TP0: Hit do Verão

Aluno: Rafael Castro Araujo Beirão

1. Introdução

O objetivo deste trabalho é desenvolver um programa que leia uma lista de relações e propriedades dos nós de um grafo, e calcular quantos nós serão afetados por uma propagação a partir de um nó escolhido inicialmente. A regra da propagação depende do atributo “idade” dos nós, onde, se ela for menor que 35, o nó fará parte dos nós afetados, e tentará propagar para todos os seus vizinhos seguindo a mesma regra. O resultado esperado é o número de nós afetados.

Uma abstração utilizada para melhor calcular esse número de nós afetados, é a remoção de arestas com nós com “idade” maior ou igual a 35, transformando assim o problema em um problema análogo de calcular o número de nós no subgrafo isolado que contém o nó inicial, criado a partir da remoção dessas arestas.

2. Estruturação do Problema e Complexidade de Tempo

A entrada do programa é uma descrição do grafo na forma:

```
V //numero de nós
E //numero de arestas
ID Idade //id do nó e sua "idade"
ID1 ID2  //aresta do nó ID1 e ID2 (não-direcionado)
ID //id do primeiro nó afetado
```

O problema foi dividido em duas partes, primeiro a construção do grafo, em seguida a propagação do efeito e a contagem dos elementos do sub-grafo.

- **Construção do grafo**

O grafo é baseado na estrutura de dados chamada Lista de Adjacência[1], alocando dinamicamente um vetor de objetos “Nó”, de tamanho |V|, e, para cada Nó, criando uma lista unicamente encadeada contendo ponteiros para os vizinhos(não direcionados) desse Nó. Para facilitar a propagação do efeito, nenhuma adjacência onde algum dos nós possui “idade” maior ou igual a 35.

A escolha de um vetor para conter os Nós se dá pela facilidade de lookup pelo id do nó, que é O(1). Além disso, os nós não ficam diretamente armazenados no vetor, mas sim apontadores de memória para suas posições reais, que são alocados dinamicamente. Toda estrutura é representada por um apontador no código para evitar blocos grandes de memória contínua, e para evitar o overhead de passar estruturas grandes para funções por valor.

A leitura do tamanho e alocação do vetor leva tempo O(1) (assumindo o tempo da rotina embutida “malloc” como O(1)). A alocação das estruturas Nó e armazenamento de suas idades toma tempo O(1) cada, |V| vezes, portanto O(|V|).

|E| adjacencias(sem repetições) devem ser lidas, e o processo de adicionar a adjacência nas listas possui complexidade de tempo O(1), portanto a construção das listas de adjacências leva tempo O(|E|)

Portanto, a construção do grafo por Lista de Adjacência leva tempo O(|V|) + O(|E|) = O(|V| + |E|)

- **Propagação do Efeito**

Na segunda parte do problema, é feita uma propagação para todos os vizinhos do primeiro nó, e para todos os vizinhos subsequentes, e contado por quantos nós são passados. Como todas as arestas com nós com idade maior ou igual a 35 foram removidas, então a unica checagem necessária é se o nó já foi percorrido, o que é indicado por uma flag na estrutura do Nó, que é inicializada como 0.

Para fazer uma implementação da propagação semelhante à busca Breadth-First, foi implementada uma fila, com criação, enfilamento e remoção todos O(1). O primeiro nó é adicionado à essa fila como inicialização, e o processo a seguir é executado:

[1] CLRS, Introduction to Algorithms, capitulo 22

```
numero = 0
enquanto !fila.vazia:
      No = fila.dequeue()
      numero++
      Para todo vizinho de No:
            Se !vizinho.passado:
                  vizinho.passado = TRUE
                  fila.enqueue(vizinho)
return numero
```

Esse procedimento passará por todos os nós no subgrafo isolado do nó inicial, para cada nó terá uma iteração no segundo loop para cada um de seus vizinhos, portanto cada nó terá custo

$$T(G) = \sum_{i}^{|V'|} O(b'_i) = O(|E'|)$$

Onde V', E' e b_i' são os nós, arestas, e grau do nó i no subgrafo isolado do primeiro nó. No pior caso, V' = V, E' =E, e b_i' = b_i, então o pior caso dessa parte do programa será O(|E|).

- **Liberação da memória**

  Liberar uma fila vazia é O(1).

  Para liberar um grafo, primeiro é necessário liberar todas as listas de adjacência, que é O(|E|), e em seguida liberar todos os nós, O(|V|), e liberar o vetor, O(1).

  Portanto, a liberação de memória também é O(|V| + |E|)

- **Conclusão**

  Como o programa é dividido em três partes independentes, o custo em tempo será a soma das 3 complexidades:

$$T(G(V, E)) = O(|V| + |E|) + O(|E'|) + O(|V| + |E|) = O(|V| + |E|)$$

  Portanto, a complexidade de tempo do programa é O(|V| + |E|).

