Soutěž dětí a mládeže v programování

Obvodní kolo Prahy 1 a 2, rok 2025

Časté chyby, zajímavosti řešení

Úloha 1: Zákazy

Úloha je zaměřená na vytvoření/zobrazení jednoduché grafiky.

Zajímavou chybou několik účastníků byla nemožnost vytvořit prázdný zákaz (tedy červený kruh bez piktogramu).

Několik řešení zvolilo jako usnadnění řešení rozdělení trojznačky horizontálně, místo úhlů ze středu. Zadání neodpovídá, ale mohlo to znamenat ztrátu pouze několika bodů - a nemusí člověk řešit stupně a radiány, že ano :)

Součástí úlohy bylo rozhraní k typu vygenerované značky. Záměrně nebylo pevně specifikováno a v řešeních se objevily minimalistické argumenty příkazu v řádce (CLI = Command Line Interface) (referenční úloha), interaktivní řádkové programy, i práce s různými UI (UI = User Interface) frameworky. Pro Python třeba Tkinter. Pokud bylo řešení odladěné a umožnilo zvolit značku dle zadání, udělovali jsme plný počet bodů. Nekompletní zadání je v praxi dost častým jevem, a pochopit co uživatel potřebuje bývá důležitou součástí soft-skills vývojáře.

Úloha 2: Počet slov

- Pokud je účel programu práce s více-řádkovým textem, není vhodné podmiňovat ovládání programu novou řádkou nebo prázdnou řádkou
- Dejte si pozor na kódování textu ve vašem programovacím jazyce. V C# je kódování stringů UTF-16 a “char” je také UTF-16. Standardní knihovna zaručuje převod kódování při čtení a prací s stdio. Ale třeba emoji se v UTF-16 zapisují dvěma chary, tedy s v délce stringu budou počítat jako2 znaky. Viz “UTF-16 surrogates”.

Úloha 3: Zmatené Šachy

Hledání do šířky (BFS) je jeden ze základních algoritmů pro hledání cesty v grafu - ale rozhodně nejde o jednoduše pochopitelnou znalost.

Inicializace: Zvolíme počáteční políčko a označíme je jako navštívené.
Použití fronty: Vložíme počáteční políčko do fronty (resp. *list.append* do seznamu zprava).
Iterace:

- Odebereme prvek z fronty (resp. *list.pop* zleva odebere prvek seznamu).
- Prozkoumáme všechny jeho sousedy (tedy políčka dosažitelné aktuální figurkou), kteří ještě nebyli navštíveni.
- Každé nově objevené políčko označíme jako navštíveného a vložíme do fronty.

Opakujeme proces, dokud není fronta prázdná.

Údaj o navštívenost lze například mít v dictionary/hash tabulce, či v poli polí reprezentujícím šachovnici.
Pro pozdější vyhledání celé cesty se místo samotného políčka ukládá jednoduchá struktura (políčko, předchozípolíčko, číslo_tahu) (efektivnější, vyžaduje backtracking) - a nebo (políčko, cesta_k_tomuto_políčku, číslo_tahu) (pomalejší ale u 8x8 šachovnice v pohodě)

Některá řešení bez této znalosti byla úžasně kreativní, a některá i omezeně funkční (např. pouze na jeden tah či na dva tahy, náhodná procházka, atp.)

Drobné chyby byly způsobeny příliš velkým množstvím kódu a překlepy (ten může udělat i AI!). Například mít pro každý tah samostatné ověřování, že je cíl uvnitř šachovnice. To je lepší nahradit ověřovací funkcí *validMove(x,y).*

Úloha 4: Logik přes síť

Optimální řešení algoritmu

Logik je NP-úplný problém. Nejlepší řešení spočívá ve výčtu všech kombinací a postupné filtrování. Při odeslání kombinace dostaneme zpět počet barev, kterými jsme se trefili a počet barev které v kombinaci jsou, ale na jiných pozicích. Podle odpovědi ze seznamu všech kombinací filtrujeme ty, které v porovnání s hádanou kombinací mají tyto počty stejné s odpovědí. Poté zkoušíme znovu hádat.
Pokud vybíráme pokusy k uhádnutí náhodně nebo sekvenčně, tento princip by měl nalézt řešení za cca 8 pokusů. Můžeme ale vybírat kombinace k hádání chytřeji. Pro každou kombinaci vyhodnotíme nejhorší odpověď, kterou můžeme dostat a použijeme kombinaci, která má tuto nejhorší odpověď nejlepší. To vyhodnocujeme podle toho, kolik by každá možná odpověď mohla vyškrtnout ze všech zbývajících kombinací. Použití tohoto algoritmu garantuje řešení do 6 pokusů.

Optimální řešení podle Donalda Knutha je garantované do 5 pokusů. Toho docílíme tím, že připustíme hádání kombinace kterou jsme již škrtli, pokud má nejlepší skóre na škrtání ze seznamu možných kombinací.

Síťová komunikace

Síťová komunikace je “podobná” jako práce s příkazovou řádkou nebo se soubory. Jenže při práci po síti nejsou nutně vždy řešeny problémy těchto věcí za nás. Když provedeme zápis, nemusí se provést hned. OS pro nás často bufferuje komunikaci, aby neposílal velké množství malých síťových packetů. Proto je nutné na konci odeslání jedné zprávy provést “flush” síťového spojení. Zároveň si musíme uvědomit, že čtení ze síťového spojení není nutně “po řádcích”. Knihovny mají často způsob, jak číst ze sítě po řádcích. Často tak, že zabalíme síťový stream do objektu, který komunikaci bufferuje a dovoluje číst “do určeného znaku”. V Javě “Scanner”, v C# “StreamReader”, v Rustu “BufReader”. V Pythonu se to dělá metodou na síťovém spojení “socket.makefile()”.