Úloha č. 3 Lampy

Zpracoval Oskar Petr

Tato úloha je zpracována jako konzolová aplikace v jazyce C#, kde je vstupní bod umístěn v hlavním adresáři projektu v souboru `Program.cs`.

Validace zadání

Validace zadání se řeší nejdříve a funguje jako samostatný malý algoritmus. Pokud konkrétní zadání není proveditelné, tak je lepší toto zadání odhalit dříve, než ho řešit složitějším algoritmem a po chvíli zjistit, že není řešitelné (kvůli efektivitě algoritmu).

Počítejme se situací, kde všechny lampy budou aktivní a následně zjistíme, jestli každý důležitý bod je dostatečně osvícen.

1. Vyjádření rozsahů lamp r_i jako pole

Původní přístup, založený na procházení světelného rozsahu každé lampy, byl sice jednoduchý, ale časově náročný, s časovou složitostí $O(n^2)$.

Použil jsem tedy jiný způsob, a to konkrétně rozdílové pole (difference array). Rozdílové pole mi umožňuje zapsat světelné rozsahy lamp do jednorozměrného pole, kde jednotlivé položky reprezentují skoky ve světelnosti mezi důležitými body.

Podle standardní implementace rozdílového pole platí, že pro rozsah každé lampy se na jejím počátečním indexu tohoto rozsahu hodnota zvýší o 1 a na koncovém indexu + 1 sníží o 1. Výsledkem je pole zachycující změny ve světelnosti.

Začáteční a koncový index každého světelného rozsahu lze získat z jednoduchých výpočtů, které nejsou potřeba zde zmiňovat.

Ukázka pro $r_i = [2, 1, 2]$

[	0	0	0	0	]	inicializace pole
[	1	0	-1	0	]	$r_0 = 2$ (rozsah 0–1)
[	1	1	-2	0	]	$r_1 = 1$ (rozsah 1–1)
[	1	2	-2	-1	]	$r_2 = 2$ (rozsah 1–2)

2. Získání osvětlenosti na bodech i

Pokud máme vypočítané skoky v osvětlenosti, hodnotu v každém důležitém bodě i získáme jako součet prvních $i + 1$ prvků pole. To lze efektivně spočítat pomocí prefixového součtu, kde každá hodnota vzniká sečtením předchozího součtu a aktuální hodnoty z rozdílového pole.

Ukázka pro $r_i = [2, 1, 2]$

[	1	2	-2	-1	]	rozdílové pole
[	0	1	3	1	0	] prefixový součet (reálná osvětlenost)

První i poslední prvek v prefixovém poli zde slouží pouze jako pomocné hodnoty, takže je můžeme odstranit, aby v dalším kroku bylo porovnání jasnější.

3. Zajištění dostatečné osvětlenosti bodů i

Nyní už zbývá jen porovnat skutečnou osvětlenost chodby s požadavky na důležitá místa. To znamená ověřit, že hodnota na každé pozici i v prefixovém součtu reálné osvětlenosti je alespoň tak velká jako odpovídající požadovaná osvětlenost o_i.

Ukázka pro $r_i = [2, 1, 2]$ a $o_i = [1, 2, 1]$

[	1	3	1	]	reálná osvětlenost
[	1	2	1	]	požadovaná osvětlenost o_i
[	✓	✓	✓	]	zadání je proveditelné

Časová složitost validace

Časová složitost O se zde odvíjí od těchto třech kroků a jejich smyček, které všechny mohou v nejhorších případech dosahovat hodnoty celkového počtu lamp n.

$$O(3n) \Rightarrow O(n) \tag{1}$$

Nejmenší počet lamp

Moje řešení využívá greedy algoritmus, který vždy vybere lampu pokrývající co nejvíce bodů směrem doprava. Toho docílíme tím, že pro každou pozici i udržujeme seznam lamp, které ji mohou osvítit, a vždy vybereme optimum pro budoucí kroky.

1. Zpracování dat

Na začátku si nejprve zpracujeme data, tak aby se nám s nimi později lépe pracovalo. Vytvoříme si list lamp, kde pro každou lampu i budeme uchovávat kde začíná (`left`) i končí (`right`) její světelný rozsah, společně i s jeho pozicí v chodbě (i).

