

5.3.1

- $x_1=0 \ x_2=0 \ x_3=0$ ali $x_1=1 \ x_2=1 \ x_3=1$

- $x_1=0 \ x_2=1 \ x_3=0$ ali $x_1=0 \ x_2=1 \ x_3=1$

- $x_1=0 \ x_2=1 \ x_3=0$

- ni rešitve

OK.

5.3.2

1.) Vzamemo prvi stavek ki še ni zadovoljen in naštimamo prvo spremenljivko tako da bo true.

2.) Pogledamo še ostale stavke v katerih naštimana spremenljivka nastopa in jo ustrezno nastavimo.

3.) Pri dobljenih stavkih kjer naštimana vrednost povzroči false, naštimamo drugo spremenljivko tako da bo true.

4.) Če to povzroči da nek izraz postane v celoti false gremo celoten postopek od začetka le da tokrat pri prvem stavku ki še ni zadovoljen izberemo drugo spremenljivko in jo naštimamo tako da bo true. Če zopet naletimo na izraz, ki je v celoti false rešitve za ta problem ni. Drugače ponavljamo prvi korak dokler ne naštimamo vseh spremenljivk na ustrezne vrednosti.

Ideja je OK, vendar premalo splošna - manjka psevdokoda, ki za m stavkov in n spremenljivk zadovolji izraz.

5.1.3 V naslednjih izrazih poiščite vse vrednosti spremenljivk x_1 , x_2 in x_3 , ki izračunajo izraz na true.

- $(x_1 \vee \neg x_2) \wedge (x_2 \vee \neg x_3) \wedge (x_3 \vee \neg x_1) \rightarrow x_1 = \text{false}, x_2 = \text{false}, x_3 = \text{false}$ +x1=1,x2=1,x3=1
- $(\neg x_1 \vee x_2) \wedge (x_1 \vee x_2) \wedge (\neg x_2 \vee \neg x_1) \rightarrow x_1 = \text{false}, x_2 = \text{true}, x_3 = \text{false}$ +x1=0,x2=1,x3=1
- $(\neg x_1 \vee x_3) \wedge (x_1 \vee x_2) \wedge (\neg x_3 \vee x_1) \wedge (\neg x_1 \vee \neg x_3) \rightarrow x_1 = \text{false}, x_2 = \text{true}, x_3 = \text{false}$
- $(\neg x_1 \vee x_3) \wedge (x_1 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (x_1 \vee x_3) \rightarrow //$

5.3.2

Na začetku vse spremenljivke nastavimo na npr. **true**.

Vzamemo prvi izraz in sprememjamo vrednosti spremenljivk dokler ne bo izraz enak true.

Potem vzamemo naslednji izraz in preverimo rezultat izraza (s prejšnjo nastavitevjo spremenljivk). Če ni true, potem moramo poiskati neko drugo nastavitev spremenljivk, ki še proizvedejo izid true pri prvem izrazu in hkrati proizvedejo true pri sedanjem izrazu.

Ideja je OK, vendar premalo splošna - manjka psevdokoda, ki za m :

5.3.3

Moja rešitev je navadna brut-force metoda, torej je časovna kompleksnost eksponentna: $O(2^n)$, kjer n je število spremenljivk.

OK

Naloga 5.3

Problem 2-SAT. V študiju ste že spoznali pojma konjunkcij (logični in) in disjunkcije (logični ali) ter iz njiju izvirajoč pojem *konjunktivne oblike izraza*, ki pomeni konjunkcijo disjunkcij spremenljivk. Iskanje takšnih vrednosti posameznih spremenljivk (true, false), da izračunajo celoten izraz v true, če je to mogoče, pravimo *zadovoljitev* izraza (angl. *satisfiability* ali *SAT*). Posebno obliko problema SAT, kjer sta v disjunkcijah le po dve spremenljivki, imenujemo 2-SAT. Sami disjunkciji rečemo *stavek* in ga v splošnem zapišemo (x_1, x_2) - očitno je lahko tako x_1 ali x_2 (ali oba) negiran ali pa ne.

Vprašanja:

5.3.1 V naslednjih izrazih poiščite vse vrednosti spremenljivk x_1 , x_2 in x_3 , ki izračunajo izraz na true.

$$\cdot (x_1 \vee \neg x_2) \wedge (x_2 \vee \neg x_3) \wedge (x_3 \vee \neg x_1)$$

x_1	x_2	x_3	$x_1 \vee \neg x_2$	\wedge	$x_2 \vee \neg x_3$	\wedge	$x_3 \vee \neg x_1$
0	0	0	1	1	1	1	1
0	0	1	1	0	0	0	1
0	1	0	0	0	1	0	1
0	1	1	0	0	1	0	1
1	0	0	1	1	1	0	0
1	0	1	1	0	0	0	1
1	1	0	1	1	1	0	0
1	1	1	1	1	1	1	1

Pravilne vrednosti:

1. $X_1=\text{true}; X_2=\text{true}; X_3=\text{true}$.
2. $X_1=\text{false}; X_2=\text{false}; X_3=\text{false}$

$$\cdot (\neg x_1 \vee x_2) \wedge (x_1 \vee x_2) \wedge (\neg x_2 \vee \neg x_1)$$

x_1	x_2	x_3	$\neg x_1 \vee x_2$	\wedge	$x_1 \vee x_2$	\wedge	$\neg x_2 \vee \neg x_1$
0	0	0	1	0	0	0	1
0	0	1	1	0	0	0	1
0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	1
1	0	0	0	0	1	0	1
1	0	1	0	0	1	0	1
1	1	0	1	1	1	0	0
1	1	1	1	1	1	0	0

Pravilne vrednosti:

1. $X_1=\text{false}; X_2=\text{true}; X_3=\text{false}$.
2. $X_1=\text{false}; X_2=\text{true}; X_3=\text{true}$.

$$\cdot (\neg x_1 \vee x_3) \wedge (x_1 \vee x_2) \wedge (\neg x_3 \vee x_1) \wedge (\neg x_1 \vee \neg x_3)$$

x1	x2	x3	$\neg x_1 \vee x_3$	\wedge	$x_1 \vee x_2$	\wedge	$\neg x_3 \vee x_1$	\wedge	$\neg x_1 \vee \neg x_3$
0	0	0	1	0	0	0	1	0	1
0	0	1	1	1	0	0	0	0	1
0	1	0	1	1	1	1	1	1	1
0	1	1	1	1	1	0	0	0	1
1	0	0	0	0	1	0	1	0	1
1	0	1	1	1	1	1	1	0	0
1	1	0	0	0	1	0	1	0	1
1	1	1	1	1	1	1	1	0	0

Pravilne vrednosti:

1. $X1=false; X2=true; X3=false;$

$$\cdot (\neg x_1 \vee x_3) \wedge (x_1 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (x_1 \vee x_3)$$

x1	x2	x3	$\neg x_1 \vee x_3$	\wedge	$x_1 \vee \neg x_3$	\wedge	$\neg x_1 \vee \neg x_3$	\wedge	$x_1 \vee x_3$
0	0	0	1	1	1	1	1	0	0
0	0	1	1	0	0	0	1	0	1
0	1	0	1	1	1	1	1	0	0
0	1	1	1	0	0	0	1	0	1
1	0	0	0	0	0	0	1	0	1
1	0	1	1	1	0	0	0	0	1
1	1	0	0	0	0	0	1	0	1
1	1	1	1	1	0	0	0	0	1

Pravilne vrednosti:

Pri tem primeru ne moremo nastaviti spremenljivk tako, da bi dobili true.

OK.

5.3.2 Peter Zmeda je dobil za nalogo, da napiše algoritem, ki reši problem 2-SAT. Nekaj časa je gledal problem, potem brskal po knjigi in zapiskih ter našel magično besedo - *požrešna metoda*. Tako je pričel pisati idejo algoritma nekako takole:

1. Na začetku ima vsaka spremenljivka bodisi vrednost true ali false.
2. Vzemi prvi stavki (x_1, x_2), ki še ni nezadovoljen, ter določi vrednost spremenljivke x_1 (ali x_2) tako, da se stavki izračuna na true - postane zadovoljen.
3. Nastavi še vrednosti spremenljivke v vseh ostalih stavkih, kjer nastopa x_1 (ali x_2), in izračunaj posamezne stavke: nekateri bodo postali zadovoljeni.
4. Ponovi korak 2.

Poglejmo si primer $(x_1 \vee \neg x_2) \wedge (x_2 \vee \neg x_1)$:

- na začetku sta oba stavki $(x_1 \vee \neg x_2)$ in $(x_2 \vee \neg x_1)$ še nedoločena in, recimo, začnimo pri stavku $(x_1 \vee \neg x_2)$: naj bo x_1 true ter stavki $(x_1 \vee \neg x_2)$ postane zadovoljen;
- nato v stavku $(x_2 \vee \neg x_1)$ dobimo $(x_2 \vee \text{false})$, kar pomeni, da moramo nastaviti tudi x_2 na true;
- preverimo prvi stavki in smo uspeli najti pravo nastavitev.

Osnovni iziv Petrovega pristopa je v 3. koraku, saj bodo nekateri stavki postali zadovoljeni, kar je super; pri nekaterih stavkih bo ena spremenljivka še nenastavljen, kar se bo rešilo kasneje; največji problem pa so stavki, kjer sta obe spremenljivki določeni in stavki se izračuna na false. Kaj pa sedaj?

Pomagajte Petru, da sestavi celoten algoritem zasnovan na njegovi ideji. Napišite psevdokodo algoritma.

Algoritmom:

1. all variables to false
2. foreach expression of variables do:
 - 2/1. expression <- String
 3. while expression != True do:
 - 3/1. Take first sentence of expression and turn all variables till they are not True
 - 3/2. foreach sentence do:
 - 3/2/1. Turn all variables to same value like in 3/1
 - 3/2/2. if step from 3/2/1 is False do:
 - 3/2/2/1. write in variable's String a new state
 - 3/2/2/2. repeat from step 2.

OK

5.3.3 Kakšna je časovna zahtevnost vašega algoritma glede na število spremenljivk?

V povprečju bomo rešitev dobili v 2^n korakov.

OK

5.3.4 Ker je domača naloga pri poglavju o grafih, se da izraz preoblikovati v graf? Kaj bi bile povezave in kaj vozlišča? Poisci algoritmom, ga opišite, utemeljite njegovo pravilnost in analizirajte njegovo časovno zahtevnost.

Graf iz domače naloge bi pretvorili tako, da v vozlišča bi shranil true vrednost in povezave bi pa predstavili kot implementacijo med vozlišči. Časovna zahtevnost bi bila v povprečju enaka.

Graf ne bo pravi. Za rešitev poglejte v Sedgewick - Algorithms 4th ed., str. 600. as. zahtevnost pa je linearna.

5.3.1

$(x_1 \vee \neg x_2) \wedge (x_2 \vee \neg x_3) \wedge (x_3 \vee \neg x_1)$
->rešitev ($x_3=0, x_2=0, x_1=0$), rešitev($x_3=1, x_2=1, x_1=1$)

$(\neg x_1 \vee x_2) \wedge (x_1 \vee x_2) \wedge (\neg x_2 \vee \neg x_1)$
-> rešitev($x_2=1, x_1=0$)

$(\neg x_1 \vee x_3) \wedge (x_1 \vee x_2) \wedge (\neg x_3 \vee x_1) \wedge (\neg x_1 \vee \neg x_3)$
->rešitev($x_3=0, x_2=1, x_1=0$)

$(\neg x_1 \vee x_3) \wedge (x_1 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (x_1 \vee x_3)$
->ni rešitev

OK

5.3.2

For spremenljivka in spremenljivke:

Spremenljivka = false

Ne razumem algoritma. Vzamete vsak stavek, ga izraunat

For stavek in stavki:

izračunaj(stavek) -> while not true

If(izračun in stavek == true)
 Nastavi drugi izračun na true

5.3.3

Časovna zahtevnost algoritma je: n^2

Iz vaše psevdokode je razvidno, da je asovna zahtevn

5.3.4

Vozlišča bi bile spremenljivke v izrazih, povezave pa operacije, kot so konjunkcija, disjunkcija. In kako bi dobili vrednosti spremenljivk?

