IPS Instituto Politécnico de Setúbal
Escola Superior de Tecnologia de Setúbal

ATAD 2018/19

Algoritmos e Tipos Abstratos de Dados

Relatório Técnico

Turma: INF-04
Docentes:
TP: Bruno Silva
PL: Hugo Santos
Alunos:
- 180221070 – Rafael Trindade
- 180221109 – Joanã Costa

Índice

1. TAD Utilizados 3
1.1. Estrutura `ClinicalData` 3
1.1.1. Estrutura 3
1.2. Estrutura `Patient` 3
1.1.1. Funções 3
1.1.2. Funções 3
1.1. Estrutura ClinicalDataStats 4
1.1.1. Estrutura 4
1.1.1. Funções 4
1.2. List - ArrayList 4
1.3. Map – ArrayList 4
1.4. Queue – ArrayList 5
2. Complexidades Algorítmicas 5
2.1. Comandos/Funções 5
LOAD 5
Sort 6
AVG 6
NORM 7
QUEUE 8
CHECKDISTRICT 9
3. Limitações 10
4. Conclusão 10

1. TAD Utilizados

Os TAD utilizados para o este projeto foi o ***List***, ***Map*** e ***Queue*** onde cada um teve uma implementação específica na estrutura do programa.

1.1. Estrutura `ClinicalData`

Estrutura que guarda os dados clínicos de cada consulta.

1.1.1. Estrutura

```
typedef struct clinicalData {
    float age;
    float bmi;
    float glucose;
    float insulin;
    float mcp1;
    int disease_type;
    int clinicalDataCount;

} ClinicalData;

/* Estrutura dos dados Clínicos*/
```

1.2. Estrutura `Patient`

Guarda os dados de um determinado paciente, incluindo os seus dados clínicos.

1.1.1. Funções

```
typedef struct patient {
    int id;
    Date birthdate;
    char gender;
    String hospital;
    String district;
    ClinicalData clinicalData;
} Patient;


/* Estrutura dos dados do Paciente*/
```

1.1.2. Funções

```
Patient patientCreate(int id, Date birthdate, char gender, char *hospital, char *district);

void patientPrint(Patient patient);

void patientNormPrint(Patient patient);
```

1.1. Estrutura ClinicalDataStats

Estrutura que guarda os valores médio de cada dado clínico.

1.1.1. Estrutura

```
typedef struct clinicalDataStats{
    float age;
    float bmi;
    float glucose;
    float insulin;
    float mcp1;
    int disease_type;
    int clinicalDataCount;
    /* Usado apenas na opcao NEURALNET */
    float c1;
    float c2;
    float c3;
    float c4;

} ClinicalDataStats;
```

1.1.1. Funções

```
ClinicalDataStats clinicalDataStatsCreate();

void clinicalDataStatsPrint(ClinicalDataStats *clinicalDataStats);
```

1.2. List - ArrayList

Usado maioritariamente em toda a estrutura do programa onde a estrutura de acesso é baseada por ***ranks*** permitindo assim o acesso a aleatório a qualquer instante a um elemento presente na lista. Esta estrutura foi usada para fazer carregar a lista de pacientes e dados clínicos.

1.3. Map – ArrayList

Um mapa representa um contentor de elementos que implementa uma memória associativa (dicionário) onde são armazenados tuplos, **{chave: valor}.** A chave está apenas associada a um valor para além de não existir chaves duplicadas, há valores duplicados, nuca chaves duplicadas. Os acessos são feitos indicando a **chave**.

Esta estrutura foi escolhida para fazer a média de dados clínicos de cada paciente. A estrutura ficou com a seguinte configuração:

```
/* definicao do tipo do chave*/

typedef String MapKey;

/* definicao do tipo do valor*/

typedef ClinicalDataStats MapValue;
```

1.4. Queue – ArrayList

A fila é um contentor de elementos que são inseridos e removidos de acordo com o princípio FIFO.

- Os elementos podem ser inseridos na fila a qualquer altura, mas apenas o que se encontra na fila há mais tempo pode ser removido.
- Os elementos são inseridos no final da fila e removidos do início da fila.

Este TAD foi utilizado na implementação para ver todos pacientes que se enquadram num determinado tipo de critério, os dados clínicos médios num intervalo mínimo e máximo e de uma idade mínima e máxima, onde a cada comando **NEXT** dentro comando **QUEUE** mostra os dados do paciente um após o outro sucessivamente.

