75. ročník Matematické olympiády – 2025/2026

Úlohy ústředního kola kategorie P – 2. soutěžní den

Na řešení úloh máte 4.5 hodiny čistého času. Při soutěži je zakázáno používat jakékoliv pomůcky kromě psacích potřeb a přiděleného počítače (tzn. knihy, kalkulačky, mobily, apod.).

Řešením prvních dvou příkladů je zdrojový kód programu zapsaný v jednom z následujících programovacích jazyků: C, C++ 20, Python 3.13, C# 12, Java 25 nebo Pascal. Vzhledem k různé výkonnosti stejného algoritmu implementovaného v různých programovacích jazycích nezaručujeme, že je ve všech podporovaných jazycích možné získat plný počet bodů – může se stát, že program nestihne doběhnout do časového limitu, i když implementuje algoritmus s optimální časovou složitostí. Časové limity jsou (s rezervou) nastavené dle autorských řešení implementovaných v C++.

Řešení odevzdáváte pomocí soutěžního systému CMS, který ho automaticky otestuje na připravených sadách testovacích dat. Detaily hodnocení naleznete v letáku s popisem prostředí. Podrobnější informace o testovacích datech najdete na konci zadání každé úlohy.

K poslední úloze v soutěžním prostředí naleznete vstupní soubory, v CMS se k nim odevzdávají se pouze soubory obsahující odpovídající správné výstupy. Autorské řešení této úlohy je založeno na využití řešiče ILP, výstupy pro tuto úlohu ale můžete vytvořit libovolným způsobem (např. ručně nebo pomocí programu v jednom z podporovaných programovacích jazyků).

Zadání naleznete na další straně.

P-III-4 Mafiáni u doktora

Lucce se podařilo najít nové zaměstnání jako recepční u doktora v městečku Palermu. Její práce spočívá v tom, že postupně volá pacienty z čekárny do ordinace. Den teprve začal, ale Lucka už chápe, že to nebude snadné. Na Sicílii vůbec nezáleží na tom, jak je kdo nemocný, a jen trochu se bere ohled na to, kdy kdo přišel. Na čem skutečně záleží je, z jaké rodiny pochází. Nikdo si netroufne nechat pacienta z vlivné rodiny dlouho čekat!

Pacienti během dne postupně přichází do čekárny. Lucka vidí, kdy kdo přichází, a postupně jim přiřazuje *pořadová čísla* 0, 1, 2 atd. V kartotéce také snadno dohledá *vliv* daného pacienta (nezáporné celé číslo) a identifikátor *rodiny*, do které patří (také nezáporné celé číslo).

Bohužel se ale z kartotéky ztratily informace o tom, jak vlivná je která rodina. Lucka v městečku zatím vůbec nikoho nezná a musí si sama udělat představu o místních rodinách. V každém okamžiku Lucka předpokládá, že vliv každé rodiny je roven maximu z vlivů těch členů, které kdy viděla (včetně těch, kteří již byli léčeni a odešli). Pokud podle Lucky mají dvě rodiny přesně stejný vliv, považuje za vlivnější tu s menším identifikátorem.

Občas si doktor vyžádá dalšího pacienta, a Lucka musí poslat jednoho z těch v čekárně do ordinace. Tohoto pacienta vybírá následovně: musí pocházet z nejvlivnější rodiny v čekárně, a pokud má tato rodina v čekárně více než jednoho člena, musí to být ten, který tam čeká nejdéle (tedy ten s nejmenším pořadovým číslem).

Soutěžní úloha

Pomozte Lucce a napište pro ni program, který bude podle výše uvedených pravidel obsluhovat čekárnu.

Formát vstupu

Jednotlivé řádky vstupu popisují události v chronologickém pořadí.

- Vstup je ukončen řádkem obsahujícím pouze číslo 0.
- Je-li na aktuálním řádku trojice čísel 1, v a r ($0 \leq v \leq 10^9$, $0 \leq r < 200\,000$), znamená to, že do čekárny vstoupil pacient s vlivem v patřící do rodiny r.
- Je-li na aktuálním řádku pouze číslo 2, znamená to, že si doktor vyžádal dalšího pacienta.

Počet řádků vstupu (tj. počet událostí) označme jako q ($1 \leq q \leq 400\,001$).

Formát výstupu

Pokaždé, když si doktor vyžádá dalšího pacienta (tedy pro každý řádek vstupu tvořený číslem 2) vypište jeden řádek obsahující pořadové číslo pacienta, kterého má Lucka poslat do ordinace, či číslo -1, je-li aktuálně čekárna prázdná.

Poznamenejme, že tato úloha není testována jako interaktivní, není tedy nutné, abyste výstupní řádek vypisovali okamžitě po načtení odpovídajícího vstupního řádku, řešili vyprázdnění bufferů po vypsání řádku, a podobně.

Bodování

Vstupy jsou rozděleny do sedmi sad s různými dodatečnými omezeními, která jsou splněna pro všechny vstupy v dané sadě. Za správné vyřešení všech vstupů v sadě dostane váš program níže uvedený počet bodů.

- Sada 1 (1 bod): Pro každého pacienta platí $r = 0$, tj. všichni jsou z té samé rodiny.
- Sada 2 (1 bod): Nejprve přijdou všichni pacienti a poté si doktor několik z nich pozve do ordinace; tedy všechny události typu 1 nastanou před všemi událostmi typu 2.
- Sada 3 (1 bod): Pro každého pacienta platí $v = 0$, tj. všichni mají stejný vliv.
- Sada 4 (1 bod): Platí $q \leq 1000$, tedy vstup má nejvýše 1000 řádků.
- Sada 5 (1 bod): Všechna čísla v jsou navzájem různá a všechna čísla r jsou navzájem různá, tedy žádní dva pacienti nejsou z té samé rodiny ani nemají stejný vliv.
- Sada 6 (2 body): Pro každého pacienta platí $r < 10$, tj. všichni jsou z nejvýše 10 různých rodin.
- Sada 7 (3 body): Bez dodatečných omezení.

