algowiki.miraheze.org

Ügyfélszolgálat – Algowiki

6-8 perc

A feladat elvolvasása után a következő körvonalazódhat az olvasóban: a feladat nem más, mint a megadott folyamatot szimulálni. Hogy ezt sikeresen megtegyük, a következő dolgokat érdemes átgondolni: hogyan kezeljük a várakozó, kiszolgálás alatt lévő, és kiszolgáláson túlesett ügyfeleket, továbbá mi is történjen, amikor egy ember odakerül egy kiszolgálópulthoz, és amikor távozik onnan.

Korlátok:

* $1 \leq$ kiszolgálópultok száma $\leq 5 \cdot 10^4$
* $1 \leq$ ügyfelek száma $\leq 10^5$
* $1 \leq$ ügyfelek érkezési ideje, ügyintézés hossza $\leq 10^9$
* futásidő $\leq$ 0,8 másodperc
* memóriahasználat $\leq$ 32 MiB

ügyfelek tárolása[szerkesztés | forrásszöveg szerkesztése]

Nagyon egyszerű dolgunk van a kiszolgáláson túlesett ügyfelekkel, ugyanis a feladat szempontjából teljesen közömbös, hogy mi történik. A várakozó ügyfelek sem

okoznak komolyabb gondot, hiszen a bemeneten a kiszolgálás sorrendjében szerepelnek, az itt mutatott megoldás szempontjából ez a sorrend a legjobb. Az érdekesebb kérdés, a kiszolgálás alatt lévő ügyfelek hogyan kerülnek tárolásra. Az egyértelmű, hogy valami olyan adatszerkezetet érdemes használni, amelybe "gyorsan" lehet behelyezni, és kivenni elemeket, mivel minden ügyfél pontosan egyszer kerül be, majd kerül ki, s az ügyfelek száma potenciálisan 100000 is lehet. A szimuláció szempontjából valójában egy adott pillanatban csak azok a kiszolgálás alatt lévő ügyfelek érdekesek, akik éppen az adott pillanatban hagyják el az ügyfélszolgálatot. Ha minden kiszolgálás alatt álló ügyfélhez hozzárendeljük azt az időpontot, amikor elhagyja az épületet, akkor értelemszerűen egy adott pillanatban azokat szükséges vizsgálni, akik távozási ideje a minimális. Erről eszünkbe juthat egy ismert adatszerkezet: a prioritási sor. Ha egy prioritási sorban tároljuk a kiszolgálás alatt álló ügyfeleket, akkor $O(N \cdot \log N)$ futásidejű komplexitással kezelhető az ügyfelek megérkezése egy kiszolgálópulthoz, és távozásuk.

események kezelése[szerkesztés | forrásszöveg szerkesztése**]**

ügyfél távozása[szerkesztés | forrásszöveg szerkesztése**]**

Egy ügyfél távozásakor mindenképpen valamilyen módon fel kell szabadítani az általa eddig elfoglalt kiszolgálópultot, azonban ennek módja attól függ, hogy hogyan is tartjuk azt számon. Erről az *ügyfél érkezése* részben található több

információ. A tárolás módjától függetlenül a feladat első kérdésére (munkanap végének időpontja) akár itt is számolhatjuk a választ.

ügyfél érkezése[szerkesztés | forrásszöveg szerkesztése**]**

A megérkezett ügyfelekkel akkor szükséges elkezdeni foglalkozni, amikor van szabad kiszolgálópult, ugyanis ha nincs, akkor várakoznak a feladat leírása alapján. Egy ügyfél pulthoz jutásakor tárolható el, hogy melyik pultnál szolgálták ki, és hogy mennyit várakozott.

A leglényegesebb kérdés, ami itt felmerül, az az, hogy hogyan is tároljuk a szabad pultokat, hogy "gyorsan" megtaláljuk a legkisebb sorszámú szabad pultot, és hogy egy pult felszabadulását is "gyorsan" az adatszerkezetbe tudjuk menteni.

első ötlet[szerkesztés | forrásszöveg szerkesztése**]**

Első, legegyszerűbb ötletünk lehet egy logikai értékeket tároló tömb (vagy pl. C++ esetén std::bitset), azonban gyorsan észrevehető, hogy noha konstans időben lehetséges egy pult felszabadítása, a legkisebb sorszámú szabad pultot meghatározni nem elégségesen gyors, hiszen egy lineáris keresés szükséges hozzá, ami a korlátok mellett nem használható.

