FLP 2023/2024 – logický projekt: Prolog

Toto je zadání logického projektu do předmětu Funkcionální a logické programování 2023/2024. Za projekt zodpovídá **Ing. Radek Hranický, Ph.D.** Konzultace jsou možné osobně po předchozí domluvě, prostřednictvím diskusního fóra předmětu, které najdete v IS VUT v sekci e-Learning, případně elektronickou poštou (`hranicky@fit.vut.cz`).

Obecné pokyny

Cílem projektu je v jazyce SWI Prolog implementovat řešení jednoho z následujících problémů:

- Turingův stroj
- Rubikova kostka
- Babylonská věž
- Kostra grafu
- Hamiltonovská kružnice
- (Vlastní zadání)

Jednotlivé varianty jsou blíže popsány v tomto dokumentu. V případě, že předmět opakujete a nebylo Vám garantem uznáno loňské řešení, je nutné **zvolit jinou variantu zadání, než jste měli loni!**

U všech zadání je kladen důraz především na správné chování v případě správného vstupu. Pokud existuje cesta k vyřešení problému, Váš program ji musí nalézt. Existuje-li více cest, může Váš program zvolit libovolnou z nich (Výjimkou je extrémně velký počet tahů u Rubikovy kostky nebo Babylonské věže). Program bude implementován tak, aby byl přeložitelný a spustitelný na serveru **`merlin`**.

Za vypracovaný projekt lze získat až 8 bodů. Minimální počet bodů pro získání zápočtu jsou 3. **Hodnocena bude míra splnění zadání, kvalita řešení, čistota a kvalita kódu a vhodné užití komentářů. Předmětem hodnocení je také dokumentace.** Za inovativní přístup, obzvláště kvalitní řešení, či rozšíření nad rámec zadání, lze získat prémiové body navíc.

Projekt je nutné vypracovat samostatně – duplikáty jsou kontrolovány i strojově a pokus o opsání nebo přílišnou "inspiraci" cizím kódem je velmi dobře poznat! Případné nejasnosti konzultujte včas, na námitky po odevzdání nebude brán zřetel.

Zdrojový kód

Úvod všech zdrojových textů musí obsahovat *název projektu*, *login* a *jméno autora*. Dbejte na psaní čistého, přehledného a vhodně komentovaného kódu. Komentáře pište v jednom jazyce (česky, slovensky, anglicky). U každého predikátu v komentáři uveďte, k čemu slouží a jaký je význam jednotlivých argumentů. Doporučeno je využít standardizovaný **dokumentační zápis** predikátů (viz dokumentace prologu, či slajdy ke 4. cvičení).

Snažte se program navrhovat tak, aby se dobře ladil i četl. Tam, kde se to nabízí, využijte dynamické predikáty. Předávání všeho pouze přes argumenty může vést k příliš komplikovaným predikátům a nepřehlednému, špatně čitelnému kódu. Stejně tak z řešení prosím odstraňte veškeré pozůstatky nepoužitého/zakomentovaného kódu, nejde-li o záměr (např. implementace rozšíření, které je pro aktivaci nutné odkomentovat). Predikáty, které nejsou v programu smysluplně využity, z kódu odstraňte. V odevzdaném programu nemají, co dělat.

Povolené knihovny

Ve vašem řešení můžete využít:

- Jakékoli zdrojové kódy ze souboru **`input2.pl`**, který máte k dispozici v materiálech na systému Moodle. Soubor obsahuje implementaci predikátů pro vstup a výstup.
- Jakékoli predikáty ze **SWI-Prolog Library**, které budou fungovat na verzi `swipl`, která je nainstalována na serveru Merlin. Viz https://www.swi-prolog.org/pldoc/man?section=libpl.

Vstup a výstup

Program bude číst ze **standardního vstupu** a řešení bude vypisovat na **standardní výstup**. Program samotný přímo nečte žádné textové soubory, ani žádné nevytváří a nezapisuje do nich. Formát vstupu a výstupu je textový a je blíže upřesněn v jednotlivých variantách zadání. Poslední znak vstupu může obsahovat konec řádku. Výstup může končit také (jedním) novým řádkem.

