Trabalho Prático 3

Breno Cláudio de Sena Pimenta

[1]Departamento de Ciência da Computação
Universidade Federal de Minas Gerais
DCC216 - Algoritmos I

1. Introdução

O trabalho prático consiste em calcular o mínimo de depósitos de vacinas necessários para abastecer vilas em uma área florestal. Na primeira tarefa esse mínimo deve ser calculado para uma malha que não forma ciclos, já na segunda tarefa as malhas podem formar ciclos e deve-se fazer uma aproximação do valor mínimo de depósitos. Em ambos os casos, uma vila é dita abastecida se nela houver um depósito ou se estiver conectada por um único caminho a outra vila que contém um depósito.

2. Modelagem Computacional

O problema foi modelado como um grafo $G(V, E)$, no qual os vértices representam as vilas agrícolas e as arestas representam os caminhos nãos-direcionados e não-ponderados existentes na malha. O número mínimo de depósitos de vacinas que devem ser construídos para possibilitar que todos as vilas sejam atendidas é o tamanho do conjunto solução do problema de cobertura de vértices para o grafo G.

Assim, para solucionar o problema proposto pela tarefa 1, foi implementado o algoritmo em programação dinâmica que encontra a solução ótima para a cobertura de vértices (Algoritmo 1). Para solucionar o problema proposto pela tarefa 2 foi feito um algoritmo para aproximar a solução de modo que não seja duas vezes maior que a solução ótima. Os algoritmos estão descritos a seguir.

2.1. Tarefa 1

Para solucionar a tarefa 1, foi necessário utilizar do paradigma de programação dinâmica para encontrar a solução ótima de forma eficiente. No conjunto solução S que contém o menor número de postos possível, ou seja, ótima, cada vértice $v \in G$ pode conter um depósito ou não. Assim é necessário computar para cada posto:

- $v \in S$: nesse caso o tamanho de S aumenta em 1 devido a presença de v e deve-se calcular recursivamente o mesmo problema para os vértices vizinhos de v. Eles podem ou não fazer parte do conjunto solução. O custo de incluir v é somado ao custo de incluir ou não cada um de seus vizinhos.
- $v \notin S$: nesse caso, cada vizinho de v deve ser incluído em S e deve ser calculado o mesmo problema para os vizinhos dos vizinhos de v. Assim, o custo para excluir v aumenta com o mínimo de se incluir seus vizinhos.

O algoritmo deve escolher o menor custo dado os dois casos apresentados anteriormente. O valor do custo de incluir ou excluir cada vértice da solução é salvo em uma memória de custos, assim, para vértices já previamente calculados, basta consultar a memória para evitar executar uma computação já realizada. Assim, a equação de recorrência do problema de encontrar a cobertura de vértices CV para um vértice i é a seguinte:

$$CV(i) = min\left(\sum_{j:j\in vizinhos(i)}\left(1+\sum_{k:k\in vizinhos(j)} CV(k)\right), 1+\sum_{j:j\in vizinhos(i)} CV(j)\right) \tag{1}$$

Algorithm 1: Tarefa 1: Encontrar a menor cobertura de vértices

input : Grafo G e vértices de origem v
output: Custo mínimo r de incluir ou excluir v

visitados $\leftarrow v$
foreach $V \in G$ **do**
    if *V não foi visitado* **then**
        Rode o algoritmo da tarefa 1 recursivamente em v
custoIncluir(v) $\leftarrow$ 1
custoExcluir(v) $\leftarrow$ 0
foreach *vizinho* $n \in v$ **do**
    custoIncluir(v) $\leftarrow$ custoIncluir(v) + **min**(custoIncluir(n), custoExcluir(n))
    custoExcluir(v) $\leftarrow$ custoExcluir(v) + custoIncluir(n)
r $\leftarrow$ **min**(custoIncluir(v), custoExcluir(v))
return r

Ao fim das execuções recursivas, o algoritmo terá calculado para todos os vértices a melhor opção entre incluí-lo ou excluí-lo. Assim, terá retornado o tamanho mínimo da cobertura de vértices para o grafo dado. Para os casos triviais em que o número de vértices é menor que 2, basta retornar 0.

