UNIVERSITEIT GENT

FACULTEIT WETENSCHAPPEN

Systeemprogrammeren

Academiejaar 2018–2019

sysprog@lists.UGent.be

Oefeningenset 4
C++

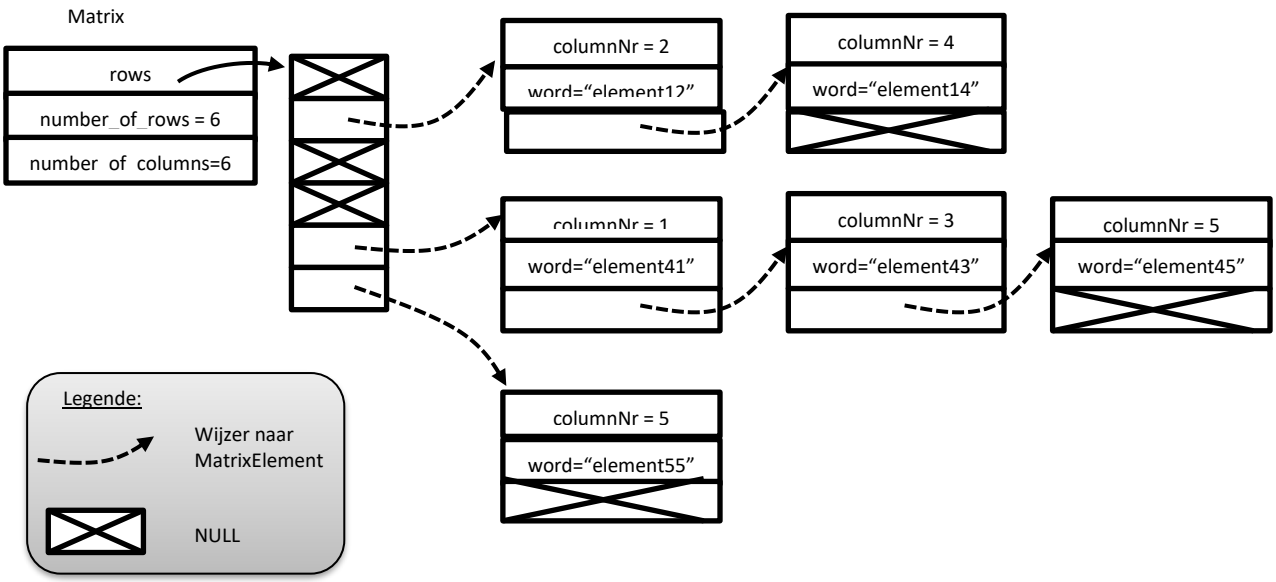
Ijle matrices

Situering

IJle matrices (Engels: **sparse matrices**) zijn matrices waar de meerderheid van de elementen leeg of nul zijn. Het is dan vaak minder efficiënt een array-datastructuur te gebruiken om een ijle matrix voor te stellen aangezien dan veel geheugen verloren gaat voor het bijhouden van irrelevante elementen. Er bestaan efficiëntere structuren voor dergelijke matrices. Hierbij moet dan een afweging gemaakt worden tussen snelle toegang tot de individuele elementen van de matrix, en geheugengebruik.

Een mogelijkheid om een ijle matrix te implementeren, is een **array** bij te houden van de rijen van de matrix. Elk element in deze array is een **geschakelde lijst** van matrix-elementen (`MatrixElement`s). Deze `MatrixElement`s bevatten naast het element ook het kolomnummer, om opzoekingen in de matrix mogelijk te maken. De geschakelde lijst wordt gesorteerd bijgehouden op kolomnummer.

Schematisch kan de datastructuur als volgt worden voorgesteld:



Dit komt overeen met volgende matrix:

		"element12"		"element14"	
	"element41"		"element43"		"element45"
					"element55"

Opgave

Aangezien C++ een objectgeoriënteerde taal is, wordt in deze opgave gevraagd om een aantal klassen te implementeren, waaruit een ijle matrix van strings opgebouwd wordt. Een headerfile (`SparseMatrix.h`) waarin de publieke methodes voor deze klassen reeds zijn ingevuld, en een testfile (`testMatrix.cpp`), zijn reeds gegeven bij de opgave. Implementeer de gevraagde klassen in een nieuw aan te maken bestand `SparseMatrix.cpp`.

