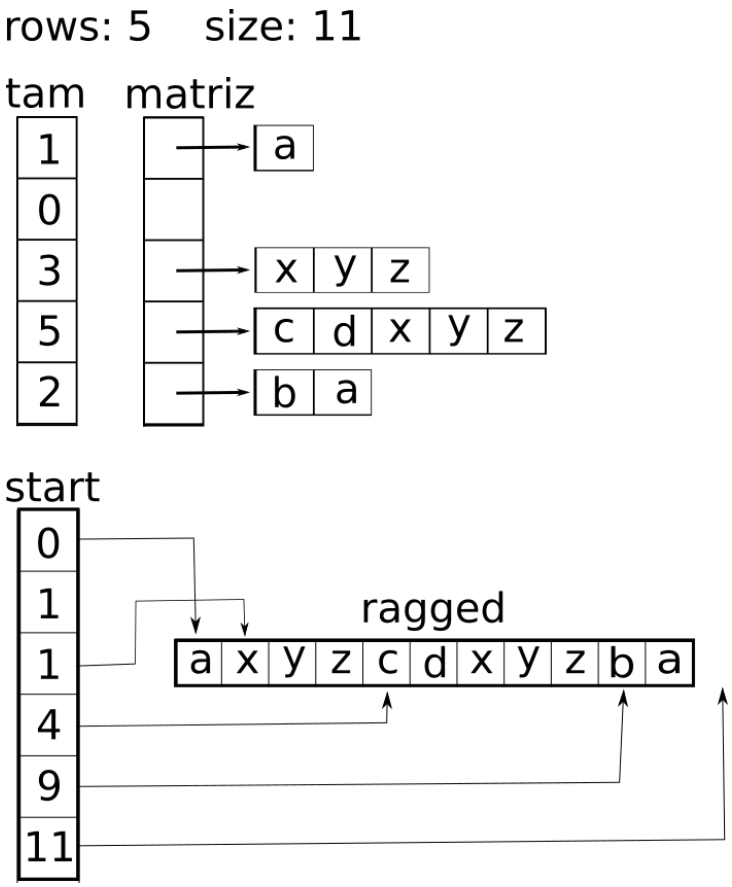
→ LEMBREM-SE DE USAR PAPEL E CANETA COMO RASCUNHO ANTES DE IMPLEMENTAR ←

Trabalho 1 - Ragged array

Arquivos úteis:
https://drive.google.com/file/d/1gkLXvHBHjcKgqEV53DyDvZg0hgBxk2SZ/view?usp=sharing



Neste trabalho implementaremos uma representação alternativa de matrizes chamada de "ragged array". O objetivo será praticar o uso de templates, ponteiros e alocação dinâmica de memória. Entender este trabalho lhe ajudará na implementação do trabalho 2.

Considere uma matriz onde cada linha pode conter um número diferente de colunas. Normalmente essa matriz poderia ser armazenada usando um array de apontadores, onde cada apontador aponta para um array (alocado de forma dinâmica) contendo as colunas da matriz. Adicionalmente, também é necessário armazenar os tamanhos de cada coluna. Veja um exemplo na imagem acima (parte superior).

→ LEMBREM-SE DE USAR PAPEL E CANETA COMO RASCUNHO ANTES DE IMPLEMENTAR ←

Uma representação alternativa, conhecida como “ragged array”, consiste em linearizar a matriz acima, armazenando-a em um array grande. Essa representação está exemplificada na parte inferior da imagem acima. Note que é utilizado um array “start” de inteiros para indicar em qual posição se encontra o primeiro elemento de cada linha. Exemplo: a linha 2 começa no elemento 1 (x) do array e, como há 3 elementos na linha 2, então a linha 3 começa no elemento 4 (c), …

Observe que, como sabemos onde cada linha começa, podemos facilmente calcular o número de elementos em cada linha com base em sua posição inicial e a posição inicial da próxima linha. Como a última linha não possui uma “próxima linha”, armazenamos um elemento extra no array start que representa onde a próxima linha começaria (se ela existisse). Isso facilita o cálculo do tamanho de cada linha.

