Priebeh krajského kola

Krajské kolo 41. ročníka Olympiády v informatike, kategória A, sa koná 20. 1. 2026 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 nejakom bežnom programovacom jazyku (napr. C++, Python, Java, Pascal).
- 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, najdôležitejšie dve kritériá hodnotenia sú v prvom rade **správnosť** a v druhom rade **efektívnosť** navrhnutého algoritmu. Na výslednom počte bodov sa môže prejaviť 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. Nezávisí pri tom na konštantných faktoroch, len na rádovej rýchlosti rastu tejto funkcie.

V zadaní úloh uvádzame časť „Hodnotenie“, v ktorej nájdete približné limity na veľkosť vstupných údajov. Pod pojmom „efektívne vyriešiť“ chápeme to, že váš program spustený na modernom počítači by mal dať odpoveď nanajvýš do niekoľkých sekúnd.

Údaje z tejto časti zadania by mali slúžiť hlavne na to, aby ste o riešení, ktoré vymyslíte, vedeli približne povedať, koľko bodov zaň dostanete.

A-II-1 Sizyfos a balvany

V rade vedúcom zľava doprava je n balvanov. Pozície, na ktorých stoja, sú očíslované zaradom od 0 po $n-1$. Hmotnosť balvanu, ktorý stojí na pozícii i, budeme označovať $H[i]$.
Sizyfos má za úlohu usporiadať balvany od najľahšieho po najťažší, a to tak, aby na konci stáli na presne tých istých n pozíciách, len v správnom poradí – teda aby platilo $H[0] \leq H[1] \leq H[2] \leq \cdots \leq H[n-1]$

Sizyfos si vymyslel algoritmus, pri ktorom nebude musieť prenášať žiaden balvan na veľkú vzdialenosť. Základom Sizyfovho algoritmu bude postup, ktorý si nazveme *kolo*. Kolo bude vyzerať nasledovne:

1. Na začiatku kola Sizyfos príde na začiatok radu (teda k pozícii 0).
2. Postupne pôjde popri balvanoch zľava doprava.
3. Vždy, keď stretne balvan, ktorý má bezprostredne naľavo od seba balvan ostro ťažší, tak ich medzi sebou vymení: odtiahne ťažší o pozíciu doprava a ľahší o pozíciu doľava. Formálnejšie, keď Sizyfos stojí pri pozícii i a vidí, že $H[i-1] > H[i]$, tak vymení balvany na pozíciách $i-1$ a i.
4. Kolo skončí, keď Sizyfos prejde aj okolo posledného balvanu v rade.

Sizyfos bude celý tento postup opakovať dovtedy, kým v nejakom kole nespraví žiadnu výmenu balvanov.

Podúloha A (2 body):
Dokážte, že pre ľubovoľnú postupnosť balvanov Sizyfov postup po konečnom počte kôl skončí.

Podúloha B (3 body):
Ukázalo sa, že Sizyfos nevládze pohnúť balvan, ktorého hmotnosť je ostro väčšia ako w. Upravil preto svoj postup tak, že keby mal vymeniť dvojicu, v ktorej je niektorý balvan priťažký, tak namiesto toho nespraví nič. Na vstupe sú čísla n, w a pole H obsahujúce začiatočné poradie hmotností balvanov. Navrhnite algoritmus, ktorý vypočíta výsledné poradie balvanov po tom, ako Sizyfos dokončí svoj postup. (Výstupom má teda byť obsah poľa H na konci kola, v ktorom už Sizyfos nič nevymení.)

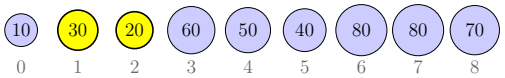
Podúloha C (5 bodov):
Popíšte algoritmus, ktorý pre rovnaký vstup ako v podúlohe B efektívne vypočíta, koľko kôl bude trvať Sizyfov postup pre túto konkrétnu postupnosť balvanov. **Dokážte správnosť** svojho algoritmu.

