Algoritmos e Tipos Abstratos de Dados 2019/2020

Licenciatura em Engª Informática

Turma: EI-07	Data/Horário Lab: Terça-Feira	Docente: Bruno Silva

Nº aluno	Nome
190221009	Bernardo Mota
190221032	Tiago Branco

Índice

Introdução

Este relatório é feito no âmbito da disciplina de Algoritmos e Tipos Abstratos de Dados e tem o objetivo de demonstrar o desenvolvimento de algoritmos capazes de filtrar dados e devolver informação compacta e útil utilizando a linguagem de programação C, sendo o tema abordado o Covid-19.

São utilizados os "Abstract Data Types" mapa e lista, sendo o mapa utilizado para guardar informações sobre regiões geográficas e a lista para listar informações de pacientes com um histórico de infeção. Tanto para o mapa de regiões como para a lista de pacientes foi utilizada a implementação de ArrayList visto que a maioria das operações pedidas implicam percorrer toda a lista (complexidade linear) e/ou extrair dados de elementos que podem estar em qualquer posição no respetivo ADT (complexidade constante).

Dentro da pasta do código fonte desenvolvido com o Visual Studio Code poderão ser encontrados os seguintes módulos:

- date - para criar uma estrutura representativa de uma data e implementar métodos auxiliares que realizem operações com a mesma.
- patient – para criar uma estrutura que contém informações úteis sobre um paciente e implementar métodos que atuem sobre a mesma.
- region – utilizada para estabelecer a estrutura representativa de uma região e respetivo método print.
- regionName – implementa a estrutura base para as chaves do mapa de regiões
- main – contem os métodos principais que implementam as funcionalidades dos comandos e o método que executa o programa;
- utils – serve principalmente para implementar métodos para ler e importar dados de ficheiros externos e funções auxiliares aos métodos no módulo main.

Para além destes existem também os módulos de definição e implementação dos tipos abstratos de dados Map e List com ArrayList.

Foram implementadas todas as funcionalidades propostas.

Descrição de Comandos

LOADP

Utiliza o método importPatients que por sua vez utiliza importPatientsFromFile para importar os dados de um ficheiro de pacientes, sendo estes passados para uma estrutura PtList em que cada elemento contém dados de um paciente. O método createPatientFromString é responsável por extrair da linha de texto os atributos necessários para instancializar um novo paciente.

Complexidade O(n) resultante da implementação da função importPatientsFromFile, cuja duração de execução irá variar segundo o número de linhas do ficheiro patients.csv.

```
void importPatients(PtList *listPatients)
{
    char input[20];
    printf("Insira nome de um ficheiro de pacientes> ");

    fgets(input, sizeof(input), stdin);
    input[strlen(input) - 1] = '\0';

    importPatientsFromFile(input, listPatients);
}
```

LOADR

Utiliza um método similar ao anterior e passa os dados lidos para uma estrutura do tipo mapa, em que cada elemento representa dados de uma região.

Também de forma similar ao anterior, a complexidade O(n) resulta da implementação da função importRegionsFromFile que está dependente do número de linhas no ficheiro regions.csv.

```
void importRegions(PtMap *mapRegions)
{
    char input[20];
    printf("Insira nome de um ficheiro de regiões> ");

    fgets(input, sizeof(input), stdin);
    input[strlen(input) - 1] = '\0';

    importRegionsFromFile(input, mapRegions);
}
```

CLEAR

Utiliza os métodos listClear e mapClear da implementação de estrutura de dados respetiva para apagar os elementos presentes na lista de pacientes e mapa de regiões, devolvendo a quantidade de registos apagados.

```
void clear(PtList listPatients, PtMap mapRegions)
{

    int numberPatients;
    listSize(listPatients, &numberPatients);

    int numberRegions;
    mapSize(mapRegions, &numberRegions);

    listClear(listPatients);
    mapClear(mapRegions);

    printf("%d records deleted from Patients\n", numberPatients);
    printf("%d records deleted from Regions\n", numberRegions);
}
```

AVERAGE

O comando AVERAGE utiliza o método showAverage que, por sua vez, utiliza o método getPatientsByStatus para produzir 3 novas listas de pacientes que filtram os pacientes por cada tipo de estatuto.

