Řešení 6. úlohy 3. kola - Ryby

Úvod do problému

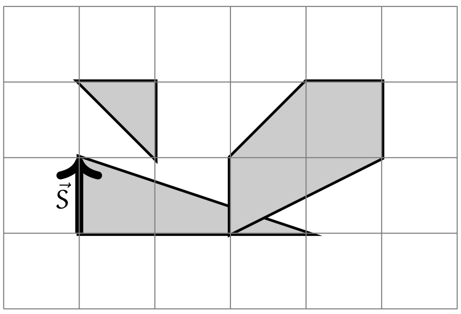
Úkolem je najít maximální počet průniků mezi hejny a trajektorií lodi. Hejna tvoří konvexní polygony udané pomocí vrcholů a trajektorie je přímka zadaná pomocí směrového vektoru.

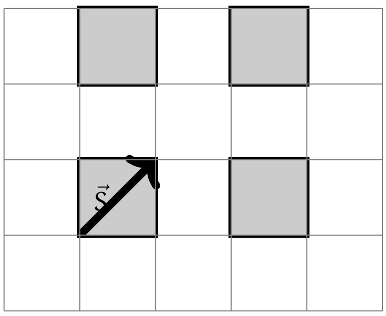
Teorie za řešením

Použiji příkladové vstupy pro demostraci

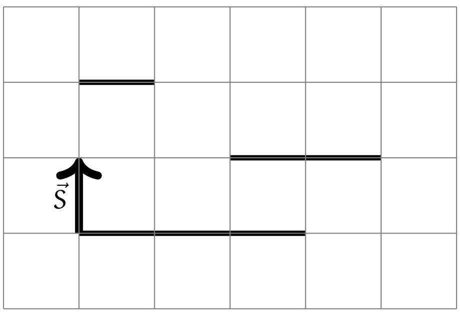
```
3 0 1
3 0 0 0 1 3 0
3 0 2 1 2 1 1
5 2 0 2 1 3 2 4 2 4 1
```

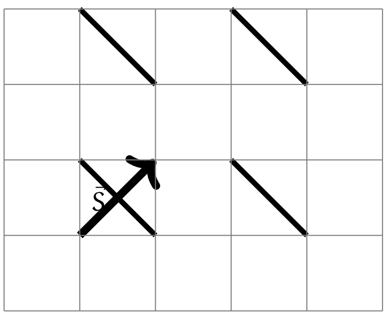
```
4 1 1
4 0 0 0 1 1 1 1 0
4 2 0 2 1 3 1 3 0
4 0 2 0 3 1 3 1 2
4 2 2 2 3 3 3 3 2
```





Toť zadání. Můžeme teď zkoušet všechny pozice, kam dát přímku a procházet hejna, jestli s ní mají průsečík. Ale to je moc práce a jde to udělat lépe. Hejna si zjednodušíme na úsečky. A to na úsečku, která má koncové body ve vrcholech hejn, které byly nejvíc vlevo a vpravo při pohledu ve směru vektoru.





Tyto čáry si pak můžeme ještě zjednodušit. Pokud si vyjádříme přímku trajektorie s pomocí obecné rovnice přímky, tak víme, že

$$s : s_y * x - s_x * y + c = 0$$

Výhodou této rovnice je, že nám pomůže určit, jak moc vlevo nebo vpravo jsou body na přímkách. Jak? Parametr c vyjadřuje posunutí přímky. A dá se jednoduše vypočítat

$$c = s_x * y - s_y * x$$

Pomocí tohoto vzorce pak v algoritmu i zjistíme, které body jsou ty krajní. Přiřaďme jim hodnoty parametru c na c_l a c_r. Tyto krajní body nám také říkají, že pokud bude mít přímka s parametr $c \in [c_l; c_r]$ tak bude mít průsečík s hejnem. Takto můžeme převést všechna hejna na interval. Hraniční body si seřadíme podle velikosti (přednost mají počáteční) a všechny je projdem. Předtím si nastavíme dvě proměnné jednu pro aktuální počet průsečíků a jednu pro maximální. Při procházení vždy, když narazíme na počáteční bod, zvětšíme aktuální počet o 1 a pokud je větší než maximální, tak ho aktuální počet nastavíme na maximální. Pokud narazíme na koncový bod, tak od aktuálního počtu 1 odečteme. Na konci pak budeme mít maximální počet hejn, ze kterých můžeme pochytat ryby při plavbě v jedné přímce.

