Zadání:

Mějme na vstupu posloupnost kladných, navzájem různých celých čísel, seřazených vzestupně. Pro konstantu k na vstupu navrhněte algoritmus, který nalezne k-té chybějící číslo. Chceme optimálně rychlý algoritmus.

Protože uvažujeme všechna kladná celá čísla, pokud posloupnost nezačíná číslem 1, je 1 první chybějící číslo.

Předpoklady: Hodnota k je kladné celé číslo. Vstup je reprezentován jako pole n hodnot indexovaných $1, \ldots, n$ a dvě proměnné k a n.

Příklady vstupů:

- $k = 2$,$n = 5$, data: $1, 3, 4, 5, 7$, odpověď: 6
- $k = 2$,$n = 3$, data: $4, 5, 6$, odpověď: 2
- $k = 2$,$n = 5$, data: $1, 2, 3, 4, 5$, odpověď: 7

Plnohodnotné řešení by mělo obsahovat

- Stručný popis metody řešení (není slovní pseudokód)
- Pseudokód (nízkoúrovňový, stylu "hezkého" RAM)
- Ošetření speciálních případů
- Důkaz korektnosti
- Analýza časové složitosti (se zdůvodněním)

Pro zápis pseudokódu můžete používat veškeré přirozené konstrukce, které lze na RAM "simulovat" s konstantním zpomalením. Tj. smíte používat proměnné, přiřazení s komplexními výpočty, cykly, if-else podmínky s logickými výrazy, vrátit výstup pomocí return, ...

Řešení

Pomocí principu binárního vyhledávání najdeme dva prvky t.ž. v mezeře mezi nimi se nachází k-té chybějící číslo. Určíme počet prvků chybějících před levým okrajem mezery a odvodíme o kolik je k-té chybějící číslo větší než okraj.

Pozorování: Pro libovolný index i platí, že rozdíl $[i] - i$ je počet prvků chybějících nalevo od pozice i. Protože počet prvků chybějících nalevo je neklesající, umíme se v poli binárně orientovat.

Pozorování: Označme si k-té chybějící číslo X. Jakmile najdeme index I t.ž. $[I] < X < [I+1]$ (tedy nejvyšší index před kterým chybí méně než k čísel), potom X je $k - ([I] - I)$-té číslo po $[I]$, tedy $X = [I] + k - [I] + I = I + k$.

Pseudokód:

$L \leftarrow 0$, $P \leftarrow n + 1$
while $L < P - 1$ **do**
  $M \leftarrow \lfloor \frac{L+P}{2} \rfloor$
  if $[M] - M \geq k$ **then**
    $P \leftarrow M$
  else
    $L \leftarrow M$
return $L + k$

Konečnost a složitost:

Stejně jako na cvičení, podmínka cyklu je zvolena tak, že platí invariant $L < M < P$ při každém určení M. Uvážíme-li tedy $P - L$, tento rozdíl nutně klesá v každém cyklu, dle volby M navíc zhruba o polovinu. Po $\mathcal{O}(\log(n))$ krocích tedy cyklus skončí, celková složitost algoritmu je $\mathcal{O}(\log(n))$.

Korektnost:

Pro účel analýzy dodefinujeme hodnoty $[0] := 0$ a $[n+1] = +\infty$. Povšimněme si, že tím se výsledek nezmění a navíc přistupujeme pouze k prvku na pozici M, (dle invariantu $L < M < P$) a tak nikdy nepřistoupíme na index 0 nebo $n + 1$ (L pouze roste, P pouze klesá).

Označme si X správný výstup algoritmu. Dle zadání předpokládáme $k > 0$.