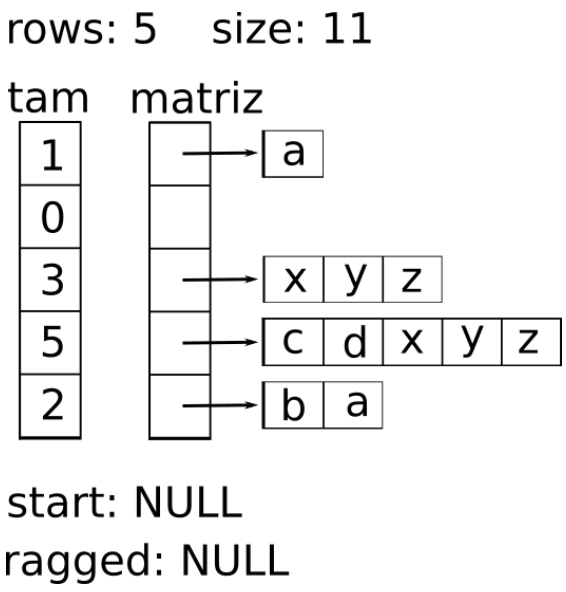
Sua implementacao

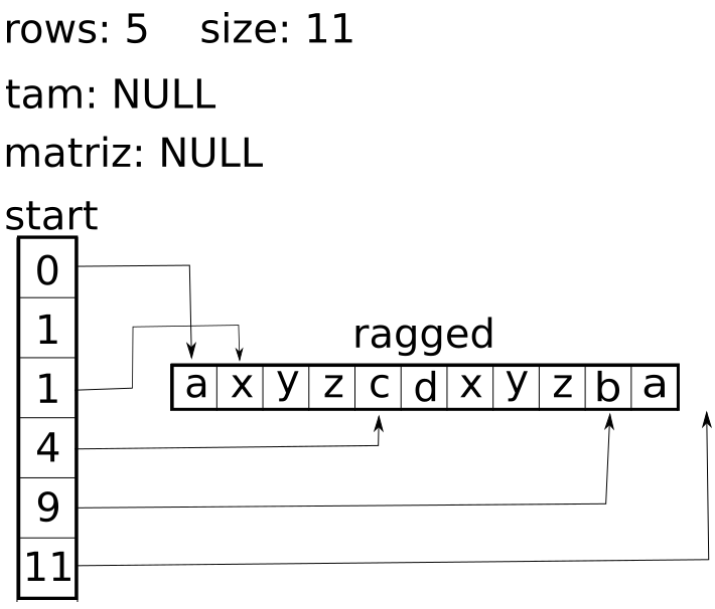
Você deverá implementar uma classe chamada “MyMatrix” (implemente-a em um arquivo com nome “MyMatrix.h”), capaz de armazenar matrizes de tipos arbitrários das duas formas descritas acima: tradicional e “ragged”.

Sua classe deverá ter, como membros de dados **protegidos**, as variáveis “rows”, “size”, “tam”, “matriz”, “start” e “ragged” (use **EXATAMENTE** esses nomes). **Nenhum** outro membro de dados será permitido na sua classe (nem protegido, nem privado e, certamente, nem público).

Se a matriz estiver sendo representada no modo “ragged”, tam e matriz deverão ser NULL. Caso contrário, “start” e “ragged” deverão ser NULL. Sua estrutura de dados nunca estará, simultaneamente, nos dois modos de operação.

Veja, abaixo, um exemplo de como sua estrutura de dados deveria estar quando a matriz estiver representada no modo tradicional (esquerda) e no modo ragged (direita):



