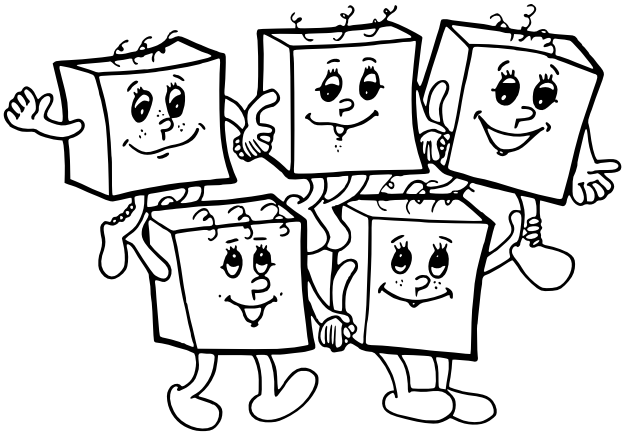
OLYMPIÁDA V INFORMATIKE

NA STREDNÝCH ŠKOLÁCH

http://oi.sk/



dvadsiaty šiesty ročník
školský rok 2010/11

zadania celoštátneho kola
kategória A

1. súťažný deň

Priebeh celoštátneho kola

Celoštátne kolo 26. ročníka Olympiády v informatike, kategórie A, sa koná v dňoch 30. 3. – 2. 4. 2011. Na riešenie úloh prvého, teoretického dňa majú súťažiaci 4,5 hodiny čistého času. Rôzne úlohy riešia súťažiaci na samostatné listy papiera. Akékoľvek pomôcky okrem písacích potrieb (napr. knihy, výpisy programov, kalkulačky) sú zakázané.

Čo má obsahovať riešenie úlohy?

- Slovne popíšte algoritmus.
  Slovný popis riešenia musí byť jasný a zrozumiteľný i bez nahliadnutia do samotného algoritmu/programu.
- Zdôvodnite správnosť vášho algoritmu.
- Uveďte a zdôvodnite jeho časovú a pamäťovú zložitosť.
- Podrobne uveďte dôležité časti algoritmu, ideálne vo forme programu v Pascale alebo C/C++.
- V prípade, že používate vo svojom programovacom jazyku knižnice, ktoré obsahujú implementované dátové štruktúry a algoritmy (napr. STL pre C++), v popise algoritmu stručne vysvetlite, ako by ste napísali program s rovnakou časovou zložitosťou bez použitia knižnice.

Hodnotenie riešení prvého (teoretického) dňa

Za každú úlohu môžete získať od 0 do 10 bodov.

Pokiaľ nie je v zadaní povedané ináč, základným kritériom hodnotenia riešenia je **správnosť** a **efektívnosť** navrhnutého algoritmu. Hodnotí sa aj kvalita popisu riešenia a zdôvodnenie tvrdení o jeho správnosti a efektívnosti.

Efektívnosť algoritmu posudzujeme vypočítaním jeho časovej zložitosti – funkcie, ktorá hovorí, ako dlho vykonanie algoritmu trvá v závislosti od veľkosti vstupných parametrov.

V zadaní úlohy môžu byť uvedené limity na veľkosť premenných. Tieto môžete použiť na odhad toho, ako dobré vaše riešenie je. Na počítači, ktorý vykoná miliardu inštrukcií za sekundu, zrieši vzorové riešenie ľubovoľný povolený vstup nanajvýš za niekoľko sekúnd.

A-III-1 Romantické básničky

Hermi má rád dievčatá, to nie je žiadna novinka. Pre mnohých z vás ani nie je novinkou, že jeho otec je Hviezdoslav. A keďže genetika melie pomaly ale isto, má aj on básnické črevo. A iste je vám jasné, na čo ho využíva – na romantické básničky.

Neviem, či ste už niekedy písali milostnú básničku, ale vedzte, nie je to jednoduché. Keby bol človek zviazaný iba myšlienkami, ešte by to nejak šlo. No básničky sa predsa musia rýmovať! A hľadať také rýmy nebýva jednoduché. Teraz sa ale blíži sústredenie s orchestrom a Hermi musí napísať každej spoluhráčke básničku. A to by sa z toho zjašil, keby musel všetky tie rýmy vymýšľať. Našťastie to môže prepašovať do zadaní olympiády a všetko budete musieť odmakať vy. S programom na hľadanie rýmov už potom Hermi básničky stvorí raz-dva.

Súťažná úloha

Na vstupe je zoznam všetkých n pekných slov, ktoré Hermi pozná. Tento zoznam budeme volať *lexikón*. Ďalej nasleduje m otázok – slov, ku ktorým treba v lexikóne nájsť čo najlepší rým.

Pre jednoduchosť budeme predpokladať, že rým je tým lepší, čím viac spoločných písmen majú dve slová na konci. Odborne povedané, dĺžka spoločného sufixu má byť čo najväčšia. Napríklad slová „`tvari`“ a „`podari`“ majú najdlhší spoločný sufix „`ari`“ dĺžky 3, zatiaľčo „`tvari`“ a „`frajeri`“ majú len spoločný sufix dĺžky 2. Preto ku slovu „`tvari`“ je slovo „`podari`“ lepším rýmom ako slovo „`frajeri`“.

