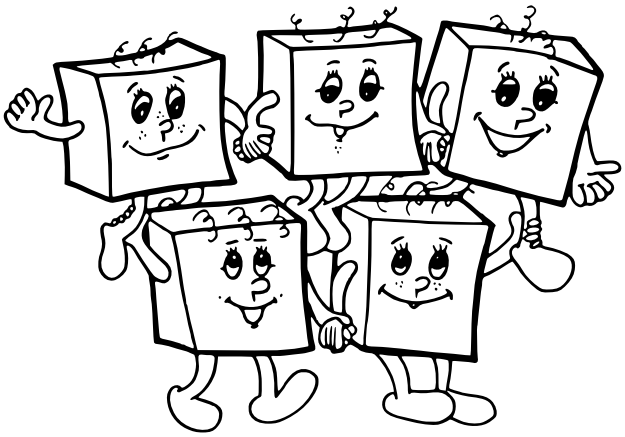
OLYMPIÁDA V INFORMATIKE

NA STREDNÝCH ŠKOLÁCH

http://oi.sk/



dvadsiaty šiesty ročník
školský rok 2010/11

zadania krajského kola
kategória A

Priebeh krajského kola

Krajské kolo 26. ročníka Olympiády v informatike, kategória A, sa koná 11. 1. 2011 v dopoludňajších hodinách. Na riešenie úloh majú súťažiaci 4 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

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 do niekoľko sekúnd.

A-II-1 Vlak

Na nákladnej stanici v istom nemenovanom meste stojí utajený vlak. Teda, presnejšie povedané, iba vozne. A v nich prísne strážený náklad – v jednom gumové medvedíky, v inom fialové kravy, v ďalších zase iné dobroty. Pretože vlak obsahuje spomínané tajné materiály, riaditeľ stanice by ho chcel čím skôr vypraviť, aby sa po jednom anonymnom tipe náhodou celá stanica nezmenila na bojisko proti organizovanému detskému zločinu. (Predstavte si vagón plný čokolády, stovky detí snažiacich sa ho vyžrať a železničnú políciu márne sa snažiacu zabrániť tejto katastrofe.)

Lenže je tu istý háčik – podľa nového nariadenia musí výpravca poslať do konečnej stanice vlaku rozpis, v akom poradí majú prísť vozne. Na prvý pohľad sa zdá, že je to triviálna otázka, ale lokomotíva môže byť počas medzizastávky v niektorej stanici preradená na druhý koniec vlaku. A nikto nevie (z dôvodov utajenia), ktorou trasou ten vlak vlastne pôjde, nie to ešte koľkokrát bude lokomotíva preradená. Preto sa na stanici rozhodli, že z vlaku vyradia niekoľko vagónov tak, aby zostávajúce vagóny vyzerali rovnako, nech je už lokomotíva na ktoromkoľvek konci. A samozrejme, chcú vyradiť čo najmenej vagónov.

Súťažná úloha

Na vstupe je postupnosť písmen – typov vagónov. Výstupom vášho programu majú byť pozície vagónov (písmen), ktoré treba vyradiť z vlaku, aby bol výsledný vlak (postupnosť písmen) rovnaký pri čítaní spredu aj odzadu (teda aby výsledný vlak bol palindrómom).

Pokiaľ by existovalo viacero možností, nájdite a vypíšte tú, pri ktorej treba odstrániť najmenší možný počet vagónov. Pokiaľ by takýchto možností stále bolo viac, vypíšte ľubovoľnú z nich.

Formát vstupu

Na vstupe sú dva riadky. V prvom je jediné číslo n ($1 \leq n \leq 10\,000$) udávajúce počet vagónov vlaku. Druhý riadok obsahuje postupnosť n znakov, ktoré reprezentujú jednotlivé typy vagónov.

Formát výstupu

Na výstup vypíšte dva riadky. V prvom riadku vypíšte jediné číslo k ($0 \leq k \leq n-1$) udávajúce najmenší počet vagónov, ktoré je potrebné z vlaku vyradiť. V druhom riadku vypíšte k rôznych čísel oddelených medzerou: čísla vagónov, ktoré treba vyradiť. Vagóny sú očíslované od 1 po n v poradí, v akom sú na vstupe.