MatrixElement

De klasse MatrixElement stelt een element voor in een matrixrij, en bevat een kolomnummer, een wijzer naar het volgende MatrixElement en een string. De klasse-definitie, met constructor, wordt hieronder weergegeven:

```
class MatrixElement{
      int columnNr;
      std::string word;
      MatrixElement* next;


      public:
      MatrixElement(int columnNr, MatrixElement* next, const
      std::string& word): columnNr(columnNr), word(word), next(next) {}
      int getColumnNr() const;
      std::string& getWord();
      MatrixElement* getNext() const;
      void setNext(MatrixElement*);
};
```

Gevraagd in `SparseMatrix.cpp`:

Implementeer de volgende methodes van de klasse `MatrixElement`:

- `int getColumnNr()`
  geeft het kolomnummer terug (getter).
- `std::string& getWord()`
  geeft een referentie naar het bijgehouden `word` terug (getter).
- `MatrixElement* getNext()`
  Geeft de wijzer naar het volgende `MatrixElement` terug (getter).
- `void setNext(MatrixElement*)`

Wijzigt de wijzer naar het volgende `MatrixElement` naar de gegeven parameter (setter).

MatrixRow

Een klasse die een rij in de matrix voorstelt, en geïmplementeerd is als een geschakelde lijst. De details van de implementatie (`MatrixElements`) worden afgeschermd voor de gebruiker van de klasse. De elementen in de geschakelde lijst worden gesorteerd bijgehouden op kolomnummer.

De klasse-definitie wordt hieronder gegeven:

```
class MatrixRow {
    MatrixElement* first;

    public:
    MatrixRow();
    ~MatrixRow();

    int add(int columnNr, const std::string& word);
    int remove(int columnNr);
    int get(int columnNr, std::string& word) const;
    RowIterator getIterator() const;
};
```

Gevraagd in `SparseMatrix.cpp`:

Implementeer de volgende methodes van de klasse `MatrixRow`:

- `MatrixRow()`
  Constructor: initialiseert een nieuwe matrixrij, geïmplementeerd als geschakelde rij.
- `~MatrixRow()`
  Destructor: geeft alle geheugen van de matrixrij en zijn elementen vrij.
- `int add(int columnNr, const std::string& word)`
  Voegt een nieuw element toe aan de matrixrij, met kolomnummer `columnNr`. Geeft -1 terug indien er reeds een element aanwezig is op de aangegeven kolomindex, en 0 indien het toevoegen gelukt is.
- `int remove(int columnNr)`
  Verwijdert het element met kolomnummer `columnNr` uit de geschakelde lijst. Geeft -1 terug wanneer er geen element aanwezig is op het aangegeven kolomnummer, en 0 indien het verwijderen gelukt is.
- `int get(int columnNr, std::string& word) const`
  Geeft het element terug op kolomnummer `columnNr` via `word`. Geeft -1 terug wanneer er geen element aanwezig is op het aangegeven kolomnummer, en 0 indien het ophalen gelukt is.
- `RowIterator getIterator() const`
  Geeft een `RowIterator` terug voor de matrixrij (zie hieronder).

RowIterator

Deze klasse implementeert een iterator over een geschakelde lijst. Een iterator is een object dat toelaat om op efficiënte wijze alle elementen van een collectie (in dit geval een geschakelde lijst) te doorlopen, zonder de implementatie van deze collectie te moeten kennen. De RowIterator moet als volgt gebruikt kunnen worden:

```
RowIterator ri = row.getIterator();
while(ri.hasNext()){
    int i;
    string s = ri.next(i);
    doSomething(i,s);
}
```

Gevraagd in `SparseMatrix.cpp`:

Implementeer de volgende methodes van de klasse `RowIterator`:

- `RowIterator(MatrixElement* start)`
  Constructor. Initialiseert de iterator op `MatrixElement*` start.
- `bool hasNext() const`
  Geeft terug of er nog een volgend element in de lijst is.
- `std::string& next(int& columnNr)`
  Geeft een referentie terug naar de `string` van het huidige `MatrixElement` (ondiepe kopie!), en het bijhorende kolomnummer via `columnNr`; beweegt de iterator naar het volgende element in de rij.