Hermi je navyše veľmi metodický. Ak je v lexikóne na výber viacero slov, ktoré sa s jeho slovom rýmujú rovnako dobre, chce vždy použiť to, ktoré je z nich prvé v abecede. A toto slovo musíte vy nájsť. (Odborne hovoríme, že hľadáme lexikograficky najmenšie slovo. Napr. „`had`“ < „`hadica`“ < „`pes`“.)

Keďže otázok môže byť veľa, snažte sa optimalizovať časovú zložitosť na zodpovedanie jednej otázky – a to aj za cenu pomalšieho predspracovania lexikónu.

Poznámka: Ak úlohu neviete efektívne vyriešiť, uveďte aspoň riešenie, ktoré spomedzi najviac sa rýmujúcich slov vypíše ľubovoľné (teda nie nutne to, ktoré je prvé v abecede).

Formát vstupu

V prvom riadku je číslo n. Nasleduje lexikón: n navzájom rôznych reťazcov zložených z malých písmen anglickej abecedy, každý v samostatnom riadku.

V nasledujúcom riadku je číslo m. Nasledujú otázky: m reťazcov zložených z malých písmen anglickej abecedy, každý v samostatnom riadku.

Môžete predpokladať, že súčet dĺžok všetkých slov v slovníku nepresiahne 10^6 znakov a žiadne slovo na vstupe nebude dlhšie ako 10^4 znakov.

Formát výstupu

Pre každú z Hermiho otázok vypíšte na samostatný riadok slovo z lexikónu, ktoré s ním má najdlhší spoločný sufix. Ak je viac možností, musíte vypísať tú z nich, ktorá je prvá v abecednom poradí.

Príklad

vstup

```
5
prasa
pomysel
ziari
tvari
zabronel
3
krasa
zmari
bager
```

výstup

```
prasa
tvari
pomysel
```

Pri druhom rýme mal Hermi na výber dve slová, ale `tvari` *je v abecede skôr ako* `ziari`.

Pri tretej otázke si všimnite, že žiadne zo slov sa s otázkou vôbec nerýmuje. V takomto prípade je samozrejme správnym výstupom to z nich, ktoré je úplne prvé v abecede.

A-III-2 Ministerstvo

Ministerstvo pre minimalizáciu byrokracie zamestnáva takmer milión úradníkov. Úradníci sú usporiadaní v typickej hierarchickej štruktúre, kde každý úradník má práve jedného priameho nadriadeného. Výnimkou je minister, ktorý nadriadeného nemá. Navyše každý úradník vykonáva len jedinú činnosť – niektorí vyškrtávajú kolónky sprava doľava, iní zľava doprava, iní dávajú okrúhle pečiatky, ... Výnimkou je opäť minister, ktorý ako jediný môže a vie robiť všetko.

Keď chce bežný človek na úrade niečo vybaviť, najprv navštívi nejakého úradníka, ktorý má za úlohu styk s verejnosťou. Ten už ale nevie robiť nič iné, preto ho pošle za svojím nadriadeným. Ak ten robí to, čo klient potrebuje, tak mu pomôže, ináč ho pošle za svojím nadriadeným. A toto sa opakuje, kým daného klienta neobslúžia. (V najhoršom prípade sa klient dostane k samotnému ministrovi a ten ho už zaručene obslúži.)

Po niekoľkých analýzach chodu ministerstva sa zistilo, že niektorí úradníci fakt nič nerobia a ani nikdy robiť nebudú. (Napríklad nikdy nič nerobia úradníci, ktorí nemajú žiadneho podriadeného pre styk s verejnosťou. Iným príkladom sú takí, ktorých všetci priami podriadení robia to isté, čo oni.) No a v duchu hesla „kto nič nerobí, nič nepokazí“, treba nájsť a odmeniť **všetkých** takýchto úradníkov.

Súťažná úloha

Úradníci sú očíslovaní číslami $1, 2, \ldots, n$, pričom číslom 1 je označený minister. Pre každého úradníka i okrem ministra vieme číslo p_i jeho priameho nadriadeného, pričom platí $p_i < i$.

Činnosti sú očíslované číslami $1, 2, \ldots, m$, pričom styk s verejnosťou má číslo 1. Pre každého úradníka i okrem ministra vieme číslo c_i činnosti, ktorú vykonáva.

Vypíšte čísla všetkých úradníkov, ktorí zaručene nič nerobia a ani nikdy nebudú robiť.

Formálne, nech úradník u môže robiť činnosť c. Kedy **môže** tento úradník niečo robiť? Jeden prípad je, ak $c = 1$. Druhý prípad je, ak $c > 1$ a existuje postupnosť úradníkov $u_1, u_2, \ldots, u_t = u$, taká, že platí:

- úradník u_1 vykonáva činnosť 1 (styk s verejnosťou),
- pre každé $i < t$ je úradník u_{i+1} priamym nadriadeným úradníka u_i
- pre žiadne $i < t$ úradník u_i nevykonáva činnosť c.