Příklad

Vstup:

```
2
1 1000 3
1 40 2
1 1000 2
2
1 1001 3
2
2
1 47 0
2
1 10 3
2
0
```

Výstup:

```
-1
1
0
3
2
5
```

Den probíhal následovně:

1. *Do čekárny ještě nikdo nedorazil, ale doktor si vyžádal pacienta; do ordinace tedy nikdo nejde.*
2. *Do čekárny dorazil pacient* 0 *(vliv* 1000*, rodina* 3*), pak pacient* 1 *(vliv* 40*, rodina* 2*), a poté pacient* 2 *(vliv* 1000*, rodina* 2*).*
3. *Doktor si zavolal pacienta. O rodinách* 2 *i* 3 *se aktuálně domníváme, že mají obě vliv* 1000*, vlivnější je tedy rodina* 2*, která má menší identifikátor. Z rodiny* 2 *nejdéle čeká pacient* 1*, ten tedy jde do ordinace.*

4. *Dorazil pacient* 3 *(vliv* 1001*, rodina* 3*), což pro nás zvýšilo vliv rodiny* 3*.*
5. *Když si nyní doktor zavolal pacienta, nejvlivnější je rodina* 3 *a do ordinace jde její člen* 0*.*
6. *Dále se na řadu dostal další člen rodiny* 3*, pacient* 3*.*
7. *Přišel pacient* 4 *(vliv* 47*, rodina* 0*, kterou jsme ještě neviděli).*
8. *Doktor zavolal dalšího pacienta. V čekárně zbývá pacient* 2 *z rodiny* 2 *a pacient* 4 *z rodiny* 0*. Rodina* 2 *je vlivnější a pacient* 2 *tedy jde do ordinace.*
9. *Dorazil pacient* 5 *(vliv* 10*, rodina* 3*).*
10. *Doktor si vyžádal dalšího pacienta. Právě dorazivší pacient* 5 *je z vlivné rodiny* 3*, a proto je na řadě.*
11. *Na konci dne v čekárně zbývá pacient* 4*, který má smůlu, nedostane se do ordinace ani na výstup.*

P-III-5 Voda

Speleologové objevili pomocí ultrazvuku novou jeskyni ve Slovenském krasu. Má tvar svislého obdélníku: je široká, hluboká, ale velmi úzká. K dispozici máte mapu této jeskyně, složenou ze znaků „`.`“ představujících prázdné políčko a „`X`“ představující políčko vyplněné skálou. Mapa je zadaná odshora dolů, první řádek mapy je tedy vertikálně nejvýše. Jeskyně je v současné době zcela pod zemí a nepřístupná, všechny znaky na obvodu obdélníka jsou proto `X`.

Jeskyně může být částečně nebo zcela zaplavena vodou. Voda se chová tak, jak byste očekávali, rozlévá se do stran a dolů a každá souvislá oblast naplněná vodou má všude hladinu ve stejné výšce.

Přesněji řečeno, *jeskyně s vodou* vznikne tak, že se vezme jeskyně a některá (možná všechna nebo žádná) prázdná políčka se zcela zaplní vodou. Vodu v jeskyni můžeme rozdělit na *propojené oblasti* tak, že dvě políčka s vodou patří do stejné oblasti, právě pokud by se z jednoho z nich na druhé dokázala dostat (možná i přes nějaká další políčka) ryba, která se může pohybovat pouze ve vodě a pouze ve směrech nahoru, dolů, doleva a doprava. Jeskyně s vodou je *stabilní*, pokud platí následující podmínky:

- Žádné prázdné políčko nemá hned nad sebou, vlevo od sebe ani vpravo od sebe políčko s vodou.
- Má-li prázdné políčko x hned pod sebou políčko s vodou, pak celá propojená oblast, do níž patří tato voda, musí ležet níže než políčko x.

Soutěžní úloha

Pro danou mapu jeskyně spočítejte, kolika způsoby může být zaplavena vodou, tj. kolik různých stabilních jeskyní s vodou z ní můžeme vytvořit. Přesněji řečeno, nechť p je přesně počet stabilních jeskyní s vodou, které můžeme vytvořit z dané jeskyně. Abyste se ve svých programech nemuseli zabývat aritmetikou s velkými čísly, vaším úkolem je vypočítat hodnotu $q = (p \bmod 1\,000\,000\,007)$, tj. zbytek, který p dává po dělení prvočíslem $10^9 + 7$.

Formát vstupu

Na prvním řádku vstupu jsou přirozená čísla r a s ($1 \leq r, s \leq 1000$), udávající počet řádků a sloupců mapy. Každý z r následujících řádků obsahuje řetězec délky s ze znaků „`.`“ a „`X`“, popisující mapu jeskyně odshora dolů. Můžete předpokládat, že první a poslední řádek a sloupec mapy je tvořen pouze znaky „`X`“.

Formát výstupu

Vypište jediný řádek obsahující jedno nezáporné celé číslo, hodnotu q pro zadanou jeskyni.

