Matematická olympiáda – kategorie P

Co se dozvíte?

- Co je MO-P?
- Jak by mělo vypadat řešení?

Matěj:

- Jak získat hodně bodů?

Co je MO-P?

- Programovací soutěž, ale
- zaměřená na efektivní řešení konkrétních úloh
- matematického/teoretického typu.

Jaké úlohy neřešíme

„Naprogramujte upravenou hru Pac-Man, ve které hráč sbírá jídlo, obchází zdi, vyhýbá se bombám a snaží se dojít do cíle dříve, než mu dojde všechna energie. Můžete také naprogramovat umělou inteligenci, která Pac-Mana sama dovede bezpečnou cestou až do cíle.“ (+ další 3 stránky zadání).

- Podobné běžnému programování v praxi.
  - Komplikovaný vstup/výstup.
  - Interakce s uživatelem, uživatelské prostředí.
  - Volnost ve specifikaci.
- Efektivita řešení nehraje velkou roli.

Jaké úlohy řešíme

Radek se vypravil na veletrh dortů, tvořený stánky očíslovanými 1 až n; na i-tém z nich může za a_i Kč ochutnat dort. Radek má k Kč a chce ochutnat co nejvíc dortů. Jakmile ale poprvé ochutná dort v nějakém stánku i, nedá mu to a ochutná poté dort ve vedlejší stánku $i + 1$, následně ve stánku $i + 2$ a tak dále, než mu dojdou peníze nebo než dojde na konec řady. Ve kterém stánku má Radek začít, aby ochutnal co nejvíce dortů?

Příklad vstupu:

```
7 6                    7 stánků, 6 korun
2 1 2 3 2 2 1    ceny dortů na stáncích
```

Najděte řešení, které problém efektivně vyřeší pro $n = 1\,000\,000$.

Úlohy v MO-P

- Jeden konkrétní přesně zadaný problém.
- Vstup a výstup v jednoduchém formátu, žádná další interakce s uživatelem.
- Důraz na efektivitu řešení.
- Úlohy zajímavé, ale většinou čistě teoretické.

Co by mělo obsahovat řešení – praktické úlohy

- Zdrojový kód řešení:
  - Pascal, C, C++, Java, Python

Vyhodnocení:

- Automatické testování na konkrétních vstupech různé velikosti.
- Musí doběhnout dostatečně rychle a vrátit správnou odpověď.

body	n
2	$n \leq 10$
2	$n \leq 1\,000$
3	$n \leq 100\,000$
3	$n \leq 1\,000\,000$

Co by mělo obsahovat řešení – teoretické úlohy

- Co nejsrozumitelnější popis řešení.
- Zdůvodnění jeho správnosti.
- Určení a zdůvodnění časové a paměťové složitosti.
- Zdrojový kód řešení:
  - v libovolném programovacím jazyce nebo pseudokódu
  - je možné vynechat nedůležité části (vstup, výstup, standardní algoritmy)

Vyhodnocení:

- Chybné/neoptimální řešení: 0 bodů.
- Jinak:
  - Cca 3 body za kvalitu popisu a zdůvodnění.
  - $0 - 7$ bodů dle efektivity.

Jaké úlohy řešíme

Radek se vypravil na veletrh dortů, tvořený stánky očíslovanými 1 až n; na i-tém z nich může za a_i Kč ochutnat dort. Radek má k Kč a chce ochutnat co nejvíc dortů. Jakmile ale poprvé ochutná dort v nějakém stánku i, nedá mu to a ochutná poté dort ve vedlejší stánku $i + 1$, následně ve stánku $i + 2$ a tak dále, než mu dojdou peníze nebo než dojde na konec řady. Ve kterém stánku má Radek začít, aby ochutnal co nejvíce dortů?

Příklad vstupu:

```
7 6                    7 stánků, 6 korun
2 1 2 3 2 2 1    ceny dortů na stáncích
```

Najděte řešení, které problém efektivně vyřeší pro $n = 1\,000\,000$.

Popis řešení

Pro každý stánek z, ve kterém Radek může začít jíst dorty, si určíme číslo stánku e_z, ve kterém skončí, a budeme si také pamatovat, jakou částku m_z za dorty utratí. Pro první stánek $z = 1$ prostě projdeme následující stánky a přičítáme ceny jejich dortů, dokud by jejich cena nepřekročila k (nebo dokud nedojdeme na konec).

Předpokládejme nyní, že už máme spočítáno e_z a m_z pro nějakou hodnotu z. Ukážeme si, jak je spočítáme pro $z + 1$. Zjevně stačí snížit m_z o hodnotu a_z dortu na stánku z a poté ji postupně zvyšovat o hodnoty dortů na stáncích $e_z + 1$, $e_z + 2$, ..., dokud bychom opět nepřekročili k nebo nedošli na konec.

Skončíme ve chvíli, kdy e_z dorazí na konec řady stánků (od té chvíle se počet snězených dortů může jen snižovat). Vrátíme maximum z hodnot $e_z - z + 1$ za celý běh programu. Samozřejmě si nemusíme ukládat všechny hodnoty e_z a m_z, stačí si pamatovat pouze ty pro aktuální z a maximum počítat průběžně.