Mislim, da gre tukaj za Boolean satisfiability problem in ga lahko rešimo z 3-SAT.
Problem reši v $\exp(o(n))$.

Problem 2-SAT je lažji od 3-SAT. Optimizacijski problem slednjega je NP-poln, medtem ko je 2-SAT v P in ga lahko reši

NALOGA 5.3

1. Pravilnost izrazov: najdi x_1 , x_2 in x_3 , da bo izraz resničen (true).

- $(x_1 \vee \neg x_2) \wedge (x_2 \vee \neg x_3) \wedge (x_3 \vee \neg x_1) \rightarrow \text{false} + \text{false} + \text{false}$ ali $\text{true} + \text{true} + \text{true}$
- $(\neg x_1 \vee x_2) \wedge (x_1 \vee x_2) \wedge (\neg x_2 \vee \neg x_1) \rightarrow \text{false} + \text{true} + \text{false}$ ali $\text{false} + \text{true} + \text{true}$
- $(\neg x_1 \vee x_3) \wedge (x_1 \vee x_2) \wedge (\neg x_3 \vee x_1) \wedge (\neg x_1 \vee \neg x_3) \rightarrow \text{false} + \text{true} + \text{false}$
- $(\neg x_1 \vee x_3) \wedge (x_1 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (x_1 \vee x_3) \rightarrow \text{ne obstaja}$

*program, ki sem ga uporabil za preverjanje je na koncu dokumenta.

OK

2. Obstaja način vračanja, ko pridemo do take situacije. Žal pa je omejen le na zadnjo spremembo. Imenuje se »Limited backtracking«, opis rešitve a najdemo na spletni strani <http://en.wikipedia.org/wiki/2-satisfiability>.

Opis pravi nekako tako: če najdemo točko, ko imamo nastavljene spremenljivke, naš stavek pa je napačen, potem lahko razveljavimo zadnjo spremembo in se vrnemo na točko, ki smo določili vrednost spremenljivke. Če pa smo se že vrnili in kljub spremembì ni nova rešitev boljša, potem program vrne, da izjava ni rešljiva.

Algoritem naj bi se izvedel v linearinem času.

Manjka psevdokoda in razumevanje algoritma. Pod

e imate prisotno sestopanje (backtracking), se program ne bo rešil v linearinem asu, ampak v eksponentner

PROGRAM ZA PREVERBO 5.3.1

```

package vaja.aps2;

public class UganiTrueFalse {

    public static void main(String[] args) {

        for(int izraz = 1; izraz < 5; izraz++ ){
            System.out.println("IZRAZ "+izraz+":");
            boolean x1, x2, x3;
            x1=x2=x3=false;
            for (int i = 0; i < 2; i++){
                if (i == 1)
                    x1 = true;
                for (int j = 0; j < 2; j++){
                    if (j == 1)
                        x2 = true;
                    if(j == 0)
                        x2 = false;
                    for (int k = 0; k < 2; k++){
                        if (k == 1)
                            x3 = true;
                        if(k == 0)
                            x3 = false;
                        if(najdiIzraz1(izraz,x1,x2,x3))
                            System.out.println("x1:"+x1+
                                x2:"+x2+" x3:"+x3);
                        else
                            System.out.println("");
                    }
                }
            }
        }

        public static boolean najdiIzraz1(int izraz, boolean x1, boolean x2, boolean
x3){

            if(izraz == 1){
                System.out.print("izraz"+izraz+": (x1 || !x2) && (x2 || !x3) &&
(x3 || !x1) ");
                return (x1 || !x2) && (x2 || !x3) && (x3 || !x1);
            }
            if(izraz == 2){
                System.out.print("izraz"+izraz+": (!x1 || x2) && (x1 || x2) &&
(!x2 || !x1) ");
                return (!x1 || x2) && (x1 || x2) && (!x2 || !x1);
            }
            if(izraz == 3){
                System.out.print("izraz"+izraz+": (!x1 || x3) && (x1 || x2) &&
(!x3 || x1) && (!x1 || !x3) ");
                return (!x1 || x3) && (x1 || x2) && (!x3 || x1) && (!x1 ||
!x3);
            }
            if(izraz == 4){
                System.out.print("izraz"+izraz+": (!x1 || x3) && (x1 || !x3) &&
(!x1 || !x3) && (x1 || x3) ");
                return (!x1 || x3) && (x1 || !x3) && (!x1 || !x3) && (x1 ||
x3);
            }
            return false;
        }
    }
}

```

Naloga 5.3

Problem 2-SAT. V študiju ste že spoznali pojma konjunkcij (logični in) in disjunkcije (logični ali) ter iz njiju izvirajoč pojem *konjunktivne oblike* izraza, ki pomeni konjunkcijo disjunkcij spremenljivk. Iskanje takšnih vrednosti posameznih spremenljivk (`true`, `false`), da izračunajo celoten izraz v `true`, če je to mogoče, pravimo *zadovoljitev* izraza (angl. *satisfiability* ali *SAT*). Posebno obliko problema SAT, kjer sta v disjunkcijah le po dve spremenljivki, imenujemo 2-SAT. Sami disjunkciji rečemo *stavek* in ga v splošnem zapišemo (x_1, x_2) - očitno je lahko tako x_1 ali x_2 (ali oba) negiran ali pa ne.

Vprašanja:

5.3.1 V naslednjih izrazih poiščite vse vrednosti spremenljivk x_1 , x_2 in x_3 , ki izračunajo izraz na true.

1 = true

0 = false

- $(x_1 \vee \neg x_2) \wedge (x_2 \vee \neg x_3) \wedge (x_3 \vee \neg x_1)$

x1	x2	x3
1	1	1
0	0	0

- $(\neg x_1 \vee x_2) \wedge (x_1 \vee x_2) \wedge (\neg x_2 \vee \neg x_1)$

x1	x2
0	1

- $(\neg x_1 \vee x_3) \wedge (x_1 \vee x_2) \wedge (\neg x_3 \vee x_1) \wedge (\neg x_1 \vee \neg x_3)$

x1	x2	x3
0	1	0

- $(\neg x_1 \vee x_3) \wedge (x_1 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (x_1 \vee x_3)$

Ni rešitev.

Tretji stavek pravi da morata x_1 in x_3 biti različnih vrednosti ali pa oba "false".

Četrти pravi da morata biti raličnih vrednosti ali pa oba "true". Iz the dveh vemo, da morata biti x_1 in x_3 različnih vrednosti, vendar bo to povzročilo "false" stanje prvega ali drugega stavka.

OK

5.3.2 Peter Zmeda je dobil za nalogu, da napiše algoritmom, ki reši problem 2-SAT. Nekaj časa je gledal problem, potem brskal po knjigi in zapiskih ter našel magično besedo - *požrešna metoda*. Tako je pričel pisati idejo algoritma nekako takole:

1. Na začetku ima vsaka spremenljivka bodisi vrednost `true` ali `false`.
2. Vzemi prvi stavek (x_1, x_2) , ki še ni nezadovoljen, ter določi vrednost spremenljivke x_1 (ali x_2) tako, da se stavek izračuna na `true` - postane zadovoljen.
3. Nastavi še vrednosti spremenljivke v vseh ostalih stavkih, kjer nastopa x_1 (ali x_2), in izračunaj posamezne stavke: nekateri bodo postali zadovoljeni.
4. Ponovi korak 2.

Poglejmo si primer $(x_1 \vee \neg x_2) \wedge (x_2 \vee \neg x_1)$:

- na začetku sta oba stavka $(x_1 \vee \neg x_2)$ in $(x_2 \vee \neg x_1)$ še nedoločena in, recimo, začnimo pri stavku $(x_1 \vee \neg x_2)$: naj bo x_1 `true` ter stavek $(x_1 \vee \neg x_2)$ postane zadovoljen;
- nato v stavku $(x_2 \vee \neg x_1)$ dobimo $(x_2 \vee \text{false})$, kar pomeni, da moramo nastaviti tudi x_2 na `true`;
- preverimo prvi stavek in smo uspeli najti pravo nastavitev.

Osnovni izliv Petrovega pristopa je v 3. koraku, saj bodo nekateri stavki postali zadovoljeni, kar je super; pri nekaterih stavkih bo ena spremenljivka še nenastavljena, kar se bo rešilo kasneje; največji problem pa so stavki, kjer sta obe spremenljivki določeni in stavek se izračuna na `false`. Kaj pa sedaj?

Pomagajte Petru, da sestavi celoten algoritmom zasnovan na njegovi ideji. Napišite psevdokodo algoritma.

```
//iz = iz
//v = vrednosti
//s = stavek
//origIz = originalen izraz
origIz
fun res(iz){//GLAVNA METODA
    if origIz==null
        origIz = iz
    endif
    //pogleda predzname vrednosti v stavku
    if iz[0][0]>0 and iz[0][1]>0 then //x & y
        if z.length == 1 then
            v[iz[0][0]]=true
            v[iz[0][1]]=true
            if preveri(origIz,v) == true then
                return v
            endif
            v[iz[0][0]]=false
            v[iz[0][1]]=true
            if preveri(origIz,v) == true then
                return v
            endif
            v[iz[0][0]]=true
            v[iz[0][1]]=false
            if preveri(origIz,v) == true then
                return v
            endif
        else
            s = iz[0]
            iz.remove(0)
            v[s[0]]=true
            v[s[1]]=true
            v2=res2(iz,v)
            iz.add(s)
            if preveri(origIz,v2) == true then
                return v2
            endif
        endif
    else
        if iz[0][0]>0 and iz[0][1]<0 then //x & ~y
            if z.length == 1 then
                v[iz[0][0]]=true
                v[iz[0][1]]=false
                if preveri(origIz,v) == true then
                    return v
                endif
                v[iz[0][0]]=false
                v[iz[0][1]]=true
                if preveri(origIz,v) == true then
                    return v
                endif
            else
                s = iz[0]
                iz.remove(0)
                v[s[0]]=true
                v[s[1]]=false
                v2=res2(iz,v)
                iz.add(s)
                if preveri(origIz,v2) == true then
                    return v2
                endif
            endif
        else
            if iz[0][0]<0 and iz[0][1]>0 then //~x & y
                if z.length == 1 then
                    v[iz[0][0]]=false
                    v[iz[0][1]]=true
                    if preveri(origIz,v) == true then
                        return v
                    endif
                    v[iz[0][0]]=true
                    v[iz[0][1]]=true
                    if preveri(origIz,v) == true then
                        return v
                    endif
                else
                    s = iz[0]
                    iz.remove(0)
                    v[s[0]]=false
                    v[s[1]]=true
                    v2=res2(iz,v)
                    iz.add(s)
                    if preveri(origIz,v2) == true then
                        return v2
                    endif
                endif
            else
                if iz[0][0]<0 and iz[0][1]<0 then //~x & ~y
                    if z.length == 1 then
                        v[iz[0][0]]=false
                        v[iz[0][1]]=false
                        if preveri(origIz,v) == true then
                            return v
                        endif
                    else
                        s = iz[0]
                        iz.remove(0)
                        v[s[0]]=false
                        v[s[1]]=false
                        v2=res2(iz,v)
                        iz.add(s)
                        if preveri(origIz,v2) == true then
                            return v2
                        endif
                    endif
                endif
            endif
        endif
    endif
}
```

```

v[s[0]]=false
v[s[1]]=true
v2=res2(iz,v)
iz.add(s)
if preveri(origIz,v2) == true then
    return v2
endif
v[s[0]]=true
v[s[1]]=false
v2=res2(iz,v)
iz.add(s)
if preveri(origIz,v2) == true then
    return v2
endif
endif
else if iz[0][0]<0 and iz[0][1]>0 then // -x & y
    if z.length == 1 then
        v[iz[0][0]]=false
        v[iz[0][1]]=false
        if preveri(origIz,v) == true then
            return v
        endif
        v[iz[0][0]]=false
        v[iz[0][1]]=true
        if preveri(origIz,v) == true then
            return v
        endif
        v[iz[0][0]]=true
        v[iz[0][1]]=true
        if preveri(origIz,v) == true then
            return v
        endif
    else then
        s = iz[0]
        iz.remove(0)
        v[s[0]]=false
        v[s[1]]=false
        v2=res2(iz,v)
        iz.add(s)
        if preveri(origIz,v2) == true then
            return v2
        endif
        v[s[0]]=false
        v[s[1]]=true
        v2=res2(iz,v)
        iz.add(s)
        if preveri(origIz,v2) == true then
            return v2
        endif
        v[s[0]]=true
        v[s[1]]=true
        v2=res2(iz,v)
        iz.add(s)
        if preveri(origIz,v2) == true then
            return v2
        endif
    endif
else if iz[0][0]>0 and iz[0][1]<0 then // x & -y
    if z.length == 1 then
        v[iz[0][0]]=false
        v[iz[0][1]]=false
        if preveri(origIz,v) == true then
            return v
        endif
        v[iz[0][0]]=true
        v[iz[0][1]]=false
        if preveri(origIz,v) == true then
            return v
        endif
        v[iz[0][0]]=true
        v[iz[0][1]]=true
        if preveri(origIz,v) == true then
            return v
        endif
    else then
        s = iz[0]
        iz.remove(0)
        v[s[0]]=false
        v[s[1]]=false
        v2=res2(iz,v)
        iz.add(s)
        if preveri(origIz,v2) == true then
            return v2
        endif
        v[s[0]]=true
        v[s[1]]=false
        v2=res2(iz,v)
        iz.add(s)
        if preveri(origIz,v2) == true then
            return v2
        endif
        v[s[0]]=true
        v[s[1]]=true
        v2=res2(iz,v)
        iz.add(s)
        if preveri(origIz,v2) == true then
            return v2
        endif
    endif
endif
else if iz[0][0]<0 and iz[0][1]<0 then // -x & -y