Bodování

Testovací vstupy jsou rozděleny do 10 sad, za vyřešení všech vstupů v každé z nich lze získat bod. Vstupy v každé sadě splňují různá omezení, specifikovaná v tabulce níže. V ní používáme následující terminologii:

- *Místnost* je maximální propojený úsek jeskyně tvořený prázdnými políčky (políčka jsou propojena pouze horizontálně a vertikálně, ne diagonálně).
- Jeskyně je *souvislá*, má-li jen jednu místnost.
- V *jeskyni se stalagmity* (krápníky rostoucími z podlahy) platí, že každé políčko, na němž je `X` a není v prvním ani posledním řádku mapy, má přímo pod sebou další `X`.
- V *jeskyni se stalaktity* (krápníky rostoucími ze stropu) platí, že každé políčko, na němž je `X` a není v prvním ani posledním řádku mapy, má přímo nad sebou další `X`.

sada	max(r, s)	další omezení
1	10	jeskyně je souvislá a její jediná místnost má tvar obdélníka
2	1000	všechny místnosti mají tvar obdélníka
3	50	souvislá jeskyně se stalagmity
4	1000	jeskyně se stalagmity
5	50	souvislá jeskyně se stalaktity
6	1000	jeskyně se stalaktity
7	50	jeskyně je souvislá
8	50	žádná další omezení
9	300	jeskyně je souvislá
10	1000	žádná další omezení

Příklady

Vstup:

```
4 9
XXXXXXXXX
X.XX.XXXX
XX.X.X..X
XXXXXXXXX
```

Výstup:

```
24
```

Tato jeskyně má čtyři místnosti. Pro každou jednopolíčkovou místnost a pro horizontální dvoupolíčkovou místnost máme dvě možnosti: buď zůstane prázdná, nebo v ní bude voda. Pro vertikální dvoupolíčkovou místnost máme tři možnosti: může být buď prázdná, napůl plná (spodní část obsahuje vodu, horní část ne) nebo zcela plná vody.

Tento vstup by mohl být v sadě 2 (každá místnost je obdélník).

Pokračování na další stránce...

Vstup:

```
4 5
XXXXX
X...X
X.X.X
XXXXX
```

Výstup:

```
5
```

Zde jsou následující možnosti (znak `O` představuje políčko s vodou):

```
XXXXX   XXXXX   XXXXX   XXXXX   XXXXX
X...X   X...X   X...X   X...X   XOOOX
X.X.X   XOX.X   X.XOX   XOXOX   XOXOX
XXXXX   XXXXX   XXXXX   XXXXX   XXXXX
```

Tento vstup by mohl být v sadě 3 (souvislá jeskyně se stalagmity).

Vstup:

```
4 7
XXXXXXX
X.X.X.X
X.....X
XXXXXXX
```

Výstup:

```
3
```

Zde jsou následující možnosti:

```
XXXXXXX   XXXXXXX   XXXXXXX
X.X.X.X   X.X.X.X   XOXOXOX
X.....X   XOOOOOX   XOOOOOX
XXXXXXX   XXXXXXX   XXXXXXX
```

Tento vstup by mohl být v sadě 5 (souvislá jeskyně se stalaktity).

Vstup:

```
5 9
XXXXXXXXX
X.X.X...X
X...X.X.X
X.X.XXX.X
XXXXXXXXX
```

Výstup:

```
42
```

Tento vstup by mohl být pouze v sadách 8 *a* 10*.*

P-III-6 Návrat řešiče

Tato úloha je open-data, testovací vstupní soubory tedy máte přímo k dispozici v soutěžním prostředí a v CMS odevzdáváte pouze odpovídající výstupní soubory. Za každý odevzdaný správný výstupní soubor získáte jeden bod. Výstupy můžete vytvořit libovolným způsobem; je na vás, zda či jak při tom použijete řešič ILP.

K úloze se vztahuje studijní text uvedený na následujících stranách, který je stejný jako v domácím a krajském kole. Úloha se skládá ze dvou nezávislých podúloh, řešit tedy můžete kteroukoliv z nich nebo obě dvě v libovolném pořadí.

Podúloha A (5 bodů): Ještě komplikovanější investice

Tato podúloha vychází z podúlohy „investice“ z domácího kola. Pro připomenutí, k dispozici máme e eur, které můžeme využít k podpoře některých z n projektů (očíslovaných od 1 do n). U každého projektu známe částku s_i potřebnou k jeho podpoře a hodnotu v_i, kterou časem získáme zpět jako výnos, pokud jej podpoříme. Všechny podpory jsou čerpány dříve, než obdržíme výnosy, součet částek s_i všech podpořených projektů tedy nesmí přesáhnout e.

Některé dvojice projektů si přímo konkurují a antimonopolní úřad vydal nařízení, že můžeme podpořit maximálně jeden projekt z každé takové dvojice. Každý z k takových konfliktů je zadán dvojicí čísel projektů $c_{i,1}$ a $c_{i,2}$, které nemůžeme oba zároveň podpořit.

Mezi projekty také mohou být závislosti: například projekt pěstování bio zelí nelze realizovat, pokud není podpořen projekt instalace efektivního zavlažovacího systému, takže pokud chceme podpořit zelí, musíme podpořit také zavlažování. Každou z celkem z závislostí máme zadánu dvojicí čísel projektů $d_{i,1}$ a $d_{i,2}$: Pokud chceme podpořit projekt číslo $d_{i,1}$, musíme podpořit také projekt číslo $d_{i,2}$. Jinými slovy, tato závislost nám brání podpořit $d_{i,1}$ a zároveň nepodpořit $d_{i,2}$. Závislosti mohou být zcela libovolné; například může jak projekt x záviset na projektu y, tak projekt y záviset na projektu x, a v tom případě je nutné podporovat buď oba projekty x a y zároveň, nebo žádný z nich.