Zdůvodnění správnosti

Popsaný algoritmus zjevně zaručí, že po zpracování začátku z dorty v úseku mezi z a e_z stojí $m_z \leq k$ Kč, ale dorty v úseku mezi z a $e_z + 1$ (nebo v jakémkoliv delším úseku) stojí více než k Kč. Pro každý začátek z tedy přesně určíme počet dortů $e_z - z + 1$, které by Radek snědl, kdyby začal u stánku z. Nutně tedy mezi nimi nalezneme i optimální hodnotu.

Časová složitost – O-notace

- Určit počet provedených operací přesně je složité.
- Operace trvají různě dlouho, čas jejich provedení závisí na stavu paměti, ...

Místo toho: „Časová složitost je $O(f(n))$.“

- Existuje nějaká konstanta c taková, že pro každé $n \geq 2$ algoritmus provede nejvýše $c \cdot f(n)$ operací.
- Více viz ksp.mff.cuni.cz/kucharky/slozitost/

Určení a zdůvodnění časové a paměťové složitosti

Povšimněme si, že hodnota e_z se v průběhu programu pouze zvyšuje, může se tedy změnit pouze n-krát. Hodnotu m_z měníme pouze tehdy, když se zvýší z nebo e_z. Celkový čas strávený úpravami e_z a m_z je tedy $O(n)$.

Kromě upravování hodnot e_z a m_z pro každé z provádíme jen konstantně mnoho operací (úprava aktuálního maxima, zvýšení z, ...), ty tedy dohromady také zaberou pouze čas $O(n)$. Celková časová složitost je tedy $O(n)$.

Pamětová složitost: Kromě pole délky n obsahujícího vstup si udržujeme pouze několik proměnných konstantní velikosti, paměťová složitost je tedy $O(n)$.

Běžné složitosti

Současné počítače provádí řádově 10^9 operací za sekundu:

- $O(n)$ algoritmy bývají efektivní pro cca $n \leq 10^6$
- $O(n^2)$ algoritmus možná pro $n \leq 10\,000$
- $O(2^n)$ pro $n \leq 30$

Zdrojový kód

Přeskakuji načtení vstupu do pole `ceny`. Do `ceny`[*n*] si uložím ∞, abych nemusel zvlášť ošetřovat konec.

```
void prepocet_e_a_m (void) {
  while (m + ceny[e + 1] <= k) {
    e++; m += ceny[e]
  }
  vysledek = max (vysledek, e - z + 1);
}

z = e = m = 0;
prepocet_e_a_m ();
while (e != n - 1) {
  m -= ceny[z]; z++;
  prepocet_e_a_m ();
}
```

Co u teoretických úloh nedělat

- Odevzdat pouze zdrojový kód (i komentovaný).
- Přepis programu do češtiny:
  - „Nejprve nastavíme *z*, *e* a *m* na 0. Pak zvyšujeme *e* o 1 a pokaždé k *m* přičteme `ceny`[*e*], dokud *e* nebude větší než *k*. . . .“
- Nevhodně pojmenované proměnné:
  - „Zavedeme si pole `pole`. Dále si zavedeme pole `pole2`.“
  - „K právě spočítanému číslu přičteme druhý z počtů určených v předminulém odstavci.“
- Popis běhu algoritmu na jednom konkrétním příkladě.
- Rozebírání nedůležitých technických detailů.

Co naopak dělat

- Popis srozumitelný i bez nahlédnutí do programu.
  - A ideálně i bez znalosti programování.
- Soustředit se na vysvětlení významu, ne na technikálie.
  - „e_z bude číslo stánku, kde Radek skončí, pokud začne na stánku číslo z“
- Příklady ke složitějším definicím.
  - „Například ve vzorovém vstupu budeme mít $e_3 = 4$, jelikož dorty na stáncích 3 a 4 dohromady stojí $5 \leq k$ Kč, ale na stáncích číslo 3 až 5 by stály $7 > k$ Kč.“
- Inspirujte se autorskými řešeními.

Model

Poslední teoretická úloha:

- delší studijní text
- netradiční výpočetní prostředky
- platí vše co pro ostatní teoretické úlohy
  - „časová složitost“ a „zdrojový kód“ mohou mít jiný význam, popsaný ve studijním textu

Organizace soutěže

Domácí kolo:

- Do 15. listopadu.
- 2 teoretické a 2 praktické úlohy
- Odevzdává se na mo.mff.cuni.cz.

Krajské kolo:

- Typická postupová hranice: Cca 10 bodů.
- 18. ledna.
- 4 hodiny, 4 teoretické úlohy řešené na papíře.

Ústřední kolo:

- 30 nejlepších účastníků.
- 2×5 hodin, 3 teoretické a 3 praktické úlohy
- 23.–25. března v Teplicích

Výběrové/přípravné soustředění:

- pro cca 10 nejlepších účastníků

Mezinárodní soutěže

Mezinárodní olympiáda v programování (IOI)

- 4 nejlepší účastníci
- srpen, Indonésie

Středoevropská olympiáda v programování (CEOI)

- 4 nejlepší nematuranti, kteří se nekvalifikovali na IOI
- ???, Chorvatsko

Proč MO-P řešit?

- Zajímavé úlohy.
- Mezinárodní soutěže.
- Přijetí a příprava na VŠ.