Ak pre nejakého úradníka nenastáva ani prvý, ani druhý prípad, tak máme istotu, že tento úradník nikdy nič robiť nebude. (Všimnite si, že minister vykonáva všetky činnosti vrátane styku s verejnosťou, preto sa mu môže stať, že niečo robí.)

Pri odhade časovej zložitosti (a analýze, ktoré z možných riešení je ako dobré) môžete predpokladať, že počet činností m je rádovo menší ako počet úradníkov n.

Formát vstupu

V prvom riadku sú čísla n a m. V druhom riadku je pre každého úradníka (od 2 do n) číslo jeho priameho nadriadeného. V treťom riadku je pre každého úradníka (od 2 do n) číslo jeho činnosti.

Formát výstupu

Vypíšte v ľubovoľnom poradí čísla úradníkov, čo nikdy nič nerobia.

Príklad

vstup

```
10 3
1 2 2 2 1 6 6 4 5
2 3 2 2 2 2 1 1 1
```

výstup

```
2 3 7
```

A-III-3 Grafový počítač pomáha krtkovi

V tomto ročníku OI-A sa v poslednej úlohe stretávame so špeciálnym grafovým počítačom. Jeho popis (rovnaký ako v domácom kole) nájdete v študijnom texte za zadaním tejto úlohy.

Súťažná úloha

a) (3 body) Napíšte funkciu pre grafový počítač, ktorá načíta zo vstupu n prirodzených čísel a na výstup ich vypíše usporiadané od najmenšieho po najväčšie. Môžete predpokladať (a využiť) to, že zadané čísla sa zmestia do dátového typu `Value`.

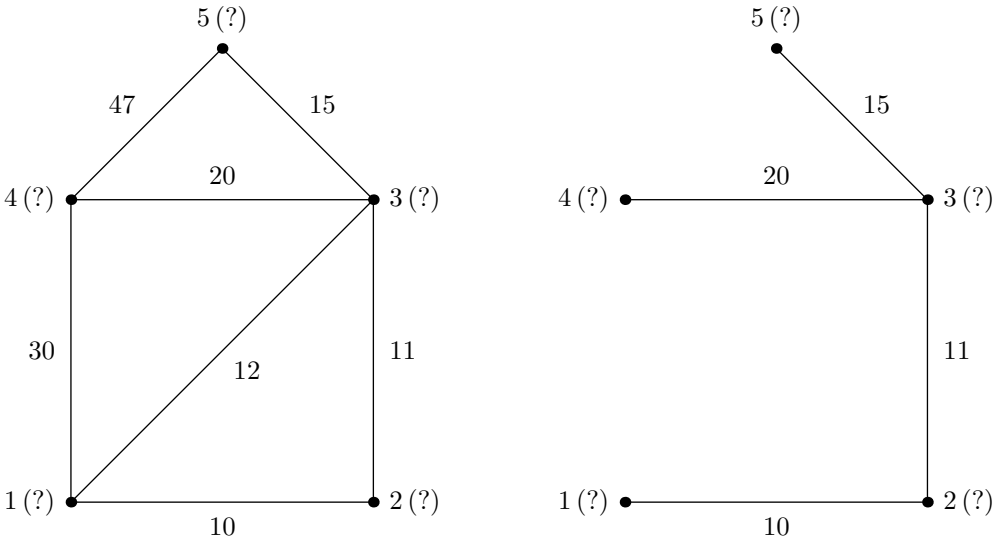
Riešenie s časovou zložitosťou $\Theta(n \log n)$ môže získať nanajvýš jeden bod, pomalšie riešenia nedostanú žiadne body.

Príklad. Pre $n = 5$ a čísla 42, 47, 3, 1, 2 by mala vaša funkcia vypísať na výstup 1, 2, 3, 42, 47.

b) (7 bodov) Malému lenivému krtkovi pri nedávnych dažďoch podmáčalo celú noru. Nora sa kedysi skladala z n brlôžkov a nejakých **navzájom rôzne dlhých** chodbičiek medzi nimi. Po chodbičkách sa dalo (nie nutne priamo) prejsť medzi hociktorými dvoma brlôžkami. Teraz sú ale všetky chodbičky samé blato. Krtko by potreboval **niektoré** chodbičky spevniť, a to tak, aby sa potom cez spevnené chodbičky vedel dostať do všetkých brlôžkov. A keďže je náš krtko lenivý, hľadá takú množinu chodbičiek, ktorá má **najmenšiu celkovú dĺžku**.

Napíšte funkciu pre grafový počítač, ktorá dostane na vstupe graf predstavujúci pôvodnú sieť brlôžkov a chodbičiek a vráti na výstupe jeho podgraf tvorený len tými chodbičkami, ktoré má krtko spevniť. Vo vstupnom grafe má každá hrana váhu predstavujúcu dĺžku príslušnej chodbičky. Váhy sú kladné a navzájom rôzne.

Aj v tejto podúlohe bude pri hodnotení kladený dôraz na to, či je vaše riešenie efektívnejšie ako riešenia, ktoré sa dajú naprogramovať na klasickom počítači. Nezabudnite uviesť dôkaz správnosti vášho algoritmu.