Pro ústřední kolo do úlohy přibývají *synergie a antisynergie*: Když podporujeme celou skupinu konkrétních projektů najednou, někdy můžeme díky pozitivní interakci mezi nimi vydělat více, jindy naopak podpora konkrétní skupiny projektů celkový zisk sníží (např. když kromě chovu ryb podporujeme také chemickou továrnu v horní části toku).

Formálněji řečeno, máme zadáno q synergií, očíslovaných od 1 do q. Synergie i přináší zisk t_i (pokud je toto číslo záporné, jedná se o antisynergii) a týká se ℓ_i projektů. Čísla těchto projektů jsou $u_{i,1}, \ldots, u_{i,\ell_i}$. Pokud budou všechny tyto projekty podpořeny, náš zisk se změní o t_i, jinak se nic nestane.

Formát vstupu

Vstupní soubory najdete v `tasks/investice/input_s?.txt` ve vašem domácím adresáři (pozor, do podadresáře `tasks` nejde zapisovat, tak si soubor zkopírujte).

Každý vstupní soubor nejprve obsahuje jeden řádek s kladným celým číslem τ, udávajícím počtem testovacích vstupů, a poté obsahuje postupně τ testů. Každý

z testů je reprezentován následujícím způsobem:

- První řádek: kladná celá čísla e a n udávající náš kapitál a počet projektů.
- Druhý řádek: kladná celá čísla $s_1, \ldots, s_n$ udávající částky potřebné na podporu jednotlivých projektů.
- Třetí řádek: kladná celá čísla $v_1, \ldots, v_n$ udávající výnosy z podpořených projektů.
- Čtvrtý řádek: nezáporné celé číslo z udávající počet závislostí.
- Následujících z řádků: na i-tém z nich jsou čísla projektů $d_{i,1}$ a $d_{i,2}$ popisující i-tou závislost.
- Následující řádek: nezáporné celé číslo k udávající počet konfliktů.
- Následujících k řádků: na i-tém z nich jsou čísla projektů $c_{i,1}$ a $c_{i,2}$ popisující i-tý konflikt.
- Následující řádek: nezáporné celé číslo udávající počet synergií q
- Následující řádek: celá čísla (mohou být i záporná) $t_1, \ldots, t_q$ udávající výnosy ze synergií
- Následující řádek: kladná celá čísla $\ell_1, \ldots, \ell_q$, udávající počty projektů v jednotlivých synergiích
- Následujících q řádků: na i-tém z nich jsou čísla projektů $u_{i,1}, \ldots, u_{i,\ell_i}$ tvořících i-tou synergii.

Formát výstupu

Pro každý ze vstupních souborů `input_s1.txt` až `input_s5.txt` vytvořte a odešlete výstupní soubor `output_s?.txt` obsahující τ řádků, jeden pro každý test ve vstupu. Každý řádek výstupu by měl obsahovat jedno celé číslo: největší možnou částku peněz, kterou můžeme mít na konci daného testu, pokud optimálně vybereme, které projekty podpoříme a které ne.

Omezení

Všechna čísla na vstupu i na výstupu se zaručeně bez problémů vejdou do 32-bitových celočíselných proměnných; speciálně součet e, všech v_i a všech $|t_j|$ dohromady nepřekročí 150 000 000. Jednotlivé vstupní soubory navíc splňují následující podmínky:

- `input_s1.txt`: $\tau = 100$ testů, v každém z nich je $n \leq 400$ projektů, ve všech testech platí $q = 0$, tedy žádné synergie (řádky testu pro t_i a ℓ_i jsou prázdné).
- `input_s2.txt`: $\tau = 100$ testů, v každém z nich je $n \leq 400$ projektů, ve všech testech platí $q \leq 1$ (nejvýše jedna synergie) a tato synergie splňuje $t_1 > 0$.
- `input_s3.txt`: $\tau = 100$ testů, v každém z nich je $n \leq 400$ projektů, ve všech testech pro pro každou synergii platí $\ell_i = 2$.
- `input_s4.txt`: $\tau = 100$ testů, v každém z nich je $n \leq 150$ projektů, ve všech testech pro pro každou synergii platí $t_i > 0$.
- `input_s5.txt`: $\tau = 100$ testů, v každém z nich je $n \leq 600$ projektů.

Příklad

V souborech `input_s0.txt` a `output_s0.txt` naleznete příklad vstupu a odpovídajícího správného výstupu. V prvním testu je optimální podpořit projekty 1 a 2 (lepší než 1 a 3, protože získáme synergický bonus), jeden z dvojice 4 a 5 (aby nedošlo k negativní synergii) a projekty 6 a 7 (7 chceme a 6 musíme přidat kvůli závislosti).

Podúloha B (5 bodů): řežeme tyče

Ve stavebninách prodávají tyče o délce přesně ℓ milimetrů. My ale potřebujeme takové tyče o různých délkách: pro každé i od 1 do n potřebujeme p_i tyčí o délce d_i milimetrů. Máme k dispozici dokonalou pilu, pomocí které můžeme tyče řezat na libovolné kusy bez ztráty materiálu. Potřebujeme zjistit, kolik tyčí musíme minimálně koupit a jak je řezat, abychom mohli vyrobit vše, co potřebujeme.

Formát vstupu

Vstupní soubory najdete v `tasks/tyce/input_t?.txt` ve vašem domácím adresáři (pozor, do podadresáře `tasks` nejde zapisovat, tak si soubor zkopírujte).

Každý vstupní soubor nejprve obsahuje jeden řádek s kladným celým číslem τ, udávajícím počtem testovacích vstupů, a poté obsahuje postupně τ testů. Každý z testů je reprezentován následujícím způsobem:

- První řádek: kladné celé číslo ℓ, udávající délku prodávaných tyčí.
- Druhý řádek: kladné celé číslo n, udávající počet různých délek tyčí, které potřebujeme.
- Třetí řádek: kladná celá čísla $d_1, \ldots, d_n$ udávající délky tyčí, které potřebujeme.
- Čtvrtý řádek: kladná celá čísla $p_1, \ldots, p_n$ udávající počty tyčí těchto délek, které potřebujeme.

