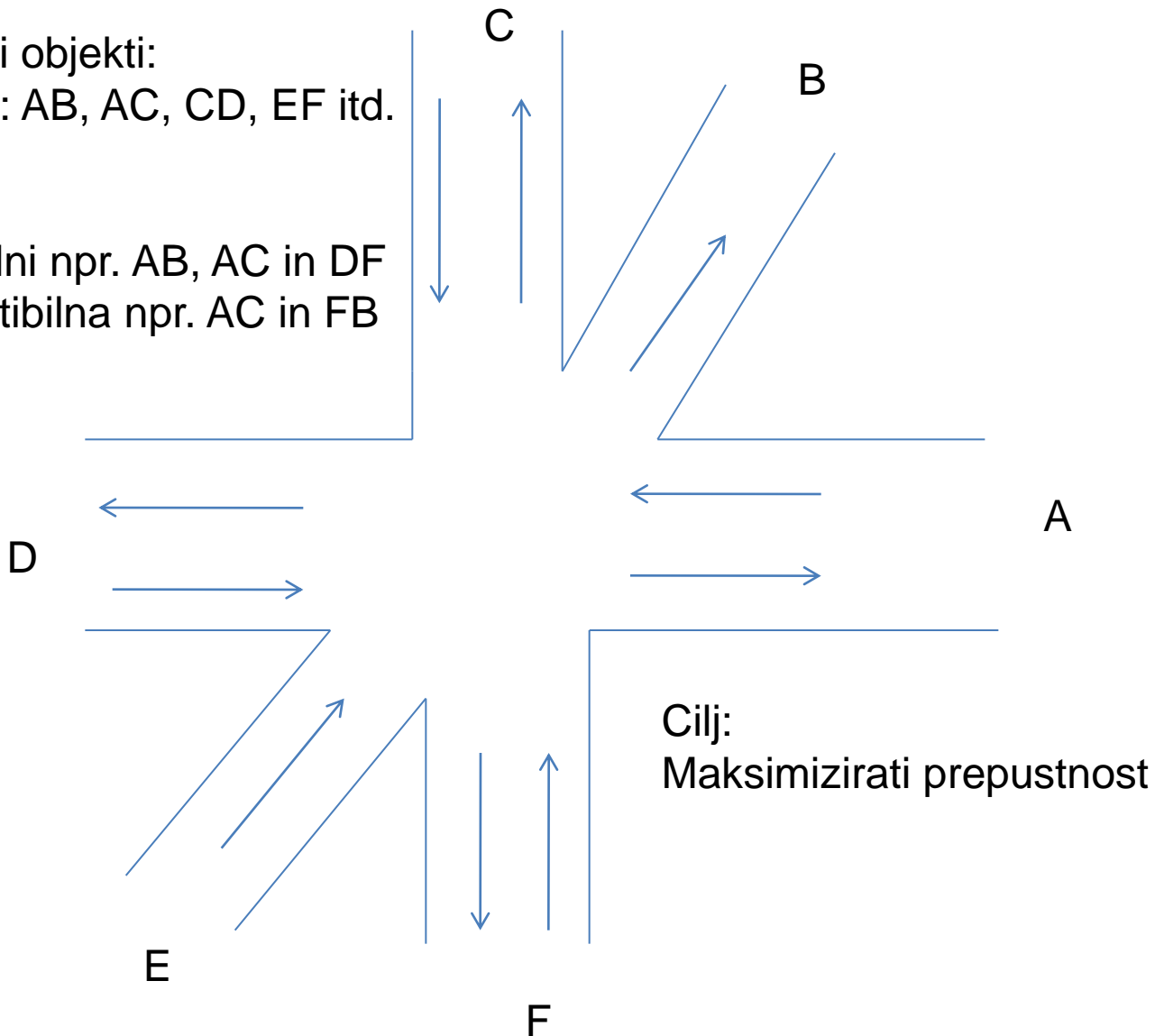
# Primer razvoja algoritma: križišče



# Primer razvoja algoritma: križišče

Pomembni objekti:  
Ovinki-npr: AB, AC, CD, EF itd.