Príklad

vstup	výstup
6 ABCDBA	1 3

Odstránením tretieho vagóna vznikne vlak s vagónmi ABDBA.
Iným optimálnym riešením je odstrániť vagón D.

vstup	výstup
7 ABECEDA	2 2 6

Odstránením druhého a šiesteho vagóna vzniká AECEA.

vstup	výstup
7 ABECADA	4 3 4 6 7

Výstupu zodpovedá vlak ABA. Iných, tiež optimálnych riešení je v tomto prípade mnoho.

A-II-2 Jabloňový sad

V jednom malom kráľovstve vyrástol strom so zlatými jablkami. Kráľ prikázal záhradníkom, nech takých jabloní vypestujú celý sad. Keby však vysadili zlaté jadierka len tak, vyrástli by z nich obyčajné plánky. Našťastie alchymisti vedia vypočítať, kam zasadiť nasledujúce jadierko, aby aj z neho vyrástol strom so zlatými jablkami.

Aby zlaté jablká nik neukradol, musel kráľ čoskoro začať so stavbou oplotenia. Najlacnejšie riešenie, ktoré vymyslel, vyzeralo nasledovne: Teraz nechá postaviť najkratší možný plot okolo všetkých n už rastúcich stromov. V budúcnosti len bude tento plot podľa potreby rozširovať vždy, keď bude treba nový strom zasadiť mimo oplotenej oblasti. Navyše pri rozširovaní sa vždy dá časť plota rozobrať a použiť znova na inom mieste, takže bude treba dokúpiť len toľko materiálu, o koľko bude nový plot dlhší od starého.

Súťažná úloha

Na začiatku dostanete kartézske súradnice n bodov (jabloní) v rovine: $[x_1, y_1]$, ..., $[x_n, y_n]$. Musíte zistiť dĺžku najkratšieho plota takého, že všetky jablone ležia na obvode alebo vo vnútri oplotenej oblasti.

Potom budete postupne dostávať súradnice ďalších m bodov: miesta, kde budú zasadené nasledujúce jablone. Vždy, keď načítate súradnice ďalšieho bodu, musíte vypočítať a vypísať, o koľko práve narástla dĺžka najkratšieho plota. Vami vypočítaný údaj zodpovedá tomu, koľko pletiva je potrebné dokúpiť. (Uvedomte si, že nový plot nikdy nebude kratší od starého. Ak nový bod leží na oplotení alebo v jeho vnútri, výstupom je nula.)

Očakávajte, že čísla n a m budú veľké. Efektívne riešenie by teda malo pri spracúvaní nového bodu šikovne využiť skôr vypočítané informácie. (Nejaké body však samozrejme získate aj za korektné riešenie, ktoré zakaždým odznova správne zostrojí celý plot.)

Dôležité: Ak v riešení tejto úlohy použijete nejakú všeobecne známu dátovú štruktúru (napr. haldu alebo binárny vyhľadávací strom), stačí uviesť popis jej vlastností, ktoré v riešení využívate. Nie je potrebné rozpisovať jej implementáciu.

Formát vstupu

V prvom riadku vstupu bude číslo n, ktoré udáva začiatočný počet jabloní. Nasleduje n riadkov, v i-tom z nich sú dve čísla x_i a y_i oddelené medzerou: súradnice jedného z už zasadených stromov.

Ďalší riadok obsahuje číslo m. Nasleduje m riadkov, v i-tom z nich sú dve čísla x_{n+i} a y_{n+i} oddelené medzerou: súradnice stromu, ktorý bude v budúcnosti vysadený ako i-ty v poradí.

Formát výstupu

Na výstup vypíšte $m + 1$ riadkov. V prvom riadku bude súčasná dĺžka oplotenia, t.j. obvod najmenšieho obrazca, ktorý obsahuje všetky body $[x_1, y_1]$ až $[x_n, y_n]$. Nasleduje m riadkov, v i-tom z nich má byť uvedené, o koľko sa obvod obrazca zväčší po pridaní bodu $[x_{n+i}, y_{n+i}]$ k ostatným.

