70. ročník Matematické olympiády – 2020/2021

Úlohy ústředního kola kategorie P – 1. soutěžní den

Na řešení úloh máte 5 hodin čistého času. Řešení každé úlohy odevzdejte do soutěžního systému OSMO jako samostatný soubor.

Řešení každé úlohy musí obsahovat:

- **Popis řešení,** to znamená slovní popis principu zvoleného algoritmu, argumenty zdůvodňující jeho správnost, diskusi o efektivitě vašeho řešení (časová a paměťová složitost). Slovní popis řešení musí být jasný a srozumitelný i bez nahlédnutí do samotného zápisu algoritmu (do programu). Není povoleno odkazovat se na vaše řešení předchozích kol, opravovatelé je nemají k dispozici; na autorská řešení se odkazovat můžete.
- **Zápis algoritmu.** Ve všech úlohách je třeba uvést zápis algoritmu, a to buď ve tvaru zdrojového textu nejdůležitějších částí programu v libovolném běžném programovacím jazyce či nějakém dostatečně srozumitelném pseudokódu. Nemusíte detailně popisovat jednoduché operace jako vstupy, výstupy, implementaci jednoduchých matematických vztahů, vyhledávání v poli, třídění apod.

Za každou úlohu můžete získat maximálně 10 bodů. Hodnotí se nejen správnost řešení, ale také kvalita jeho popisu (včetně zdůvodnění správnosti) a efektivita zvoleného algoritmu. Algoritmy posuzujeme zejména podle jejich časové složitosti, tzn. podle závislosti doby výpočtu na velikosti vstupních dat. Záleží přitom pouze na řádové rychlosti růstu této funkce. U úlohy P-III-3 hraje velkou roli i komunikační složitost.

V zadání každé úlohy najdete přibližné limity na velikost vstupních dat. Efektivním vyřešením úlohy rozumíme to, že váš program spuštěný s takovými daty na současném běžném počítači dokončí výpočet během několika sekund.

P-III-1 Dobíjecí stanice

Matěj si chce koupit elektromobil. S dobíjecími stanicemi v jeho končinách je to však zatím bída, takže začal přemýšlet, jak často by se mu mohlo stát, že se není schopen dostat z jednoho města do druhého.

Soutěžní úloha

Matějovo okolí se skládá z n měst a $n-1$ obousměrných silnic propojujících nějaké dvojice měst. Silnice se jinde než ve městech nepotkávají, a navíc mezi každými dvěma městy existuje právě jedna trasa. (Informatik by tedy řekl, že silniční síť tvoří *strom.*) V některých městech jsou navíc dobíjecí stanice.

Elektromobil, který má Matěj vyhlídnutý, je na jedno dobití schopen projet k silnic (pro jednoduchost předpokládáme, že jsou všechny silnice stejně dlouhé). V dobíjecí stanici lze elektromobil dobít znovu na plnou kapacitu.

Pomozte Matějovi spočítat dvojice měst, mezi kterými bude na elektromobilu schopný projet. Formálněji, spočítejte, kolik existuje uspořádaných dvojic (a, b) pro $a \neq b$ takových, že Matěj může vyrazit z a s plným nabitím a dorazit do b, aniž by někdy po cestě jel na vybitou baterii.

Formát vstupu

Na prvním řádku dostanete tři mezerou oddělená nezáporná celá čísla n, k, d. Na dalším řádku dostanete d mezerou oddělených přirozených čísel $a_1, \ldots, a_d$ ($1 \leq a_i \leq n$ pro $1 \leq i \leq d$) označujících města, ve kterých jsou umístěné dobíjecí stanice. Na každém z dalších $n-1$ řádků dostanete vždy dvě mezerou oddělená přirozená čísla u_i a v_i ($1 \leq u_i, v_i \leq n$, $u_i \neq v_i$) říkající, že města u_i a v_i jsou propojena obousměrnou silnicí.