Součástí řešení může být (jednoduchá) kontrola správnosti vstupu. Může k ní být přihlédnuto např. při nerozhodném počtu bodů. Obecně však **kontrola vstupu není předmětem zadání projektu**. Je tedy zbytečné ztrácet drahocenný čas tvorbou sofistikované kontroly. Cílem projektu je vyzkoušet si principy logického programování, nikoliv dokonalé ošetření vstupu.

Podstatné je, aby pro **korektní vstup** program vypsal **korektní výstup**.

Dejte si ale POZOR na **dodržení předepsaného formátu výstupu!** Podstatná část hodnocení bude realizována automatizovanými skripty. V případě formátu výstupu jiného než uvedeného v zadání (např. čárky, či jiné znaky místo pomlček u grafových zadání), může automatizované zpracování selhat. Stejně tak **program nesmí vypisovat nic navíc** (oddělovače, prompt `|:|:`, ladící výpisy, čas výpočtu apod.). V případě nedodržení formátu výstupu bude projekt hodnocen 0 body.

Odevzdávání

Odevzdávání probíhá výhradně prostřednictvím IS VUT. Všechny odevzdané soubory budou zkomprimovány do jediného archivu zabaleného metodou ZIP, který bude pojmenován podle vzoru:

`flp-log-{login}.zip`

Například při odevzdání projektu s loginem xnovak00 odevzdáte soubor **`flp-log-xnovak00.zip`**.

Uvnitř archivu bude přímo v kořenovém adresáři soubor **`Makefile`** pro překlad programu (projekty budou překládány pomocí `make`), cílový program pojmenujte **`flp23-log`**. Makefile ani přeložený program **neumisťujte do podadresářů**.[1] V kořenovém adresáři bude také soubor **`README.md`** se stručnou dokumentací (viz níže). Archiv bude dále obsahovat zdrojové kódy vašeho řešení. Je doporučeno také přiložit **několik vstupních souborů**, na nichž je možné váš projekt otestovat. Do dokumentace uveďte pro každý takový soubor přibližnou dobu výpočtu. Jakékoli další přiložené soubory, které zadání přímo nespecifikuje, musí mít jasnou souvislost s řešením a je potřeba je zmínit v dokumentaci. Neodevzdávejte nadbytečné soubory (Uživatelé Apple si dají pozor __MACOSX a .DS_STORE v archivu). Před odevzdáním Váš projekt vyzkoušejte na serveru `merlin`!

[1] Tzn. např. sekvencí příkazů

```
unzip flp-log-xlogin00.zip
make
./flp23-log < /cesta_k_testum/vstup1.txt > /cesta_k_vysledkum/vystup1.txt
```

dojde k rozbalení odevzdaného archivu, zkompilování projektu a spuštění na testovacích datech. Ověřte.

Rozšíření

Součástí řešení může být rozšíření (dle vlastní úvahy) nad rámec zadání projektu. Takovéto rozšíření ale **nesmí narušovat** základní funkcionalitu programu. Bude tedy aktivováno ručně, například příslušným spouštěcím parametrem programu, nebo odkomentováním příslušné části kódu. Implementace rozšíření může pozitivně ovlivnit bodové hodnocení projektu, přičemž konkrétní dopad je na uvážení opravujícího. Pokud vaše řešení obsahuje rozšíření, je potřeba tuto skutečnost zmínit v dokumentaci (viz níže), kde vysvětlíte, v čem rozšíření spočívá a jak jej aktivovat.

Dokumentace

Součástí odevzdaného archivu bude soubor README.md se stručnou dokumentací, konkrétně:

- V hlavičce uveďte Vaše jméno, login, akademický rok a název zadání.
- **Popis použité metody řešení** – Stručně popište, jak program funguje: jak jsou interně reprezentovány/kódovány stavy řešení, jaký postup pro nalezení řešení váš program používá.
- **Návod k použití** – Jak program přeložit a spustit (v souladu s pokyny). Dále prosím uveďte:
  - **Jak program spustit s vlastními vstupy** (jsou-li odevzdávány),
  - **Jak používat přiložené testovací skripty** (jsou-li odevzdávány).
- **Rozšíření** – Uveďte prosím dostatečně viditelně (např. „`ROZSIRENI: …`“) případná rozšíření nad rámec zadání a návod, jak je použít.
- **Omezení** – Pokud vaše řešení není úplné a nepodařilo se vám implementovat část zadání, doporučujeme tuto skutečnost zmínit v dokumentaci, tj. napsat, co nefunguje. Stejně tak, pokud řešení vykazuje nějaké problémy a nedostatky, např. pro určité vstupy trvá výpočet příliš dlouho, tuto skutečnost prosím uveďte. Opravujícímu tím usnadníte práci a zkrátíte tak dobu čekání na výsledky hodnocení.

