EP0 - Aeroporto

Relatório do Exerício-Projeto 0 - Aeroporto - da matéria de MAC0323 - Algoritmos e Estruturas de Dados 2 - da Universidade de São Paulo, ministrada pelo Prof. Dr. Carlos E. Ferreira no ano de 2020. Esse relatório foi redigido pelo aluno Lucas Paiolla Forastiere - 11221911 - informando sobre as decisões de projeto e comportamento do programa em testes.

Sobre as decisões de projeto

Implementação da Fila

Solicitou-se que os aviões fossem atendidos em uma estrutura de *fila*, onde o primeiro a fazer contato é o primeiro a sair da fila. Mas o problema conta com diversos casos emergenciais em que é preciso passar aviões na frente de outros e possivelmente realocar aviões que já estavam alocados nas filas.

Por esses motivos, a primeira decisão de projeto foi de fazer uma fila convencional, mas com acesso a pontos especificios para remoção e insersão no meio da fila. Todas essas decisões podem ser vistas no arquivo `Queue.h`. Nele há quatro classes: `QueueNode`, `Queue`, `Pointer_Queue` e `Queue_Iterator`.

Ou seja, implementou-se uma fila em lista ligada com tipo de dado abstrato, como o professor ensinou nas primeiras aulas do semestre. O único problema encontrado foi o fato de que a fila conteria ponteiros para aviões e, por isso, foi preciso criar uma classe `Pointer_Queue`, herdada de `Queue`, que dá override nos métodos necessários para adaptar o comportamento esperado de uma fila para uma fila de ponteiros.

Além disso, implementou-se a classe `Queue_Iterator` inspirada nos iteradores que são encontrados na *STL* do C++. Um *Queue_Iterator* é um "ponteiro" com a capacidade de percorrer a fila e guardar posições estratégicas para o funcionamento do programa. Com eles, foram possíveis de implementar os métodos de remoção e inserção no meio da fila:

- `T removeFrom(Queue_Iterator<T> &it);`
- `void addIn(Queue_Iterator<T> *it, T e);`

Além do método

- `void iterate(const std::function<void(T)> &func);` ,

que usa internamente os iteradores para aplicar a cada elemento da fila a função `func`.

Implementação do Avião

A implementação dos aviões pode ser encontrada nas classes `Plane.h` e `Plane.cpp`. O avião está implementado como uma classe que possui uma série de propriedades listadas e explicadas no arquivo `Plane.h`. A única que se deve chamar atenção é a propriedade `priority` que acabou por não ser utilizada na implementação final do Aeroporto (inicialmente a ideia era utilizar uma Fila de Prioridades, mas isso foi descartado).

Os únicos métodos que valem a pena ser mencionados são:

- `void update();`
- `Plane *createRandomPlane();`
- `Plane *createUserPlane();`

, onde os dois últimos não fazem parte propriamente da classe `Plane`.

O `update` é um método que deve ser chamado a cada instante de tempo e é responsável por atualizar as propriedades do avião com base em sua categoria (se ele está pousando ou decolando - campo `flying`). Uma responsabilidade importante dele é fazer com que o avião se torne uma emergência caso a quantidade de combustível seja menor que o tempo que ele vai esperar para pousar (caso esteja voando) e também caso seu tempo de espera para decolagem passe de 10% do tempo de viagem esperado (caso esteja decolando).

Os métodos `createRadomPlane` e `createUserPlane` são usados respectivamente caso a simulação seja randômica ou não. O primeiro cria um avião randômico com base no banco de dados de possíveis companhias aéreas e destinos (além de sortear o número do avião etc.). Já o segundo vai pedir todas as informações para o usuário. Ambos retornam um ponteiro para o avião criado.

Implementação do Aeroporto

A implementação do aeroporto foi feita em uma classe que pode ser encontrada nos arquivos `Airport.cpp` e `Airport.h`. A classe conta com uma série enorme de variáveis para mostrar estatísticas ao usuário e suas explicações podem ser encontradas em `Airport.h`.

Vale a pena falar apenas dos campos:

- `Pointer_Queue<Plane *> queue[3]` ;
- `int timeToBeFree[3]` ;
- `Queue_Iterator<Plane *> lastVIP[3]` ;

No primeiro campo tempos um array com três filas, uma para cada pista. Essa é uma decisão de projeto bastante crucial, pois significa que vamos decidir em qual fila o avião vai pousar assim que ele

se comunica com a torre. Outra solução possível poderia ser a de ter apenas uma única fila e passar a decisão de qual pista um avião utiliza para o momento em que ele está saindo da fila.