Formát výstupu

Na výstup vypište jedno číslo, totiž počet uspořádaných dvojic různých měst (a, b) takových, že Matěj může vyrazit s plným nabitím z a a dorazit do b.

Příklady

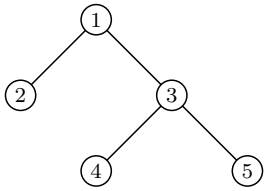
Vstup:

```
5 2 0

1 2
1 3
3 4
3 5
```

Výstup:

```
16
```



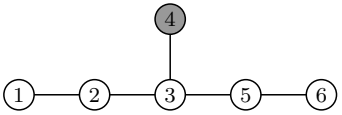
Nemáme žádné dobíjecí stanice. Cesta z 2 do 4 se skládá ze tří silnic (2–1, 1–3, 3–4), elektromobil však zvládne bez dobití projet jen dvě; podobně pro cesty $2 \to 5$, $4 \to 2$, $5 \to 2$. *Mezi všemi ostatními dvojicemi měst už umíme projet.*

Vstup:

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

Výstup:

```
30
```



V tomto případě se Matěj umí dopravit z každého města do každého jiného, potenciálně s mezizastávkou v dobíjecí stanici ve městě 4.

Bodování

Ve všech vstupech platí $n \geq 2$, $1 \leq k \leq 100$ a $0 \leq d \leq n$. Následující tabulka popisuje, kolik bodů můžete získat za řešení, která budou předpokládat příslušné horní meze na n:

body	n
10	300 000
6	8 000
4	1 000

Nějaké body můžete také získat, budete-li předpokládat, že v žádném městě není dobíjecí stanice.

P-III-2 Základny na Marsu

Během své mise na Marsu mají kosmonauti Kocourkovské vesmírné agentury za úkol postavit m vysílačů a n základen na přesně určených místech. Aby bylo ze základny možné komunikovat se Zemí, musí existovat tři vysílače takové, že základna leží uvnitř trojúhelníka s těmito vysílači jako vrcholy. Ředitel Kocourkovské vesmírné agentury dnes zjistil, že lokace pro základny a pro vysílače vybíraly dva různé týmy, které o sobě navzájem nevěděly. Pomozte mu ověřit, zda každá základna bude moci komunikovat se Zemí.

Soutěžní úloha

Na vstupu dostanete m bodů v rovině reprezentujících lokace pro vysílače a n bodů v rovině reprezentujících pozice základen. Pro každou základnu najděte tři vysílače takové, že základna leží uvnitř trojúhelníka jimi tvořeného, nebo vypište, že takový trojúhelník neexistuje. Pokud existuje více takových trojic vysílačů, vypište kteroukoliv z nich.

Můžete předpokládat, že všech $m + n$ zadaných bodů je po dvou různých, že pozice vysílačů jsou v obecné poloze (tj. žádné tři neleží na přímce) a že žádná pozice základny neleží na přímce tvořené pozicemi dvou vysílačů.

Ve svém řešení můžete bez detailnějšího popisu používat základní geometrické operace proveditelné v konstantním čase, například zjištění, na které straně orientované přímky AB leží bod C.

Formát vstupu

Na prvním řádku dostanete dvě mezerou oddělená přirozená čísla m a n. Na i-tém z následujících m řádků dostanete dvě mezerou oddělená celá čísla x_i a y_i, souřadnice i-tého vysílače. Na i-tém z následujících n řádků dostanete dvě mezerou oddělená celá čísla x_i a y_i, souřadnice i-té základny.

