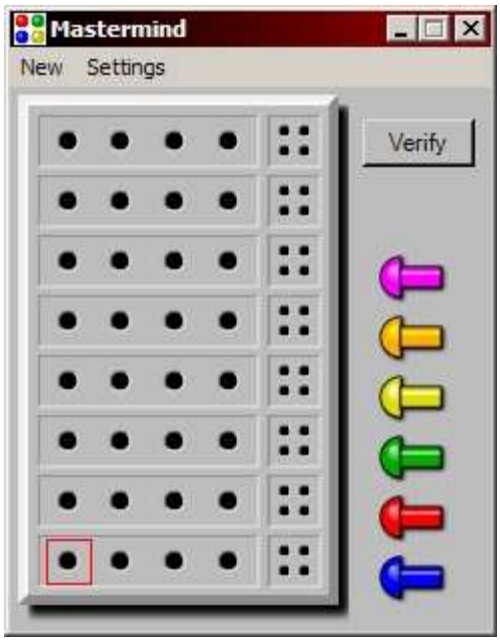
Zápočtový program pre zimný semester 2006/2007
z predmetu PRG005 Neprocedurální programování

Mastermind (Logik)

autor: Peter Černo, 2. ročník, paralelka I-2-Y37
email: petercerno@gmail.com
vývojové prostredie: SWI-Prolog
zdrojové súbory: mastermind.pl

Predpokladám, že čitateľ pozná hru mastermind (logik). Hlavnou úlohou môjho programu mastermind je dávať hráčovi dobré rady (v tom zmysle, aby sa čo najrýchlejšie dopracoval k správnej odpovedi). Na úvod by som začal ukážkou toho, čo program dokáže.

Predpokladajme, že hráme so 6 farbami a tipujeme štvorice farieb (TJ máme 4 stĺpce, v mojom programe ich nazývam tokeny).



Farby si najskôr očíslujeme:
farba číslo 1 je modrá,
farba číslo 2 je červená,
farba číslo 3 je zelená,
farba číslo 4 je žltá,
farba číslo 5 je oranžová a
farba číslo 6 je ružová.

Program mastermind spustíme zavolaním predikátu `mastermind.`
Zobrazí sa nám niečo ako:

```
vitajte v programe mastermind
autor: Peter Cerno
email: petercerno[at]gmail.com
zadajte pocet farieb:
```

Hneď prvá otázka je počet farieb. Zadáme `6.` (nesmieme zabudnúť napísať bodku)

```
zadajte pocet stlpcov:
```

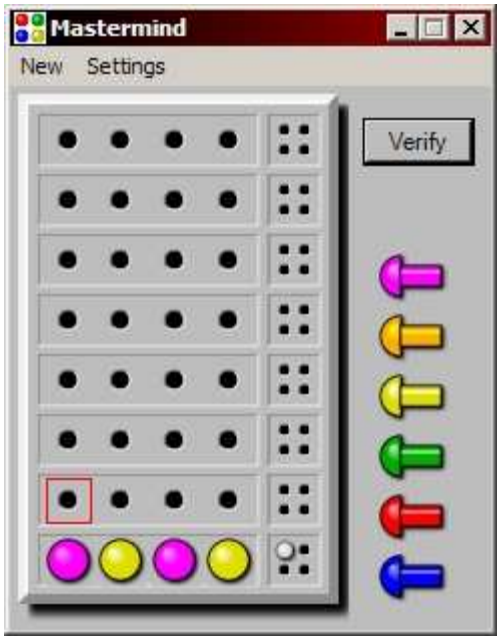
Ako počet stĺpcov zadáme `4.`

```
zadajte tip/prikaz:
```

Prvý príkaz, ktorý si vyskúšame je príkaz `hint.` Týmto príkazom žiadame program o radu. Program nám odporučí najviac tak 10 tipov, z ktorých si potom môžeme vybrať. V mojom prípade vypísal niečo takéto:

```
[6, 4, 6, 4] [6, 2, 2, 6] [4, 6, 6, 4] [3, 6, 6, 3] [3, 5, 5, 3]
[2, 1, 1, 2] [1, 1, 6, 6] [6, 6, 4, 3] [6, 6, 2, 5] [6, 5, 1, 1]
```

Tipy sú zoradené od najlepšieho. Vyskúšame preto napríklad hneď prvý.
Zadáme `[6, 4, 6, 4].` TJ vyskúšame štvoricu (ružová, žltá, ružová, žltá).



