4ªaula prática - Algoritmos de ordenação

Instruções

- Faça download do ficheiro ***aed2122_p04.zip*** da página da disciplina e descomprima-o (contém a pasta ***lib,*** a pasta ***Tests*** com os ficheiros ***funSortProblem.h, funSortProblem.cpp, product.h, product.cpp, piece.cpp, piece.h*** 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.

Considere a classe **FunSortProblem:**

```
class FunSortProblem {
public:
   FunSortProblem();
   static void expressLane(vector<Product> &products, unsigned k);
   static int minDifference(const vector<unsigned> &values, unsigned nc);
   static unsigned minPlatforms(const vector<float> &arrival,
                                const<float> &departure);
   static void nutsBolts(vector<Piece> &nuts, vector<Piece> &bolts);
};
```

a) Implemente o membro-função:

void FunSortProblem::expressLane(vector<Product> &products, unsigned k)

O João foi ao supermercado e, como está com pressa, pretende usar a caixa expresso para efetuar o pagamento das suas compras. A caixa expresso limita o número de produtos a comprar a *k* unidades, pelo que o João tem de escolher *k* produtos de entre os que pretendia comprar inicialmente (*products*). Como o João é ganancioso, vai escolher os produtos mais baratos; se dois produtos têm o mesmo preço, escolhe o mais leve. Esta função retorna o vetor dos produtos que o João escolhe.

Implemente esta função usando apenas estruturas de dados lineares, e algum(uns) algoritmo(s) de ordenação. Qual a complexidade temporal desta função?

b) Implemente o membro-função:

int FunSortProblem::minDifference (const vector<unsigned> &values, unsigned nc)

O vetor *values* é um vetor de inteiros que representa o número de chocolates em um pacote. Os pacotes de chocolate serão distribuídos por *nc* crianças de modo que:

- cada criança recebe exatamente um pacote
- a diferença entre o maior e o menor número de chocolates dados a uma criança é mínima.

A função retorna a diferença mínima entre o maior e menor número de chocolates dados a uma criança. Se o número de criança é superior ao número de pacotes de chocolate existentes, a função retorna -1.

Implemente esta função usando apenas estruturas de dados lineares, e algum(uns) algoritmo(s) de ordenação. Qual a complexidade temporal desta função?

c) Implemente o membro-função:

unsigned FunSortProblem::minPlatforms (const vector<float> &arrival, const<float> &departure)

Dados os horários de chegada (*arrival*) e partida (*departure*) de todos os comboios que chegam a uma estação ferroviária, encontre o número mínimo de plataformas necessárias, de modo que nenhum comboio fique em espera. Em qualquer momento, a mesma plataforma não pode ser usada tanto para a partida de um comboio como para a chegada de outro. A função retorna o número mínimo de plataformas que a estação ferroviária deve ter.

Implemente esta função usando apenas estruturas de dados lineares, e algum(uns) algoritmo(s) de ordenação. Qual a complexidade temporal desta função?

d) (exercício extra) * Implemente o membro-função:

void FunSortProblem::nutsBolts(vector<Piece> &nuts, vector<Piece> &bolts)

O vetor *nuts* representa um conjunto de porcas (objetos da classe *Piece*) identificadas por um *id* e *diâmetro*. O vetor *bolts* representa um conjunto de parafusos (objetos da classe *Piece*) identificados por um *id* e *diâmetro*. Cada porca corresponde exatamente a um parafuso e cada parafuso corresponde exatamente a uma porca. Uma porca e um parafuso são correspondentes se e só se possuem o mesmo diâmetro. É possível comparar uma porca com um parafuso, mas não é possível comparar diretamente duas porcas ou dois parafusos. A função deve atualizar os vetores *nuts* e *bolts* que deverão conter no mesmo índice a porca e o parafuso correspondentes.

Implemente esta função usando apenas estruturas de dados lineares, devendo a solução apresentar complexidade temporal $O(n \times log n)$.

Sugestão: baseie-se no conceito de partição do algoritmo QuickSort. Escolha um parafuso aleatório, compare-o com todas as porcas e encontre a porca correspondente. Compare a porca correspondente agora encontrada com todos os parafusos, dividindo assim o problema em dois problemas: um consistindo em porcas e parafusos menores que o par encontrado e outro consistindo em porcas e parafusos maiores que o par encontrado.