Formát výstupu

Pro každý ze vstupních souborů `input_t1.txt` až `input_t5.txt` vytvořte a odešlete výstupní soubor `output_t?.txt` obsahující výstupy pro všech τ testovacích vstupů. Každý výstup uveďte v následujícím formátu:

- První řádek: kladné celé číslo m, udávající počet tyčí délky ℓ, které je potřeba koupit.
- Druhý řádek: kladné celé číslo x, udávající počet instrukcí na řezání.
- Následujících x řádků: instrukce na řezání. Každá taková instrukce začíná kladným celým číslem, udávajícím počet nakoupených tyčí, na kterou ji máme použít. Zbytek řádku je tvořen jedním či několika záznamy typu „y `x` d“, který znamená, že z tyče máme y-krát odříznout kus délky d. Místo `1x` d můžete též vypsat pouze číslo d. Pro každou instrukci musí platit, že součet délek odříznutých kusů je nejvýše ℓ.

  Například, instrukce „`3 1x100 5x40 1x10`“ i „`3 5x40 10 100`“ nám obě říkají, že máme vzít 3 zakoupené tyče a každou z nich nařezat tak, aby-

chom získali jeden kus délky 10, jeden délky 100 a pět délky 40. Aby tyto instrukce byly možné, musí platit, že nakoupené tyče mají délku $\ell \geq 310$.

Pozor: Zbytky tyčí automaticky zahazujeme. Například, pokud pro $\ell = 2$ chceme vyrobit dvě tyče délky 1, nestačí napsat „`1 1`“, ale musíme explicitně napsat „`1 2x1`“ (nebo například „`1 1 1`“).

Výstup bude uznán jako správný, jestliže hodnota m je nejmenší možná, $x \leq 1000$, a zbylé řádky výstupu obsahují popis nějakého způsobu, jak vyrobit alespoň předepsaný počet tyčí požadovaných délek (tj. budou akceptována i řešení s optimálním m, která kromě potřebných tyčí vyrobí i nějaké další navíc).

Omezení

Jednotlivé vstupní soubory navíc splňují následující podmínky:

- `input_t1.txt`: $\tau = 50$ testů, v každém z nich chceme nejvýše 38 tyčí, které lze vyrobit ze dvou nakoupených tyčí.
- `input_t2.txt`: $\tau = 50$ testů, v každém z nich chceme nejvýše 22 tyčí, celková délka těchto tyčí je ostře větší než 3ℓ a je možné je vyrobit z $m = 4$ nakoupených tyčí.
- `input_t3.txt`: $\tau = 50$ testů, v každém z nich platí $10^6 \leq \ell \leq 10^7$, všechny požadované tyče mají délku nanejvýš 30 a celkovou délku nejvýše 15ℓ, a vždy existuje řešení, ve kterém je celková délka odpadu menší než 100 000.
- `input_t4.txt`: $\tau = 30$ testů, v každém z nich je $\ell = 10^7$, $n \leq 36$ a pro každé i platí $d_i > 2\,000\,000$ a $p_i = 1$.
- `input_t5.txt`: $\tau = 30$ testů, v každém z nich je $\ell = 10^7$, $n \leq 30$ a pro každé i platí $d_i > 2\,000\,000$.

Příklad

V souborech `input_t0.txt` a `output_t0.txt` naleznete příklad vstupu a odpovídajícího správného výstupu. V prvním testu koupíme dvě tyče, z první vyrobíme dva kusy délky 334 a z druhé třetí takový kus. V druhém testu koupíme dvě tyče a každou na vhodném místě přeřízneme. Ve třetím testu koupíme 7 tyčí. Tři z nich jen zkrátíme na 670 a zbytek zahodíme. Ze dvou vyrobíme po jednom kusu délky 780 (a z jedné z nich i zbytečný kus délky 220). Ze šesté tyče dostaneme kus délky 560 a ze sedmé dva kusy délky 450 (z nichž nám stačí jeden).

Odevzdávání výstupů

Výstupní soubory nemusíte odevzdávat všechny najednou. Odevzdáte-li jen některé, ty zbývající zůstanou obodované z předchozích odevzdání.

Řešiče

Na svých počítačích máte nainstalované řešiče `lp_solve` (z příkazové řádky i jako knihovnu pro C++) a `pulp` (knihovna pro Python). Návod k použití, který byl dostupný v domácím kole na *oi.sk,* najdete na pracovní ploše.

Studijní text: Celočíselné lineární programování

V letošním ročníku olympiády se budeme zabývat optimalizačními problémy, tedy problémy, které mají mnoho různých řešení, a naším úkolem je najít to nejlepší. Například nás může zajímat:

- Jak nejlevněji navštívit všechna města na Slovensku?
- Kolik krabic potřebuji k zabalení všech svých knih při stěhování?
- Jaká je velikost největší podmnožiny řešitelů letošní MO-P, ve které se všichni navzájem znají?

Mnoho optimalizačních problémů má jednu společnou nepříjemnou vlastnost: neznáme pro ně žádné efektivní algoritmické řešení. Empiricky si dovolíme tvrdit, že do této smutné kategorie spadá drtivá většina optimalizačních problémů, s nimiž se setkáváme všude v praxi – ať už v počítačích (např. plánování procesů, směrování paketů v sítích) nebo v reálném životě (např. logistika všeho druhu, optimalizace nákladů nebo různé problémy v bioinformatice). Mimochodem, všechny tři výše uvedené problémy sem také patří.

