Trabalho 3

Fila de prioridades com heap

(PARTE 1: 7 pontos)

ACH2002 - Introdução à Análise de Algoritmos
Profa. Karina Valdivia Delgado
Monitora: Adriana Jacoto Unger

Grupo: de até 2 alunos.

1 Problema

Uma fila de prioridade é um tipo abstrato de dados que mantém um conjunto S de elementos, cada qual com uma prioridade associada e que suporta as seguintes operações:

- Maximum(S): retorna o elemento de S que possui a maior prioridade
- Extract-max(S): remove e devolve o elemento de S que possui a maior prioridade
- Insert(S, x, prior): insere o elemento x que tem uma prioridade prior no conjunto S

A estrutura de dados heap, por sua vez, é uma estrutura de dados que tem a seguinte propriedade: o conteúdo de um nó é maior ou igual que o conteúdo dos nós na subárvore enraizada nele. Um heap pode ser visualizado como uma árvore binária e representado por um vetor. A árvore é completamente preenchida em todos os níveis, exceto possivelmente o último nível em que as folhas estão o mais à esquerda possível.

Neste trabalho você deve implementar uma fila de prioridade usando um max-heap.

2 Implementação

Considere que cada elemento do conjunto tem um identificador (**id**) de tipo inteiro e uma prioridade (**prior**) de tipo float, sendo que 0<=prior<=900000. Se você escolheu a linguagem C, você deve criar uma estrutura ELEMENTO que tem esses dois campos. Se você escolheu a linguagem Java, você deve criar uma classe com esses 2 atributos.

Considere que a fila de prioridade tem:

- um vetor **A** que representa um heap de elementos do tipo ELEMENTO
- **maxElementos** de tipo inteiro que representa o tamanho máximo de elementos de A.
- **m** de tipo inteiro que representa a quantidade atual de elementos do heap.

Se você escolheu a linguagem C, você deve criar uma estrutura FILADEPRIORIDADE que contém esses três campos. Se você escolheu a linguagem Java, você deve criar uma classe com esses 3 atributos.

Os métodos da fila de prioridade usando max-heap que devem ser implementados são:

- Heap-Max(A,m): devolve o elemento que possui a maior prioridade do vetor A e que contém m elementos. Se não existem elementos na fila de prioridade o método deve devolver -1 -1,0.
- Heap-Extract-Max(A,m): remove e devolve o elemento que possui a maior prioridade do vetor A e que contém m elementos. Se não existem elementos na fila de prioridade o método deve devolver -1 -1,0.
- Heap-Insert(A, m, elemento): insere o elemento no vetor A que contém m elementos. Se foi possível inserir o elemento, o método devolve T. Caso contrário devolve F.
- Heap-Print(A,m): imprime o vetor A que contém m elementos.

Os métodos auxiliares a serem implementados são:

- Max-Heapify(A,i): recebe o vetor A e o índice i, tal que as árvores com raízes nos filhos esquerdo e direito do nó i são max-heaps. Utilizado como método auxiliar do Heap-Extract-Max.
- Heap-Increase-Key(A,i, prior): aumenta o valor da prioridade para prior do elemento que está na posição i do vetor A. Utilizado como método auxiliar do Heap-Insert.

No programa principal deve ser criada uma fila de prioridades vazia com maxElementos= 4000.

Caso necessário, podem criar métodos auxiliares, usar outras assinaturas para os métodos e passar mais parâmetros caso necessário.

3 Linguagem

O trabalho deve ser desenvolvido na linguagem C, C++ ou Java.

4 Casos de teste

Entrada
A entrada contém apenas um caso de teste. A primeira linha contém um inteiro Q, que é o número de operações que devem ser executadas na fila de prioridades. As próximas Q linhas são as operações, sendo:

1: devolve o elemento que possui a maior prioridade (1 ponto)
2: remove e devolve o elemento que possui a maior prioridade (1 ponto)
3 id prior: insere o elemento com identificador id e prioridade prior na fila de prioridade (2 pontos)
4: imprime a fila de prioridade (0,5 ponto)

Saída

O programa deve imprimir uma linha para cada operação executada.