Poznámka k dokumentování řešení: K jednotlivým zadáním mohou existovat různé způsoby, jak problém řešit. Snažte se, aby popis použitých metod řešení byl jasný a výstižný, aby bylo zřejmé, jaký postup jste použili. Pozitivně může být hodnoceno i zdůvodnění použitého postupu. Z dokumentace a komentářů v kódu by měl být opravující schopen zjistit, **jak váš program funguje**. Obzvláště kvalitní/zajímavý/elegantní způsob řešení může být oceněn prémiovými body. Naopak, nedostatečně komentovaný a zdokumentovaný kód může být důvodem k bodové penalizaci. Pokud váš program obsahuje nějaké rozšíření navíc, nezapomeňte to v dokumentaci zmínit.

Hodnocení

Za logický projekt můžete získat při perfektním splnění požadavků zadání až 8 bodů. Za inovativní přístup, obzvláště povedené řešení, či kvalitní rozšíření, je možné získat prémiové body navíc. Pokud si dáte práci s vytvořením automatizovaných testovacích skriptů, můžete je popsat v `README.md` a také odevzdat. Obzvláště kvalitní testy mohou pozitivně přispět k hodnocení. Naopak, pokud se vám nepodaří implementovat funkcionalitu v plném rozsahu, doporučujeme odevzdat, co se podařilo a rozsah vyřešeného popsat v přiložené dokumentaci (`README.md`). Kvalita dokumentace bude také předmětem hodnocení, zejména pak popis použité metody řešení.

Za referenční stroj je považován server `merlin`. **Nebude-li projekt přeložitelný a funkční na serveru `merlin`, nebudou uděleny body**. Podstatná část hodnocení bude realizována automatizovanými skripty. Dejte si proto pozor na dodržení správného formátu výstupu.

Automatizované testy vyzkouší váš program na sadě různých vstupů, zkoušejících různé situace. Všechny vstupy budou voleny tak, aby je i méně optimální program zvládl v řádech max. několika sekund. Pro každou situaci však bude stanoven časový limit z důvodu detekce zacyklení a extrémně nevhodných řešení.

Rovněž pečlivě komentujte zdrojové kódy. Kvalita kódu je součástí hodnotících kritérií. Doporučujeme také připojit vlastní testovací vstupy prokazující funkčnost řešení. Mohou vám zachránit body v případě, že váš program z nějakého důvodu neprojde standardními testy. Vlastní vstupy a popis dosažených výstupů připojte zejména v případě, pokud vaše řešení není úplné a naprogramovali jste například jen polovinu požadovaného algoritmu.

Velikost zásobníku

Při náročných vstupech se může stát, že vám nebude stačit výchozí velikost alokované paměti pro globální zásobník (global stack):

```
ERROR: Prolog initialisation failed:
Out of global stack
```

Tento problém je možné vyřešit zvýšením paměti pomocí parametrů překladu. V takovém případě proveďte vhodné úpravy Makefile. Parametry SWI Prolog se však pro různá prostředí (Windows, Linux) mohou lišit. Na serveru merlin viz `man swipl`. Pro přeložitelné řešení na Windows i Linux je možné do Makefile zadat např.:

```
swipl -G16g ... || swipl --stack_limit=16g ...
```

Plagiátorství

Všechna odevzdaná řešení budou testována na původnost jak mezi sebou, tak s řešeními z minulých let, i řešeními, dostupnými na Internetu. Při opakování předmětu a neuznaném projektu je nutné zvolit jinou variantu zadání. Pozor také na zrádnou AI (ChatGPT, Github Copilot, …), která se učí na kódech jiných autorů a může vám „nagenerovat cizí kód.“ V případě plagiátorství jakéhokoliv druhu bude projekt hodnocen nulovým počtem bodů, navíc nebude udělen zápočet a bude zváženo zahájení disciplinárního řízení. **Odevzdejte raději neúplné řešení než plagiát!**

Uznávání bodů z předchozího roku

Uznávání bodů za projekt z minulého roku je zcela v kompetenci doc. Koláře.