3. Complexidade de Tempo

- **Construção do grafo**

  Nessa parte do programa, é a alocada toda a estrutura do grafo, que é composta de:

  - Um vetor de ponteiros de tamanho |V|
  - |V| Estruturas de nós com O(1) atributos
  - 2 * |E'| relações em listas de relações, com pior caso |E'| = |E|

  Portanto o grafo possui complexidade de espaço O(|E'| + |V|) com pior caso O(|E| + |V|)

- **Propagação do Efeito**

Nessa parte do programa, a única alocação é a da fila, que tem complexidade de espaço O(n) com n sendo o número de elementos dentro da pilha, que no pior caso pode ser |V'| - 1 com |V'| = |V| (quando o primeiro elemento tem uma aresta com todos os outros elementos), portanto complexidade de memória O(|V|) no pior caso.

- **Liberação da memória**

Não há alocação extra de memória nessa parte do programa, portanto a imagem de memória é O(1) pelas variáveis locais apenas

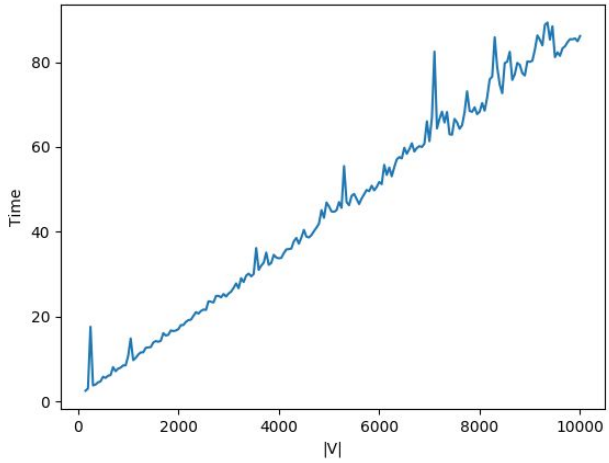
$$M(G(V, E)) = O(|V| + |E|) + O(|V|) + O(1) = O(|V| + |E|)$$

4. Analise dos Experimentos

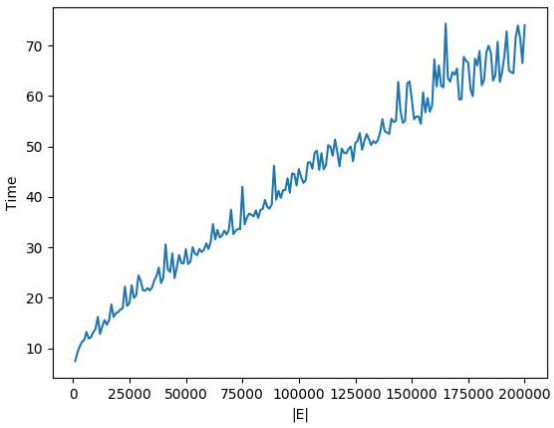
Utilizando algumas ferramentas para geração de grafos e gráficos[2], foram criados dois gráficos, o fixando fixando o valor esperado de |E| em 10.000 aumentando linearmente |V| de 150 até 10.000 com step=50, o segundo fixando |V| em 5.000 e aumentando linearmente o valor esperado de |E| de 1.000 até 200.000 com step=1000,

O grafo é gerado com o modelo Erdos-Renyi[3], com $|E| = \sum_{i}^{|V|^2/2} p = \frac{|V|^2 p}{2}$

$\rightarrow p = \frac{2|E|}{|V|^2}$



Como observável nesse gráfico, com |E| esperado fixo, o tempo gasto do programa possui uma relação observável linear com |V|, com alguma variancia provavelmente devida à diferença entre V e V' ou devido à variância no valor real de |E|.



[2] https://github.com/RCastroAB/tp0_helper/
[3] https://en.wikipedia.org/wiki/Erd%C5%91s%E2%80%93R%C3%A9nyi_model

Também é bem marcada a relação linear entre o tempo e |E|, embora tenha uma variância mais marcada pois |E| nesse caso é o valor esperado de |E|, devido à imprecisão do método de geração de grafos.

5. Conclusão

Neste trabalho foi resolvido o problema de propagação de um efeito por um grafo de acordo com alguma propriedade binária (no caso, $a > 35$), utilizando a representação da lista de relações e um algoritmo de propagação Breadth-First. O algoritmo é eficiente para grafos esparsos, tendo complexidade O(|V| + |E|) tanto em tempo quanto espaço. Parte do processo envolve a separação do grafo em subgrafos isolados, e contar o número de nós no subgrafo isolado do primeiro nó, então um trabalho futuro seria explorar algoritmos que consigam calcular o tamanho de um subgrafo isolado eficientemente.