75. ročník Matematické olympiády – 2025/2026

Úlohy ústředního kola kategorie P – 1. soutěžní den

Na řešení úloh máte 4.5 hodiny čistého času. Řešení každé úlohy pište na samostatný list papíru. Při soutěži je zakázáno používat jakékoliv pomůcky kromě psacích potřeb (tzn. knihy, kalkulačky, mobily, apod.).

Řešení každé úlohy má obsahovat *popis řešení,* to znamená slovní popis principu zvoleného algoritmu, *argumenty zdůvodňující jeho správnost* (případně důkaz správnosti algoritmu), *diskusi o efektivitě* vašeho řešení (časová a paměťová složitost). Není možné se odkazovat na vaše řešení úloh z předchozích kol.

Do řešení nemusíte psát odpovídající program, algoritmus stačí zapsat ve vhodném pseudokódu nebo dokonce jenom slovně, je-li popis dostatečně podrobný a srozumitelný. Nemusíte detailně popisovat jednoduché operace jako vstupy, výstupy, implementaci jednoduchých matematických vztahů, vyhledávání v poli, třídění apod. Podrobnější seznam algoritmů a datových struktur, které považujeme za obecně známé a můžete je používat bez podrobnějšího popisu, naleznete na konci zadání.

Za každou úlohu můžete získat maximálně 10 bodů. Hodnotí se nejen správnost řešení, ale také kvalita jeho popisu a efektivita zvoleného algoritmu. Algoritmy posuzujeme podle jejich časové složitosti, tzn. závislosti doby výpočtu na velikosti vstupních dat. Záleží přitom pouze na řádové rychlosti růstu této funkce. 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.

Zadání naleznete na další straně.

P-III-1 Věže z kostek

Pavlínka vlastní $n \leq 300\,000$ kostek. Na každé z nich je napsáno jedno přirozené číslo, různé od čísel na všech ostatních kostkách. Dnes ráno vzala Pavlínka $v \geq 2$ podstavců s čísly od 1 do v a na každý podstavec položila jeden sloupec kostek. Pro zjednodušení budeme tyto sloupce nazývat *věže*. U každé věže víme, ze kterých kostek se skládá a v jakém pořadí na sobě kostky leží.

Pavlínka by ráda kostky přeuspořádala tak, aby v každé z věží odshora dolů čísla kostek rostla, tedy aby věže byly *uspořádané*. Samozřejmě by mohla jednoduše všechno rozebrat a znovu postavit, ale to by nebyla žádná zábava. Proto se Pavlínka rozhodla kostky přesouvat pouze tak, že vždy vezme horní kostku z nějaké věže a umístí ji *kamkoliv* (třeba i úplně dolů nebo úplně nahoru) do libovolné *jiné* věže. Zdůrazněme, že nesmí kostku vložit do té stejné věže, ze které ji odebrala. Každou takovou změnu budeme nazývat *krok*.

Pro upřesnění ještě dodejme, že během tohoto postupu může Pavlínka některou z v věží zcela vyprázdnit, a do takové prázdné věže pak později naopak může nějaké kostky přidat. Nesmí ale přidávat nové podstavce a vytvářet tak zcela nové prázdné věže. Prázdnou věž také považujeme za uspořádanou.

Formát vstupu v následujících podúlohách si můžete zvolit libovolně, například můžete předpokládat, že na začátku běhu vašich algoritmů jsou čísla n a v a složení věží již uložena v proměnných.

Podúloha A (3 body):

Navrhněte algoritmus, který vypočítá minimální počet kroků k potřebných k uspořádání všech věží.

Podúloha B (4 body):

Navrhněte algoritmus, který dá Pavlínce přesné pokyny, jak v optimálním počtu kroků k uspořádat všechny věže. Každý pokyn musí mít tvar „vezmi kostku z vrcholu věže x_i a umísti ji do věže y_i tak, aby byla z_i-tá odshora“. Stačí, když váš algoritmus na výstup vypíše tyto trojice čísel (x_1, y_1, z_1), …, (x_k, y_k, z_k) v pořadí, v jakém má Pavlínka operace provádět.