Relacija:  
Kompatibilni npr. AB, AC in DF  
Nekompatibilna npr. AC in FB



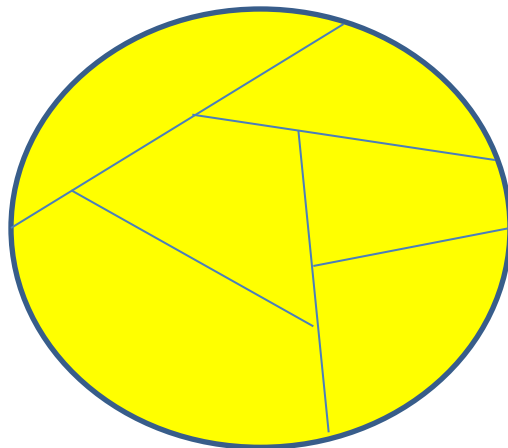
# Primer razvoja algoritma: križišče

Maksimizirati prepustnost:

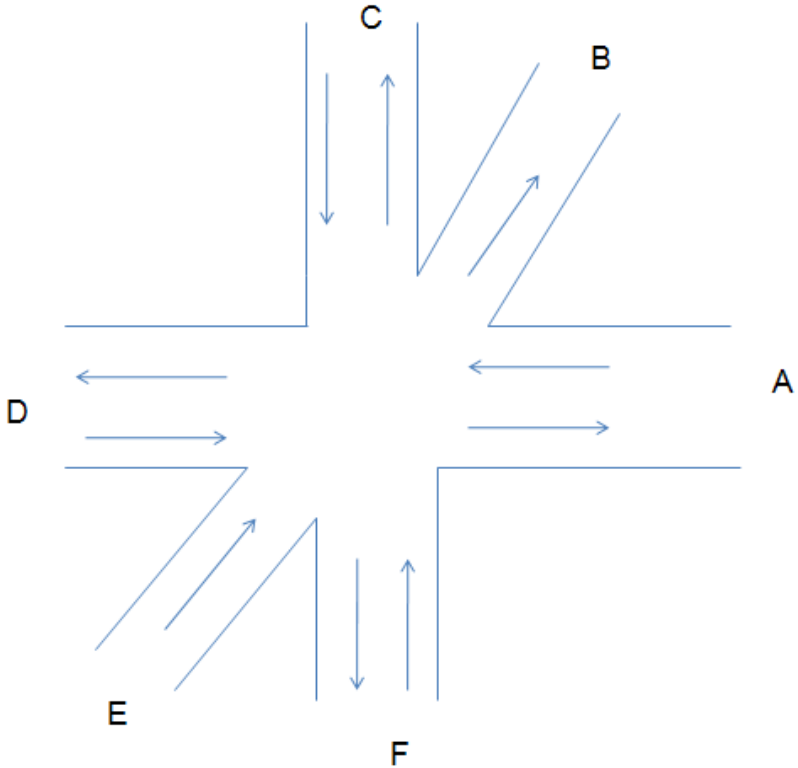
- Dobo čakanja zmanjšati na minimum.
- Cikel preklapljanja semaforjev čim krajši.
- Čim več kompatibilnih ovinkov naj ima naenkrat zeleno luč.

Naloga poiskati (disjunktne) podmnožice kompatibilnih ovinkov, ki imajo lahko hkrati zeleno luč.

Želimo, da je teh podmnožic čim manj.

