1) Připomenutí asymptotické složitosti. Grafy, multigrafy, jejich vlastnosti a reprezentace v počítači. Prohledávání grafu, prioritní fronta.

1. Úvod
    a) představení, kancelář, konzultace
    b) lichý - sudý týden, obsah seminářů a počítačových cvičení
    c) stránky předmětu: ???, vytvoření studentských účtů s heslem pro odesílání zápočtových úloh ???
    d) možná interakce s předměty z dřívějška
    2. semestr magisterského studia, dříve:
    Algoritmizace (1.ročník, bakalář), mj. přednáška na složitost
    Diskrétní matematika a logika (5.semestr, bakalář), mj. teorie grafů
    PUI (4.semestr, bakalář),
    Složitost algoritmů,
    Programovací techniky (5.semestr, bakalář), základy teorie složitosti.
    Další interakce: KUI, Matematika 1, 2, 3 (limity, derivace, integrály, řady, dif. rovnice, základy psti), Algebra
2. Složitost
    a) Zopakování pojmu složitosti algoritmu. Zopakování definic asymptotických složitostí (horní O, dolní Ω, optimální Θ).
    b) Asymptotická složitost - proč lze ignorovat roli konstant - porovnání funkcí (lze přidat i N^2/2 vs. 100lgN (N=32) nebo N^3/4 vs. 1000lgN (N=27))
    c) k čemu vede zrychlení výpočtu 100x a 1000x?
3. Grafy
    a) Připomeňte pojem grafu jako uspořádaná dvojice vrcholů a hran (G=(V,E)).
    b) Připomeňte následující základní grafové pojmy: orientovanost, váženost, acyklicita, stromy, multigrafy.
    c) Připomeňte možné reprezentace grafu v počítači (matice sousednosti, matice incidence, seznam hran, jako pointrová datová struktura). Na těchto datových strukturách ukažte možnou realizaci základních grafových pojmů.

2) Prohledávání grafu, prioritní fronta.

1. Úloha: **jednoduchý make**
   Naprogramujte make <cíl>, který provede <cíl> podle níže popsaných pravidel, které se budou nacházet v souboru ./makefile.
   Pravidla mají následující tvar:

```
target ... : prerequisites ...
             command
             ...
             ...
```

Kde *target* je jméno cíle (pokud jich je víc jsou odděleny mezerami), k jehož provedení je třeba nejprve provést všechny *prerequisites* cíle (jsou oddělené mezerami a pořadí ve kterém se provedou je libovolné) a potom provést posloupnost zadaných příkazů *command*. Před každým příkazem (*command*) je tabulátor.

Pokud nebude možné cíl provést, vypíše program chybovou zprávu. Pro navržený algoritmus řešící make určete jeho asymptotickou časovou a paměťovou složitost.

Příklad makefile:

```
all: hello

hello: main.o hello.o

	g++ main.o hello.o -o hello

main.o: main.cpp functions.h

	g++ -c main.cpp

hello.o: hello.cpp functions.h

	g++ -c hello.cpp

hello.cpp:

	echo '#include <iostream.h>' > hello.cpp

	echo '#include "functions.h"' >> hello.cpp

	echo 'void print_hello(void){ cout << "Hello World!"; }' >> hello.cpp

main.cpp:

	echo '#include <iostream.h>' > main.cpp

	echo '#include "functions.h"' >> main.cpp

	echo 'int main() { print_hello();' >> main.cpp

	echo 'cout << endl; return 0; } ' >> main.cpp

functions.h:

	echo 'void print_hello(void);' > functions.h

clean:

	rm -rf *.o *.cpp *.h hello
```

Řešení:

Řešení je postaveno na procházení orientovaného grafu (srovnejte možnost prohledáváním do šířky a do hloubky a vysvětlete pojem prioritní fronty).

2. Úloha: **převod hradlové sítě do jednoduchého assembleru**
   Je daná hradlová síť jako DAG (Directed Acyclic Graph), kde uzly jsou hradla s jedním (not) nebo dvěma vstupy (and, or) a listy (list zde chápeme jako uzel, do nějž nevede žádná orientovaná hrana) jsou názvy vstupních registrů typu bool. Celá síť má pouze jeden kořen (kořen je zde uzel, ze kterého nevede žádná orientovaná hrana).

