TP0

O Hit do Verão
Algoritmos e Estruturas de Dados III

Giovanni Ferreira Martinelli

1 de abril, de 2018

1 Introdução

O objetivo deste trabalho é a resolução do problema de contagem das pessoas que gostaram de uma música em uma Rede social. As informações disponíveis são as identidade, idades, os familiares das pessoas e quem foi a primeira pessoa a ouvir e compartilhar. Sendo necessário implementar a estrutura de grafos para a solução do problema. Sendo a definição de grafos G = (V, E), os componentes básicos tomam forma como as pessoas sendo os nós (V) e os familiares sendo as arestas (E)

2 Implementação

2.1 Abordagem

A abordagem do problema visava em:

1. Criar um grafo $G = (V, F)$ e preenche-lo com os valores do arquivo, sendo assim G = $(pessoas, familiares)$ da entrada
2. Criar uma lista de estados para representar se uma pessoa já "Viu ou não" a publicação e se "Gostou ou não"
3. Iterar sobre as listas de adjacência do grafo feito, calculando quantos compartilhamentos foram feitos, ao iterar sobre um nó, marca-lo no vetor de estados como "Não viu", "Viu-Gostou" ou "Viu-Não-Gostou" na lista de Estados
4. Calcular quantas pessoas gostaram
5. Mostrar na saída padrão o valor

2.2 Modelagem do Problema

O “O Hit do Verão” foi implementado utilizando a estrutura Grafos, através de lista de adjacências. Essas listas contendo a estrutura Pessoa, para facilitar a coesão e processamento de dados. Também foi implementada a estrutura de dados lista de estados para facilitar a contagem de curtidas.
A estrutura pessoa é utilizada para agrupar as identidades e as idades lidas da entrada, facilitando a coesão e a passagem de parâmetros. Cada estrutura pessoa comporta um Id e uma idade, por isso é utilizada uma lista de estruturas ”pessoa” ao ler os valores da entrada passada.

Implementação:

```
//Struct Pessoa
typedef struct
{
    int ID;
    int IDADE;
}pessoa;
```

A estrutura ”lista de estados” é utilizada para iterar sobre o grafo e serve como índex para os nós que já foram percorridos, sendo informado também se gostaram ou não. Para a implementação da mesma foi usado um vetor de inteiros, cujas posições $[0...V]$ representam o ID da pessoa e o conteúdo de cada casa, contem um dos 3 valores possíveis: 0 para representar se não foi visitado, 2 para representar se foi ”Viu” e ”não gostou” e 3 para representar que ”Viu” o compartilhamento e ”Gostou”.

Estado	Valor
Não viu	0
Viu e não gostou	2
Viu e Gostou	3

Implementação:

```
int* estado = (int*) calloc((size_t)numeroPessoas, sizeof(int));
```

Por último temos a estrutura grafo, seus vértices representam quantas pessoas são no problema, já o outro elemento é um vetor de estruturas lista. As estruturas lista são ponteiros para a estrutura nó, sendo efetivamente um ponteiro para o nó inicial da lista encadeada, que por sua vez é composto de um ponteiro para pessoa e um ponteiro para a próxima estrutura (o seguinte bloco da lista encadeada).

Implementação:

```
//Struct do NO
```

```
typedef struct no {
    pessoa* objPessoa;
    struct no * prox;
}no;

//Struct Lista de NOs
typedef struct lista{
    struct no* cabeca;
}lista;

//Struct Grafo
typedef struct grafo{
    int vertices;
    struct lista* array;
}grafo;
```

Para iterar sobre a lista de adjacência e contar as curtidas utiliza-se do vetor de estados para verificar o que já ocorreu e assim otimizar o percurso pelo grafo, como pode-se ver mais claramente no pseudocódigo abaixo:

Pseudocódigo:

```
    for(i = 0; i < Total de Vertices; i++)
    {
        //No tmp para percorrer array
        no* tmp = grafoPt->array[i].cabeca;

        //Percorre a lista
        while(tmp != NULL)
        {
            //Se o amigo[raiz] nao gostou (nao precisa olhar
            //se os amigos gostaram pois ele nao compartilhou)
            if(estado[i] == 2) break;

            //Se o [No] amigo nao foi visitado
            if(amigo[No]] == 0)
            {
                //Se o [No] amigo tem idade para gostar
                if(amigo[No]->Idade < 35)
                {
                    //Se o amigo[raiz] gostou e compartilhou
                    if(amigo[raiz] == 3)
                    {
                        curtidas++;
                        estado[amigo[No]] = 3; //visitou e gostou
                    }
                    else//A mensagem nunca chegou ate amigo[No]
                    {
                        estado[amigo[No]] = 0; // 0 = nao visitou
```

```
                }
            }
            //Amigo e mais velho, viu a publicacao e nao gostou else
            {
                estado[amigo[No]] = 2;
            }
        }
        //Se o amigo[No] for quem tiver curtido
        else if (amigo[No] == 3)
        {
            //Utilizo a lista de pessoas para checar idade de
                amigo[raiz]
            //Entao se tiver idade o amigo[raiz] curte tambem
            if(amigo[raiz] == 0 && Pessoas[i]->IDADE < 35)
            {
                like++;
                estado[i] = 3;
            }
        }
        //vai para o proximo No na lista encadeda
        tmp = tmp->prox;
    }
}
//mostra o valor no final
printf("%d\n", like);
```

