INF 213 - Roteiro da Aula Pratica 10

O objetivo desta aula e' praticar o uso e implementacao de pilhas, filas e filas de prioridade.

Arquivos fonte e diagramas utilizados nesta aula:

Considere o programa "driver.cpp". Implemente cada funcao "etapaN()" adicionando a implementacao da respectiva etapa (você pode usar includes na area de cada etapa para incluir classes ou dividir seu codigo em vários arquivos).

(para testar a sua implementacao das etapas 1 e 2, crie um arquivo Mediana.cpp vazio e submeta junto ao seu trabalho -- se não fizer isso seu codigo não compilara. A implementacao do arquivo Mediana.cpp sera realizada na etapa 3)

Etapa 1

Crie uma funcao que le 5 numeros, armazena cada um em uma fila e em uma pilha. Ao final, ele deve imprimir o conteudo da pilha (com um espaco em branco após cada elemento), uma nova linha, o conteudo da fila e uma nova linha.

Qual a diferenca entre a ordem dos numeros tirados da fila e da pilha?

Etapa 2

Desenvolva uma funcao que, dada uma entrada contendo vários caracteres e, entre eles, simbolos (,), [,], { e }, verifica se tais simbolos estao consistentes (balanceados). Sua funcao devera imprimir "Consistente" caso a entrada esteja consistente e "Inconsistente" caso a entrada esteja inconsistente. Imprima uma nova linha (vazia) após o resultado.

Exemplos de entradas consistentes (nesse caso cada linha e' uma entrada):
(3*(5+2))
[3+(2-1)*5]
[] {} () ([])
(5+ 1=500)
(inf213 123)

Exemplos de entradas inconsistentes:
))((
)([]
[5+2. *(4+2])
[5+ 2*(4+1]
([)]

Cada arquivo de teste terá apenas uma entrada (uma linha contendo a entrada).

Etapa 3

Este é um problema com estilo similar ao de entrevistas de emprego em grandes empresas de tecnologia como Google e Facebook.

Crie uma estrutura de dados (usando uma classe) chamada Mediana (salve o cabecalho com o nome Mediana.h e a implementacao com o nome Mediana.cpp), que possui as seguintes funcoes publicas:
- insere(x) : insere o inteiro x na estrutura de dados -- a complexidade maxima devera ser O(log n)
- getMediana() : retorna a mediana dos numeros ja inseridos na classe -- a complexidade maxima devera ser O(1).

Observe que, por simplicidade, não utilizaremos templates (vamos supor que apenas inteiros serao armazenados na classe).

Note que caso o numero de elementos na estrutura seja par, a mediana sera a media de dois valores. Calcule a media usando divisao inteira (i.e., a mediana da lista [3,6] sera' 4, não 4.5).

Utilize apenas as estruturas de dados vistas em sala de aula (você não pode utilizar a STL, mas pode reutilizar o codigo das estruturas de dados mostradas em sala)

Dica: utilize duas filas de prioridade na sua classe! Em que elas poderiam ajudar?

Utilizando sua classe, implemente a funcao "etapa3()" de modo que ela le um numero N. Entao, N numeros deverao ser lidos da entrada padrao. Para cada numero lido, imprima a mediana de todos os numeros lidos ate o momento e um espaco em branco. Imprima uma nova linha (vazia) após o resultado.

Entrada: 5 1 2 3 4 5	Saida esperada: 1 1 2 2 3

Explicacao: a mediana da lista [1] e' 1, a mediana (parte inteira) de [1,2] e' 1, a mediana de [1,2,3] e' 2, a mediana de [1,2,3,4] e' 2, a mediana de [1,2,3,4,5] e' 3.

Os casos de teste poderão ter até 1 milhão de números.

Observe que uma implementação ineficiente (MedianaLenta) e trivial foi provida como exemplo. Qual a ordem de complexidade da etapa3() utilizando essa implementação? E usando sua implementação mais eficiente?

Utilizando o utilitário time() do Linux, Meca o tempo da versão provida para as entradas input_n.txt (n=1,2,3,4,5,6) e compare com o tempo da sua versão. Como os tempos crescem a medida em que n cresce?

A implementação que você entregar pelo submitty deverá usar a sua classe mediana, não a classe MedianaLenta.

Exemplo de aplicação para este problema: uma empresa de cartao de credito pode verificar se o valor gasto em um dia está muito maior do que a mediana dos gastos do cliente ate o momento (isso poderia indicar que o cartao foi roubado).

Etapa 4 (esta etapa não devera ser entregue)

Uma importante aplicacao de filas e pilhas e' realizar operacoes do tipo "busca" (isso sera estudado em mais detalhes no final da disciplina de Estrutura de Dados, em Projeto e Analise de Algoritmos e em Teoria e Modelos de Grafos). Mais especificamente, pilhas podem ser facilmentes utilizadas para realizar buscas "em profundidade", enquanto filas podem ser utilizadas para implementar buscas "em largura" (não se preocupe com a definicao formal desses dois tipos de busca agora).

Um exemplo de problema que pode ser resolvido com busca e' determinar se e' possivel atingir uma saida de um labirinto (representado por uma matriz retangular onde cada elemento representa uma "celula" do labirinto) a partir de um ponto inicial.

Um algoritmo para resolver esse problema consiste no seguinte:
Crie uma pilha ou fila S e adicione a S as coordenadas do ponto inicial
Enquanto S não estiver vazia, remova o primeiro elemento de S e se ele ainda não tiver sido visitado marque-o como visitado e adicione seus vizinhos (que possam ser visitados, ou seja, que não representarem um muro) a S. Se o elemento ja tiver sido visitado ignore-o.

Pode-se mostrar que sera possivel sair do labirinto se, e somente se, a saida dele for eventualmente visitada. Observe que ao marcar um elemento como visitado evitamos visita-lo duas vezes (o que poderia causar, por exemplo, um loop infinito -- por que?).

Teste os programas resolveLabirinto.cpp e resolveFila.cpp (utilizando os labirintos de exemplo) . O primeiro utiliza uma pilha para determinar se e' possivel achar a saida de um labirinto a partir da entrada enquanto o segundo utiliza uma pilha. Durante o processo as celulas do labirinto que forem sendo visitadas sao marcadas e exibidas em tela.

Observe que a letra X representa uma parede do labirinto, $ representa a saida, @ representa o inicio e "." representa uma area vazia. Observe tambem que supomos que cada celula e' vizinha de, no maximo, 4 outras celulas (as vizinhas que estiverem na mesma linha ou na mesma coluna).

Qual a diferenca que você observou entre a busca em largura e em profundidade?

Submissao da aula pratica:

A solucao (exceto a implementacao da etapa 4, que não precisa ser entregue) deve ser submetida ate as 18 horas do dia 11/06 utilizando o sistema submitty (submitty.dpi.ufv.br). Atualmente a submissao so pode ser realizada dentro da rede da UFV.