Trabalho 1 - Geração de árvores aleatórias

Objetivo e regras

O objetivo deste trabalho é praticar a implementação de algoritmos em grafos.

O trabalho é em equipe de até duas pessoas. O compartilhamento de informações entre as equipes é permitido e aconselhado, mas **o compartilhamento de código não é permitido**. Trabalhos que tenham porções de código iguais ou copiados da internet, serão anulados.

Se você já fez o trabalho no ano passo, você terá que fazer o trabalho novamente do zero, sem reutilizar o código que você já tem.

Será feito uma comparação de todos os trabalhos desse ano com todos os trabalhos do ano passado, por favor, faça o seu próprio trabalho para evitar transtornos.

Descrição

Uma árvore é um grafo não orientado conexo e acíclico. Neste trabalho vamos implementar e testar três algoritmos para geração de árvores aleatórias: baseado em passeio aleatório, no algoritmo de Kruskal e no algoritmo de Prim.

Cada algoritmo recebe como entrada um número $n > 0$ e produz uma árvore aleatória com n vértices. Os algoritmos são:

RANDOM-TREE-RANDOM-WALK(n)

```
 1  Crie um grafo G com n vértices
 2  for each vertex u ∈ G.V
 3       u.visitado = FALSE
 4  u = um vértice qualquer de G.V
 5  u.visitado = TRUE
 6  while |G.E| < n − 1
 7       v = um vértice aleatório de G.V
 8       if v.visitado == FALSE
 9            Adicione (u, v) em G.E
10            v.visitado = TRUE
11       u = v
12  return G
```

RANDOM-TREE-KRUSKAL(n)

1 Crie um grafo completo G com n vértices
2 **for** each edge $(u, v) \in G.E$
3 $\quad (u, v).w =$ valor aleatório entre 0 e 1
4 MST-KRUSKAL(G, w)
5 **return** Um grafo construido com as arestas da árvore produzida por MST-KRUSKAL

RANDOM-TREE-PRIM(n)

1 Crie um grafo completo G com n vértices
2 **for** each edge $(u, v) \in G.E$
3 $\quad (u, v).w =$ valor aleatório entre 0 e 1
4 $s =$ um vértice qualquer de $G.V$
5 MST-PRIM(G, w, s)
6 **return** Um grafo construindo com as arestas da árvore produzida por MST-PRIM

Desenvolvimento

A implementação de cada algoritmo deve seguir o processo de criação de programas que discutimos em sala. Todas as funções e procedimentos devem ter o propósito e os testes. Os testes para os algoritmos determinísticos, como MST-KRUSKAL e MST-PRIM, devem ser escritos de forma semelhante ao que fizemos em sala: utilizando exemplos de entrada e verificando se a saída está de acordo com o esperado.

Para testar os algoritmos não determinísticos (que geram as árvores aleatórias) precisamos de outra estratégia, afinal, como verificar se uma saída é a esperada se o algoritmo é não determinístico!?

Neste caso, vamos verificar propriedades das saídas e não as saídas em si. Vamos considerar dois tipos de verificação: uma propriedade que cada saída deve ter e uma propriedade que um conjunto de saídas deve ter.

Uma propriedade simples que podemos verificar de cada saída é se ela é uma árvore. Pode parecer simples esta verificação, mas ela é importante. Lembre-se, um dos objetivos dos testes é identificar erros no código, como é plausível que um erro faça a função gerar saídas que não sejam árvores, então é importante fazer a verificação.