Dále je dán jednoduchý assembler s následujícími instrukcemi, kde v závorkách je uvedena sémantika každé instrukce:
NOT Reg1, Reg2 (Reg1 := not Reg2)
AND Reg1,Reg2,Reg3 (Reg1 := Reg2 and Reg3)
OR Reg1,Reg2,Reg3 (Reg1 := Reg2 or Reg3)
A boolovské registry R1, R2, … a pomocné boolovské registry T1,T2 … (pomocné registry se v zadané hradlové síti nevyskytují).
Úkolem je vygenerovat assemblerovský program, který spočítá zadanou hradlovou síť a výsledek uloží do registru T1.
Příklad:
???
Řešení:
Řešení je postaveno na nalezení topologického uspořádání v hradlové síti.
Poznámka:
Úlohu lze modifikovat zrušením pomocných registrů a zavedením zásobníkových instrukcí PUSH Reg1 a POP Reg1.

3) Minimální kostry, algoritmus Borůvkův a Jarníkův. Problém Union-find.

1. Úloha: **počítačová síť** (zdroj: 53. ročník MO kategorie P-II-1)

   Firma Truhlík a syn má ve městě N budov a chce všechny svoje budovy propojit počítačovou sítí. Vedení firmy rozhodlo, že pro K (1≤K≤N) budov zakoupí vysokorychlostní připojení na Internet. Kromě toho mezi některými dvojicemi budov vybudují propojení optickým kabelem.

   Dvě budovy se nacházejí v téže komponentě sítě, pokud lze mezi nimi komunikovat pomocí optických kabelů (buď mají přímé spojení, nebo jsou spojeny nepřímo přes několik jiných budov). Aby bylo možné komunikovat mezi dvěma budovami ležícími v různých komponentách sítě, musí každá z těchto komponent obsahovat aspoň jeden počítač připojený na Internet.

   Úloha: Na vstupu jsou dána čísla N a K a pro každou dvojici budov jedno kladné celé číslo -- cena za vybudování optického kabelu, který by propojil tuto dvojici budov. Navrhněte efektivní algoritmus, jenž určí, kterých K budov se má připojit na Internet a které dvojice budov se mají propojit optickým kabelem tak, aby mezi každými dvěma budovami bylo možné komunikovat a přitom aby celková cena vybudovaných optických kabelů byla co nejmenší.

   Příklad:

   Vstup:

   ```
   N=5, K=2
   Ceny spojení:
   (1,2): 100
   (1,3): 10
   ```

```
(1,4): 100
(1,5): 300
(2,3): 100
(2,4): 10
(2,5): 300
(3,4): 47
(3,5): 27
(4,5): 74
```

Výstup:

```
Na Internet připojíme budovy 1 a 2,
kabelem spojíme dvojice budov (1,3), (2,4) a (3,5).
Cena kabelů bude 47.
```

Řešení:

Zadanou úlohu si převedeme do řeči teorie grafů. Budovy firmy představují *vrcholy* našeho grafu, *hrany* grafu odpovídají možným propojením optickým kabelem. Úlohu vyřešíme nejprve pro případ K=1. V tomto případě je naším úkolem vybrat takovou množinu hran, aby všechny vrcholy byly navzájem propojeny (ne nutně přímo). Taková množina hran se nazývá *kostra grafu* a jelikož chceme, aby součet cen hran v kostře byl co nejmenší, řešíme problém hledání *minimální kostry.*

Rozmyslete si, že minimální kostra grafu neobsahuje žádný cyklus -- kdyby totiž nějaký obsahovala, mohli bychom jeho libovolnou hranu odstranit. Na druhé straně když do minimální kostry přidáme libovolnou hranu, vznikne nám cyklus, neboť vrcholy, mezi nimiž vede přidaná hrana, byly už spojeny pomocí nějakých hran kostry.