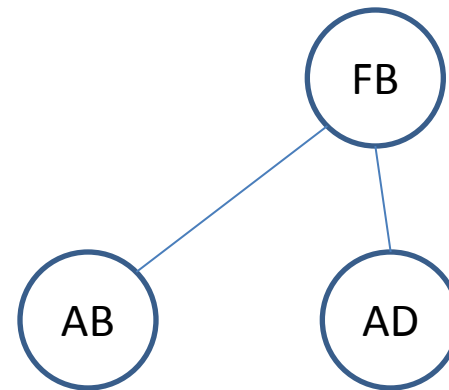


# Primer razvoja algoritma: križišče

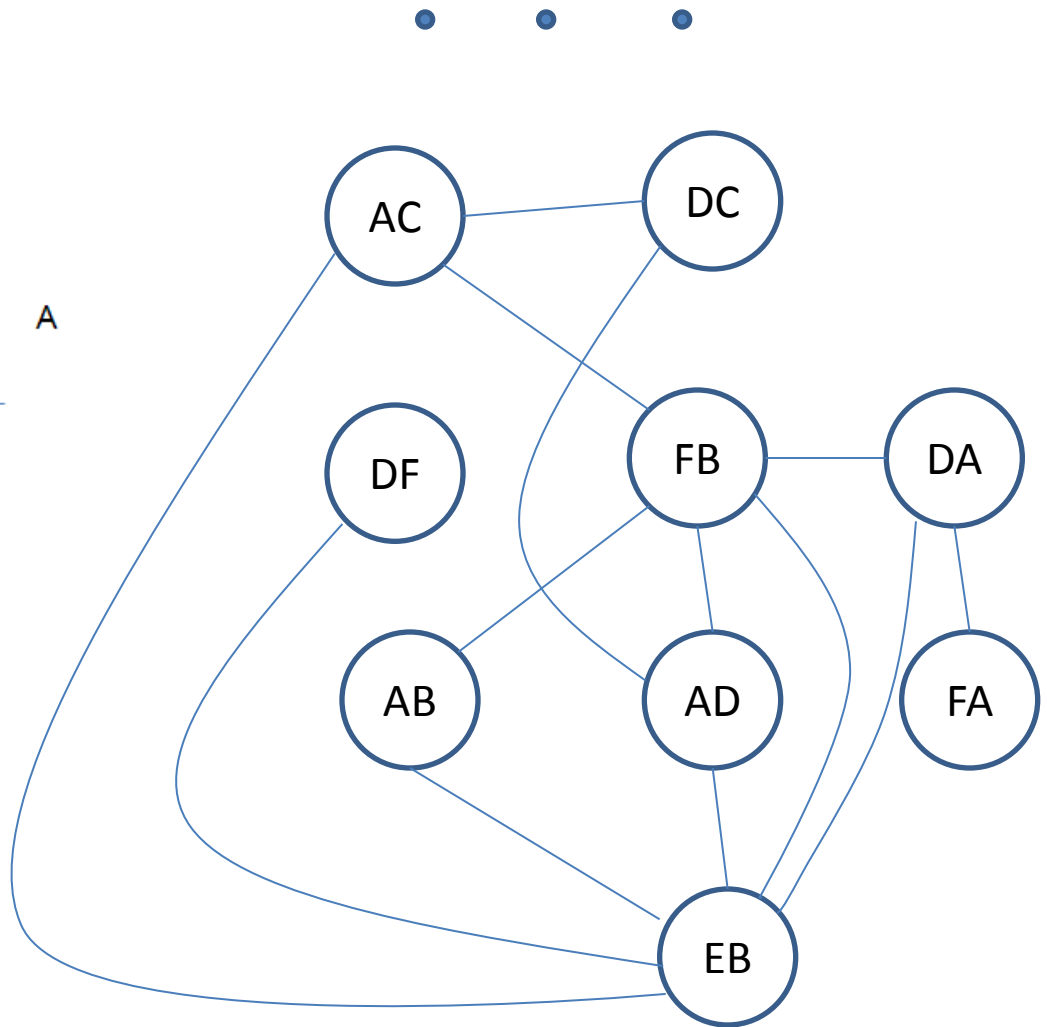
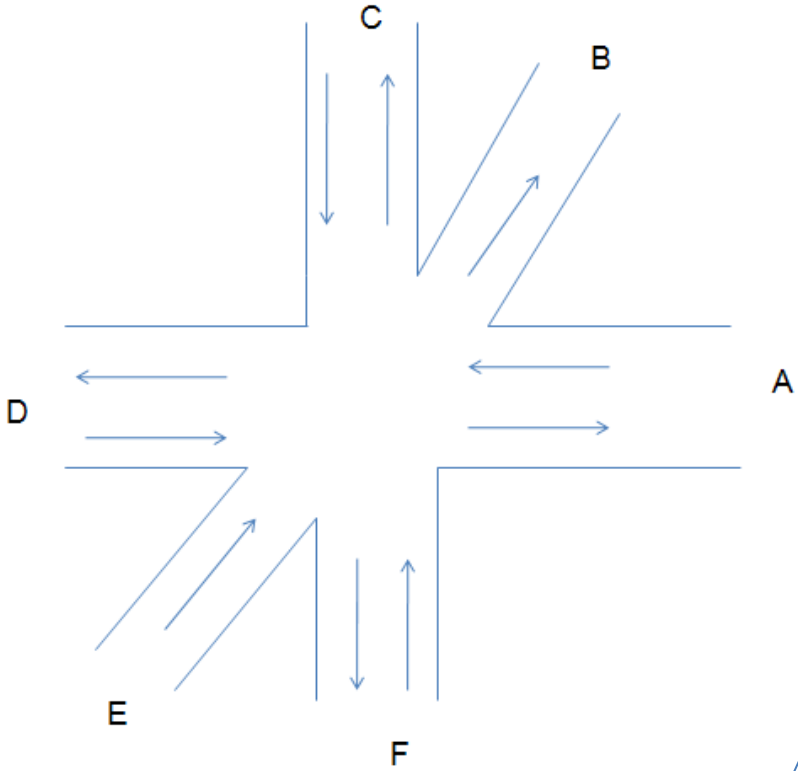