A complexidade é O(n) devido à utilização de funções que se limitam a percorrer as listas de pacientes (ArrayList) de forma linear.

```
void showAverage(PtList listPatients)
{
    int total;
    listSize(listPatients, &total);

    if (total > 0)
    {
        PtList deceasedPatients = getPatientsByStatus(listPatients, "deceased");
        PtList releasedPatients = getPatientsByStatus(listPatients, "released");
        PtList isolatedPatients = getPatientsByStatus(listPatients, "isolated");

        float avgd = getAverageAge(deceasedPatients);
```

```
        float avgr = getAverageAge(releasedPatients);
        float avgi = getAverageAge(isolatedPatients);

        listDestroy(&deceasedPatients);
        listDestroy(&releasedPatients);
        listDestroy(&isolatedPatients);

        printf("Average Age for deceased patients: <%f>\n", avgd);
        printf("Average Age for released patients: <%f>\n", avgr);
        printf("Average Age for isolated patients: <%f>\n", avgi);
    }
    else
    {
        printf("No patient data available...\n");
    }
}



PtList getPatientsByStatus(PtList list, char *status)
{
    PtList patientsByStatus = listCreate(10);

    int pbsSize;
    listSize(patientsByStatus, &pbsSize);

    int size;
    listSize(list, &size);

    for (int i = 0; i < size; i++)
    {
        ListElem pat;
        listGet(list, i, &pat);
        if (equalsStringIgnoreCase(pat.status, status))
        {
            listAdd(patientsByStatus, pbsSize, pat);
            listSize(patientsByStatus, &pbsSize);
        }
    }

    return patientsByStatus;
}
```

De seguida é aplicado o método getAverageAge a cada uma das 3 listas auxiliares, que calcula e devolve a respetiva média de idades dos pacientes nelas contidos, sendo estas apresentadas ao utilizador.

As listas auxiliares são libertadas de memória antes de terminar a função.

```
float getAverageAge(PtList list)
{
    int size;
    listSize(list, &size);

    int count = 0;
    int sum = 0;

    for (int i = 0; i < size; i++)
    {
        Patient pat;
        listGet(list, i, &pat);
        if (pat.birthYear > 0)
        {
            sum += patientGetAge(pat);
            count++;
        }
    }
    if (count > 0)
        return ((float)sum / (float)count);
    else
        return -1;
}
```

Por sua vez o método getAverageAge utiliza outro método auxiliar patientGetAge para calcular a idade de cada paciente relativamente ao ano atual.

```
int patientGetAge(Patient p)
{
    if (p.birthYear == -1)
    {
        return -1;
    }
    time_t t = time(NULL);
    struct tm tm = *localtime(&t);
    return (tm.tm_year + 1900 - p.birthYear);
}
```

FOLLOW

Este comando implementa um método que recebe input do utilizador, percorre a lista recebida para verificar se o input corresponde a um id válido de paciente e, caso seja, apresenta informação relativa ao mesmo. Se o paciente tiver referência relativa ao paciente que o infetou, a informação desse paciente também é mostrada, este processo repete-se até que um paciente não tenha tal referência.

A complexidade é O(n), já que a função getPatientById tem o potencial de percorrer toda a lista de pacientes.

```
void follow(PtList listPatients)
{
    char input[20];
    printf("Insira ID de um paciente> ");

    fgets(input, sizeof(input), stdin);
    input[strlen(input) - 1] = '\0';

    long int id = strtol(input, NULL, 10);

    int success;

    Patient p;
    success = getPatientById(listPatients, id, &p);

    if (success == 0)
    {
        printf("Paciente não encontrado.\n");
        return;
    }
    printf("Following Patient: ");
    printPatientShort(p);

    while (p.infectedBy != -1)
    {
        success = getPatientById(listPatients, p.infectedBy, &p);
        if (success == 0)
        {
            break;
        }
        printf("contaminated by: ");
        printPatientShort(p);
    }
    printf("contaminated by: unknown\n");
}
```

SEX

Comando implementado com o método showSex que, semelhante à implementação do comando AVERAGE, começa por filtrar os pacientes por sexo através do método getPatientsBySex (que gera duas listas auxiliares) e calcula a percentagem respetiva de pacientes por sexo usando o número de pacientes nesta listas auxiliares.