Formát riešení a hodnotenie:
Podúlohy môžete riešiť každú zvlášť v ľubovoľnom poradí. Podúlohy B a C môžete tiež vyriešiť obe naraz pomocou jedného algoritmu. Na zisk plného počtu bodov je v každej z podúloh B a C potrebné nájsť algoritmus s časovou zložitosťou $O(n \log n)$. Za riešenie podúloh B a C s časovou zložitosťou kvadratickou od n (alebo horšou) môžete získať dokopy nanajvýš 2 body.
Jeden bod strhneme algoritmom, ktoré fungujú len za predpokladu, že všetky hmotnosti v poli H sú navzájom rôzne. Ak tento predpoklad vaše riešenie využíva, *explicitne to uveďte*.

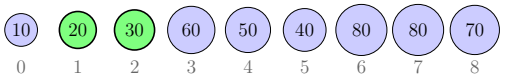
Príklady

Uvažujme $n = 9$ a postupnosť balvanov s hmotnosťami $H = (10, 30, 20, 60, 50, 40, 80, 80, 70)$.
Pozrime sa najskôr na situáciu, kedy Sizyfos vie hýbať všetkými balvanmi (teda $w \geq 80$).
V prvom kole by Sizyfos postupoval nasledovne:

- Na pozícii 1 netreba nič robiť.
- Na pozícii 2 je balvan ľahší od toho naľavo:

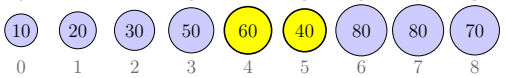


- Tieto balvany vymení:

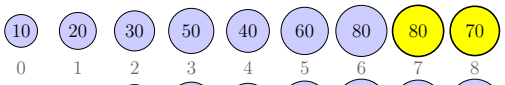


- Na pozícii 3 netreba nič robiť.
- Na pozícii 4 je balvan ľahší od toho naľavo, nastane výmena:

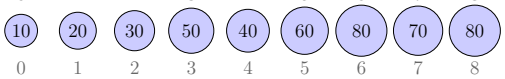
- Po ich výmene je na pozícii 5 ďalší taký prípad:



- Na pozíciách 6 a 7 netreba nič robiť.
- Na pozícii 8 spraví ďalšiu výmenu:

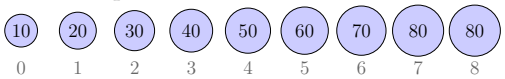


- Na konci prvého kola vyzerá pole H nasledovne:



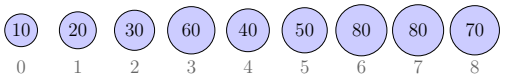
V druhom kole by Sizyfos spravil výmenu na pozícii 4 (balvany s hmotnosťami 50 a 40) a potom na pozícii 7 (balvany s hmotnosťami 80 a 70).
V treťom kole by už Sizyfos nespravil žiadnu výmenu. Proces teda skončil po troch kolách.

Výsledné pole H je usporiadané:



Ak by Sizyfos bol o čosi slabší ($w = 59$), v prvom kole by postupoval nasledovne:

- Na pozícii 1 netreba nič robiť.
- Na pozícii 2 spraví výmenu (hmotnosti 30 a 20).
- Na pozícii 3 netreba nič robiť.
- Na pozícii 4 výmenu spraviť nevládze, takže nespraví nič.
- Na pozícii 5 spraví výmenu (hmotnosti 50 a 40).
- Na pozíciách 6 a 7 netreba nič robiť.
- Na pozícii 8 opäť nevládze spraviť výmenu, takže aj tieto dva balvany ostanú na miestach.
- Na konci prvého kola vyzerá pole H nasledovne:



V druhom kole by už Sizyfos nespravil žiadne výmeny.
Proces teda skončí po dvoch kolách vo vyššie zobrazenom stave.

Ak by bol Sizyfos v tej istej začiatočnej situácii ešte slabší ($w = 27$), hneď v prvom kole nespraví žiadne zmeny a tým celý proces skončí.