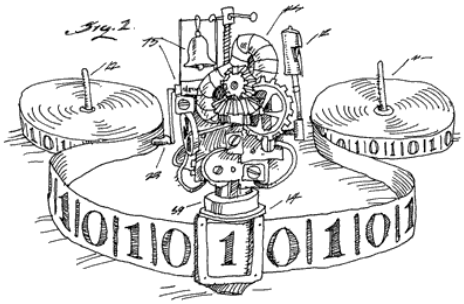
Doporučení

- Neztrácejte drahocenný čas implementací složitých kontrol syntaxe vstupu. Jejich vliv na hodnocení není významný. Raději řešte samotný problém a pokud vám zbyde čas, můžete se ke zpracování vstupu vrátit.
- Při návrhu programu můžete využít doporučení z 6. cvičení. Tj. nejprve vymyslet, jak vhodně kódovat jednotlivé stavy řešení, pak se pustit do jejich generování atd.
- Nebojte se využít dynamické predikáty tam, kde je to vhodné.
- Při ladění dynamických predikátů může přijít vhod predikát `listing(Pred)`.
- Dále se vám může hodit trasování (`trace`), režim ladění (`debug`), či Vaše vlastní ladící výpisy – ty ale nezapomeňte před odevzdáním odstranit.
- Po odevzdání si archiv z IS VUT stáhněte, nahrajte na server `merlin` a dle výše uvedeného postupu (`unzip`, `make`, …) jej vyzkoušejte. Je zbytečné přijít o body kvůli špatné struktuře odevzdaného archivu v důsledku nepozornosti.
- Nenechávejte projekt na poslední chvíli. Nemáte-li s logickým programováním bohatější zkušenosti, může vám chvíli trvat, než přijdete na vhodný postup, jak problém řešit.

Následuje popis jednotlivých variant zadání.

(verze zadání 2024-03-03)

Turingův stroj



Vaším úkolem je vytvořit simulátor nedeterministického Turingova stroje. Na vstupu váš program obdrží pravidla pro Turingův stroj a vstupní obsah pásky. Výstupem bude posloupnost konfigurací stroje při simulovaném běhu.

Vnitřní stavy stroje jsou označeny velkými písmeny (A-Z), vstupní/páskovou abecedu tvoří malá písmena (a-z), prázdný symbol je mezera, počáteční stav je „S“, koncový stav je „F“. Výpočet stroje začíná na začátku pásky a končí přechodem do koncového stavu, abnormálním zastavením, nebo zacyklením. Pravidla jsou zadána ve tvaru <stav> <symbol na pásce> <nový stav> <nový symbol na pásce nebo „L“, „R“>. Jednotlivé části jsou odděleny mezerou, každé pravidlo bude na samostatném řádku. Symboly L/R značí posun čtecí hlavy doleva/doprava. Na posledním řádku vstupu je uveden vstupní obsah pásky (nekonečná posloupnost prázdných symbolů na konci pásky není uvedena).

Jednotlivé konfigurace stroje vypisujte na samostatné řádky. Konfiguraci uvádějte v pořadí <obsah pásky před hlavou><stav><symbol pod hlavou><zbytek pásky> (bez oddělovačů).

Zaměřte se na korektní vstupy a situace, kdy existuje sekvence přechodů do koncového stavu. V takovém případě Váš program musí toto řešení najít.

V případě **abnormálního zastavení** program skončí s chybovým kódem (K tomuto účelu můžete využít predikát halt(C), kde C > 0.), přičemž před skončením může vypsat chybovou hlášku (nejvýše 1 řádek textu, ukončený pomocí newline). Při **zacyklení** Turingova stroje je (výjimečně) v pořádku, pokud se Váš program také zacyklí. Ještě lepší však bude, pokud Váš program dokáže zacyklení detekovat. Při detekovaném zackylení postupujte stejně jako u abnormálního zastavení – tedy ukončení s chybovým kódem a případně jednořádkové hlášení.

