

# GESTÃO DE HORÁRIOS

Adriano Machado /Francisco Pires da Ana /José Pedro Evans



# DESCRIÇÃO DO PROBLEMA

Este projeto tem como objetivo o desenvolvimento de um sistema capaz de ajudar na gestão de horários (alteração, visualização). Para este efeito, deve-se escolher as estruturas de dados mais apropriadas e eficientes.



# Estruturas de dados utilizadas

Ao longo do nosso trabalho usamos quatro estruturas de dados da stl (set, map, vector, queue). Set's e map's são implementados recorrendo a árvores binárias balanceadas (*red black tree*).

## ● Set

Esta estrutura de dados é utilizada no armazenamento de estudantes pois permite uma pesquisa logarítmica. Ao longo do nosso programa usamos as seguintes operações

Método	Complexidade Temporal
<b>Insert()</b>	$O(\log n)$
<b>Erase(valor)</b>	$O(\log n)$
<b>Size()</b>	$O(1)$
<b>Find()</b>	$O(\log n)$

## ● Map

Esta estrutura de dados é usada, para fazer corresponder um dia da semana a um conjunto de aulas.

Método	Complexidade Temporal
<b>Operador [ ]</b>	$O(\log n)$
<b>Empty()</b>	$O(1)$

# Estruturas de dados utilizadas

## ● Vector

Esta estrutura de dados foi usada, por exemplo, no armazenamento de horários de um determinado estudante.

Método	Complexidade temporal
Push_back()	$O(1)$
At()	$O(1)$
Size()	$O(1)$

Implementamos um método, `binarySearchSchedules` que aplica pesquisa binária no vetor de horários e retorna o índice no horário da UcClass pretendida. Este método apresenta complexidade temporal  $O(\log N)$ .

## ● Queue

Esta estrutura de dados foi utilizada no armazenamento temporário de pedidos de trocas/inscrições/desinscrições de turmas.

Método	Complexidade Temporal
Push()	$O(\log N)$
Empty()	$O(1)$
Pop()	$O(\log N)$
Front()	$O(1)$



# Classes

## Student

id: str  
name: str  
classes: vector<UcClass>

## UcClass

ucId: str  
classId: str

## Slot

weekDay: str  
startTime: float  
endTime: float  
type: str

## Request

student: Student  
desiredClass: UcClass  
type: str

## ClassSchedule

ucClass: UcClass  
slots: vector<Slot>  
students: set<Student>

## ScheduleManager

students: set<Student>  
schedules: vector<ClassSchedule>  
changingRequests: queue <Request>  
removalRequests: queue <Request>  
enrollmentRequests: queue<Request>  
rejectedRequests: vector<Request>

# Funcionalidades implementadas

## Leitura dos dados fornecidos

O método `readFiles()` é chamado. Este por sua vez chama os métodos:

- `createSchedules()` – Lê o ficheiro `classes_per_uc.csv` e adiciona ao vetor de horários(`schedules`) objetos da classe `ClassSchedule` com a devida `UcClass`, mas com o vetor de slots e sets de estudantes vazios;
- `setSchedules()` – Lê o ficheiro `classes.csv` e atualiza os objetos do vetor `schedules` com os devidos slots,
- `createStudents()` – Lê o ficheiro `students_classes.csv` e adiciona objetos `Student` ao set `Students`; adiciona também a cada objeto do vetor `schedules` os estudantes que pertencem ao mesmo par turma/cadeira (`UcClass`);

1	UcCode,ClassCode
2	L.EIC001,1LEIC01
3	L.EIC001,1LEIC02

Fig 1. – `classes_per_uc.csv`

1	ClassCode,UcCode,Weekday,StartHour,Duration,Type
2	1LEIC01,L.EIC001,Monday,10.5,1.5,TP
3	1LEIC02,L.EIC001,Thursday,9.5,1.5,TP

Fig 2. – `classes.csv`

1	StudentCode,StudentName,UcCode,ClassCode
2	201920727,Ines,L.EIC001,1LEIC05
3	201920727,Ines,L.EIC002,1LEIC05

Fig 3. – `students_classes.csv`

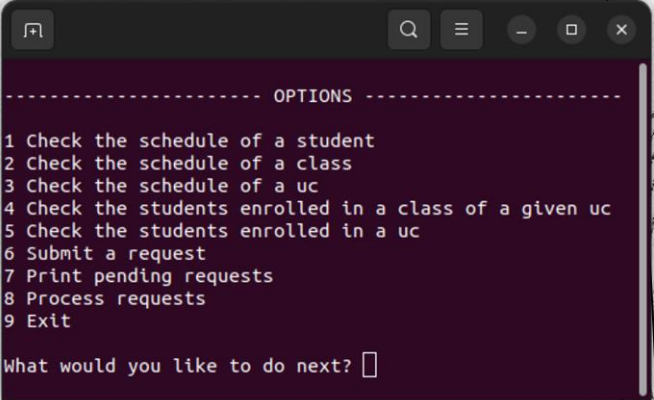


# Funcionalidades implementadas

## Visualização dos dados fornecidos

O nosso programa permite-nos visualizar:

- Horário de um estudante;
- Horário de uma turma;
- Horário de uma cadeira;
- Estudantes inscritos num dado par turma/cadeira;
- Estudantes inscritos numa cadeira



```
----- OPTIONS -----
1 Check the schedule of a student
2 Check the schedule of a class
3 Check the schedule of a uc
4 Check the students enrolled in a class of a given uc
5 Check the students enrolled in a uc
6 Submit a request
7 Print pending requests
8 Process requests
9 Exit

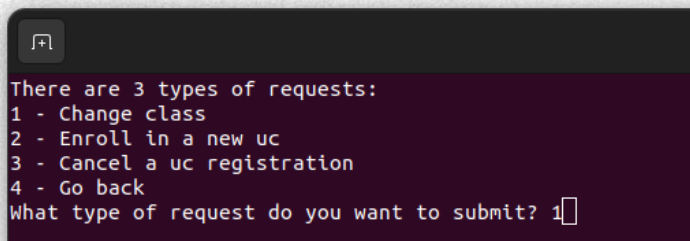
What would you like to do next? █
```

# Funcionalidades implementadas

## Pedidos de alteração de horários

Existem três tipos de pedidos:

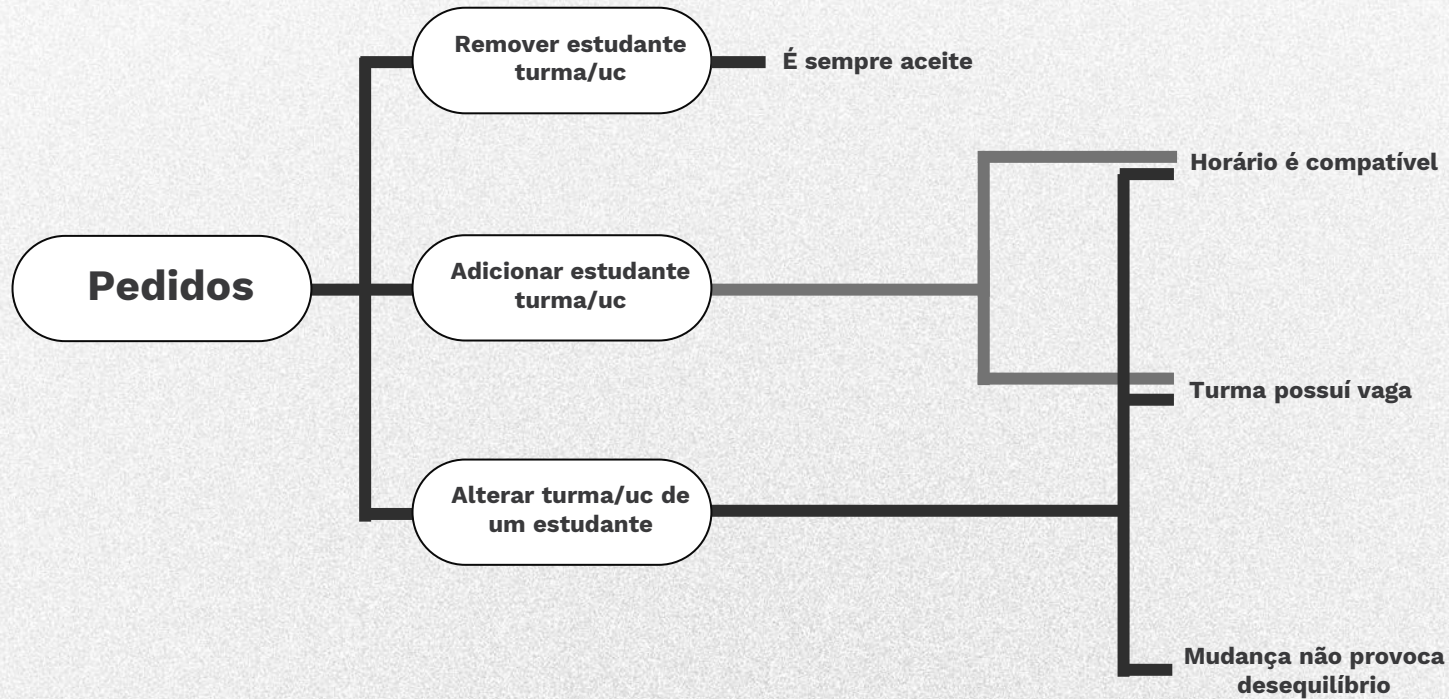
- Pedido de mudança de turma a uma dada cadeira (Changing);
- Pedido de inscrição numa dada cadeira (Enrollment)
- Pedido de cancelamento de inscrição (Removal)



```
There are 3 types of requests:  
1 - Change class  
2 - Enroll in a new uc  
3 - Cancel a uc registration  
4 - Go back  
What type of request do you want to submit? 1
```