Situace je ještě horší. Nejenže neznáme žádný algoritmus, který by tyto problémy dokázal efektivně vyřešit (tj. s polynomiální časovou složitostí vzhledem k velikosti vstupu), ale máme dokonce velmi dobré důvody se domnívat, že žádný takový algoritmus neexistuje. To souvisí s jednou z nejdůležitějších otevřených otázek současné informatiky: otázkou, zda P se rovná NP. Zjednodušeně řečeno, jde o otázku, zda každou úlohu, u které můžeme efektivně zkontrolovat správnost řešení, lze také efektivně vyřešit. Intuitivně většina vědců věří, že je to nepravděpodobné – porovnejte například, jak obtížné může být ruční vyřešení i jednoduché sudoku a jak snadné je zkontrolovat, zda bylo sudoku vyřešeno správně. Tento příklad nám také ukazuje, že znalost toho, jak zkontrolovat správnost řešení, nám obecně nic neříká o tom, jak efektivně hledat řešení.

V praxi je to ale vlastně celkem jedno. Mezi situacemi, kdy pro náš obtížný úkol neexistuje žádný efektivní algoritmus a kdy existuje, ale žádný neznáme, není z praktického hlediska velký rozdíl. Pokud potřebujeme optimálně vyřešit zadání, jsme v obou případech závislí na hrubé síle, tj. na vyzkoušení všech možností.

Ne všechna řešení založená na hrubé síle jsou však stejně dobrá. Často můžeme taková řešení zefektivnit tím, že neprohledáváme všechny možnosti, ale chytře přeskočíme co nejvíce částí vyhledávání, o kterých víme, že nevedou k nejlepšímu řešení. Pro mnoho optimalizačních problémů jsme vyvinuli specifické algoritmy, které nejsou efektivní (jejich časová je stále exponenciální vzhledem k velikosti vstupu), ale díky vhodnému „ořezání“ vyhledávání mohou vyřešit mnohem větší vstupy v rozumném čase než přímé řešení, které zkouší možnosti úplně všechny.

Někteří chytří lidé však o tom přemýšleli a uvědomili si: v mnoha z těchto jednotlivých algoritmů provádíme velmi podobně vypadající optimalizace. Mohli bychom to nějak zobecnit? V letošním ročníku olympiády se budeme zabývat jednou z kladných odpovědí na tuto otázku.

*Celočíselné lineární programování** je způsob matematického popisu určitých optimalizačních problémů. Jeho výhodou je, že někdo již za nás odvedl veškerou opravdu náročnou práci – v současné době existuje několik velmi dobře optimalizovaných *řešičů,* které dokážou najít optimální řešení úlohy zadané takovým matematickým popisem. Navíc díky mnoha optimalizacím tyto řešiče často fungují efektivněji, než kdybychom sami psali a vylepšovali specializovaný algoritmus pro náš konkrétní úkol. To nám dává nový způsob řešení obtížných problémů: místo implementace vlastního řešení můžeme přemýšlet o tom, zda a jak můžeme tento problém zapsat jako ILP. Pokud se nám to podaří, můžeme k řešení našeho problému použít řešič ILP. A přesně to budete dělat při řešení soutěžních úloh v letošním ročníku olympiády.

Formální definice ILP

V dalším textu bude slovo *konstanta* označovat jakékoli konkrétní (případně i záporné) *celé* číslo a slovo *proměnná* bude označovat neznámou, která může nabývat jakékoli *nezáporné celé* hodnoty.

Celočíselný lineární program (ve své základní, tzv. kanonické formě) se skládá z následujících částí:

- *Omezení:* Sada lineárních nerovností, každá ve tvaru

  $$a_{i,1} \cdot x_1 + \cdots + a_{i,n} \cdot x_n \leq b_i,$$

  kde všechny $a_{i,j}$ a b_i jsou konstanty. Řešení musí *všechna* tato omezení splňovat.

- *Cíl:* Lineární výraz ve tvaru

  $$c_1 \cdot x_1 + \cdots + c_n \cdot x_n,$$

  kde c_i jsou konstanty a x_i jsou proměnné. Hodnotu tohoto výrazu chceme maximalizovat.

Jakékoli přiřazení hodnot proměnným, pro které jsou splněna všechna omezení, se nazývá *platným* řešením. Platná řešení, pro která má cílový výraz největší možnou hodnotu, se nazývají *optimální*.

Samozřejmě existují také ILP, které nemají optimální řešení. Může to mít dvě příčiny: buď jsou *nesplnitelné* (např. máme omezení $x_1 \leq 7$ a $-x_1 \leq -8$, čili $x_1 \geq 8$) nebo jsou *neomezené* (např. nemáme žádná omezení a chceme maximalizovat hodnotu $x_1 + 2x_2$).

* Pro tuto techniku jako celek i pro jednotlivé celočíselné lineární programy budeme v následujícím textu používat zkratku ILP, tedy Integer Linear Programming.

Flexibilnější a praktičtější definice ILP

Aby se nám s formalismem ILP pracovalo příjemněji, dovolíme trochu obecnější tvar programů:

- Povolíme také programy, jejichž cílem je minimalizovat hodnotu konkrétního výrazu, přičemž tento výraz může obsahovat také konstantní sčítanec.
- Povolíme také omezení, ve kterých je znaménko $\leq$ nahrazeno znaménkem $\geq$ nebo $=$.
- V podmínkách můžeme provádět všechny standardní aritmetické úpravy, např. vynechat sčítance ve tvaru $0 \cdot x_i$, libovolně přesouvat sčítance mezi levou a pravou stranou a používat závorky podle potřeby.