Hledání minimální kostry (Primův algoritmus)

Algoritmus je založen na následující myšlence. Vrcholy grafu rozdělíme na dvě skupiny: na připojené a nepřipojené. Na začátku algoritmu zvolíme libovolný vrchol a prohlásíme ho za připojený, ostatní vrcholy jsou zatím nepřipojené. V každém kroku algoritmu připojíme jeden vrchol k dosud vytvořené síti následujícím způsobem. Najdeme nejkratší hranu spojující připojený a nepřipojený vrchol. Tuto hranu přidáme do sítě a její druhý konec se stane připojeným vrcholem. Skončíme ve chvíli, když jsou všechny vrcholy připojeny.

Aby byl algoritmus efektivní, potřebujeme umět rychle nalézt nejkratší hranu spojující připojený a nepřipojený vrchol. To zařídíme tak, že pro každý dosud nepřipojený vrchol si budeme pamatovat, ze kterého připojeného vrcholu k němu vede nejkratší hrana. Pokaždé, když přidáme k připojeným vrcholům další vrchol, musíme si informaci o nejbližších připojených vrcholech aktualizovat. Projdeme všechny nepřipojené vrcholy a pokud je nově připojovaný vrchol bližší, naši informaci změníme.

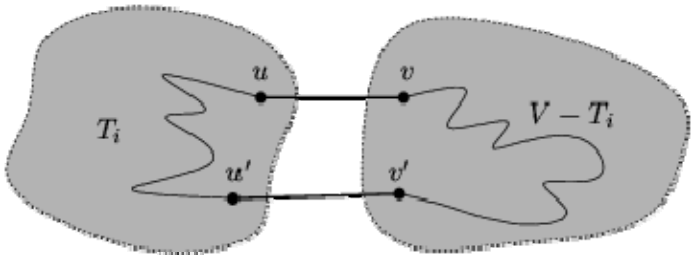
Skutečnost, že výsledná množina hran tvoří kostru, je zřejmá. Je však třeba dokázat, že je tato kostra minimální. Představme si libovolnou minimální kostru (dále ji budeme označovat MK) a porovnávejme ji s výsledkem našeho algoritmu (dále VNA).

Jestliže MK a VNA jsou shodné, VNA je minimální kostra. Předpokládejme tedy, že MK a VNA nejsou shodné. Nechť T_i je množina připojených vrcholů po i-tém kroku našeho algoritmu. Seřadíme hrany ve VNA podle toho, jak jsme je přidávali, a najdeme první

hranu, která se vyskytuje ve VNA, ale není obsažena v MK. Nechť tato hrana byla přidána v kroku i+1 a nechť spojuje vrchol $u \in T_i$ a vrchol $v \notin T_i$.

Přidejme hranu (u,v) do MK. Tím vznikne v MK cyklus, který začíná v T_i, přejde po hraně (u,v) ven z T_i a potom se vrátí nějakou cestou zpět do T_i (viz obr. 1). Na této cestě musí existovat aspoň jedna hrana (u',v'), která má jeden konec v T_i a druhý konec mimo T_i. Cena této hrany musí být aspoň taková, jako je cena hrany (u,v). V opačném případě by si náš algoritmus v kroku i+1 musel vybrat hranu (u',v') namísto hrany (u,v). Proto pokud hranu (u',v') odebereme z MK a přidáme tam místo ní hranu (u,v), cena MK se nezvýší. Nemůže se však ani snížit, neboť MK je minimální. Upravená MK bude tedy nadále minimální kostrou v grafu. Navíc VNA a MK se nyní shodují v prvních i+1 hranách. Stejným způsobem postupně přeměníme MK na VNA, přičemž nezvýšíme její cenu, takže VNA musí být také minimální kostrou.

Přidáním hrany (u,v) vznikne v MK cyklus, který začíná v T_i, přejde po hraně (u,v) ven z T_i a potom se vrátí nějakou cestou zpět do T_i.



Řešení pro obecné K

Dosud jsme předpokládali K=1. Jestliže K>1, nemusíme hranami pospojovat všechny vrcholy. Ke komunikaci totiž můžeme využít také Internet. Stačí, když se naše síť bude skládat z K souvislých částí, v každé z těchto souvislých částí vybereme jeden vrchol, který připojíme na Internet, a tak bude moci komunikovat každý vrchol s každým.

Takovouto síť můžeme získat například odebráním K-1 nejdražších hran z minimální kostry MK. Tím se nám totiž MK rozpadne právě na K souvislých částí. Jediným problémem je ukázat, že toto řešení je skutečně nejlevnější možné.