Příklad vstupu a výstupu:

```
S a B a
B a B b
B b B R
B c B a
B c F c
B c B a
aaacaa
```

```
Saaacaa
Baaacaa
Bbaacaa
bBaacaa
bBbacaa
bbBacaa
bbBbcaa
bbbBcaa
bbbFcaa
```

Rubikova kostka

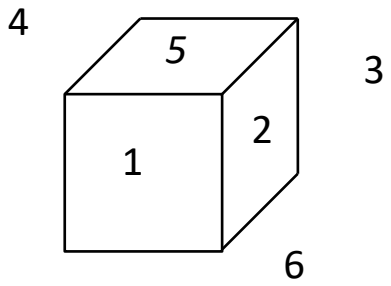
Rubikova kostka je hlavolam ve tvaru krychle, každá strana je vybarvena odlišnou barvou a je rozdělena na 3x3 dílů, jednotlivé vrstvy lze vzájemně pootočit o násobek 90 °. Cílem je kostku, která byla zamíchána náhodným otáčením jednotlivých vrstev, uvést do složeného stavu (každá strana jednou barvou).

Vstupem vašeho programu bude popis zamíchané kostky, výstupem návod, jak ji složit. Návod bude obsahovat popis kostky po každém kroku řešení, jednotlivé popisy oddělené prázdným řádkem. Nejprve uveďte popis zadané zamíchané kostky, pak postupně jednotlivé kroky řešení. V posledním kroku bude složená kostka.

Jednotlivé barvy v každém dílku jsou popsány číslicemi 1—6, každá strana kostky je tedy popsána devíti číslicemi. Číslice z každé strany jsou zapisovány po řádcích a jednotlivé strany jsou uvedeny v pořadí podle následujícího schématu. Číslice ve skupinkách nejsou nijak odděleny, nejprve je uvedena horní strana kostky, pak přední, pravá, zadní a levá odděleny mezerou a pak spodní strana (viz následující ukázka – nákres kostky, její plášť a zápis v požadovaném formátu). Kostka složená v posledním kroku musí být orientována přesně tak, jak je uvedeno v zadání (tedy horní hrana obsahuje barvu 5 apod.).

Nalezený postup pro složení kostky nemusí být optimální (samozřejmě, pokud bude, je to skvělé – a nezapomeňte to uvést v dokumentaci), ale měl by být alespoň trochu rozumný – tj. neměl obsahovat extrémně velké množství zbytečných tahů. Např. stačí-li ke složení kostky jediný tah, není v pořádku, pokud program vypíše tahů 20.

Ukázka složené kostky:



5	5	5									
5	5	5									
5	5	5									
1	1	1	2	2	2	3	3	3	4	4	4
1	1	1	2	2	2	3	3	3	4	4	4
1	1	1	2	2	2	3	3	3	4	4	4
6	6	6									
6	6	6									
6	6	6									

```
555
555
555
111 222 333 444
111 222 333 444
111 222 333 444
666
666
666
```

Příklad vstupu:

```
553
553
554
225 322 644 111
115 322 633 444
115 322 633 444
662
661
661
```

Příklad výstupu:

```
553
553
554
225 322 644 111
115 322 633 444
115 322 633 444
662
661
661

555
555
555
222 333 444 111
111 222 333 444
111 222 333 444
666
666
666

555
555
555
111 222 333 444
111 222 333 444
111 222 333 444
666
666
666
```

Uvažujte následující standardních 18 pohybů:

- **U** – rotace horní stěny kostky o 90° ve směru hodinových ručiček (z pohledu shora)
- **U‘** – rotace horní stěny kostky o 90° proti směru hodinových ručiček (z pohledu shora)
- **D** – rotace spodní stěny kostky o 90° ve směru hodinových ručiček (z pohledu zespod)
- **D‘** – rotace spodní stěny kostky o 90° proti směru hodinových ručiček (z pohledu zespod)
- **R** – rotace pravé stěny kostky o 90° ve směru hodinových ručiček (z pohledu zprava)
- **R‘** – rotace pravé stěny kostky o 90° proti směru hodinových ručiček (z pohledu zprava)
- **L** – rotace levé stěny kostky o 90° ve směru hodinových ručiček (z pohledu zleva)
- **L‘** – rotace levé stěny kostky o 90° proti směru hodinových ručiček (z pohledu zleva)
- **F** – rotace přední stěny kostky o 90° ve směru hodinových ručiček (z pohledu zepředu)
- **F‘** – rotace přední stěny kostky o 90° proti směru hodinových ručiček (z pohledu zepředu)
- **B** – rotace zadní stěny kostky o 90° ve směru hodinových ručiček (z pohledu zezadu)
- **B‘** – rotace zadní stěny kostky o 90° proti směru hodinových ručiček (z pohledu zezadu)
- **M** – rotace vertikálního vnitřního řezu kostky, paralelního k levé a pravé stěně kostky (spojuje přední + zadní a horní + spodní stěnu) o 90° ve směru hodinových ručiček (z pohledu zleva)
- **M‘** – rotace vertikálního vnitřního řezu kostky, paralelního k levé a pravé stěně kostky (spojuje přední + zadní a horní + spodní stěnu) 90° proti směru hodinových ručiček (z pohledu zleva)
- **E** – rotace horizontálního vnitřního řezu kostky o 90° proti směru hodinových ručiček (z pohledu zespod). Pozn. při pohledu zepředu se kostičky pohybují zleva doprava.
- **E‘** – rotace horizontálního vnitřního řezu kostky o 90° proti směru hodinových ručiček (z pohledu zespod). Pozn. Při pohledu zepředu se kostičky pohybují zprava doleva.
- **S** – rotace vertikálního vnitřního řezu kostky, paralelního k přední a zadní stěně (spojuje levou + pravou a horní + spodní stěnu) o 90° ve směru hodinových ručiček (z pohledu zepředu)
- **S‘** – rotace vertikálního vnitřního řezu kostky, paralelního k přední a zadní stěně (spojuje levou + pravou a horní + spodní stěnu) o 90° proti směru hodinových ručiček (z pohledu zepředu)

Pohyby jsou ilustrovány na následující stránce.

Formou rozšíření můžete implementovat také následující pohyby:

- Ekvivalenty předchozích pohybů o 180°
- Rotace celé kostky o po ose x/y/z o 90° / 180° ve směru/proti směru hodinových ručiček.

Rozšíření však musí být nutné aktivovat ručně (např. spouštěcím parametrem). Při standardním spuštění bude program využívat **pouze standardních 18 pohybů**.

U - Turn the top (up) face clockwise.

U' - Turn the top (up) face counter-clockwise.

D - Turn the bottom (down) face clockwise.

D' - Turn the bottom (down) face counter-clockwise.

R - Turn the right face clockwise.

R' - Turn the right face counter-clockwise.

L - Turn the left face clockwise.

L' - Turn the left face counter-clockwise.

F - Turn the front face clockwise.

F' - Turn the front face counter-clockwise.

B - Turn the back face clockwise.

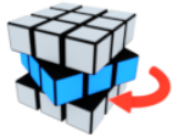
B' - Turn the back face counter-clockwise.

M - Turn the vertical slice clockwise.

M' - Turn the vertical slice counter-clockwise.

E - Turn the horizontal slice clockwise.

E' - Turn the horizontal slice counter-clockwise.



S - Turn the top slice clockwise.

S' - Turn the top slice counter-clockwise.

Babylonská věž

Babylonská věž je hlavolam ve tvaru válce složeného z otočných prstenců, kolmo na prstence jsou umístěny výřezy, ve kterých jsou umístěny různobarevné kuličky. U sestavené věže jsou v každém sloupci (výřezu) kuličky stejné barvy, seřazeny od nejsvětlejší po nejtmavší. Otáčením prstenců je možné přesouvat kuličky mezi sloupci (horizontálně). V jednom výřezu je o kuličku méně, čímž vniká volný prostor, pomocí něhož je možné přesouvat kuličky také mezi prstenci (vertikálně).

Každá kulička je popsána barvou – označena popořadě písmeny od A (maximálně A—Z, bude se tedy vyskytovat nejvýše 26 barev) a úrovní odstínu (číslovány od 1 do nejvýše 9), přičemž 1 znamená nejsvětlejší odstín. Věž na fotografii je pouze jedním příkladem. Obecně může mít věž na vstupu programu libovolné rozměry, nejvýše však 26 sloupců x 9 řádků. Aktuální rozměry věže musíte odvodit z jejího popisu. Volný prostor je označen dvěma hvězdičkami **. U složené věže se tento prostor nachází za poslední kuličkou sloupce A. Každý prstenec je vypsán na samostatný řádek, kuličky jsou odděleny mezerami.

