Algoritmos e Estruturas de Dados

Davide Teixeira – up202109860

Pedro Oliveira – up202108669

Descrição do Problema

01 A Funcionalidade

Pretende-se fazer um programa que permita ajudar na gestão de horários;

02 As Estruturas de Dados

Utilização das estruturas de dados certas para cada dado que se pretende armazenar;

03 Os Objetos

Utilizar a Programação Orientada ao Objeto de forma a organizar e modular o problema;

04 Os Algoritmos

Necessidade de criação de algoritmos para pesquisar, alterar, inserir, remover e listar objetos das estruturas de dados.

Descrição da Solução

1. Criação das classes que melhor representam o problema dado;
2. Criação de métodos que permitam ler e escrever ficheiros .csv e que criem os objetos a partir deles;
3. Utilização das Estruturas de Dados que melhor se adequavam ao problema dado e implementá-las dentro das classes;
4. Criação de vários algoritmos úteis(de forma mais eficiente possível) que permitissem relacionar as classes – Ex: Algoritmo que, dado um objeto da classe “UcTurma”, retornava o objeto da classe “Horario” correspondente;
5. Criação de um “menu amigável” que permita aceder e consultar e alterar os dados guardados nas estruturas de dados.

Algoritmos Relevantes

- Procura de um Estudante;
- Verificação da Sobreposição de Horários;
- Override de vários operadores de várias classes;

```
bool UcTurma::operator< (const UcTurma &other) const{
    if (codUC == other.getcodUC()){
        return (codTurma < other.getcodUC());
    }
    return codUC < other.getcodUC();
}
```

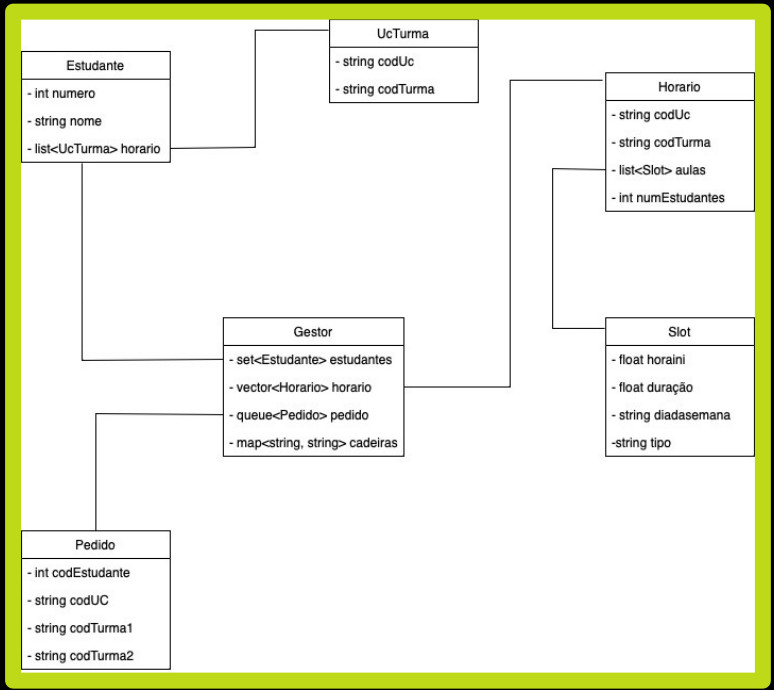
```
bool Pedido::verifyOverlap(list<Slot> horario)  {
    filterTP( &: horario);
    for(auto it1 : iterator<Slot, void *> = horario.begin(); it1 != horario.end(); it1++) {
        auto it2 : iterator<Slot, void *> = it1;
        for(++it2; it2 != horario.end(); it2++) {
            if(it1->getDiaDaSemana() == it2->getDiaDaSemana()) {
                float start1 = it1->gethoraini(), start2 = it2->gethoraini();
                float end1 = start1 + it1->getduaracao(), end2 = start2 + it2->getduaracao();
                if(start2 < end1 && start1 < end2 ) return false;
            }
        }
    }

    return true;
}
```

```
_Rb_tree_const_iterator<Estudante> Gestor::searchStudent(int code) {
    list<UcTurma> temp;
    temp.emplace_back( i: "",  j: "");
    Estudante ghost = Estudante( i: code,  j: "",  k: temp);
    return estudantes.find( k: ghost);
}
```

```
void Pedido::filterTP(list<Slot> &horario) {
    auto it : iterator<Slot, void *> = horario.begin();
    while(it != horario.end()) {
        if(it->gettipo() != "TP") {
            it = horario.erase( p: it);
        } else {
            it++;
        }
    }
}
```