Izbira formalnega sistema:

Binarne relacije pogosto predstavljamo z grafi.



# Primer razvoja algoritma: križišče



# Primer razvoja algoritma: križišče

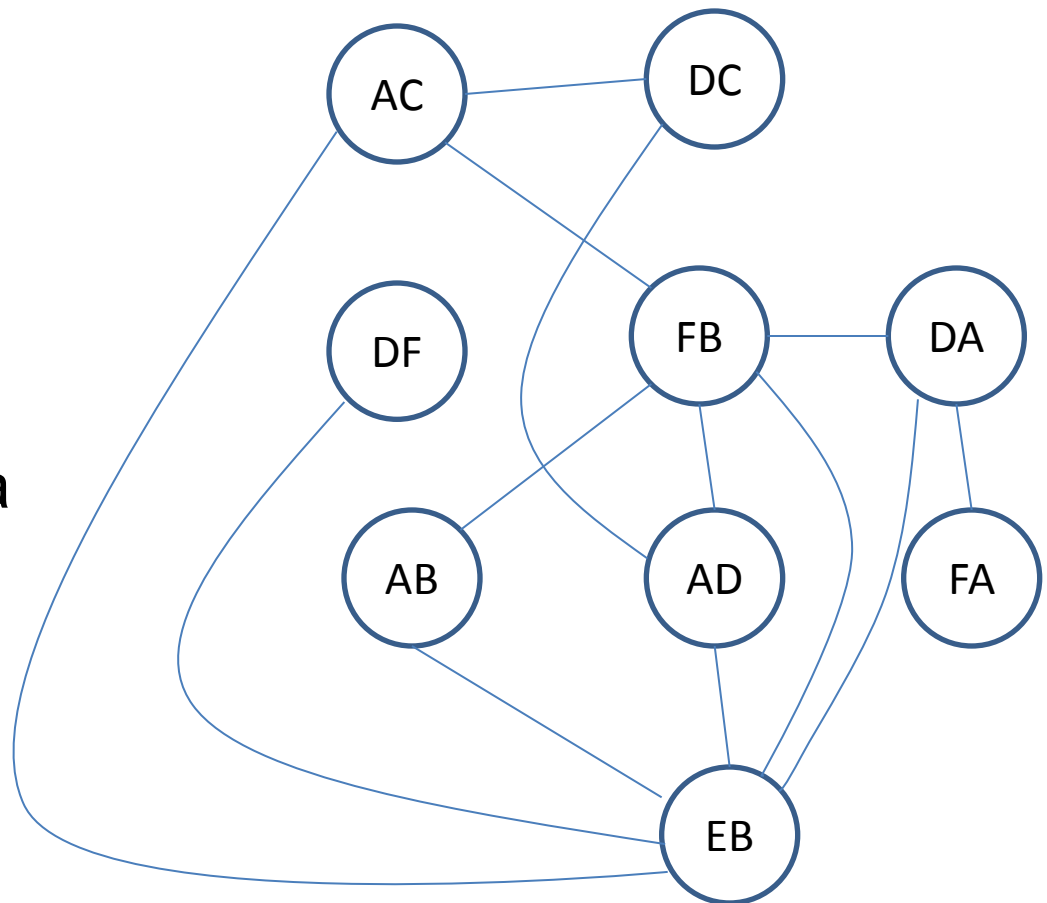


1. Razumevanje problema
2. Abstrakcija problema
3. Izbira notacije
4. Razbitje na podprobleme
5. Podobnost med problemi
6. Izbira preiskovalne strategije



# Primer razvoja algoritma: križišče

Podmnožica kompatibilnih ovinkov:  
vozlišča, ki med seboj niso povezana.