```

```

if z.length == 1 then
    v[iz[0][0]]=false
    v[iz[0][1]]=false
    if preveri(origIz,v) == true then
        return v
    endif
    v[iz[0][0]]=true
    v[iz[0][1]]=false
    if preveri(origIz,v) == true then
        return v
    endif
    v[iz[0][0]]=false
    v[iz[0][1]]=true
    if preveri(origIz,v) == true then
        return v
    endif
else then
    s = iz[0]
    iz.remove(0)
    v[s[0]]=false
    v[s[1]]=false
    v2=res2(iz,v)
    iz.add(s)
    if preveri(origIz,v2) == true then
        return v2
    endif
    v[s[0]]=true
    v[s[1]]=false
    v2=res2(iz,v)
    iz.add(s)
    if preveri(origIz,v2) == true then
        return v2
    endif
    v[s[0]]=false
    v[s[1]]=true
    v2=res2(iz,v)
    iz.add(s)
    if preveri(origIz,v2) == true then
        return v2
    endif
endif
else then
    return "ni rešitev";
endif
}

fun preveri(origIz,v){
    for i in iz.length do
        if iz[i][0]>0 and iz[i][1]>0 then
            if v[iz[i][0]] == false and v[iz[i][1]] == false then
                return false;
            endif
        else if iz[i][0]<0 and iz[i][1]>0 then
            if v[iz[i][0]] == true and v[iz[i][1]] == false then
                return false;
            endif
        else if iz[i][0]>0 and iz[i][1]<0 then
            if v[iz[i][0]] == false and v[iz[i][1]] == true then
                return false;
            endif
        else if iz[i][0]<0 and iz[i][1]<0 then
            if v[iz[i][0]] == true and v[iz[i][1]] == true then
                return false;
            endif
        endif
    endfor
    return true;
}

```

Algoritem ni splošen, ampak je v svojem bistvu tabela vnaprej izraunanih rešitev.

5.3.3 Kakšna je časovna zahtevnost vašega algoritma glede na število spremenljivk?

$$O(n^2)$$

asovna zahtevnost drži. Kako pa ste jo izraunali?

5.3.4 Ker je domača naloga pri poglavju o grafih, se da izraz preoblikovati v graf? Kaj bi bile povezave in kaj vozlišča? Poiščite algoritom, ga opišite, utemeljite njegovo pravilnost in analizirajte njegovo časovno zahtevnost.

Naloga 5.3

Vprašanja:

5.3.1 V naslednjih izrazih poiščite vse vrednosti spremenljivk x_1 , x_2 in x_3 , ki izračunajo izraz na true:

- $(x_1 \vee \neg x_2) \wedge (x_2 \vee \neg x_3) \wedge (x_3 \vee \neg x_1)$

Rešitev: $X_1 = \text{false}$, $X_2 = \text{false}$, $X_3 = \text{false}$;
 $X_1 = \text{true}$, $X_2 = \text{true}$, $X_3 = \text{true}$

- $(\neg x_1 \vee \bar{x}_2) \wedge (\bar{x}_1 \vee x_2) \wedge (\neg x_2 \vee \bar{x}_1)$

Rešitev: $X_1 = \text{false}$, $X_2 = \text{true}$

- $(\neg x_1 \vee x_3) \wedge (x_1 \vee x_2) \wedge (\neg x_3 \vee x_1) \wedge (\neg x_1 \vee \neg x_3)$

Rešitev: $X_1 = \text{false}$, $X_2 = \text{true}$, $X_3 = \text{false}$

- $(\neg x_1 \vee x_3) \wedge (x_1 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (x_1 \vee x_3)$

Rešitev: Izraz ni rešljiv.

OK

5.3.2 Peter Zmeda je dobil za nalogo, da napiše algoritmom, ki reši problem 2-SAT. Nekaj časa je gledal problem, potem brskal po knjigi in zapiskih ter našel magično besedo - *požrešna metoda*. Tako je pričel pisati idejo algoritma nekako takole:

Pomagajte Petru, da sestavi celoten algoritmom zasnovan na njegovi ideji. Napišite psevdokodo algoritma.

algoritmom:

for spremenljivka S in stavek do:

S = false

zapiši vrednost S v njegov »char prostor«

dokler cel izraz ni true:

vzemi prvi stavek, ki še ni true in določi vrednost spremenljivke x_1 ali x_2 tako, da bo izraz postal true

nastavi spremenljivko še v ostalih stavkih, kjer nastopa in jih izračunaj

for stavek S v izraz do:

if S = false then:

zapiši stanja spremenljivk v »char prostor« vsake spremenljivke in

vzemi druge vrednosti

OK. Kaj tono je "char prostor"?

5.3.3 Kakšna je časovna zahtevnost vašega algoritma glede na število spremenljivk?

Časovna zahtevnost algoritma je $O(2^n)$. Do take zahtevnosti pride, ker ima vsaka spremenljivka 2 možni stanji in algoritmom poiščuje v najslabšem primeru uporabiti vsa možna stanja vseh spremenljivk, da bi prišel do rešitve.

OK

5.3.4 Ker je domača naloga pri poglavju o grafih, se da izraz preoblikovati v graf? Kaj bi bile povezave in kaj vozlišča? Poiščite algoritmom, ga opišite, utemeljite njegovo pravilnost in analizirajte njegovo časovno zahtevnost.

Izraz se da preoblikovati v graf. Vozlišča bi bila spremenljivke, povezave pa so implikacije med vozlišči. Pri takih grafih po navadi dobimo krepko povezane komponente.

- 1) Implementiramo graf in poiščemo krepko povezane komponente s katerimkoli linearo časovnim algoritmom.
- 2) Primerjamo, če katerakoli krepko povezana komponenta vsebuje spremenljivko in njen negacijo. Če je tako, končamo z algoritmom.
- 3) Skonstruiramo manjši graf, ki vsebuje le eno vozlišče za vsako krepko povezano komponento in povezavo od komponente i do komponente j. To naredimo vedno, ko pri povezavi uv u pripada komponenti i in v pripada komponenti j. Iz tega nastane povezan acikličen graf.
- 4) Topološko uredimo vozlišča.
- 5) Za vsako komponento v obratnem topološkem urejanju, če spremenljivka še ni true, jo nastavimo na true. To lahko povzroči da se nekatere komponente nastavijo na false.

OK.

naloga 5.3

Problem 2-SAT. V študiju ste že spoznali pojma konjunkcij (logični in) in disjunkcije (logični ali) ter iz njiju izvirajoč pojem *konjunktivne oblike izraza*, ki pomeni konjunkcijo disjunkcij spremenljivk. Iskanje takšnih vrednosti posameznih spremenljivk (*true*, *false*), da izračunajo celoten izraz *vtrue*, če je to mogoče, pravimo *zadovoljitev* izraza (angl. *satisfiability* ali *SAT*). Posebno obliko problema SAT, kjer sta v disjunkcijah le po dve spremenljivki, imenujemo 2-SAT. Sami disjunkciji rečemo *stavek* in ga v splošnem zapišemo (x_1, x_2) - očitno je lahko tako x_1 ali x_2 (ali oba) negiran ali pa ne.

Vprašanja:

5.3.1 V naslednjih izrazih poiščite vse vrednosti spremenljivk x_1 , x_2 in x_3 , ki izračunajo izraz na *true*.

- $(x_1 \vee \neg x_2) \wedge (x_2 \vee \neg x_3) \wedge (x_3 \vee \neg x_1)$
- $(\neg x_1 \vee x_2) \wedge (x_1 \vee x_2) \wedge (\neg x_2 \vee \neg x_1)$
- $(\neg x_1 \vee x_3) \wedge (x_1 \vee x_2) \wedge (\neg x_3 \vee x_1) \wedge (\neg x_1 \vee \neg x_3)$
- $(\neg x_1 \vee x_3) \wedge (x_1 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (x_1 \vee x_3)$

5.3.2 Peter Zmeda je dobil za nalogo, da napiše algoritem, ki reši problem 2-SAT. Nekaj časa je gledal problem, potem brskal po knjigi in zapiskih ter našel magično besedo - *požrešna metoda*. Tako je pričel pisati idejo algoritma nekako takole:

1. Na začetku ima vsaka spremenljivka bodisi vrednost *true* ali *false*.
2. Vzemi prvi stavek (x_1, x_2) , ki še ni nezadovoljen, ter določi vrednost spremenljivke x_1 (ali x_2) tako, da se stavek izračuna na *true* - postane zadovoljen.
3. Nastavi še vrednosti spremenljivke v vseh ostalih stavkih, kjer nastopa x_1 (ali x_2), in izračunaj posamezne stavke: nekateri bodo postali zadovoljeni.
4. Ponovi korak 2.

Poglejmo si primer $(x_1 \vee \neg x_2) \wedge (x_2 \vee \neg x_1)$:

- na začetku sta oba stavka $(x_1 \vee \neg x_2)$ in $(x_2 \vee \neg x_1)$ še nedoločena in, recimo, začnimo pri stavku $(x_1 \vee \neg x_2)$: naj bo x_1 *true* ter stavek $(x_1 \vee \neg x_2)$ postane zadovoljen;
- nato v stavku $(x_2 \vee \neg x_1)$ dobimo $(x_2 \vee \text{false})$, kar pomeni, da moramo nastaviti tudi x_2 na *true*;
- preverimo prvi stavek in smo uspeli najti pravo nastavitev.

Osnovni izliv Petrovega pristopa je v 3. koraku, saj bodo nekateri stavki postali zadovoljeni, kar je super; pri nekaterih stavkih bo ena spremenljivka še nenastavljena, kar se bo rešilo kasneje; največji problem pa so stavki, kjer sta obe spremenljivki določeni in stavek se izračuna na *false*. Kaj pa sedaj?

Pomagajte Petru, da sestavi celoten algoritem zasnovan na njegovi ideji. Napišite psevdokodo algoritma.

5.3.3 Kakšna je časovna zahtevnost vašega algoritma glede na število spremenljivk?

5.3.4 Ker je domača naloga pri poglavju o grafih, se da izraz preoblikovati v graf? Kaj bi bile povezave in kaj vozlišča? Poiščite algoritom, ga opišite, utemeljite njegovo pravilnost in analizirajte njegovo časovno zahtevnost.