Também precisamos verificar se as árvores estão sendo geradas segundo a distribuição esperada. Neste caso não testamos uma saída apenas, mas sim se uma medida estatística de muitas saídas está de acordo com o esperado. Para este trabalho vamos calcular a média do diâmetro das árvores. O diâmetro de uma árvore T é o comprimento do maior caminho em T. Podemos calcular o diâmetro de uma árvore usando o algoritmo de busca em largura da seguinte maneira:

DIAMETER(T)

1 $s =$ vértice qualquer de $T.V$
2 $a =$ o vértice com valor máximo de d obtido por BFS(T, s)
3 $b =$ o vértice com valor máximo de d obtido por BFS(T, a)
4 **return** A distância entre a e b

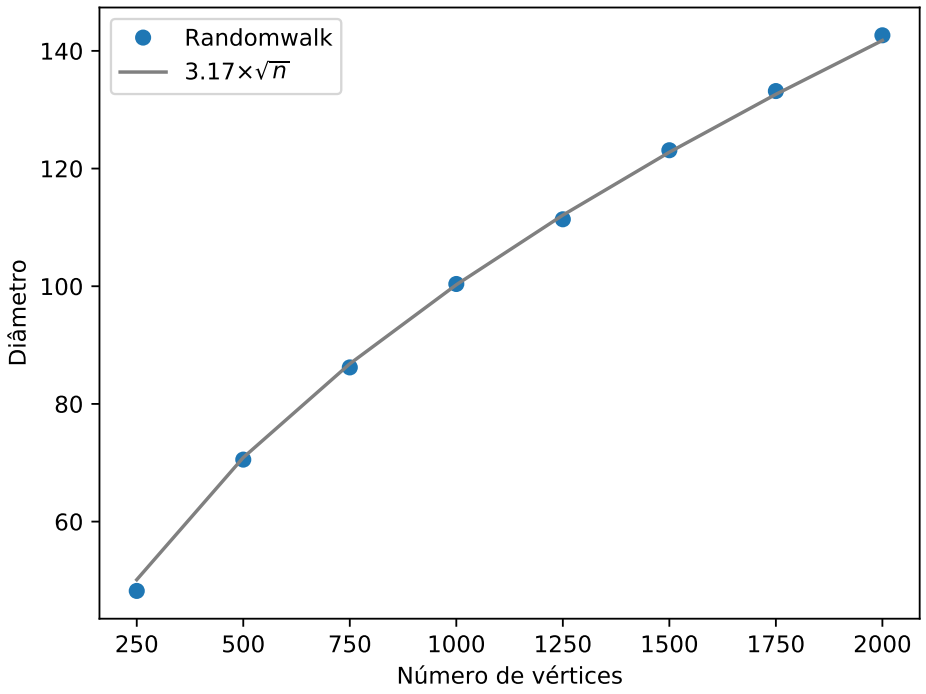
Qual é o diâmetro esperado para uma árvore gerada por cada um dos três algoritmos que vamos implementar? No algoritmo RANDOM-TREE-RANDOM-WALK cada árvore de n vértices tem a mesma chance de ser gerada, e neste caso o diâmetro esperado é $O(\sqrt{n})$ (Szekeres 1983). Já nos algoritmos RANDOM-TREE-KRUSKAL e RANDOM-TREE-PRIM existe uma tendência de gerar árvores com diâmetros menores, neste caso, o diâmetro esperado é $O(\sqrt[3]{n})$ (Addario-Berry *at al.* 2006).

Então, para testar os algoritmos de árvores aleatórias vamos executar cada algoritmo 500 vezes para os valores de $n = \{250, 500, ... 2000\}$ e verificar se todas as saídas produzidas são árvores. Além disso, para cada algoritmo vamos calcular o diâmetro médio das árvores para cada valor de n e comparar com o valor esperado.

Para facilitar a comparação do diâmetro você pode utilizar o programa disponível na página dessa atividade. Para executar o programa é necessário o Python 3 com as bibliotecas `numpy`, `scipy` e `matplotlib`. Para usar o programa você deve salvar os resultados do experimento em um arquivo como o exemplo a seguir:

```
250 48.218
500 70.534
750 86.204
1000 100.382
1250 111.378
1500 123.114
1750 133.172
2000 142.648
```

