CSE 241 Programming Assignment 3

DUE

April 8, 2018, 23:55

Description

- This is an individual assignment. Please do not collaborate
- If you think that this document does not clearly describes the assignment, ask questions before its too late.

This assignment is about implementing and testing classes for sparse matrix operations.

Sparse Matrix/Vector

- A sparse matrix/vector holds only the non-zero data but acts as a regular matrix/vector.

Basic Elements

SparseVector Class

- Represents a single dimensional sparse data.
- Requirements:
  - `SparseArray` : Constructors
    - Write the required constructors. For example, you need a constructor which takes a string filename data, opens the file, reads the contents, creates and populates an object.
  - `operator+` : Adds two `SparseVectors`
    - Usage: `sparse_vec_1 + sparse_vec_2`.
    - Creates another `SparseVector` object.
  - `operator-` : Subtracts one `SparseVector` from another
    - Similar to `operator+`
  - `operator-` : Negates elements of a `SparseVector`
    - Creates another `SparseVector` object which is element-by-element negative of the operant.
  - `operator=` : Assigns one `SparseVector` to another
    - Usage: sparse_vec_1 = sparse_vec_2
  - `operator<<` : Sends contents of a `SparseVector` to a `std::ostream` object.
    - Creates the text representation of a `SparseVector` and sends it to a `std::ostream` object. (See **Text Representations** section for more details)
  - function `dot` : Calculates the dot product(inner product) of two `SparseVectors`
    - Returns a real number (See **Dot Product** Section for more details)

SparseMatrix Class

- Represents a two dimensional sparse data.
- Requirements:
  - `SparseMatrix` : Constructors.
    - Similar to `SparsVector` class description.
  - `operator+` : Adds two matrices
    - Similar to `SparsVector` class description.
  - `operator-` : Subtracts one matrix from another

    * Similar to SparsVector class description.
  - operator- : Negates elements of a matrix
    * Similar to SparsVector class description.
  - operator= : Assigns one matrix to another
    * Similar to SparsVector class description.
  - operator<< : Sends contents of a SparseMatrix to a std::ostream object.
    * Similar to SparsVector class description.
  - operator* : Multiplies two matrices (Regular matrix multiplication)
    * Similar to SparsVector class description.
  - function transpose : Returns the transpose of a matrix
    * Creates another SparseMatrix which is the transpose of the original object.

Driver Program

- This part describes how you test various operations for the classes you created.
- Your classes will be tested by a driver program. The driver program perform various SparseVector and SparseMatrix operations and incrementally fill a file with the changing contents of the objects created
- Below is an example driver program.(Not all operations are shown)

```
#include <iostream>
#include <fstream>
#include <string>
#include "SparseVector.h"
#include "SparseMatrix.h"

using namespace std;

int main()
{
    ofstream outfile;
    outfile.open("output.txt", ios::out | ios::trunc );


    //Creating a SparseVector from file
    SparseVector a1("a1.txt");
    outfile<<"a1"<<endl<<a1<<endl;

    //Binary operations and assignment
    a1 = a1 + a1;
    outfile<<"a1"<<endl<<a1<<endl;

    //Creating SparseMatrix from file
    SparseMatrix m1("m1.txt");
    SparseMatrix m2("m2.txt");

    outfile<<"m2"<<endl<<m2<<endl;

    //Transpose
    outfile<<m2.transpose()<<endl;

    //Dot product
    outfile<<dot(a1,a1)<<endl;

    return 0;
}
```

Text Representations

Text Representation of SparseVector

- format:

```
<index>:<data> <index>:<data> <index>:<data>...
```

- `index` is in ascending order (natural number)
- example:

```
4:23.8 7:10.7 10:34 12:20 1012:5
```

- For the above example non-zero indices are `4,7,10,12,1012`

Text Representation of SparseMatrix

- format:

```
<row_index> <index>:<data> <index>:<data> <index>:<data>...
<row_index> <index>:<data> <index>:<data> <index>:<data>...
<row_index> <index>:<data> <index>:<data> <index>:<data>...
    .
    .
    .
```

- `index` and `row_index` are in ascending order (natural numbers)
- example:

```
3 3:24.6 4:5.5
4 1:1.15
8 5:6.4 8:34.1 9:13.1
```