Vidíme, že sme trafili jednu farbu, ktorá ale nie je na správnom mieste.

```
zadajte odpoved/prikaz:
```

Program čaká na odpoveď, ktorá je v našom prípade `answer(0,1).`

Prvé číslo udáva počet čiernych zásahov (TJ zásah, kde sme správne určili farbu aj pozíciu),
druhé číslo udáva počet bielych zásahov (TJ zásah, kde sme správne určili farbu, ale nie pozíciu).

Pokiaľ by sa nám nechcelo písať slovo `answer`, kľudne môžeme odpoveď skrátiť na `(0,1).`

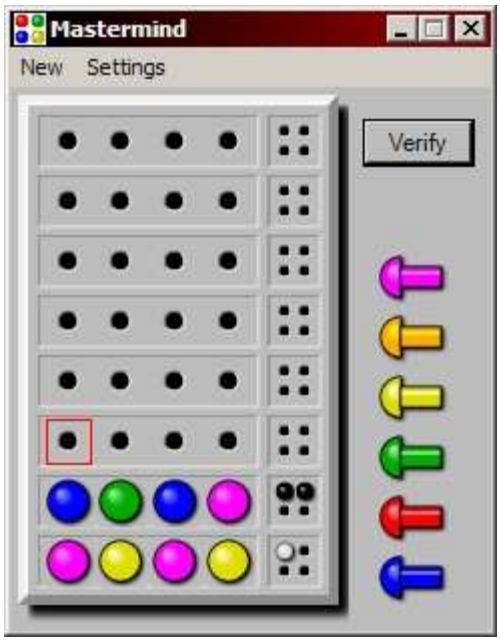
```
zadajte tip/prikaz:
```

Znovu zadáme príkaz `hint.`

V mojom prípade program vypísal tieto tipy:

```
[1, 3, 1, 6] [1, 6, 1, 5] [2, 3, 4, 3] [2, 5, 2, 6] [2, 5, 4, 5]
[2, 6, 2, 1] [3, 1, 4, 1] [3, 2, 4, 2] [3, 6, 3, 5] [4, 1, 5, 1]
```

Opäť zadáme prvú možnosť, TJ tip `[1, 3, 1, 6].`
ktorý predstavuje štvoricu (modrá, zelená, modrá, ružová).




```
zadajte odpoved/prikaz:
```

Ako odpoveď zadáme `answer(2,0)`.
Máme totiž dva čierne body a žiaden biely.

```
zadajte tip/prikaz:
```

Tu sa na chvíľku pozastavíme.
Vyskúšame ďalší príkaz, a to je príkaz `moves`.
Tento príkaz nám zobrazí doteraz odohrané ťahy.

```
[6, 4, 6, 4] answer(0, 1)
[1, 3, 1, 6] answer(2, 0)
```

Tu vidíme, že sme odohrali 4 ťahy (tip a odpoveď počítam za dva ťahy).
Ako posledný je uvedený najneskôr odohraný ťah.
Ďalej si môžeme nechať vypísať množinu možných správnych odpovedí pomocou `show`.

```
[1, 2, 2, 6] [1, 2, 5, 6] [1, 3, 4, 2] [1, 3, 4, 3] [1, 3, 4, 5]
[1, 5, 2, 6] [1, 5, 5, 6] [2, 2, 1, 6] [2, 3, 2, 6] [2, 3, 3, 6]
[2, 3, 5, 6] [2, 5, 1, 6] [3, 3, 2, 6] [3, 3, 3, 6] [3, 3, 5, 6]
[4, 3, 1, 2] [4, 3, 1, 3] [4, 3, 1, 5] [5, 2, 1, 6] [5, 3, 2, 6]
[5, 3, 3, 6] [5, 3, 5, 6] [5, 5, 1, 6]
```

Naše 4 odohrané ťahy predstavujú akési obmedzujúce podmienky pre to, ako môže vyzerať správna odpoveď. Tu vidíme zoznam všetkých štvoríc, ktoré vyhovujú týmto obmedzujúcim podmienkam.
Ako príklad si vezmime štvoricu `[1, 1, 1, 1]`, ktorá sa v zozname nenachádza.
Táto štvorica rozhodne nemôže byť správna odpoveď, pretože nevyhovuje obmedzujúcej podmienke `[6, 4, 6, 4] answer(0, 1)`.
Keby štvorica `[1, 1, 1, 1]` bola správnou odpoveďou, museli by sme na tip `[6, 4, 6, 4]` dostať odpoveď `answer(0, 0)`.