# Destaque de Funcionalidade



## Turma possui vaga

```
bool SchedulerManager::requestExceedsCap(const Request &request) const{
    vector < ClassSchedule> classesUc = classesOfUc( ucId: request.getDesiredUcClass().getUcId()); //0(n)
    sort( first: classesUc.begin(), last: classesUc.end(), comp: [](const ClassSchedule &c1, const ClassSchedule &c2)
        return c1.getStudents().size() < c2.getStudents().size();
    });

    int cap = classesUc.back().getNumStudents();
    if(classesUc[0].getNumStudents() == cap) cap++;
    return cap < getNumberOfStudentsUcClass( ucClass: request.getDesiredUcClass()) + 1;
}
```

No nosso programa o nº de máximo de estudantes permitidos (Cap) corresponde ao nº máximo de estudantes atualmente inscritos numa turma dessa cadeira. No da cadeira tiver todas as turmas com o mesmo nº de alunos, incrementamos o cap.

## Horário é compatível

```
bool SchedulerManager::requestHasCollision(const Request &request) const{
    Student student = request.getStudent(); //0(1)
    UcClass desiredClass = request.getDesiredUcClass(); //0(1)
    vector<UcClass> studentClasses = student.getClasses(); //0(1)
    for (const UcClass &ucClass : studentClasses){
        if(classesOverlap( c1: ucClass, c2: desiredClass)) return true; //(log n) + 0(ln)
    }
    return false;
}
```

Para verificar se o horário do aluno é compatível com a mudança, percorremos todas as aulas a que o aluno está atualmente inscrito e verificamos se esta tem conflito de horário com as aulas a inscrever.

## Alteração provoca desequilíbrio

```
bool SchedulerManager::requestProvokesDisequilibrium(const Request &request) const {
    int numFormerClass = getNumberOfStudentsUcClass( ucClass: (getFormerClass(request))) -1;
    int numNewClass = getNumberOfStudentsUcClass( ucClass: request.getDesiredUcClass()) + 1;

    return (numNewClass) - (numFormerClass) >= 4;
}
```

Dizemos que uma mudança provoca desequilíbrio numa turma se após a mudança, a diferença entre o nº de alunos é superior ou igual a quatro.

É de notar que a passagem de uma turma com mais alunos para uma turma com menos alunos é sempre permitido(melhora o equilíbrio)



# Exemplo de execução

```
There are 3 types of requests:
1 - Change class
2 - Enroll in a new uc
3 - Cancel a uc registration
4 - Go back
What type of request do you want to submit? 1
Please insert the student's UP number: 202028632

Student: Fatima - 202028632
Classes: L.EIC003 1LEIC16 | L.EIC011 2LEIC05 | L.EIC012 2LEIC06 | L.EIC013 2LEIC07 | L.EIC014 2LEIC07

The following information is related to the class you want to change to, for a certain curricular unit.
Please insert the uc code: L.EIC003
Please insert the class code: 1LEIC15
>> Request submitted successfully.

Insert any key to continue:
```

A Fátima submeteu um pedido de alteração de turma a FP (L.EIC003) da turma 16 para a turma 15.

```
----- OPTIONS -----
1 Check the schedule of a student
2 Check the schedule of a class
3 Check the schedule of a uc
4 Check the students enrolled in a class of a given uc
5 Check the students enrolled in a uc
6 Submit a request
7 Print pending requests
8 Process requests
9 Exit

What would you like to do next? 8

Do you want to see all pending changingRequests first? (y/n) n

>> Accepted removal requests:

>> Accepted changing requests:
    Student: Fatima - 202028632 | Requested class: L.EIC003 - 1LEIC15
>> Accepted enrollment requests:

>> All Requests were accepted!

Insert any key to continue:
```

Acionada a opção de processar os pedidos no menu, verificamos que o pedido da Fátima foi executado com sucesso. Após o termino do programa a informação no ficheiro student\_classes.csv será devidamente atualizada.

# Dificuldades

Este projeto foi um desafio por diversos motivos, especialmente por ser o nosso primeiro. Uma das maiores dificuldades que encontramos foi o cálculo da complexidade dos nossos métodos uma vez que misturou diversas variáveis.

Qualquer que tenha sido o desafio, eventualmente superamos e sem dúvida aprofundamos os nossos conhecimentos com este trabalho.

**Esforço de cada elemento:** Adriano Machado: 1/3

Francisco Pires da Ana: 1/3

José Pedro Evans: 1/3