Batch System Roulette

Alapötlet

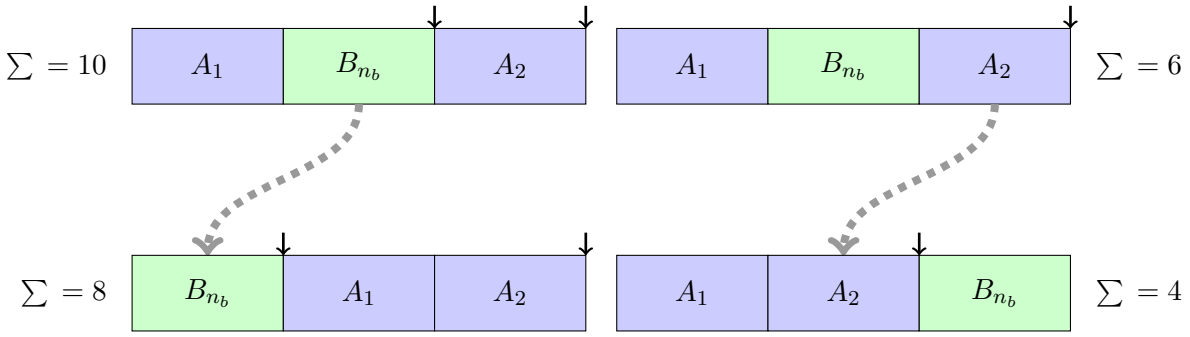
„Általánosságban elmondható, hogy első lépésként célszerű megvizsgálni a feladatban leírt világ működését, a lehetséges megoldások felépítését, majd törvényszerűségeket keresni” – írtuk az előző feladatnál. Esetünkben ez most annak végiggondolását jelenti, hogy miként is nézhet ki egy *optimális* ütemezés.

Mivel minden felhasználónál kizárólag az utolsó munka befejezési időpontja járul hozzá az átlaghoz, nem tűnik optimálisnak, hogy két (vagy több) felhasználó munkáit keverve (interleave-elve: egyszer az egyiktől egy munkát, aztán a másiktól egyet, stb.) hajtsuk végre, hiszen így csak egymás utolsó munkájának befejezését késleltetik.

Ez valóban igaz. Tegyük ugyanis fel, hogy az állítólagos optimális ütemezésünkben egy „A” felhasználó munkái közé más felhasználók munkái ékelődtek. Két esetet különíthetünk el:

a.) Van olyan közbeékelődött munka, amely egy másik felhasználó utolsó munkája. Legyen „B” az a felhasználó, akihez (időben) az első ilyen munka tartozik. Ekkor ezt (és „B” esetleges további közbeékelődött munkáit) mozgassuk az ablak elejére, azaz épp „A” felhasználó első munkája elé. Ekkor sem „A”, sem az esetlegesen közbeékelődött többi felhasználó befejezési ideje nem változik, „B” befejezési ideje csökken, így az átlagos befejezési idő is. Ellentmondás, az eredeti ütemezés nem lehetett optimális.

b.) Egyik közbeékelődött munka sem utolsó. Ebben az esetben pedig „A” felhasználó munkáit a vizsgált ablak legelejére mozgatva az ő befejezési ideje csökken, a többieké nem változik, így jutunk ellentmondásra.



(a) „B” utolsó munkája, B_{n_b} ékelődött közbe.

(b) Egy nem-utolsó munka ékelődött közbe.

1. ábra. Legegyszerűbb ütemezések a két eset szemléltetésére. Minden munka 2 időegység hosszú, a nyilak az adott felhasználó utolsó munkájának végét jelölik.

Tehát egy adott felhasználó munkái minden optimális megoldásban folytonosan egymás után, egy *blokkban* kerülnek végrehajtásra, melynek hossza tehát az adott felhasználó munkáinak összhossza. Mit tudunk vajon a blokkok sorrendjéről?

Tekintsünk egy blokkot egy pillanatra egyetlen hosszú munkának. Így az ütemezéselmélet egyik alapfeladatát kapjuk, miszerint m darab munka átlagos befejezsési időre optimalizált ütemezését keressük. A fentihez hasonlóan, indirekt módon itt is belátható, hogy az optimumot akkor érjük el, ha a munkákat (itt: blokkokat) hosszúság szerinti növekvő sorrendben végezzük el. Azonos hosszúság esetén tetszőleges a sorrend.

Összegezve azt kaptuk, hogy jelen feladatnál az egyes felhasználókhoz a munkákat közvetlenül egymás után, egy *blokkban* kell futtatni, ezen blokkok pedig növekvő összmunkahossz szerinti sorrendben kell hogy kövessék egymást. Több optimális megoldás két szinten keletkezhet: azonos hosszú blokkok sorrendje tetszőleges, illetve egy blokkon belüli a munkák sorrendje tetszőleges (hiszen egy blokk épp egy felhasználó munkáiból áll definíció szerint, így csak a blokk végének időpontja számít).

Algoritmus

A feladat szövege szerint minden lehetséges permutáció egyformán valószínű, tehát elvileg jó megoldás lenne, hogy legeneráljuk a fentieknek megfelelő összes lehetséges ütemezést, majd vesszük az egyes ütemezéseknél az egyes munkákra a befejezési idők átlagát. Sajnos ez sok esetben kivitelezhetetlen, gondoljunk csak pl. arra az esetre, amikor egyetlen felhasználó 50 munkáját szeretnénk az 50! módon sorbatenni.

Helyette vegyük észre, hogy a permutációk párokba állíthatóak vízszintes tükörképükkel, így pl. egy „A” felhasználó A_i munkája pontosan annyiszor előzi meg A_j-t, mint amennyiszer mögötte van: az esetek felében. Ugyanez elmondható a blokkok szintjén is: „A” blokkja pontosan annyiszor van a „B” felhasználó (ugyanolyan hosszú) blokkja előtt, mint ahányszor mögötte, így viszont „A” minden A_i munkája is az esetek felében egy adott B_j munka előtt, a másik felében mögött van.

Összegezve tehát azt mondhatjuk, hogy pl. egy „A” felhasználó A_i munkáját minden optimális ütemezésben megelőzi minden olyan munka, amely olyan felhasználóhoz tartozik, aki blokkjának hossza rövidebb az „A” blokkjánál, illetve az esetek felében megelőzi minden olyan más munka, ami „A”-val azonos hosszúságú blokkba tartozik (beleértve „A” blokkját is).

Az algoritmus tehát mindössze annyi, hogy minden felhasználóra kigyűjtjük a munkáit, így meghatározzuk a blokkokat és hosszukat, majd ez alapján minden munkára a fentiek szerint összeadjuk a fixen előtte lévő munkák összeidejét (rövidebb blokkban van), az esetek felében előtte lévő munkák összidejének felét (az azonos hosszú blokkokból), illetve az adott munka időigényét (hiszen befejezési időt keresünk).

Komplexitásvizsgálat

A blokkok hosszainak számításához célszerű egy kiegyensúlyozott (pl. piros-fekete) fát használni, melyben a felhasználónevek a kulcsok, a hozzájuk tartozó jelenlegi blokkhosszok pedig az értékek. Így a munkák csoportosítása $O(n \cdot logn)$ időben, az összeg kiszámítása pedig (naiven, egymásba ágyazott ciklusokkal) $O(n^2 \cdot logn)$ időben megoldható.