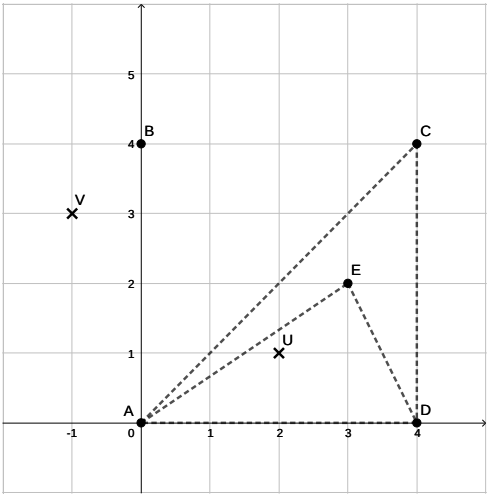
Formát výstupu

Na výstup vypište n řádků, na i-tém z nich vypište buď trojici čísel z množiny $\{1, \ldots, m\}$, totiž čísla vysílačů, které tvoří trojúhelník, uvnitř nějž leží i-tá základna, anebo -1, pokud takové tři vysílače neexistují.

Příklady

Vstup:	*Výstup:*
5 2 0 0 0 4 4 4 4 0 3 2 2 1 -1 3	1 5 4 -1

Vstup je nakreslený na následujícím obrázku, základny jsou zvýrazněny křížky a označeny jako U a V, vysílače jsou zvýrazněny puntíky a označeny jako A, B, C, D a E. Základna U leží například ve trojúhelníku ADE (což jsou základny číslo 1, 4 a 5), jiná vyhovující odpověď by ale byla například také `1 3 4` odpovídající trojúhelníku ACD. Základna V neleží uvnitř žádného trojúhelníka tvořeného vysílači.



Bodování

Ve všech vstupech platí $m \geq 3$ a $n \geq 1$. Pro všechny zadané souřadnice platí $-10^9 \leq x_i, y_i \leq 10^9$. Následující tabulka popisuje, kolik bodů můžete získat za řešení, která budou předpokládat příslušné horní meze na m a n:

body	m	n
10	10^5	10^5
7	10^5	50
5	10^3	10
2	100	10

P-III-3 Hledání na disku

Tato soutěžní úloha navazuje na úlohy z domácího a krajského kola. Za vlastním zadáním úlohy najdete studijní text, který je totožný se studijním textem z domácího a krajského kola.

V této úloze budeme chtít sestrojit datovou strukturu uloženou na disku pro vyhledávání v množině. Na začátku dostaneme množinu $M = \{x_1, \ldots, x_N\}$. Chceme ji předzpracovat a uložit na disk tak, abychom v ní uměli efektivně hledat. Pak budou přicházet dotazy na různá y a pro každé z nich budeme chtít rozhodnout, zda y leží v M.

Chceme dosáhnout co nejlepší komunikační složitosti hledání – tedy při hledání každého y přečíst co nejméně bloků z disku. Složitost předzpracování množiny (časová ani komunikační) nás nezajímá, dokud je polynomiální. Stačí, když popíšete, jak chcete data na disku uspořádat.

Prvky $x_1, \ldots, x_N$ a y nemusí být čísla. Můžete nicméně předpokládat, že každé dva umíme v konstantním čase porovnat a dozvíme se, který je větší, případně že jsou stejné. Prvky zabírají stejně paměti jako čísla, do jednoho bloku se jich tedy vejde B.

Podúloha A (3 body)

Prvky množiny můžeme na disk uložit v setříděném pořadí a pak v nich binárně vyhledávat (neboli hledat půlením intervalu). Časová složitost vyhledávání je $\mathcal{O}(\log N)$, nepočítáme-li načítání dat z disku. Jaká je komunikační složitost? Předpokládejte, že algoritmus z disku nečte opakovaně tentýž blok.

Podúloha B (5 body)

Vymyslete reprezentaci množiny s co nejmenší komunikační složitostí. Zachovejte přitom časovou složitost vyhledávání $\mathcal{O}(\log N + B \cdot \textit{počet přečtených bloků})$.

Podúloha C (2 body)

Dokažte, že komunikační složitost, které jste dosáhli v předchozí podúloze, je nejlepší možná. Přesněji řečeno že neexistuje žádný deterministický algoritmus, který by byl *řádově* efektivnější.

Za tuto podúlohu můžete získat body i tehdy, pokud podúlohu B nevyřešíte, ale dokážete nějakou netriviální dolní mez pro složitost všech algoritmů.