Exemplo de entrada 1

```
11
1
3 30 2,5
3 40 5,5
3 20 8,3
1
2
3 80 6,3
3 10 1,5
4
2
4
```

Exemplo de saída 1

```
-1 -1,0
T
T
T
20 8,3
20 8,3
T
T
80 6,3 30 2,5 40 5,5 10 1,5
80 6,3
40 5,5 30 2,5 10 1,5
```

Exemplo de entrada 2

```
12
3 2 5,2
3 1 3,6
4
3 3 17,4
4
3 8 10,8
```

4
2
4
3 10 9,5
1
4

Exemplo de saída 2

T
T
2 5,2 1 3,6
T
3 17,4 1 3,6 2 5,2
T
3 17,4 8 10,8 2 5,2 1 3,6
3 17,4
8 10,8 1 3,6 2 5,2
T
8 10,8
8 10,8 10 9,5 2 5,2 1 3,6

Exemplo de entrada 3

12
3 1 20
3 2 5
3 3 8
3 4 10
3 5 15
3 6 7
3 7 6
3 8 2
4
2
2
4

Exemplo de saída 3

T
T
T
T
T
T
T
T
1 20,0 5 15,0 3 8,0 2 5,0 4 10,0 6 7,0 7 6,0 8 2,0
1 20,0
5 15,0

4 10,0 7 6,0 3 8,0 2 5,0 8 2,0 6 7,0

5 O que você deve entregar

Apenas dois arquivos deverão ser submetidos no Tidia:

- O arquivo fonte contendo a implementação incluindo comentários no código.
- Um relatório de máximo 2 páginas incluindo:
    - O pseudocódigo de Heap-Increase-Key, uma descrição curta do pseudocódigo e o cálculo do consumo de tempo. (1 ponto)
    - O pseudocódigo de Heap-Insert, uma descrição curta do pseudocódigo e o cálculo do consumo de tempo. (1 ponto)
    - Uma descrição das principais dificuldades encontradas na implementação. (0,5 ponto)

6 Links relacionados

https://www.codesdope.com/blog/article/priority-queue-using-heap/
https://www.ime.usp.br/~pf/analise_de_algoritmos/aulas/priority.html

Trabalho 3

Tentativa e Erro

(PARTE 2: 3 pontos)

Composição dos grupos: de 2 alunos.

Cada grupo deverá resolver o problema 1122 Livro-Caixa do URI Online Judge.

O trabalho consiste de 2 **partes:**

- **Código fonte da solução que foi aceita na submissão: 50% da nota.** Se o trabalho resolver todos os casos de teste (inclusive testes adicionais ao enunciado) porém falhar na submissão ao sistema URI Online Judge, receberá 75% da nota dessa parte. A técnica de tentativa e erro deve ser usada para resolver o problema.

- **Gravação da apresentação do trabalho: 50% da nota.** gravar uma apresentação de mínimo 5 e no máximo 8 minutos. Gravações com duração fora do tempo estabelecido serão penalizadas. A apresentação deve mostrar como o problema foi resolvido incluindo a análise de complexidade.

Apenas um integrante do grupo deverá enviar no URI o código. A atividade no URI estará aberta a partir do dia **14/11/2020**. Adicionalmente, apenas um aluno do grupo deverá enviar em **Atividades,** na plataforma Tidia, **um arquivo com o vídeo não compactado de no máximo 25MB**. A entrega do trabalho de todos os grupos no Tidia e no URI é até o final do **dia 12/12/2020**.

Para não ter problemas de leitura ou escrita, a seguir o programa em Java com essa parte do código implementada. Caso vocês precisem de outras variáveis no Main, podem acrescentar essas linhas. Vocês podem implementar a solução também nas linguagens C e C++. Outras linguagens não serão aceitas.

DICA: **Guardar os resultados já obtidos em alguma estrutura para não recalcular de novo o mesmo.**

==========

```
import java.util.Scanner;

class Main {
	static int[] numeros;
	static int F;
	static int N;

	public static void main(String args[]) {
		Scanner scan = new Scanner (System.in);
		while(scan.hasNextInt()) {
			N = scan.nextInt();
```

```
                        if(N == 0) break;

                        F = scan.nextInt();

                        numeros = new int[N];


                        for(int i = 0; i < N; i++) {
                                numeros[i] = scan.nextInt();

                        }

                        boolean sucesso = tenta(...);

                        if(!sucesso) System.out.println("*");
                        else System.out.println(...);
                }
        }

        public static boolean tenta( ... ) {

                //Completar o código aqui
        }

}
```
==========