SparseMatrix

Schrijf een klasse `SparseMatrix`, die bovenstaande datastructuur voor ijle matrices implementeert. Maak hiervoor enkel gebruik van de `MatrixRow` en `RowIterator` klassen!

Gevraagd in `SparseMatrix.h`:

- Vul de private attributen van de klasse `SparseMatrix` aan

Gevraagd in `SparseMatrix.cpp`:

Implementeer volgende methodes van de klasse `SparseMatrix`:

- `SparseMatrix(int r, int c)`
  Constructor: initialiseert een nieuwe ijle matrix met `r` rijen en `c` kolommen.
- `SparseMatrix(const SparseMatrix&);`
  Copy constructor: maakt een nieuw object van de klasse SparseMatrix aan als kopie van de meegegeven parameter.
- `~SparseMatrix()`
  Destructor: geeft alle geheugen voor de matrix (en zijn elementen) terug vrij.
- `int add_element(int r, int c, const std::string& s)`

Deze methode voegt een kopie van `s` toe aan de matrix op rij `r` en kolom `c`. Return waarde:

- 0 : toevoegen is gelukt
- -1 : op rij `r` en kolom `c` was al een `string` toegekend
- -2 : `c` valt buiten de grenzen van de matrix
- -3 : `r` valt buiten de grenzen van de matrix

- `int remove_element(int r, int c)`

  Deze methode verwijdert het element op rij `r` en kolom `c` uit de matrix, en geeft het geheugen vrij. Return waarde:

  - 0 : verwijderen is gelukt
  - -1 : er is geen element op rij `r` en kolom `c`
  - -2 : `c` valt buiten de grenzen van de matrix
  - -3 : `r` valt buiten de grenzen van de matrix

- `int const get(int r, int c, std::string& s)`

  Deze methode haalt het element op rij `r` en kolom `c` op, en geeft dit terug via `string s`. Return waarde:

  - 0 : ophalen is gelukt
  - -1 : er is geen element op rij `r` en kolom `c`
  - -2 : `c` valt buiten de grenzen van de matrix
  - -3 : `r` valt buiten de grenzen van de matrix

- `void const print()`

  Deze methode print de matrix af op het scherm. Print enkel de aanwezige elementen af, zoals volgt:

  ```
  element(1,2): "element12"
  element(1,4): "element14"
  …
  end of matrix
  ```

- `void transpose()`

  Deze methode transponeert de matrix: elk element (i,j) wordt nu element (j,i). Deze methode geeft echter geen nieuwe `SparseMatrix` terug, maar past de huidige aan. Denk dus goed na over hoe je dit het gemakkelijkst kan doen met de datastructuur die gegeven is.

- `SparseMatrix operator+(const SparseMatrix& sm)`

  Overladen + operator die twee `SparseMatrix` objecten optelt en het resultaat hiervan teruggeeft. Let er op dat de op te tellen SparseMatrix objecten niet noodzakelijk eenzelfde aantal rijen en kolommen hebben. De resulterende SparseMatrix bevat alle elementen van beide `SparseMatrix` objecten (bij locatie-conflicten i.e. twee elementen met eenzelfde locatie in beide `SparseMatrix` objecten, voeg je het element van parameter `sm` object niet toe)

Tips

- Je maakt in deze opgave gebruik van C++ `strings` (bibliotheek `<string>`). C++ `strings` zijn een wrapper klasse rond C `char*`s, en zijn ontworpen om zelf het eigen geheugen te beheren. Maak gebruik van deze eigenschap!
- Zorg ervoor dat je programma geen geheugenlekken vertoont!
- Indien je geheugen alloceert met `new` of `new[]`, vergeet dit dan niet terug vrij te geven (met `delete` of `delete[]` respectievelijk).
- Schrijf duidelijke code, geen cryptische constructies.