Eötvös Loránd Tudományegyetem

Informatikai Kar

Programozási Nyelvek és Fordítóprogramok Tanszék

Osztott rendszerek specifikációja és implementációja

IP-08bORSIG

Dokumentáció a *3.* beadandóhoz

`Gonda Dávid`
`BIXUOS`

2019. május 23.

1. Kitűzött feladat

A bemeneti fájlban (world.map) elsőként egy hexagonális (hatszögletű) rácson alapuló világ térképe található. A térképnél esetén feltesszük, hogy annak "hasznos része" az, amit a fájlból tudunk beolvasni, az összes, fájlban nem szereplő mezőre a világot körülölelő tengerként tekintünk.

A térkép hasznos részéhez meg van adva hány sorból és hány oszlopból áll (N x M), ezt követik az egyes mezők azonosítói.

A korábbiak alapján az összes civilizáció már választott magának fővárost, így ezeket is a térképből tudjuk kiolvasni (11-es szám jelzi a városok helyét, összesen C db. ilyen van).

A térképhez tartozó adatok után olvashatóak a kereskedők információi. Elsőként egy egész érték, a kereskedők száma (K >= 0), utána összesen K sorban egy-egy kereskedőhöz tartozó adatokat lehet beolvasni.

A kereskedelmi szempontokból fontos nyersanyagok a fa, arany, vas, hal és gabona (azok a mezők, amikből ezeket lehet szerezni: FIELD::FOREST, FIELD::GOLD_MINE, FIELD::IRON, FIELD::LAKE, FIELD::WHEAT).

A városok elsődleges szempontja, hogy mind az 5 típusú nyersanyagból legalább 1 db legyen a birtokukban. A kezdőállapotukat a körülöttük 2 sugárban lévő mezők alapján lehet megállapítani.

A kereskedés módja:

- Meg kell nézni, hogy van -e olyan áruja (készleten) a kereskedőnek, amiből a város hiányt szenved (nincs neki egy se).
- Ha nem, akkor ez a város nem kereskedik.
- Ha igen, akkor meg kell nézni, van -e olyan másik, kereskedés szempontjából fontos nyersanyag, amiből kellő többlete van a városnak (legalább x+1 db, ahol x jelzi a kereskedő mohósági indexét).
- Ha nem, akkor ez a város nem kereskedik.
- Ha igen, akkor x:1 arányban megtörténik a csere (pl. x egység fát ad a város, ami levonásra kerül a készletéből, és gyarapszik egy vassal, a kereskedőnél pedig a fa mennyisége nő x-szel, a vaskészlet pedig eggyel csökken).
- Amíg a város tud kereskedni, addig azt megteszi.
- Ha már nem kereskedik tovább a város, akkor a kereskedő a meglátogatja a következő várost, amelyik ugyan ilyen alapon fog működni.
- Egy csere során egyszerre csak egy típusú nyersanyagot ad a város, tehát nincs vegyes trade.

A feladatunk meghatározni azt, hogy az egyes kereskedők, miután végigmentek az összes városon milyen nyersanyagokkal rendelkeznek! Az így kiszámított eredményt (az egyes kereskedők megmaradt árukészletét) írjuk ki az output.txt fájlba (a bemenet sorrendjében)!

Tekintve, hogy egy-egy kereskedés sok időt vehet igénybe, ezért a kereskedők nem egyesével, egymást bevárva fogják végigjárni a városokat, hanem egymás után, amint "felszabadult" egy város egyből mennek a következőbe (O(C*K) helyett O(C+K)). Programozási szempontból ez az adatcsatorna tételének fog megfelelni!

Az inputfájl felépítése az alábbi:

Az első sorban két nemnegatív egész szám olvasható (jelölje ezt N és M), ezt követően egy N x M méretű, azaz N sorból és M oszlopból felépített világ térképe.

Az ezt következő N sor a térkép egy-egy sorának reprezentációját tartalmazza, pozitív egész számokat az [0..11] intervallumból szóközzel tagolva.

Ezt egy nemnegatív egész szám követi (jelöljük K-val), ami azt mondja meg, összesen hány kereskedő fogja végiglátogatni a városokat.

Az ezt követő K sor mindegyikében 5+1, azaz 6 egész számot lehet olvasni, ebből az első 5 azt jelzi, hogy a kereskedőnél kezdetben a "fontosnak" ítélt nyersanyagokból mennyi található (kezdő árukészlet), a következő szám pedig azt jelzi, hogy az adott kereskedővel milyen arányban lehet kereskedni, azaz hány db/egység (azonos típusú) árut kell a városnak odaadnia ahhoz, hogy a kereskedő egy árut átadjon a saját készletéből (mohósági index (mi.)). A 1 0 4 3 5 1 sor jelentése így tehát pl.: 1 fa, 0 arany, 4 vas, 3 hal, 5 gabona, 1:1 arányú cserelehetőség.

