Informácie a pravidlá

Pre koho je súťaž určená?

- Do **kategórie B** sa smú zapojiť len tí žiaci základných a stredných škôl, ktorí ešte ani v tomto, ani v nasledujúcom školskom roku nebudú končiť strednú školu.
- Do **kategórie A** sa môžu zapojiť všetci žiaci (základných aj) stredných škôl.

Odovzdávanie riešení domáceho kola

Riešitelia domáceho kola odovzdávajú riešenia sami, v elektronickej podobe, a to priamo na stránke olympiády: http://oi.sk/. Odovzdávanie riešení bude spustené najneskôr do konca septembra.

Termíny 41. ročníka OI

- Riešenia domáceho kola je potrebné v kategórii A odovzdať najneskôr **15. 11. 2025**, v kategórii B najneskôr **30. 11. 2025**.
- Krajské kolo oboch kategórií prebehne 20. 1. 2026 zvlášť v každom kraji.
- Celoštátne kolo kategórie A sa bude konať v dňoch 18.-21. 3. 2026 (súťaží sa vo štvrtok a piatok).
- Detailnejšie informácie o priebehu súťaže nájdete na webe OI: https://oi.sk/?s=pravidla

Ako majú vyzerať riešenia úloh?

V **praktických úlohách** je vašou úlohou vytvoriť program, ktorý bude riešiť zadanú úlohu. Program musí byť v prvom rade korektný a funkčný, v druhom rade sa snažte aby bol čo najefektívnejší.

V kategórii B môžete použiť ľubovoľný programovací jazyk.

V kategórii A musíte riešenia praktických úloh písať v jednom z podporovaných jazykov (napr. C++, Python alebo Java). Presný zoznam podporovaných jazykov vrátane konkrétnych verzií ich kompilátorov a interpreterov nájdete v systéme na odovzdávanie riešení. Taktiež tam nájdete presnejší popis toho, ako sa majú vaše programy správať, napr. detaily realizácie vstupu a výstupu.

Odovzdaný program bude automaticky otestovaný na viacerých vopred pripravených testovacích vstupoch. Podľa toho, na koľko z nich dá správnu odpoveď, vám budú pridelené body. Výsledok testovania sa dozviete krátko po odovzdaní. Ak váš program nezíska plný počet bodov, budete ho môcť vylepšiť a odovzdať znova, až do uplynutia termínu na odovzdávanie.

Ak nie je v zadaní povedané ináč, riešenia **teoretických úloh** musia obsahovať:

- podstatné časti algoritmu ako program (v ľubovoľnom bežnom prog. jazyku) alebo ekvivalentný pseudokód
- podrobný slovný popis použitého algoritmu
- zdôvodnenie jeho správnosti
- diskusiu o efektivite zvoleného riešenia (odhad časovej a pamäťovej zložitosti)

Ak používate v programe netriviálne algoritmy alebo dátové štruktúry (napr. rôzne súčasti STL v C++), súčasťou slovného popisu algoritmu by mal byť stručný popis ich implementácie.

Usporiadateľ súťaže

Olympiádu v informatike (OI) vyhlasuje *Ministerstvo školstva SR* v spolupráci so *Slovenskou informatickou spoločnosťou* (odborným garantom súťaže) a *Slovenskou komisiou Olympiády v informatike*. Súťaž organizuje *Slovenská komisia OI* a v jednotlivých krajoch ju riadia *krajské komisie OI*. Na jednotlivých školách ju zaisťujú učitelia informatiky. Celoštátne kolo OI, tlač materiálov a ich distribúciu po organizačnej stránke zabezpečuje NIVAM v tesnej súčinnosti so Slovenskou komisiou OI.

A-I-1 Semafory

*Toto je **praktická úloha**. Pomocou webového rozhrania odovzdaj **funkčný, odladený program**.*

V Absurdistane je n miest, medzi ktorými sa dá cestovať pomocou m jednosmerných ciest. Každá cesta má svoju dĺžku, ktorá udáva, ako dlho ňou trvá prejsť z východzieho mesta do cieľového mesta. Avšak, cestovať sa nedá len tak ľubovoľne: na začiatku každej jednosmernej cesty je umiestnený *semafor*, na ktorom striedavo svieti zelená a červená. Na cestu i smieme vojsť jedine vtedy, keď na jej semafore svieti zelená. V opačnom prípade musíme počkať, kým sa semafor prepne na zelenú.

Každý semafor má štyri parametre:

- $z_i > 0$: ako dlho na semafore svieti zelená.
- $c_i \geq 0$: ako dlho na semafore svieti červená. Ak $c_i = 0$, tak na semafore svieti vždy len zelená.
- $s_i \in \{Z, C\}$, $f_i > 0$: dva parametre udávajúce počiatočný stav semafora. Prvý z nich (s_i) hovorí, aké svetlo svieti na semafore na začiatku. Druhý z nich (f_i) hovorí, ako dlho ešte bude toto svetlo svietiť, kým sa prepne na druhú farbu. Ak na začiatku svieti zelená, tak $f_i \leq z_i$; ak na začiatku svieti červená, tak $f_i \leq c_i$. Ak $c_i = 0$, tak semafor nemôže začínať na červenej – zaručene teda bude platiť $s_i = Z$.

Napr. ak $z_i = 5$, $c_i = 2$, na začiatku svieti zelená a $f_i = 1$, tak sa na semafore budú striedať svetlá nasledovne:

1. Najprv bude zelená svietiť ešte f_i sekúnd, teda od času 0 (vrátane) po čas $0 + f_i = 1$ (bez).
   (V čase 0.998 teda ešte svieti zelená, ale v čase 1 už práve naskočila červená a na cestu vôjsť nesmieme.)
2. Potom bude svietiť červená od času 1 (vrátane) po čas $1 + c_i = 3$ (bez).
3. Potom zelená od času 3 (vrátane) po čas $3 + z_i = 8$ (bez).
4. Potom červená od času 8 (vrátane) po čas 10 (bez).
5. Potom opäť zelená od času 10 (vrátane) po 15 (bez).
6. Potom opäť červená od času 15 (vrátane) po čas 17 (bez).
7. ...

Ak sa teda pozeráme len na celočíselné časy, môžeme na túto cestu vôjsť v čase 0, potom v časoch $3, 4, 5, 6, 7$, potom v časoch $10, 11, 12, 13, 14$, atď.

Máte popis celej cestnej siete v Absurdistane: počet miest n, počet jednosmerných ciest m, a pre každú z jednotlivých ciest i čas potrebný na jej prejdenie a parametre semafora na vstupe do nej. V čase 0 sa nachádzate v meste 1 a chcete sa čo najrýchlejšie dostať do mesta n. Zistite, kedy najskôr vieme byť v meste n.

