HF06 - Ritkás Mátrix Hasítótáblával

Kiíró: Vajna Levente

Adatszerkezetek és Algoritmusok, 2024 ősz

Leadási határidő: 2024.11.19. 23:59

Ritkás mátrixok esetén adott nagy méretű mátrixban csupán kevés mező van tényleges értékekkel kitöltve. 2 Dimenziós tömbbel a tárolásuk nagyon pazarló, hiszen az előre definiált (default) értéket fölösleges annyiszor eltárolni. Például egy $n \times m$ -es ritkás mátrix $n, m = 1000$ esetén ha a ténylegesen kitöltött elemek száma pl. 1000, akkor a tároláshoz szükséges hely mindössze $\sim 0.1\%$-a hordoz lényegi információt, a többi helyen default értékek duplikátumait tároljuk csupán. A ritkás mátrixokat semmiképp sem érdemes a megszokott módon eltárolni, mert úgy túl sok memóriát igényel, és redundáns módon tároljuk el a default értéket. Azzal, hogy csak egyszer tároljuk el a default értéket, nagyon sok memóriát spórolunk meg, és annak érdekében, hogy a gyorsaságot is garantálhassuk, hasításos technikákat alkalmazunk.

Igaz, hogy a kulcsütközések elkerülése érdekében minimális időt veszítünk keresés esetén, PF fával ezt is tovább minimalizálhatjuk[1]. Annak ellenére, hogy hasításos technikákat tipikusan a keresés gyorsítására használjuk (nagyobb memóriahasználat árán), jelen esetben nagyban tudja csökkenteni a memóriaigényt.

Feladat:

A feladat egy ritkás mátrixot tároló adatszerkezet megvalósítása – hasítótábla segítségével. (Tehát a mátrix elemeit egy hasítótáblában kell eltárolni.) Ahhoz, hogy kellő memórianyereséget érjünk el, az üres hasítótábla méretére is van korlátozás, ez példányosításkor kerül beállításra, a konstruktor egyik paraméterével (`hashLim`). Természetesen ennél a számnál több darab értéket is el kell tudni tárolni, legrosszabb esetben akár az egész mátrixot (ilyenkor igazából nem is *sparse matrix*-ról beszélünk), viszont ilyen esetekben kulcsütközés fog előjönni. Természetesen ezt kezelni kell, a beszúrandó elemeket el kell tárolni.

A feladat részét képezi a hashfüggvény megírása és az előforduló kulcsütközések feloldása is (a tárgy során bemutatott módszerek egyikével). A hashfüggvényre speciális megkötés nincs, viszont tegyen eleget a meghatározott kritériumoknak[2]:

- Determinisztikus – ugyanazt a kulcsot mindig ugyanarra az indexre képezi le.
- Egyenletes eloszlású kimenet – nem képez le sok elemet ugyanoda.
- Konstans időben számolható.

Ujjlenyomat függvényt nem szükséges írni, egybe szabad vonni a hashfüggvénnyel.

Megvalósítandó függvények:

- `SparseMatrix(int size_x, int size_y, double defaultValue, size_t hashLim);`
  Konstruktor, bemenete a tárolandó mátrix méretei, a default érték, és a hashfüggvény értékkészletét felülről korlátozó paraméter (maximum `hashLim` darab különböző érték).
- `~SparseMatrix();`
  Destruktor.

[1] https://en.cppreference.com/w/cpp/container/map
[2] https://moodle.ppke.hu/mod/book/view.php?id=34836&chapterid=1101

- `void set(int row, int col, double value);`
  Beállítja az (row, col) helyen található értéket, `defaultValue` esetén törli az elemet. Kiindexeléskor exception-t dob.
- `void clear(int row, int col);`
  Törli a mátrix (x, y) helyen található értékét. Kiindexeléskor exception-t dob.
- `double operator() (int row, int col) const;`
  „get" függvény operátorokkal, visszaadja a mátrix [row,col] helyén található értéket. Kiindexeléskor exception-t dob.
- `SparseMatrix operator+ (const SparseMatrix& other) const;`
  Mátrix összeadás. A művelet nem lehet $\mathcal{O}(n \times m)$, ahol n és m a mátrix dimenziói, hanem $\mathcal{O}(l)$ komplexitású kell legyen, ahol l a `defaultValue` értéktől különböző elemek száma. Ha a két mátrix dimenziói eltérnek, exceptiont dob. Figyelj arra, hogy két mátrix összeadásakor a `defaultValue` megváltozhat.
- `void printMatrix() const;`
  Kiírja az összes nem default elem helyét és értékét.

Alkalmazása

A ritkás mátrixokat olyan területeken alkalmazzák, ahol a mátrixokban csak kevés nem nulla érték található. Előfordulhat a numerikus analízisben, ahol például parciális differenciálegyenletek megoldása során keletkezhetnek ritkás mátrixok, vagy hálózatelemzés során, ahol a gráfok szomszédsági mátrixai is gyakran ritkásak, de találkozhatunk velük gépi tanulás és képfeldolgozás során is. Ilyen esetekben nagyon hasznos tud lenni, ha kis memória-, és időigénnyel tudunk műveleteket végezni ezeken a mátrixokon.

Beadás

Ellenőrzés: A megoldás működésének ellenőrzését néhány teszt is segíti, melyek a main.cpp fájlban találhatóak. Ezen tesztek eredményei a consola-ra íródnak ki a program futtatásakor, a console-ra kiírt pontszám csupán a sikeresen lefutott tesztek számát jelzi, nem a feladatra kapott pontszámot. Hogy minél több eset tesztelve legyen, érdemes saját teszteket is írni hozzá, mivel ezek nem feltétlen fednek le minden esetet.

Leadás: A `sparsematrix.hpp` és a `SparseMatrixExceptions.hpp` fájlokat kell az SVN repo szerveren keresztül leadni. A megadott flagekkel (**-Werror -Wall -Wextra -pedantic**) nem forduló kód **0 pontos** házi feladatot eredményez. Ennél a házinál a repo szerveren automata tesztek is futnak, melyek ellenőrzik a helyes működést. A pontozásnál a kódminőség, és az algoritmus hatékonysága is számít a kód helyes működése mellett. Alkalmazzuk a kurzus során tanult módszereket és adatszerkezeteket! A kódba magyarázó kommentek is kerüljenek, amelyek leírják a kód működését!