**Problem barvanja grafa:**

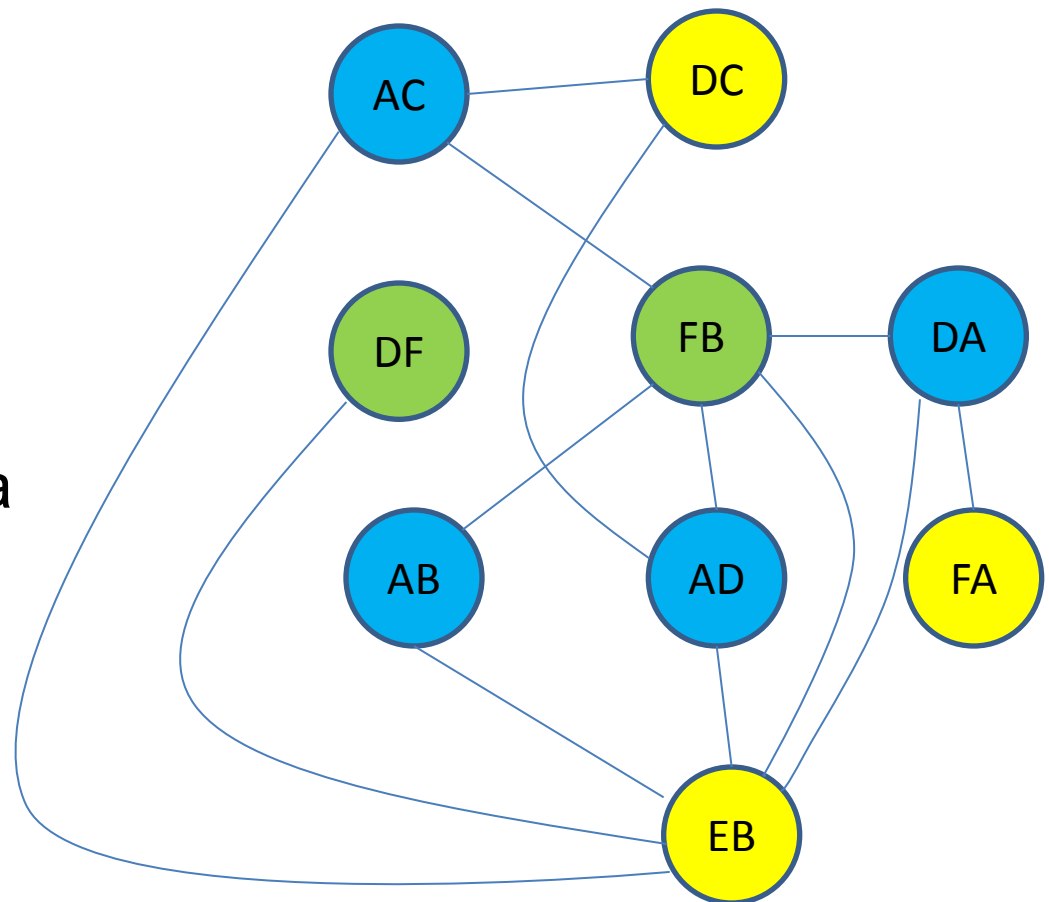
Pobarvaj vozlišča tako, da nobeni dve sosednji vozlišči nista pobarvani z isto barvo.

**barva = podmnožica**

Uporabi čim manj barv!

# Primer razvoja algoritma: križišče

Podmnožica kompatibilnih ovinkov:  
vozlišča, ki med seboj niso povezana.



Problem barvanja grafa:

Pobarvaj vozlišča tako, da nobeni dve sosednji vozlišči nista pobarvani z isto barvo.

barva = podmnožica

Uporabi čim manj barv!

# Primer razvoja algoritma: križišče

Problem barvanja grafa je NP poln problem.



Zadovoljiti se moramo s približno rešitvijo.