Complexidade O(n) pois o método getPatientsBySex percorre toda a lista de pacientes.

```
void showSex(PtList listPatients)
{
    int total;
    listSize(listPatients, &total);

    if (total > 0)
    {
        PtList malePatients = getPatientsBySex(listPatients, "male");
        PtList femalePatients = getPatientsBySex(listPatients, "female");

        int male;
        int female;

        listSize(malePatients, &male);
        listSize(femalePatients, &female);

        listDestroy(&malePatients);
        listDestroy(&femalePatients);

        float percentFem = (((float)female / (float)total) * 100);
        float percentMale = (((float)male / (float)total) * 100);
        float percentUnknown = 100 - (percentFem + percentMale);

        printf("Percentage of Females: %04.2f%%\n", percentFem);
        printf("Percentage of Males: %04.2f%%\n", percentMale);
        printf("Percentage of unknown: %04.2f%%\n", percentUnknown);
        printf("Total of patients: %d\n", total);
    }
    else
    {
        printf("No patient data available...\n");
    }
}


PtList getPatientsBySex(PtList list, char *sex)
{
    PtList patientsBySex = listCreate(10);
```

```
    int pbsSize;
    listSize(patientsBySex, &pbsSize);

    int size;
    listSize(list, &size);

    for (int i = 0; i < size; i++)
    {
        ListElem pat;
        listGet(list, i, &pat);
        if (equalsStringIgnoreCase(pat.sex, sex))
        {
            listAdd(patientsBySex, pbsSize, pat);
            listSize(patientsBySex, &pbsSize);
        }
    }

    return patientsBySex;
}
```

SHOW

SHOW, implementado com o método showPatient, apresenta informação detalhada sobre um único paciente caso o input do utilizador seja equivalente a um id de paciente. A formatação devida da informação é feita com o método auxiliar printPatientLong.

Complexidade O(n) derivado da execução da função getPatientById que tem o potencial de percorrer toda a lista de pacientes.

```
void showPatient(PtList listPatients)
{
    char input[20];
    printf("Insira ID de um paciente> ");

    fgets(input, sizeof(input), stdin);
    input[strlen(input) - 1] = '\0';

    long int id = strtol(input, NULL, 10);

    int success;

    Patient p;
    success = getPatientById(listPatients, id, &p);

    if (success == 0)
    {
        printf("Paciente não encontrado.\n");
```

```
        return;
    }

    printPatientLong(p);
}

void printPatientLong(Patient p)
{
    printf("ID: %ld\n", p.id);
    printf("SEX: %s\n", p.sex);
    int age = patientGetAge(p);
    if (age != -1)
    {
        printf("AGE: %d\n", age);
    }
    else
    {
        printf("AGE: unknown\n");
    }
    printf("COUNTRY/REGION: %s/%s\n", p.country, p.region);
    printf("INFECTION REASON: %s\n", p.infectionReason);
    printf("STATE: %s\n", p.status);

    int illnessDays = patientGetIllnessDays(p);
    if (illnessDays != -1)
    {
        printf("NUMBER OF DAYS WITH ILLNESS: %d\n", patientGetIllnessDays(p));
    }
    else
    {
        printf("NUMBER OF DAYS WITH ILLNESS: unknown\n");
    }
}
```

TOP5

Este comando exibe os 5 pacientes que mais demoraram a recuperar. É primeiramente percorrido a lista de pacientes, se o estatuto do paciente for “released”, é utilizada a função auxiliar patientGetIllnessDays para calcular a diferença de dias entre a data de diagnóstico e a data de recuperação do mesmo e, caso este número seja maior que algum dos pacientes já no array fixo top5 (onde são guardados os pacientes que mais demoraram a recuperar), ou caso este array não esteja completamente preenchido, o paciente é inserido no array (movendo outros elementos se necessário para criar espaço) que será depois exibido na consola após o final do ciclo.

