1. nagy házifeladat

2021 Október 2.

1 Bevezetés

A szomszédos Spar üzletben új sorbanállási rendszerrel kísérleteznek, hogy megnehezítsék a vásárlók életét. A feladatod az lesz, hogy készíts egy szimulációt, ami megmutatja hogyan működne a tervezett rendszer.

2 A vásárlók

- 0. Mindenki becsületes, betartja az alábbi szabályokat.
- 1. Az egyes vásárlók t terméket vásárolnak $0 < t < 30$ között, a kasszánál a termékek leolvasása egység idő alatt történik.
- 2. Minden vásárló különböző mértékben siet. A 0. pont jegyében kitalálnak maguknak egy p prioritást $0 < p < 1000$ között (p egész szám), amit felírnak a homlokukra.

3 A kasszák

- 3. Van n db kassza ($0 < n < 100$), amik egyenként $m_1, m_2, ...m_n$ embert tudnak fogadni ($0 < m_x < 100$). Ezek a kasszák "k-prioritásos sorok".
- 4. Minden k-prioritásos sornak van egy k paramétere, ahol 1¡k¡2 közötti valós szám.
- 5. Egy vásárló csak akkor engedi maga elé a mögötte lévőt, ha annak legalább k-szoros prioritása van, mint önmagának.
- 5.1. Pl.: Jön egy 100-as prioritású ember egy $k = 1.5$ sorba. Ha előtte 66-os ember áll, akkor az előre engedi, ha 67-es, akkor már nem. $k = 1$ a hagyományos prioritásos sor.

4 A vásárlás menete

- 6. A fizetni akaró vásárlók egy "globális", hagyományos sorba állnak, innen mennek ahhoz a kasszához, ahol van hely. Ha 1 lépésben több hely is felszabadult, akkor oda mennek, ahol azt remélik, hogy előbb végeznek.
- 7. Ha van üres hely bármelyik kasszánál, akkor a nem-teli kasszák közül oda megy, ahol a jelenlegi állapot szerint leghamarabb kerül sorra, figyelembe véve a kasszáknál álló vásárlók prioritását és terméklistáját. Ha több egyforma van, akkor bármelyiket választhatja.
- 7.1. Praktikusan kelleni fog egy függvény a k-prio sorhoz, ami megmondja egy vásárlóra, hogy mikor kerülne sorra a jelenleg kasszánál sorbanállók alapján.
- 8. Ha valaki végez a vásárlással, akkor mielőtt a következő vásárló beállhatna a kasszához, minden kasszánál álló prioritása növekszik t_0-val, ahol t_0 a távozó által vásárolt termékszám. Ilyen módon a prioritás elérheti az 1000-et, de nagyobb nem lehet.
- 8.1. A sorokban mozgások a termékleolvasáshoz képest elhanyagolhatóan rövid idő alatt történnek meg (azonnal), ezért t_0 megegyezik a várakozási idővel, amit a távozó vásárló okozott. (ld. 1.)

5 A feladatmegoldás formája

- 9. Miután minden vásárló távozott meg kell tudnunk mondani, hogy az érkezés sorrendjében számozva őket, milyen sorrendben távoztak. Egy fájlba adjuk vissza a sorszámokat, amiket a "n" karakter válassza el.
- 10. Az egyes szimulációs esetek adatait fájlból kell beolvasni, a fájl formátuma a kövtkező:
- 10.1. Az első sor egyetlen egész számot (n) tartalmaz, a kasszák számát.
- 10.2. A következő (n) sorban egy valós szám van 1 és 2 között ez követi egy egész szám a kettő között szóköz karakter.
- 10.2.1. A valós szám 1 és 2 közötti értékű, ezek a kasszák (k) paraméterei. A szám tizedes PONTot használ.
- 10.2.2. Az egész szám az adott kasszánál maximum sorbanállók száma (m_x).
- 10.3. Az ezt követő ($n + 2$.dik) sorban egyetlen egész szám (m) van, a vásárlók száma. Ez maximum 1.000.000 lehet.
- 10.4. Az ezt követő (m) db sorban a vásárlók adatai találhatóak a következő formátumban.

- 10.4.1. Szóközzel elválasztva 3 egész számot tartalmaz minden sor.
- 10.4.2. Az első szám egy azonosító szám, hogy hányadik vásárlóról van szó. Ez alapján kerülnek be a globális sorba és ezeknek az azonosítóknak kell egy permutációját visszaadni eredményként.
- 10.4.3. A második szám a (p) prioritás, ami alapján rendezi őket a k-prioritásos sor.
- 10.4.4. A harmadik szám a vásárolni kívánt (t) termékek száma.

6 Egyéb megjegyzések

A kiadott kódban van minta bemeneti fájl és minta teszt. A nagyházik gyakorlatvezetők által lesznek pontozva. Bármilyen teszt fut le sikeresen az SVN mögött, az a nagyházi esetén nem garantál semmilyen pontot, de jelzi, ha biztosan nem tökéletes az implementáció. Lehet írni extra segédfüggvényeket az osztályokon belül, de ezek legyenek mind **private**-ak. A nem forduló kód 0 pont. A pontok eloszlása *hozzávetőlegesen* a következő:

- Queue.cpp : 20%
- Shop.cpp : 30%
- KPriorityQueue.cpp : 50%

Pontozás kapcsán a változtatás jogát fenntartjuk.