Studijní text

V tomto ročníku olympiády se budeme zabývat počítáním s obrovským množstvím dat. Budeme zpracovávat vstupy, které se nám ani celé nevejdou do hlavní paměti počítače. Proto naše programy budou muset pracovat s daty uloženými na disku a vyrovnat se s tím, že disk je mnohem pomalejší než hlavní paměť. Budeme přitom uvažovat následující zjednodušený model disku.

Náš počítač má k dispozici *vnitřní a vnější paměť.* Pro zjednodušení budeme vnitřní paměti říkat prostě *paměť* a té vnější *disk.*

K datům ve (vnitřní) pamětí můžeme přistupovat přímo – tak, jak jsme zvyklí v běžných algoritmech. Tato paměť ale má jenom omezenou kapacitu: M položek, kde M je nějaký parametr, který určíme později. Do každé položky můžeme uložit jedno „rozumně velké“ číslo. Tím myslíme čísla, která nejsou řádově větší než čísla na vstupu.

Disk je neomezeně velký. Je rozdělený na *bloky* o velikosti B položek (další parametr). S těmito položkami ale nelze počítat přímo. Můžeme pouze načíst celý blok do paměti, anebo naopak B po sobě jdoucích položek paměti zapsat do jednoho bloku na disku. Čtení budeme v programech zapisovat jako volání funkce Read(b), která dostane pořadové číslo bloku b a vrátí obsah bloku. Podobně Write(b, *obsah*) zapíše blok na disk. Jak čtení, tak zápis trvají čas $\mathcal{O}(B)$.

Efektivitu algoritmů posuzujeme pomocí dvou druhů složitosti: *časové složitosti* výpočtů v paměti a *komunikační složitosti,* což je počet přenosů bloků (čtení a zápisů) mezi pamětí a diskem.

Budeme se snažit hledat algoritmy, které dosahují stejné časové složitosti jako na klasickém počítači, a k tomu mají co nejlepší komunikační složitost. Nebude-li řečeno jinak, naše algoritmy by měly fungovat pro libovolné hodnoty parametrů M a B. Můžeme přitom předpokládat, že $\mathcal{O}(B)$ položek se do paměti vejde. Speciálně nám tedy paměť vystačí na konstantní počet číselných proměnných. Ale pokud bychom vytvořili rekurzivní funkci, musíme si dát pozor na prostor potřebný na zásobník.

Příklad: Sekvenční průchod

Práci s diskem si vyzkoušíme na jednoduchém příkladu. Na disku dostaneme posloupnost N celých čísel. Čísla budou rozdělena do bloků: v prvním bloku prvních B čísel, pak dalších B atd. Celkem tedy vstup zabírá $\lceil N/B \rceil$ bloků. Naším úkolem bude najít mezi zadanými čísly to největší.

Půjdeme na to jednoduše: budeme postupně procházet bloky vstupu jeden po druhém. Každý blok načteme do paměti, projdeme všechny jeho prvky, přičemž si udržujeme průběžné maximum. Poté už blok nebudeme potřebovat, takže tutéž paměť můžeme využít na další blok. V pseudokódu by to vypadalo takto:

1. $i \leftarrow 0$ ◁ *Aktuální pozice ve vstupu*
2. $m \leftarrow -\infty$ ◁ *Průběžné maximum*
3. Dokud $i < N$: ◁ *Ještě něco zbývá*

```
4.      b ← READ(i/B)                    ◁ Přečteme další blok vstupu
5.      n ← min(B, N − i)                ◁ Kolik v něm je položek?
6.      Pro j od 0 do n − 1:             ◁ Projdeme položky
7.          m ← max(m, b[j])
8.      i ← i + z
9.  Vypíšeme m.
```

Časová složitost tohoto algoritmu je jistě lineární s velikostí vstupu, tedy $\mathcal{O}(N)$. Jak je to s komunikační složitostí? Každý blok vstupu načteme právě jednou, takže přeneseme $\lceil N/B \rceil \leq N/B + 1$ bloků. To je jistě nejlepší možné, protože na každý blok se musíme alespoň jednou podívat.