Dot Product

- Dot product of two vectors is a scalar operation
- Dot product of `vector_1` and `vector_2`:

```
dot_product = vector_1[0]*vector_2[0] + vector_1[1]*vector_2[1] + vector_1[2]*vector_2[2] + ...
```

Transpose

- Matrix:

```
<row_index> <index>:<data1> <index>:<data2> <index>:<data3>...
<row_index> <index>:<data4> <index>:<data5> <index>:<data6>...
<row_index> <index>:<data7> <index>:<data8> <index>:<data9>...
    .
    .
    .
```

- Transpose of the Matrix

```
<row_index> <index>:<data1> <index>:<data4> <index>:<data7>...
<row_index> <index>:<data2> <index>:<data5> <index>:<data8>...
<row_index> <index>:<data3> <index>:<data6> <index>:<data9>...
    .
    .
    .
```

File I/O

File I/O objects are defined in `<fstream>` header.

In order to write to a file, first wee need to create the file stream object. A file stream object is similar to `std::cout`. For output, It is type is `std::ofstream`. This type is derived from `std::ostream`.

```
//create the file stream object
ofstream couttofile;

//open the file and associate it with the object
couttofile.open("output.txt", ios::out | ios::trunc );

//write to stream object
couttofile<<"Test"<<endl;
couttofile<<"Test2"<<endl;
.
.
.
```

In order to write to a file, first wee need to create the file stream object. A file stream object is similar to `std::cin`. For input, It is type is `std::ifstream`. This type is derived from `std::istream`.

```
//create the file stream object
ifstream cinfromfile;

//open the file and associate it with the object
cinfromfile.open("input.txt");

//read "12:23.5" from stream object
int a;
double b;
char c;
cinfromfile>>a>>c>>b;

//in order to read the a line from a file, you can use getline()
// function from <string> library.
string s;
std::getline(cinfromfile, s);

//reading lines in a loop
//a helper function in order to secure file read operations
int check_errors(ifstream* f) {
    int stop = 0;
    if (f->eof()) {
        // EOF after std::getline() is not the criterion to stop processing
        // data: In case there is data between the last delimiter and EOF,
        // getline() extracts it and sets the eofbit.
        stop = 0;
        }
    if (f->fail()) {
        stop = 1;
        }
    if (f->bad()) {
        stop = 1;
        }
    return stop;
}
```

```
//Create a string
string line;

//Create an ifstream object by providing a filename
// This opens the file as well
ifstream f ("file.txt");

//check if it is open
if (f.is_open())
{
    while(1) {
        getline(f, line);
        if (check_errors(&f)) {
            //skip the data processing and break
            break;
            }
        // This is the actual operation on the data obtained and we want to
        // protect it from errors during the last IO operation on the stream
        cout << "data line " << ": " << line << endl;
    }
}
```

Remarks

- Write comments in your code.
- If your code does not compile you will get `0`
- Do not share your code with your classmates.

Turn in:

- "SparseVector.h"
- "SparseMatrix.h"
- `.cpp` implementations of classes and everything else you created.
- Your code will be compiled according to the following `GNU make` script
- You can also provide your own makefile script
- Do not send any IDE specific files.

```
SRC_DIR := .
OBJ_DIR := .
SRC_FILES := $(wildcard $(SRC_DIR)/*.cpp)
OBJ_FILES := $(patsubst $(SRC_DIR)/%.cpp,$(OBJ_DIR)/%.o,$(SRC_FILES))
LDFLAGS := ...
CPPFLAGS := ...
CXXFLAGS := ...

main.out: $(OBJ_FILES)
    g++ $(LDFLAGS) -o $@ $^

$(OBJ_DIR)/%.o: $(SRC_DIR)/%.cpp
    g++ $(CPPFLAGS) $(CXXFLAGS) -c -o $@ $<

CPPFLAGS += -std=c++11
CXXFLAGS += -MMD
-include $(OBJ_FILES:.o=.d)
```

Late Submission

- `(0,24]` hours: `-20%`
- `(24,48]` hours: `-40%`
- `(48,72]` hours: `-60%`
- `(72,-)` hours: `-100%`