5.3.1

$$(x_1 \vee \neg x_2) \wedge (x_2 \vee \neg x_3) \wedge (x_3 \vee \neg x_1)$$

$x_1 = \text{true}$, $x_2 = \text{true}$, $x_3 = \text{true}$

$x_1 = \text{false}$, $x_2 = \text{false}$, $x_3 = \text{false}$

$$(\neg x_1 \vee x_2) \wedge (x_1 \vee x_2) \wedge (\neg x_2 \vee \neg x_1)$$

$x_1 = \text{false}$, $x_2 = \text{true}$

$$(\neg x_1 \vee x_3) \wedge (x_1 \vee x_2) \wedge (\neg x_3 \vee x_1) \wedge (\neg x_1 \vee \neg x_3)$$

$x_1 = \text{false}$, $x_2 = \text{true}$, $x_3 = \text{false}$

$$(\neg x_1 \vee x_3) \wedge (x_1 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (x_1 \vee x_3)$$

OK

Nikoli ni true!

5.3.2

2-SAT algoritmom:

Dobimo izraz...

Nastavimo x_1 na TRUE

Gremo v while(TRUE):

Vzamemo prvi stavek (primer $(x_1 \vee \neg x_2)$)

Vidimo da bo stavek TRUE nato vzamemo drugi stavek: i+1

Vzamemo drugi stavek (primer $(x_2 \vee \neg x_1)$)

Da bo stavek TRUE nastavimo x_2 na TRUE.

V primeru $(\neg x_1 \vee \neg x_2)$ da ne moremo stavka dati na true dobimo UNSATISFIABLE.

Lahko se edino vrnemo na prvi stavek in ga poskušamo še drugače preoblikovati in nato nadaljujemo z drugim korakom rekurzivno. Ko pregledujemo vse stavke tako nadaljujemo do konca da pridemo do konca izraza z vrednostjo TRUE.

Ideja je ok, vendar manjka psevdokoda, ki reši splošen problem z n

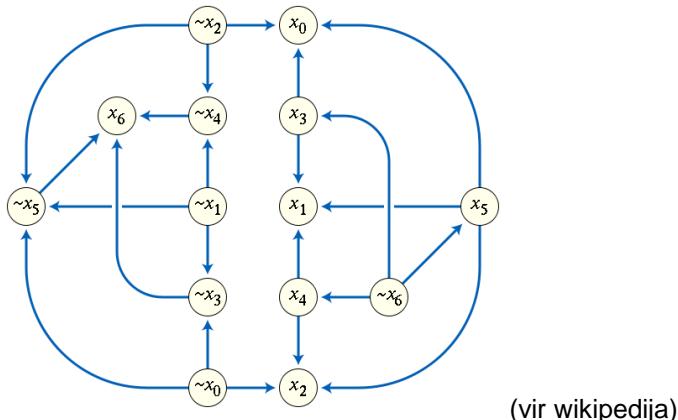
5.3.3

Časovna zahtevnost algoritma glede na spremenljivke je $O(n^2)$.

Ne bo držalo. Če ne najde zadovoljive vrednosti spremenljivke, algoritom sestopa in lahko potrebuje eksponen-

5.3.4

Izraz se da preoblikovati v graf. Vozlišča v grafu bi bile spremenljivke: $x_1, \neg x_1, x_2, \neg x_2, x_3, \neg x_3$. Odnosi med spremenljivkami (\wedge, \vee) pa povezave. Se pravi da sta dve spremenljivki v povezavi, če sta skupaj v izrazu.



Tak algoritem bo najbolj učinkovit v linearinem času.

Povezave in njihova smer niso le "odnosi med spremenljivkami", ampak nekaj ve! Koliko dobimo povezav za m st

5.3.1

$$(x_1 \vee \neg x_2) \wedge (x_2 \vee \neg x_3) \wedge (x_3 \vee \neg x_1)$$

x_1 - false

x_2 - false

x_3 - false

manjka $x_1=true$, $x_2=true$, $x_3=true$

$$(\neg x_1 \vee x_2) \wedge (x_1 \vee x_2) \wedge (\neg x_2 \vee \neg x_1)$$

x_1 - false

x_2 - true

$$(\neg x_1 \vee x_3) \wedge (x_1 \vee x_2) \wedge (\neg x_3 \vee x_1) \wedge (\neg x_1 \vee \neg x_3)$$

x_1 - fase

x_2 - true

x_3 - false

$$(\neg x_1 \vee x_3) \wedge (x_1 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (x_1 \vee x_3)$$

Zadnja možnost je ne rešljiva.

5.3.2

Če pride do tega primera se vrneš en stavek nazaj, če lahko preoblikuješ vrednosti tako da je stavek true potem rekurzivno popraviš vse prejšnje in preverjaš da so vsi true če niso sevrnemo na predzadnji niz in zopet preoblikujemo vrednosti.. Ko rekurzivno popravimo vrednosti nadaljujemo z popravljanjem nizov.

Ideja ok, vendar manjka psevdokoda, ki reši poljuben 2-SAT z m stavki in n spremenljivkami.

5.3.3

Časovna zahtevnost je n^2

Ne bo držalo.

5.3.4

Vozlišča bi bila spremenljivke povezave pa medsebojni odnos. Se pravi povezava bi bila med spremenljivkama v primeru da se skupaj nahajata v nizu. Z dodatno spremenljivko označimo ali je vozlišče true(1) ali fasle(0).

Kako bi s tem rešili 2-SAT?

APS2

Naloga 5.3

5.3.1

- $(x_1 \vee \neg x_2) \wedge (x_2 \vee \neg x_3) \wedge (x_3 \vee \neg x_1)$

x1-false, x2-false, x3-false

x1-true, x2-true, x3-true

- $(\neg x_1 \vee x_2) \wedge (x_1 \vee x_2) \wedge (\neg x_2 \vee \neg x_1)$

x1-false, x2-true

- $(\neg x_1 \vee x_3) \wedge (x_1 \vee x_2) \wedge (\neg x_3 \vee x_1) \wedge (\neg x_1 \vee \neg x_3)$

x1-false, x2-true, x3-false

- $(\neg x_1 \vee x_3) \wedge (x_1 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (x_1 \vee x_3)$

ni take vrednosti za x1 in x3, da se bo izraz izračunal v true

OK

5.3.3.

Časovna zahtevnost je 2^n , kjer je n število spremenljivk.

Za kateri algoritem?

5.3.4.

Izraz se da preoblikovati v graf, in sicer tako, da bi povezava predstavljal konjunkcijo/disjunkcijo, vozlišča pa spremenljivke.

In kako bi s tem rešili 2-SAT?

5.3

1. V naslednjih izrazih poiščite vse vrednosti spremenljivk x_1 , x_2 in x_3 , ki izračunajo izraz na true.

$(x_1 \vee \neg x_2) \wedge (x_2 \vee \neg x_3) \wedge (x_3 \vee \neg x_1)$: **x1=0, x2=0, x3=0 ali x1=1, x2=1, x3=1**

x1	x2	x3	$\neg x_1$	$\neg x_2$	$\neg x_3$	$(x_1 \vee \neg x_2)$	$(x_2 \vee \neg x_3)$	$(x_3 \vee \neg x_1)$	R
0	0	0	1	1	1	1	1	1	1
0	0	1	1	1	0	1	0	1	0
0	1	0	1	0	1	0	1	1	0
0	1	1	1	0	0	0	1	1	0
1	0	0	0	1	1	1	1	0	0
1	0	1	0	1	0	1	0	1	0
1	1	0	0	0	1	1	1	0	0
1	1	1	0	0	0	1	1	1	1

$(\neg x_1 \vee x_2) \wedge (x_1 \vee x_2) \wedge (\neg x_2 \vee \neg x_1)$: **x1=0, x2=1**

x1	x2	$\neg x_1$	$\neg x_2$	$(\neg x_1 \vee x_2)$	$(x_1 \vee x_2)$	$(\neg x_2 \vee \neg x_1)$	R
0	0	1	1	1	0	1	0
0	0	1	1	1	0	1	0
0	1	1	0	1	1	1	1
0	1	1	0	1	1	1	1
1	0	0	1	0	1	1	0
1	0	0	1	0	1	1	0
1	1	0	0	1	1	0	0
1	1	0	0	1	1	0	0

$(\neg x_1 \vee x_3) \wedge (x_1 \vee x_2) \wedge (\neg x_3 \vee x_1) \wedge (\neg x_1 \vee \neg x_3)$: **x=0, x2=1, x3=0**

x1	x2	x3	$\neg x_1$	$\neg x_3$	$(\neg x_1 \vee x_3)$	$(x_1 \vee x_2)$	$(\neg x_3 \vee x_1)$	$(\neg x_1 \vee \neg x_3)$	R
0	0	0	1	1	1	0	1	1	0
0	0	1	1	0	1	0	0	1	0
0	1	0	1	1	1	1	1	1	1
0	1	1	1	0	1	1	0	1	0
1	0	0	0	1	0	1	1	1	0
1	0	1	0	0	1	1	1	0	0
1	1	0	0	1	0	1	1	1	0
1	1	1	0	0	1	1	1	0	0

$(\neg x_1 \vee x_3) \wedge (x_1 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (x_1 \vee x_3)$: **ni rešitev**

x1	x3	$\neg x_1$	$\neg x_3$	$(\neg x_1 \vee x_3)$	$(x_1 \vee \neg x_3)$	$(\neg x_1 \vee \neg x_3)$	$(x_1 \vee x_3)$	R
0	0	1	1	1	1	1	0	0
0	1	1	0	1	0	1	1	0
0	0	1	1	1	1	1	0	0
0	1	1	0	1	0	1	1	0
1	0	0	1	0	1	1	1	0
1	1	0	0	1	1	0	1	0
1	0	0	1	0	1	1	1	0
1	1	0	0	1	1	0	1	0

OK

2. Osnovni izliv Petrovega pristopa je v 3. koraku, saj bodo nekateri stavki postali zadovoljeni, kar je super; pri nekaterih stavkih bo ena spremenljivka še nenastavljena, kar se bo rešilo kasneje; največji problem pa so stavki, kjer sta obe spremenljivki določeni in stavek se izračuna na false. Kaj pa sedaj?

V tem primeru eno spremenljivko sprememimo in to zabeležimo v tabelo, nato ponovno preverimo vse izraze in v primeru da spet pridemo do false sprememimo drugo spremenljivko.

Katero drugo spremenljivko? Kaj pa, e imate n spremenljivk? Manjka algoritmom, ki v splošnem reši

1)

- $(x_1 \vee \neg x_2) \wedge (x_2 \vee \neg x_3) \wedge (x_3 \vee \neg x_1)$
 $(x_1 = \text{True} \wedge x_2 = \text{True} \wedge x_3 = \text{True})$
ali
 $(x_1 = \text{False} \wedge x_2 = \text{False} \wedge x_3 = \text{False})$
- $(\neg x_1 \vee x_2) \wedge (x_1 \vee x_2) \wedge (\neg x_2 \vee \neg x_1)$
 $x_2 = \text{True} \wedge x_1 = \text{False}$
- $(\neg x_1 \vee x_3) \wedge (x_1 \vee x_2) \wedge (\neg x_3 \vee x_1) \wedge (\neg x_1 \vee \neg x_3)$
 $(x_1 = \text{False} \wedge x_2 = \text{True} \wedge x_3 = \text{False})$
- $(\neg x_1 \vee x_3) \wedge (x_1 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (x_1 \vee x_3)$
Ni rešitve.

OK

2)

1. sat2(stavki):
2. for stavek in stavki:
3. if stavek == none:
4. for beseda in stavek:
5. if beseda == none:
6. setBeseda tako, da stavek = True;
7. break;
8. else if stavek == False:
9. poskusi drugi vrednosti spremenljivk;

TULE je potrebno sestopiti in preveriti tudi vse stavke :

3)

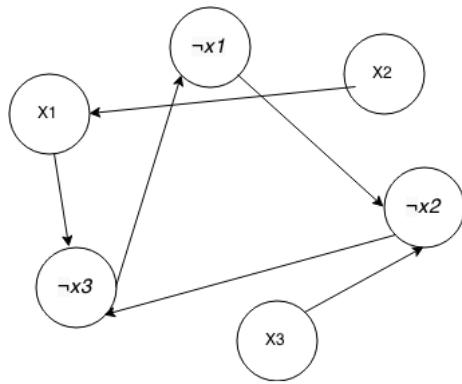
Čez vse stavke se moramo sprehoditi dvakrat. Časovna zahtevnost je $O(n)$, kjer je n število stavkov.