Príklad. Na obrázku vľavo je príklad vstupného grafu. Čísla pri vrcholoch sú ich id, každý vrchol má na začiatku značku `undef`. Čísla pri hranách sú ich váhy. Vpravo je zodpovedajúci výstupný graf. Vo výstupnom grafe môžu byť značky vrcholov a váhy hrán ľubovoľné, zaujíma nás len množina hrán, ktoré obsahuje. Celková dĺžka chodbičiek, ktoré treba spevniť, je 56.

Súhrn povolených inštrukcií

príkaz	účinok príkazu
`AddV(G,z)`	Pridá nový vrchol so značkou `z` a najväčším id.
`DelV(G,id)`	Zmaže vrchol s identifikátorom `id`.
`AddE(G,x,y,w)`	Pridá hranu medzi `x` a `y` s váhou `w`.
`DelE(G,x,y)`	Zmaže hranu medzi `x` a `y`.
`TestE(G,x,y)`	Zistí, či sú `x` a `y` spojené hranou.
`GetV(G,id)`	Vráti značku daného vrcholu.
`SetV(G,id,z)`	Nastaví značku daného vrcholu na danú hodnotu.
`SetAllV(G,z)`	Nastaví každému vrcholu v `G` značku na `z`.
`ReplaceV(G,zold,znew)`	Každému vrcholu v `G`, ktorý mal doteraz značku `zold`, dá značku `znew`.
`GetE(G,x,y)`	Vráti váhu danej hrany.
`SetE(G,x,y,w)`	Nastaví váhu danej hrany na danú hodnotu (aj vyrobí takú hranu, ak treba).
`SetAllE(G,w)`	Nastaví každej hrane v `G` váhu na `w`.
`ReplaceE(G,wold,wnew)`	Každej hrane v `G`, ktorá mala doteraz váhu `wold`, dá váhu `wnew`.
`CountV(G)`	Vráti počet vrcholov v grafe.
`SumV(G)`	Vráti súčet značiek všetkých vrcholov, pričom `undef` sa počíta ako 0.
`CountE(G)`	Vráti počet hrán v grafe.
`SumE(G)`	Vráti súčet váh všetkých hrán, pričom `undef` sa počíta ako 0.
`Iso(G,H,veq,eeq)`	Zistí, či sú grafy `G` a `H` izomorfné pri danom porovnávaní vrcholov a hrán.
`Find(G,H,veq,eeq)`	Nájde jeden najľahší podgraf grafu `G`, ktorý je izomorfný s `H` pri danom porovnávaní vrcholov a hrán. Vráti `EmptyG` ak taký podgraf neexistuje.
`Common(G,H,veq,eeq)`	Nájde jeden spoločný podgraf grafov `G` a `H` s najviac vrcholmi (a ak sa nevie rozhodnúť, tak aj s najviac hranami).
`Join(G,H,veq,eeq)`	Spojí grafy `G` a `H` do jedného, pričom ich akoby zlepí za `Common(G,H,veq,eeq)`.

konštanta	hodnota / význam
`EmptyG`	Prázdny graf (s 0 vrcholmi a 0 hranami).
`undef`	Špeciálna hodnota používaná ako značka vrcholu alebo váha hrany.
`any`	Ľubovoľné dva vrcholy/hrany sa rovnajú (na ohodnotenie sa nehľadí).
`value`	Zodpovedajúce si vrcholy/hrany musia mať rovnaké ohodnotenie. Hodnotu `undef` považujeme za „žolíka“, ktorý sa rovná ľubovoľnej hodnote.
`value_strict`	Vrcholy/hrany musia mať presne rovnaké ohodnotenie, `undef` sa rovná len `undef`-u.
`value_defined`	Vrcholy/hrany musia mať rovnaké ohodnotenie, `undef` sa nerovná ničomu, ani `undef`-u.
`id`	Vrcholy musia mať rovnaké id. (Pri hranách sa toto nedá použiť.)
`none`	Žiadne dva vrcholy/hrany nie sú identické.

Študijný text

Bežné počítače, ktoré máme všetci doma, všetko počítajú v číslach v dvojkovej sústave. Mnohým ľuďom to vyhovuje. Nie však železničným inžinierom v Tasmánii. Tí si jedného dňa uvedomili, že väčšina problémov, ktoré oni potrebujú riešiť, sa týka grafov. Preto si pre vlastnú potrebu navrhli *grafový počítač*, ktorý namiesto čísel pracuje priamo s grafmi. Vie s nimi robiť všetky bežné operácie, a to dokonca v konštantnom čase.

Formálne, *graf* je usporiadaná dvojica (V, E), kde V je konečná množina objektov, ktoré voláme *vrcholy*, a E je konečná množina obsahujúca niektoré neusporiadané dvojice vrcholov. Prvky E voláme *hrany*. Neformálne si graf môžeme predstaviť ako železničnú sieť. Vrcholy sú jednotlivé zastávky, hrany sú priame úseky železničnej trate medzi dvojicami zastávok.