Podúloha C (3 body):

Po vyřešení předchozích podúloh se Pavlínka rozhodla, že si to celé ještě trochu zkomplikuje. Proto navíc předepsala čísla p_1, …, p_v udávající, z kolika kostek se na konci mají skládat jednotlivé věže. Stále také platí, že na konci všechny věže musí být uspořádané. Můžete předpokládat, že $p_1 + \cdots + p_v$ je přesně rovné celkovému počtu kostek.

Navrhněte algoritmus, který vypočítá minimální počet kroků nutných k uspořádání všech věží tak, aby na konci všechny věže měly předepsanou výšku. Přesné instrukce, jak toho dosáhnout, ale nemusí vypisovat.

Bodování

V podúlohách A a C získá dva body libovolný algoritmus implementovatelný v polynomiálním čase (tedy například nikoliv probrání všech možných posloupností

pokynů), u kterého dokážete a pečlivě zdůvodníte jeho správnost. Plné tři body za tyto podúlohy získá algoritmus efektivní i pro $n = 300\,000$, tj. s časovou složitostí $\mathcal{O}(n \log n)$ či obdobnou.

V podúloze B může získat plný počet bodů algoritmus efektivní i pro $n = 300\,000$, tj. s časovou složitostí $\mathcal{O}(n \log n)$ či obdobnou. Algoritmus efektivní i pro $n = 5\,000$, tj. s časovou složitostí $\mathcal{O}(n^2)$ či obdobnou, může získat až 2 body.

Příklady

Mějme $v = 3$ věže. První tvoří shora dolů kostky s čísly $[90, 80, 30, 10]$, druhou kostky s čísly $[60, 40, 50]$ a třetí kostky s čísly $[20, 70]$.

V této situaci potřebujeme k uspořádání věží 4 kroky; například můžeme postupovat následujícím způsobem.

- Horní kostku z věže 1 (s číslem 90) vložíme do věže 3 na pozici 3 shora (úplně dolů, pod kostku s číslem 70).
- Horní kostku z věže 1 (s číslem 80) vložíme do věže 3 na pozici 3 shora (pod kostku s číslem 70 a nad kostku s číslem 90).
- Horní kostku z věže 2 (s číslem 60) vložíme do věže 1 na pozici 3 shora (úplně dolů, pod kostku s číslem 10).
- Horní kostku z věže 1 (s číslem 30) vložíme do věže 2 na pozici 1 shora (úplně nahoru, nad kostku s číslem 40).

Po těchto čtyřech operacích věže vypadají následovně: první je $[10, 60]$, druhá je $[30, 40, 50]$ a třetí je $[20, 70, 80, 90]$.

Kdybychom měli předepsáno, že tyto věže mají mít na konci výšky $p_1 = 2$, $p_2 = 2$ a $p_3 = 5$, stále by stačily čtyři kroky – ve výše uvedeném řešení by stačilo změnit poslední krok a kostku s číslem 30 místo toho vložit do věže 3 na pozici 2 shora.

Kdybychom měli předepsané výšky $p_1 = 3$, $p_2 = 5$ a $p_3 = 1$, potřebovali bychom pět kroků.

P-III-2 Hnusný plot

Strýček Skrblík si u Donalda objednal krásný plot z n svislých latěk, které mají navzájem různé celočíselné výšky od 1 do n. Donald již všechny laťky objednal, ale pak mu Skrblík sdělil, že za plot zaplatí pouze polovinu slíbené částky. To Donalda pochopitelně rozzlobilo, a tak začal na plot zleva postupně přibíjet laťky tak, jak mu zrovna přišly pod ruku. Když však jeho počáteční vztek opadl, vymyslel ještě lepší pomstu. Místo krásného plotu postaví Skrblíkovi nejhnusnější plot na světě!