Feltehetjük, hogy az input fájl a fent leírtaknak megfelelően van kitöltve, és nem található benne pl. negatív, valós vagy olyan szám, ami nem felel meg mezőknek. A kezdeti mennyiségek mindegyike nemnegatív szám, azaz >= 0. A kereskedők minden esetben legfeljebb 1:1 arányú cserét biztosítanak, vagy ennél kedvezőtlenebbet. Ingyen egy kereskedő sem fog adni árut, és negatív értéket sem lehet olvasni (mi. >= 1 minden esetben). A program olvassa be a térképet (a már implementált módon), illetve a kereskedőkhöz tartozó adatokat, majd hívja meg calculate_trades(..) metódust. A program kimenete az egyes kereskedők árukészlete azután, hogy az összes várost meglátogatták abban a sorrendben, ahogy elindultak!

Egy példa bemenet (world.map):

9 13
4 5 2 5 2 7 9 5 8 2 5 0 7
0 2 2 10 2 3 10 2 7 4 11 3 9
9 0 8 4 4 1 9 2 7 7 5 6 6
5 5 3 4 5 0 8 2 9 6 2 8 1
10 10 9 10 8 10 2 10 2 7 10 4 6
3 10 2 10 5 11 5 10 2 6 7 6 11
10 6 8 8 5 9 7 4 1 7 6 0 0

7 9 3 6 5 3 3 4 5 9 6 8 1
6 3 5 2 8 6 6 10 5 7 7 10 11
3
4 1 1 4 1 3
4 2 5 2 3 3
4 0 1 0 2 2
Az ehhez tartozó elvárt kimeneti fájl (output.txt):
4 1 4 3 1
4 2 5 2 3
3 0 1 2 2

2. Felhasználói dokumentáció

2.1. Rendszer-követelmények, telepítés

A programunk több platformon is futtatható, dinamikus függősége nincsen, bármelyik, manapság használt PC-n működik. Külön telepíteni nem szükséges, elég a futtatható állományt elhelyezni a számítógépen.

2.2. A program használata

A program használata egyszerű, külön paramétereket nem vár, így intézőből is indítható. A futtatható állomány mellett kell elhelyezni a *world.map* nevű fájlt, mely a bemeneti adatokat tartalmazza, a fenti specifikációnak megfelelően. Figyeljünk az ebben található adatok helyességére és megfelelő tagolására, mivel az alkalmazás külön ellenőrzést nem végez erre vonatkozóan. A futás után az alkalmazás mellett található *output.txt* fájl tartalmazza a kapott eredményt, azaz a kereskedők végleges készleteit.

3. Visszavezetés az adatcsatorna tételére

3.1. Az absztrakt tétel és a finomított specifikáció

$$\begin{array}{rcccccccccc} A & = & \underset{x_0}{Ch(T)} & \times & \underset{\overline{x_0}}{Ch(T)} & \times & \ldots & \times & \underset{x_n}{Ch(T)} & \times & \underset{\overline{x_{n+1}}}{Ch(T)} \\ B & = & \underset{x'_0}{Ch(T)} & \times & \underset{\overline{x_0}'}{Ch(T)} & \times & \ldots & \times & \underset{x'_n}{Ch(T)} & \times & \underset{\overline{x_{n+1}}'}{Ch(T)} \end{array}$$

- $Q ::= (x_0 = \overline{x_0} = x'_0 = \overline{x_0}' = D \wedge x_{n+1} = \overline{x_{n+1}} = x'_{n+1} = \overline{x_{n+1}}' = <>)$
- $Q \in INIT_h$, ahol $h = x'_0 \overline{x_0}' x'_n \overline{x_{n+1}}'$

- $FP_h \Rightarrow \overline{x_{n+1}} = F(\overline{x_0}') = F(D)$
- $FP_h \Rightarrow \forall i \in [0..n] : (f_i(\overline{x_i} - x_i) = \overline{x_{i+1}}) \in inv_h$
- $Q \in TERM_h$
- $(\overline{x_0} = \overline{x_0}' = D) \in inv_h$
- Variáns függvénynek tekintsük az $(|x_0|, |x_1|, \ldots, |x_n|)$ számot úgy, hogy a rendezett n-es elemeit helyiértékkel súlyozzuk, ahol az egyes csatornákon lévő elemek száma rendre egy $m + 1$ alapú számrendszerben felírt szám számjegyeinek felelnek meg.