Grafy niekedy môžu byť *ohodnotené*: každému vrcholu, resp. každej hrane môžeme priradiť nejaké číslo. Ich význam môže byť v rôznych situáciách rôzny: raz to môže byť dĺžka koľajníc, inokedy cena cestovného lístka,

a ešte inokedy môžeme pomocou čísel reprezentovať vlastnosti objektov: napríklad stanice, kde stoja rýchliky, môžu mať priradené číslo 1 a ostatné stanice číslo 2. Upresnime ešte, že naše grafy nebudú nikdy obsahovať násobné hrany ani slučky. Teda medzi každými dvomi vrcholmi bude viesť nanajvýš jedna hrana a žiadna hrana nebude začínať a končiť v tom istom vrchole.

Reprezentácia grafu

Každý graf uložený v grafovom počítači má vrcholy očíslované prirodzenými číslami od 1 do ich počtu. Týmto číslam budeme hovoriť *identifikátory* vrcholov (skrátene: *id*).

Hranám identifikátory netreba, každá hrana je totiž jednoznačne určená pomocou id vrcholov, ktoré spája.

Každý vrchol má priradené svoje ohodnotenie, ktoré budeme volať *značka vrcholu*. Každá hrana má priradené svoje ohodnotenie, ktoré budeme volať *váha hrany*. Každé ohodnotenie je buď nezáporné celé číslo alebo špeciálna hodnota `undef` (nedefinované).

Programy budeme písať v bežnom programovacom jazyku, ktorý len rozšírime o niekoľko nových typov premenných a niekoľko nových funkcií. V tomto študijnom texte použijeme Pascal, ale pokojne môžete písať riešenia aj v iných jazykoch, do ktorých si nové súčasti doplníte analogicky.

Nové typy premenných a konštánt

- Typ `Graph` slúži na uloženie grafu. Do premennej typu `Graph` teda môžeme uložiť jeden graf, je jedno ako veľký. Premenné typu `Graph` vieme priradzovať a vieme testovať rovnosť obsahu dvoch takýchto premenných. Obe tieto operácie počítač vykoná v konštantnom čase.
- Konštanta `EmptyG` typu `Graph` obsahuje prázdny graf, teda graf s 0 vrcholmi (a teda nutne aj 0 hranami).
- Typ `Value` slúži na uloženie jedného ohodnotenia. Premenná typu `Value` teda môže nadobudnúť ako hodnotu ľubovoľné nezáporné celé číslo alebo špeciálnu hodnotu `undef`. Až na `undef` fungujú operácie s týmto typom rovnako ako operácie s bežnými celočíselnými typmi premenných.

Operácie meniace štruktúru grafu

- Funkcia `AddV(G,z)` pridá do grafu `G` nový vrchol a priradí mu značku `z`. Do nového vrcholu nevedú žiadne hrany a jeho id je rovné novému počtu vrcholov. Návratová hodnota funkcie `AddV(G,z)` je toto id.
- Procedúra `DelV(G,id)` zmaže z grafu `G` vrchol s identifikátorom `id` a všetky hrany, ktoré do tohto vrcholu viedli. Následne poposúva id vrcholov tak, aby nevznikla diera v ich číslovaní. Teda z vrcholu s číslom `id`+1 sa stane `id`, z `id`+2 sa stane `id`+1, atď.
- Procedúra `AddE(G,x,y,w)` vytvorí v grafe `G` hranu medzi vrcholmi s id `x` a `y` a priradí jej ohodnotenie `w`.
- Procedúra `DelE(G,x,y)` odstráni z grafu `G` hranu medzi vrcholmi s id `x` a `y`.
- Funkcia `TestE(G,x,y)` vráti `true` ak sú vrcholy s id `x` a `y` v grafe `G` spojené hranou a `false` ak nie sú.

Všetky tieto funkcie a procedúry bežia v konštantnom čase. Je nutné ich volať so správnymi parametrami, inak nastane chyba a program bude ukončený. Napr. procedúru `DelE` je povolené volať len s id vrcholov, ktoré skutočne sú v danom grafe spojené hranou.

Operácie meniace značky vrcholov a váhy hrán

- Funkcia `GetV(G,id)` vráti značku zadaného vrcholu.
- Procedúra `SetV(G,id,z)` nastaví značku zadaného vrcholu na zadanú hodnotu.
- Procedúra `SetAllV(G,z)` nastaví každému vrcholu v `G` značku na `z`.
- Procedúra `ReplaceV(G,zold,znew)` každému vrcholu v `G`, ktorý mal doteraz značku `zold`, zmení značku na `znew`.
- Funkcia `GetE(G,x,y)` a procedúry `SetE(G,x,y,w)`, `SetAllE(G,w)` a `ReplaceE(G,wold,wnew)` sú definované rovnako ako funkcie pracujúce s vrcholmi. Hrana je popísaná id vrcholov `x` a `y`, ktoré spája.

  Ak pri volaní `SetE(G,x,y,w)` hrana medzi `x` a `y` neexistovala, tak je najskôr do grafu pridaná.

Štatistické funkcie

- Funkcia `CountV(G)` vráti počet vrcholov v grafe.
- Funkcia `SumV(G)` vráti súčet značiek všetkých vrcholov, pričom `undef` sa počíta ako 0. Pokiaľ graf nemá žiadne vrcholy, funkcia vráti 0.
- Analogicky definujeme funkcie `CountE(G)` a `SumE(G)` pre hrany a ich váhy.