Formát vstupu a výstupu

V prvom riadku vstupu dostanete dve celé čísla n, m: počet miest a počet ciest. Nasleduje m riadkov, i-ty z nich popisuje cestu i a pozostáva zo siedmich medzerou oddelených údajov (šiestich celých čísel a jedného znaku) v nasledovnom poradí:

1. a_i: Z ktorého mesta cesta vychádza. Mestá sú číslované $1, 2, \ldots, n$.
2. b_i: Do ktorého mesta cesta vchádza.
3. t_i: Ako dlho trvá prejazd touto cestou. Platí $t_i \geq 1$.
4. z_i, c_i, s_i, f_i: Parametre semafora, s rovnakým významom a obmedzeniami, ako sú uvedené vyššie.
   Ak $s_i = \mathrm{Z}$, tak na začiatku na semafore svieti zelená; inak $s_i = \mathrm{C}$ a na začiatku svieti červená.

Medzi dvomi mestami môže existovať viacero rôznych ciest. Cesty vždy končia v inom meste, ako v tom, v ktorom začínali.

Na jediný riadok výstupu vypíšte jedno číslo: najmenší možný čas, kedy vieme byť v meste n, ak sa v čase 0 nachádzame v meste 1. Toto číslo môže byť veľké; dajte si pozor, aby ste na jeho reprezentáciu použili číselný typ s dostatočne veľkým rozsahom (napr. `long long` v C++). Ak sa z mesta 1 nedá dostať do mesta n, vypíšte namiesto toho -1.

Obmedzenia a hodnotenie

Vo všetkých vstupoch platí $n \leq 100\,000$, $m \leq 200\,000$ a $t_i, z_i, c_i \leq 10^9$. Vstupy sú rozdelené do niekoľkých testovacích sád, s rôznymi dodatočnými obmedzeniami, ktoré menia obtiažnosť úlohy. Body za každú sadu dostanete, ak váš program správne vyrieši všetky vstupy, ktoré do nej patria.

sada	body	dodatočné obmedzenia
1	2	pre všetky cesty i platí $z_i = 1$, $c_i = 1$ a $t_i \leq 10$
2	1	pre všetky cesty i platí $z_i + c_i = 1\,000$, $n, m \leq 5\,000$ a $t_i = 1$
3	1	pre všetky cesty i platí $z_i + c_i \leq 8$, $n, m \leq 5\,000$ a $t_i = 1$
4	2	všetky semafory svietia vždy na zelenú ($c_i = 0$)
5	2	všetky semafory majú rovnaké parametre z_i, c_i, s_i, f_i a v čase 0 na všetkých semaforoch práve naskočila zelená ($s_i = Z$, $f_i = z_i$)
6	2	(žiadne dodatočné obmedzenia)

Príklady

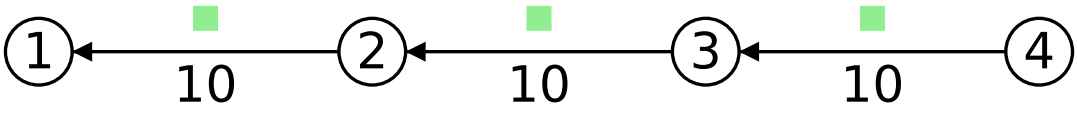
vstup

```
4 3
4 3 10 1 0 Z 1
3 2 10 1 0 Z 1
2 1 10 1 0 Z 1
```

výstup

```
-1
```

Cesty sú jednosmerné. Vieme sa síce dostať z mesta 4 *do mesta* 1*, naopak to ale nejde.*



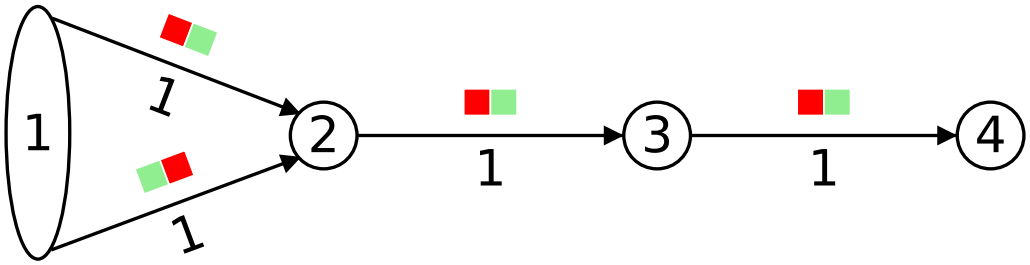
vstup

```
4 4
1 2 1 1 1 C 1
1 2 1 1 1 Z 1
2 3 1 1 1 C 1
3 4 1 1 1 C 1
```

výstup

```
4
```

Jediná možná cesta je $1 \to 2 \to 3 \to 4$*. Z mesta* 1 *vieme vyraziť do* 2 *hneď, lebo na jednej z dvoch ciest* $1 \to 2$ *máme zelenú. Do mesta* 2 *prídeme v čase* 1*, čo je super, lebo akurát sa preplo z červenej na zelenú, takže môžeme tiež hneď vyraziť. Do mesta* 3 *prídeme v čase* 2*, kedy sa akurát semafor prepol zo zelenej na červenú. Musíme preto počkať* 1 *jednotku času, kým sa semafor prepne naspäť na zelenú, až potom môžeme vyraziť. Dokopy sme* 3 *jednotky času cestovali a* 1 *jednotku času čakali.*



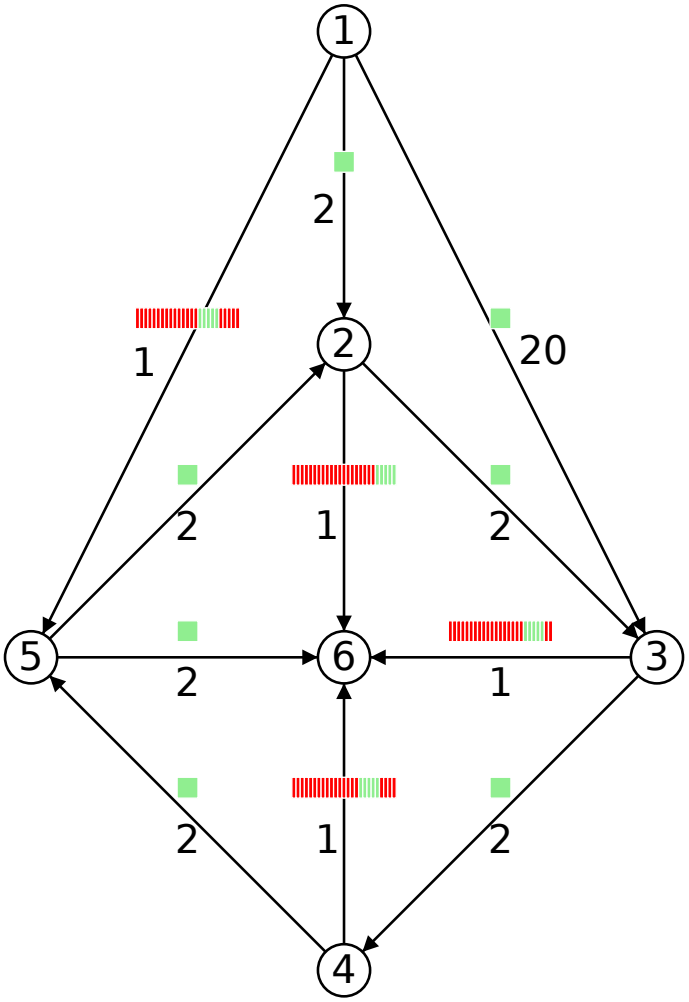
vstup

```
6 11
1 2 2 1 0 Z 1
1 3 20 1 0 Z 1
1 5 1 5 20 C 15
2 3 2 1 0 Z 1
3 4 2 1 0 Z 1
4 5 2 1 0 Z 1
5 2 2 1 0 Z 1
2 6 1 5 20 C 20
3 6 1 5 20 C 18
4 6 1 5 20 C 16
5 6 2 1 0 Z 1
```