Označme tedy symbolem P množinu K-1 nejdražších hran kostry MK. Jejich odebráním z MK dostaneme množinu hran Q, která se skládá z K souvislých částí. Nechť existuje levnější množina hran T, která rovněž tvoří síť složenou z K souvislých částí.

Budeme uvažovat graf tvořený kostrou MK a hranami z množiny T. Jelikož už MK je souvislá, tento graf je jistě souvislý. Proto lze zvolit několik hran z MK, jimiž se dají jednotlivé komponenty T pospojovat. Každá přidaná hrana spojí dvě komponenty do jedné větší, takže stačí přidat K-1 hran. Množinu těchto přidaných hran označíme symbolem S.

Všimněte si následujících dvou skutečností:

- Množina hran S určitě není dražší než P, neboť obě obsahují K-1 hran z kostry MK, ale P jsme vybrali tak, aby obsahovala nejdražší hrany.

- Podle našeho předpokladu množina hran T je levnější než množina hran Q.

Z toho ale vyplývá, že kostra S+T je levnější než MK=P+Q, což je spor s tím, že MK je minimální kostra. Tím jsme ukázali, že k vyřešení úlohy stačí z MK odebrat K-1 nejdražších hran.

Časová složitost

Při hledání minimální kostry se v každém kroku přidá jeden vrchol do množiny připojených, vykoná se tedy celkem N-1 kroků. V každém kroku nejprve v čase θ(N) najdeme nejkratší hranu spojující připojený a nepřipojený vrchol. Potom aktualizujeme informaci o nejbližším připojeném vrcholu pro všechny dosud nepřipojené vrcholy. Tato aktualizace představuje opět provedení θ(N) operací. Celková časová složitost je proto kvadratická, tj. $\theta(N^2)$.

Z výsledné minimální kostry potom potřebujeme odebrat K-1 nejdražších hran. To můžeme udělat tak, že hrany kostry setřídíme. Třídění lze provést v čase θ(N Log(N)), v tomto případě nám ovšem stačí použít jednoduché třídění pracující v čase $\theta(N^2)$. Celková časová složitost je $\theta(N^2)$.

```
program Sit_P_II_1;

const
   maxn = 1000;
   nekonecno = 10000;

var
   N,K: integer; { počet budov, počet komponent }
   a: array[1..maxn,1..maxn] of integer; { ceny spojení }
   sit: array[1..maxn,1..2] of integer; { seznam hran výsledné sítě }

procedure nacti_vstup;
var
   i,j: integer;
begin
   write('Počet budov N:'); readln(N);
   write('Počet internetových připojení K:'); readln(K);
   for i:=1 to N do
      for j:=i+1 to N do begin
         write('Cena (',i,',',j,'):'); readln(a[i,j]);
         a[j,i]:=a[i,j];
      end;
end; {nacti_vstup}

procedure minimalni_kostra;
{najde minimální kostru a uloží její hrany do pole sit}
var
   pripojene: array[1..maxn] of boolean;
   nej: array[1..maxn] of integer;
   i,j: integer;
   min, nejlepsi: integer;

begin
   {na začátku je jenom vrchol 1 připojený}
   pripojene[1]:=true;
   for i:=2 to N do begin
      {v poli nej si budeme pro každý dosud nepřipojený vrchol udržovat nejbližší
připojený vrchol}
      pripojene[i]:=false;
      nej[i]:=1;
   end;

   for i:=1 to N-1 do begin
      {najdeme nejkratší hranu, která spojuje připojený a nepřipojený vrchol}
      min:=nekonecno;
      for j:=1 to N do begin
         if not pripojene[j] then begin
```

```
            if a[j,nej[j]]<min then begin
               nejlepsi:=j; min:=a[j,nej[j]]
            end;
         end;
      end;

      {nalezený vrchol připojíme}
      pripojene[nejlepsi]:=true;
      {spojení (nejlepsi,nej[nejlepsi]) patří do sítě}
      sit[i,1]:=nejlepsi; sit[i,2]:=nej[nejlepsi];

      {přepočítáme pole nej: pro každý vrchol zjistíme, zda právě
      připojený vrchol nezkrátí jeho vzdálenost k připojeným vrcholům}
      for j:=1 to N do begin
         if not pripojene[j] then begin
            if a[j,nej[j]]>a[j,nejlepsi] then nej[j]:=nejlepsi;
         end;
      end;
   end;
end; {minimalni_kostra}

procedure utrid_hrany_site;
{setřídí hrany sítě od nejlevnější po nejdražší}
var
   i,j,k,min: integer;
begin
   for i:=1 to N-1 do begin
      min:=i;
      for j:=i+1 to N-1 do begin
         if a[sit[j,1],sit[j,2]]<a[sit[min,1],sit[min,2]] then
            min:=j;
      end;
      k:=sit[min,1]; sit[min,1]:=sit[i,1]; sit[i,1]:=k;
      k:=sit[min,2]; sit[min,2]:=sit[i,2]; sit[i,2]:=k;
   end;
end; {utrid_hrany_site}

procedure vypis_vysledek;
{vypíše výsledné hrany sítě}
var
   i: integer;
begin
   writeln('Je třeba vybudovat spojení mezi následujícími budovami:');
   for i:=1 to N-K do begin
      writeln('(',sit[i,1],',',sit[i,2],')');
   end;
end; {vypis_vysledek}

begin
   nacti_vstup;
   minimalni_kostra;
   utrid_hrany_site;
   vypis_vysledek;
end.
```