Příklad věže se třemi barvami (A, B, C) a čtyřmi prstenci. Chybí kulička A4. Kuličky A2, A3 jsou posunuty o pozici níže:

```
A1 B1 C1
** B2 C2
A2 B3 C3
A3 B4 C4
```

Vstupem vašeho programu bude popis zamíchané věže, výstupem návod, jak ji složit. Návod bude obsahovat popis věže po každém kroku řešení, jednotlivé popisy oddělené prázdným řádkem. Nejprve uveďte popis zadané zamíchané věže, pak postupně jednotlivé kroky řešení. V posledním výpisu bude složená věž. V jednom tahu je možné rotovat i více prstenců současně, pokud se všechny rotují o stejný počet pozic. Rotaci různých prstenců o různý počet pozic reprezentuje samostatnými tahy.

U složené věže půjdou ve sloupcích barvy popořadě (A, B, C, …). Volný prostor bude na poslední pozici sloupce A. Příklad složené věže:

```
A1 B1 C1 D1 E1 F1
A2 B2 C2 D2 E2 F2
A3 B3 C3 D3 E3 F3
A4 B4 C4 D4 E4 F4
** B5 C5 D5 E5 F5
```

Nalezený postup pro složení věže nemusí být optimální (samozřejmě, pokud bude, je to skvělé – a nezapomeňte to uvést v dokumentaci), ale také by neměl být mnohonásobně složitější. Jinými slovy, není žádoucí, aby řešení obsahovalo extrémně velké množství tahů, které nejsou nutné. Tedy např. stačí-li ke složení věže jediný tah, není v pořádku, když program vypíše tahů 20.

Ukázka vstupu:

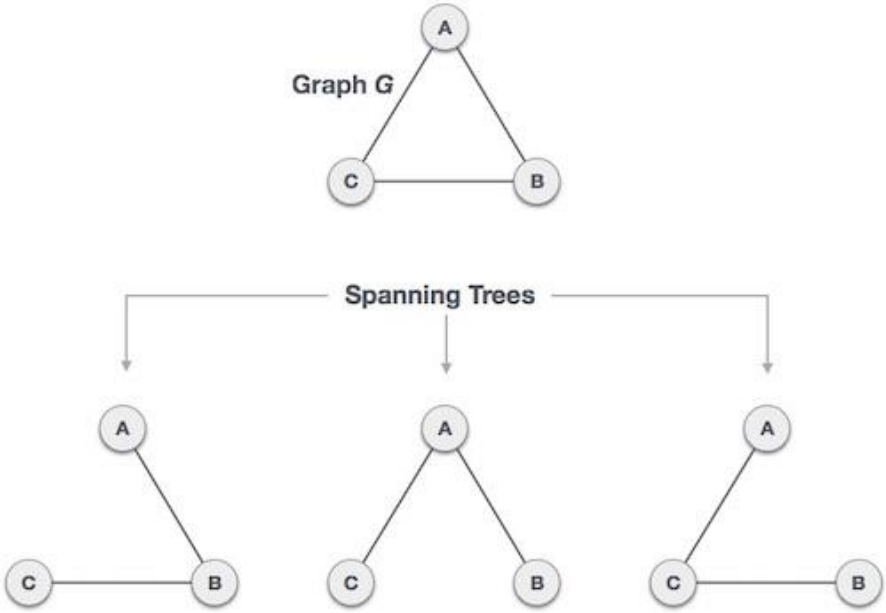
```
A1 B1 C1 D1 E1
A2 B2 C2 D2 E2
B3 C3 D3 E3 E4
A3 B4 C4 D4 **
```

Příklad požadovaného výstupu:

```
A1 B1 C1 D1 E1
A2 B2 C2 D2 E2
B3 C3 D3 E3 E4
A3 B4 C4 D4 **

A1 B1 C1 D1 E1
A2 B2 C2 D2 E2
B3 C3 D3 E3 **
A3 B4 C4 D4 E4

A1 B1 C1 D1 E1
A2 B2 C2 D2 E2
** B3 C3 D3 E3
A3 B4 C4 D4 E4

A1 B1 C1 D1 E1
A2 B2 C2 D2 E2
A3 B3 C3 D3 E3
** B4 C4 D4 E4
```

Kostra grafu

Vašim úkolem je napsat program schopný nalézt všechny kostry zadaného neorientovaného grafu. Kostra grafu (angl. spanning tree) je takový podgraf, který je tvořen všemi vrcholy původního grafu a zároveň je to strom (je souvislý a neobsahuje žádnou kružnici).



