Relatório do ep0 - mac323

Davi de Menezes Pereira, 11221988 16/03/2020

Como foi feito o Exercício Programa

Breve explicação de como eu entendi o enunciado do ep e como fiz o meu programa (sem muitos detalhes).

Primeiramente, para cada instante t de tempo, o programa lê um número K, o número de aviões entrando em contato naquele momento. Depois disso, ele envia cada avião para sua respectiva fila.

Foram feitas duas filas principais, uma para aviões de emergência e outra para aviões "normais" (que não são de emergência) e na classe dos aviões foi deixado um bool para dizer se o avião é de emergência ou não. Ou seja, não há especificação sobre qual é o tipo da emergência (isto vale para todo o programa: as emergências foram tratadas igualmente porque isso não estava muito claro no enunciado).

Depois disso, temos que mandar os aviões paras as pistas, que são representadas no código por três variáveis do tipo "Plane". Para fazer isso, eu checo quanto tempo o avião "está" na pista (considerando que cada avião gasta 3 tempos para fazer uma ação) e, além disso, checo o tempo estimado do avião. Esse útlimo passo foi uma alternativa encontrada para evitar que o avião seja enviado mais de uma vez pra fila dos que pousaram ou decolaram (para cada avião que já foi mandado pra lista de pouso ou decolagem, eu coloco um valor negativo no tempo estimado). Caso a pista esteja livre, eu coloco primeiro os aviões de emergência, e, se não tiver mais aviões de emergência, coloco algum da fila normal (nesse passo há pequenas alterações para o caso da pista 3). Para maior eficiência na alocação dos aviões, essa etapa é feita primeiramente para a pista 3.

Sobre a eficiência do programa: No enunciado do ep foi pedido pra fazer o gerenciamento das pista de forma eficiente, entretando devido a algumas restrições do enunciado e alguns detalhes não terem ficado muito claros, optei por fazer dessa forma explicada acima (com duas filas principais). Não acho que é a forma mais eficiente, porém se for considerado o número de aviões que devem chegar para 3 pistas, essa maior eficiência não iria ter um impacto tão grande no tempo gasto pelo programa. Além disso, optei por não criar mais classes, porque vi que dava para trabalhar com o que tinha já. (Por exemplo, poderia ter criado uma classe para as pistas, mas preferi usar a classe avião que já havia sido criada).

Sobre a organização do programa: foram feitas classes para os aviões (na verdade é para representar um voo) e uma estrutura de fila, embora essa estrutura não seja a tradicional, uma vez que foram necessárias algumas funções auxiliares que não existem numa fila tradicional. Ainda sobre as filas, optei por não usar template, considerando que a fila só foi usada para um tipo conhecido. Sobre a classe dos aviões, optei por deixar os atributos como público.

Formato do input

O formato considerado para input é de acordo com classe "Plane" (mostrado na figura abaixo).

Listing 1: Parte do código em plane.h.

```
1     class Plane
2     {
3         public:
4             string id;
5             string from_to;
6
7             bool emergency;
8
9             int estimated_time;
10            int waiting_time;
11            /*auxiliar variable to control the time when the flight track ↩
                 will be ocupied*/
12            /*on this atribute, everyone begins with 3*/
13
14        public:
15            Plane(); /* default constructor */
16            Plane(string t_id, string t_from_to, bool t_landing, bool ↩
                 emergency, int t_fuel, int t_estimated_time);
17            /*~plane();*/
18            int track_time;
19            int fuel;
20            bool landing;
21    };
```

O código abaixo indica como estão sendo lidos os aviões. As parte que não são lidas é porque já tem valor definido (como wating time que inicia com 0 e track time que inicia com 3)

Listing 2: Parte do código em airport.cpp que lê os aviões (C++).

```
cin >> K;
for(k = 0; k < K; k++)
{
    cin >> t_id;
    cin >> t_from_to;

    cin >> t_landing;
    cin >> t_emergency;

    cin >> t_fuel;
    cin >> t_estimated_time;

    Plane aux {t_id, t_from_to, t_landing, t_emergency, t_fuel, ↩
       t_estimated_time};

    if(t_emergency) emergencies.push(aux);
    else normal.push(aux);
}
```

Formato da saída

Algum texto aqui.