Complexidade O(n^2) porque, enquanto que a lista total de pacientes é percorrida apenas uma vez, cada novo paciente encontrado necessita de ser comparado com todos o que já se encontram no top5.

```
void showTop5(PtList listPatients)
{
    int size;
    listSize(listPatients, &size);

    Patient top5[5];
    int count = 0;

    for (int i = 0; i < size; ++i)
    {
        Patient p1;
        listGet(listPatients, i, &p1);
        if (strcmp(p1.status, "released") != 0)
            continue;
        int illnessDays = patientGetIllnessDays(p1);

        if (illnessDays > 0)
        {
            bool inserted = false;
            for (int j = 0; j < count; ++j)
            {
                int illnessDays2 = patientGetIllnessDays(top5[j]);

                if ((illnessDays > illnessDays2) || (illnessDays == illnessDays2 &&
patientGetAge(p1) > patientGetAge(top5[j])))
                {
                    for (int k = 4; k > j; --k)
                    {
                        top5[k] = top5[k - 1];
                    }
                    top5[j] = p1;
                    inserted = true;
                    break;
                }
            }
            if (!inserted && count < 5)
            {
                top5[count] = p1;
                inserted = true;
            }
            if (inserted && count < 5)
                ++count;
        }
```

```
    }

    printf("TOP 5: \n\n");
    for (int i = 0; i < count; ++i)
    {
        printPatientLong(top5[i]);
        printf("\n");
    }
}
```

OLDEST

Este comando mostra os pacientes mais idosos por sexo.

Complexidade O(n^2) devido ao uso de um algoritmo selection sort, através da implementação do método sortPatientsByOldest, de modo a percorrer as listas filtradas por sexo, obtidas através do método getPatientsBySex, e colocar os pacientes mais idosos no início da respetiva lista, de onde são depois extraídos e apresentados ao utilizador no último ciclo da função.

```
void showOldest(PtList listPatients)
{
    PtList femalePatients = getPatientsBySex(listPatients, "female");
    PtList malePatients = getPatientsBySex(listPatients, "male");

    sortPatientsByOldest(femalePatients);
    sortPatientsByOldest(malePatients);

    int fSize, mSize;
    listSize(femalePatients, &fSize);
    listSize(malePatients, &mSize);

    int maxBirthYear = -1;

    printf("FEMALES:\n");
    for (int i = 0; i < fSize; ++i)
    {

        Patient p;
        listGet(femalePatients, i, &p);
        if (p.birthYear == -1)
            break;

        if (maxBirthYear == -1)
            maxBirthYear = p.birthYear;

        if (p.birthYear != maxBirthYear)
```

```
            break;

        printf("%d - ", i + 1);
        printPatientShort(p);
    }

    maxBirthYear = -1;

    printf("MALES:\n");
    for (int i = 0; i < mSize; ++i)
    {
        Patient p;
        listGet(malePatients, i, &p);
        if (p.birthYear == -1)
            break;

        if (maxBirthYear == -1)
            maxBirthYear = p.birthYear;

        if (p.birthYear != maxBirthYear)
            break;

        printf("%d - ", i + 1);
        printPatientShort(p);
    }

    listDestroy(&femalePatients);
    listDestroy(&malePatients);
}
```

GROWTH

Este comando implementa a função showGrowth que começa por validar que o input de o utilizador tem um formato de uma data válida, e utiliza os métodos auxiliares totalDeceasedAtDate e totalIsolatedAtDate para apresentar o número de pacientes mortos e isolados no dia pedido e no anterior, de modo a calcular o respetivo grau de variação.