Diagrama/Enumeração das Classes



Funcionalidades Implementadas

- Pesquisa de um estudante e do seu horário(parcial);

```
void Gestor::HorariodoEstudante(int numero){

    Estudante student = PesquisarEstudante(numero);

    if (student.getnome() == "error"){
    }
    else{
        list<UcTurma> turmas = student.gethorario();
        vector<pair<Slot, UcTurma>> mondays;
        vector<pair<Slot, UcTurma>> tuesdays;
        vector<pair<Slot, UcTurma>> wednesdays;
        vector<pair<Slot, UcTurma>> thursdays;
        vector<pair<Slot, UcTurma>> fridays;

        for (UcTurma turma : turmas){
            list<Slot> k;
            Horario temp = getHorariobyUcTurma(turma);
            for (Slot i : temp.getaulas()){
                if (i.getDiaDaSemana() == "Monday")
                    mondays.push_back(pair<Slot, UcTurma>( &: i,  &: turma));
                else if (i.getDiaDaSemana() == "Tuesday")
                    tuesdays.push_back(pair<Slot, UcTurma>( &: i,  &: turma));
                else if (i.getDiaDaSemana() == "Wednesday")
                    wednesdays.push_back(pair<Slot, UcTurma>( &: i,  &: turma));
                else if (i.getDiaDaSemana() == "Thursday")
                    thursdays.push_back(pair<Slot, UcTurma>( &: i,  &: turma));
                else if (i.getDiaDaSemana() == "Friday")
                    fridays.push_back(pair<Slot, UcTurma>( &: i,  &: turma));
                    }
        }
```

```
        sort( first: mondays.begin(),  last: mondays.end(),  comp: cmp);
        sort( first: tuesdays.begin(),  last: tuesdays.end(),  comp: cmp);
        sort( first: wednesdays.begin(),  last: wednesdays.end(),  comp: cmp);
        sort( first: thursdays.begin(),  last: thursdays.end(),  comp: cmp);
        sort( first: fridays.begin(),  last: fridays.end(),  comp: cmp);


        cout << "====================================" << "\n";
        cout << "Segunda-Feira: ";

        printHorario( vetor: mondays, cadeiras);

        cout << "Terça-Feira: ";

        printHorario( vetor: tuesdays, cadeiras);

        cout << " " << "\n";
        cout << "Quarta-Feira: ";

        printHorario( vetor: wednesdays, cadeiras);

        cout <<" " << "\n";
        cout << "Quinta-Feira: ";

        printHorario( vetor: thursdays, cadeiras);

        cout << " " << "\n";
        cout << "Sexta-Feira: ";

        printHorario( vetor: fridays, cadeiras);
    }
}
```

Funcionalidades Implementadas

- Listagem dos estudantes com mais de n UC's (Ordem Alfabética, crescente de número de UC´s e por número) – Listagem parcial.

```
void Gestor::maisNUcsAlfabetico(int n) {
    set<Estudante, ordemAlfabeticaStruct> temp( f: estudantes.begin(), l: estudantes.end());
    set<Estudante>::iterator it;
    cout << "Id|Nome|n" << "\n";
    for (it = temp.begin(); it != temp.end(); it++) {
        if (it->gethorario().size() >= n){
            cout << it->getcodigo() << "|" << it->getnome() << "|" << it->gethorario().size() <<"\n";
        }
    }
}
void Gestor::maisNUcsNumero(int n){
    set<Estudante, ordemNumUcStruct> temp( f: estudantes.begin(), l: estudantes.end());
    set<Estudante>::iterator it;
    cout << "Id|Nome|n" << "\n";
    for (it = temp.begin(); it != temp.end(); it++){
        if(it->gethorario().size() >= n){
            cout << it->getcodigo() << "|" << it->getnome() << "|" << it->gethorario().size() << "\n";
        }
    }
}

void Gestor::maisNUcs(int n){
   set<Estudante>::iterator it;
   cout << "Id|Nome|n" << "\n";
   for (it = estudantes.begin(); it != estudantes.end(); it++){
       if (it->gethorario().size() >= n){
            cout << it->getcodigo() << "|" << it->getnome() << "|" << it->gethorario().size() << "\n";
       }
   }
}
```

Funcionalidades Implementadas

- Listagem das turmas (ordenação por UC, ordenação crescente e decrescente) – Listagem completa

```
vector<Horario> temp = gestor.getHorario();
printOcupacao(temp, gestor);
op1 = 'q';
```

```
void printOcupacao(vector<Horario> temp, Gestor gestor){
    cout << "Codigo UC|Nome Uc|Código Turma|Número de Estudantes" << "\n";
    for (Horario hor : temp){
        cout << hor.getcodUC() << "|" << gestor.cadeiras[hor.getcodUC()] << "|" <<
            hor.getcodTurma() << "|" << hor.getNumEstudantes() << "\n";
    }
}
```

```
vector<Horario> temp = gestor.getHorario();
sort( first: temp.begin(), last: temp.end(), comp: compBynumEstudantes);
printOcupacao(temp, gestor);
sort( first: temp.begin(), last: temp.end());
op1 = 'q';
```

```
vector<Horario> temp = gestor.getHorario();
sort( first: temp.begin(), last: temp.end(), comp: compBynumEstudantes);
reverse( first: temp.begin(), last: temp.end());
printOcupacao(temp, gestor);
sort( first: temp.begin(), last: temp.end());
op1 = 'q';
```

Destaque de Funcionalidade

Algoritmo que para um objeto da classe UcTurma, utiliza a pesquisa binária para encontrar o objeto da classe Horario que lhe é homólogo

```
Horario Gestor::getHorariobyUcTurma(UcTurma turma){
    list<Slot> lista;
    Horario temp = Horario( i: turma.getcodUC(), j: turma.getcodTurma(), k: lista);
    std::vector<int>::iterator it;
    int low = 0;
    int high= horario.size()-1;
    int middle;
    while (low < high){
        int middle = low + (high - low) / 2;
        if (horario[middle] < temp) low = middle+1;
        else high = middle;
    }
    middle = low + (high - low) / 2;
    if (horario[middle] == temp) return horario[middle];
    else return temp;
}
```

Principais Dificuldade Encontradas

- Dificuldades na modulação do Problema – não saber por onde começar a realização do trabalho

Esforço de Cada elemento do Grupo

Davide Teixeira

1. Elaboração das classes;
2. Algoritmos de listagem;
3. Criação do menu “amigável”;

Pedro Oliveira

1. Leitura e escrita dos Ficheiros .csv;
2. Processamento e armazenamento dos pedidos (incluindo algoritmos de verificação de sobreposição, etc.);