TP O

Stable Marriage Problem

Algoritmos e Estruturas de dados III

Thalia de Almeida Magalhães Campolina

STABLE MARRIAGE PROBLEM

1. Introdução

O problema apresentado a seguir é conhecido como Problema do Casamento Estável, para o qual será apresentado o algoritmo para sua resolução.

O SMP (Stable Marriage Problem) consiste em encontrar a melhor combinação de casais, ou seja, de casamentos, para que esses sejam todos estáveis, dado um conjunto de homens e mulheres, e suas respectivas escolhas de parceiros. Essas escolhas de parceiros são apresentadas, respectivamente, pela primeira opção de casamento, seguida das outras opções em ordem de pteferência. O número de homens e mulheres precisa ser equivalente, para que o algoritmo funcione.

Um exemplo de lista de preferências pode ser dado como:
M1: H3 H1 H2
M2: H3 H2 H1
M3: H1 H3 H2
H1: M1 M2 M3
H2: M2 M1 M3
H3: M3 M2 M1
Sendo que M é representação de Mulher, e H representação de Homem.

O casamento é instável caso um dos conjuges A do casal prefera outra pessoa C para estar casada, e essa pessoa C também prefera a pessoa A no lugar do seu atual cônjuge. Se isso nunca ocorrer entre os casais listados, os casamentos serão Estáveis.

A solução foi dada por David Gale e Lloyd Shapley, conhecida como solução de Gale-Shapley, que provaram que sempre é possível resolver esse problema para que todos os casamentos sejam estáveis. O algoritmo consiste em iniciar a busca pelo parceiro ideal sempre pelo homem, e isso pode ser feito com um homem por vez, ou todos os homens juntos, realizando essa busca. O homem pede em casamento a primeira mulher da sua lista de preferências. Se essa estiver solteira, eles se casam, se essa estiver casada ela pode ficar com o parceiro que esteja na posição mais privilegiada de sua lista. Então o homem que foi rejeitado continua a busca a partir a sua segunda mulher predileta.

2. Modelagem e Solução

A estrutura utilizada para a resolução foi Lista Encadeada Circular. Dado um arquivo de entrada input.txt, foram geradas duas listas,uma para os homens e uma para as mulheres, chamadas de womansCrush e mensCruch respectivamente. Essas listas possuem um índice, representadas através de um vetor alocado dinamicamente, e cada elemento dessa lista representa outra lista com as preferências da pessoa.

2.1. Lendo o arquivo

A forma com que o arquivo é lido e salvo na lista é representado, em partes, no trecho de código a seguir, em que um arquivo de entrada é aberto e as informações tiradas dele.

Após a entrada ser lida, as listas são modificadas em funções que a analizam, para que seja feita a saída.
Cada homem, ou cada mulher, possui em sua lista a informação chamada status, que possui valor Zero (0) caso a pessoa esteja solteira, ou então o número relativo à identidade da pessoa que está casada. O struct lista será mostrado a seguir.

Após essa informação ser salva no arquivo de saída, as listas que foram alocadas dinamicamente são deletadas:

```
fclose(entrada);
   for(i=1; i <= number; i++){
      clear(&mensCrushes[i]);
      clear(&womansCrushes[i]);
   }
```

Toda a memória alocada é depois desalocada ao final do procedimento.

2.2. Criando o algoritmo para solução, baseado no algoritmo de Gale-Shapley

Para facilitar o entendimento do código, foram criadas várias funções auxiliares na solução do problema, tais como a função isMarried,que verifica de uma pessoa já é casada,a função Divórcio que separa as pessoas, Prefer que mostra se uma mulher prefere seu atual companheiro ou o seu pretendente (ambos os homens não passados como parâmetro), e a função AllMarried que verifica se todas as pessoas já estão casadas, sendo essa última a condição para o fim da execução do algoritmo.

2.3. Criando o elemento lista

A lista é composta por nós, havendo portando um Struct que defina esse novo tipo não existente ainda na linguagem C, sendo esse nó formado pelo seu valor (key), ponteiro para o nó posterior e ponteiro para o nó anterior:

```
typedef struct node {
   int key;
   struct node* next;
   struct node* prev;
} Node;
```

Também foi criado o Struct lista, que contém o nó sentinela para marcar o final da lista, chamado de end_, um tamanho, e um status que será útil para definir os casamentos das pessoas envolvidas nesse problema:

```
typedef struct list {
   Node* end_;
   int size_;
   int status_;
} List;
```

Após a criação dessas estruturas de dados, foram criadas várias funções auxiliarem que realizam operações importantes para a manipulação de uma lista, tais como NewNode (para criar um novo nó), NewList (cria nova lista), Back (Retorna ponteiro para ultimo elemento da lista), Begin (Retorna ponteiro para primeiro elemento da lista), ListEnd(Retorna ultimo elemento da lista),ListFirst(Retorna primeiro elemento da lista), Key(Retorna chave do nó), InsertBack (insere elemento ao final da lista, Insert(Insere elemento antes do elemento indicado) e Erase e Clear (apaga elemento ou a próprια lista)

3. Complexidade de Tempo e Espaço do algoritmo

Para criar a função principal, foram utilizadas funções auxiliares, que realizam a manipulação das listas e laços de “for” e de “while”.
Por exemplo, a função que verifica se todas as pessoas de uma determinada lista estão casadas está representada abaixo:

```
int AllMarried(List* list){
    number = list[0].size_;
    int i;
    for (i = 0; i < number; i++){

        if(isMarried(i, list)){
        return 0;
        }
    }
    return 1;
 }
```

Na função que realmente faz o algoritmo do SMP são realizados dois laços:

```
while (!AllMarried (mensCrushes)){
    for (i = 1;  i<= number; i++){
```

Esses laços são importantes para garantir que nenhum homem fique solteiro ao final, e percorre um por um para realizar a busca pelo casamento estável.

A complexidade do algoritmo, portanto, é $O(n^2)$.