Complexidade O(n) devido à utilização dos métodos totalDeceasedAtDate e totalIsolatedAtDate.

```
void showGrowth(PtList listPatients)
{
    char input[20];
    printf("Insira data> ");
```

```
    fgets(input, sizeof(input), stdin);
    input[strlen(input) - 1] = '\0';

    Date d = stringToDate(input);

    if (d.day < 1 || d.day > 31 || d.month < 1 || d.month > 12 || d.year < 1)
    {
        printf("Data introduzida é inválida.\n");
        return;
    }

    Date prev = datePrevious(d);

    int deceasedAtDate = totalDeceasedAtDate(listPatients, d);
    int isolatedAtDate = totalIsolatedAtDate(listPatients, d);

    if (deceasedAtDate == 0 && isolatedAtDate == 0)
    {
        printf("There is no record for day ");
        datePrint(d);
        printf("\n");
        return;
    }

    int deceasedAtPrev = totalDeceasedAtDate(listPatients, prev);
    int isolatedAtPrev = totalIsolatedAtDate(listPatients, prev);

    if (deceasedAtPrev == 0 && isolatedAtPrev == 0)
    {
        printf("There is no record for day ");
        datePrint(prev);
        printf("\n");
        return;
    }

    printf("\nDate: ");
    datePrint(prev);
    printf("\nNumber of dead: %d\n", deceasedAtPrev);
    printf("Number of isolated: %d\n", isolatedAtPrev);

    printf("\nDate: ");
    datePrint(d);
    printf("\nNumber of dead: %d\n", deceasedAtDate);
    printf("Number of isolated: %d\n", isolatedAtDate);

    printf("\nRate of new dead: %f\n",
                (float)(deceasedAtDate - deceasedAtPrev) / (float)(deceasedAtPrev))
;
```

```
    printf("Rate of new infected: %f\n",
                (float)(isolatedAtDate - isolatedAtPrev) / (float)(isolatedAtPrev))
;
}



int totalIsolatedAtDate(PtList listPatients, Date d)
{
    int size;
    listSize(listPatients, &size);

    int count = 0;

    for (int i = 0; i < size; ++i)
    {
        Patient p;
        listGet(listPatients, i, &p);
        if (equalsStringIgnoreCase(p.status, "isolated"))

        {
            if (dateDaysBetween(p.confirmedDate, d) == 0)
            {
                ++count;
            }
        }
    }

    return count;
}

int totalDeceasedAtDate(PtList listPatients, Date d)
{
    int size;
    listSize(listPatients, &size);

    int count = 0;

    for (int i = 0; i < size; ++i)
    {
        Patient p;
        listGet(listPatients, i, &p);
        if (equalsStringIgnoreCase(p.status, "deceased"))

        {
            if (dateDaysBetween(p.deceasedDate, d) == 0)
            {
                ++count;
            }
```

```
        }
    }

    return count;
}
```

MATRIX

O método showMatrix é utilizado de forma a apresentar uma tabela que correlaciona o número de pacientes com o respetivo estatuto e idade. Para isso é criado um array bidimensional de números inteiros que contém todos os totais em memória estática ao decorrer da função.

Complexidade O(n) visto que a lista de pacientes é percorrida uma vez.

```
void showMatrix(PtList listPatients)
{
    int totals[6][3] = {
            {0, 0, 0},
            {0, 0, 0},
            {0, 0, 0},
            {0, 0, 0},
            {0, 0, 0},
            {0, 0, 0},
    };

    int size;
    listSize(listPatients, &size);

    for (int i = 0; i < size; ++i)
    {
        Patient p;
        listGet(listPatients, i, &p);

        int coll, line;

        if (equalsStringIgnoreCase(p.status, "isolated"))
        {
            coll = 0;
        }
        else if (equalsStringIgnoreCase(p.status, "deceased"))
        {
            coll = 1;
        }
        else if (equalsStringIgnoreCase(p.status, "released"))
        {
            coll = 2;
```

```
        }
        else
        {
            continue;
        }

        int age = patientGetAge(p);

        if (age >= 0 && age <= 15)
        {
            line = 0;
        }
        else if (age >= 16 && age <= 30)
        {
            line = 1;
        }
        else if (age >= 31 && age <= 45)
        {
            line = 2;
        }
        else if (age >= 46 && age <= 60)
        {
            line = 3;
        }
        else if (age >= 61 && age <= 75)
        {
            line = 4;
        }
        else if (age >= 76)
        {
            line = 5;
        }
        else
        {
            continue;
        }

        ++totals[line][coll];
    }

    printf("          | isolated | deceased | released |\n");
    printf(" [ 0-
15]  | %8d | %8d | %8d |\n", totals[0][0], totals[0][1], totals[0][2]);
    printf(" [16-
30]  | %8d | %8d | %8d |\n", totals[1][0], totals[1][1], totals[1][2]);
    printf(" [31-
45]  | %8d | %8d | %8d |\n", totals[2][0], totals[2][1], totals[2][2]);
```

```
    printf(" [46-
60]  | %8d | %8d | %8d |\n", totals[3][0], totals[3][1], totals[3][2]);
    printf(" [61-
75]  | %8d | %8d | %8d |\n", totals[4][0], totals[4][1], totals[4][2]);
    printf(" [76...[  | %8d | %8d | %8d |\n", totals[5][0], totals[5][1], totals[5][
2]);
}
```

REGIONS

Este comando mostra informação sobre as regiões e o respetivo número de infetados. Em primeiro lugar são organizadas as chaves por ordem alfabética através de uma algoritmo selection sort e de seguida é feita a contagem do número de pacientes isolados em cada região.