Rozmyslete si, že všechny tyto změny slouží pouze k lepší čitelnosti našich programů: například minimalizace $x + 3y + 1000$ je to samé jako jako maximalizace $-x - 3y$, podmínka $2x - 6y \geq y - 13$ je pouze jiný způsob zápisu podmínky $-2x + 7y \leq 13$ a podmínka $2x = 5y$ je stejná jako dvě podmínky $2x \leq 5y$ a $2x \geq 5y$.

Příklad: Kuřecí nugety

Stánek prodává tři různé balení kuřecích nugetů: 6 kusů za 2 eura, 9 kusů za 2,90 nebo 20 kusů za 6,10. Kolik nejvíc nugetů můžeme koupit za 32 eur?

Nesprávné hladové řešení: Když spočítáme, kolik zaplatíme za jeden nuget v každém balení, nejlepší možností je to největší. Za 32 eur můžeme koupit 5 největších balení, což nám dá 100 nugetů. To však není optimální řešení – všimněte si, že s tímto řešením nám kromě 100 nugetů zbude 1,50 eur, za které si nemůžeme koupit nic jiného. Existuje jiný způsob, jak lépe využít peníze, které máme, a získat více nugetů!

Tento úkol nelze obecně řešit hladově. Náš příklad s nugety je zvláštním případem dobře známého typu optimalizačního problému, který je obecně známý pod názvem *problém batohu.* Pro malé vstupy můžeme najít optimální řešení pomocí dynamického programování, ale obecně je řešení tohoto problému obtížné.

Lineární program: Označme x_1 jako počet malých balení, x_2 jako počet středních balení a x_3 jako počet velkých balení, které zakoupíme. Naším cílem je maximalizovat celkový počet nuget, které zakoupíme, tedy hodnotu $6x_1 + 9x_2 + 20x_3$. Musíme dodržet omezení, že celková kupní cena nesmí překročit náš rozpočet – to znamená, že (v centech, aby všechna čísla byla celá) musí platit následující: $200x_1 + 290x_2 + 610x_3 \leq 3200$.

Praktické řešení: Náš lineární program je zapsán v syntaxi, které rozumí řešič `lp_solve`, takto:

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

Když požádáme `lp_solve` o řešení tohoto programu, dostaneme následující výstup:

```
Value of objective function: 102.00000000

Actual values of the variables:
x_1                           1
x_2                           4
x_3                           3
```

Zjistili jsme, že můžeme získat až 102 nugetů, když koupíme 1 malé, 4 střední a 3 velká balení. Celková cena nákupu je 31,90, takže na konci budeme mít 102 nugetů a 10 centů nazbyt.

Výběr řešiče

Pro tento studijní text jsme vybrali jeden konkrétní řešič: `lp_solve`. V řešeních příkladů používáme syntaxi, kterou tento řešič rozumí.

Na adrese *https://oi.sk/apps/ilp/* najdete několik různých návodů, který řešič zvolit a jak jej použít k řešení ILP problémů v závislosti na vašem preferovaném operačním systému a programovacím jazyce. Pro domácí kolo si také na internetu můžete najít libovolný jiný řešič a použít ten, pokud se vám náš výběr nelíbí.

Příklad: Sudoku

Někdy místo optimalizace (hledání *nejlepšího* řešení z mnoha) nás může zajímat pouze nalezení jakéhokoli platného řešení nebo rozhodnutí, zda vůbec nějaké platné řešení existuje. Samozřejmě můžeme i k řešení takových problémů použít také řešič ILP: stačí mu nedat žádný cíl (nebo mu například dát cíl maximalizovat hodnotu výrazu „0“).

Podívejme se na známý logický problém: Sudoku. V tomto problému je cílem vyplnit tabulku 9×9 čísly od 1 do 9 tak, aby každý řádek, sloupec a „velký“ čtverec 3×3 obsahoval každé číslo od 1 do 9 právě jednou.

V tomto příkladu ukážeme, jak můžeme formulovat pravidla sudoku jako ILP. Zdálo by se, že bychom potřebovali 81 proměnných: pro každý čtverec tabulky jednu proměnnou reprezentující hodnotu, která by v něm měla být. A ano, to je jeden způsob, jak formulovat sudoku jako ILP, ale to necháme na později. V tomto příkladu použijeme jiný přístup: použijeme $9 \times 9 \times 9$ booleovských (tj. logických nebo binárních) proměnných. Proměnná $x_{i,j,k}$ bude 1, pokud má být hodnota k na souřadnicích (i, j), nebo 0, pokud tam hodnota k být nemá.

Podívejme se nyní, jak by mohly vypadat všechny pravidla sudoku, pokud by byly zapsány jako lineární rovnice a nerovnice.

- V každé buňce je přesně jedno číslo. Pro každé i a j tedy platí podmínka
$$x_{i,j,1} + x_{i,j,2} + \cdots + x_{i,j,9} = 1.$$
- Každé číslo se v každém řádku objevuje přesně jednou. Pro každé i a k tedy platí podmínka
$$x_{i,1,k} + x_{i,2,k} + \cdots + x_{i,9,k} = 1.$$

- Pro každý sloupec a každý čtverec platí analogické podmínky jako pro řádky.