A primeira coluna é o valor de n e a segunda o valor médio do diâmetro das árvores geradas (para o valor de n correspondente). Supondo que os resultados para o algoritmo Random-Tree-Random-Walk estejam em um arquivo chamado `randomwalk.txt`, o comando `python3 plot.py randomwalk < randomwalk.txt` vai gerar um arquivo chamado `randomwalk.pdf` com um gráfico semelhante a este:



O gráfico mostra que os valores obtidos (pontos azuis) estão de acordo com os valores esperados (linha cinza). Para o algoritmo Random-Tree-Kruskal você deve trocar o argumento `randomwalk` para `kruskal` e para o algoritmo Random-Tree-Prim para `prim`. Os resultados para estes dois últimos algoritmos poderão não ser tão "bonitos" (a curva não ficará tão justa) quanto para o primeiro algoritmo.

O programa pode ser escrito em: C, C++, Java, Python ou Rust.

Dicas de implementação

- Para cada etapa do trabalho, siga o processo de construção de funções que vimos em sala;
- Para uso nos procedimentos MST-Kruskal e MST-Prim represente o grafo com matriz de adjacências, para os demais procedimentos use lista de adjacências;
- Não use variáveis globais!

Entrega

O trabalho deve ser entregue por partes de acordo com o cronograma a seguir:

25/03: Função Diameter

09/04: Função Random-Tree-Random-Walk

19/04: Função Random-Tree-Kruskal

29/04: Função Random-Tree-Prim

09/05: Relatório

Observe que para implementar e testar cada uma dessas funções outras funções deveram ser implementadas. Cada entrega deve estar completa, com propósito e testes para cada função e não apenas para a função principal.

O código deve ser escrito e entregue em apenas **um arquivo de código fonte**. A cada entrega todo o código já escrito, mesmo das entregas anteriores, deve ser entregue.

O relatório deve ser entregue em um arquivo PDF, ter no máximo 5 páginas e conter:

- Introdução
- Desenvolvimento contendo pelo menos
  - Qual linguagem foi utilizada
  - Como os grafos e atributos foram representados
  - Informações importantes sobre a implementação
- Resultados contendo pelo menos
  - Configuração do computador que os testes foram executados
  - Uma tabela comparativa dos tempos de execução médio de cada algoritmo para cada valor de n e a avaliação desse resultado
  - Os gráficos dos diâmetros médios de cada algoritmo e uma avaliação desses gráficos
- Descrição da experiência de desenvolvimento do trabalho contendo pelo menos
  - As dificuldades encontradas e como elas foram superadas
  - Destaques sobre foi interessante, importante ou surpreendente
  - Resultado final do aprendizado
  - O que poderia ter sido diferente

Avaliação

Cada entrega de código será avaliada individualmente de acordo com os seguintes critérios:

- Corretude e tempo de execução: o programa deve funcionar de acordo com a descrição e deve passar em todos os testes. Além disso, os algoritmos devem ser implementados com tempo de execução de acordo com a análise feita em sala;

- Completude: o programa deve estar completo, incluindo descrição e testes automatizados;
- Organização: o programa deve estar bem organizado e usar boas práticas de programação;
- Entendimento: o aluno deve entender o programa que entregou e ser capaz de fazer alterações no código.

O relatório será avaliado de acordo com os seguintes critérios:

- Completude: deve conter todos os itens que foram pedidos;
- Escrita: o texto deve estar coeso, coerente e escrito usando a normal-culta.

Referências

- Szekeres, G. Distribution of labelled trees by diameter. In: CASSE, L. R. A. (Ed.). Combinatorial Mathematics X. Springer Berlin Heidelberg, 1983. (Lecture Notes in Mathematics), p. 392–397. Disponível em: https://doi.org/10.1007/BFb0071532.
- Addario-Berry, L., Nicolas B. and Bruce A. R. The Diameter of the Minimum Spanning Tree of a Complete Graph. 2006. Proceedings of the Fourth Colloquium on Mathematics and Computer Science, Ed. P. Chassaing, 237-248. Nancy: Discrete Mathematics and Theoretical Computer Science.