Complexidade O(n^2) devido ao uso de um algoritmo selection sort sobre o mapa de regiões.

```
void showRegions(PtList listPatients, PtMap mapRegions)
{
    int size;
    mapSize(mapRegions, &size);

    MapKey *keys = mapKeys(mapRegions);

    for (int i = 0; i < size - 1; ++i)
    {
        MapKey min = keys[i];
        int indexMin = i;
        for (int j = i + 1; j < size; ++j)
        {
            if (strcmp(min.name, keys[j].name) > 0)
            {
                min = keys[j];
                indexMin = j;
            }
        }
        MapKey swap = keys[i];
        keys[i] = min;
        keys[indexMin] = swap;
    }

    int *totalsIsolated = (int *)calloc(size, sizeof(int));

    int sizeP;
    listSize(listPatients, &sizeP);
```

```
    for (int i = 0; i < sizeP; ++i) //patients
    {
        Patient p;
        listGet(listPatients, i, &p);
        for (int j = 0; j < size; ++j) //regions
        {
            if (strcmp(p.region, keys[j].name) == 0)
            {
                if (strcmp(p.status, "isolated") == 0)
                    totalsIsolated[j]++;
                break;
            }
        }
    }

    for (int i = 0; i < size; ++i)
    {
        if (totalsIsolated[i] > 0)
        {
            Region r;
            mapGet(mapRegions, keys[i], &r);
            regionPrint(r);
            printf("Number of patients isolated: %d\n\n", totalsIsolated[i]);
        }
    }

    free(totalsIsolated);
    free(keys);
}
```

REPORT

Cria um ficheiro de texto que apresenta informações estatísticas sobre a mortalidade, incidência e letalidade da doença no seu total e por região.

São usados dois arrays de ints dinâmicos para conter a contagem de casos e mortes por região (mesmo index que o array de chaves do mapa de regiões). Ao percorrer a lista de pacientes, a região destes é verificada sobre o mapa de regiões e, caso coincida, o estado do paciente é contado nos arrays auxiliares previamente mencionados.

De seguida é percorrido o mapa de regiões para obter um total de habitantes.

Por fim são guardados em ficheiro os valores calculados relativamente ao total das regiões e individualmente de todas as regiões que contenham dados suficientes.

Complexidade O(n^2) devido à necessidade de encadear ciclos para verificar se a região associada ao paciente coincide com uma região existente no mapa de regiões. No entanto, é de

apontar que este ciclo é interrompido no momento em que tal região é encontrada e que a dimensão da lista de pacientes e de regiões não é o mesmo.

```
void report(PtList listPatients, PtMap mapRegions)
{
    FILE *f;

    f = fopen("report.txt", "w+");

    int sizeP;
    listSize(listPatients, &sizeP);

    int sizeR;
    mapSize(mapRegions, &sizeR);

    MapKey *keys = mapKeys(mapRegions);

    int *casesCount = (int *)calloc(sizeR, sizeof(int));
    int *deathsCount = (int *)calloc(sizeR, sizeof(int));

    int totalDeaths = 0;

    for (int i = 0; i < sizeP; ++i)
    {
        Patient p;
        listGet(listPatients, i, &p);

        for (int j = 0; j < sizeR; ++j)
        {
            if (equalsStringIgnoreCase(p.region, keys[j].name))
            {
                ++casesCount[j];
                if (equalsStringIgnoreCase(p.status, "deceased"))
                {
                    ++deathsCount[j];
                    ++totalDeaths;
                }
                break;
            }
        }
    }

    int totalPopulation = 0;
    for (int i = 0; i < sizeR; ++i)
    {
        Region r;
        mapGet(mapRegions, keys[i], &r);
        totalPopulation += r.population;
```

```
    }

    fprintf(f, "TOTAL:\n\tMortality: %f%%, Incident Rate: %f%%, Lethality: %f%%\n\n"
,
                    (float)totalDeaths / totalPopulation * 10000,
                    (float)sizeP / totalPopulation * 100,
                    (float)totalDeaths / sizeP * 100);

    for (int i = 0; i < sizeR; ++i)
    {
        Region r;
        mapGet(mapRegions, keys[i], &r);
        if (casesCount[i] == 0 || r.population == 0)
        {
            fprintf(f, "%s:\n\t(no population data)\n\n", keys[i].name);
            continue;
        }
        fprintf(f, "%s:\n\tMortality: %f%%, Incident Rate: %f%%, Lethality: %f%%\n\n
", keys[i].name,
                        (float)deathsCount[i] / r.population * 10000,
                        (float)casesCount[i] / r.population * 100,
                        (float)deathsCount[i] / casesCount[i] * 100);
    }

    free(keys);
    free(casesCount);
    free(deathsCount);
    if (fclose(f) != 0)
    {
        printf("Report not created.\n");
        return;
    };
    printf("Report created successfully.\n");
}
```