Na vstupu budete mít zadány neorientované hrany grafu ve formě dvojic oddělených mezerou. Tato písmena značí vrcholy grafu. Pokud by vstupní graf nebyl souvislý, tak se na výstup nevytiskne nic.

Vrcholy jsou vždy označeny pomocí velkých písmen anglické abecedy (A až Z). Jednotlivé řádky vstupního souboru mají tvar „<V1> <V2>“ (bez uvozovek). Oddělovače mezi vrcholy jsou mezery. Jakýkoliv jiný řádek je ignorován. Jako výstup programu se očekává seznam koster, kde na jednom řádku je jedna kostra. Každá kostra má výstup ve tvaru „<V1>-<V2> <V3>-<V4> … <Vn-1>-<Vn>”. Každá dvojice vrcholů reprezentuje hranu dané kostry. Pořadí vrcholů v hraně, jednotlivých hran, nebo pořadí koster ve výstupu je libovolné, každá kostra však musí být unikátní.

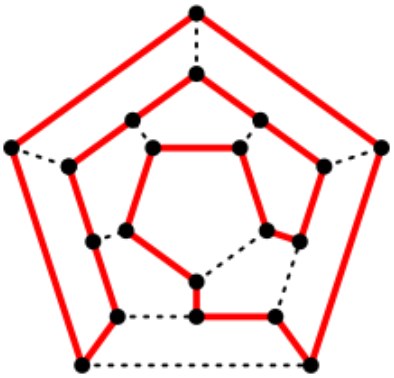
Ukázka vstupu a výstupu:

```
A B
A C
A D
B C
C D
```

```
A-B A-C A-D
A-B A-C C-D
A-B A-D B-C
A-B A-D C-D
A-B B-C C-D
A-C A-D B-C
A-C B-C C-D
A-D B-C C-D
```

Hamiltonovská kružnice

Vašim úkolem je napsat program schopný nalézt všechny Hamiltonovské kružnice zadaného neorientovaného grafu. Hamiltonovská kružnice prochází každým vrcholem grafu právě jednou přičemž začíná a končí v tom samém vrcholu.



Na vstupu budete mít zadány neorientované hrany grafu ve formě dvojic oddělených mezerou. Tato písmena značí vrcholy grafu. Pokud by vstupní graf nebyl hamiltonovský, tak se na výstup nevytiskne nic.

Vrcholy jsou vždy označeny pomocí velkých písmen anglické abecedy (A až Z). Jednotlivé řádky vstupního souboru mají tvar „<V1> <V2>“ (bez uvozovek). Oddělovače mezi vrcholy jsou mezery. Jakýkoliv jiný řádek je ignorován. Jako výstup programu se očekává seznam Hamiltonovských kružnic, kde na jednom řádku je jedna kružnice. Každá kružnice má výstup ve tvaru „<V1>-<V2> <V3>-<V4> … <Vn-1>-<Vn>”. Každá dvojice vrcholů reprezentuje hranu dané kružnice. Pořadí vrcholů v hraně, jednotlivých hran, nebo pořadí kružnic ve výstupu je libovolné, každá kružnice však musí být unikátní.

Ukázka vstupu a výstupu:

```
A B
A C
A D
B C
B D
C D
```

```
A-B A-C B-D C-D
A-B A-D B-C C-D
A-C A-D B-C B-D
```

Vlastní zadání

Vymyslete si a implementujte vlastní zadání. Může jít o cokoliv: jiný hlavolam/hra, řešení jiného problému na prohledávání stavového prostoru, simulace, řešení rozhodovacích problémů, řešení problému z reálného světa, z oblasti matematiky/IT apod. Fantazii se meze nekladou.

Vaše zadání musí být předem schváleno opravujícím. Rozhodnutí o schválení či zamítnutí zadání záleží pouze na opravujícím. Zadání by mělo být odlišné od zadání řešených v minulých letech i od úkolů logického programování řešených v jiných předmětech. Zadání musí být alespoň tak obtížné, jako jsou ta zde uvedená.