Bruno Petrus - GBajkBA

3. Liečivá alpa - popis programu

Úvod

V tejto úlohe ide o nájdenie najlepšej možnej kombinácie použitia alpy bez mrhania. Riešenie sme hľadali tak, že sme si zobrali celý rad bojovníkov. Potom sme si určili miesto kde začať, a skúšali ako najefektívnejšie môžeme použiť alpu, ak začneme na danej pozícii. Na konci sme porovnali počet vyliečených bojovníkov pre jednotlivé štartovacie pozície a najväčší nájdený počet vyliečených bojovníkov je naše riešenie úlohy.

Dátové štruktúry

Počas výpočtu si musíme zapamätať vstupy. Rad zranených sme sa rozhodli uložiť v obyčajnej *C++ array.* Rozhodli sme sa tak preto, lebo náš algoritmus bude veľa krát iterovať cez celý rád bez toho, aby niečo menil. Na tieto podmienky sa podľa nás hodí *array.*

Algoritmus

Začneme tým, že uložíme jednotlivé stupy. V celom programe používame typ *long*, kvôli tomu že *int* nestačí. Ako sme už spomenuli rad bojovníkov sme uložili do *array,* ktorá je dynamicky alokovaná. Riešenie prebieha následovne:

Hlavný cyklus

1. Určíme si začiatočnú polohu *i.* Spočítame, koľko je súčet zranení od tohoto indexu *i* v rade po koniec rady.
2. Ak je tento súčet menší ako počet kvapiek v jednej ampulke, môžeme uzavrieť, že nemusíme ani pokračovať a výsledok je 0. Hocičo urobíme, vždy bude premárnená alpa.
3. Ak najväčšie zatiaľ nájdené riešenie(v prvej iterácii 0) je už väčšie ako počet bojovníkov od indexu *i*, môžeme takisto ukončiť program, lebo sme už našli najväčšie možné riešenie. Napríklad, ak je zatiaľ najväčšie nájdené riešenie 3 a sú 3 zranený bojovníci, nemá cenu začínať na indexe 1.
4. Ak je súčet zranení deliteľný počtom kvapiek v ampulke, vieme, že ani jedna kvapka nevýjde nazmar a riešenie (nemusí byť ešte najväčšie) pre index *i* je *celkový počet bojovníkov - i.* Môžme ísť na ďalšiu iteráciu s *i* o jedno väčšie (krok 1).

Ak sme doteraz nenašli žiadne riešenie, znamená to, že pre daný rad bojovníkov od indexu *i* až po koniec radu neexistuje riešenie. Musíme teda skúsiť od indexu *i* po iný koniec. Tento index, ktorý začína na konci, je *j.*

Vnútorný cyklus

1. Ak je súčet zranení od indexu *i* po *j*:
    a. menší ako počet kvapiek vo fľaške, nemá zmysel pokračovať v cykle, lebo by sme mrhali alpou. Inkrementujeme *i* a na krok 1. v hlavnom cykle.

b. deliteľný počtom kvapiek v ampulke bezo zvyšku, našli sme ďalšie možné riešenie a ak je väčšia ako doteraz naše najväčšie riešenie, uložíme si ho. Inkrementujeme *i* a skok na krok 1. v hlavnom cykle.
c. inkrementujeme *j.*

Už nám stačí vypísať len najväčšie možné riešenie.

Správnosť

Náš algoritmus je založený na princípe, že vyskúšame všetky možné podmienky, kde mohol Anur začať aj skončiť s liečením. Tým pádom musíme nájsť aj správne riešenie. Tento spôsob sme však vylepšili tým, že sme eliminovali prípady, keď je určité, že nemá zmysel iterovať ďalej.
Dva cykly sa nemôžu zacykliť, lebo hlavný cyklus bude prebiehať zakiaľ platí *i* < *počet zranených.* Vnútorný zakiaľ *j* >= *i*.

Komplexita

Ak dostaneme *n* počet bojovníkov, budeme musieť v najhoršom prípade prejsť *n* počet počiatočných indexov a zhruba (bude to menej) aj toľko konečných indexov (*i* a *j*). V najhoršom prípade bude komplexita programu zhruba $O(n * n)$ (*1). Vzhľadom na implementované testy a podmienky je však tento počet zredukovaný na menej.
Pamäťová zložitosť programu závisí čisto lineárne od vstupov.

---

*1 - v skutočnosti prejdeme omnoho menej, lebo každou iteráciou *i* bude vnútorný cykus menej krát spustený. Pre jednoduchosť, sme povedali, že to bude ale *n*.