3.2. A csatornán áramló típusok leírása

Az adatcsatornán az ábrázolt T típus megfelel a következőnek:

$$T = \mathbb{R}^6$$
$$trader$$

3.3. Az egyes függvénykomponensek működése

Esetünkben $i \in [0..n] : f_i$ a kereskedést fogja végrehajtani. Először összeállítunk egy raktárkészletet a város körüli hasznos nyersanyagokból (2 sugarú körben), majd elkezdünk "tárgyalni" a kereskedőkkel. Ha nem tudunk megállapodni (mohóságuk nagyobb, mint a készletünkből bármi, vagy nincs semmilyük amivel szolgálni tudnának), egyből megyünk a következő kereskedőre, az aktuálisat pedig útnak eresztjük (továbbítjuk a csatornán). Amennyiben találunk olyan nyersanyagot, ami kell a városnak, és van a kereskedőnél, és olyat is, amivel tudunk szolgálni a kereskedőnek, megtörténik a csere (a kereskedő mohóságának függvényében).

4. Fejlesztői dokumentáció

4.1. Megoldási mód

A kódunkat logikailag két részre bonthatjuk, egy fő-, illetve több alfolyamatra. A fő folyamatunkat a `main()` függvény fogja megvalósítani, mely beolvassa az inputként kapott fájl tartalmát, majd egy $N \times M$ mátrixot (mapot), és egy vektort tölt fel annak adataiból. A mapunk *calculate_trades* metódusa fogja megvalósítani a feladat megoldását a beolvasott vektor (*traders*) segítségével.

4.2. Implementáció

Az említett mátrixot `std::vector<std::vector<FIELD>> map_;` típussal fogjuk megvalósítani a `Map;` osztályonon belül. A `calculate_trades();` a következők szerint jár el. A `find_cities();` metódus segítségével megkeresi az aktuális városokat (ezek a *cities_*

változóban lesznek tárolva), majd létrehozunk egy csatorna vektort (*pipes*, és egy threadeket tároló vektort (*working_cities*). Ezután elindítunk threadekre helyezve *city_trade();* függvényeket (ennek működése később), amiket a *working_cities;* vektorba rakunk. Ha ez megvan, a csatorna elejére ráhelyezzük a *traders;* vektor elemeit. Ha ez is megvan, kiíratjuk az *output.txt* nevű fájlba az eredményeket a csatorna végéről. Végül megvárjuk az összes threadet.
A threadekben lévő *city_trade()* a következőképpen működik. Összeszedjünk egy vektorban a megadott város 2 sugarú körében lévő mezőket. Ezeket egy *std::array<int, 5> resources;* nevű tömbbe filterezzük úgy, hogy csak azok a mezők maradjanak benne, amikkel lehet kereskedni. Ezután a kereskedők számáig (*count*) ciklust indítunk, amiben leszedjük a csatornában érkező kereskedőt. Utána, ha van a városnak valamilyen nyersanyaga (*have_sellable()*), akkor keresünk egy árut, amire szüksége van a városnak (azaz 0 van belőle). Ha találtunk ilyet, keresünk egy olyan nyersanyagot, ami a kereskedőnek kell, és van *mi*-nél (*mi* a kereskedő mohósága) nagyobb számú nyersanyagja a városnak, akkor megtörténik a csere (a csatornáról leszedett *in* megfelelő helyéről leveszünk egyet (ez az a nyersanyag amit a város akar), és hozzáadunk *mi*-t (az a nyersanyag, amit a kereskedő akar). Ugyanígy eljárunk a város nyersanyagát tároló vektorral (*resourses*).
Végül átadjuk a csatornának az aktuális kereskedőt.

4.3. Fordítás menete

A programunk forráskódját a `main.cpp` fájl tartalmazza. A fordításhoz elengedhetetlen egy `C++11` szabványt támogató fordítóprogram a rendszeren. Ehhez használhatjuk az *MSVC*, *g++* és *clang* bármelyikét. A fordítás menete (4.9.2-es verziójú g++ használata esetén) a következő: `'g++ main.cpp types.hpp pipe.hpp impl.cpp -std=c++11'`. A speciális, `-std=c++11` kapcsoló azért szükséges, mert alapértelmezés szerint ez a verziójú fordítóprogram még a régi, C++98-as szabványt követi, melyben a felhasznált nyelvi elemek még nem voltak jelen. Unix alapú operációs rendszer esetén szükség lehet a `-lpthread` fordítási kapcsolóra is a párhuzamos könyvtárak linkelése miatt.

4.4. Tesztelés

A számítógépemben 6 mag van (AMD FX-6100). A 6 magból 5-öt kikapcsolva (időben) szekvenciális lefutást tudtam előállítani. Különböző input méretek esetén a futási időket a következő két táblázat szemlélteti.
A tesztbemeneteket egy python szkrpit segítségével hoztam létre (specifikációnak megfelelnek). Látható, hogy gyorsabb a párhuzamos futása a programnak.
Az időmérést az std::chrono osztály segítségével végeztem. Az eredmények átlagolva vannak, milliszekundumban értendőek.

Térképméret	Kereskedők száma	1 mag	6 mag
5x5	20	25ms	3ms
10x10	100	50ms	22ms
15x15	500	150ms	95ms
20x20	1000	856ms	750ms