Uporabili bomo požrešno strategijo iskanja:

**ponavlajaj**

izberi barvo;

**za vsako** nepobarvano vozlišče

**če** ga lahko pobarvaš, ga pobarvaj;

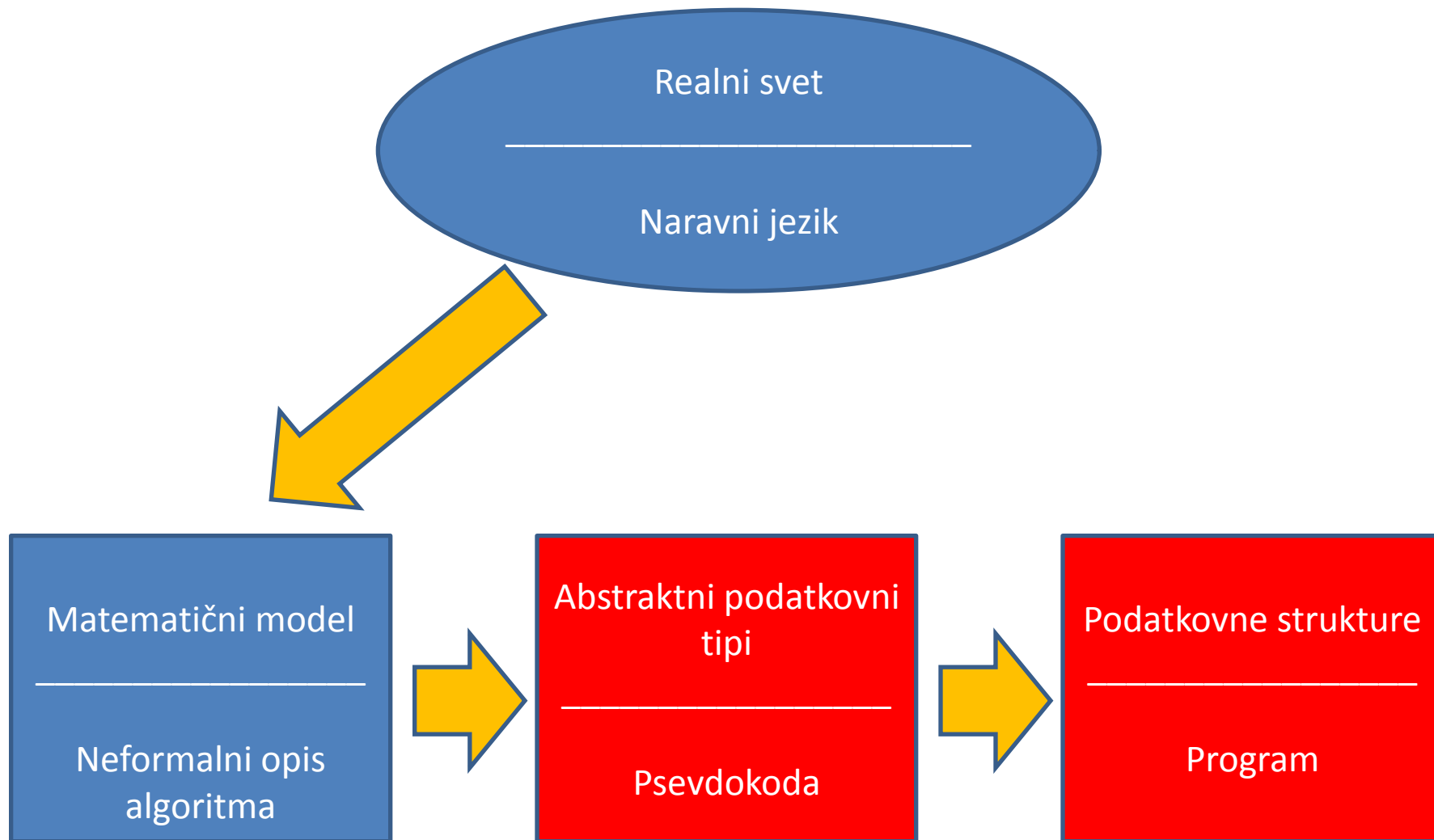
**dokler** niso vsa vozlišča pobarvana;

# Primer razvoja algoritma: križišče



1. Razumevanje problema
2. Abstrakcija problema
3. Izbira notacije
4. Razbitje na podprobleme
5. Podobnost med problemi
6. Izbira preiskovalne strategije