Glede na zgornji algoritem drži, vendar ne deluje pravilno.

4)

Vozlišča so spremenljivke in negirani spremenljivke. Povezava med x in z obstaja, če obstaja stavek $\neg x$ in z. Stavek je zadovoljen takrat, ko v grafu ne obstaja pot od x do $\neg x$ ali obratno. Preverimo, da nimamo nobene poti od x do $\neg x$, če taka pot obstaja ni rešitve, sicer izberemo eno spremenljivko in nastavimo jo na True in vse vozlišča, ki lahko dosežemo. Nastavimo False njihovim negacijam. Nadalujemo dokler ne nastavimo vsi vozlišča.

OK. Kaj pa asovna zahtevnost?



Slika 1: Primer grafa. $(x_1 \vee \neg x_2) \wedge (x_2 \vee \neg x_3) \wedge (x_3 \vee \neg x_1)$

5.3

3. junij 2015
17:51

naloga 5.3

Problem 2-SAT. V študiju ste že spoznali pojma konjunkcij (logični in) in disjunkcije (logični ali) ter iz njiju izvirajoč pojem *konjuktivne oblike* izraza, ki pomeni konjunkcijo disjunkcij spremenljivk. Iskanje takšnih vrednosti posameznih spremenljivk (`true`, `false`), da izračunajo celoten izraz v `true`, če je to mogoče, pravimo *zadovoljitev* izraza (angl. *satisfiability* ali *SAT*). Posebno obliko problema SAT, kjer sta v disjunkcijah le po dve spremenljivki, imenujemo 2-SAT. Sami disjunkciji rečemo *stavek* in ga v splošnem zapišemo (x_1, x_2) - očitno je lahko tako x_1 ali x_2 (ali oba) negiran ali pa ne.

Vprašanja:

5.3.1 V naslednjih izrazih poiščite vse vrednosti spremenljivk x_1 , x_2 in x_3 , ki izračunajo izraz na `true`.

- $(x_1 \vee \neg x_2) \wedge (x_2 \vee \neg x_3) \wedge (x_3 \vee \neg x_1)$
- $(\neg x_1 \vee x_2) \wedge (x_1 \vee x_2) \wedge (\neg x_2 \vee \neg x_1)$
- $(\neg x_1 \vee x_3) \wedge (x_1 \vee x_2) \wedge (\neg x_3 \vee x_1) \wedge (\neg x_1 \vee \neg x_3)$
- $(\neg x_1 \vee x_3) \wedge (x_1 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (x_1 \vee x_3)$

Prvi primer (X1=F, X2=F, X3=F), (X1=T, X2=T, X3=T)

Drugi primer (X1=F, X2=T, X3=F) (X1=F, X2=T, X3=T)

Tretji Primer (X1=F, X2=T, X3=F)

Četrti primer: Ni rešitev

OK

5.3.2 Peter Zmeda je dobil za nalogu, da napiše algoritem, ki reši problem 2-SAT. Nekaj časa je gledal problem, potem brskal po knjigi in zapiskih ter našel magično besedo - *požrešna metoda*. Tako je pričel pisati idejo algoritma nekako takole:

1. Na začetku ima vsaka spremenljivka bodisi vrednost `true` ali `false`.
2. Vzemi prvi stavek (x_1, x_2) , ki še ni nezadovoljen, ter določi vrednost spremenljivke x_1 (ali x_2) tako, da se stavek izračuna na `true` - postane zadovoljen.
3. Nastavi še vrednosti spremenljivke v vseh ostalih stavkih, kjer nastopa x_1 (ali x_2), in izračunaj posamezne stavke: nekateri bodo postali zadovoljeni.
4. Ponovi korak 2.

Poglejmo si primer $(x_1 \vee \neg x_2) \wedge (x_2 \vee \neg x_1)$:

- na začetku sta oba stavka $(x_1 \vee \neg x_2)$ in $(x_2 \vee \neg x_1)$ še nedoločena in, recimo, začnimo pri stavku $(x_1 \vee \neg x_2)$: naj bo x_1 `true` ter stavek $(x_1 \vee \neg x_2)$ postane zadovoljen;
- nato v stavku $(x_2 \vee \neg x_1)$ dobimo $(x_2 \vee \text{false})$, kar pomeni, da moramo nastaviti tudi x_2 na `true`;
- preverimo prvi stavek in smo uspeli najti pravo nastavitev.

Osnovni iziv Petrovega pristopa je v 3. koraku, saj bodo nekateri stavki postali zadovoljeni, kar je super; pri nekaterih stavkih bo ena spremenljivka še nenastavljena, kar se bo rešilo kasneje; največji problem pa so stavki, kjer sta obe spremenljivki določeni in stavek se izračuna na `false`. Kaj pa sedaj?

Pomagajte Petru, da sestavi celoten algoritem zasnovan na njegovi ideji. Napišite psevdokodo algoritma.

2_SAT:

Stavek = razrežiNaPodprobleme(stavek)

X1 = true

X2 = true

Za vsak podproblem:

If podproblem==false:

 X1=false

 PreveriPrejšnje stavke

If podproblem=false:

 X2=false

 PreveriPrejšnjeStavke

If podproblem=false:

 X1=true

 preveriPrejšnjeStavke

Če je podproblem še false, nimamo rešitve, vrnemo neuspeh

Če konča zanko vrnemo uspeh in spremenljivke

Kaj tono so podproblemi? Pri tehniki deli in vlač

5.3.3 Kakšna je časovna zahtevnost vašega algoritma glede na število spremenljivk?

$O(n)$

Ne bo držalo. Odvisno od števila podproblemov. Vsak podproblem, kot kaže algoritem, pa potrebuje linearen as

naloga 5.3

Problem 2-SAT. V študiju ste že spoznali pojma konjunkcij (logični in) in disjunkcije (logični ali) ter iz njiju izvirajoč pojem *konjunktivne oblike* izraza, ki pomeni konjunkcijo disjunkcij spremenljivk. Iskanje takšnih vrednosti posameznih spremenljivk (`true`, `false`), da izračunajo celoten izraz v `true`, če je to mogoče, pravimo *zadovoljitev* izraza (angl. *satisfiability* ali *SAT*). Posebno obliko problema SAT, kjer sta v disjunkcijah le po dve spremenljivki, imenujemo 2-SAT. Sami disjunkciji rečemo *stavek* in ga v splošnem zapišemo (x_1, x_2) - očitno je lahko tako x_1 ali x_2 (ali oba) negiran ali pa ne.

Vprašanja:

5.3.1 V naslednjih izrazih poiščite vse vrednosti spremenljivk x_1 , x_2 in x_3 , ki izračunajo izraz na `true`.

- $(x_1 \vee \neg x_2) \wedge (x_2 \vee \neg x_3) \wedge (x_3 \vee \neg x_1)$ // dve rešitvi
- $(\neg x_1 \vee x_2) \wedge (x_1 \vee x_2) \wedge (\neg x_2 \vee \neg x_1)$ // ena rešitev
- $(\neg x_1 \vee x_3) \wedge (x_1 \vee x_2) \wedge (\neg x_3 \vee x_1) \wedge (\neg x_1 \vee \neg x_3)$ // ena rešitev
- $(\neg x_1 \vee x_3) \wedge (x_1 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (x_1 \vee x_3)$ // ni rešitve

	X_1	X_2	X_3
a)	T / F	T / F	T / F
b)	F	T	/
c)	F	T	F
d)	/	/	/

OK

5.3.2 Peter Zmeda je dobil za nalogo, da napiše algoritem, ki reši problem 2-SAT. Nekaj časa je gledal problem, potem brskal po knjigi in zapiskih ter našel magično besedo - *požrešna metoda*. Tako je pričel pisati idejo algoritma nekako takole:

1. Na začetku ima vsaka spremenljivka bodisi vrednost `true` ali `false`.
2. Vzemi prvi stavek (x_1, x_2) , ki še ni nezadovoljen, ter določi vrednost spremenljivke x_1 (ali x_2) tako, da se stavek izračuna na `true` - postane zadovoljen.
3. Nastavi še vrednosti spremenljivke v vseh ostalih stavkih, kjer nastopa x_1 (ali x_2), in izračunaj posamezne stavke: nekateri bodo postali zadovoljeni.
4. Ponovi korak 2.

Poglejmo si primer $(x_1 \vee \neg x_2) \wedge (x_2 \vee \neg x_1)$:

- na začetku sta oba stavka $(x_1 \vee \neg x_2)$ in $(x_2 \vee \neg x_1)$ še nedoločena in, recimo, začnimo pri stavku $(x_1 \vee \neg x_2)$: naj bo x_1 `true` ter stavek $(x_1 \vee \neg x_2)$ postane zadovoljen;
- nato v stavku $(x_2 \vee \neg x_1)$ dobimo $(x_2 \vee \text{false})$, kar pomeni, da moramo nastaviti tudi x_2 na `true`;
- preverimo prvi stavek in smo uspeli najti pravo nastavitev.

Osnovni izziv Petrovega pristopa je v 3. koraku, saj bodo nekateri stavki postali zadovoljeni, kar je super; pri nekaterih stavkih bo ena spremenljivka še nenastavljenata, kar se bo rešilo

kasneje; največji problem pa so stavki, kjer sta obe spremenljivki določeni in stavek se izračuna na `false`. Kaj pa sedaj?

ODGOVOR:

Moj način bi bil takšen: Pri prvem koraku bi shranil vrednost prve spremenljivke in i-ti indeks spremenljivke. Če pride do izraza, kjer sta obe spremenljivki `FALSE`, bi začel računati od začetka vendar z drugačno vrednostjo v istem indeksu. Če bi ponovno prišlo do izraza `FALSE`, vzamem naslednjo spremenljivko z indeksom $i+1$. Časovna zahtevnost z večanjem spremenljivk dvojiško narašča.

Pomagajte Petru, da sestavi celoten algoritem zasnovan na njegovi ideji. Napišite psevdokodo algoritma.

1. Na začetku ima vsaka spremenljivka bodisi vrednost `true` ali `false` **in statusno spremenljivko**.
2. Vzemi prvi stavek (x_1, x_2) , ki še ni nezadovoljen, ter določi vrednost spremenljivke x_1 (ali x_2) tako, da se stavek izračuna na `true` - postane zadovoljen.
3. Nastavi še vrednosti spremenljivke v vseh ostalih stavkih, kjer nastopa x_1 (ali x_2), in izračunaj posamezne stavke: nekateri bodo postali zadovoljeni.
4. **Če pride do nezadovoljnega izraza, povečam statusno spremenljivko in spremenim vrednost prve nastavljene spremenljivke iz true v false ali iz false v true.**
5. **Če je statusna spremenljivka > 1 vzamem drugo spremenljivko in nastavim statusno spremenljivko = 0, drugače**
6. Ponovi korak 2.

OK.

5.3.3 Kakšna je časovna zahtevnost vašega algoritma glede na število spremenljivk?

ODGOVOR: Časovna zahtevnost mojega algoritma bi bila vsaj $O(2^n)$, saj moramo vsako spremenljivko preveriti $2x$ (`true` ali `false`). **OK.**

5.3.4 Ker je domača naloga pri poglavju o grafih, se da izraz preoblikovati v graf? Kaj bi bile povezave in kaj vozlišča? Poiščite algoritem, ga opišite, utemeljite njegovo pravilnost in analizirajte njegovo časovno zahtevnost.

ODGOVOR: Da se narisati graf. Vozlišča bi bile spremenljivke.



Kako bi s tem rešili 2-SAT?