Ještě musíme ověřit, kolik potřebujeme vnitřní paměti. Spotřebujeme konstantní počet položek na proměnné a B položek na právě zpracovávaný blok. To se určitě vejde do $\mathcal{O}(B)$, a tím pádem i do paměti.

Příklad: Test symetrie

Opět bude zadána nějaká posloupnost čísel. Tentokrát budeme chtít zjistit, zda je symetrická, tedy zde první prvek je roven poslednímu, druhý předposlednímu a tak dále.

Kdyby byla délka vstupu dělitelná B, stačilo by porovnat první blok s posledním (první má být zrcadlovým obrazem posledního), druhý s předposledním atd. Přitom bychom každý blok přečetli právě jednou.

Pokud délka vstupu dává po dělení B nějaký nenulový zbytek $z = N \bmod B$, je to trochu složitější: zrcadlový obraz prvního bloku bude zčásti v z položkách posledního bloku a zčásti v $B - z$ položkách předposledního.

Představíme si proto, že spustíme současně dva programy: jeden bude číst vstup od začátku do konce, druhý od konce k začátku. Každý z nich si bude udržovat svůj aktuální blok a čas od času přečte nový. Pokaždé porovnáme aktuální položku přečtenou oběma programy.

První program se bude chovat stejně jako sekvenční čtení z minulého příkladu, druhý bude číst tytéž bloky pozpátku. Dohromady tedy přečtou nejvýše $2\lceil N/B \rceil \leq 2N/B + 2$ bloků. Dokonce se mohou zastavit po zpracování $N/2$ párů prvků, čímž se komunikační složitost zlepší na $N/B + 2$. Časová složitost je stále $\mathcal{O}(N)$.

Dodejme ještě, že zapisování komunikační složitosti pomocí $\mathcal{O}$-čkové notace je poněkud zrádné. Funkci $2N/B + 2$ totiž nelze jen tak zjednodušit na $\mathcal{O}(N/B)$ – pro funkce dvou (a více) proměnných už neplatí, že přičítanou konstantu můžeme jen tak „schovat do $\mathcal{O}$“.* Nejjednodušší možný zápis tedy je $\mathcal{O}(N/B + 1)$.

* Notace $f(n) = \mathcal{O}(g(n))$ totiž znamená, že existuje konstanta c taková, že pro všechna n kromě konečně mnoha výjimek platí $f(n) \leq c \cdot g(n)$. U funkcí více proměnných definujeme $\mathcal{O}$ obdobně a povolíme konečně mnoho výjimek pro každou proměnnou. Jenže zlomek N/B může být libovolně malý i po zakázání konečně mnoha hodnot N i B, takže $+1$ nepřebijeme sebevětší konstantou c.

Příklad: Otočení posloupnosti

Pokračujeme v předchozím příkladu: co kdybychom chtěli posloupnost čísel otočit? Tedy přesunout první prvek na poslední místo, druhý na předposlední a tak dále.

Opět nejprve uvažujme případ, kdy je délka posloupnosti dělitelná velikostí bloku. Tehdy můžeme postupovat po blocích. Nejprve načteme do paměti první a poslední blok, otočíme obsah obou bloků a zapíšeme je zpět na disk v opačném pořadí. Pak provedeme totéž s druhým a předposledním blokem atd. A nakonec, máli posloupnost lichý počet bloků, přečteme prostřední blok, otočíme ho a zapíšeme zpět.

Všimněte si, že takto každý blok jednou přečteme a jednou zapíšeme, tudíž komunikační složitost činí opět $\mathcal{O}(N/B)$.

V případě, kdy zbytek $N \bmod B$ není nulový, použijeme osvědčenou úvahu se současně běžícími programy, které si vyměňují prvky. Komunikační složitost pak bude $\mathcal{O}(N/B + 1)$.