výstup

```
10
```

Jedna možná trasa s trvaním 10 *je* $1 \to 2 \to 3 \to 4 \to 5 \to 6$. *Ak by sme do mesta* 6 *chceli ísť z iného mesta ako* 5, *tak by sme museli čakať na červenej aspoň do času* 16, *a* $16 > 10$. *Takže určite je najlepšie to zakončiť* $5 \to 6$. *No a ako najrýchlejšie sa dá dostať do* 5*? Okrem vyššie uvedenej trasy máme len možnosti* $1 \to 3 \to 4 \to 5$ *(ale* $1 \to 3$ *je dlhá cesta) alebo rovno* $1 \to 5$ *(ale vtedy dlho čakáme na semafore). Teda nič nie je lepšie ako vyššie uvedená trasa.*



V testovači je ešte aj štvrtý ukážkový vstup. Tento si viete z práve uvedeného tretieho príkladu vyrobiť tak, že ceste $1 \to 5$ *zmeníte hodnotu* f_i *z* 15 *na* 6. *V takto upravenom vstupe je optimálnym riešením počkať v meste* 1 *kým na tomto semafore naskočí zelená (teda do času* 6*), potom prejsť do mesta* 5 *a odtiaľ do mesta* 6. *Takto sa do cieľa dostaneme už v čase* 9.

A-I-2 Pokazený robot

*Toto je **praktická úloha**. Pomocou webového rozhrania odovzdaj **funkčný, odladený program**.*

Na dlhú rovnú cestu vedúcu zo západu na východ sme postavili robota. Robotovi sme zadali postupnosť n príkazov hovoriacich, ako sa má hýbať. Každý príkaz sa skladal zo smeru (`V` pre východ alebo `Z` pre západ) a vzdialenosti (celé číslo v metroch).

Robot je však starý a trochu pokazený a vykonal niečo trochu iné ako to, čo sme od neho chceli. Presnejšie, vieme, že nastali nasledovné odchýlky od zadaného programu:

- Robot nemal dosť energie, preto nevykonal celý program, len niektorých p **po sebe idúcich** príkazov.
- Občas mu zblbnú senzory a tak niektoré príkazy vykoná presne opačne – teda prejde správnu vzdialenosť, len opačným smerom ako sme chceli. Takýchto chýb nastalo počas vykonávania príkazov nanajvýš k.

Napíš program, ktorý zo zadaných n príkazov a čísel p a k vypočíta, ako najďalej od svojej začiatočnej pozície mohol tento robot byť, keď dokončil vykonávanie príkazov.

Formát vstupu a výstupu

V prvom riadku vstupu sú čísla n, p a k udávajúce počet zadaných príkazov, počet vykonaných príkazov a maximálny počet chýb počas vykonávania. Zvyšok vstupu tvorí n riadkov, v i-tom z nich je znak s_i udávajúci smer a kladné celé číslo d_i udávajúce vzdialenosť i-teho príkazu.

Na výstup vypíšte jeden riadok s jedným celým číslom: maximálnou možnou vzdialenosťou medzi pozíciami robota pred vykonaním príkazov a po vykonaní príkazov. (Upozorňujeme, že táto hodnota sa nemusí zmestiť do 32-bitovej celočíselnej premennej.)

Obmedzenia a hodnotenie

Vo všetkých vstupoch platí $0 \leq k \leq p \leq n \leq 200\,000$.

Vo všetkých vstupoch pre každé i platí, že s_i je znak `V` alebo `Z`, a že $1 \leq d_i \leq 10^9$.

Vstupy sú rozdelené do niekoľkých sád. Za každú sadu dostaneš body len vtedy, ak tvoj program všetky vstupy v nej správne a dostatočne efektívne vyrieši. Popisy sád sú v tabuľke.

sada	body	dodatočné obmedzenia
1	2	$k = 0$
2	2	$n \leq 1000$ a $p \leq 10$
3	2	$n \leq 1000$
4	2	$n \leq 70\,000$
5	2	(žiadne dodatočné obmedzenia)

Príklady

vstup	výstup

`6 4 0`
`Z 30`
`V 10`
`V 25`
`Z 5`
`V 10`
`Z 30`
	`40`

Tento robot nerobí chyby. Ak vykonal prvé alebo posledné štyri príkazy, skončil tam, kde začínal. Ak vykonal druhý až piaty príkaz, skončil 40 m východne od miesta, kde začínal.

vstup	výstup

`6 4 2`
`V 30`
`Z 10`
`Z 15`
`Z 5`
`Z 10`
`V 30`
	`60`

Jedno optimálne riešenie je že robot vykonal prvé štyri inštrukcie, ale pri prvej z nich spravil chybu a išiel na západ.

vstup	výstup

`6 6 6`
`Z 1`
`V 2`
`V 3`
`Z 4`
`V 5`
`Z 6`
	`21`

A-I-3 Piškóty

*Toto je **teoretická úloha**. Pomocou webového rozhrania odovzdaj **súbor vo formáte PDF**, obsahujúci riešenie, spĺňajúce požiadavky uvedené v pravidlách.*

Emmika má $n \leq 10^5$ polomáčaných piškót. Poukladala ich všetky na stôl do jedného dlhého radu. Každá piškóta je otočená hore buď čokoládovou alebo piškótovou stranou.

Emmika by samozrejme chcela všetky piškóty zjesť. Aby si jedenie piškót spestrila, vymyslela si k tomu nasledovné pravidlá. Zjesť smie len piškótu, ktorá je otočená čokoládovou stranou hore. A aby aj ostatné piškóty mali šancu byť zjedené, vždy, keď zje nejakú piškótu, obráti jej *priamych susedov* hore nohami. Priamymi susedmi myslíme tie piškóty, vedľa ktorých táto piškóta ležala na začiatku. Teda ak dve piškóty nesusedia na začiatku, nestanú sa susedmi ani keď Emmika zje všetky piškóty ležiace medzi nimi.

