Documentação TP1

Giovanna Louzi Bellonia
Algoritmos I

24 de junho de 2019

1 Descrição do Problema

Alexander quer a criação de um algoritmo que permita que bandeiras sejam distribuídas por pontos em uma trilha de forma que todo caminho existente entre dois pontos sempre seja incidente a pelo menos uma bandeira. Esse algoritmo será divididos em dois casos, sendo o primeiro quando não existem ciclos na trilha, e o segundo quando existem. Para o primeiro caso, será retornado a menor quantidade de bandeiras necessária para a resolução do problema, enquanto que pro segundo será retornado, além disso, os pontos em que as bandeiras foram colocadas, sendo que essa resposta é até 2x menos pior que a solução ótima.

2 Estruturas de dados e algoritmos utilizados

2.1 TADs Utilizados

Foi criado um tad **aresta_t** com os seguintes componentes:

- **v1:** com o índice do vértice 1 que compõe a aresta;
- **v2:** com o índice do vértice 2 que compõe a aresta;

Foi criado um tad **grafo_t** com os seguintes componentes:

- **int quant_vert:** indica a quantidade de vertíces que o grafo possui;
- **int quant_arestas:** indica a quantidade de arestas que o grafo possui;
- **aresta_t *arestas:** um vetor de arestas, indicando as arestas presentes no grafo;

Foi criado um tad **arvore_t** com os seguintes componentes:

- **int quant_vert:** indica a quantidade de vertíces que a árvore possui;
- **lista_t **lista_adj:** um vetor de listas onde a posição n do vetor indica o vértice n, e o primeiro nó da lista dessa posição é o próprio vértice n, e os seguintes da lista são todos os filhos desse vértice.

2.2 Algoritmos utilizados

2.2.1 Vertex Cover Árvore

O algoritmo do vertex cover para árvore funciona considerando que se um vértice não participa da resposta é certo que seus filhos (quando existem) deverão fazer parte da resposta. Além disso, é considerado que entre uma folha e seu pai, sempre será escolhido o pai como parte da resposta, já que não faria diferença para a aresta que compartilham qual dos dois escolher, mas o pai pode ser vantajoso por cobrir mais arestas.

Pensando nisso, o algoritmo quebra em problemas menores, procurando a melhor resposta para cada conjunto de vértices conectados e considerando qual o melhor peso (quantidade de vértices que fazem parte da resposta) caso ele faça parte ou não do resultado. O menor problema é o da folha, que vai retornando recursivamente até a raiz, sendo que as folhas possuem um peso de 0 e a raiz da árvore terá soma dos melhores pesos possíveis, resultando na resposta final.

O algoritmo transforma um grafo não dirigido em uma árvore. Isso é feito transformando o nó 0 como a cabeça da árvore e fazendo com que seus filhos não tenham o 0 presente em sua lista de adjacência (que será a lista de filhos). Isso é feita a cada recursão de forma que todo nó filho não tenha mais o pai em sua lista.

2.2.2 Vertex Cover Grafo

O algoritmo de vertex cover para grafos funciona caminhando por todas as arestas do grafo. É guardado em um vetor com tamanho da quantidade de vértices se o vértice já foi escolhido como parte da resposta ou não (1 se foi e 0 se não foi). Para cada aresta, se ambos os vértices dela não tiverem sido escolhidos ainda, serão agora parte da resposta.

2.3 Makefile

Para gerar o compilável tp3 é só utilizar o comando *make*.

3 Análise de Complexidade

3.1 Espaço

Para o primeiro caso a árvore utilizada guarda os vértices e as arestas na lista de adjacência, mas de forma que cada vértice apareça apenas uma vez, e inicialmente, cada aresta aparece duas vezes (uma em cada vértice que a possui). Dessa forma, o gasto de espaço é $O(V + 2E) = O(V + E)$, sendo V os vértices e E as arestas.

Para o segundo caso, o grafo utilizado guarda todas as arestas e todos os vértices cada um em um vetor. Logo, o gasto de espaço é $O(V + E)$, sendo V os vértices e E as arestas.

3.2 Tempo

O algoritmo do vertex cover para árvore calcula o vertex cover pra todos os vértices uma única vez (já que fica guardado esse valor no vértice depois da primeira vez que foi calculado). E, para cada vértice, deve percorrer sua llsta de filhos, que ao somar a quantidade terão um valor da quantidade de arestas. Logo, o tempo gasto é $O(V + E)$.

Já o algoritmo do vertex cover para grafos percorre todas as arestas, visitando uma única vez cada uma. Além de percorrer a lista de vértices de resposta para imprimi-la, que no pior caso terá todos os vértices. Logo, o tempo gasto é $O(V + E)$.

4 Prova de Corretude

4.1 Vertex Cover Árvore

A resposta final do problema a partir de um vértice depende de dois fatores: (1) se o vértice participa ou não do resultado e (2) qual é o valor do resultado para seus filhos e netos. Ao utilizarmos da programação dinâmica e dividirmos o problema em problemas menores, conseguimos a resposta ótima para os vértices que não possuem netos e filhos.