2. Complexidades Algorítmicas

2.1. Comandos/Funções

LOAD

Esta função lê dois ficheiros. O primeiro são os dados dos pacientes onde é inicializado os dados clínicos do paciente a zero, a seguir lê-se os dados do ficheiro que contém os dados clínicos associando assim a cada paciente onde é feito uma pesquisa na lista de pacientes para atualizar os dados clínicos de cada um paciente. A complexidade é **O (n^2)**.

```
rank = findPatientRankById(*patients, atoi(tokens[0]));

if (rank != -1) {
    //Caso o rank seja valido atualiza os dados clinicos

    listGet(*patients, rank, &patientElem);

    int day, month, year;
    sscanf_s(tokens[1], "%d/%d/%d", &day, &month, &year);
    Date data2 = dateCreate(day, month, year);

    float age = getAge(patientElem.birthdate, data2);
    patientElem.clinicalData.age = calculateAVG(patientElem.clinicalData.age, age, patientElem.clinicalData.clinicalDataCount);

    patientElem.clinicalData.bmi = calculateAVG(patientElem.clinicalData.bmi, atof(tokens[2]), patientElem.clinicalData.clinicalDataCount);
    patientElem.clinicalData.glucose = calculateAVG(patientElem.clinicalData.glucose, atof(tokens[3]), patientElem.clinicalData.clinicalDataCount);
    patientElem.clinicalData.insulin = calculateAVG(patientElem.clinicalData.insulin, atof(tokens[4]), patientElem.clinicalData.clinicalDataCount);
    patientElem.clinicalData.mcp1 = calculateAVG(patientElem.clinicalData.mcp1, atof(tokens[5]), patientElem.clinicalData.clinicalDataCount);
    patientElem.clinicalData.clinicalDataCount++;
    listSet(*patients, rank, patientElem, &oldClinicalData);

}
```

Sort

Mostra os pacientes importados de forma ordenada crescentemente. O critério de ordenação deve ser solicitado ao utilizador e pode ser um dos seguintes:

- - Data de nascimento
- - Hospital (desempate pela data de nascimento)
- - Distrito (desempate pelo hospital);

A complexidade deste comando é **O(n^2)**.

```
void sortByBirthdate(PtList patients) {
    int size;
    listSize(patients, &size);
    ListElem patient1, patient2;

    for (int i = 0; i < size; i++) {
        for (int j = 0; j < size - i - 1; j++) {

            listGet(patients, j, &patient1);
            listGet(patients, (j + 1), &patient2);

            int result = compareBirthdate(patient1, patient2);
            if (result == 1) {
                swapPatients(patients, j, (j + 1), patient1, patient2);
            }
        }
    }

}
```

AVG

Mostra a média dos dados clínicos de cada paciente (age, bmi, glucose, insulina e mcp1) em cada distrito, e ordenados por distrito. A complexidade deste algoritmo é **O(n^3)**.

```
PtList auxiliar = listCreate(size);
auxiliar = copyPtList(patients);

sortByDistrict(auxiliar);
PtMap map = mapCreate(490);
averageClinicalData(auxiliar, &map);
```

NORM

Mostra, para cada paciente, os seus dados clínicos normalizados entre -k e k, segundo a normalização *min-max*.

```
int size;
ListElem elem, oldElem;

listSize(patients, &size);

ClinicalDataStats min = clinicalDataStatsCreate(), max = clinicalDataStatsCreate(), averageValue = clinicalDataStatsCreate();
findMinAndMaxAndAVG(patients, &min, &max, &averageValue);
```

```
for (int i = 0; i < size; i++) {
    listGet(patients, i, &elem);
    elem.clinicalData.age = calculateNorm(elem.clinicalData.age, min.avgAge, max.avgAge, k);
    elem.clinicalData.bmi = calculateNorm(elem.clinicalData.bmi, min.avgBmi, max.avgBmi, k);
    elem.clinicalData.glucose = calculateNorm(elem.clinicalData.glucose, min.avgGlucose, max.avgGlucose, k);
    elem.clinicalData.insulin = calculateNorm(elem.clinicalData.insulin, min.avgInsulin, max.avgInsulin, k);
    elem.clinicalData.mcp1 = calculateNorm(elem.clinicalData.mcp1, min.avgMcp1, max.avgMcp1, k);
    listSet(patients, i, elem, &oldElem);
}
return patients;
```

Figura 1 - Função normalizeClinicalData

```
for (int i = 0; i < size; i++) {
    listGet(list, i, &patient);
    if (minValue->avgAge > patient.clinicalData.age) minValue->avgAge = patient.clinicalData.age;
    if (maxValue->avgAge < patient.clinicalData.age) maxValue->avgAge = patient.clinicalData.age;
```

Figura 2 - Função findMinMaxAndAVG

No **NORM**, depois de ter recebido a lista de pacientes e o valor inserido de **k** pelo utilizador, executamos a função ***normalizeClincalData()*** que vai percorrer toda a lista normalizando os dados clínicos de cada paciente. A função ***findMinAndMaxAVG()*** devolve os valores médios e mínimos de todos os dados clínicos.