2. Úloha: **nepostradatelná silnice** (zdroj: 43. ročník MO kategorie P-III-2)

   V zemi je N měst označených čísly od 1 do N. Mezi městy je vybudována silniční síť. Každá silnice spojuje vždy dvojici měst. Všechny silnice jsou obousměrné. Mezi některými dvojicemi měst přímá silnice nevede, ale z každého města je možné dojet po silnicích do libovolného jiného města (třeba i více různými způsoby). Všechna případná křížení silnic mimo města jsou mimoúrovňová (pomocí mostů) a neumožňují vozidlům přejet z jedné silnice na druhou.

   Silnici nazveme nepostradatelnou, pokud by se jejím zničením úplně přerušilo silniční spojení mezi některou dvojicí měst.

Napište program, který vyhledá a vypíše všechny nepostradatelné silnice. Vstupem programu je počet měst N a dále seznam všech silnic vedoucích mezi městy. Každá silnice je zadána dvojicí čísel měst, mezi nimiž vede.

Řešení:

Úloha P-III-2 je jednou z klasických úloh teorie grafů. Silniční síť představuje souvislý neorientovaný graf, v němž vrcholy grafu odpovídají městům a hrany grafu silnicím. Nepostradatelné silnice, tak jak jsou definovány v zadání úlohy, odpovídají v teorii grafů zvláštním hranám zvaným mosty. Úkolem tedy je nalézt v daném grafu všechny mosty.

Algoritmus řešení je založen na procházení zadaným grafem do hloubky. Při procházení bude každý vrchol grafu navštíven právě jednou. Způsob procházení lze znázornit stromem. Kořenem stromu procházení je vrchol, z něhož bylo procházení zahájeno. Za kořen můžeme zvolit libovolný vrchol grafu. Bezprostředními následníky některého vrcholu V jsou všechny ty vrcholy, do nichž prohledávání z vrcholu V bezprostředně pokračovalo. Protože zadaný graf je souvislý, budou ve stromu procházení obsaženy všechny vrcholy původního grafu (města). Ze všech hran (silnic) budou ve stromě procházení obsaženy jen ty, které nás v průběhu procházení dovedly do nového, doposud nenavštíveného vrcholu.

Představme si, že do stromu procházení dokreslíme zelenou barvou všechny zbývající hrany grafu. To jsou tedy takové, kterými průchod do hloubky nepokračoval. Jinak řečeno, v průběhu procházení tyto silnice vedly z právě procházeného města do jiného, již dříve navštíveného města. Doplněný strom tedy bude izomorfní s původním grafem.