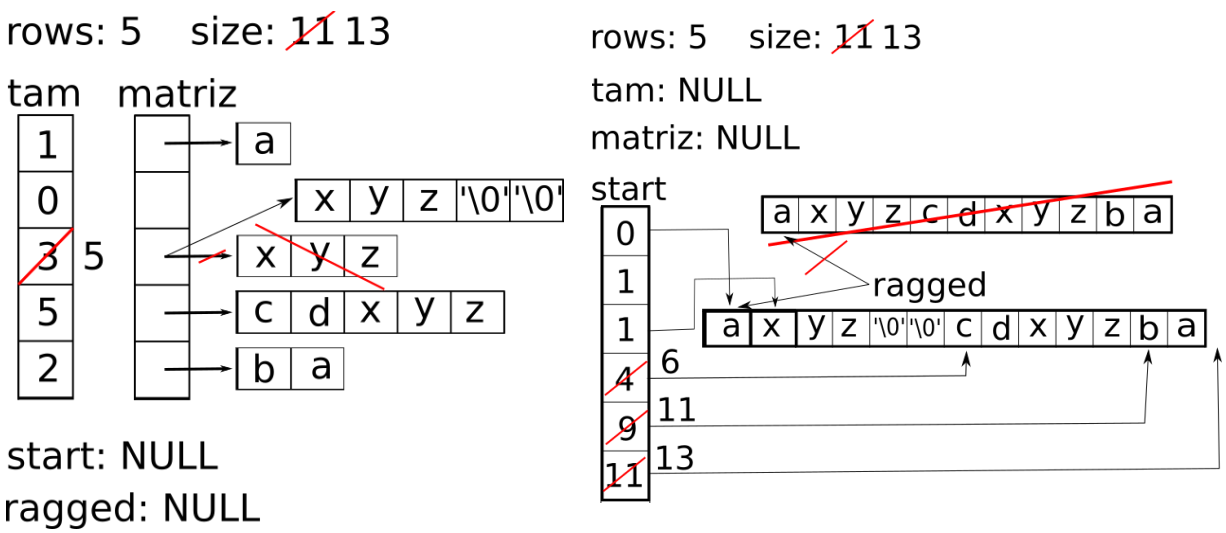


→ LEMBREM-SE DE USAR PAPEL E CANETA COMO RASCUNHO ANTES DE IMPLEMENTAR ←

Sua implementação deverá seguir os diagramas acima de forma exata. Note que em toda situação os arrays terão EXATAMENTE o tamanho necessário para armazenar os elementos que queremos (ou seja, não vamos usar uma "capacidade extra" igual à usada na classe MyVec para reduzir a necessidade de realocações de memória ao adicionar elementos à estrutura de dados -- isso poderia ajudar em algumas situações).

Você não poderá utilizar nenhuma estrutura de dados pronta na sua classe (ou seja, você deverá gerenciar a memória de forma explícita usando operadores new[] e delete[]).

Note que, ao redimensionar uma linha da matriz, o comportamento da sua classe dependerá do modo dela. Por exemplo, veja abaixo o que deve acontecer ao redimensionar a linha 2 (mudando o tamanho de 3 para 5) da matriz apresentada no exemplo anterior. Na esquerda temos o caso da matriz estar no formato tradicional e na direita ela estará no formato "ragged":



Observe que, no caso do formato tradicional, o redimensionamento será similar ao que ocorreria no caso de um vetor dinâmico do tipo "MyVec" (mas o novo tamanho será exatamente 5). No segundo caso o "vetor grande" ragged terá que ser completamente redimensionado (e vários valores em "start" deverão ser atualizados). No caso de aumento de tamanho das linhas (como no exemplo acima), os novos valores deverão ser inicializados com **o valor padrão do tipo utilizado na matriz** (ex: 0 para inteiros e floats, '\0' para char, "" para strings, etc).

→ LEMBREM-SE DE USAR PAPEL E CANETA COMO RASCUNHO ANTES DE IMPLEMENTAR ←

O construtor da sua classe deverá receber tres argumentos: o número de linhas da matriz (n), um array (de tamanho n) com o número de elementos em cada linha e um booleano indicando se a matriz será ragged (true) ou não (false). Seu construtor não deverá inicializar os elementos da matriz (eles começarão com lixo)

Você deverá trocar o formato interno da MyMatrix APENAS quando solicitado (por meio das funções convertToRagged() e convertToTraditional()). Por exemplo, alguém poderia achar mais fácil implementar a função resizeRow apenas para matrizes no formato tradicional e, se a matriz estiver no formato ragged, chamar convertToTraditional() para convertê-la para o formato tradicional, alterar o tamanho da linha e, por fim, converter de volta para o formato ragged. Isso facilitaria a implementação (e provavelmente seria menos eficiente), mas NÃO DEVERÁ ser feito no trabalho (ou seja, sua função resizeRow deverá ter um "if" para verificar em qual formato a matriz está e, então, redimensionar usando um código especializado para cada um dos dois formatos).