Emmiku by zaujímalo, či a ako vie zjesť úplne všetky piškóty.

Príklad

Predstavme si, že Emmika začala s troma piškótami, pričom len stredná bola otočená čokoládou hore. V tomto prípade nemá na výber: ako prvú musí zjesť strednú piškótu. Keď tak spraví, obe susedné piškóty obráti z piškótovej-strany-hore na čokoládovú-stranu-hore. Následne ich postupne (v ľubovoľnom poradí) môže tiež zjesť. Keďže tieto dve piškóty nie su priamy susedia, po zjedení jednej z nich druhú Emmika neobráti.

Jedna možná postupnosť jedenia je na nasledujúcom obrázku (po riadkoch zhora dole):

Súťažná úloha

Nasleduje niekoľko podúloh. Každú môžete riešiť samostatne, vďaka čomu viete získať čiastočné body.

Riešenia by mali pri každej podúlohe uvádzať *dôkaz správnosti*, nezabudnite preto všetky svoje tvrdenia poriadne odôvodniť.

Pre účely tejto úlohy si piškóty ktoré sú hore čokoládovou stranou, zaznačíme ako „`C`“, a tie hore piškótovou stranou ako „`P`“.

a) (1 bod) Predstavme si, že sú piškóty na začiatku v rade uložené nasledovne: „`CCCCCCC`“ (teda 7 piškót otočených čokoládou hore). Vie ich Emmika zjesť všetky? Ak áno, ukážte ako. Ak nie, zdôvodnite prečo.

b) (1 bod) Predstavme si, že sú piškóty na začiatku v rade uložené nasledovne: „`CPPPPPC`“. Vie ich Emmika zjesť všetky? Ak áno, ukážte ako. Ak nie, zdôvodnite prečo.

c) (5 bodov) Navrhnite algoritmus, ktorý na vstupe dostane reťazec písmen „`C`“ a „`P`“ reprezentujúci piškóty v rade a rozhodne, či existuje poradie, v akom ich Emmika vie zjesť všetky (pri dodržaní vyššie uvedených pravidiel).

   Nezabudnite poriadne dokázať, že váš algoritmus je správny. (Takýto dôkaz sa napríklad môže skladať z postupu, ako niektoré rady piškót celé zjesť, a zo zdôvodnenia, prečo sa žiaden iný rad piškót celý zjesť nedá.)

Za vyriešenie tejto podúlohy pre špeciálny prípad, že všetky piškóty sú na začiatku otočené čokoládovou stranou hore (reťazec tvoria samé „`C`“) viete získať až 2 body.

d) (3 body) Kým Emmika nedávala pozor, zjedla jej niektoré piškóty mačka. Situáciu na stole si teraz vieme popísať reťazcom znakov „`C`“, „`P`“ a „`-`“ (prázdne miesto po piškóte).

Piškótu x na stole nazveme *pekná*, ak platí, že ju môže Emmika zjesť ako prvú – teda existuje taký postup, ktorým vie Emmika zjesť všetky zostávajúce piškóty, a piškótu x zje ako prvú.

Navrhnite algoritmus, ktorý spočíta, koľko je na stole pekných piškót. (Ak Emmika nevie zjesť všetky piškóty na stole, odpoveď je 0.)

vstup	výstup
7 CCC-CP	3

Na stole sú najskôr tri piškóty čokoládovou stranou hore, potom medzera, a potom ďalšie dve piškóty. Emmika ich vie všetky zjesť. Aby to dosiahla, musí začať prvou, treťou, alebo piatou (v originálnom poradí) piškótou. Všimnite si, že druhá piškóta je síce čokoládovou stranou hore, ale ak ju Emmika zje ako prvú, nebude vedieť dojesť ostatné piškóty.

vstup	výstup
3 C-P	0

V tomto prípade Emmika nevie zjesť obe piškóty – zjedenie prvej neotočí druhú.

A-I-4 Nechaj to na solver

K tejto úlohe patrí študijný text uvedený na nasledujúcich stranách. Najskôr si prečítaj ten, až potom sa pusti do riešenia samotných súťažných úloh. Aj v nasledujúcich kolách tohto ročníka OI stretneš ten istý študijný text a úlohy, ktoré naň budú nadväzovať.

Táto úloha má teoretické aj praktické časti. Odovzdať treba ***jeden súbor vo formáte PDF*** *obsahujúci programy, ich slovné popisy a optimálne riešenia pre zverejnené testovacie vstupy.*

Testovacie vstupy k tejto úlohe sa dajú stiahnuť zo stránky oi.sk tu: verzia pre Windows, verzia pre Linux.

Podúlohy A-E: investície

Máme e eur. Existuje n projektov (očíslovaných od 1 po n), do ktorých ich vieme investovať. Každý projekt buď podporíme alebo nie. O každom projekte vieme, akou sumou s_i ho treba podporiť a akú hodnotu v_i časom dostaneme späť ako výnos, ak ho podporíme.

Medzi projektmi existujú konflikty: niektoré dvojice projektov si priamo konkurujú a protimonopolný úrad vydal nariadenie, že z každej takej dvojice projektov smieme podporiť najviac jeden. Existuje k konfliktov. Pre každý z nich poznáme čísla $c_{i,1}$ a $c_{i,2}$ projektov, ktoré nesmieme oba podporiť.

Medzi projektmi tiež existujú závislosti: napr. projekt na pestovanie bio kapusty sa nevie uskutočniť ak nebude podporený projekt na montáž efektívneho zavlažovania. Takže ak chceme podporiť kapustu, musíme podporiť aj zavlažovanie. Závislostí je z. Pre každú závislosť sú dané dve čísla projektov: ak chceme podporiť projekt $d_{i,1}$, musíme podporiť aj projekt $d_{i,2}$. (Inými slovami, táto závislosť nám zakazuje podporiť $d_{i,1}$ a zároveň nepodporiť $d_{i,2}$.) Závislosti môžu byť úplne ľubovoľné. Špeciálne, ak máme dve závislosti hovoriace, že x závisí na y a že y závisí na x, znamená to, že môžeme buď podporiť oba tieto projekty alebo ani jeden z nich.