Nyní vyslovíme jedno pomocné tvrzení: Oba koncové vrcholy každé zelené hrany leží na téže větvi stromu procházení. Tvrzení snadno dokážeme sporem. Předpokládejme, že by některá zelená silnice spojovala dvě města A a B, která neleží na jedné větvi stromu procházení. Označme je tak, že během průchodu bylo A poprvé navštíveno dříve než B. Město B jistě neleží v podstromu procházení s kořenem A (jinak by A a B ležela na téže větvi). Pro postup procházení to znamená, že nejprve bylo (případně několikrát) navštíveno město A a teprve potom (případně několikrát) navštíveno město B. Všimněme si okamžiku během procházení, kdy jsme město A navštívili naposledy. To bylo v situaci, kdy jsme se z něho vraceli zpět (kdybychom šli vpřed, museli bychom do A přijít ještě na zpáteční cestě). V tomto okamžiku jsme se ale nezachovali správně podle algoritmu procházení grafem do hloubky: vraceli jsme se zpět, a přitom jsme ještě měli projít silnicí AB, protože ta vedla do tehdy ještě nenavštíveného města B.

K tomu, aby hrana byla mostem, je nutné a stačí, aby se jejím odstraněním oddělil podstrom ve stromu procházení, který nebude dostupný ani po některé zelené hraně. Podle předchozího tvrzení by takové spojení zelenou hranou muselo vést do vyšší vrstvy ve stromě procházení.

Na základě provedených úvah již lze zformulovat algoritmus. Zadaný graf budeme procházet do hloubky, začít můžeme libovolným vrcholem (např. vrcholem číslo 1). Během procházení si budeme u každého vrcholu M pamatovat jeho hloubku ve stromě

procházení H_M. Kořen stromu procházení bude mít hloubku 0. Postupně během průchodu budeme pro každý procházený vrchol M určovat číslo Z_M definované takto: Z_M je minimum z H_M a z hloubek koncových měst všech zelených silnic, které vycházejí z vrcholů v podstromu s kořenem M. Z_M je tedy číslo nejvyšší hladiny ve stromě procházení, do které vede přímé spojení zelenou silnicí z nějakého města v podstromu s kořenem M. Přitom si všímáme jen hladin nad vrcholem M. Nastane-li pro některý vrchol M nerovnost $Z_M < H_M$, existuje zelená silnice, která spojuje podstrom s kořenem ve vrcholu M se zbytkem grafu. Je-li $Z_M = H_M$, je silnice, po níž jsme do M během procházení přišli, mostem.

Zbývá ukázat, jak budeme počítat hodnoty H_M a Z_M pro vrchol M. Hodnotu H_M určíme snadno při prvním vstupu do vrcholu M během procházení grafem - je o 1 větší než odpovídající hodnota H_X vrcholu X, z něhož do M přicházíme. Stanovení hodnoty Z_M je o něco obtížnější. Hodnota Z_M je rovna minimu z hodnoty H_M a z hodnot Z_i všech vrcholů ležících v podstromu s kořenem ve vrcholu M. Při prvním vstupu do vrcholu M můžeme tudíž inicializovat hodnotu Z_M již známou hodnotou H_M a pak ji během procházení podstromu vrcholu M budeme případně zmenšovat, bude-li to možné. Při každém dalším příchodu do vrcholu M (tj. při návratu z nějakého následníka vrcholu M) lze hodnotu Z_M snížit na Z-hodnotu tohoto následníka. K dalšímu snížení Z_M mohou přispět hrany, které vedou z vrcholu M do již navštívených uzlů a po nichž se tudíž při průchodu nepostupuje. Hodnotu Z_M můžeme snížit na H-hodnotu koncových vrcholů těchto zelených hran. Definitivní hodnotu Z_M získáme až při posledním opuštění vrcholu M.