Funções que deverão ser implementadas

Pelo menos as seguintes funções públicas deverão ser implementadas: *get, set, getNumRows, getNumElems,getNumCols, resizeRow, resizeNumRows, isRagged, convertToRagged, convertToTraditional* e *print*.

O arquivo "ExemplosDeUso.cpp" contém exemplos de uso de sua classe (com as funções básicas que você deverá implementar -- siga exatamente aqueles nomes, pois sua classe será testada utilizando testes automatizados). **Estude esses exemplos com atenção (muitos detalhes não apresentados aqui no texto estão descritos la).**

Lembre-se de usar **boas práticas de engenharia de software e teste bastante sua classe** -- ela será testada usando não apenas com o exemplo disponibilizado naquele arquivo (por exemplo, a correta gerência da memória também será testada). Saber projetar corretamente a classe e escolher as assinaturas das funções é parte da avaliação.

Exceto para a função *print* (que deve imprimir no stdout), nenhuma função de sua classe deve imprimir saídas (nem no stdout nem no stderr).

Experimentos a serem realizados e conclusões

Ragged arrays possuem vantagens e desvantagens em relação a matrizes tradicionais (que poderiam ser facilmente criadas usando, por exemplo, um MyVec<MyVec<T> >. Uma vantagem é que os dados ficam armazenados de forma mais contígua na memória (em uma matriz, as linhas podem estar armazenadas bem longe umas das outras na memória) e, com isso, há um melhor uso da *cache* do processador. Adicionalmente, em geral é muito mais eficiente alocar um array grande em vez de muitos arrays pequenos. Essa diferença normalmente é sentida em programas de alto desempenho implementados para processadores com muitos núcleos e, principalmente, GPUs.

→ LEMBREM-SE DE USAR PAPEL E CANETA COMO RASCUNHO ANTES DE IMPLEMENTAR ←

O arquivo “Benchmark.cpp” mede o tempo gasto para fazer algumas operações na sua classe (usando os dois modos). Benchmarks mais “científicos” normalmente são mais complexos, mas para este trabalho já é possível usar tal arquivo para observar algumas características interessantes da sua classe.

Realize alguns experimentos e observe o tempo das diferentes operações nos dois modos de sua estrutura de dados. Escolha bem os argumentos do programa “benchmark”. Por exemplo, fazer experimentos apenas com matrizes pequenas (exemplo: 100 linhas e, no máximo, 50 colunas) provavelmente não é uma boa ideia (mas mesmo assim sugiro incluir algumas matrizes pequenas nos testes). O importante é ser capaz de, com os experimentos, ver em quais situações a versão ragged funciona melhor, em quais funciona de forma parecida e em quais funciona pior do que o modo tradicional. Dica: em geral, conseguimos observar mais fatos interessantes fazendo alguns experimentos “bem diferentes uns dos outros” do que fazendo vários experimentos parecidos.

Descreva, no seu README.txt, seus experimentos (indicando o computador/SO utilizado nos testes, os parâmetros e tempos observados) e as conclusões que você obteve com eles (ex: em quais operações ragged parece funcionar melhor? Para quais matrizes? etc). Justifique sucintamente suas observações.

Ordens de complexidade

Escreva na linha anterior à da implementação de cada função pública da sua classe a complexidade de tempo dessa função supondo:
- Que a matrix possui R linhas
- Cada linha possui, em média, C elementos. (suponha que o número de colunas em cada linha seja razoavelmente uniforme)
- O número total de elementos na matriz é T

Use apenas as variáveis R, C e T nas ordens de complexidade. Lembre-se de simplificar suas ordens de complexidade (ex: usar O(R) em vez de O(2R + 3))

Obs: em cada função você deverá ter duas ordens de complexidade (uma em cada linha): a do modo tradicional e a do modo ragged (nessa ordem). Justifique sucintamente suas análises.