Údaje o takýchto projektoch uložíme do textového súboru nasledovne:

- jeden riadok: kapitál a počet projektov — kladné celé čísla e a n
- jeden riadok: sumy potrebné na podporu — kladné celé čísla $s_1, \ldots, s_n$
- jeden riadok: výnosy — kladné celé čísla $v_1, \ldots, v_n$
- jeden riadok: počet konfliktov — nezáporné celé číslo k
- k riadkov: popis jednotlivých konfliktov — v i-tom z týchto riadkov sú čísla projektov $c_{i,1}$ a $c_{i,2}$
- jeden riadok: počet závislostí — nezáporné celé číslo z
- z riadkov: popis jednotlivých závislostí — v i-tom z týchto riadkov sú čísla projektov $d_{i,1}$ a $d_{i,2}$

A) Predpokladajme, že nie sú žiadne konflikty ani závislosti. Zostroj ILP, ktorého optimálne riešenie nám povie, ktoré projekty máme podporiť, ak chceme mať na konci (keď dostaneme výnosy) čo najviac peňazí.

B) Uprav ILP z predchádzajúcej podúlohy tak, aby bral do úvahy konflikty medzi projektmi.

C) Uprav ILP z podúlohy a) alebo b) tak, aby bral do úvahy závislosti medzi projektmi.
(Túto podúlohu teda môžeš riešiť aj ak nevieš riešiť podúlohu b).)

D) V súbore `D.txt` je testovací vstup vo vyššie uvedenom formáte. V tomto súbore platí $k = z = 0$, teda nemáme žiadne konflikty ani závislosti medzi projektmi. Nájdi optimálnu sadu projektov, ktoré podporiť. Ako riešenie odovzdaj celkovú sumu peňazí, ktorú budeme mať na konci, a čísla podporených projektov.

E) Aj pre súbor `E.txt` nájdi a odovzdaj jeho optimálne riešenie.

Príklad

vstup

```
100000 4
50000 50000 20000 40000
100000 95000 26000 900000
1
1 2
2
4 3
3 2
```

výstup

```
Optimálne riešenie: 151000
Podpor projekty: 2 3
```

Projekt 4 nevieme podporiť, lebo by sme museli podporiť aj projekty 2 a 3 a na to nemáme dosť peňazí. Projekt 1 sa neoplatí podporiť, lebo potom kvôli konfliktu nevieme podporiť projekt 2 a následne kvôli nesplnenej závislosti ani projekt 3.

Podúlohy F-J: chovatelia prasiat

V krajine je s sýpok a c chovateľov prasiat. Sýpky sú očíslované od 1 po s a chovatelia od 1 po c.

Na jeseň môžeme zbierať bukvice a žalude do sýpok. Sýpka i má kapacitu na k_i kg plodov a túto kapacitu vieme medzi bukvice a žalude rozdeliť ľubovoľne. Oboch typov plodov sa dá v okolitých lesoch nazbierať ľubovoľne veľa, ale za povolenie na zber plodov musíme zaplatiť jednorázový poplatok 5000 eur. (Jedno povolenie stačí na nazbieranie ľubovoľne veľa plodov do všetkých sýpok.)

V zime budú chovatelia chcieť u nás nakúpiť krmivo. To sa predáva v 1 kg sáčkoch. Chovateľ j je ochotný kúpiť nanajvýš $g_{j,1}$ sáčkov bukvíc a nanajvýš $g_{j,2}$ sáčkov žaluďov. Za každý sáčok bukvíc je ochotný zaplatiť $h_{j,1}$ a za každý sáčok žaluďov $h_{j,2}$ eur. Tieto objednávky poznáme už pred zberom plodín.

Sáčky s plodmi chovateľom pošleme kuriérom. Kuriér je ochotný *z každej sýpky každému chovateľovi* zobrať jednu zásielku vážiacu nanajvýš w kg. Každému chovateľovi teda vieme poslať dokopy nanajvýš $s \cdot w$ sáčkov.

Cena zásielky závisí od odosielateľa (sýpku) a adresáta (chovateľa), ale vždy je priamo úmerná hmotnosti zásielky: za každý kg plodov poslaných zo sýpky i chovateľovi j zaplatíme $\ell_{i,j}$ eur.

Všetky vyššie uvedené číselné údaje sú kladné celé čísla. Do textového súboru ich uložíme nasledovne:

- jeden riadok: počty objektov — čísla s a c
- jeden riadok: kapacity sýpok — čísla $k_1, \ldots, k_s$
- c riadkov: popis chovateľov — v j-tom z týchto riadkov sú čísla $g_{j,1}$, $g_{j,2}$, $h_{j,1}$ a $h_{j,2}$
- jeden riadok: maximálna hmotnosť zásielky — číslo w
- s riadkov: ceny transportu zo sýpok ku chovateľom — v i-tom z týchto riadkov sú čísla $\ell_{i,1}, \ldots, \ell_{i,c}$

F) Predpokladaj, že všetci chovatelia chcú kupovať len bukvice (t.j. všetky $g_{j,2} = 0$). Zostroj ILP, ktorého optimálne riešenie bude zodpovedať najväčšiemu možnému zisku, ktorý vieme dosiahnuť optimálnym zberom, distribúciou a predajom krmiva.

G) Uprav ILP z riešenia podúlohy F tak, aby optimálne vyriešil celú všeobecnú úlohu.

H) Nájdi a odovzdaj optimálne riešenie pre testovací vstup v súbore `H.txt`. V tomto súbore platí, že existuje len jedna sýpka ($s = 1$). Stačí nájsť optimálny čistý zisk, netreba rozpisovať, koľko čoho odkiaľ kam voziť.

I) Nájdi a odovzdaj optimálne riešenie pre testovací vstup v súbore `I.txt`. V tomto súbore platí, že nik nechce kupovať žalude (všetky $g_{j,2} = 0$).

J) Nájdi a odovzdaj optimálne riešenie pre testovací vstup v súbore `J.txt`.

Príklad

vstup

```
2 3
100 200
1000 1000 10 10
120 70 100 30
20 30 50 60
80
9 15 23
11 28 12
```

výstup

```
Optimálny zisk: 6980
sýpka 1: bukvice 60 žalude 40
sýpka 2: bukvice 100 žalude 30
balík s1->ch1: bukvice 20
balík s1->ch2: bukvice 40 žalude 40
balík s2->ch2: bukvice 80
balík s2->ch3: bukvice 20 žalude 30
```

Odovzdávanie a hodnotenie

Riešenia teoretických podúloh môžeš odovzdať v dvoch podobách:

- Buď ako slovný popis toho, ako z konkrétnych vstupných hodnôt (e, n, s_i, v_i, atď.) vyrobiť ILP, ktorého optimálne riešenie zodpovedá optimálnemu riešeniu vyššie definovaného optimalizačného problému.
- Alebo ako okomentovaný program v bežnom programovacom jazyku, ktorý na vstupe načíta konkrétne vstupné hodnoty a vyrobí z nich (napr. vypíše na výstup) im zodpovedajúci ILP.