Binární vyhledávání udržuje invariant $[L] < X < [P]$. Na začátku cyklu invariant platí (po dodefinování $[0]$ a $[n + 1]$). Z pozorování výše, kdykoliv nastavíme P na novou hodnotu, před $[M]$ chybí alespoň k čísel a tedy nerovnost platí (rovnost nemůže nastat). Komplementárně nerovnost platí i pro nové hodnoty L. Po skončení cyklu máme tedy $[L] < X < [P]$ a $P \leq L + 1$, a tedy $P = L + 1$. Dle pozorování výše je výstup $L + k$ správná hodnota.

Alternativně můžeme na algoritmus nahlížet tak, že vyhledává index I. Z korektnosti bin. vyhledávání ze cvičení plyne, že I je korektně nalezen indexem L. (Možné hodnoty I jsou $0, \ldots, n$, tedy je korektní začít s $L = 0$).

Poznamenejme, že tím jsou vyřešeny i krajové případy. Pokud $X > [n]$, cyklus skončí s $L = n = I$, a $n + k$ je korektní vysledek dle pozorování. Pokud $X < [1]$ nebo $n = 0$, potom na vstupu chybí všechna čísla $1, \ldots, k$. Po skončení cyklu (resp. neproběhnutí) bude $L = 0 = I$, a tedy $L + k = k$ je korektní výstup.

Nejčastější problémy

- zaokrouhlování - kvůli zaokrouhlení byly některé hodnoty pro vyhledávání nedosažitelné. Typicky když se udržovala velikost kroku, a klesala příliš rychle.
- zacyklení - spousta řešení měla špatně podmínky ukončení vyhledávání. Typicky se "prostřední" index rovnal "dolnímu" indexu a algoritmus nastavoval dolní index do nekonečna.
- návratové hodnoty - často utekl index o ± 1, ale hlavně okrajové případy a fakt, že mezery mohou být větší než 1 číslo způsobily dost problémů
- složitost - do složitosti se z definice modelu RAM nepočítá načítání vstupu (což by bylo vždy $\mathcal{O}(n)$), vstup je vždy již v paměti v kroce 0. To je přirozené, v praxi můžeme dostat celá data odkazem. Poznámka na okraj, stejně tak se vstupní data nepočítají do prostorové složitosti. Tyto detaily formalismu jsou ale důležité pouze pokud je čas/prostor sublineární - takové alg. ale už na ADS nepotkáme.
- korektnost - a samozřejmě věc, kterou nemohu zdůraznit dostatečně, podceněná korektnost. Zamyšlení nad korektností mohlo předejít všem problémům výše.

Nejčastější komplikace

- indexace - spousta řešení se rozhodla použít indexování pole od 0. Nic proti tomu, to je v praxi určitě silně preferovaný způsob indexování. Nicméně posun indexů způsobil chyby ve velké části zmíněných řešení zatímco řešení s indexací od 1 stejné problémy neměly vůbec. Z praxe, třeba implementace binomiální haldy s indexací od 1 obvykle fungují bez debugování, a implementace s indexací od 0 trvá velmi dlouho zcela debugovat. Dvě řešení dokonce celé pole posunula, aby bylo indexované od 0 - což pouze způsobilo více problémů a zabilo časovou složitost.
- reprezetace intervalu pomocí jednoho pointeru - některá řešení si neudržovala oba konce intervalu. Vždy to dopadlo špatně. Typicky kvůli zaukrouhlování nebo zacyklení.
- speciální případy - většinu speciálních případů nebylo třeba řešit zvlášť. Stačilo spustit algoritmus se správnými výchozími hranicemi. Obecně platí dvě poučky, pokud to nezkomplikuje algoritmus, nechceme to řešit zvlášť. Pokud to vytvoří nové případy uvnitř algoritmu, chceme to odchytit před vstupem do hlavního těla algoritmu.
- duplicitní výpočty - celá řada řešení počítala nové hodnoty prostředního prvku hned na několika místech. Stejně tak v několika řešeních bylo několik různých návratových hodnot, které ale byly efektivně totožné (např. $L + k$ pokud $L > 0$ a k pokud $L = 0$).