Implementace

Po načtení vstupu projdeme všechny hejna a jejich vrcholy a vypočítáme maximální a minimální hodnotu c

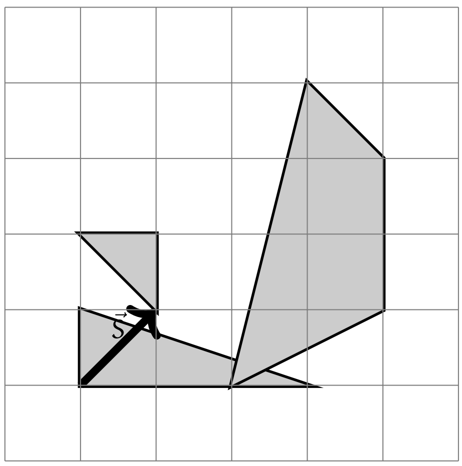
```
pro kazde hejno v hejna{
    maxC = -inf
    minC = inf
    pro kazdy vrchol v hejno{
        c = s.x*vrchol.y - s.y*vrchol.x
        maxC = max(c, maxC)
        minC = min(c, minC)
    }
    pridej interval od minC do maxC do intervaly
}

body = serad krajni body z intervaly
aktualni = 0
maximalni = 0
pro kazdy bod v body{
    kdyz bod.pocatecni{
        aktualni++
        maximalni = max(aktualni, maximalni)
        continue
    }
    aktualni--
}
```

Důkaz správnosti

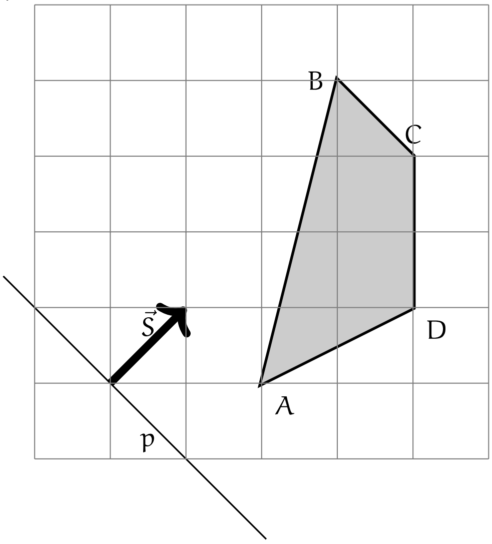
Pro důkaz použiji poupravený příkladový vstup

```
3 1 1
3 0 0 0 1 3 0
3 0 2 1 2 1 1
4 2 0 3 4 4 3 4 1
```

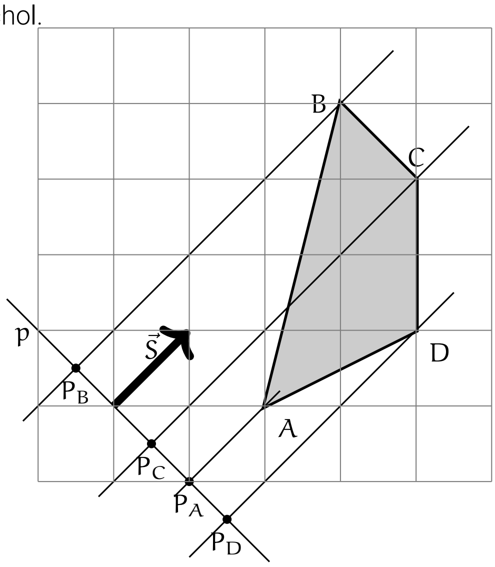


Jediné, co potřebuje důkaz, je že hodnoty c opravdu ohraničují hejna, že všechny možné hodnoty $c \in [c_l; c_r]$ budou mít průnik s hejnem a pro $c \notin [c_l; c_r]$ nebude průnik. Správnost algoritmu na spočítání maximálního počtu průniků je zřejmá.

Pro v důkazu se budu soustředit pouze na čtyřúhelník, ale samozřejmě platí pro všechna tělesa. Začnu s tím, že si popíšu jednotlivé vrcholy a zároveň nakreslím přímku p, která je kolmá na vektor $\vec{S}$ a prochází bodem $[0;0]$



Teď z každého vrcholu spustím kolmici k přímce p. A průsečík s ní si označím P_X kde X je vrchol.



Jak můžeme vidět, tak P_B a P_D mají jsou na krajích, tudíž mají maximální respektive minimální c. To platí proto, že parametr c určuje posunutí přímky. Toto posunutí se projeví i na průsečících s přímkou kolmou.

Dále můžeme vidět, že za P_B a P_D už žádné další průsečíky nejsou a hejno jimi končí. Zároveň vidíme, že jakýkoli bod na úsečce spojující body B a D bude mít c takové, že $c \in [c_l; c_r]$. Protože BC a CD jsou součástí hejna, takže při tažení kolmice někde skrz $P_B P_C$ a $P_C P_D$ bude procházet i skrz hejno. A protože mají obě úsečky společný bod, tak se jedná o souvislý interval.

Složitost

Načítání dat má lineární složitost, jak časovou, tak paměťovou. Samotný algoritmus má na zpracování pro každé z N hejn časovou složitost $O(p)$ a paměťovou $O(1)$, takže celková složitost zpracování je $O(v)$, kde v je celkový počet vrcholů v zadání, v čase a $O(N)$ v paměti. Seřazení neznámého vstupu pak má $O(N * \log N)$ složitost, při použití mergesortu nebo quicksortu. A paměťovou složitost $O(N)$ respektive $O(\log N)$. N proto, že se bude řadit array, který má pro každé hejno právě jeden interval. Následné spočítání průniků má složitost $O(N)$ v čase a $O(1)$ v paměti.

Výsledná složitost: čas - $O(N * \log N + v)$ a paměť - $O(N)$