Za každú správne vyriešenú podúlohu je bod. Body za správne vyriešené vstupy dostaneš bez ohľadu na techniku, ktorou ich riešenie získaš (teda aj ak pri ich výpočte vôbec nepoužiješ ILP).

Študijný text: Celočíselné lineárne programovanie

V tomto ročníku Olympiády sa budeme pozerať na optimalizačné problémy – teda na problémy, ktoré majú veľa rôznych riešení a našou úlohou je nájsť to najlepšie z nich. Napríklad nás môže zaujímať:

- Ako najlacnejšie precestovať všetky mestá na Slovensku?
- Do najmenej koľkých škatúľ viem pri sťahovaní zbaliť všetky svoje knihy?
- Akú veľkosť má najväčšia podmnožina riešiteľov tohto ročníka OI, v ktorej sa všetci navzájom poznajú?

Mnohé optimalizačné problémy majú jednu spoločnú nepríjemnú vlastnosť: nepoznáme pre ne žiadne efektívne algoritmické riešenie. Empiricky si dovolíme tvrdiť, že do tejto smutnej kategórie patrí značná väčšina optimalizačných problémov, ktoré stretneme všelikde v praxi – či už v počítačoch (napr. scheduling procesov, routing paketov v sieťach) alebo v „reálnom živote“ (napr. logistika všetkého druhu, optimalizácia nákladov, či rôzne problémy v bioinformatike). A mimochodom, patria sem aj všetky tri vyššie uvedené problémy.

Situácia je ešte o čosi horšia. Nielen, že k týmto úlohám nepoznáme žiaden algoritmus, ktorý by ich riešil efektívne (teda v časovej zložitosti polynomiálnej od veľkosti vstupu), my dokonca máme aj veľmi dobré dôvody domnievať sa, že takýto algoritmus ani neexistuje. Toto celé súvisí s jednou z najdôležitejších otvorených otázok súčasnej informatiky: otázkou, či sa P rovná NP. Veľmi zjednodušene povedané, ide o otázku, či každú úlohu, v ktorej vieme efektívne skontrolovať riešenie, vieme aj efektívne vyriešiť. Intuitívne sa väčšine vedcov zdá, že to skôr nebude pravda – porovnajte si napríklad, ako ťažké môže byť ručne vyriešiť čo i len obyčajné sudoku, a ako ľahké je pre ľubovoľné sudoku skontrolovať, či je vyriešené správne. Na tomto príklade si tiež môžeme uvedomiť, že znalosť postupu kontroly správnosti riešenia nám vo všeobecnosti nič nepovie o tom, ako nejaké riešenie efektívne hľadať.

Ale to je nám v praxi vlastne jedno. V situácii, kedy pre našu ťažkú úlohu neexistuje žiaden efektívny algoritmus, sme na tom presne rovnako ako v situácii, kedy existuje, ale nepoznáme ho. Ak potrebujeme optimálne vyriešiť nejaký vstup, sme tak či onak odkázaní na hrubú silu, čiže na prezretie všetkých možností.

Ani riešenia hrubou silou však nie sú všetky rovnocenné. Často takéto riešenie vieme zefektívniť tak, že nebudeme prezerať úplne všetky možnosti, ale šikovne vynecháme čo najviac častí prehľadávania, ktoré k najlepšiemu riešeniu nevedú. Pre mnohé optimalizačné problémy sme takto vyvinuli konkrétne algoritmy, ktoré síce nie sú efektívne (ich časová zložitosť je naďalej exponenciálna od veľkosti vstupu), ale vďaka vhodnému „orezaniu“ prehľadávania vedia v rozumnom čase vyriešiť omnoho väčšie vstupy ako priamočiare riešenie skúšajúce všetky možnosti.

Tu sa však niektoré múdre hlavy zamysleli a uvedomili si: v mnohých týchto jednotlivých algoritmoch robíme veľmi podobne vyzerajúce optimalizácie. Nevedeli by sme to zovšeobecniť? V tomto ročníku Olympiády sa budeme zaoberať jednou z kladných odpovedí na túto otázku.

Celočíselné lineárne programovanie (po anglicky integer linear programming[1]) je spôsob, ako matematicky popísať niektoré optimalizačné problémy. ILP je dobré v tom, že niekto už za nás spravil všetku tú skutočne ťažkú prácu – v súčasnej dobe už existuje viacero veľmi kvalitne optimalizovaných *solverov,*[2] ktoré vedia z takéhoto matematického popisu nájsť optimálne riešenie popísanej úlohy. A navyše často platí, že vďaka všeobecným optimalizáciám to takýto solver spraví efektívnejšie, ako keby sme si sami písali a následne sami vylepšovali špecializovaný algoritmus pre našu konkrétnu úlohu. Vďaka tomu dostávame nový spôsob, ako riešiť ťažké problémy: namiesto implementácie celého vlastného riešenia sa môžeme zamyslieť nad tým, či a ako tento problém vieme zapísať ako ILP. Ak sa nám to podarí, môžeme potom na riešenie nášho problému použiť ILP solver. A presne toto budete robiť pri riešení súťažných úloh v tomto ročníku Olympiády.

Formálna definícia ILP

V celom ďalšom texte bude slovo *konštanta* označovať ľubovoľné konkrétne (**možno aj záporné**) **celé** číslo a slovo *premenná* označovať neznámu, ktorá môže nadobúdať ľubovoľnú **nezápornú celočíselnú** hodnotu.

Celočíselný lineárny program (v základnej, tzv. kanonickej podobe) sa skladá z nasledujúcich častí:

[1] Pre techniku ako celok aj pre jednotlivé celočíselné lineárne programy budeme v ďalšom texte používať anglickú skratku ILP.
[2] Pojem „solver“ sme sa rozhodli neprekladať, „riešiteľ“ je človek a „riešič“ znie divne :)

Obmedzenia: Sada lineárnych nerovníc, z ktorých i-ta má tvar $a_{i,1} \cdot x_1 + \cdots + a_{i,n} \cdot x_n \leq b_i$, pričom všetky $a_{i,j}$ aj b_i sú konštanty. Tieto obmedzenia musia byť **všetky** dodržané.

Cieľ: lineárny výraz tvaru $c_1 \cdot x_1 + \cdots + c_n \cdot x_n$, kde c_i sú konštanty a x_i premenné.

Každé priradenie hodnôt premenným, pre ktoré sú splnené všetky obmedzenia, budeme volať *platné* riešenie. Tie platné riešenia, pre ktoré **cieľový výraz nadobúda najväčšiu možnú hodnotu**, budeme volať *optimálne*.

Existujú samozrejme aj ILP, ktoré nemajú žiadne optimálne riešenie. To môže mať dva dôvody: buď sú *nesplniteľné* (napr. máme obmedzenia $x_1 \leq 7$ a $-x_1 \leq -8$, čiže $x_1 \geq 8$) alebo sú *neohraničené* (napr. nemáme žiadne obmedzenia a chceme maximalizovať hodnotu $x_1 + 2x_2$).

