Universidade Federal de São Paulo - Campus São José dos Campos
Disciplina: Projeto e Análise de Algoritmos

Problema 4: Festas em comunidades

Rosângela Miyeko Shigenari, 92334

23 de junho de 2018

1 Introdução

Este trabalho visa a resolução de um problema da festa em comunidades com a aplicação dos conceitos e programação em grafos, vistos em aula, em linguagem C/C++, com um tempo limite de 0,5s usando um limite de memória de 50.000 KB, e em seguida, realizar o estudo de sua complexidade.

2 Problema

Sabendo do sucesso das redes sociais, um novo clube da cidade está planejando organizar festas fechadas em que todos os convidados são membros de alguma comunidade de uma rede social, sendo que a cada festa escolhe-se uma comunidade como público alvo. Como regra dessas festas, cada participante deve ser amigo de pelo menos um certo número de participantes. Para fazer com que cada festa seja bem sucedida é preciso convidar o maior número possível de pessoas de cada comunidade. Considerando que a lista de amigos de cada pessoa de uma comunidade pode ser obtida a partir da rede social, a sua tarefa é, dado o conjunto de pessoas de uma certa comunidade e uma lista de relações entre essas pessoas, determinar quantas pessoas devem ser chamadas para que a festa tenha o maior número possível de participantes que satisfaçam a regra da festa.

Entrada: A primeira linha de um caso de teste contém três inteiros N, P e K. N representa o número total de pessoas em uma comunidade($1 \leq N \leq 1000$). Cada pessoa é identificada por um número entre 1 e N. P representa o total de relações de amizades entre as pessoas da comunidade; e K é o número mínimo de amigos convidados que uma pessoa necessita ter para poder participar da festa ($1 \leq K \leq N\text{-}1$), como descrito acima. As próximas P linhas descrevem as relações de amizades representadas por dois inteiros a e b, onde a e b são amigos ($1 \leq a \leq N$, $1 \leq b \leq N$ e $a \neq b$).

Saída: Seu programa deverá imprimir uma única linha, contendo o tamanho do maior conjunto de pessoas convidadas para a festa encontrado pelo seu programa.

Exemplos de entrada e saída

- **Entrada 1**:
  6 6 2
  1 3
  3 5
  2 3
  2 4
  4 6

6 2
Saída 1:
3

- **Entrada 2**:
  6 6 3
  1 2
  2 3
  3 1
  4 5
  5 6
  6 4
  Saída 2:
  0

- **Entrada 3**:
  10 11 2 1 2
  1 3
  3 2
  3 5
  5 4
  5 6
  9 10
  8 9
  8 7
  6 7
  6 8
  Saída 3:
  7

3 Solução

A solução é baseada na representação de um grafo por lista de adjacência, implementada pela estrutura de dados representada na Figura 1.

```
struct NoListaAdj
{
    int dest;
    struct NoListaAdj* prox;
};
```

Figura 1: Estrutura da lista de adjacência

Onde *dest* representa o maior "id"do vértice do grafo e o *prox* representa as ligações, ou seja, as conectividades dos vértices.
A Figura 2 mostra a criação de uma aresta de um grafo Podemos observar na figura acima também uma representação de uma lista de adjacência, onde as ligações entre os vértices são representados, ao implementar os "dois lados", o grafo criado é não-orientado.

3.1 Implementação da Solução

A função mostrada na Figura 3 representa a função solução do problema.
Para cada cada uma das linhas da lista de adjacência é contada a quantidade de itens existentes, se o contador for menor que o K (número mínimo de amigos), significa que um dado vértice, que representa uma pessoa, não

```c
//adiciona uma aresta nao direcionada
void criaAresta(struct Grafo* g, int ini, int dest)
{

    struct NoListaAdj* criaNo = newNoListaAdj(dest);
    //adiciona a ida
    criaNo->prox = g->vet[ini].head;
    g->vet[ini].head = criaNo;

    //adiciona sentido contrario
    criaNo = newNoListaAdj(ini);
    criaNo->prox = g->vet[dest].head;
    g->vet[dest].head = criaNo;
}
```

Figura 2: Função de criação de uma aresta

```c
int ResolveProblem(struct Grafo* g, int K, int N)
{
    int i;
    int cont;
    int satisfaz = 0;
    int aux[N];
    struct NoListaAdj* item;

    for (i = 0; i < g->V; i++)
    {    cont = 0;
         item = g->vet[i].head;
        while (item)
        {   cont++; //conta a quantidade de itens de uma linha da lista
            item = item->prox;
        }
        if(cont<K)
          aux[i] = 0;
        else
          aux[i] = 1;
    }
    for (i = 0; i < g->V; i++)
    {   cont = 0;
        item = g->vet[i].head;
        while (item)
        {
          if(aux[item->dest]==1)
            cont++;
          item = item->prox;
        }
        if(cont>=K)
          satisfaz++;
    }
    return satisfaz;
}
```

Figura 3: Função Solução

possui o número suficiente de amigos para a festa.
Foi criado um vetor *aux* que armazena essas verificações, se o K for maior, a posição i do vetor é atribuido com o valor 1, ou seja, essa pessoa poderá participar da festa, senão é atribuido o valor 0.
Posteriormente, a lista de adjacência é percorrida novamente para que seja contada a quantidade de pessoas que podem ir à festa, isso significa a quantidade de posições do vetor *aux* em que o valor é 1, representando assim o maior conjunto de pessoas que poderão participar da festa.

3.2 Função Principal

A função principal do algoritmo é apresentada na Figura 4. Foi criado um grafo de N vértices e de acordo com

```
int main()
{
    int N, P, K;
    int i;
    int a,b;
    int result;

    scanf("%d", &N);
    scanf("%d", &P);
    scanf("%d", &K);
    struct Grafo* g = criaGrafo(N);
    for(i = 1; i<=P; i++){
      //insere as arestas
      scanf("%d", &a);
      scanf("%d", &b);
      if(a==0 || b==0)
        return 0;
      else
        criaAresta(g, a-1, b-1);
    }

    //imprime a solucao
    result = ResolveProblem(g, K, N);
    printf("%d", result );

    return 0;
}
```

Figura 4: Função Principal

as entradas são criadas as P arestas, logo em seguida, a função solução é aplicada.

4 Análise do Algoritmo

A ideia principal do algoritmo é percorrer uma lista de adjacência para buscar as pessoas/vértices que satisfaçam a condição para participar da festa.
Essa lista é percorrida 2 vezes, uma para preencher o vetor solução (*aux*) e o outro para realizar a contagem dos itens atribuidos com 1 desse vetor. A busca em uma lista é dada pela complexidade O(| P | + | N |), onde K e e N representam o número de arestas e o de vétices, respectivamente.