Plot nazýváme *hnusný*, pokud se v něm výšky latěk po řadě střídavě zmenšují a zvětšují; či přesněji řečeno, pro každé tři po sobě jdoucí laťky výšek a, b a c platí buď $a < b > c$, nebo $a > b < c$. Například plot s výškami latí $(7, 1, 8, 5, 6, 2, 4, 3)$ je hnusný, protože když jdeme podél plotu, vidíme, že druhá laťka je menší než první, třetí je větší než druhá, čtvrtá je menší než třetí a tak dále. Dalším hnusným plotem pro $n = 8$ je plot $(1, 8, 2, 7, 3, 6, 4, 5)$. Plot $(4, 2, 3, 5, 1)$ není hnusný, protože laťka 3 je vyšší než předcházející laťka 2 a menší než následující laťka 5.

Soutěžní úloha

Na vstupu máte dána čísla n (celkový počet latěk), k (počet latěk na začátku plotu, které Donald již přibil) a $v_1, \ldots, v_k$ (výšky těchto latěk v pořadí, v jakém jsou přibity k plotu). Navrhněte algoritmus, který vypočítá, kolika způsoby může Donald přibít zbývajících $n - k$ latěk na pozice $k + 1$ až n tak, aby vytvořil hnusný plot. Připomínáme, že výšky latěk jsou navzájem různá celá čísla od 1 do n, na přibití mu tedy zbývají právě laťky s výškami v množině $\{1, \ldots, n\} \setminus \{v_1, \ldots, v_k\}$.

Správná odpověď může být poměrně velké číslo. Abyste se nemuseli zabývat aritmetikou s velkými čísly, zavedeme v této úloze dvě zjednodušení:

- Můžete předpokládat, že čísla podobné velikosti jako správná odpověď se vejdou do standardních celočíselných proměnných.
- Při analýze časové složitosti můžete předpokládat, že základní aritmetické operace (sčítání, násobení) s tak velkými čísly lze provádět v konstantním čase.

Formát vstupu

V prvním řádku vstupu jsou čísla n a k ($1 \leq n \leq 7\,000$, $0 \leq k \leq n$). Na druhém řádku vstupu jsou přirozená čísla $v_1, \ldots, v_k$ ($1 \leq v_i \leq n$ pro každé i). Můžete předpokládat, že čísla $v_1, \ldots, v_k$ jsou navzájem různá.

Formát výstupu

Na výstup vypište jedno celé číslo, udávající počet různých hnusných plotů, které Donald dokáže postavit.

Bodování

Plný počet bodů může získat řešení s časovou složitostí $\mathcal{O}(n^2)$, tedy efektivní pro $N \leq 7\,000$. Za řešení s časovou složitostí $\mathcal{O}(n^3)$, tedy efektivní pro $N \leq 500$, můžete získat až 7 bodů. Za libovolné správné (i neefektivní) řešení můžete získat až tři body.

Řešení s těmito časovými složitostmi fungující pouze za dodatečného předpokladu, že $k = 0$, tj. že Donald ještě nepřibil žádnou laťku, získají o dva body méně.

Příklady

Vstup:	*Výstup:*
`10 3` `5 7 9`	`0`

Výsledný plot nemůže být hnusný, protože to kazí už první tři laťky.

Vstup:	*Výstup:*
`5 2` `2 4`	`2`

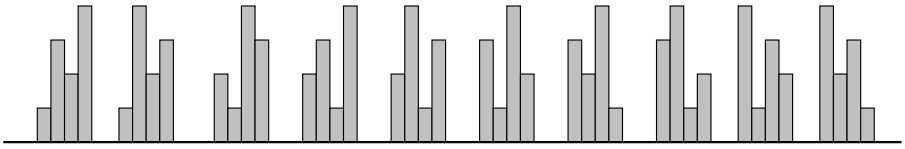
Donald může postavit buď plot $(2, 4, 1, 5, 3)$, *nebo plot* $(2, 4, 3, 5, 1)$.

Vstup:	*Výstup:*
`5 2` `2 3`	`1`

Zde má Donald jedinou možnost $(2, 3, 1, 5, 4)$.