A-II-2 Obdĺžnik

V rovine je daných $n \geq 10$ **navzájom rôznych** bodov. Navrhnite algoritmus, ktorý zistí, či všetky tieto body ležia na obvode nejakého (ľubovoľne otočeného) obdĺžnika. Ak áno, jeden taký obdĺžnik nájdite.

Formát vstupu a výstupu

V prvom riadku vstupu je číslo n.
Zvyšok vstupu tvorí n riadkov, v i-tom z nich sú dve **celé čísla** x_i a y_i: súradnice i-teho z bodov.
Na výstup vypíšte buď reťazec NIE, ak hľadaný obdĺžnik neexistuje, alebo súradnice troch vrcholov nejakého obdĺžnika, na ktorom ležia všetky zadané body.

Obmedzenia a hodnotenie

Pri riešení môžete predpokladať, že všetky operácie s reálnymi číslami sú presné. Inými slovami, nie je potrebné zaoberať sa zaokrúhľovacími chybami, ktoré by mohli nastať pri praktickej implementácii.
Na plný počet bodov je potrebné nájsť riešenie s (asymptoticky) optimálnou časovou zložitosťou a dokázať jeho správnosť. Pomalšie riešenia môžu dostať nanajvýš 8 bodov ak sú efektívne pre $n \leq 100\,000$, nanajvýš 6 bodov ak sú efektívne pre $n \leq 5\,000$, resp. nanajvýš 4 body ak sú efektívne pre $n \leq 50$.

Príklady

vstup
```
10
0 0
1 1
2 2
3 3
9 9
8 8
7 7
6 6
5 5
4 4
```

výstup
```
0 0
10 10
11 9
```

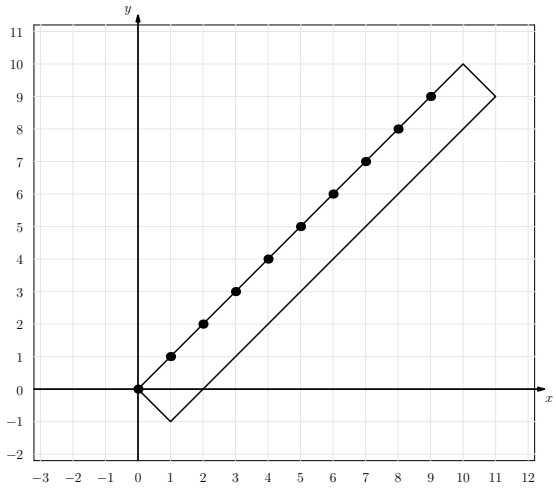
vstup
```
10
0 0
1 0
2 0
3 0
0 1
1 1
2 1
3 1
0 2
3 2
```

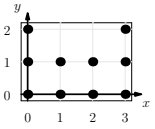
výstup
```
NIE
```

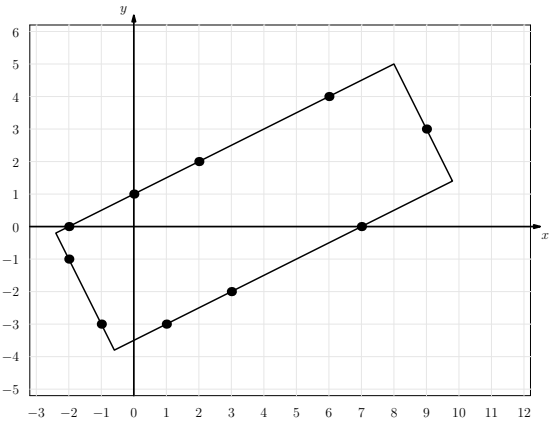
vstup
```
10
-2 0
0 1
2 2
6 4
9 3
7 0
3 -2
1 -3
-1 -3
-2 -1
```

výstup
```
-2.4 -0.2
8 5
9.8 1.4
```

V prvom príklade vstupu ležia všetky body na jednej priamke, riešením je ľubovoľný obdĺžnik, ktorého jedna strana leží na tejto priamke a obsahuje všetky naše body. V druhom príklade hľadaný obdĺžnik neexistuje. V treťom príklade existuje jediné riešenie. Všimnite si, že niektoré vrcholy obdĺžnika majú neceločíselné súradnice. Všetky tri príklady sú na obrázkoch nižšie.