# Primer razvoja algoritma: križišče



# Primer razvoja algoritma: križišče



**ponavljaj**

izberi barvo;

**za vsako** nepobarvano vozlišče

**če** ga lahko pobarvaš, ga pobarvaj;

**dokler** niso vsa vozlišča pobarvana;

Potrebujemo:

- ADT graf
- ADT seznam

# Primer razvoja algoritma: križišče



**GRAPH** - graf z operacijami:

- **FIRST\_UNCOLORED( $G$ )** - vrne prvo nepobarvano vozlišče v grafu  $G$ , če tako vozlišče obstaja, sicer vrne *null*.
- **NEXT\_UNCOLORED( $G, v$ )** - vrne naslednje nepobarvano vozlišče danega vozlišča  $v$  v grafu  $G$ .
- **EDGE( $G, v, w$ )** - vrne *true*, če sta v grafu  $G$  vozlišči  $v$  in  $w$  povezani .
- **MARK( $G, v$ )** - označi, da je vozlišče  $v$  v grafu  $G$  pobarvano.

# Primer razvoja algoritma: križišče

LIST - seznam z operacijami:

- $\text{MAKENULL}(L)$  - naredi seznam  $L$  prazen.
- $\text{FIRST}(L)$  - vrne položaj prvega elementa v seznamu  $L$ . Če je seznam prazen, vrne  $\text{END}(L)$ .
- $\text{NEXT}(p, L)$  - vrne naslednji položaj položaja  $p$ . Če je položaj  $p$  položaj zadnjega elementa v seznamu, potem funkcija vrne isto vrednost kot funkcija  $\text{END}(L)$ . Če seznam nima položaja  $p$ , rezultat ni definiran.
- $\text{END}(L)$  - vrne položaj, ki sledi zadnjemu elementu v seznamu. Ta položaj ni del seznama.
- $\text{OVEREND}(p, L)$  - vrne true, če je položaj  $p$  enak  $\text{END}(L)$ , sicer false.
- $\text{RETRIEVE}(p, L)$  - vrne element na položaju  $p$  v seznamu  $L$ . Rezultat je nedefiniran, če seznam  $L$  nima položaja  $p$ .
- $\text{INSERT}(x, L)$  - vstavi element  $x$  na poljuben položaj v seznam  $L$ . Izbrani položaj je odvisen od posamezne implementacije.



# Primer razvoja algoritma: križišče

Zapisali bomo del algoritma...

**ponavlja**

izberi barvo;

**za vsako** nepobarvano vozlišče

**če** ga lahko pobarvaš, ga pobarvaj;

**dokler** niso vsa vozlišča pobarvana;

# Primer razvoja algoritma: križišče

```
public void greedy(ColGraph g, List newClr) {
    boolean found //ali vozlisce v meji na ze pobarvano vozlisce
    Vertex v, w ;

    newClr.makeNull() ;
    v = g.firstUncolored();
    while (v != null) {
        found = false;
        for (Object pos = newClr.first() ; ! newClr.overEnd(pos) ;
            pos = newClr.next(pos)) {
            w = (Vertex)newClr.retrieve(pos);
            if (g.edge(v, w))
                found = true;
        } // for
        if (! found) {
            g.mark(v);
            newClr.insert(v) ;
        } // if
        v = g.nextUncolored(v) ;
    } // while v
} // greedy
```

# Primer razvoja algoritma: križišče

```
public void greedy(ColGraph g, List newClr) {  
    boolean found //ali vozlisce v meji na ze pobarvano vozlisce  
    Vertex v, w ;  
  
    newClr.makeNull() ;  
    v = g.firstUncolored(); } O(1)  
  
    while (v != null) {  
        found = false;  
        for (Object pos = newClr.first() ; ! newClr.overEnd(pos) ;  
            pos = newClr.next(pos)) {  
            w = (Vertex)newClr.retrieve(pos);  
            if (g.edge(v, w)) } O(1)  
                found = true;  
        } // for  
        if (! found) {  
            g.mark(v);  
            newClr.insert(v) ; } O(1)  
        } // if  
        v = g.nextUncolored(v) ;  
    } // while v  
} // greedy
```

$O(n^2)$

$O(n)$

$O(1)$

$O(1)$

$O(1)$