Naloga 5.3

5.3.1

1.) $x_1 = \text{True}$

$x_2 = \text{True}$

$x_3 = \text{True}$

manjka $x_1=\text{false}$, $x_2=\text{false}$, $x_3=\text{false}$

2.) $x_1 = \text{False}$

$x_2 = \text{True}$

3.) $x_1 = \text{False}$

$x_2 = \text{True}$

$x_3 = \text{False}$

4.) Ta izraz nima logične rešitve.

5.3.2

Menim da bi bilo lažje če opustimo petrovo idejo ter se lotimo z grafom saj bi hitreje našli rešitev kot pa tako. Vendar lahko pa tudi popravimo Petrov algoritem, ki ni čisto napačen. Popravil bi ga na sledeč način:

sprva bi shranil vse spremenljivke katere vsebuje v neko tabelo. Jih tam pregledal in nastavil si nastavil vse kombinacije tako da bi imel shranjene vrednosti. Zatem pa bi pogledal in poiskal pravo rešitev.

Problema se lotimo z naslednjimi koraki:

1. Zapisal vse spremenljivke, ter vse kombinacije teh spremenljivk po vrednostih.
2. Preglej ali so vsi stavki zadovoljeni, če niso opusti kombinacijo se vrni v tabelo vzemi naslednjo kombinacijo ter preglej stavek ali je ta kombinacija zadovoljiva.
3. Če smo zadovoljili vsem ukazom ter je problem rešen, vrnemo kombinacijo, ki je rešila problem.
4. Če smo prišli do konca vseh kombinacij in nismo našli rešitve pomeni da ne obstaja logična kombinacija spremenljivk ki bi rešila problem.

OK.

5.3.3

Menim da ta rešitev ni najbolša saj je časovno ter prostorsko potratna. Časovna zahtevnost je $O(2^n * n)$, če pričakujemo da za gradnjo tabel upoštevamo predpripravo problema.

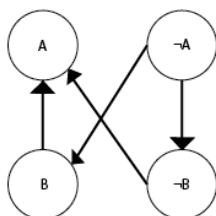
OK.

5.3.4

Izraz se da preoblikovati in sicer na naslednji način: Vse spremenljivke ki jih imamo pretvorimo v vozlišča prav tako pa dodamo še njihove negirane vrednosti (recimo da imamo $(\neg a \vee b)$ moramo dodati a in $\neg a$ ter b in $\neg b$). Nato pa za vsak stavek $(\neg a \vee b)$, naredimo povezavo iz $\neg a$ do b in še od $\neg b$ do a , če je a nastavljen na False in b na True.

Tako povežemo cev graf z usmerjenimi povezavami. Če pa pride do cikla med katerima kolik vizliščama pa pomeni da graf ni rešljiv. Časovna odvisnost pa je $O(n)$.

Primer: $(a \vee b) \wedge (\neg a \vee \neg b)$



Odlino.

5.3.1

$$(x_1 \vee \neg x_2) \wedge (x_2 \vee \neg x_3) \wedge (x_3 \vee \neg x_1)$$

x1=True, x2=True, x3=True

x1=False, x2=False, x3=False

$$(\neg x_1 \vee x_2) \wedge (x_1 \vee x_2) \wedge (\neg x_2 \vee \neg x_1)$$

x1=False, x2=True

$$(\neg x_1 \vee x_3) \wedge (x_1 \vee x_2) \wedge (\neg x_3 \vee x_1) \wedge (\neg x_1 \vee \neg x_3)$$

x1=False, x2=True, x3=False

$$(\neg x_1 \vee x_3) \wedge (x_1 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (x_1 \vee x_3)$$

Rezultat je v edno False

OK

5.3.3

Časovna zahtevnost požrešne metode je $O(n^k)$. Kjer je n število stakov in k število spremenljiv.

5.3.4

Zakaj?

V grafu so vozlišča posamezne spremenljivke in njihove negacije. Povezave predstavljajo vrednost true. Rešitev najdemo tako da poiščemo trdne povezave. Zahtevnost je linearна.

Kako bi s tem rešili 2-SAT?

Naloga 5.3

5.3.1

- $(x_1 \vee \neg x_2) \wedge (x_2 \vee \neg x_3) \wedge (x_3 \vee \neg x_1)$: $x_1 = \text{True}$, $x_2 = \text{True}$,
 $x_3 = \text{True}$ ali $x_1=\text{false}$, $x_2=\text{false}$, $x_3=\text{false}$
- $(\neg x_1 \vee x_2) \wedge (x_1 \vee x_2) \wedge (\neg x_2 \vee \neg x_1)$: $x_1 = \text{False}$, $x_2 = \text{True}$
- $(\neg x_1 \vee x_3) \wedge (x_1 \vee x_2) \wedge (\neg x_3 \vee x_1) \wedge (\neg x_1 \vee \neg x_3)$: $x_1 = \text{False}$,
 $x_2 = \text{True}$, $x_3 = \text{False}$
- $(\neg x_1 \vee x_3) \wedge (x_1 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (x_1 \vee x_3)$: Izraz je
protislovje

5.3.2

Zgeneriraj vse mozne kombinacije spremenljivk.

Vzemi prvi stavek (x_1 , x_2), ki še ni nezadovoljen, ter določi vrednost spremenljivke x_1 (ali x_2) tako, da se stavek izračuna na true - postane zadovoljen.

Nastavi še vrednosti spremenljivke v vseh ostalih stavkih, kjer nastopa x_1 (ali x_2), in izračunaj posamezne stavke: nekateri bodo postali zadovoljeni.

Če postanejo nezadovoljeni označi kombinacijo kot uporabljeno in naključno izberi naslednjo. Če postanejo vsi zadovoljni ,vrni rešitev.
Če zmanjka kombinacij, vrni ni rešljivo.

OK

5.3.3

Časovna zahtevnost :

$O(2^n)$ n – število spremenljivk.

Zakaj?

5.3.4

5.3

- $(1, 1, 1), (0, 0, 0)$
- $(0, 1)$
- $(0, 1, 0)$
- Ni rešitve

OK

```
x = [0, 0, 0, ...];
mem[];
i = 0;
while(i < stavki.len){
    if(stavki[i]){
        i++;
    } else {
        // vzemi spremenljivko, ki najdlje ni bila spremenjena
        n = stavki[i].changeFirst ? stavki[i].f : stavki[i].s;
        x[n] != x[n];
        if(x in mem) return false; // stanje je že videno, rešitve ni
        mem.add(x);
        stavki[i].changeFirst != stavki[i].changeFirst;
        i = 0;
    }
}
return x;
```

OK

Časovna zahtevnost v najslabšem primeru je $O(2^n)$, v povprečnem primeru pa med $\Omega(n)$ in $O(n^2)$.

Kako izgleda povprečen 2-SAT problem? :)

Izraz lahko preoblikujemo v graf tako, da stavek $(a \vee \neg b)$ zapišemo kot povezavo iz vozlišča $\neg a$ v vozlišče b in povezavo iz vozlišča $\neg b$ v vozlišče a . V takšnem grafu nato poiščemo krepko povezane komponente. Če ista komponenta vsebuje a in $\neg a$, je tak problem nerešljiv. Ker je med komponentami implikacija v smeri povezav, komponente topološko uredimo. Ker v topološko urejenem grafu vse povezave potekajo v isto smer (od prve komponente proti zadnji), to pomeni, da če ima vozlišče znotraj komponente K vrednost 1, zagotovo ne bo prišlo do konfliktov neglede na vrednosti vozlišč v komponentah, ki so topološko pred K . Da to dosežemo, nedoločena vozlišča znotraj komponent nastavimo 1 v obratnem topološkem vrstnem redu (od topološko zadnje komponente do topološko prve komponente), njihove negirane vrednosti pa na 0.

Za gornji algoritem moramo torej poiskati krepke komponente grafa, jih topološko urediti, in nato še vsaki spremenljivki določiti vrednost. Ker je vsaka od teh operacij linearна, je celotna zahtevnost tudi linearна.

OK.

5.1

- $(x_1 \vee \neg x_2) \wedge (x_2 \vee \neg x_3) \wedge (x_3 \vee \neg x_1)$

x1= false, x2 = false, x3= true/false

Ne bo držalo.

- $(\neg x_1 \vee x_2) \wedge (x_1 \vee x_2) \wedge (\neg x_2 \vee \neg x_1)$

x1= false, x2 = true

- $(\neg x_1 \vee x_3) \wedge (x_1 \vee x_2) \wedge (\neg x_3 \vee x_1) \wedge (\neg x_1 \vee \neg x_3)$

x1= false, x2 = true , x3= false

- $(\neg x_1 \vee x_3) \wedge (x_1 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (x_1 \vee x_3)$

ne obstaja

Naloga 5.3

5.3.1.)

- $(x_1 \vee \neg x_2) \wedge (x_2 \vee \neg x_3) \wedge (x_3 \vee \neg x_1)$
 - $x_1=1, x_2=1, x_3=1$
 - $x_1=0, x_2=0, x_3=0$
 - $x_1=0, x_2=1, x_3=0$
 - ~~$x_1=0, x_2=1, x_3=1$~~
- $(\neg x_1 \vee x_2) \wedge (x_1 \vee x_2) \wedge (\neg x_2 \vee \neg x_1)$
 - $x_1=0, x_2=1$
 - ~~$x_1=1, x_2=0$~~
- $(\neg x_1 \vee x_3) \wedge (x_1 \vee x_2) \wedge (\neg x_3 \vee x_1) \wedge (\neg x_1 \vee \neg x_3)$
 - $x_1=0, x_2=1, x_3=0$
 - $x_1=0, x_2=1, x_3=1$
- $(\neg x_1 \vee x_3) \wedge (x_1 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (x_1 \vee x_3)$
 - Ni situacij kjer je izraz resničen

5.3.2.)

Algoritem Petra Zmede predelamo tako:

1. Na začetku so vse spremenljivke nedoločene.
2. Vzemi prvi stavek (x_1, x_2) , ki še ni nezadovoljen, ter določi vrednost spremenljivke x_1 (ali x_2) tako, da se stavek izračuna na true - postane zadovoljen. Dodaj na sklad kateri stavek si izbral in katero spremenljivko si spreminal.
3. Nastavi še vrednosti spremenljivke v vseh ostalih stavkih, kjer nastopa x_1 (ali x_2), in izračunaj posamezne staveke: nekateri bodo postali zadovoljeni.
4. Če je rezultat od kakšnega stavka z obema spremenljivkama false, nadaljuj, če ne ponovi korak 2.
5. Iz sklada odstrani zadnjo spremenjeno spremenljivko in jo od-nastavi.
6. Če je druga spremenljivka v nazadnje izbranem stavku ni nastavljena jo nastavi tako, da je stavek true in nadaljuj z korakom 3, če ne odstrani zadnji izbrani stavek in nadaljuj s korakom 5.
7. Ponovi korak 2. 6. korak - katero spremenljivko izbrati?

5.3.3.)

Zahetvost algoritma v najslabšem primeru je $O(2^n)$, kjer je n število spremenljivk, ker moramo v najslabši situaciji pregledat vse možne kombinacije vrednosti spremenljivk.

Sumim, da ste hoteli napisati $O(2^n)$, ker omenjate vse kombinacije vrednosti spremenljivk.

5.3.1

a. $(x_1 \vee \neg x_2) \wedge (x_2 \vee \neg x_3) \wedge (x_3 \vee \neg x_1)$

$x_1 = \text{true}$

$x_2 = \text{true}$

manjka $x_1=\text{false}, x_2=\text{false}, x_3=\text{false}$

$x_3 = \text{true}$

b. $(\neg x_1 \vee x_2) \wedge (x_1 \vee x_2) \wedge (\neg x_2 \vee \neg x_1)$

$x_1 = \text{false}$

$x_2 = \text{true}$

c. $(\neg x_1 \vee x_3) \wedge (x_1 \vee x_2) \wedge (\neg x_3 \vee x_1) \wedge (\neg x_1 \vee \neg x_3) =$

$$(x_1 \Rightarrow x_3) \wedge (x_1 \vee x_2) \wedge (x_3 \Rightarrow x_1) \wedge (\neg x_1 \vee \neg x_3)$$

$x_1 = \text{false}$

$x_2 = \text{true}$

$x_3 = \text{false}$

V tem izrazu nisem našel pravilne rešitve, zato sem uporabil zakon izjavnega računa; lastnost implikacije. $A \Rightarrow B \sim \neg A \vee B$

a. $(\neg x_1 \vee x_3) \wedge (x_1 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (x_1 \vee x_3) =$

$$(x_1 \Rightarrow x_3) \wedge (x_1 \vee \neg x_3) \wedge (x_1 \wedge x_3) \wedge (x_1 \vee x_3)$$

$x_1 = \text{true}$

Tretji len x_1 IN x_3 je negiran!