Vidíme, že možných správnych odpovedí je dohromady 23. Príkaz `hint.` nám z nich vyberie desať najlepších tipov.

```
[5, 2, 1, 6] [2, 5, 1, 6] [1, 5, 2, 6] [1, 2, 5, 6] [5, 5, 1, 6]
[2, 2, 1, 6] [1, 5, 5, 6] [1, 2, 2, 6] [5, 3, 2, 6] [2, 3, 5, 6]
```

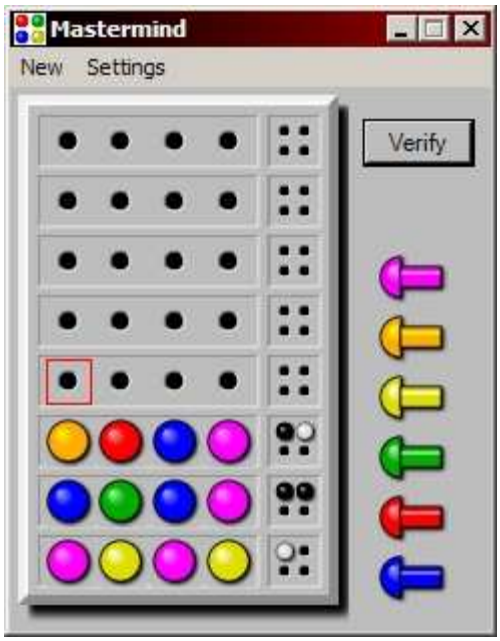
Necháme si poradiť a zadáme `[5, 2, 1, 6]`.

```
zadajte odpoved/prikaz:
```

Zadáme `answer(1,1).`

```
zadajte tip/prikaz:
```

Keď si dáme znovu vypísať možné správne odpovede pomocou príkazu `show.` dostaneme už iba šesť možností:




```
[4, 3, 1, 5] [4, 3, 1, 2] [3, 3, 5, 6] [3, 3, 2, 6] [2, 3, 3, 6]
[2, 3, 2, 6]
```

Príkaz `hint.` nám ich usporiada:

```
[2, 3, 3, 6] [3, 3, 5, 6] [2, 3, 2, 6] [3, 3, 2, 6] [4, 3, 1, 2]
[4, 3, 1, 5]
```

Vyskúšame `[2, 3, 3, 6].`

Zadáme odpoveď:

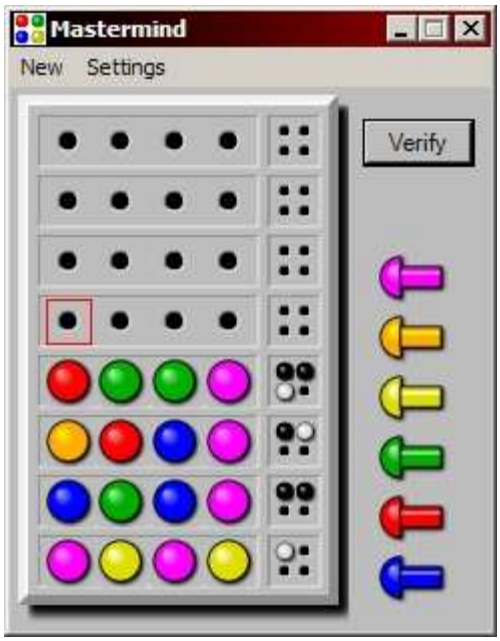
```
zadajte odpoved/prikaz: (2,1).
```

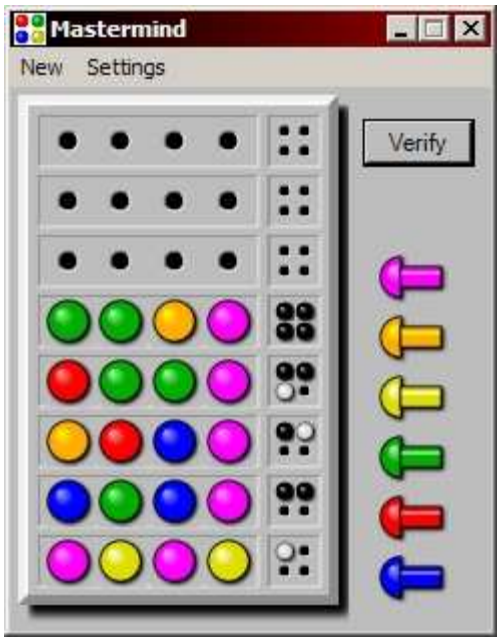
A po zadaní odpovedi si znovu dáme vypísať všetky potenciálne správne tipy pomocou `show.`

Dostaneme už iba jednu možnosť:

```
[3, 3, 5, 6]
```

To je zároveň víťazná kombinácia.