Vstup:	*Výstup:*
`4 0`	`10`

Následující obrázek ukazuje všech 10 hnusných plotů ze 4 *latěk:*



P-III-3 Zhasínání

Tom žije ve velkém domě s n pokoji. Právě se chystal jít spát, když si uvědomil, že nechal všechna světla rozsvícená. Na noc samozřejmě musí všechna světla zhasnout. Problém je v tom, že světla v domě ovládá automatický systém. Ten vždy, když někdo vstoupí do pokoje, v tomto pokoji přepne stav světla: bylo-li rozsvícené, zhasne ho, a naopak. Tom proto potřebuje vymyslet způsob, jak projít domem tak, aby na začátku i na konci byl ve své ložnici a aby na konci byla všechna světla zhasnutá. Přitom může každým pokojem projít i vícekrát. Dokonce se může i okamžitě vracet – například může z ložnice vstoupit do sousedního pokoje, a poté se okamžitě zase vrátit do ložnice (čímž jednou přepne stav světla v sousedním pokoji i v ložnici).

Podúloha A: na čtvercové síti

Na vstupu je dána mapa Tomova domu jako obdélník znaků „`X`“ a „`.`“. Každá tečka představuje jeden pokoj v domě. Pokoje, které jsou těsně vedle sebe ve stejném řádku nebo sloupci, sousedí a lze mezi nimi přecházet (diagonálně přecházet nelze). Pokoje očíslujeme od 0 do $n-1$ v pořadí, v jakém se objevují na vstupu (pokoje v prvním řádku tedy mají zleva doprava čísla 0, 1, …, k_1, pokoje ve druhém řádku čísla k_1+1, k_1+2, …, k_2, a tak dále).

Podúloha B: obecně

Na vstupu je dána mapa Tomova domu číslem n (počet pokojů), číslem d (počet dveří mezi pokoji) a d páry čísel pokojů spojených dveřmi. Pokoje jsou očíslovány od 0 do $n-1$.

V obou případech je Tomova ložnice pokoj číslo 0. Napište algoritmus, který určí, zda Tom může začít ve své ložnici, projít nějak domem, skončit zpět ve své ložnici a dosáhnout tím toho, že všechna světla jsou zhasnutá. Pokud ano, váš algoritmus by měl také pro Toma sestrojit jednu možnou cestu po domě, která toho dosáhne. Je zaručeno, že z každého pokoje se dá dostat do každého jiného.

Formát vstupu

V podúloze A jsou na prvním řádku vstupu přirozená čísla r a s označující počet řádků a sloupců mapy. Dalších r řádků obsahuje řetězce délky s tvořící mapu.

V podúloze B obsahuje první řádek vstupu přirozená čísla n a d. Dalších d řádků obsahuje dvojice čísel pokojů propojených dveřmi.

Formát výstupu

Na výstup vypište buď zprávu „NELZE“, jestliže řešení neexistuje, nebo libovolnou správnou posloupnost čísel pokojů v pořadí, v jakém do nich má Tom vstoupit.

Bodování

Na zisk plného počtu bodů stačí popsat řešení podúlohy B s časovou složitostí $\mathcal{O}(n+d)$, tedy efektivní pro $n \leq 500\,000$ a $d \leq 10^6$.

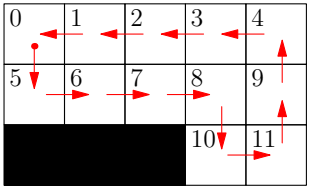
Částečné body je možné získat za pomalejší řešení podúlohy B a/nebo za řešení podúlohy A:

- Za řešení podúlohy B s časovou složitostí $\mathcal{O}(nd)$ či obdobnou, tedy efektivní pro $n \leq 3\,000$ a $d \leq 5\,000$, je možné získat 6 bodů.
- Za řešení pouze podúlohy A s časovou složitostí $\mathcal{O}(n)$, tedy efektivní pro $n \leq 500\,000$, je možné získat 6 bodů.
- Za obě tato řešení (méně efektivní pro podúlohu B a efektivní pro podúlohu A) lze získat 8 bodů.
- Za řešení pouze podúlohy A s časovou složitostí $\mathcal{O}(n^2)$ či obdobnou, tedy efektivní pro $n \leq 3\,000$, je možné získat 4 body.
- Dva body můžete získat za to, že přesně popíšete všechny vstupy, pro které v podúloze B neexistuje řešení, a dokážete, že pro tyto vstupy řešení skutečně neexistuje. Pro zisk těchto dvou bodů není potřeba, abyste popisovali algoritmus, který tyto vstupy pozná, a ani nemusíte dokazovat, že pro ostatní vstupy řešení existuje.