Voľnejšia, praktickejšia definícia ILP

Programy, ktoré budeme neskôr písať my, budú o niečo všeobecnejšie:

- Dovolíme aj programy, v ktorých je cieľom minimalizovať hodnotu konkrétneho výrazu a tento výraz môže navyše obsahovať aj sčítanec, ktorý je len konštanta.
- Dovolíme aj obmedzenia, v ktorých je namiesto znamienka $\leq$ znamienko $\geq$ alebo $=$.
- V obmedzeniach môžeme robiť všetky štandardné aritmetické úpravy, napr. vynechávať sčítance tvaru $0 \cdot x_i$, ľubovoľne prehadzovať sčítance medzi ľavou a pravou stranou a vhodne používať zátvorky.

Rozmyslite si, že všetky tieto zmeny slúžia len k lepšej čitateľnosti našich programov: totiž napr. minimalizovať $x + 3y + 1000$ je to isté ako maximalizovať $-x - 3y$, obmedzenie $2x - 6y \geq y - 13$ je len ináč zapísaný výraz $-2x + 7y \leq 13$, a obmezenie $2x = 5y$ je to isté ako dve obmedzenia $2x \leq 5y$ a $2x \geq 5y$.

Príklad: kuracie nugetky

Zadanie: Stánok predáva tri rôzne balenia kuracích nugetiek: 6 ks za 2 eurá, 9 ks za 2,90 alebo 20 ks za 6,10. Koľko najviac nugetiek vieme dostať za 32 eur?

Nesprávne pažravé riešenie: Keď si pre každé balenie spočítame, koľko zaplatíme za jednu nugetku, najlepšie vychádza to najväčšie. Za 32 eur môžeme nakúpiť 5 najväčších balení, čím dostaneme 100 nugetiek. Toto ale nie je optimálne riešenie – existuje iný spôsob ako využiť peniaze, ktoré máme, a skončiť s viac nugetkami. (Všimnite si, že pri tomto riešení nám okrem 100 nugetiek ostalo nevyužitých 1,50 eura, za ktoré si už nič nevieme kúpiť.)

Táto úloha sa vo všeobecnosti nedá riešiť pažravo. Náš príklad s nugetkami je špeciálnym prípadom známeho typu optimalizačných úloh známych pod spoločným názvom *problém batoha* (anglicky: knapsack). Pre malé vstupy vieme optimálne riešenia nájsť pomocou dynamického programovania, ale vo všeobecnosti je riešenie tohto problému ťažké.

Lineárny program: Označme si x_1 počet malých, x_2 počet stredných a x_3 počet veľkých balení, ktoré kúpime. Naším *cieľom* je maximalizovať celkový počet nugetiek, ktoré kúpime, teda hodnotu $6x_1 + 9x_2 + 20x_3$. Dodržať pritom musíme *obmedzenie*, že celková cena za nákup nesmie prekročiť náš rozpočet, teda (v centoch, aby všetko boli celé čísla) musí platiť: $200x_1 + 290x_2 + 610x_3 \leq 3200$.

Praktické riešenie: Náš lineárny program sa v syntaxi, ktorej rozumie solver `lp_solve`, zapíše nasledovne:

```
max: 6x_1 + 9x_2 + 20x_3;
200x_1 + 290x_2 + 610x_3 <= 3200;
int x_1, x_2, x_3;
```

Keď zadáme `lp_solve` vyriešiť tento program, dostaneme na výstupe nasledovné:

```
Value of objective function: 102.00000000

Actual values of the variables:
```

```
x_1                    1
x_2                    4
x_3                    3
```

Dozvedeli sme sa teda, že najviac vieme získať až 102 nugetiek, a to tak, že kúpime 1 malé, 4 stredné a 3 veľké balenia. Celková cena nákupu je 31,90, na konci teda budeme mať 102 nugetiek a nepoužitých 10 centov.

Vyber si solver

My sme si pre tento študijný text vybrali jeden konkrétny solver: `lp_solve`. V riešeniach príkladov používame syntax, ktorej tento solver rozumie.

Na stránke https://oi.sk/apps/ilp/ nájdeš niekoľko rôznych návodov, aký solver si vybrať a ako ho použiť na praktické riešenie ILP podľa toho, aký OS a programovací jazyk preferuješ. V domácom kole si taktiež môžeš nájsť na internete nejaký iný solver a použiť ten, ak sa ti náš výber nebude páčiť.

Príklad: sudoku

Občas nás namiesto optimalizácie (nájdi *najlepšie* riešenie spomedzi mnohých) môže zaujímať jednoducho nájdenie úplne ľubovoľného platného riešenia, resp. rozhodnutie, či vôbec nejaké platné riešenie existuje.

Samozrejme, aj na riešenie takýchto úloh vieme „zneužiť“ ILP solver: stačí mu nedať žiadny cieľ (alebo napr. dať maximalizovať hodnotu výrazu „0“).

Pozrime sa napríklad na známu logickú úlohu: sudoku. V tejto úlohe je cieľom vyplniť mriežku rozmerov 9×9 číslami od 1 po 9 tak, aby sa v každom riadku, stĺpci aj „veľkom“ štvorci 3×3 vyskytovalo každé z čísel 1 až 9 práve raz.

V tomto príklade si ukážeme, ako vieme pravidlá sudoku sformulovať ako ILP. Zdalo by sa, že budeme chcieť 81 premenných: pre každé políčko tabuľky premennú predstavujúcu hodnotu, ktorá na ňom má byť. A áno, aj takouto cestou sa vieme dostať ku sformulovaniu sudoku ako ILP, necháme si ju ale na neskôr. V tomto príklade sa vyberieme inou cestou: použijeme $9 \times 9 \times 9$ boolovských (t.j. logických, resp. binárnych) premenných. Premenná $x_{i,j,k}$ bude 1, ak má na súradniciach (i, j) byť hodnota k, resp. to bude 0, ak tam hodnota k byť nemá.

Pozrime sa teraz, ako môžu vyzerať všetky pravidlá sudoku zapísané ako lineárne rovnice a nerovnice.

- Na každom políčku je práve jedno číslo.
  Pre každé i a j máme podmienku $x_{i,j,1} + x_{i,j,2} + \cdots + x_{i,j,9} = 1$.
- V každom riadku sa každé číslo nachádza práve raz.
  Pre každé i a k máme podmienku $x_{i,1,k} + x_{i,2,k} + \cdots + x_{i,9,k} = 1$.
- Analogické podmienky ako pre riadky máme aj pre každý stĺpec a každý štvorec.