Esetleg megpróbálhatjuk finomítani ezt az ötletet úgy, hogy számon tartunk egy "jelölt" pultot, és mindig attól indítjuk a keresést. Amikor felszabadul egy pult, akkor a jelöltet frissítjük a jelenlegi jelölt és az éppen felszabadított pult

sorszámának minimumára. Egy pult elfoglalásakor pedig beállítjuk az éppen elfoglalt pult sorszáma plusz egy értékre. Így garantálható, hogy a jelölt pult "előtt" nem lehet szabad pult, és bemenettől függően sokszor a jelölt lesz éppen a megfelelő pult. Ezzel jelentős sebességnövekedés érhető el, a futásideje semmiképpen sem rosszabb, mint a teljesen naiv megoldásé, azonban nem elégségesen gyors minden bemenetre.

második ötlet[szerkesztés | forrásszöveg szerkesztése**]**

A boolean tömb ötletet nem szükséges teljesen kidobni, mindössze arra kell valami megoldást találni, hogy hogyan lehet "gyorsabban" megtalálni a legkisebb indexű igaz értéket. Mivel a lineáris keresés nem elégségesen gyorsan, gondolkodhatunk esetleg, hogy valamilyen formában bináris keresés alkalmazható-e. Például, ha minden pult mellett azt is tárolnánk, hogy előtte hány szabad pult van, akkor remekül alkalmazhatnánk bináris keresést, és megtalálhatnák a megfelelő pultot $O(\log N)$ aszimptotikus komplexitással, viszont a pult felszabadulásakor minden utána következő pultnál tárolt darabszámot növelni szükséges eggyel, így azonban nem lett a program gyorsabb, sőt, valójában lassítottunk rajta. Mindazonáltal, ha arra találnák megoldást, hogy hogyan tudjuk csökkenteni a felszabadításkor frissítendő adatok számát, akkor lenne egy elégségesen gyors megoldásunk. Eszünkbe juthat, hogy pl. valamilyen bináris fa adatszerkezetben, amelynek levelei a logika tömb elemei, és csúcsaiban a részfa

összege található, is remekül tudnák bináris keresést alkalmazni, és a frissítendő adatok száma a fa magasságával lenne arányos, amely $O(\log N)$ futásidejű lekezelését tenné lehetővé egy pult felszabadulásának.

harmadik ötlet[szerkesztés | forrásszöveg szerkesztése**]**

Ha nem jutna eszünkbe elgondolkodni bináris fákon, és csak arra koncentrálunk, hogy nekünk csak a legkisebb sorszámú pult számít, akkor visszagondolva az éppen kiszolgálás alatt lévő ügyfelek tárolásának módjára, rájöhetünk, hogy pontosan ugyanazt elsüthetjük itt is. Ha egy prioritási sorban tároljuk a szabad pultokat, akkor (implementációtól függően, de legtöbb esetben) $O(\log N)$ komplexitással kezelhető egy pult elfoglalása és felszabadulása is.

további meggondolandók a szimuláció folyamatával kapcsolatban[szerkesztés | forrásszöveg szerkesztése**]**

Az időértékek mérete miatt nem lehet minden időpontot külön feldolgozni, a következő releváns időpontra szükséges ugrani; amelyet könnyen megtehetünk, ugyanis az az időpont a következő ügyfél érkezésének (ha van még szabad pult), és a következő távozásnak minimuma.

Minden szimulációs "kör" végén számolhatjuk, hogy éppen hányan várakoznak, s ezáltal kaphatjuk meg a választ a harmadik kérdésre (váróteremben egyszerre várakozó emberek legnagyobb száma). Ezt lineáris időben

megtehetjük, ugyanis a várakozók a bemenetnek egy [a; b] indexek intervallumába eső ügyfelek. Amikor egy ügyfél sorra kerül, akkor az "a" index nő; amikor egy ügyfél érkezik, akkor pedig a "b" index nő. Ezzel a lényegében *two pointers* technikával elégségesen gyorsan számolható ezen részfeladat megoldása.