2.1.1. Prova de Corretude da Tarefa 1

Lemma 2.1. *A implementação do algoritmo de gerar um solução para a cobertura de vértices encontra o menor conjunto solução possível.*

Demonstração. Por indução, considere um único vértice m sem arestas. Nesse caso, o algoritmo funciona corretamente pois retorna um cobertura de tamanho zero. Suponha que para um vértice n, o algoritmo calcula corretamente a cobertura de todos os subconjuntos formados pelos seus vizinhos, e pelos vizinhos de seus vizinhos, assim por diante. Assim, uma cobertura de vértices para o vértice n possui dois casos:

- Inclua n: assim, todas as arestas entre n e seus vizinhos são cobertas, e a solução é a união de n e as coberturas de vértices dos vizinhos de n. Esse caso entra na parte (2) da Equação 1.
- Exclua n: nesse caso, a solução deve incluir todos os vizinhos de n, juntamente com a solução já computada para a cobertura de vértices dos vizinhos de n. Esse caso entra na parte (1) da Equação 1.

Assim, basta escolher o menor valor entre incluir e excluir n, encontrando assim uma solução ótima. ■

2.2. Tarefa 2

Para solucionar a tarefa, 2 foi necessário utilizar do paradigma algoritmos gulosos para encontrar a solução no máximo duas vezes pior que a ótima. O algoritmo para encontrar o conjunto solução S^{aprox} que contém um número mínimo aproximado de postos para um dado grafo $G = (V, E)$ e uma aresta qualquer u, v é o seguinte:

Algorithm 2: Tarefa 2: Encontrar uma aproximação para a cobertura de vértices

input : Grafo G, e uma aresta qualquer $\{u, v\}$
output: Conjunto solução S^{aprox}

$S^{\text{aprox}} \leftarrow \emptyset$
while $E \neq \emptyset$ **do**
    Escolha qualquer aresta $\{u, v\} \in E$
    $S^{\text{aprox}} \leftarrow S^{\text{aprox}} \cup \{u, v\}$
    Remova de E todas as arestas que incidem em u ou em v
return S^{aprox}

2.2.1. Prova de Corretude da Tarefa 2

Lemma 2.2. *O algoritmo retorna uma cobertura de vértices.*

Demonstração. Pode-se afirmar que todas as arestas $e \in S^{\text{aprox}}$ estão claramente cobertas. Caso uma aresta $e \notin S^{\text{aprox}}$ não está coberta isso implica que o algoritmo irá realizar $S^{\text{aprox}} \leftarrow S^{\text{aprox}} \cup e$, portanto não haverá aresta não coberta no grafo restante. ■

Lemma 2.3. *O algoritmo retorna uma cobertura de vértices de tamanho no máximo duas vezes maior que a solução ótima.*

Demonstração. Seja A as arestas que contem os vértices incluídos na solução pelo algoritmo da tarefa 2. A cobertura ótima $|C^{\text{opt}}|$ deve conter pelo menos uma das extremidades das arestas em A. Nenhuma duas arestas em A compartilham das mesmas extremidades, pois essas arestas são removidas pelo algoritmo. Como nenhuma duas arestas em A são cobertas pelo mesmo vértice em $|C^{\text{opt}}|$, temos um limite inferior para $|C^{\text{opt}}|$:

$$|C^{\text{opt}}| \geq |A| \tag{2}$$

Um limite superior para a solução do algoritmo S^{aprox} pode ser obtido ao considerar que, a cada vez que uma aresta é válida para ser adicionada na solução, uma dupla de vértices $\{u, v\}$ que representam suas extremidades é inserida, assim temos:

$$|S^{\text{aprox}}| = 2|A| \tag{3}$$

Combinando as duas equações anteriores temos que:

$$2|S^{\text{aprox}}| \leq 2|C^{\text{opt}}| \tag{4}$$

■

3. Implementação da Solução

A solução do problema proposto foi desenvolvida em C++ utilizando o paradigma de orientação a objetos.

3.1. Estruturas de dados

A representação de um grafo em todo o prolema foi feita por meio de um vetor de vetores de inteiros da biblioteca STL (Standard Template Library) por meio da declaração *vector<vector<int>>*.

