ARTHUR HERBERT SILVA MELO

TRABALHO PRÁTICO 1
ALGORITMOS I

Descrição

O Trabalho Prático 1, da matéria Algoritmos I consistiu em resolver um problema fictício de organização de candidatos e vagas em universidades. O desafio foi estabilizar todos os pares candidato-universidade, para que não houvesse nenhuma instabilidade e que todos os estudantes ficassem o mais feliz possível.

Estruturas de Dados e Algoritmos

Tipos (structs)

Para ajudar na manipulação e abstração dos dados, foram criados alguns tipos, são eles:

```c
typedef struct {
  int id;
  int numberOfPlaces;
  int minimumScore;
} University;
```

```c
typedef struct {
  int id;
  int score;
  int numberOfApplications;
  int* priorityList;
} Applicant;
```

```c
typedef struct {
  University university;
  Applicant applicant;
} Match;
```

```c
typedef int bool;
#define true 1
#define false 0
```

O tipo *University* é a abstração das universidades, contendo id, número de lugares e pontuação mínima. O *Applicant* é a abstração do candidato, e contém seu id, nota, pontuação e lista de prioridade, que é um vetor. O tipo *Match* representa uma combinação,

e contém uma *University* e um *Applicant.* O tipo *bool* e seus *#defines* ajudou na representação de tipos *booleanos(true e false).*

Estruturas

Foram vetores dinamicamente alocados para representar a lista de universidades (*University**), a lista de candidatos(*Applicant**), a lista de candidatos não alocados(*Applicant**) e a lista de vagas já preenchidas(*Match**).

Estrutura do projeto

Para, novamente, ajudar na abstração o projeto foi divido em uma pasta *models,* que contém os tipos vistos anteriormente; Uma pasta *utility* que contém as funções de leitura de arquivos e montagem dos modelos, são elas:

```
1 University* getUniversitiesFromFile(char* fileName);
2 Applicant* getApplicantsFromFile(char* fileName);
3 int getQuantity(char* fileName);
```

A primeira, lê e formata os dados das universidades e retorna um ponteiro para um vetor de *University*, esta função tem complexidade O(n), onde n é o número de universidades, pois basta um loop nesta quantidade para preencher todos os dados.

A segunda, faz o mesmo para os candidatos, contudo, esta tem complexidade O(n + na), onde n é o número de candidatos, e na a soma da quantidade de aplicações de todos os candidatos.

A terceira só retorna o primeiro número do arquivo, neste problema, se resume na quantidade de universidades ou candidatos;

E por fim num arquivo *tp1.c*, que contém as funções principais, não convém detalhar todas elas, mas o algoritmo consiste numa adaptação do algoritmo de Gale-Shapley, o

pseudo-código do loop principal é mostrado abaixo:

```
while(Existe um candidato não alocado e que tem alguma faculdade em sua lista de prioridades) {
    retorna universidade da lista de prioridades
    if(nota > nota mínima da universidade) {
        if(universidade possui vaga cuja nota atual é menor que a nota do candidato) {
            coloca o candidato nesta vaga
        } else {
            verifica se é possível inserir na lista de não alocados
        }
    } else {
        verifica se é possível inserir na lista de não alocados
    }
}
```

Podemos analisar cada uma destas ações separadamente:

- **AÇÃO DO WHILE:** Esta consiste basicamente em passar por cada candidato e verificar se ele já está alocado em alguma vaga, então, no pior caso, passaremos por todos os candidatos e por todas as alocações, gerando uma complexidade O(n*na), onde n é o número de candidatos e na o número de vagas atualmente alocadas. Se o candidato não possuir nenhuma universidade em sua lista de prioridades, significa que ele já está no grupo dos não alocados.
- **RETORNAR UNIVERSIDADE:** Esta ação retorna a universidade na primeira posição da lista de prioridades, bem como a remove da lista, movendo a próxima para a primeira posição. Como precisamos percorrer o vetor de universidades procurando pelo id, no pior caso O(número de universidades).
- **SEGUNDO IF:** Aqui, temos que passar por todas as vagas atualmente alocadas, para verificar se existe uma vaga vazia, ou se há vaga atualmente alocada com uma nota menor. No pior caso, passaremos por todas as vagas atualemte alocadas. O(n).
- **INSERÇÃO NA VAGA:** Como foi utilizado um vetor de structs, não foi possível realizar esta ação em O(1) em todas os casos, pois no caso em que a vaga já está preenchida, mas com uma nota menor, é necessário percorrer todas as vagas já locadas, para trocar o candidato. Por isso, no pior caso, essa ação terá complexidade O(número de vagas atualmente alocadas).
- **INSERIR NA LISTA DE NÃO ALOCADOS:** Basta alocar memória e inserir o candidato no grupo dos não alocados se este não tiver mais universidades para tentar, por isso O(1).