Prefiriu-se optar pela primeira solução. Dessa forma, pode-se saber exatamente quanto tempo levará para um avião sair da fila:

```
* Se queue[i] está vazia, então o avião levará o tempo de manutenção da pista i,
* caso contrário, o avião levará o tempo do último avião na fila + 3 unidades.
```

O `+3` refere-se a uma unidade para pouso/decolagem mais duas unidades de manutenção da fila.

Para que essa implementação dê certo, utilizamos o array `timeToBeFree[]` que é o tempo para que a pista `i` volte ao seu serviço. Um avião pode utilizar a pista `i` se, e somente se, `timeToBeFree[i] == 0` e, ao fazer isso, fazemos `timeToBeFree[i] = 3` , pois a fila fica duas unidades em manutenção e decrementamos seu valor ao fim de cada instante (ou seja, ao final do instante em que o avião pousou, `timeToBeFree[i]` tornar-se-á `2`).

* Decisões sobre aviões emergenciais

Por fim, sobre os iteradores `lastVIP` . Como foram utilizadas uma fila para cada pista, uma decisão de projeto que segue naturalmente é manter a região frontal da fila para os aviões emergenciais (também chamados de *Very Important Planes* dentro do código). Ou seja, sempre que um avião emergencial contacta a torre, devemos escolher a fila em que ele levará menos tempo para sair e inseri-lo imediatamente após o último avião emergencial daquela fila (ou no começo da fila, caso ela não tenha nenhum avião emergencial).

Pois o array `lastVIP[i]` são iteradores que apontam exatamente para onde o último avião emergencial está localizado na fila `i` (ou para a cabeça frontal, caso não existam emergências em `queue[i]`).

Disso, segue o algoritmo que deduz o tempo que o avião emergencial levará para sair da fila `i` :

```
* Se lastVIP[i] == queue[i].getFrontIterator(),
     então o avião levará o tempo de manutenção da pista i,
* caso contrário, o avião levará o tempo de *lastVIP[i] + 3 unidades.
```

, onde `queue[i].getFrontIterator()` representa o iterador para a cabeça frontal e `*lastVIP[i]` indica o último avião emergencial da fila `i` .

Daí surgem algumas possibilidades e coisas a serem feitas:

Como um os aviões emergenciais entram na frente de outros aviões não-emergenciais, é preciso atualizar o tempo em que estes aviões levarão na fila e ver se algum deles passa a se tornar uma emergência. Ou seja, caso o avião esteja numa fila e esteja sobrevoando o aeroporto, mas, por conta de uma emergência, passou a não ter combustível o suficiente para esperar, então nós infelizmente o enviamos para outro aeroporto. Perceba que ao fazer isso, não deixamos que o avião chegue a zero de combustível, pois estamos sempre verificando se os aviões que estão na fila têm combustível suficiente para esperar sobrevoando até sua vez chegar.

Além disso, existe a possibilidade de o avião emergencial que acabou de chegar não ter combustível suficiente nem mesmo para esperar as outras emergências. Nesse caso, como recomendado pelo próprio professor, infelizmente enviamos essa emergência a outro aeroporto.

Sobre os métodos do aeroporto

Vale a pena citar apenas três:

- `void addPlane(Plane *p);`
- `void removePlane(int t_index);`
- `void update();`

Os dois primeiros servem para adicionar e remover aviões das filas do aeroporto. O primeiro adiciona um avião (através de seu ponteiro) escolhendo a fila de acordo com os critérios já mencionados. O segundo remove um avião da fila `t_index` se a fila não estiver fora de serviço.

Já o método `update` é o método que vai gerenciar tudo o que deve ser feito em um instante de tempo dentro da simulação:

- Saber (do usuário ou randômicamente) quantos aviões contatarão a torre;
- Adicionar esses aviões gerados nas filas adequadas;
- Remover os aviões que estão na frente de cada fila (se possível);
- Decrementar o tempo de serviço das pistas;
- Iterar sobre as filas atualizando cada avião (chamando o `update` de cada avião);
- E, por fim, vendo se algum avião virou emergência e tomar a ação adequada.

Sobre os testes solicitados pelos monitores