Príklad

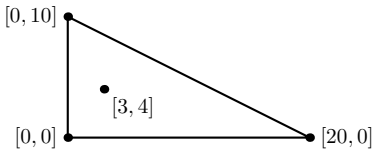
vstup

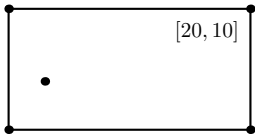
```
4
0 0
0 10
20 0
3 4
2
20 10
-5 5
```

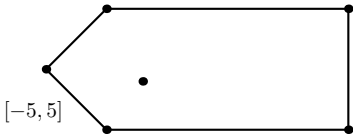
výstup

```
52.36068
7.63932
4.14214
```

Na obrázku vľavo je začiatočný plot, jeho dĺžka je $10 + 20 + \sqrt{10^2 + 20^2}$. *Po pridaní piateho bodu dostaneme obrázok v strede, jeho obvod je 60, teda narástol o 7.64 oproti predchádzajúcemu. Po pridaní šiesteho bodu dostaneme obrázok napravo.*







A-II-3 Mažoretky

Na planéte Diks majú v hlavnom meste už vyše sto rokov slávny súbor mažoretiek. Každé ráno slávia príchod nového dňa sprievodom po uliciach mesta, pri ktorom kráčajú v rade jedna za druhou a predvádzajú rôzne prvky. Občania mesta sledujú vystúpenie a hneď sa im radostnejšie vstáva do školy a do práce.

Mažoretiek v súbore je presne n. Aby sa občania mesta nenudili, každý deň idú v sprievode v inom poradí. Zhodou okolností má rok na planéte Diks presne $1 \cdot 2 \cdot \ldots \cdot n = n!$ dní, takže za jeden rok použijú práve raz každé možné poradie.

Aby sa im to nepoplietlo, poradie si určujú systematicky, a to nasledovne. Každá mažoretka má svoje jednoznačné číslo od 1 po n. Ich poradie v konkrétny deň teda vieme popísať ako postupnosť čísel. Ľubovoľné dve poradia $(a_1, \ldots, a_n)$ a $(b_1, \ldots, b_n)$ môžeme teraz porovnať nasledovne: Nech i je najmenší index, pre ktorý $a_i \neq b_i$. Potom to poradie, ktoré má na i-tej pozícii mažoretku s menším číslom, bude použité v skorší deň v roku ako to druhé.

Napríklad pre $n = 4$ bude poradie $(3, 1, 2, 4)$ použité skôr ako $(3, 4, 1, 2)$, lebo na druhej pozícii má prvé z nich mažoretku s menším číslom.

Počas jedného roka by pre $n = 3$ mažoretky postupne použili nasledovných 6 usporiadaní:
$(1, 2, 3)$, $(1, 3, 2)$, $(2, 1, 3)$, $(2, 3, 1)$, $(3, 1, 2)$ a nakoniec $(3, 2, 1)$.

Súťažná úloha

Na vstupe je číslo $n \geq 2$ a jedno konkrétne poradie mažoretiek.
a) (3 body) Napíšte program, ktorý vypíše poradie mažoretiek v nasledujúci deň.
b) (7 bodov) Napíšte program, ktorý vypíše poradie mažoretiek presne o pol roka.
V oboch podúlohách nezabudnite na to, že dotyčný deň môže byť v nasledujúcom kalendárnom roku.

Príklad pre podúlohu a)

vstup

```
5
1 4 2 5 3
```

výstup

```
1 4 3 2 5
```

Príklady pre podúlohu b)

vstup

```
3
1 3 2
```

výstup

```
3 1 2
```

Rok na tejto planéte má $3! = 6$ *dní, hľadáme teda poradie o* $6/2 = 3$ *dni. V zadaní je uvedený zoznam všetkých poradí pre* $n = 3$*, z neho vidíme, že o 3 dni po poradí* $(1, 3, 2)$ *bude použité poradie* $(3, 1, 2)$.

vstup

```
3
3 1 2
```

výstup

```
1 3 2
```

O ďalšie 3 dni bude opäť použité poradie $(1, 3, 2)$.

A-II-4 Grafový počítač a kompletné grafy

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) (5 bodov) Napíšte funkciu pre grafový počítač, ktorá dostane ako parameter číslo n a na výstupe vráti premennú obsahujúcu kompletný graf K_n. To je graf s n vrcholmi, z ktorých každé dva sú spojené hranou.