A complexidade do comando **NORM** é quadrática **O(n^2)**.

QUEUE

Este comando copiar para fila todos os pacientes que se enquadrem nos seguintes critérios:

- - A sua idade seja inferior ao valor médio do intervalo [min(age), max(age)] das idades de todos os pacientes.

ou

- - A sua idade seja superior ao valor médio do intervalo [min(age), max(age)] das idades de todos os pacientes.

- O valor dos atributos bmi, glucose, insulina e mcp1 sejam inferiores ao valor médio do intervalo entre o [min(atr),max(atr)] para cada um destes 4 atributros. A complexidade deste comando é **O(n^2)**.

```
    findMinAndMaxAndAVG(patients, &min, &max, &averageValue);

    addToQueue(patients, &queuePatients, &averageValue);

    printf("Available commands: NEXT & STOP\n");
    do {

        printf("COMMAND> ");
        fgets(command, sizeof(command), stdin);
        command[strlen(command) - 1] = '\0';

        if (strcmp(command, "NEXT") == 0) {
            if (queueIsEmpty(queuePatients) == 1) {
                printf("Queue is empty");
                system("pause");
                quit = 1;
            }
            ListElem queueElem;
            queuePeek(queuePatients, &queueElem);
            printf("          Indice BirthDate Sex Hospital
            queueElemPrint(queueElem);
            queueDequeue(queuePatients, &queueElem);
        }
        else if (strcmp(command, "STOP") == 0) {
            quit = 1;
        }
```

```
void addToQueue(PtList list, PtQueue *queue, PtClinicalDataStats averageValue) {

    unsigned int size;
    listSize(list, &size);
    ListElem patientList;
    int count = 0;

    for (int i = 0; i < size; i++) {
        listGet(list, i, &patientList);
        if (patientList.clinicalData.age < averageValue->avgAge) {
            queueEnqueue(*queue, patientList);
            count++;
        }
        else if (patientList.clinicalData.age >= averageValue->avgAge) {

            if (patientList.clinicalData.bmi < averageValue->avgBmi &&
                patientList.clinicalData.glucose < averageValue->avgGlucose  &&
                patientList.clinicalData.insulin < averageValue->avgInsulin  &&
                patientList.clinicalData.mcp1 < averageValue->avgMcp1) {

                queueEnqueue(*queue, patientList);
                count++;
            }
        }

    }
    queuePrint(*queue);
    printf("\n%d elements were copied to queue!\n\n", count);
}
```

CHECKDISTRICT

Este comando segmenta os pacientes com as suas médias de dados clínicos de acordo com o seu distrito de residência. A complexidade deste comando é **O(n^2)**.

```
listSize(patients, &size);

PtMap map = mapCreate(size);
averageClinicalData(patients, &map);
MapValue value;

do {
    printf("\nDISTRICT> ");
    fgets(command, sizeof(command), stdin);
    command[strlen(command) - 1] = '\0';

    if (mapContains(map, command) == 1) {
        mapGet(map, command, &value);
        printf("District              Age    BMI       Glucose Insulina  MCP1\n");
        mapKeyPrint(command);
        mapValuePrint(value);
    }
    else if (strcmp(command, " ") == 0 || strcmp(command, "") == 0) {
        quit = 1;
        mapDestroy(&map);
        system("pause");
        clrscr();
        return;
    }
    else {
        printf("\033[0;31m District not found.\n");
        printf("\033[0m");
    }
```

3. Limitações

Não conseguimos implementar o comando **NEURALNET**.

4. Conclusão

Após a conclusão da implementação do projeto e dado que este cumpre quase tudo o que é pedido. O que resulta da tentativa de simplificar ao máximo os processos através da implementação de Tipos Abstratos de Dados é bastante satisfatório e permite uma melhor interpretação e organização do código e, por isso, maior eficiência.

O objetivo principal do projeto foi cumprido sendo que foram usados os TADs ***Queue***, ***List*** *e* ***Map*** com a variante do tipo ***ArraList***.

Dito isto prevemos que o projeto foi uma enorme ajuda na captação e utilização dos conceitos utilizadas nas aulas teóricas e práticas que este foi concluído com sucesso.