Com isso, para problemas maiores, já teremos o segundo fator necessário para a resposta. Logo, para o primeiro fator, é so compararmos os valores com o vértice participando ou não do resultado e saberemos que o menor entre os dois é o ótimo para aquele problema.

Se para cada subproblema estamos garantindo que a resposta é ótima, podemos garantir que a resposta do problema final (que é a junção desses subproblemas) também será ótima.

4.2 Vertex Cover Grafo

Ao caminhar pelas arestas do grafo, temos os seguintes casos para os vértices que as compõe:

1. Nenhum vértice foi escolhido
2. Ambos os vértices já foram escolhidos
3. Um vértice já foi escolhido e outro não

No nosso algoritmo, colocamos apenas os vértices do primeiro caso como parte da resposta, e isso funciona pois não é necessário inserir o dos outros casos já que:

1. Para o primeiro caso, eles já são parte da resposta e a aresta já é com certeza coberta.
2. Para o segundo caso, se um deles já foi escolhido, quer dizer que a aresta que estamos analisando já foi coberta, ou seja, não precisamos escolher o outro vértice para isso. Entretanto, o vértice que não foi escolhido ainda pode ter outras arestas que foram ou não escolhidas. Ele só deverá também participar da resposta final se houver arestas que conectam-se a ele que ainda não foram cobertas, e isso só ocorrerá se o outro vértice participante também não foi escolhido, que será uma aresta a ser coberta pelo caso 1.

Provando que o algoritmo realmente é um vertex cover.

Podemos comprovar que o algoritmo é no máximo duas vezes pior que a solução ótima pois sempre escolhemos os dois vértices que cobrem uma única aresta, sendo que, na solução ótima, só escolheriamos um dos vértices, por que a aresta já estaria coberta dessa forma.

5 Avaliação Experimental

Para a análise experimental foram criados testes aleátorios, sendo agrupados em árvores binárias e árvores lineares, de forma que houvesse uma quantidade crescente de vértices e arestas a cada teste feito (no caso a quantidade de arestas era sempre uma unidade a menos que o vértice, para facilitar a criação das árvores). Cada teste foi rodado 5 vezes e tirado a média do tempo gasto para a resolução do mesmo. No fim os valores dos tempos médios gastos em cada teste foram plotados nos gráficos abaixo de acordo com a quantidade de vértices utilizadas.

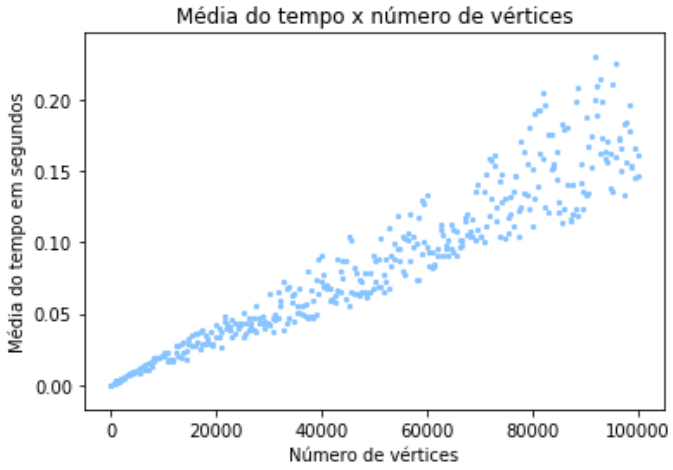

Figura 1: Gráfico para Vértices X Tempo para o caso 1

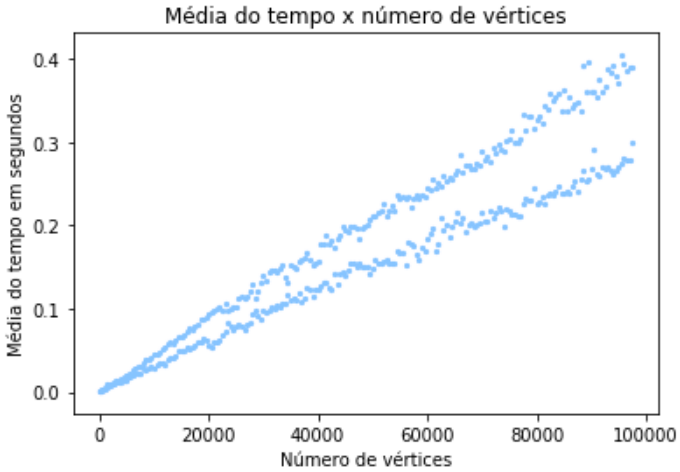

Figura 2: Gráfico para Vértices X Tempo para o caso 2

Com isso, pode-se perceber, pelo formato do gráfico, que o tempo gasto realmente é linear em relação a quantidade de vértices ($O(V)$), logo linear também para a quantidade de arestas (já que nesses teste é quase igual a de vértices), tanto para o algoritmo 1 quanto para o algoritmo 2.