A-II-3 Továreň

Máme továreň s p pracovníkmi a n objednávok. Objednávky sú očíslované od 0 po $n-1$ v poradí, v akom sme ich dostali.
Na vyriešenie každej objednávky potrebujeme dva **súvislé** kalendárne mesiace času, pričom na objednávke i musí počas prvého mesiaca robiť $A[i]$ a počas druhého mesiaca $B[i]$ pracovníkov.
Sme zmluvne zaviazaní dodržať poradie objednávok pri ich plnení – pre žiadne $i < j$ nesmieme neskôr prijatú objednávku j splniť ostro pred skôr prijatou objednávkou i. Ak na to ale máme dosť zamestnancov, môžeme ľubovoľne veľa po sebe prijatých objednávok splniť v tých istých dvoch mesiacoch.
Navrhnite algoritmus, ktorý vypočíta minimálny počet mesiacov potrebný na splnenie všetkých objednávok.

Formát vstupu a výstupu

V prvom riadku vstupu sú čísla p a n. Zvyšok vstupu tvorí n riadkov. Na i-tom riadku sú parametre i-tej objednávky $A[i]$ a $B[i]$.

Na výstup vypíšte jediné číslo: minimálny počet mesiacov, za ktorý vieme spracovať všetky objednávky.

Obmedzenia a hodnotenie

Všetky čísla na vstupe sú celé a nezáporné. Pre každú objednávku platí $0 \leq A[i], B[i] \leq p$.

Plný počet bodov môžu získať riešenia efektívne pre $p \leq 10^9$ a $n \leq 5000$.
Nanajvýš 8 bodov môžu získať riešenia efektívne pre $p \leq 10^9$ a $n \leq 300$.
Nanajvýš 6 bodov môžu získať riešenia efektívne pre $p, n \leq 100$.
Nanajvýš 4 body môžu získať riešenia efektívne pre $p \leq 10^9$ a $n \leq 12$.

Príklady

vstup	výstup
1000 4 300 100 300 100 100 800 700 200	3

vstup	výstup
1000 3 100 800 100 800 100 800	4

vstup	výstup
1000 5 500 499 500 500 1 1 500 500 499 500	4

Prvý príklad: Počas prvých dvoch mesiacov by sme vedeli robiť na objednávkach 0, 1 a 2 súčasne. Objednávku 3 by sme potom ale nevedeli spraviť skôr ako počas tretieho a štvrtého mesiaca. Lepším riešením bude počas prvého mesiaca začať len objednávky 0 a 1 (potrebujeme 300+300 = 600 zamestnancov), počas druhého mesiaca ich dokončiť a zároveň začať objednávky 2 a 3 (potrebujeme 100+100+100+700 = 1000 zamestnancov) a počas tretieho mesiaca dokončiť objednávky 2 a 3 (potrebujeme 800+200 = 1000 zamestnancov).

Druhý príklad: Nemôžeme nikdy začať dve objednávky naraz – síce by sme na to mali prvý mesiac dosť zamestnancov, ale druhý mesiac by sme ich potrebovali aspoň 1600, čo nemáme. Optimálne je každý mesiac začať jednu objednávku.

A-II-4 Nechaj to na solver II

K tejto úlohe patrí študijný text uvedený nižšie. Je zhodný so študijným textom z domáceho kola.