Příklady

Vstup:
```
3 5
.....
.....
XXX..
```

Výstup:
```
5 6 7 8 10 11 9 4 3 2 1 0
```



Vstup:
```
1 4
....
```

Výstup:
```
1 2 3 2 1 2 1 0
```

Vstup:
```
2 3
...
X..
```

Výstup:
```
NELZE
```

Poslední příklad by v podúloze B mohl být zadán následovně:

Vstup:
```
5 5
0 1
1 2
1 3
2 4
3 4
```

Výstup:
```
NELZE
```

Knihovna základních algoritmů

Pokud ve svém řešení teoretické úlohy chcete použít nějaký základní algoritmus, jako třeba binární vyhledávání v uspořádaném poli, nemusíte ho detailně popisovat. Někdy ale nemusí být jasné, které algoritmy jsou „dostatečně základní“.

Uvádíme proto seznam algoritmů a datových struktur, které v MO-P považujeme za natolik známé, že se na ně v řešení stačí odkázat a není potřeba uvádět detaily. Pokud si algoritmus potřebujete nějak přizpůsobit, stačí popsat pouze změny od základní verze.

Ostatní algoritmy je potřeba popsat celé.

Matematika

Euklidův algoritmus

- Nalezení největšího společného dělitele čísel x, y
- Čas $\mathcal{O}(\log x + \log y)$

Eratosthénovo síto

- Nalezení prvočísel od 2 do n
- Čas $\mathcal{O}(n \log \log n)$

Faktorizace

- Rozklad čísla n na součin prvočísel zkoušením dělitelů až po $\sqrt{n}$
- Čas $\mathcal{O}(\sqrt{n})$

Posloupnosti

Označme n délku posloupnosti.

Merge sort

- Třídění sléváním
- Čas $\mathcal{O}(n \log n)$

Bucket sort

- Přihrádkové třídění
- Čas $\mathcal{O}(n + r)$, kde n je počet prvků a r je jejich rozsah

Binární vyhledávání

- Nalezení prvku x v setříděné posloupnosti, případně největšího prvku $\leq x$
- Čas $\mathcal{O}(\log n)$

Grafy

Označme n počet vrcholů grafu a m počet jeho hran.

Prohledávání do hloubky

- Čas $\mathcal{O}(n + m)$

Prohledávání do šířky

- Nalezení nejkratší cesty ze startu do všech vrcholů v neohodnoceném grafu
- Čas $\mathcal{O}(n + m)$

Dijkstrův algoritmus

- Nalezení nejkratší cesty ze startu do všech vrcholů v ohodnoceném grafu s nezápornými délkami hran
- Čas $\mathcal{O}(m + n \log n)$

Bellman-Fordův algoritmus

- Nalezení nejkratší cesty ze startu do všech vrcholů v ohodnoceném grafu
- Čas $\mathcal{O}(nm)$

Floydův-Warshallův algoritmus

- Nalezení nejkratší cesty z každého do každého vrcholu v ohodnoceném grafu
- Čas $\mathcal{O}(n^3)$

Jarníkův algoritmus

- Nalezení minimální kostry pomocí řezů a haldy
- Čas $\mathcal{O}(m + n \log n)$

Kruskalův algoritmus

- Nalezení minimální kostry pomocí Disjoint set union
- Čas $\mathcal{O}(m \log n)$

Topologické uspořádání

- Uspořádání všech vrcholů orientovaného acyklického grafu tak, aby hrany vedly po směru uspořádání
- Čas $\mathcal{O}(n + m)$

