Enunciado: Min Heap operado por comandos (2022-23)

Universidade Aberta

Pretende-se desenvolver um programa em linguagem C++11 padrão que aceite comandos para a gestão de uma árvore binária do tipo min Heap para armazenar itens que são inteiros (positivos ou negativos). Neste caso os itens representam ambos os papéis de chave e de informação. Os comandos de um modo geral devem permitir inserir, remover, listar, além de outros comandos mais específicos. A implementação do Heap é feita com um vetor com capacidade máxima de N elementos, indexados de 0 a N-1.

O heap é inicializado por defeito vazio com capacidade máxima de N=15 itens e é alterado com comandos especificados num ficheiro de entrada fornecido ao programa pela entrada padrão (stdin em C e cin em C++), um comando por linha, podendo um comando ter vários argumentos, com o seguinte formato,

cmd arg0 arg1 ...

onde "cmd" indica o nome do comando a executar, "arg" é um tipo de argumento do comando e "..." a seguir a um argumento indica que este pode ser repetido várias vezes indefinidamente. Na descrição dos comandos, um tipo de argumento pode ser representado por: "item" o item a armazenar (inteiro); "N" capacidade do heap (inteiro >0).

A lista dos comandos a implementar é a indicada a seguir. No caso de mensagens de saída de dados, o seu formato é indicado em estilo da linguagem C.

insert item ...
Comando que insere itens no heap pela ordem apresentada. O heap pode conter itens iguais.

print_min
Comando que imprime o menor item no heap. Formato "Min= %d\n".

print
Comando que imprime toda a árvore do heap. Imprime uma linha com "Heap=\n" seguida de uma linha por cada nível (ocupado) da árvore, com os itens em cada linha separados por um espaço, formato "%d %d ...\n".

dim
Comando que imprime o número de itens no heap. Formato "Heap tem %d itens\n".

dim_max
Comando que imprime o número máximo de itens ou capacidade do heap. Formato "Heap tem capacidade %d itens\n".

clear
Comando que inicializa o heap com zero elementos (todo o conteúdo anterior é descartado/perdido).

delete
Comando que remove o menor item no heap.

heapify_up item ...
Comando que converte o vetor definido pelos itens "item ..." num min Heap. Todo o conteúdo anterior do heap é descartado/perdido. Aplique o algoritmo Bottom-up (de baixo para cima) descrito no livro recomendado.

redim_max N
Comando que redimensiona o número máximo de itens ou capacidade do heap. Todo o conteúdo anterior do heap é descartado/perdido.

Todos os comandos que não possam ser executados por o heap estar vazio devem imprimir uma mensagem com o formato "Comando %s: Heap vazio!\n".

O comando insert, se não conseguir inserir um elemento por o heap estar cheio, deve imprimir uma mensagem com o formato "Comando %s: Heap cheio!\n" e ignorar os restantes itens do comando.

É da responsabilidade do programa assegurar que nenhuma operação que comprometa a estabilidade do programa é executada, por exemplo delete de um heap vazio.

Projete as estruturas de dados (classes) adequadas ao programa que se pretende desenvolver. No que apenas respeita aos atributos (variáveis membro), o heap deve possuir um vetor de nós alocado dinamicamente, um contador que a cada momento indica quantos nós tem o heap e um valor máximo que define a capacidade do heap. Todos os heaps utilizados no programa devem ser instâncias de classes (objetos) e não um conjunto de variáveis "soltas" ou isoladas.

Os métodos (funções membro) são livres. Deverá na elaboração do programa definir e criar os métodos e construtores que considerar mais adequados. O código dos métodos deve ser definido **fora** das classes, que apenas devem ter os respetivos protótipos. Os métodos e os comandos devem ser implementados também tendo em conta a sua eficiência.

No recurso VPL encontram-se templates de 3 ficheiros para o desenvolvimento do programa com instruções, às quais deve ser

acrescentado o código necessário. Para cada caso de teste, o programa é executado com um ficheiro de entrada de dados redirecionado para a entrada padrão, sendo equivalente ao seguinte exemplo na linha de comandos, onde < é o operador de redirecção da entrada padrão,

```
$ prog < input.txt
```

sendo posteriormente os dados de saída do programa comparados com a saída esperada.