Globálne operácie

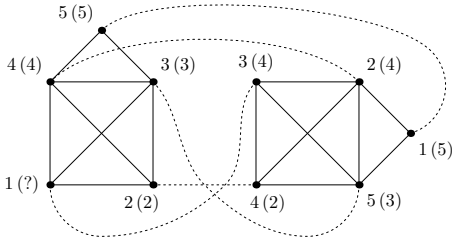
- Funkcia `Iso(G,H,veq,eeq)` zistí, či sú grafy `G` a `H` *izomorfné* – t. j., či možno jednému z grafov prečíslovať vrcholy tak, aby sa zhodoval s druhým grafom. Dva grafy sú zhodné, pokiaľ majú rovnaké množiny vrcholov aj hrán; navyše sa im musia zhodovať značky vrcholov a váhy hrán podľa toho, ako určujú parametre `veq` (pre vrcholy) a `eeq` (pre hrany). Tieto parametre môžu nadobúdať nasledujúce hodnoty:

  `any`: Ľubovoľné dva vrcholy/hrany sa rovnajú (na ohodnotenie sa nehľadí).
  `value`: Zodpovedajúce si vrcholy/hrany musia mať rovnaké ohodnotenie. Hodnotu `undef` ale považujeme za „žolíka“, ktorý sa rovná ľubovoľnej hodnote.
  `value_strict`: Vrcholy/hrany musia mať presne rovnaké ohodnotenie, `undef` sa rovná len `undef`-u.
  `value_defined`: Vrcholy/hrany musia mať rovnaké ohodnotenie, `undef` sa nerovná ničomu, ani `undef`-u. Teda akonáhle má nejaký vrchol/hrana ohodnotenie `undef`, nemôžeme ho priradiť žiadnemu vrcholu/hrane v druhom grafe.
  `id`: Vrcholy musia mať rovnaké id (toto možno aplikovať len na vrcholy, lebo hrany nemajú id). Inými slovami, zakazujeme prečíslovávať vrcholy, ale na ich ohodnotenie nehľadíme.
  `none`: Žiadne dva vrcholy/hrany nie sú identické. (I keď to vyzerá neužitočne, túto možnosť použijeme v ďalších funkciách.)

  Izomorfizmus je znázornený na nasledujúcom obrázku. Plné čiary predstavujú hrany dvoch 5-vrcholových grafov. Čísla pred zátvorkami sú id vrcholov, v zátvorkách sú ich značky (otáznik označuje hodnotu `undef`). Všetky hrany majú váhu `undef`.

  Volaním `Iso(G,H,any,any)` zistíme, či vieme zmeniť id vrcholov grafu `G` tak, aby sme dostali v `G` presne tú istú množinu vrcholov ako v `H`. Takýchto prečíslovaní môže vo všeobecnosti byť aj viac. V príklade na obrázku existujú dve možnosti, pre jednu z nich je priradenie vrcholov zobrazené čiarkovanými čiarami.

  Ak navyše požadujeme, aby sa zhodovali aj značky vrcholov, resp. váhy hrán, dosiahneme to zmenením `veq`, resp. `eeq`, na inú hodnotu ako `any`. Grafy z nášho obrázku budú izomorfné, ak nastavíme `veq` na `value` alebo `any` a zároveň `eeq` na `any`, `value` alebo `value_strict`. (Čiarkované čiary predstavujú priradenie vrcholov, ktoré vo všetkých týchto prípadoch funguje.)

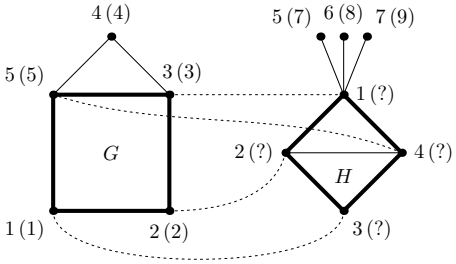


  Pri ostatných kombináciách `veq` a `eeq` tieto dva grafy izomorfné nie sú.

- Funkcia `Find(G,H,veq,eeq)` nájde podgraf grafu `G`, ktorý je izomorfný s grafom `H`. (Podgraf je taký graf, ktorý možno získať z `G` odstránením niektorých vrcholov a hrán.) Návratovou hodnotou funkcie bude tento podgraf, pričom vrcholy budú očíslované podľa grafu `H` a ohodnotenie vrcholov a hrán bude pochádzať z grafu `G`. Pokiaľ hľadaný podgraf neexistuje, funkcia vráti `EmptyG`. Parametre `veq` a `eeq` určujú rovnako ako pri funkcii `Iso`, ako sa správa izomorfizmus.

  Pokiaľ existuje viacej izomorfných podgrafov, funkcia `Find` nájde najľahší z nich (t. j. ten, ktorý má najmenší súčet váh hrán, ako by ho spočítala funkcia `SumE`). Pokiaľ aj tak existuje viacej možností, `Find` vráti ľubovoľnú z nich.
- Funkcia `Common(G,H,veq,eeq)` nájde najväčší spoločný podgraf grafov `G` a `H`. Presnejšie, nájde graf, ktorý je izomorfný (podľa `veq` a `eeq`) s niektorým podgrafom `G` i s niektorým podgrafom `H`. Zo všetkých možných riešení si navyše vyberie také, ktoré má najväčší možný počet vrcholov, a z takých potom to s najväčším počtom hrán. Pokiaľ aj tých je viacej, vyberie si ľubovoľné z nich.