Ak teraz chceme vyriešiť konkrétne sudoku pomocou `lp_solve`, spravíme to nasledovne:

- Vygenerujeme (napr. jednoduchým programom, ktorý si napíšeme v bežnom programovacom jazyku) všetky vyššie uvedené obmedzenia predstavujúce všeobecné pravidlá sudoku.
- Pridáme informáciu, že všetky $x_{i,j,k}$ sú boolovské premenné. To vieme spraviť tak, že každej pridáme obmedzenie $x_{i,j,k} \leq 1$.

  Premenné nadobudajúce len hodnoty 0 a 1 sú však pri modelovaní problémov natoľko bežné, že asi každý solver bude mať špeciálnu syntax pre priame deklarovanie takýchto premenných. Napr. v `lp_solve` stačí takéto premenné namiesto ako `int` deklarovať ako `bin`.
- Pridáme obmedzenia popisujúce konkrétne zadanie, ktoré sa snažíme vyriešiť. Ak napr. v zadaní máme v prvom riadku a treťom stĺpci už predpísané číslo 7, pridáme obmedzenie $x_{1,3,7} = 1$.

Príklad: sudoku po druhé

Ako by vyzeralo modelovanie sudoku, ak by sme chceli pre každé políčko použiť premennú $v_{i,j}$, ktorej hodnota by mala priamo zodpovedať hodnote nachádzajúcej sa na príslušnom políčku? Zjavne potrebujeme obmedzenia $v_{i,j} \geq 1$ a $v_{i,j} \leq 9$. Okrem nich by už stačilo len pridať obmedzenia hovoriace, že niektoré dvojice políčok nesmú mať rovnakú hodnotu. Takýchto obmedzení budeme potrebovať celkom veľa: jednu pre každú dvojicu políčok v rovnakom riadku, v rovnakom stĺpci, aj v rovnakom štvorci 3×3. Napr. pre každé dve políčka (i, x) a (i, y) v riadku i potrebujeme obmedzenie $v_{i,x} \neq v_{i,y}$. Tu však máme problém: Toto obmedzenie nemá ani jeden z povolených tvarov, a ani ju nevieme priamočiaro vyjadriť pomocou povolených obmedzení.

Želané obmedzenie vieme zapísať ako logický *or* dvoch podmienok: má platiť *buď* $v_{i,x} < v_{i,y}$ *alebo* $v_{i,x} > v_{i,y}$. Keďže všetky $v_{i,j}$ sú celé čísla, tieto podmienky vieme upraviť do povoleného tvaru: má platiť buď $v_{i,x} \leq v_{i,y} - 1$ alebo $v_{i,x} \geq v_{i,y} + 1$.

Toto ale stále nie je OK: v ILP musia byť splnené všetky podmienky naraz. To zodpovedá logickému *and*, nie logickému *or*. Čo s tým vieme spraviť?

Pomôžeme si drobným trikom. Zavedieme novú binárnu premennú r. (Správne by sme ju mali nazvať napr. $r_{i,x,i,y}$, keďže budeme pre každú dvojicu premenných, ktoré sa nemajú rovnať, potrebovať jednu novú premennú. Pre lepšiu čitateľnosť ju ale tu budeme volať len r.) Hodnota r nám bude hovoriť, či má byť menšia prvá alebo druhá z hodnôt v. Pozrime sa teraz na nasledujúce dve obmedzenia:

$$\begin{aligned} v_{i,x} - v_{i,y} &\geq 1 - 10r \\ v_{i,y} - v_{i,x} &\geq 1 - 10(1 - r) \qquad = 10r - 9 \end{aligned}$$

Ak $r = 0$, dostávame podmienky $v_{i,x} - v_{i,y} \geq 1$ a $v_{i,y} - v_{i,x} \geq -9$. Prvá z nich hovorí $v_{i,x} > v_{i,y}$ a druhá je triviálne splnená pre ľubovoľné $v_{i,x}, v_{i,y} \in \{1, 2, \ldots, 9\}$. (Konštantu 10 sme zvolili práve tak, aby toto platilo. Využívame teda, že poznáme rozsah hodnôt, ktoré môžu $v_{i,x}$ a $v_{i,y}$ nadobúdať.)

A naopak, ak zvolíme $r = 1$, dostaneme jednu triviálne splnenú podmienku a jednu, ktorá hovorí $v_{i,x} < v_{i,y}$. Pridaním tejto novej premennej r a vyššie uvedených dvoch obmedzení sme teda dosiahli, to, čo sme chceli: pre ľubovoľné $v_{i,x} \neq v_{i,y}$ vieme obe tieto obmedzenia splniť, zatiaľ čo pre $v_{i,x} = v_{i,y}$ ich naraz obe splniť nejde.

Príklad: čo takto robiť nevieme

Máme lietadlo, ktorým chceme preletieť 1000 km z jedného letiska na druhé. Môžeme si v rámci povolených rozsahov zodpovedajúcich modelu lietadla zvoliť letovú hladinu h (výšku v km, v ktorej poletíme, v rozsahu 10 až 13 km) a rýchlosť letu v (v km/h, v rozsahu 600 až 900 km/h). Chceli by sme minimalizovať cenu letu, teda spotrebu paliva.

Toto tiež zjavne vieme zapísať pomocou vhodných vzorcov ako optimalizačný problém. Vo veľmi zjednodušenej podobe by to mohlo vyzerať napr. nasledovne: Čas letu bude $1000/v$. Ak letíme vo výške h, optimálna rýchlosť kvôli odporu vzduchu je $v_{opt}(h) = 540 + 30h$. Výkonnosť motorov je najlepšia vo výške $h = 11.5$ km. Odchýlenie od týchto parametrov zvyšuje spotrebu, ale môže nás dostať do cieľa skôr. Rýchlosť spotreby paliva (v kg/h) sa preto dá vyjadriť vzťahom $2000 + 200(h - 11.5)^2 + 0.05(v - v_{opt}(h))^2$. Celková spotreba paliva je súčinom tejto hodnoty a času letu.

Toto je síce exaktná matematická formulácia optimalizačného problému, ale je tu jeden háčik: obmedzenia pre h a v sú síce lineárne, ale funkcia, ktorej hodnotu sa snažíme optimalizovať, nie je lineárnou funkciou premenných h a v. ILP solver nám teda s takto konkrétne sformulovaným problémom nebude vedieť pomôcť.

Pre väčšiu názornosť ešte dodáme, že ani omnoho jednoduchší výraz $h \cdot v$ nie je lineárny, keďže ide o súčin dvoch premenných.

ŠTYRIDSIATY PRVÝ ROČNÍK OLYMPIÁDY V INFORMATIKE

Príprava úloh: Michal Anderle, Truc Lam Bui, Michal Forišek, Ján Hozza, Paulína Smolárová
Recenzia: Michal Forišek
Slovenská komisia Olympiády v informatike
Vydal: NIVAM – Národný inštitút vzdelávania a mládeže, Bratislava 2025