FIKS 8. Ročník, 3. kolo, úloha 3

Ryby – řešení

Kód čte/vypisuje standartní input/output. ***python ryby.py < input.txt > output.txt***
Kód běžel úspěšně na Windows 11, Python 3.9.4
Používá knihovny sys & ***numpy****

**numpy využívá čistě jenom pro parsing inputu a není součástí algoritmu řešení*

Teorie postupu:

Úlohu budeme řešit pomocí analytické geometrie. Z čísel s_x a s_y nám vznikne vektor, budeme mu říkat *v*. Pro každé hejno budeme hledat jeho hraniční body, tzn. hraniční pozice pro start lodi, mezi kterými pokud vystartujeme, tak hejnem projedeme. „Startovat" budeme z osy x (popř. z osy y, viz. protipříklad). Pro každé hejno najdeme hraniční pozice tak, že povedeme přímku *p* podle vektoru *v* každým z vrcholů, které hejno definují. Hranice hejna pak budou nejnižší a nejvyšší hodnota x v průsečíku *p* s osou x (tzn. y = 0). Ve chvíli, kdy budeme mít tyto hranice spočítané, uděláme z nich intervaly $\langle x_{min}; x_{max} \rangle$ (jedna množina pro každé hejno). Pak už stačí najít největší počet intervalů, se společným průnikem a jejich počet je řešením úlohy.

Protipříkladem řešení bude vektor rovnoběžný s osou x. V takovém případě musíme uvažovat start na ose y a hledat průsečík přímky *p* s osou y (x = 0). Řešení s osou y uvažujeme i pro všechny případy, kdy bude přímka *p* s osou x svírat úhel menší než 45°, aby se zabránilo případnému počítání s moc velkými čísly. Kdyby svírala přímka s osou úhel například jedna miliontina stupně, průsečík s osou x by mohl být klidně v číslech neuložitelných do paměti.

Popis kódu:

Řešení je class Flock a funkce solveFromBounds, zbytek je parsování inputu atp.

class Flock:

Tento class reprezentuje jedno hejno. V této úloze by asi nebylo nutné strukturovat kód jako class, ale osobně mi to přijde přehlednější.

__init__: class Flock bere dvě proměnné, tj. vector (array složený ze dvou čísel reprezentujících souřadnice X a Y směrového vektoru) a nodes (array obsahující množinu souřadnicových dvojic X a Y, které definují body polygonu).

findBounds je metoda, která projede každý bod polygonu a vytvoří z něj přímku se směrem vektoru *vector*, načež spočítá její průsečík s osou x nebo y podle obecné rovnice přímky ax + by + c = 0 (tj. hodnota x když y=0 nebo naopak) kde a,b je normálový vektor a x,y začáteční bod. Nejvyšší a nejnižší hodnota X nebo Y jsou poté vráceny v arrayi, se kterým dále pracujeme jako s intervalem. **Interval, který dostaneme, je interval hodnot, ze kterých když vystartujeme, projedeme tímto hejnem. Vzhledem k tomu že všechna hejna počítáme pro stejnou osu, maximální možný počet navštívených hejn můžeme zjišťovat jako nejvyšší počet intervalů se společným průnikem.**

solveFromBounds je metoda, které tyto intervaly dáme a zpátky dostaneme právě tento nejvyšší počet průniků. Funguje to tak, že se všechny hodnoty hodí do arraye a ke každé se přiřadí její funkce, tzn. buďto otevírá nový interval nebo nějaký zavírá. Poté array seřadíme podle velikosti a postupně projdeme každou hodnotu. Když je to hodnota, která otevírá interval, přičteme k momentálnímu počtu 1, pokud zavírá, odečteme 1. Nejvyšší hodnota, která se při loopování objeví je náš výsledek.

Čas a paměť:

Časová náročnost vytváření množin je suma všech bodů v úloze neboli O(n), přičemž n je počet bodů, která hejna definují. Hledání největšího průřezu pak ale musí seřadit množiny, což má náročnost O(u log u), kde u je počet útvarů. Celková časová náročnost je tedy
O(u*log(u) + v) což můžeme zjednodušit na **O(u*log(u))**

Paměťová náročnost je **O(u+v)** přičemž u je počet hejn a v je počet všech bodů v úloze, a to jak v části vytváření množin, tak v části hledání průřezu.