O seu uso se assemelha às listas de adjacência, logo se a aresta $a \rightarrow b$ existe em G, ela é representada em alguma posição (a, n) do vetor de vetores g, ou seja, $g[a][n] = b$. Para implementar os algoritmo modelados anteriormente foram criadas três classes: Graph, Task1 e Task2.

3.2. Classe Graph

A classe Graph representa um grafo qualquer. Foi criada apenas com o propósito de organizar os dados de entrada. Ela possui funções para ler dados da entrada por meio de um arquivo e de retornar o grafo G.

3.3. Classe Task1

A classe Task1 contém os métodos para solucionar o problema da tarefa 1. Para um grafo não-direcionado e acíclico ela encontra o tamanho da menor cobertura de vértices. Como essa classe contém a parte principal da solução desenvolvida, seus atributos e métodos são detalhados a seguir.

3.3.1. Atributos de Task1

A classe Task1 contém apenas os seguintes atributos privados:

- **_costs** Esse vetor de pares de inteiros é usado como memória do algoritmo de programação dinâmica para armazenar na primeira posição o custo de se incluir um dado vértice e na segunda posição o custo de não se incluir um dado vértice.
- **_graph:** Esse grafo representa o grafo de entrada G. Cada vértice representa uma vila e cada aresta é uma rota existente entre as vilas.

3.3.2. Métodos de Task1

A classe Task1 contém os seguintes métodos:

- **Construtor Task1**: O construtor da classe recebe o grafo já lido e construído pela classe Graph e define o tamanho do vetor _costs para o número de vértices no grafo.
- **min**: Essa função retorna o mínimo entre dois inteiros.
- **calculateMinimumVextexCoverSize**: Essa função executa o algoritmo descrito anteriormente para a tarefa 1. Ela usa um for inicial para chamar recursivamente a função para todos os vértices e calcular o custo para cada um deles. Essa função retorna o valor mínimo para a cobertura de vértices partindo-se de um vértice inicial passado como parâmetro, juntamente com a lista de visitados.
- **printMinimumVextexCoverSize**: Esse método público é usado para chamar a função calculateMinimumVextexCoverSize já explicada anteriormente. Caso o grafo tenha menos que 2 a solução é trivial e ela imprime zero na tela. Caso contrário, ela executa a função calculateMinimumVextexCoverSize a partir do vértice 0 e de um vetor boleando passado como argumento.

3.4. Classe Task2

A classe Task2 contém os métodos para solucionar o problema da tarefa 2. Para um grafo não-direcionado e cíclico ela encontra o tamanho aproximado cobertura de vértices, sendo este não duas vezes maior que a solução ótima. Como essa classe contém a parte principal da solução desenvolvida, seus atributos e métodos são detalhados a seguir.

3.4.1. Atributos de Task2

A classe Task2 contém apenas os seguintes atributos privados:

- **_resultSet** Esse vetor é usado para armazenar os vértices que fazem parte do conjunto aproximado da solução para o problema de cobertura.
- **_graph:** Esse grafo representa o grafo de entrada G. Cada vértice representa uma vila e cada aresta é uma rota existente entre as vilas.

3.4.2. Métodos de Task2

A classe Task2 contém os seguintes métodos:

- **Construtor Task2**: O construtor da classe recebe o grafo já lido e construído pela classe Graph.
- **calculateMinimumVextexCoverAprox**: Essa função executa o algoritmo descrito anteriormente para a tarefa 2. Para iterar sobre as arestas, ela visita cada par de vértices em G por meio de dois for, caso não tenham sido ambos visitados, essa aresta deve ser adicionada ao conjunto solução e os vértices devem ser marcados como visitados.
- **printMinimumVextexCoverSizeAprox**: Esse método público é usado para chamar a função calculateMinimumVextexCoverAprox já explicada anteriormente. Caso o grafo tenha menos que 2 a solução é trivial e ela imprime zero na tela. Caso contrário, ela executa a função calculateMinimumVextexCoverAprox. O vetor de solução é um atributo da classe. Antes de imprimir na tela o seu tamanho e os vértices que o compõe, é feita uma ordenação crescente.

4. Análise de Complexidade Assintótica de Tempo e Espaço

Existem dois algoritmos necessários para solucionar ambas tarefas. Esses algoritmos foram apresentados anteriormente pelos algoritmos 1: Tarefa 1: Encontrar a menor cobertura de vértices e 2: Tarefa 2: Encontrar uma aproximação para a cobertura de vértices. A complexidade assintótica de cada um desses algoritmos será analisada a seguir.

