8ªaula prática

Filas de prioridade

- Faça download do ficheiro ***aed2223_p08.zip*** da página e descomprima-o (contém a pasta ***lib*** ; a pasta ***Tests*** com ficheiros ***box.h***, ***box.cpp***, ***packagingMachine.h***, ***packagingMachine.cpp,*** ***funPQProblem.h,*** ***funPQProblem.cpp*** e ***tests.cpp***; e os ficheiros ***CMakeLists*** e ***main.cpp***)
- No CLion, abra um ***projeto***, selecionando a pasta que contém os ficheiros do ponto anterior.

1. Nesta quadra natalícia, a loja do Sr. Silva decidiu inovar para tornar mais eficiente o envio dos seus produtos. Para isso, comprou um empacotador automático que coloca os objetos em caixas segundo o peso de cada objeto e a capacidade remanescente de cada caixa. Suponha a existência de um conjunto de caixas de capacidade (peso máximo que suporta) ***boxCapacity***, e objetos o_1, o_2, …, o_n com peso w_1, w_2, …, w_n, respetivamente. O objetivo é empacotar todos os objetos sem ultrapassar a capacidade de carga de cada caixa, usando o menor número possível de caixas. Implemente um programa para resolver este problema, usando a seguinte estratégia:
   - Comece por colocar primeiro os objetos mais pesados;
   - Coloque o objeto na caixa mais pesada, que ainda possua carga livre para conter este objeto.

   Guarde os objetos a serem empacotados numa fila de prioridade (`priority_queue<Object>`), sendo o objeto de maior peso o mais prioritário. Guarde as caixas numa fila de prioridade (`priority_queue<Box>`), sendo a caixa de maior prioridade a caixa de menor carga ainda disponível.

```
class Object {
  unsigned id;
  unsigned weight;
public:
  Object(unsigned i, unsigned w);
  bool operator < (const Object& o1)const;
   //…
};

typedef stack<Object> StackObj;
class Box {
  StackObj objects;
  unsigned capacity;
  unsigned free;
public:
  Box(unsigned cap=10);
  bool operator < (const Box& b1) const;
  //…
};
```

```
typedef priority_queue<Object> HeapObj;
typedef priority_queue<Box> HeapBox;

class PackagingMachine {
   HeapObj objects;
   HeapBox boxes;
   unsigned boxCapacity;
public:
   PackagingMachine(int boxCap=10);
    //…
};
```

Nota: deve realizar este exercício respeitando a ordem das alíneas.

1.1 Implemente o membro-função:

unsigned PackagingMachine::loadObjects(vector<Object> &objs)

Esta função recebe como argumento a paleta de objetos a serem empacotados (vector *objs*). Apenas os objetos com peso igual ou inferior à capacidade das caixas são carregados no empacotador. A função retorna o número de objetos efetivamente carregados no empacotador, sendo: o vetor *objs* atualizado com a eliminação desses objetos que são carregados; os objetos carregados guardados na fila de prioridade *objects*, de acordo com a prioridade desta (o objeto mais prioritário é o mais pesado).

1.2 Implemente o membro-função:

Box PackagingMachine::searchBox(Object& obj)

Esta função procura na fila de prioridade *boxes* a próxima caixa com carga remanescente suficiente para guardar o objeto *obj*. Se essa caixa existir, retira-a da fila de prioridade e retorna-a. Caso não exista uma caixa com carga livre suficiente para alojar *obj*, cria uma nova caixa, retornando-a.

Nota: não deve colocar o objeto *obj* na caixa

1.3 Implemente o membro-função:

unsigned PackagingMachine::packObjects()

Esta função guarda os objetos, presentes na fila *objects*, no menor número possível de caixas. Retorna o número de caixas utilizadas. Considere que inicialmente nenhuma caixa está a ser usada

1.4 Implemente o membro-função:

stack<Object> PackagingMachine::boxWithMoreObjects() const

Esta função procura na fila de prioridade *boxes* a caixa que contém o maior número de objetos, e retorna o seu conteúdo (pilha de objetos). Se não houver caixas na fila *boxes*, a função retorna uma pilha vazia.

2. Considere a classe **FunPQProblem** que possui apenas métodos estáticos. Implemente a função:

int FunPQProblem::minCost(const vector<int>& ropes)

Dado um conjunto de cordas de diferentes comprimentos, é necessário ligá-las em uma única corda. O custo para ligar duas cordas é igual à soma dos seus comprimentos. A tarefa é ligar as cordas com custo mínimo.

Sugestão: na escolha de quais cordas ligar, escolha as duas cordas de menor comprimento. Use uma fila de prioridade.

Nota: a classe ***priority_queue*** da STL implementa uma fila de prioridade de máximo. Mas parâmetro *Compare* fornecido pelo utilizador pode alterar o sentido da ordenação, por exemplo, usar std::greater<T> faz com que o menor elemento apareça em ***top()***.

exemplo (para fila de prioridade de inteiros): priority_queue<int, std::vector<int>, std::greater<int>>

Complexidade temporal esperada: $O(n \times \log n)$

Exemplo de execução:

input: ropes = {4, 3, 2, 6}

output: result = 29

Ligar as cordas de comprimento 2 e 3: {4, 5, 6}, custo = 2+3 = 5

Ligar as cordas de comprimento 4 e 5: {9, 6}, custo = 4+5 = 9

Ligar as cordas de comprimento 9 e 6: {}, custo = 9+6 = 15

custo total = 5+9+15 = 29