(Najviac 4 body môžete získať za riešenie, ktoré túto úlohu rieši za predpokladu, že n je mocninou dvoch.)

b) (5 bodov) Hovoríme, že graf obsahuje *kliku* veľkosti k, ak v ňom existuje k-tica vrcholov, ktoré sú spojené každý s každým. *Klikovosť* grafu je počet vrcholov v najväčšej klike, ktorú obsahuje.

Napíšte funkciu pre grafový počítač, ktorá dostane ako parameter premennú obsahujúcu graf G a na výstupe vráti celé číslo udávajúce jeho klikovosť.

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

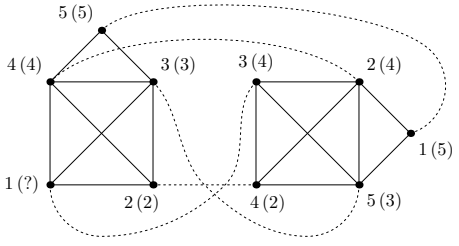
- Funkcia `Iso(G,H,veq,eeq)` zistí, či sú grafy `G` a `H` *izomorfné* – t. j., či možno jednému z grafov prečíslovať vrcholy tak, aby sa zhodoval s druhým grafom. Dva grafy sú zhodné, pokiaľ majú rovnaké množiny vrcholov aj hrán; navyše sa im musia zhodovať značky vrcholov a váhy hrán podľa toho, ako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

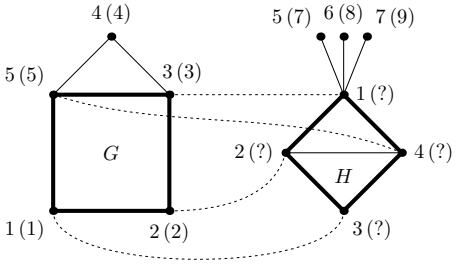


  Pri ostatných kombináciách `veq` a `eeq` tieto dva grafy izomorfné nie sú.

- 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`.

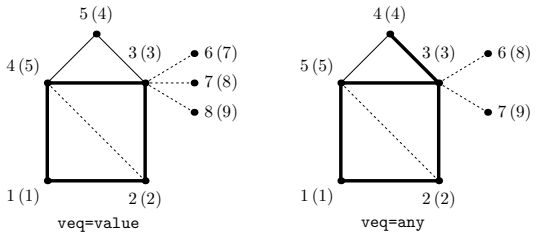


  Na obrázku sú opäť plnými čiarami nakreslené dva grafy. Pri `veq=value` a `eeq=any` je ich najväčším spoločným podgrafom „štvorec“ obtiahnutý tučne. Čiarkované hrany ukazujú jedno možné priradenie vrcholov.

  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`.

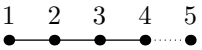
  (Keďže aj pri `veq=value`, aj pri `veq=any` existovalo viac možností, ako vybrať najväčší spoločný podgraf, existuje aj viacero možností, ako bude vyzerať výstup funkcie `Join` pre tieto dva grafy. Výstup na obrázku

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

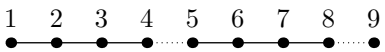
Ukážeme, ako vytvoriť cestu dĺžky n. To je graf s $n+1$ vrcholmi a n hranami, v ktorom je každý vrchol spojený hranou s nasledujúcim. Určite by sme cestu mohli vytvárať postupne, napríklad takto:




```
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:



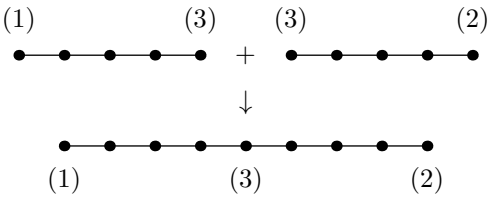

```
function cesta(n: Integer): Graph;
var vysledok: Graph;
    polovica: Integer;
begin
   if n = 0 then begin { cesta dlzky 0 je len jeden vrchol }
      vysledok := EmptyG;
      AddV(vysledok, 0);
   end else begin
      { rekurzivne vytvorime cestu polovicnej dlzky }
      polovica := (n-1) div 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