ED – Estrutura de Dados – 2015/2

T1 – 1o. Trabalho Prático de Implementação

05 de outubro de 2015

1 Descrição do Problema

Matrizes esparsas são matrizes de alta ordem (i.e., com dimensões grandes) nas quais a maior parte dos elementos são iguais a zero. Matrizes esparsas aparecem com freqüência em problemas de engenharia e ciências, o que justifica a elaboração de algoritmos eficientes para manipulação deste tipo de matriz.

O objetivo deste trabalho é elaborar estruturas de dados e algoritmos para manipulação de matrizes esparsas, sem que todos os elementos estejam presentes (apenas os elementos diferentes de zero serão armazenados). Deseja-se, com isto, economizar grande quantidade de espaço de memória na representação das matrizes e tornar as operações usuais (soma, multiplicação, inversão, etc.) mais ágeis.

2 Requisitos da implementação

Para representar as matrizes esparsas, você deverá usar uma estrutura encadeada para armazenar cada elemento diferente de zero da matriz. A sua matrix esparsa deve ser implementada da seguinte forma:

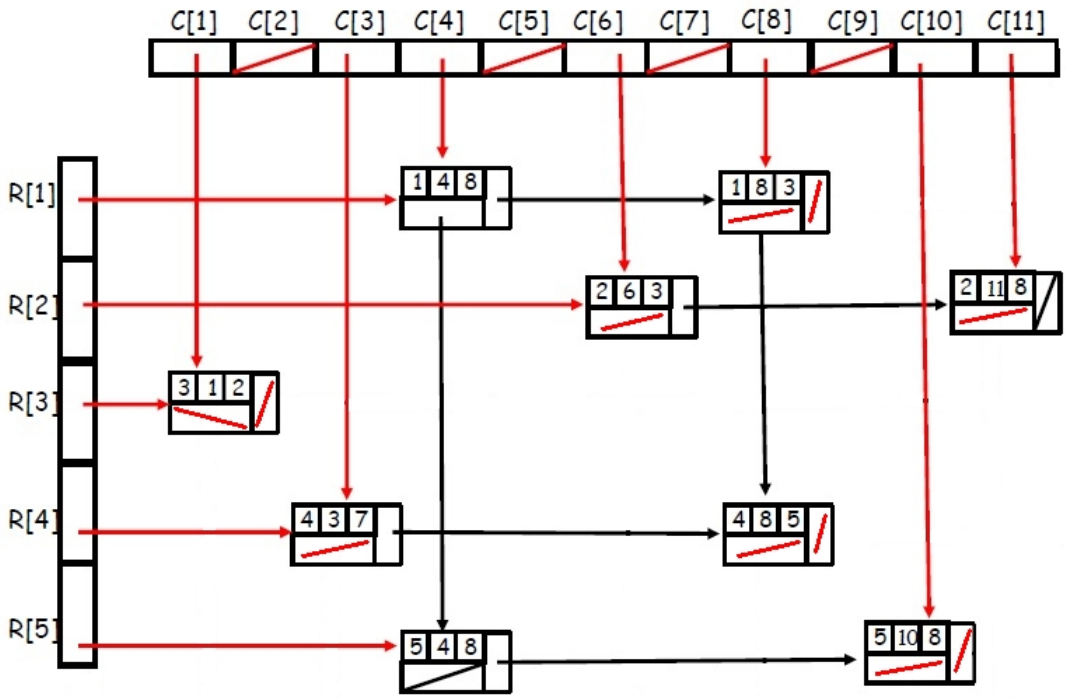
1. Cada célula da matriz contém a sua posição i, j dentro da matriz, um número *float* que é o valor diferente de zero armazenado nesta posição, além de um ponteiro para a próxima célula na linha e um ponteiro para a próxima célula na coluna. Desta forma, cada linha e cada coluna da matriz formam uma lista com encadeamento simples.
2. Um vetor de ponteiros para a cabeça de cada linha.
3. Um vetor de ponteiros para a cabeça de cada coluna.
4. Um cabeçalho que armazena um ponteiro para os vetores dos itens 2 e 3, além da(s) dimensão(ões) destes vetores.

2.1 Exemplo

A matrix 5×11 abaixo

$$\begin{matrix} 0 & 0 & 0 & 8 & 0 & 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 3 & 0 & 0 & 0 & 8 \\ 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 7 & 0 & 0 & 0 & 0 & 5 & 0 & 0 \\ 0 & 0 & 0 & 8 & 0 & 0 & 0 & 0 & 8 & 0 \end{matrix}$$

fica armazenada como na figura a seguir



Note que nesta figura não está representado o cabeçalho do item 4. IMPORTANTE: para simplificar o problema você pode criar uma implementação que só trabalha com matrizes quadradas.

2.2 Operações da matriz

A sua implementação deve prover ao menos as seguintes operações sobre uma matriz esparsa:

1. **Criação (alocação) de uma matriz vazia.**
2. **Inserir um valor numa posição i, j da matriz.**
3. **Remover um valor da posição i, j da matriz.**
4. **Imprimir a matriz.**
5. **Somar duas matrizes.**
6. **Destruição (desalocação) da matriz.**

Algumas observações importantes:

- A inserção de um valor 0.0 numa posição i, j na verdade remove uma célula que eventualmente exista naquela posição.
- A impressão da matriz deve seguir o formato de saída descrito adiante.