Tento seznam lamp setřídíme podle levého okraje `left` vzestupně. Pokud dvě lampy budou mít stejný okraj `left`, vybereme tu lampu, která dosvítí dále s jejím okrajem `right`, jelikož má větší pravděpodobnost osvítit co nejvíce lamp do budoucna. Tím zajistíme, že při procházení chodby zleva doprava budeme mít vždy přehled o lampách, které právě začínají osvětlovat aktuální pozici.

2. Prioritní fronta

Při procházení chodby zleva doprava udržujeme prioritní frontu lamp (max-heap), které mohou osvítit aktuální pozici i. Do fronty přidáváme lampy, pokud jejich levý okraj (`left`) je před pozicí i, a zároveň odstraňujeme ty, jejichž pravý okraj (`right`) již i nepokrývá. Každá lampa je do fronty přidána právě jednou a maximálně jednou i odstraněna, což zajišťuje efektivitu algoritmu.

Tato fronta je seřazena podle pravého okraje (`right`) – lampa s největším `right` je vždy připravena k výběru. Pokud zjistíme, že aktuální pozice nemá dostatek osvětlení, opakovaně vybereme lampu z vrcholu fronty. Každá vybraná lampa zvýší osvětlení nejen pro aktuální pozici, ale i pro následující části chodby. Tento postup zaručuje, že každý krok lokálně minimalizuje počet lamp potřebných pro budoucí pozice, což vede k optimálnímu řešení.

3. Vypočtení potřeb

Pro zjištění aktuální úrovně osvětlení na pozici i využíváme rozdílové pole, které udržujeme po celou dobu běhu algoritmu. Toto pole zaznamenává změny v osvětlení způsobené postupným rozsvěcením lamp.

Při zpracování pozice i nejprve přičteme hodnotu rozdílového pole na indexu i do proměnné `currentSum`. Tato proměnná udržuje prefixový součet - kumulativní osvětlení od začátku chodby až k pozici i. Hodnota `currentSum` tedy přímo odpovídá aktuálnímu osvětlení na pozici i.

Pro vypočtení současné potřeby osvětlení `needed` na pozici i nyní jen stačí od požadovaného osvětlení (o_i) odečíst `currentSum`.

4. Výběr lampy

Lampy vybíráme dokud platí podmínka `needed` > 0, protože do té doby nebude důležité místo i náležitě osvětleno.

Vybereme lampu z vrcholu prioritní fronty, která má největší pravý okraj (`right`). Tím zajišťujeme, že vybraná lampa maximalizuje osvětlení pro budoucí pozice. Přidáme index lampy do výstupního seznamu `selected`. Aktualizujeme rozdílové pole na pozici `left` vybrané lampy přičteme 1 a na pozici `right` + 1 odečteme 1.

Ted už jen zvýšíme `currentSum` o 1, jelikož se osvětlenost na pozici i zvýšila rozsvícením lampy. Také snížíme `needed` o 1, ze stejného důvodu.

Časová složitost

Na začátku algoritmu lampy seřazujeme podle kritérií `left` a `right`, čehož časová složitost dosahuje $O(n \log n)$. Následně začínáme smyčku se složitostí $O(n)$.

Ve smyčce přidáváme (popřípadě odebíráme) lampy z prioritní fronty pomocí max-heap funkcí `Enqueue()` a `Dequeue()` každé o složitostech $O(\log n)$, ale jelikož jsou ve `while` cyklu, který běží přes všechny lampy, dosahují tak $O(n \log n)$. Díky tomu, že každá lampa je přidána (popřípadě odebírána) maximálně jednou z fronty, hlavní smyčka nezvyšuje složitost fronty o další složitostní řád.

Ve výběru lamp používáme opět funkci `Dequeue()` a lampy vybíráme k-krát, kde k může dosahovat $k \leq n$. Tudíž složitost vybírání je dohromady $O(n \log n)$.

Rozdílové pole aktualizujeme maximálně n-krát, tudíž bude $O(n)$. Časová složitost tohoto greedy algoritmu je součtem těchto složitostí:

$$O(n \log n) + O(n \log n) + O(n) \Rightarrow O(n \log n). \tag{2}$$

Celková časová složitost

Každé zadání musí projít validací a případně také greedy algoritmem. V nejhorším případě je tedy celková časová složitost součtem jejich složitostí:

$$O(n) + O(n \log n) \Rightarrow O(n \log n). \tag{3}$$

Po zohlednění celkového počtu zadání t je výsledná časová složitost:

$$O(t \cdot n \log n). \tag{4}$$