Podúloha A: slabé závislosti (2 body)

Pripomeňme si úlohu z domáceho kola: 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.
Slabá závislosť medzi projektmi vyzerá nasledovne: projekt x_i môžeme podporiť len vtedy, ak zároveň s ním podporíme *aspoň jeden* spomedzi projektov $y_{i,1}, y_{i,2}, \ldots, y_{i,z_i}$. Napr. projekt na pestovanie bio kapusty sa nevie uskutočniť ak nebude podporený ani projekt na montáž automatického zavlažovania ani projekt na 3D tlač obojručných krhiel.
Popíšte, ako pre danú sadu projektov a zoznam slabých závislostí medzi nimi zostrojiť ILP, ktorého optimálne riešenie bude zodpovedať sade projektov, ktorých podporením získame maximálny celkový profit. Popíšte celý ILP, vrátane častí, ktoré ostávajú rovnaké ako v domácom kole.

Príklad: Máme $e = 110\,000$ eur a $n = 5$ projektov.
Sumy s_i na ich podporu sú 50 000, 50 000, 20 000, 40 000, 49 999 a výnosy v_i sú 50 100, 95, 26 000, 900 000, 1.
Máme dve slabé závislosti: projekt 4 závisí na projekte 3 (teda $x_1 = 4$, $z_1 = 1$ a $y_{1,1} = 3$)
a projekt 3 závisí na projektoch 2 a/lebo 5 (teda $x_2 = 3$, $z_2 = 2$, $y_{2,1} = 2$ a $y_{2,2} = 5$).
Optimálnym riešením je podporiť projekty 2, 3 a 4. Na konci budeme mať 816 095 eur.
Ak by sme v tej istej situácii mali len 100 000 eur, optimálne by bolo podporiť iba projekt 1.

Podúloha B: zjazdovky (8 bodov)

V okolí horskej dediny je k kopcov (očíslovaných od 1 po k), na ktorých sa dajú stavať zjazdovky. Na každom kopci ich vieme postaviť **najviac tri**. Každá zjazdovka postavená na kopci i musí mať celočíselnú dĺžku, a to aspoň ℓ_i metrov a nanajvýš u_i metrov. Postaviť každý meter zjazdovky na kopci i nás stojí c_i eur.
Chceli by sme, aby naše lyžiarske stredisko malo celkovú dĺžku zjazdoviek presne d metrov.
Popíšte, ako zostrojiť ILP, ktorý bude mať riešenie práve vtedy, ak sa náš cieľ dá dosiahnuť, a navyše ktorého optimálne riešenie bude zodpovedať najlacnejšiemu možnému spôsobu postavenia hľadanej sady zjazdoviek.

Príklad: Máme $k = 2$ kopce. Na prvom sa dajú stavať zjazdovky dĺžky $\ell_1 = 1000$ až $u_1 = 1100$ metrov, a to meter za $c_1 = 100$ eur. Na druhom sa dajú stavať zjazdovky dĺžky $\ell_2 = 300$ až $u_2 = 400$ metrov, a to meter za $c_2 = 70$ eur. Chceme stredisko s 2410 metrami zjazdoviek.
Jedným optimálnym riešením je postaviť na prvom kopci dve zjazdovky dĺžok 1003 a 1007 metrov a na druhom kopci jednu zjazdovku dĺžky 400 metrov. Dokopy zaplatíme 229 000 eur.

Čiastočné body za túto podúlohu môžete získať vyriešením ľahšej verzie tejto úlohy: Nanajvýš 6 bodov dostanete za vyriešenie úlohy, v ktorej na každom kopci vieme postaviť *najviac jednu* zjazdovku. Nanajvýš 3 body dostanete za vyriešenie úlohy, ak navyše k predchádzajúcej podmienke budete predpokladať, že pre každý kopec platí $\ell_i = u_i$. Ak budete odovzdávať riešenie niektorej z týchto ľahších úloh, výrazne to v ňom *uveďte*.

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, Michal Forišek, Sebastian Hajdu
Recenzia: Michal Forišek
Slovenská komisia Olympiády v informatike
Vydal: NIVAM – Národný inštitút vzdelávania a mládeže, Bratislava 2026