Geometrie

Základy

- Vzdálenost dvou bodů (čas $\mathcal{O}(1)$)
- Vzdálenost bodu a přímky (čas $\mathcal{O}(1)$)
- Průnik přímek, úhly jimi svírané (čas $\mathcal{O}(1)$)
- Obsah mnohoúhelníku (čas $\mathcal{O}(n)$, kde n je počet vrcholů)

Konvexní obal

- Nalezení konvexního obalu n bodů
- Čas $\mathcal{O}(n \log n)$, případně $\mathcal{O}(n)$ s již seřazenými body

Datové struktury

U datových struktur uvádíme operace, které podporují, s jejich složitostmi.

Fronta

- *Enqueue:* Přidání prvku na konec v $\mathcal{O}(1)$
- *Dequeue:* Odebrání prvku ze začátku v $\mathcal{O}(1)$

Zásobník

- *Push:* Přidání prvku na konec v $\mathcal{O}(1)$
- *Pop:* Odebrání prvku z konce v $\mathcal{O}(1)$

Nafukovací pole

- *Append:* Přidání prvku v *amortizovaně* $\mathcal{O}(1)$

Prefixové součty

- *Build:* Vybudování v $\mathcal{O}(n)$
- *Query:* Dotaz na součet intervalu v $\mathcal{O}(1)$

2D prefixové součty

- *Build:* Vybudování v $\mathcal{O}(nm)$, kde n a m jsou strany mřížky
- *Query:* Dotaz na součet obdélníka v $\mathcal{O}(1)$

Vyhledávací strom

- Reprezentace množiny prvků, které umíme porovnávat.
- *Find:* Nalezení pozice daného prvku, případné zahlášení jeho neexistence v $\mathcal{O}(\log n)$
- *Insert:* Přidání prvku v $\mathcal{O}(\log n)$
- *Delete:* Smazání prvku v $\mathcal{O}(\log n)$

Písmenkový strom (Trie)

- Reprezentace množiny řetězců nad konstantně velkou abecedou.
- *Find:* Zjištění, zdali je slovo ve stromě v $\mathcal{O}(\ell)$, kde ℓ je délka slova
- *Insert:* Přidání slova v $\mathcal{O}(\ell)$
- *Delete:* Smazání slova v $\mathcal{O}(\ell)$

Binární halda

- *Min:* Vrácení minima v $\mathcal{O}(1)$
- *ExtractMin:* Odstranění minima v $\mathcal{O}(\log n)$
- *Insert:* Přidání prvku v $\mathcal{O}(\log n)$
- *Delete:* Smazání prvku v $\mathcal{O}(\log n)$, známe-li jeho pozici v haldě

Disjoint set union

- *Find:* Nalezení reprezentanta množiny obsahující daný vrchol v *amortizovaně* $\mathcal{O}(\alpha(n))$, kde α je inverzní Ackermannova funkce, která sice roste do nekonečna, ale mnohem pomaleji než logaritmus.
- *Union:* Sjednocení dvou množin obsahujících vrcholy u a v v *amortizovaně* $\mathcal{O}(\alpha(n))$

Intervalový strom (s línou aktualizací)

- Pro zvolenou asociativní operaci (minimum, maximum, součet, ...) a posloupnost prvků $x_1, \ldots, x_n$
- *Build:* Vybudování v $\mathcal{O}(n)$
- *Query:* Určení hodnoty operace provedené na všechny prvky s indexy v daném intervalu v $\mathcal{O}(\log n)$
- *UpdatePoint:* Aktualizace prvku v $\mathcal{O}(\log n)$
- Je-li operace minimum, maximum či součet, můžete dále bez popisu používat i operaci *UpdateRange:* Ke všem hodnotám s indexy v daném intervalu přičte zadané číslo, pracuje v čase $\mathcal{O}(\log n)$.
- Potřebuje-li vaše řešení operaci *UpdateRange* v jiných situacích (jiná asociativní operace, jiný druh změny hodnot na intervalu), musíte vysvětlit, jak ji implementujete.