$x_3 = \text{true}$

Potreboval sem se de Morganov zakon. $\neg(A \vee B) \sim \neg A \wedge \neg B$

5.3.1

Spremenil bi algoritem, in sicer nebi uporabil naključno izbrane vrednosti spremenljivk, ampak bi začel na začetku. Na začetku ima vsaka spremenljivka vrednost false.

	X1	X2
1. korak	false	false
2. korak	false	true
3. korak	true	false
4. korak	true	true

1. Vzemi prvi stavek (x_1, x_2), ki še ni nezadovoljen, ter določi vrednost spremenljivke x_1 (ali x_2) tako, da se stavek izračuna na true - postane zadovoljen.

2. Nastavi še vrednosti spremenljivke v vseh ostalih stavkih, kjer nastopa x_1 (ali x_2), in izračunaj posamezne stavke: nekateri bodo postali zadovoljeni. Če kakšen postane nezadovoljen in vse spremenljivke so že nastavljanje, se vrni v 1. korak in zadnjo spremenljivko postavi na true. Če pa je že pa jo postavi na false in naslednjo postavi na true.

In potem, e še vedno ni izraz TRUE?

3. Ponovi korak 2.

5.3.3

Časovna zahtevnost se lahko kar poveča, saj se v najslabšem primeru v 1. koraku lahko znajde 2krat število spremenljivk. V najboljšem primeru se vrne nikoli, in je časovna zahtevnost ista kot pri prejšnjemu algoritmu. Koliko je potem asovna zahtevnost?

5.4.4

Uporabimo lahko preiskovanje v globino. Njegova časovna zahtevnost je $O(V * E)$, pri katerem $v =$ število vozlišč, $e =$ število povezav.

Kako tono bi s preiskovanjem v globino rešili 2-SAT?

5 Naloga

5.3.1 Vse vrednosti spremenljivk

Ker je spremenljivk malo lahko najdemo rešitve kar s tabelami.Dokaz je spodaj.

5.3.2 2-SAT in požrešna metoda

Peter Zmeda je na dobri poti. Našel je, podatek, da se problem lahko reši s pomočjo požrešne metode. Jaz bi mu pomagal na naslednji način.

Vemo, da se lahko problema lotimo s pomočjo teorije o močno povezanih komponentah v grafu. Definirajmo 2 vozlišči v nekem usmerjenem grafu, vozlišče A in vozlišče B, omenjeni vozlišči sta močno povezani takrat ko obstaja od vozlišča A usmerjena povezava do vozlišča B in ko obstaja usmerjena povezava od vozlišča B nazaj do vozlišča A.

Take močno povezane komponente lahko iščemo s pomočjo algoritmov, ki delujejo na DFS metodi oz. iskanju v globino.

Kaj tono so v grafu povezave in k

Celoten koncept se glasi takole. V smislu grafa implikacij dva terma pripadata enaki močno povezani komponenti, če obstaja veriga implikacij iz enega terma v drug term in obratno. Spomnimo, da smo točno tako zgoraj definirali močno povezano komponento! Če to velja potem omenjena dva terma morata imeti enako vrednost v vsaki —. Če spremenljivka in njena negacija pripadata enaki močno povezani komponenti, izrazu ne moremo zadostiti, ker je nemogoče, da bi priredili obema termoma enako logično vrednost. Torej, če povzamemo: 2-CNF formula je zadovoljiva če in samo, če ne obstaja taka spremenljivka, ki ne pripada enaki močno povezani komponenti kot njena negacija.

Ta premislek nas pripelje do algoritma, ki bo tekel v linearinem času. Izvedli bomo analizo močne povezanosti nad grafom implikacij in preverili, da vsaka spremenljivka in njena negacija pripadata različnim komponentam.

Za to nalogo si bomo izbrali Kosaraju-v algoritem, ki je zelo preprost, ampak za razliko drugih algoritmov izvede dve DFS iskanji!

Algoritem:

```
g=Graph()
gt=Graph()
s=stack()
vA=Vertex()
vT = Vertex()
visited=Set()
while(s.size() != g.getVertices().size())
    vA=random(g.getVertices())
    if(vA not in g)
        g.BFS(v)
        if(BFS finished expanding vertex u)
            s.push(u)
gt = g.reverseDirectionOfAllArcs()
while(s.notEmpty())
    vT=s.pop()
    visited=gt.DFS(vT).toSet()
    g.removeMultiple(visited)
    s.popMultiple(visited)
```

Komentar: Če smo iskreni bi DFS lahko zamenjali tudi z BFS.

5.3.3 Časovna zahtevnost grafa glede na število spremenljivk

Časovna zahtevnost zgornjega algoritma je $O(V + E)$, kjer je V število vozlišč in E število povezav ob pogoju, da uporabljam seznam sosednosti. Torej algoritom teče v linearinem času.

Če graf predstavimo naprimer z matriko sosednosti postane časovna zahtevnost algoritma $O(V^2)$. Število spremenljivk je število vozlišč.

Časovna zahtevnost algoritma se glede na število spremenljivk ne spreminja, spreminja se edino ob uporabi različnih predstavitev grafa.(matrika, seznam....)

OK

5.3.4 Preoblikovanje izraza v graf

Izraz lahko preoblikujemo v graf tako, da zgradimo dejansko graf implikacij. To je usmerjeni graf v katerem predstavlja vozlišče spremenljivka oz. negacija spremenljivke. Povezave med spremenljivkama obstaja, če med njima obstaja relacija implikacije v normalni obliki izraza. Torej izraze spremenimov v implikacije in zgradimo graf.

Algoritem, ki rešuje ta problem na predstavitevi z grafom je Aspvall, Plass & Tarjan algoritem in bazira na principu močno povezanih komponent.

1. Ustvarimo graf na način, ki je opisan zgoraj.

2. Obdelaj množico močno povezanih komponent S v obratnem topološkem redu na naslednji način:

Če je S označen ne naredi ničesar. Če je S negiran potem se ustavi (stavek ne more biti zadovoljen). V nasprotnem primeru označi S kot true in $\neg S$ kot false. Časovna zahtevnost algoritma je linearна torej $O(n)$.

Prvotni članek na to temo se nahaja na:

http://www.math.ucsd.edu/~sbuss/CourseWeb/Math268_2007WS/2SAT.pdf

Viri: <http://en.wikipedia.org/wiki/2-satisfiability>

Kaj pomeni "obstaja relacija in"

5.3.1

- $x_1=x_2=x_3=\text{true}$, $x_1=x_2=x_3=\text{false}$
- $x_1=\text{false}$ in $x_2=x_3=\text{true}$, $x_1=x_3=\text{false}$ in $x_2=\text{true}$
- $x_1=x_3=\text{false}$ in $x_2=\text{true}$
- ni rešitev

OK

5.3.2

1. Vzemi prvi stavek (x_1, x_2), ki še ni zadovoljen

2. Določi vrednost spremenljivke x_1 (ali x_2) tako, da se stavek postane zadovoljen.

2.1 Nastavi še vrednosti spremenljivke v vseh ostalih stavkih, kjer nastopa x_1 (ali x_2), in izračunaj posamezne stavke.

Če kateri stavek postane nezadovoljen, poskusi z drugo spremenljivko in ponovi 2.1

Če še vedno obstaja kateri nezadovoljen, ni rešitve

Ponovi 1.

Manjka algoritem, ki v splošnem reši 2-SAT za m

5.3.3

$$T = O(n^2)$$

Zgornji algoritem sestopa pri izbiri drugane vrednosti spremenljivke, zato potrebuje $O(2^n)$ asa in ne $O(n^2)$.

Naloga 5.3

- Rešimo tako, da nastavimo x_1 x_2 in x_3 na neko vrednost in potem gledamo, če v vseh dobimo true. Možnosti so:

$x_1=False, x_2=False, x_3=False$

$x_1=False, x_2=False, x_3=True$

$x_1=False, x_2=True, x_3=False$

$x_1=False, x_2=True, x_3=True$

$x_1=True, x_2=False, x_3=False$

$x_1=True, x_2=False, x_3=True$

$x_1=True, x_2=True, x_3=False$

$x_1=True, x_2=True, x_3=True$

Potem pa izberemo $x_1=False, x_2=False, x_3=False$ in vstavimo v :

$$(x_1 \vee \neg x_2) \wedge (x_2 \vee \neg x_3) \wedge (x_3 \vee \neg x_1)$$

$$(\text{True}) \wedge (\text{True}) \wedge (\text{True})$$

Potem pa izberemo $x_1=False, x_2=False, x_3=True$ in vstavimo v :

$$(x_1 \vee \neg x_2) \wedge (x_2 \vee \neg x_3) \wedge (x_3 \vee \neg x_1)$$

$$(\text{True}) \wedge (\text{False}) \wedge (\text{True})$$

Ponovimo postopek na vseh primerih.

- $(x_1 \vee \neg x_2) \wedge (x_2 \vee \neg x_3) \wedge (x_3 \vee \neg x_1)$

Rešitev: $x_1=False, x_2=False, x_3=False$

$x_1=True, x_2=True, x_3=True$

- $(\neg x_1 \vee x_2) \wedge (x_1 \vee x_2) \wedge (\neg x_2 \vee \neg x_1)$

Rešitev: $x_1=False, x_2=True, x_3=False$

$x_1=False, x_2=True, x_3=True$

- $(\neg x_1 \vee x_3) \wedge (x_1 \vee x_2) \wedge (\neg x_3 \vee x_1) \wedge (\neg x_1 \vee \neg x_3)$

Rešitev: $x_1=False, x_2=True, x_3=False$

- $(\neg x_1 \vee x_3) \wedge (x_1 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (x_1 \vee x_3)$

Rešitev za ta problem ne obstaja.

OK

- Vse vrednosti nastavimo na 0. Potem vzamemo v stavku prvo vrednost in jo postavimo na True ali False. Vse te vrednosti postavimo na True oz. False. Te vrednosti postavljamo od zacetka do konca, dokler niso vsi stavki zadovoljeni.

Če se Petru to zgodi, mora vrednost, ki je že nastavljena spremeniti in vse od začetka preveriti če je vredu.

Pogledati mora vse možnosti spremenljivk, ki nastopajo.

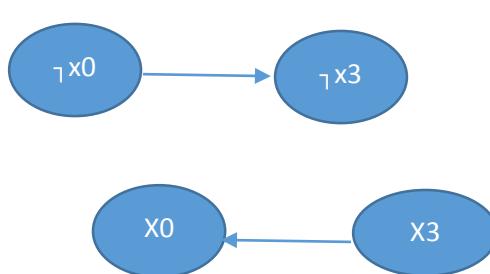
Ideja je ok, vendar manjka konkreten algoritem, ki reši podan 2-SAT.

- Časovna zahtevnost je $O(2^n)$. n -je število spremenljivk.

OK

Graf izrišemo tako, da najprej napisemo vsa vozlišča. Vozlišča so x_1, x_2, x_3, \dots . Nato gradimo povezave. Povezava je: $(x_0 \vee \neg x_3) \equiv (\neg x_0 \Rightarrow \neg x_3) \equiv (x_3 \Rightarrow x_0)$.

To pomeni da povežemo $\neg x_0$ z $\neg x_3$ in x_3 z x_0 .