Além disso, a saída está sendo impressa com um formato específico em 7 partes diferentes para cada instante de tempo (abaixo está a parte do código de impressão das filas).

1) Imprime as 5 diferentes filas de aviões. Cada fila é impressa de acordo com o código da figura abaixo.

2) Tempo de espera dos aviões que querem pousar (no enunciado está dizendo para imprimir a média, mas optei por imprimir o número total)

3) Tempo de espera dos aviões que querem decolar (no enunciado está dizendo para imprimir a média, mas optei por imprimir o número total)

4) Quantidade de combustível dos aviões em espera.

5) Quantidade de combustível dos aviões que pousaram.

6) Quantidade total de aviões em emergência

7) Informação sobre as pistas: se estão livres ou com qual avião (id) elas estão ocupadas.

Listing 3: Parte do código em queue.h que imprime informações dos aviões (C++).

```
1   void Queue::printQueue()
2   {
3       string emergencia, pouso;
4       Node * i;
5       i = ini->next;
6       cout << endl;
7       if(n > 0)
8       {
9           cout << "    |    ID    | DESTINO/ORIGEM | EMERGENCIA | TEMPO ↩
                DE ESPERA | COMBUSTIVEL | POUSO/DECOLAGEM |" << endl;
10          cout << "    ↩
                +----------+----------------+------------+-----------------+------------
                " << endl;
11      }
12      else cout << "     Essa fila nao possui avioes" << endl;
13      while(i != NULL)
14      {
15          if(i->plane.emergency)
16              emergencia = "SIM";
17          else
18              emergencia = "NAO";
19
20          if(i->plane.landing)
21              pouso = "POUSO";
22          else
23              pouso = "DECOLAGEM";
24
25          cout << "    |  " << i->plane.id << " |      " << i->plane.↩
                from_to << "       |    " << emergencia << "     |        ↩
                " << i->plane.waiting_time << "        |     ";
26          if(i->plane.fuel >= 10) cout << i->plane.fuel << "      | " <<↩
                pouso << endl;
27          else cout << i->plane.fuel << "       | " << pouso << endl;
28          i = i->next;
29      }
30      cout << endl;
31  }
```

Exemplos

Os exemplos foram feitos com base nos exemplos enviados pelos monitores no email.

EXEMPLO 1:

Entrada:

```
4 4 LA329 ACA 1 0 4 10 LA563 ADZ 1 0 4 40 LA923 ANF 1 0 4 10 LA734 AQP 1 0 4 10 0 0 0
```

Saída:

No arquivo out.txt

EXEMPLO 2:

Entrada:

```
1 4 LA329 ACA 1 0 2 20 LA563 ADZ 1 0 2 20 LA923 ANF 1 0 2 12 LA734 AQP 1 0 2 32
```

Saída:

No arquivo out2.txt

EXEMPLO 3:
Entrada:

```
19 7 LA329 GRU 0 0 10 20 LA563 GRU 0 0 4 20 LA923 GRU 0 0 5 20 LA734 GRU 0 0 6 40
LA140 GRU 0 0 23 60 JB666 GRU 0 1 10 30 LA832 BJX 1 0 3 10 0 0 0 0 6 LA485 BPS 1 0 10
20 LA300 BHI 1 0 10 20 LA887 BVB 1 0 10 30 LA993 CAU 1 0 10 23 LA554 CBB 1 0 10 13
LA111 GRU 0 0 10 100 0 3 LA344 CGR 1 0 5 10 LA461 CKS 1 0 9 10 LA875 GRU 0 0 20 40 0
0 0 0 0 0 0 3 LA673 GRU 0 1 30 60 LA899 GRU 0 1 30 40 LA505 COR 1 0 1 20 0 0 0
```

Saída:

No arquivo out3.txt

Observações finais

Não consegui resolver como passar a saída dos exemplos de forma que ficasse legível no latex, então vou mandar os arquivos separados no .tar.