Algoritmos em Pseudocódigo

Algoritmo showAverage

```
Algorithm showAverage
        input: listPatients - pointer to list of patients
        output:
BEGIN
        total <- size of listPatients

        IF total > 0 THEN

                deceasedPatients <- get patients with status = "deceased"
                releasedPatients <- get patients with status = "released"
                isolatedPatients <- get patients with status = "isolated"

                avgd <- get average age from deceasedPatients list
                avgr <- get average age from releasedPatients list
                avgi <- get average age from isolatedPatients list

                deceasedPatients <- NULL
                releasedPatients <- NULL
                isolatedPatients <- NULL

                PRINT "Average Age for deceased patients: $avgd"
                PRINT "Average Age for released patients: $avgr"
                PRINT "Average Age for isolated patients: $avgi"

        ELSE
                PRINT "No patient data available"
```

```
        END IF
END
```

Algoritmo showPatient

```
Algorithm showPatient
        input: listPatients - pointer to list of patients
        output:

BEGIN

        PRINT "Insira ID de um paciente> "
        READ ipt

        patient <- get patient with ID = ipt from listPatients

        IF patient = NULL THEN
                PRINT "Paciente não encontrado."
        ELSE
                PRINT patient details
        END IF
END
```

Algoritmo showRegions

```
Algorithm showRegions
        input: mapRegions - pointer to map of regions
             listPatients - pointer to list of patients
        output:

BEGIN
        size <- size of map
        keys <- keys of mapRegions

        FOR i <- 0 TO i < size-1 DO
                min <- keys[i]
                indexMin <- i

                FOR j <- i+1 TO j < size DO
                        IF keys[j] has alphabetical precedence over min THEN
                                min <- keys[j]
                                indexMin <- j
                        END IF
                END FOR

                SWAP keys[i], keys[indexMin]

        END FOR

        DECLARE totalsIsolated

        sizeP <- size of patient list
```

```
        FOR i <- 0 TO i < sizeP DO
                p <- listPatients[i]

                FOR j <- 0 TO j < size DO
                        IF p.region = keys[j] THEN
                                IF p.status = "isolated" THEN
                                        totalsIsolated[j]++
                                END IF
                        END IF
                END FOR
        END FOR

        FOR i <- 0 TP i < size DO
                IF totalsIsolated[i] > 0 THEN
                        r <- get value from mapRegions at keys[i]
                        PRINT r
                        PRINT "Number of patients isolated: $totalsIsolated[i]"
                END IF
        END FOR
END
```

Conclusões

Apesar de algumas informações pedidas parecerem relativamente simples os algoritmos a desenvolver podem ser mais complexos do que antecipado.

Na maior parte dos comandos não foi possível implementar complexidades constantes já que estes implicam necessariamente a verificação e, ocasionalmente, comparação de todos os elementos dentro dos respetivos ADT's de modo a apresentar a informação pedida.

Quanto a dificuldades, tivemos algumas com a tradução de alguns dados nos ficheiros csv fornecidos para um formato correto e também com a formatação de alguns dos dados gerados dentro de funções, nomeadamente na tradução de valores float para um formato mais apresentável na consola.

Por fim, pensamos que os objetivos deste projeto foram todos cumpridos.