V priemere sa nám podarí pomocou môjho programu mastermind uhádnuť štvoricu farieb tak na 4. až 5. pokus. Ešte sa mi nepodarilo prehrať, aj keď vylúčiť sa to nedá.

Všimnite si, že počas hry sa nám striedali dva prompty:

```
zadajte tip/prikaz:

zadajte odpoved/prikaz:
```

Oba tieto prompty prijímajú nejakú množinu príkazov, ktorých popis nájdete, keď zadáte `help`.
Medzi dôležité príkazy patria napríklad:
`quit.` ukončí program
`back.` vráti ťah
`show.` v prípade 1. promptu zobrazí všetky potenciálne správne tipy,
v prípade 2. promptu zobrazí všetky možné odpovede

Súčasťou programu je aj predikát `mastermindgame,` ktorý vám umožní zahrať si hru mastermind bez akýchkoľvek pomôcok. Víťaznú farebnú kombináciu program vyberie náhodne. Môžete tiež zadať `mastermind(CorrectTuple).` čím si vynútite, aby víťazná farebná kombinácia bola `CorrectTuple`.

Základné dátové štruktúry a predikáty

Medzi základné dátové štruktúry patrí farba `Color`, čo je číslo v rozsahu `1..MaxColor`.
Ďalej n-tica farieb `ColorTuple` je zoznam, ktorý pozostáva z farieb `Color` a má práve `MaxToken` členov. `MaxColor` udáva počet farieb a `MaxToken` udáva počet stĺpcov (tokenov).

`ColorSet` je vždy nejaký zoznam farebných n-tíc (`ColorTuple`). Obyčajne to je zoznam potenciálne správnych odpovedí (to čo vypisuje príkaz `show`).
Na začiatku je to zoznam všetkých farebných n-tíc. (dohromady `MaxColor^MaxToken` členov)

Ďalej máme štruktúru `answer(BlackPoints, WhitePoints)`, ktorá predstavuje odpoveď na nejaký farebný tip. `BlackPoints` je číslo, ktoré udáva počet farieb, ktoré sú určené správne a na správnej pozícii, `WhitePoints` je číslo, ktoré udáva počet farieb, ktoré sú určené správne, ale na nesprávnej pozícii.

Dôležitou dátovou štruktúrou je `response(Answer, ColorSubSet, Count)`.
Je pomerne ťažké ju definovať. Odporúčam vám pozrieť si príklady.
Predpokladajme, že máme nejakú množinu potenciálne správnych odpovedí `ColorSet` a zvolili sme si nejaký farebný tip `ColorTip`. Tento farebný tip vôbec nemusí byť z množiny `ColorSet`.
Vezmime si ľubovoľný prvok `ColorTuple` z množiny `ColorSet`. Predpokladajme, že `ColorTuple` je správna farebná kombinácia a `Answer = answer(BlackPoints, WhitePonts)` je príslušnou odpoveďou na náš tip `ColorTip`.
Potom `ColorTuple` patrí do množiny `ColorSubSet` v predikáte `response(Answer, ColorSubSet, Count)`.
Inak povedané, `ColorSubSet` je podmnožina množiny `ColorSet` a predstavuje množinu potenciálne správnych odpovedí, za predpokladu, že na farebný tip `ColorTip` je daná odpoveď `Answer`. Napokon `Count` udáva už iba veľkosť množiny `ColorSubSet`.
Poznámka: Dvojica `ColorTip, Answer` predstavuje obmedzujúcu podmienku. Množina `ColorSubSet` je potom najväčšou podmnožinou množiny `ColorSet` tak, aby všetky jej prvky spĺňali túto obmedzujúcu podmienku.