2.3 Formato de execução do programa – parâmetros de entrada

Nesta seção estão descritos os formatos dos arquivos de entrada e o formato da saída que seu programa deve exibir. *ATENÇÃO*: siga com muito cuidado os padrões estabelecidos nesta seção. Desta forma você terá a garantia que conseguirei usar o seu trabalho durante a correção. Trabalhos fora do padrão aqui estabelecido, no mínimo, perderão pontos, podendo até mesmo não serem corrigidos.

Seu programa deve receber dois argumentos indicando dois nomes de arquivos do qual serão lidas duas matrizes m_1 e m_2. Após a leitura, o seu programa deve calcular e exibir o resultado de $m_1 + m_2$.

A seguir, um exemplo de como o seu programa deverá ser executado para produzir como saída a soma de duas matrizes esparsas passadas como parâmetros:

```
./trab1 arquivo1 arquivo2
```

2.4 Formato do arquivo de entrada

Seu programa deverá ser capaz de ler arquivos contendo a descrição das matrizes esparsas. O arquivo de descrição estará sempre em formato texto, no qual em cada linha será descrito um elemento da matriz.

Cada linha do arquivo conterá um registro no seguinte formato:

```
linha;coluna;valor
```

A primeira linha do arquivo conterá a dimensão da matriz. Para facilitar a implementação das operações, assuma que os parâmetros de entrada serão sempre matrizes quadradas e de mesma dimensão (quando for o caso).

Um exemplo de um arquivo de entrada válido, representando a matriz

50	0	0	0
10	0	20	0
0	0	0	0
-30	0	-60	5

é dado a seguir:

```
4
1;1;50
2;1;10
2;3;20
4;1;-30
4;3;-60
4;4;5
```

Ainda sobre os arquivos de entrada, valem as seguinte regras:

- No arquivo estarão representados somente os elementos diferentes de zero.
- Você pode assumir que os arquivos de entrada respeitarão a ordem do exemplo anterior, ou seja, os registros estarão ordenados pelo índice da linha e depois pelo índice da coluna.
- IMPORTANTE: note que os índices da matriz no arquivo começam em 1 e não em 0.

Seu programa deverá produzir como saída uma matriz exatamente no mesmo formato que o de entrada. Porém, neste caso, não será necessário gravar o resultado em um arquivo.

ATENÇÃO: seu programa deverá produzir como saída (NA TELA) SOMENTE a matriz gerada pela operação de soma. Nada além disto. Ou seja, nenhuma outra mensagem e nenhuma formatação adicional deverá ser exibida.

Isto é absolutamente necessário porque será preparada uma bateria de testes para validação de seu trabalho, que verificará se sua resposta está correta baseado na saída do seu programa. Será disponibilizado na página da disciplina um conjunto de testes para você utilizar enquanto desenvolve.

2.5 Considerações gerais sobre a implementação

- Modularize o seu código adequadamente. Crie arquivos **.c** e **.h** para cada módulo do seu sistema. Em especial, crie arquivos exclusivos para manipular as estruturas de dados dos tipos abstratos de dados que você estiver representando.
- Utilize guarda de inclusão em todos os seus arquivos **.h**.
- Implemente a sua matriz esparsa como um tipo opaco.
- Serão disponibilizados arquivos de testes na página da disciplina. As matrizes de teste terão ordem extremamente alta, o que deixará absolutamente inviável a execução de seu programa caso a estrutura de dados armazene também os elementos zerados da matriz.

- Durante a correção, o seu trabalho será testado com uma bateria de testes diferente da disponibilizada como exemplo.
- Crie códigos claros e organizados. Utilize um estilo de programação consistente. Comente o seu código extensivamente. Isto ajudará no momento da correção.

3 Regras para desenvolvimento e entrega do trabalho

- **Data da Entrega**: O trabalho deve ser entregue até às 23:55 h do dia 27/10/2015 (Terça-feira). Não serão aceitos trabalhos após essa data.
- **Grupo**: O trabalho é *individual*.
- **Linguagem de Programação**: Você deverá implementar o seu trabalho na linguagem C.
- **Ferramentas**:
  - O seu trabalho será corrigido no Linux.
  - Use a ferramenta Valgrind (`http://valgrind.org`) para garantir a ausência de *memory leaks*.
- **Como entregar**: Pela atividade criada no Moodle. Envie um arquivo compactado com todo o seu trabalho. Envie também um arquivo LEIAME com as instruções para compilar o seu trabalho. (Faça isso mesmo que seja necessário apenas um simples comando, tal como `gcc *.c` ou algo similar.)

4 Avaliação

- O trabalho vale 20 pontos.
- Trabalhos com erros de compilação receberão nota zero.
- Caso seja detectado plágio (entre alunos ou da internet), todos os envolvidos receberão nota zero.
- Se o seu trabalho tiver *memory leaks* haverá desconto de pontos na nota do trabalho. Tal desconto será proporcional à gravidade das falhas detectadas no código.
- Serão levadas em conta, além da correção da saída do seu programa, a clareza e simplicidade de seu código.
- Uma parte da correção dos trabalhos é automática. Siga todas as orientações deste documento, em especial, siga o padrão determinado para geração da saída.
- A critério do professor, poderão ser realizadas entrevistas com os alunos, sobre o conteúdo do trabalho entregue. Caso algum aluno seja convocado para uma entrevista, a nota do trabalho será dependente do desempenho na entrevista. (Vide item sobre plágio, acima.)