3 Análise Assintótica

3.1 Análise temporal

Sendo o número de vértices V e o número de arestas E, temos que: O grafo utilizado possui uma lista de adjacência que possui V ponteiros para listas encadeadas. Cada lista encadeada possui E' nós, que correspondem ao número de arestas de V[i]. Para contar todas as curtidas é necessário percorrer todo o grafo, verificando se o nó já foi percorrido, gostou ou não gostou. Essas comparações simples são O(1), já que pegam essas informações com o vetor de pessoas. Para percorrer o vetor temos então:

```
for(i = 0; i < Total de vertices; i++) O(V)
{
    Comparacoes O(1)
    While(Vetor != NULL)
    {
        Comparacoes O(1)
        Percorre a lista encadeada
    }
```

```
}
```

$X = \sum_0^{E'} 1 = E'$

$\sum_0^V 1 = O(V)$

Se somarmos todos os E' teremos o E inicial, logo:

$\sum_0^V X = O(|V| + |E|)$

3.2 Análise Espacial

Para calcular o custo espacial do algoritmo vamos ver as alocações feitas:

```
Alocacao 1:
int* estado = (int*) calloc((size_t)numeroPessoas, sizeof(int));

Alocacao 2:
//Aloca Lista de Pessoas e seus arrays
pessoa** Pessoa = criaListaDePessoas(numeroPessoas);

Alocacao 3:

//Aloca grafo
grafo* grafoPt = criaGrafo(numeroPessoas);

no* criaNo(int x, int y)
{
    no* noPt = (no*) malloc(sizeof(no));
}

void addAresta(grafo* grafoPt, int ini, int fim, pessoa** Pessoa)
{
    no* temp = criaNo(fim, Pessoa[fim-1]->IDADE);
}

for(n = 0; n < numeroRelacoes; n++)
{
addAresta(grafoPt, ini, dest, Pessoa); //amizade ini -> fim
addAresta(grafoPt, dest, ini, Pessoa); //amizade fim -> ini
}
```

Sendo V os vértices do grafo e E as arestas temos:

A alocação 1 depende da variável numeroPessoas que é exatamente o número de vértices do grafo, sendo um vetor de inteiros o custo seria de O(V)

A alocação 2 depende também do número de vértices do grafo e aloca a estrutura que sabemos que possui dois inteiros, ou seja, O(1 + 1) = O(1) para cada estrutura, temos que essa estrutura é alocada V vezes, então seu custo é O(V)

A alocação 3 mostra que a função addAresta é executada 2 vezes por iteração em numeroRelacoes iterações, sendo numeroRelacoes o numero de arestas então é executada 2E vezes, essa iteracão ocorre ao popular o grafo, que por sua vez depende da quantidade de numeroPessoas, ou seja, vértices. Então ao popular um grafo com V ponteiros para listas encadeadas com 2E arestas, seu custo total fica como V + 2E

A complexidade total do programa é O(V, V, (V + 2E)) = O(V + 2E) = O(—V— + —E—)

4 Resultados

Nas simulações feitas com os testes dados no Moodle, pode-se testar a eficiência do programa e verificar se a análise assintótica também está de acordo.

Teste	Tempo
1	0.000099
2	0.000138
3	0.000461
4	0.003055
5	0.010630
6	0.162568
7	0.435658
8	0.693981
9	0.963710
10	3.964640

Tempo em milissegundos

Comparando as entradas do teste 9 e do teste 10, Teste 9 (V + E) = 1970000, Teste 10 (V + E) = 7860000 e dividindo 7860000/1970000 = 3,9898, ou seja uma entrada 4 vezes maior. Nota-se que (V + E) cresceu 4 vezes, vemos também que o tempo cresce linearmente 4 vezes, estando de acordo com a análise assintótica temporal

5 Conclusão

O trabalho foi muito educativo nos conceitos de implementação de Grafos e de leitura pelo Stdin, sendo a implementação não trivial e aberta a varias interpretações, despertando a curiosidade e a criatividade para a resolução do problema