Príklad všetko osvetlí:

Predpokladajme, že máme 2 farby a štyri stĺpce.
`ColorSet` je potom zoznam pozostávajúci z týchto prvkov:

```
[1, 1, 1, 1] [1, 1, 1, 2] [1, 1, 2, 1] [1, 1, 2, 2] [1, 2, 1, 1]
[1, 2, 1, 2] [1, 2, 2, 1] [1, 2, 2, 2] [2, 1, 1, 1] [2, 1, 1, 2]
[2, 1, 2, 1] [2, 1, 2, 2] [2, 2, 1, 1] [2, 2, 1, 2] [2, 2, 2, 1]
[2, 2, 2, 2]
```

Vezmime si teraz napríklad tip `[1, 2, 2, 1]`.
Možné odpovede na tento tip sú iba tieto:

```
1. answer(1, 2)
2. answer(3, 0)
3. answer(2, 2)
4. answer(2, 0)
5. answer(4, 0)
6. answer(0, 4)
```

Uvažujme odpoveď `answer(1, 2)`. Nasledujúce štvorice

```
[2, 2, 1, 2] [2, 1, 2, 2] [2, 1, 1, 1] [1, 1, 1, 2]
```

predstavujú všetky potenciálne správne odpovede za predpokladu, že na tip `[1, 2, 2, 1]` je daná odpoveď `answer(1, 2).` (Skúste si to overiť)

Ďalej si popíšeme niektoré základné predikáty:

`color(Color, +MaxColor).`
Naunifikuje `Color` s hodnotou `1..MaxColor`.
Pri `fail`ovaní prejde postupne všetky možnosti.

`colortuple(ColorTuple, +MaxToken, +MaxColor).`
Naunifikuje `ColorTuple` so zoznamom dĺžky `MaxToken` pozostávajúci z farieb `1..MaxColor`.
Pri `fail`ovaní prejde postupne všetky možnosti.

`randomtuple(-ColorTuple, +MaxToken, +MaxColor).`
Nauinfikuje `ColorTuple` s nejakým náhodným zoznamom dĺžky `MaxToken` pozostávajúci z farieb `1..MaxColor`.

`colorset(-ColorSet, +MaxToken, +MaxColor).`
Vytvorí množinu `ColorSet` `MaxToken`-tíc pozostávajúcich z farieb `1..MaxColor`.

`getanswer(+ColorTuple1, +ColorTuple2, -Answer).`
Porovná n-tice farieb `ColorTuple1` a `ColorTuple2`
`ColorTuple1` je napríklad náš tip a
`ColorTuple2` je správna farebná kombinácia.
`Answer` zunifikuje s `answer(BlackPoints, WhitePoints)`, kde
`BlackPoints` udáva na koľkých miestach sa zhodujú farby
`WhitePoints` udáva, koľko farieb je správne určených, ale
na nesprávnom mieste.

`getresponses(+ColorTip, +ColorSet, -Responses).`
Pre daný farebný tip a množinu kandidátov vráti zoznam trojíc
`response(Answer, ColorSubSet, Count)`
zotriedený zostupne podľa `Count`, kde
`Answer` predstavuje možnú odpoveď,
`ColorSet` predstavuje množinu kandidátov takých, že odpoveď na `ColorTip` je `Answer`,
`Count` je veľkosť množiny `ColorSet`.

```
colortipcandidates(+ColorTipCandidates, +ColorSet,
  +MaxBuffer -Candidates).
```
Z množiny kandidátov `ColorTipCandidates` vyberie podmnožinu `Candidates`, ktorá obsahuje najviac `MaxBuffer` prvkov (v našom prípade 10).
Predpokladáme, že `ColorSet` je množina potenciálne správnych farebných tipov (TJ v množine `ColorSet` sa určite nachádza víťazný farebný tip)
Pre každého kandidáta z množiny `ColorTipCandidates` sa vypočíta ohodnotenie (na základe nejakej heuristiky) a do množiny `Candidates` sa dostanú iba tí najlepší kandidáti.
V prípade, že má množina `ColorTipCandidates` viac ako 100 prvkov, vyberie sa z nej náhodne 100 prvkov a ostatné sa zahodia (kvôli urýchleniu výpočtu).
Tento predikát používa pre nás už známy príkaz `hint`.

Nakoniec nám zostáva už iba popísať heuristiku, podľa ktorej sa určuje ohodnotenie jednotlivých kandidátov.

Opäť predpokladajme, že máme 2 farby a štyri stĺpce.
`ColorSet` je potom zoznam pozostávajúci z týchto prvkov:

```
[1, 1, 1, 1] [1, 1, 1, 2] [1, 1, 2, 1] [1, 1, 2, 2] [1, 2, 1, 1]
[1, 2, 1, 2] [1, 2, 2, 1] [1, 2, 2, 2] [2, 1, 1, 1] [2, 1, 1, 2]
[2, 1, 2, 1] [2, 1, 2, 2] [2, 2, 1, 1] [2, 2, 1, 2] [2, 2, 2, 1]
[2, 2, 2, 2]
```

Otázka znie, ako určiť, ktoré farebné tipy sú lepšie a ktoré sú horšie.