  Výsledný graf bude mať id vrcholov v rovnakom poradí, ako ho mal zodpovedajúci podgraf v `G`, len budú prečíslované od 1 po ich počet, aby v číslovaní neboli diery. Ohodnotenie vrcholov a hrán bude taktiež zdedené z grafu `G`.

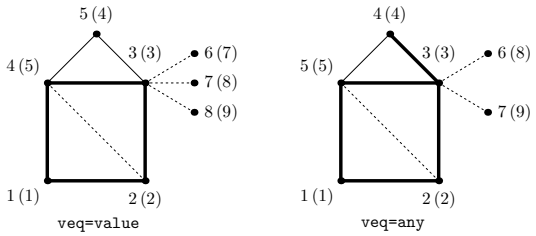


  Na obrázku sú opäť plnými čiarami nakreslené dva grafy. Pri `veq=value` a `eeq=any` je ich najväčším spoločným podgrafom „štvorec“ obtiahnutý tučne. Čiarkované hrany ukazujú jedno možné priradenie vrcholov.

  Ak zmeníme `veq` na `any`, pribudne do spoločného podgrafu ešte jedna nová hrana a jej koncový vrchol: v ľavom grafe to môže byť jedna z hrán 3-4 a 3-5, v pravom jedna z hrán 1-5, 1-6 a 1-7.

  Najväčší spoločný podgraf nemusí byť určený jednoznačne. Napríklad vo vyššie popísanej situácii ním nemusí byť len zvýraznený štvorec. Rovnako veľký je aj podgraf, ktorý je v ľavom obrázku určený vrcholmi 3, 4, 5, 1 a v pravom 4, 1, 2, 3.
- Funkcia `Join(G,H,veq,eeq)` zlúči grafy `G` a `H`. Môžete si to predstaviť tak, že ich „zlepí za ich najväčší spoločný podgraf“. Urobí to tak, že najprv nájde najväčší spoločný podgraf (tak ako vo funkcii `Common`), potom k nemu doplní zostávajúce vrcholy grafu `G` a nakoniec vrcholy grafu `H` (id vrcholov výsledného grafu teda budú v tomto poradí). Hrany, váhy a značky pritom zdedí z oboch grafov, pričom pokiaľ sa nejaký vrchol alebo hrana vyskytujú v oboch grafoch, použije sa ohodnotenie z grafu `G`.

  Ak by sme pre grafy z predchádzajúceho obrázku použili `Join`, mohli by sme dostať napr. nasledujúci výstup:



  Tučnými čiarami je vyznačená spoločná časť (všimnite si rozdiely v id vrcholov), tenké neprerušované hrany pochádzajú z grafu `G`, čiarkované hrany sú z grafu `H`.

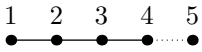
  (Keďže aj pri `veq=value`, aj pri `veq=any` existovalo viac možností, ako vybrať najväčší spoločný podgraf, existuje aj viacero možností, ako bude vyzerať výstup funkcie `Join` pre tieto dva grafy. Výstup na obrázku

vľavo zodpovedá priradeniu z obrázku k funkcii `Common`, výstup na obrázku vpravo zodpovedá tomu istému priradeniu a navyše vrchol 4 vľavo je priradený vrcholu 5 vpravo.)

Všetky operácie predpokladajú, že dostanú korektný vstup – nie je teda napríklad povolené volať ich s id neexistujúceho vrcholu alebo upravovať grafovú premennú, do ktorej ste ešte nepriradili, a podobne. Všetky grafové operácie trvajú konštantný čas.

Príklad 1: Tvorba cesty

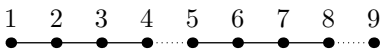
Ukážeme, ako vytvoriť cestu dĺžky n. To je graf s $n+1$ vrcholmi a n hranami, v ktorom je každý vrchol spojený hranou s nasledujúcim. Určite by sme cestu mohli vytvárať postupne, napríklad takto:




```
function cesta(n: Integer): Graph;
var i, posledny, novy: Integer;
    G: Graph;
begin
   G := EmptyG;
   posledny := AddV(G, 0);
   for i := 1 to n do begin
      novy := AddV(G, 0);
      AddE(G, posledny, novy, undef);
      posledny := novy;
   end;
   cesta := G;
end;
```

Začíname s jediným vrcholom (má id 1) a potom n-krát pridáme nový vrchol a hranu doň. (Vrcholom dávame značky 0, hranám nedefinované váhy, čo sa bude hodiť neskôr.) Celý postup teda trvá lineárne dlho a vytvorí cestu začínajúcu vo vrchole s id 1 a končiacu vo vrchole s id $n+1$.