Kako rešimo z zgornjim grafom 2-SAT?

naloga 5.3

Problem 2-SAT. V študiju ste že spoznali pojma konjunkcij (logični in) in disjunkcije (logični ali) ter iz njiju izvirajoč pojem konjunktivne oblike izraza, ki pomeni konjunkcijo disjunkcij spremenljivk. Iskanje takšnih vrednosti posameznih spremenljivk (true, false), da izračunajo celoten izraz v true, če je to mogoče, pravimo zadovoljitev izraza (angl. satisfiability ali SAT). Posebno obliko problema SAT, kjer sta v disjunkcijah le po dve spremenljivki, imenujemo 2-SAT. Sami disjunkciji rečemo stavek in ga v splošnem zapišemo (x_1, x_2) - očitno je lahko tako x_1 ali x_2 (ali oba) negiran ali pa ne.

Vprašanja:

5.3.1 V naslednjih izrazih poiščite vse vrednosti spremenljivk x_1 , x_2 in x_3 , ki izračunajo izraz na true.

$$\begin{aligned} & (x_1 \vee \neg x_2) \wedge (x_2 \vee \neg x_3) \wedge (x_3 \vee \neg x_1) \\ & (\neg x_1 \vee x_2) \wedge (x_1 \vee x_2) \wedge (\neg x_2 \vee \neg x_1) \\ & (\neg x_1 \vee x_3) \wedge (x_1 \vee x_2) \wedge (\neg x_3 \vee x_1) \wedge (\neg x_1 \vee \neg x_3) \\ & (\neg x_1 \vee x_3) \wedge (x_1 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (x_1 \vee x_3) \end{aligned}$$

Izraz: $(x_1 \vee \neg x_2) \wedge (x_2 \vee \neg x_3) \wedge (x_3 \vee \neg x_1)$

x_1	x_2	x_3	$x_1 \vee \neg x_2$	\wedge	$x_2 \vee \neg x_3$	\wedge	$x_3 \vee \neg x_1$	
0	0	0	1	1	1	1	1	T
0	0	1	1	0	0	0	1	
0	1	0	0	0	1	0	1	
0	1	1	0	0	1	0	1	
1	0	0	1	1	1	0	0	
1	0	1	1	0	0	0	1	
1	1	0	1	1	1	0	0	
1	1	1	1	1	1	1	1	T

Rešitev: $x_1=1; x_2=1; x_3=1$

Izraz: $(\neg x_1 \vee x_2) \wedge (x_1 \vee x_2) \wedge (\neg x_2 \vee \neg x_1)$

x_1	x_2	$\neg x_1 \vee x_2$	\wedge	$x_1 \vee x_2$	\wedge	$\neg x_2 \vee \neg x_1$	
0	0	1	0	0	0	1	
0	1	1	1	1	1	1	T
1	0	1	1	1	1	1	T
1	1	1	1	1	1	1	T

Rešitev: $x_1=0; x_2=1$

Izraz: $(\neg x_1 \vee x_3) \wedge (x_1 \vee x_2) \wedge (\neg x_3 \vee x_1) \wedge (\neg x_1 \vee \neg x_3)$

x_1	x_2	x_3	$\neg x_1 \vee x_3$	\wedge	$x_1 \vee x_2$	\wedge	$\neg x_3 \vee x_1$	\wedge	$\neg x_1 \vee \neg x_3$	
0	0	0	1	0	0	0	1	1	1	
0	0	1	1	0	0	0	0	0	1	
0	1	0	1	1	1	1	1	1	1	T
0	1	1	1	1	1	0	0	0	1	
1	0	0	0	0	1	0	1	1	1	
1	0	1	1	1	1	0	1	0	0	
1	1	0	0	0	1	0	1	1	1	
1	1	1	1	1	1	0	1	0	0	

Rešitev: $x_1=0; x_2=1; x_3=0$

Izraz: $(\neg x_1 \vee x_3) \wedge (x_1 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (x_1 \vee x_3)$

x_1	x_3	$\neg x_1 \vee x_3$	\wedge	$x_1 \vee \neg x_3$	\wedge	$\neg x_1 \vee \neg x_3$	\wedge	$x_1 \vee x_3$	
0	0	1	1	1	1	1	0	0	F
0	1	1	0	0	0	1	0	1	F
1	0	0	0	1	0	1	0	1	F
1	1	1	1	1	0	0	0	1	F

Rešitev: $x_1=\text{nima rešitve}; x_3=\text{nima rešitve}$

OK

5.3.2 Peter Zmeda je dobil za nalogo, da napiše algoritem, ki reši problem 2-SAT. Nekaj časa je gledal problem, potem brskal po knjigi in zapiskih ter našel magično besedo - požrešna metoda. Tako je pričel pisati idejo algoritma nekako takole:

Na začetku ima vsaka spremenljivka bodisi vrednost true ali false.

Vzemi prvi stavek (x_1, x_2), ki še ni nezadovoljen, ter določi vrednost spremenljivke x_1 (ali x_2) tako, da se stavek izračuna na true - postane zadovoljen.

Nastavi še vrednosti spremenljivke v vseh ostalih stavkih, kjer nastopa x_1 (ali x_2), in izračunaj posamezne stavke: nekateri bodo postali zadovoljeni.

Ponovi korak 2.

Poglejmo si primer $(x_1 \vee \neg x_2) \wedge (x_2 \vee \neg x_1)$:

na začetku sta oba stavka $(x_1 \vee \neg x_2)$ in $(x_2 \vee \neg x_1)$ še nedoločena in, recimo, začnimo pri stavku $(x_1 \vee \neg x_2)$: naj bo x_1 true ter stavek $(x_1 \vee \neg x_2)$ postane zadovoljen;

nato v stavku $(x_2 \vee \neg x_1)$ dobimo $(x_2 \vee \text{false})$, kar pomeni, da moramo nastaviti tudi x_2 na true;

preverimo prvi stavek in smo uspeli najti pravo nastavitev. Osnovni izziv Petrovega pristopa je v 3. koraku, saj bodo nekateri stavki postali zadovoljeni, kar je super; pri nekaterih stavkih bo ena spremenljivka še nenastavljena, kar se bo rešilo kasneje; največji problem pa so stavki, kjer sta obe spremenljivki določeni in stavek se izračuna na false. Kaj pa sedaj?

Pomagajte Petru, da sestavi celoten algoritmom zasnovan na njegovi ideji. Napišite psevdokodo algoritma.

```
boolean izracun(){
    boolean x1,x2,x3;
    int n=3;

    boolean []tab_x1={false,false,false,false,true,true,true,true};
    boolean []tab_x2={false,false,true,true,false,true,true};
    boolean []tab_x3={false,true,false,true,false,true,true};

    for(int i=0; i<Math.pow(2, n); i++){
        x1=tab_x1[i];
        x2=tab_x2[i];
        x3=tab_x3[i];

        if((!x1||x3)&&(x1||!x3)&&(!x1||!x3)&&(x1||x3))
            return true;
    }
    return false;
}
```

Kaj pa za poljubno število spremenljivk in stavkov?

5.3.3 Kakšna je časovna zahtevnost vašega algoritma glede na število spremenljivk?

Časovna kompleksnost je: $O(2^n)$... $n=\text{št.spremenljivk}$

5.3.4 Ker je domača naloga pri poglavju o grafih, se da izraz preoblikovati v graf? Kaj bi bile povezave in kaj vozlišča? Poiščite algoritmom, ga opišite, utemeljite njegovo pravilnost in analizirajte njegovo časovno zahtevnost.

Naloga 5.3

Naloga 5.3.1)

$-(x_1 \vee \neg x_2) \wedge (x_2 \vee \neg x_3) \wedge (x_3 \vee \neg x_1) \rightarrow$ vse true ali vse false

$-(\neg x_1 \vee x_2) \wedge (x_1 \vee x_2) \wedge (\neg x_2 \vee \neg x_1) \rightarrow$ 1x false, 2x true

$-(\neg x_1 \vee x_3) \wedge (x_1 \vee x_2) \wedge (\neg x_3 \vee x_1) \wedge (\neg x_1 \vee \neg x_3) \rightarrow$ 1x false, 2x true, 3x false

$-(\neg x_1 \vee x_3) \wedge (x_1 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (x_1 \vee x_3) \rightarrow$ ni rešitve

OK

5.3.2)

Popravljen Petrov algoritem:

-Vzamemo prvi stavek, ki je še nezadovoljen, ter določimo vrednost ene spremenljivke(x_1, x_2, \dots, x_n), da bo stavek zadovoljen.

-Spremenimo vrednost spremenljivke še v ostalih izrazih

-Rešujemo izraze dokler niso vsi rešeni

-Če so vsi izrazi rešeni se algoritem zaključi, drugače gremo od začetka in si izberemo prvi nezadovoljni stavek z drugo spremenljivko in ponovimo algoritem

-Algoritem se ponavlja dokler nismo rešili izraza ali smo obhodili vse spremenljivke in nismo prišli do rešitve.

5.3.4)

Kaj pomeni "obhodili" vse spremenljivke? Potrebno je preizkusiti vse kombinacije vrednosti s

Da. Vozlišča so v tem primeru vse spremenljivke v izrazu in njihove negacije. Povezava iz vozlišča A v vozlišče B pa obstaja, če obstaja izraz $\neg A \vee B$.

In kako s tem rešimo problem 2-SAT?

Naloga 5.3.1

x_1	x_2	x_3	$(x_1 \vee \neg x_2) \wedge (x_2 \vee \neg x_3) \wedge (x_3 \vee \neg x_1)$
0	0	0	1
0	0	1	0 (sredinski del je 0)
0	1	0	0 (prvi del je 0)
0	1	1	0 (prvi del je 0)
1	0	0	0 (tretji del je 0)
1	0	1	0 (sredinski del je 0)
1	1	0	0 (tretji del je 0)
1	1	1	1
			Rešitvi sta torej (False, False, False) in (True, True, True)

x_1	x_2	x_3	$(\neg x_1 \vee x_2) \wedge (x_1 \vee x_2) \wedge (\neg x_2 \vee \neg x_1)$
0	0	0/1	0 (sredinski del je 0)
0	1	0/1	1
1	0	0/1	0 (prvi del je 0)
1	1	0/1	0 (tretji del je 0)
			Rešitvi sta torej (False, True, False) in (False, True, True)

x_1	x_2	x_3	$(\neg x_1 \vee x_3) \wedge (x_1 \vee x_2) \wedge (\neg x_3 \vee x_1) \wedge (\neg x_1 \vee \neg x_3)$
0	0	0	0 (drugi del je 0)
0	0	1	0 (drugi del je 0)
0	1	0	1
0	1	1	0 (tretji del je 0)
1	0	0	0 (prvi del je 0)
1	0	1	0 (zadnji del je 0)
1	1	0	0 (prvi del je 0)
1	1	1	0 (zadnji del je 0)
			Rešitev je torej (False, True, False)

x_1	x_2	x_3	$(\neg x_1 \vee x_3) \wedge (x_1 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (x_1 \vee x_3)$
0	0	0	0 (zadnji del je 0)
0	0	1	0 (drugi del je 0)
0	1	0	0 (zadnji del je 0)
0	1	1	0 (drugi del je 0)
1	0	0	0 (prvi del je 0)
1	0	1	0 (tretji del je 0)
1	1	0	0 (prvi del je 0)
1	1	1	0 (tretji del je 0)
			Ni rešitve

OK

Naloga 5.3.2

Ker gre za požrešno metodo, sklepam da se proba vse kombinacije true/false vrednosti na spremenljivkah x_1, x_2, \dots, x_n . Tako da za vsako kombinacijo vrednosti, ko smo šli čez vsak posamezni stavek, če imajo vsi vrednost true, dodamo kombinacije spremenljivk v seznam kombinacij, in probamo naslednjo kombinacijo.

Torej, nekako takole:

- Imamo seznam kombinacij spremenljivk x_1, x_2, \dots, x_n ;
 - Za vsako kombinacijo:
 - Boolean trenutni=true;
 - Za vsak stavek:
 - Če je trenutni false;
 - Break, preveri naslednjo kombinacijo;
 - če je stavkova končna vrednost false:
 - trenutni=false;

OK

Naloga 5.3.3

Časovna zahtevnost je $O(2^n)$, ker imamo za vsako dodatno spremenljivko 2x več kombinacij.

OK