Vezmime si napríklad tip `ColorTip = [1, 1, 1, 1]`.
Keď roztriedime množinu `ColorSet` pomocou predikátu
`getresponses(+ColorTip, +ColorSet, -Responses)`,
dostaneme niečo takéto:

```
answer(2, 0)
[2, 2, 1, 1] [2, 1, 2, 1] [2, 1, 1, 2] [1, 2, 2, 1] [1, 2, 1, 2]
[1, 1, 2, 2]

answer(3, 0)
[2, 1, 1, 1] [1, 2, 1, 1] [1, 1, 2, 1] [1, 1, 1, 2]

answer(1, 0)
[2, 2, 2, 1] [2, 2, 1, 2] [2, 1, 2, 2] [1, 2, 2, 2]

answer(4, 0)
[1, 1, 1, 1]

answer(0, 0)
[2, 2, 2, 2]
```

Vidíme, že k odpovedi `answer(2, 0)` má príslušná množina `ColorSubSet` až šesť prvkov.

Keď uvažujeme tip `ColorTip = [1, 1, 2, 2]`, potom dostaneme niečo takéto:

```
answer(3, 0)
[2, 1, 2, 2] [1, 2, 2, 2] [1, 1, 2, 1] [1, 1, 1, 2]

answer(1, 2)
[2, 2, 2, 1] [2, 2, 1, 2] [2, 1, 1, 1] [1, 2, 1, 1]

answer(2, 2)
[2, 1, 2, 1] [2, 1, 1, 2] [1, 2, 2, 1] [1, 2, 1, 2]

answer(2, 0)
[2, 2, 2, 2] [1, 1, 1, 1]

answer(4, 0)
[1, 1, 2, 2]

answer(0, 4)
[2, 2, 1, 1]
```

Tu vidíme, že máme viac možných odpovedí (šesť) a v najhoršom prípade to dopadne tak, že budeme mať 4 potenciálne správne odpovede. To je ďaleko lepšie ako v predchádzajúcom prípade, kde sa nám môže stať, že budeme mať až 6 potenciálne správnych odpovedí.

Inak povedané, v prvom prípade sa nám môže stať, že na tip `[1, 1, 1, 1]` dostaneme odpoveď `answer(2, 0)` a budeme musieť preverovať množinu 6 kandidátov.
V druhom prípade sa nám v najhoršom prípade môže stať, že na tip `[1, 1, 2, 2]` dostaneme napríklad odpoveď `answer(2, 2)`, takže budeme musieť preveriť 4 kandidátov.

Z tohto pohľadu je tip `[1, 1, 1, 1]` horší ako tip `[1, 1, 2, 2]`.
Heuristika použitá v mojom programe je zhruba o tom, že najlepšie sú tie tipy, ktoré dávajú čo najviac možných odpovedí `answer` a množiny prislúchajúce k jednotlivým odpovediam majú čo najrovnomernejšie rozložené počty prvkov.
Z čisto štatistického hľadiska je pravdepodobnosť danej odpovede `answer` úmerná počtu prvkov príslušnej množiny.
Inak povedané, je väčšia pravdepodobnosť, že na tip `[1, 1, 1, 1]` dostaneme odpoveď `answer(2, 0)`, než odpoveď `answer(4, 0)`.
Z tohto dôvodu je potrebné hľadať tipy s minimálnym počtom prvkov najväčšej množiny.

K úlohe sa dá pristúpiť aj inak. Pri hľadaní najlepšieho tipu môžeme použiť MiniMax. Je to podobné ako hra dvoch hráčov. Prvý hráč vyberá tipy, druhý hráč vyberá odpovede. Prvý sa snaží minimalizovať počet potenciálne správnych odpovedí, druhý sa naopak snaží tento počet maximalizovať. Bohužiaľ, vzhľadom k tomu, že prvý hráč má často príliš veľa možností ako ťahať, nie je tento prístup tak ľahko realizovateľný. Naviac predpokladám, že takéto prehľadávanie by neprinieslo výrazné zlepšenie. Heuristika, ktorú som použil, nie je až tak časovo náročná a napriek tomu dáva veľmi dobré tipy.

Pokiaľ chcete vedieť, ako sú jednotlivé predikáty implementované, potom vám rozhodne odporúčam nahliadnuť do zdrojového kódu. Komentáre by vám mali pomôcť zorientovať sa.