Ten istý graf však vieme zostrojiť aj rýchlejšie. Predstavme si na chvíľku, že už máme v `G` zostrojenú časť cesty, povedzme tvorenú k vrcholmi. Pomocou `Join(G,G,none,none)` vytvoríme nový graf, ktorý obsahuje dve kópie tejto cesty: jednu s id $1,\ldots,k$, druhú s id $k+1,\ldots,2k$. Do toho stačí následne pridať hranu z k do $k+1$ a máme cestu dĺžky $2k$. Tento prístup využijeme v nasledujúcom (rekurzívnom) riešení úlohy:



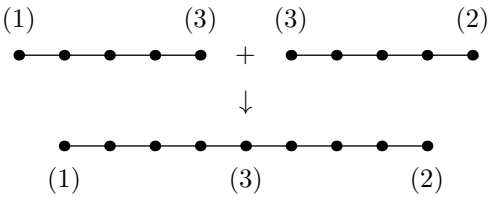

```
function cesta(n: Integer): Graph;
var vysledok: Graph;
    polovica: Integer;
begin
   if n = 0 then begin { cesta dlzky 0 je len jeden vrchol }
      vysledok := EmptyG;
      AddV(vysledok, 0);
   end else begin
      { rekurzivne vytvorime cestu polovicnej dlzky }
      polovica := (n-1) div 2;
      vysledok := cesta(polovica);
      { vyrobime 2 kopie a spojime ich }
      vysledok := Join(vysledok, vysledok, none, none);
      AddE(vysledok, polovica+1, polovica+2, undef);
      { ked polovica nevysla celociselna, pridame este hranu }
      if n mod 2 = 0 then begin
         AddV(vysledok, 0);
         AddE(vysledok, n, n+1, undef);
      end;
  end;
  cesta := vysledok;
end;
```

Pri každom rekurzívnom volaní sa n zmenší aspoň na polovicu, časová zložitosť tohto riešenia je teda $O(\log n)$.

Ukážeme ešte jedno riešenie, tentoraz založené na spájaní ciest za vrchol. Budeme vytvárať cesty, ktorých začiatočný vrchol bude mať značku 1, koncový vrchol značku 2 a všetky ostatné vrcholy značku `undef`. Keď

chceme dve cesty spojiť do jednej, preznačíme koncový vrchol prvej a začiatočný vrchol druhej na 3 a zavoláme `Join` s `veq=value_defined`. Tým spôsobíme, že sa vrcholy označené trojkou stotožnia a vznikne cesta dvojnásobnej dĺžky. (Všimnite si, že keby sme namiesto `value_defined` použili `value` alebo `value_strict`, stotožnili by sa aj vnútorné vrcholy ciest, čo nechceme.) Potom ešte odstránime pomocnú značku 3 a prepíšeme ju na `undef`. Program tentokrát pre jednoduchosť napíšeme len pre $n = 2^k$:

```
function cesta(n: Integer): Graph;
var G, T1, T2: Graph;
begin
   if n = 1 then begin
      G := EmptyG;
      AddV(G, 1);
      AddV(G, 2);
      AddE(G, 1, 2, undef);
  end else begin
      T1 := cesta(n div 2);
      T2 := T1;
      ReplaceV(T1, 2, 3);
      ReplaceV(T2, 1, 3);
      G := Join(t1, t2, value_defined, any);
      ReplaceV(G, 3, undef);
  end;
  cesta := G;
end;
```



Časová zložitosť tohto riešenia je opäť logaritmická od n.

Príklad 2: Obchodný cestujúci

Všetci poznáme prešibaných obchodníkov. Predávajú ktovie čo a najradšej by boli, keby ich po predaji už kupujúci nikdy nenašiel. Predstavme si takého obchodníka. Teraz sa nachádza v meste (t. j. vo vrchole) číslo 1. Chce prejsť celú krajinu (graf) po cestách (hranách) tak, aby navštívil každé mesto práve raz a potom sa vrátil domov. Navyše pri tom chce najazdiť čo najmenej. Inými slovami, celková váha použitých hrán by mala byť najmenšia možná.

Na obvyklom počítači pre tento problém nepoznáme riešenie v polynomiálnom čase. Pokiaľ ale máme k dispozícii grafový počítač, pôjde to veľmi efektívne.

Stačí totiž vyrobiť cyklus z n hrán a potom funkciou `Find` nájsť jeho najľahší výskyt v grafe predstavujúcom mapu krajiny. Cyklus vytvoríme tak, že podľa predchádzajúceho príkladu vytvoríme cestu s n vrcholmi a $n-1$ hranami a potom spojíme hranou jej prvý vrchol s posledným. To vieme spraviť v logaritmickom čase. Následné volanie funkcie `Find` beží v konštantnom čase, a teda nám časovú zložitosť nepokazí.

```
function cestujuci(mapa: Graph): Graph;
var trasa: Graph;
begin
   trasa := cesta(CountV(mapa)-1);
   AddE(trasa, 1, CountV(mapa), undef);
   cestujuci := Find(mapa, trasa, any, any);
end;
```

SLOVENSKÁ KOMISIA OLYMPIÁDY V INFORMATIKE
DVADSIATY ŠIESTY ROČNÍK OLYMPIÁDY V INFORMATIKE
Zodpovedný redaktor: Michal Forišek
Sadzba programom LaTeX