Exemplo:

```
//Complexidade do modo tradicional: O(R*C) -- precisamos varrer todas as R colunas da matriz e, em cada coluna, somar os C elementos
//Complexidade do modo ragged: O(1) -- no modo ragged, a resposta é obtida usando um cálculo que independe do tamanho da matriz.
template <....>
void MyMatrix::funcao123() {
        //implementacao...
```

→ LEMBREM-SE DE USAR PAPEL E CANETA COMO RASCUNHO ANTES DE IMPLEMENTAR ←

}

Arquivo README

Seu trabalho deverá incluir um arquivo README.

Tal arquivo contera (nessa ordem):

- Seu nome/matricula
- Resultados dos experimentos (conforme descrito na secao “Experimentos a serem realizados e conclusões”)
- Informacoes sobre todas fontes de consulta utilizadas no trabalho

Submissao

Submeta seu trabalho utilizando o sistema Submitty até a data limite. Seu programa será avaliado de forma automática (os resultados precisam estar corretos, o programa não pode ter erros de memória, etc), passará por testes automáticos “escondidos” e a qualidade do seu código será avaliada de forma manual.

Você deverá enviar os arquivos: README.txt e MyMatrix.h

Duvidas

Dúvidas sobre este trabalho deverão ser postadas no sistema Piazza. Se esforce para implementá-lo e não hesite em postar suas dúvidas!

Avaliacao manual

Principais itens que serão avaliados (além dos avaliados nos testes automáticos):
- Comentarios
- Indentacao
- Nomes adequados para variáveis
- Separação do código em funções lógicas
- Uso correto de const/referencia
- Uso de variáveis globais apenas quando absolutamente necessário e justificável (uso de variáveis globais, em geral, e’ uma má prática de programação).
- etc

Regras sobre plágio e trabalho em equipe

- Este trabalho deverá ser feito de forma individual.
- Porém, os alunos podem ler o roteiro e discutir ideias/algoritmos em alto nível (nível mais alto do que pseudocódigo) de forma colaborativa.
- As implementações (nem mesmo pequenos trechos de código) não deverão ser compartilhadas entre alunos. Um estudante não deve olhar para o código de outra pessoa.

→ LEMBREM-SE DE USAR PAPEL E CANETA COMO RASCUNHO ANTES DE IMPLEMENTAR ←

- Um estudante não deve utilizar o computador de outro colega para fazer/submeter o trabalho (devido ao risco de um ter acesso ao código do outro). Cuidado para não deixar seu código fonte em algum computador público (a responsabilidade pelo seu código é sua).
- Crie um arquivo README (submeta-o com o trabalho) e inclua todas as suas fontes de consulta (incluindo pessoas que lhe ajudaram em algo do trabalho).
- Não poste seu código (nem parte dele) no Piazza (ou outros sites) de forma pública (cada aluno e' responsável por evitar que outros plagiem seu código).
- Se você estiver lendo isto (deveria estar…) escreva "Eu li as regras" na primeira linha do seu README (isso será avaliado).
- Trechos de código não devem ser copiados de livros/internet. Se consultar algum livro ou material na internet essa fonte deverá ser citada no README.
- Se for detectado plágio (mesmo em pequenos trechos de código) a nota de TODOS estudantes envolvidos será 0.
- Além disso, os estudantes poderão ser denunciados aos órgãos responsáveis por plágio da UFV. Lembrem-se que um conceito F (fraude académica) no histórico escolar pode afetar severamente seu currículo.
- O plágio pode ser detectado após a liberação das notas do trabalho (ou seja, pode ser que o professor reavalie os trabalhos procurando por plágio no final do semestre). Assim, sua nota poderá ser alterada para 0 caso algum problema seja encontrado posteriormente (essa alteração na nota vale apenas para casos de plágio).