Složitost celého algoritmu je dána složitostí průchodu grafem do hloubky. Ostatní výpočty spojené s určováním hodnot H_M a Z_M mají konstantní časové nároky. Časová složitost programu pro průchod grafem do hloubky závisí na vhodné volbě vnitřní reprezentace grafu. Při vhodně zvolené reprezentaci (viz uvedená programová ukázka) je přímo úměrná počtu hran v grafu, tj. je řádu n^2, kde n je počet vrcholů grafu.

```
program Mosty (input,output);

{ Format ocekavanych vstupnich dat - zadani grafu:
  - sousedi kazdeho vrcholu vzdy na jednom radku ve tvaru
        ...
  - vrcholy musi byt ocislovany od jedne po jedne a v tomto
    poradi musi byt take radky na vstupu zadany
  - kazda hrana se tedy uvadi dvakrat (na radcich pro jeden
    a pro druhy jeji koncovy vrchol)
  - program pro jednoduchost netestuje spravnost zadanych
    vstupnich dat (nebylo by tezke testy doplnit)          }

const
  MaxPocetMest = 40;
  MaxPocetSilnic = 200;

type
  Mesto   = 1..MaxPocetMest+1;   { fiktivni mesto na konci }
  Silnice = 1..MaxPocetSilnic;

var
  GMesto : array [Mesto] of record
                              Spoje : Silnice;
                              Hloubka : integer;
                              Projito : Boolean;
                            end;
  GSilnice : array [Silnice] of Mesto;
  { pole GMesto a GSilnice predstavuji vnitrni ulozeni grafu
    - ke kazdemu vrcholu je v poli GSilnice ulozen seznam
```

```
    jeho sousedu, polozka Spoje v GMesto urcuje, kde presne
    jsou ulozeni sousedi kazdeho konkretniho vrcholu
    - viz uvodni komentar o tvaru vstupnich dat a procedura
    NactiGraf }

  PocetMest   : 0..MaxPocetMest;
  PocetSilnic : 0..MaxPocetSilnic;
  F:integer;   { pomocna promenna - je potrebna pro spravne
               volani procedury Pruchod v hlavnim programu }


procedure NactiGraf;
{ nacteni vstupnich dat - zadani zkoumaneho grafu }
var dummy:integer;
begin
  PocetMest:=0;
  PocetSilnic:=1;
  repeat
    PocetMest:=PocetMest+1;
    read(dummy);      { cislo mesta - nevyuziva se }
    GMesto[PocetMest].Spoje:=PocetSilnic;
    while not eoln do
      begin
        read(GSilnice[PocetSilnic]);
        PocetSilnic:=PocetSilnic+1;
      end;
    readln;
  until eof;
  GMesto[PocetMest+1].Spoje:=PocetSilnic;
end;

function min(X,Y:integer):integer;
{ pomocna procedura - minimum ze dvou celych cisel }
begin
  if X<> hl then
            { zelena silnice, bud nahoru, nebo dolu }
            Z := min(Z,GMesto[Nasl].Hloubka)
            { pokud vede zelena silnice nahoru, snizime
              hodnotu Z v nasem uzlu }
        end
      else
        begin
          Pruchod(Nasl,hl+1,pomZ);
          if hl+1 = pomZ then
            PisMost(Start,Nasl);    { nalezen most v grafu }
          Z := min(Z,pomZ);   { pripadne snizeni hodnoty Z }
        end;
    end;
end;

begin
  NactiGraf;
  PredPruchodem;
  Pruchod(1,0,F);
  { vzdy je F=0 }
end.
```

4) Barevnost grafu, souvislost se SAT-problémem, barvení rovinných grafů.

1. Úloha: **převod hledání obarvitelnosti grafu třemi barvami na SAT**

Na vstupu je zadán neorientovaný graf. Úkolem je vygenerovat instanci úlohy SAT, která řeší úlohu barvitelnosti grafu třemi barvami (tedy všechny vrcholy jsou obarvené a každá hrana vždy spojuje nestejně obarvené vrcholy).

Poznámka:

Úlohu lze rozšířit na obarvitelnost jiným počtem barev než tří.

1. Připomenutí asymptotické složitosti. Grafy, multigrafy, jejich vlastnosti a reprezentace v počítači. Prohledávání grafu, prioritní fronta.
2. Minimální kostry, algoritmus Borůvkův a Jarníkův. Problém Union-find.
3. Barevnost grafu, souvislost se SAT-problémem, barvení rovinných grafů.
4. Reprezentace dynamických datových struktur. Garbage collector.
5. Počítačová aritmetika. Přirozená čísla, celá čísla, čísla s pevnou řádovou čárkou, čísla s pohyblivou řádovou čárkou. Problémy zpracování čísel: přetečení, zaokrouhlování, komutativita.
6. Odhady výpočetních chyb v numerických algoritmech: hledání kořenu funkce, inverzní matice. Vzájemná kompatibilita mezi architekturami, jazyky a překladači.