Pokud nyní chceme vyřešit konkrétní sudoku pomocí `lp_solve`, postupujeme následovně:

- Vygenerujeme (např. pomocí jednoduchého programu napsaného v běžném programovacím jazyce) všechny výše uvedené podmínky představující obecná pravidla sudoku.
- Přidáme informaci, že všechny $x_{i,j,k}$ jsou booleovské proměnné. Toho dosáhneme přidáním podmínky $x_{i,j,k} \leq 1$ ke každé z nich. Proměnné, které mohou nabývat pouze hodnot 0 a 1, jsou však v modelování problémů tak běžné, že pravděpodobně každý řešitel bude mít speciální syntaxi pro přímé deklarování takových proměnných. Například v `lp_solve` stačí deklarovat takové proměnné jako `bin` místo `int`.
- Přidáme podmínky popisující konkrétní úkol, který se snažíme vyřešit. Například pokud máme číslo 7 již specifikované v prvním řádku a třetím sloupci úkolu, přidáme podmínku $x_{1,3,7} = 1$.

Příklad: Sudoku podruhé

Jak by vypadalo modelování sudoku, kdybychom chtěli použít proměnnou $v_{i,j}$ pro každou buňku, jejíž hodnota by přímo odpovídala hodnotě nalezené v příslušné buňce? Je zřejmé, že potřebujeme podmínky $v_{i,j} \geq 1$ a $v_{i,j} \leq 9$. Kromě těchto podmínek by stačilo přidat podmínky, které stanoví, že některé páry buněk nesmí mít stejnou hodnotu. Budeme potřebovat poměrně dost takových podmínek: jednu pro každý pár buněk ve stejném řádku, ve stejném sloupci a ve stejném čtverci 3×3. Například pro dvě pole (i, x) a (i, y) v řádku i potřebujeme podmínku $v_{i,x} \neq v_{i,y}$. Zde však narážíme na problém: tato podmínka nemá žádnou z povolených forem a nemůžeme ji přímo vyjádřit pomocí povolených podmínek.

Požadovanou podmínku můžeme zapsat jako logické OR dvou podmínek: musí platit buď $v_{i,x} < x_{i,y}$, nebo $v_{i,x} > v_{i,y}$. Jelikož všechna $v_{i,j}$ jsou celá čísla, můžeme tyto podmínky upravit do povolené formy: musí platit buď $v_{i,x} \leq v_{i,y} - 1$, nebo $v_{i,x} \geq v_{i,y} + 1$.

To však stále není v pořádku: v ILP musí být všechny podmínky splněny současně. To odpovídá logickému AND, nikoli logickému OR. Co s tím můžeme dělat?

Můžeme použít malý trik. Zavedeme novou binární proměnnou r (správně bychom ji měli nazvat například $r_{i,x,i,y}$, protože budeme potřebovat jednu novou proměnnou pro každou dvojici proměnných, které by neměly být stejné, pro lepší čitelnost ale indexy vynechme). Hodnota r nám řekne, zda by měla být menší první nebo druhá z hodnot v. Zvažme nyní následující dvě podmínky:

$$v_{i,x} - v_{i,y} \geq 1 - 10r$$
$$v_{i,y} - v_{i,x} \geq 1 - 10(1 - r) = 10r - 9$$

Pokud $r = 0$, dostaneme podmínky $v_{i,x} - v_{i,y} \geq 1$ a $v_{i,y} - v_{i,x} \geq -9$. První z nich říká, že $v_{i,x} > v_{i,y}$ a druhá je triviálně splněna pro libovolné $v_{i,x}, v_{i,y} \in \{1, 2, \ldots, 9\}$

(konstantu 10 jsme zvolili tak, aby toto platilo, využíváme tedy skutečnosti, že známe rozsah hodnot, kterých mohou $v_{i,x}$ a $v_{i,y}$ nabývat).

Naopak, pokud zvolíme $r = 1$, dostaneme jednu triviálně splněnou podmínku a jednu, která říká, že $v_{i,x} < v_{i,y}$. Přidáním této nové proměnné r a dvou výše uvedených podmínek jsme dosáhli toho, co jsme chtěli: pro libovolné $v_{i,x} \neq v_{i,y}$ můžeme splnit obě tyto podmínky, zatímco pro $v_{i,x} = v_{i,y}$ není možné splnit obě najednou.

Příklad: Co ve formalismu ILP nevyjádříme

Máme letadlo, se kterým chceme letět 1000 km z jednoho letiště na druhé. V rámci povolených rozsahů odpovídajících modelu letadla můžeme zvolit letovou hladinu h (výšku v km, ve které budeme létat, v rozmezí 10 až 13 km) a letovou rychlost v (v km/h, v rozmezí 600 až 900 km/h). Chtěli bychom minimalizovat náklady na let, tj. spotřebu paliva.

Toto lze zapsat pomocí vhodných vzorců jako optimalizační problém. Ve velmi zjednodušené podobě by to mohlo vypadat nějak takto: Doba letu bude $1000/v$. Pokud letíme ve výšce h, optimální rychlost vzhledem k odporu vzduchu je $v_{opt}(h) = 540 + 30h$. Výkon motoru je nejlepší ve výšce $h = 11{,}5$ km. Odchylka od těchto parametrů zvyšuje spotřebu paliva, ale může nás dostat k cíli rychleji. Spotřeba paliva (v kg/h) lze proto vyjádřit rovnicí $2000 + 200(h - 11{,}5)^2 + 0{,}05(v - v_{opt}(h))^2$. Celková spotřeba paliva je součinem této hodnoty a doby letu.

Ačkoli se jedná o přesné matematické vyjádření optimalizačního problému, je zde jeden háček: omezení pro h a v jsou lineární, ale funkce, jejíž hodnotu se snažíme optimalizovat, není lineární funkcí proměnných h a v. Řešič ILP nám proto s takto konkrétně zformulovaným problémem nepomůže.

Pro názornost dodejme, že ani mnohem jednodušší výraz $h \cdot v$ není lineární, protože je to součin dvou proměnných.