O ficheiro de entrada de dados pode ter linhas em branco e o nº de espaços que separa o comando e os argumentos entre si pode ser qualquer. Também podem existir linhas de comentário, caso em que começam obrigatoriamente pelo caracter '#' na 1ª posição.

Projete e teste uma versão do programa que implemente as especificações e comandos pedidos. O programa deve ler e processar o ficheiro de entrada uma linha inteira de cada vez, num ciclo ler-linha processar-linha até chegar ao fim do ficheiro de entrada. É estritamente proibido ler o ficheiro de entrada todo de uma vez para memória.

É importante saber ler o ficheiro de entrada corretamente e eficientemente sem usar código complexo. Sugere-se que seja lida uma linha de cada vez para uma string e analisado o primeiro carater para ver se é um '#', caso em que a linha é um comentário. Caso não seja comentário, criar uma string stream e usar o operador extrator >> para obter o nome do comando. Terá sucesso se não for uma linha em branco. Após saber o nome do comando, sabe-se quais os tipos de argumentos do comando que se seguem. Note-se que o operador extrator ignora ou salta carateres brancos (espaços, tabs e new lines) antes de efetuar uma leitura. Programas que processem uma linha de entrada carater a carater serão severamente penalizados devido à sua complexidade.

No desenvolvimento do programa em C++11 não devem ser utilizadas **variáveis globais** nem ser utilizada a **STL**, nomeadamente no que respeita às estruturas de dados e algoritmos estudados, devendo o aluno escrever o próprio código. Restrições aplicam-se aos includes <array> <deque> <forward_list> <list> <map> <queue> <set> <stack> <unordered_map> <unordered_set> <vector> e em parte de <algorithm>. Em caso de dúvida questionar o seu uso. Não existem restrições para <string>. Também não devem ser usados smart pointers.

Bom trabalho!

FIM

Requested files

minh.h

```
/*
** file: minh.h
**
** min Heap por comandos
** (binary tree on array)
** UC: 21046 - EDAF @ UAb
** e-fólio B 2022-23
**
** Aluno: NNNNN - Nome Apelido
*/

// Defina:
// em minh.h as classes da estrutura de dados
// em minh.cpp a implementação dos métodos das classes da estrutura de dados

#ifndef _MINH_H
#define _MINH_H

#include <iostream>
using namespace std;

// definir int min heap em array
class IMINH {
private:
  // atributos obrigatórios
  int *v,     // vetor com nós
      n,      // num. nós usados
      nv;     // dim max do vetor (capacidade)
```

```
public:
  // construtores
  IMINH(int nmax=15);    // cria heap vazio c/ capacidade nmax nós
  ~IMINH();
  // outros atributos e métodos (protótipos) livres

};

#endif
// EOF
```

minh.cpp

```
/*
** file: minh.cpp
**
** min Heap por comandos
** (binary tree on array)
** UC: 21046 - EDAF @ UAb
** e-fólio B 2022-23
**
** Aluno: NNNNN - Nome Apelido
*/
// Defina:
// em minh.h as classes da estrutura de dados
// em minh.cpp a implementação dos métodos das classes da estrutura de dados
#include "minh.h"



// EOF
```

main-minh.cpp

```
/*
** file: main-minh.cpp
**
** min Heap por comandos
** (binary tree on array)
** UC: 21046 - EDAF @ UAb
** e-fólio B 2022-23
**
** Aluno: NNNNN - Nome Apelido
*/

// Defina:
// em minh.h as classes da estrutura de dados
// em minh.cpp a implementação dos métodos das classes da estrutura de dados
// em main-minh.cpp
//   A entrada/saída de dados
//   As instâncias da classe da estrutura de dados
//   A implementação dos comandos através dos métodos da classe
//   Código auxiliar
//   Não utilize variáveis globais

#include <iostream>
#include <sstream>
#include <string>
#include "minh.h"


int main()
```

```
{
  IMINH minheap;   // exemplo




  return 0;
}


// EOF
```