1. **Tarefa 1 - Encontrar a menor cobertura de vértices:** Esse algoritmo é uma implementação do algoritmo de programação dinâmica, que depende de processar o número de vértices V do grafo de entrada G. Assim, esse algoritmo tem complexidade de tempo $O(V)$. Sua complexidade de espaço é $O(V)$ pois necessita de um espaço auxiliar para armazenar para cada vértice o valor do custo de adicioná-lo ou excluí-lo da solução.
2. **Tarefa 2 - Encontrar uma aproximação para a cobertura de vértices:** Esse algoritmo precisa percorrer o grafo G e verificar para cada vértice se ele e seu vizinho formam um par válido para serem adicionados ao conjunto solução. Assim, esse algoritmo tem complexidade de tempo $O(V + E)$. A complexidade de espaço é $O(V)$, dado que apenas precisa de armazenar os vértices que compõem a solução e os visitados.

5. Avaliação Experimental

Para avaliar a complexidade dos algoritmos, foram medidos os tempos de execução para os algoritmos da tarefa 1 e da tarefa 2. Para cada entrada foram feitas 10 execuções. A máquina utilizada para testes possuía as seguintes configurações:

- Windows 10 Pro 64bits 20H2 19042.804
- Intel i7-4790K 4.00GHz
- 16,0 GB RAM

Foi obtido a média e o desvio padrão como pode ser verificado a seguir:

V	E	1	2	3	4	5	6	7	8	9	10	Média	Desvio Padrão
10	9	34	26	25	23	29	24	24	23	24	34	26,60	4,27
25	24	34	26	24	25	25	24	25	25	29	31	26,80	3,39
50	49	34	27	25	24	26	32	32	25	23	26	27,40	3,84
500	499	38	30	27	27	25	27	25	25	27	28	27,90	3,87

Figura 1. Avaliação experimental para a tarefa 1, tempos em milissegundos

Os testes para o algoritmo da tarefa dois foram criados a partir dos testes fornecidos para a tarefa 1. Sendo que foi adicionado um vértice extra que contém uma aresta para todos os outros vértices. Foi obtido a média e o desvio padrão como pode ser verificado a seguir:

V	E	1	2	3	4	5	6	7	8	9	10	Média	Desvio Padrão	Saída Ótima	Saída Aprox.
11	19	35	29	24	24	26	26	26	25	31	29	27,50	3,50	6	10
26	49	35	26	50	24	24	28	25	25	29	31	29,70	7,94	9	16
51	99	36	29	27	27	28	26	27	29	26	27	28,20	2,94	16	32
501	599	76	60	57	60	60	64	56	59	65	62	61,90	5,69	166	322

Figura 2. Avaliação experimental para a tarefa 2, tempos em milissegundos

Pelos tempos da tarefa 1 pode-se perceber um pequeno aumento para maiores valores de V. Esse aumento é quase imperceptível, pois não depende de E. Portanto, pode verificar o caráter linear do algoritmo desenvolvido. Para o caso da tarefa 2, cuja complexidade depende de V e E, pode-se verificar que o aumento de tempo é significativo entre as execuções para maiores valores de entrada. Esse aumento ocorre de maneira evidentemente linear, o que comprova a complexidade esperada do algoritmo desenvolvido. Além disso, foi comprovado que o algoritmo aproximado gerou saídas de no máximo duas vezes maior que a saída ótima.

6. Considerações finais

A partir do trabalho foi possível entender mais sobre programação dinâmica, algoritmos cobertura de vértices e corretude na prática. Com a solução desenvolvida foi possível encontrar, para a tarefa 1, a cobertura mínima de vértices a fim de atender as vilas agrícolas por meio de programação dinâmica. Para a tarefa 2, foi possível fazer uma aproximação de no máximo 2 vezes a solução ótima para casos em que a malha possui ciclos. Assim pode-se concluir que foi possível cumprir com os objetivos do trabalho prático proposto.

Referências

Jon Kleinberg, E. T. *Algorithm Design*. Pearson Education.

Ziviani, N. *Projetos de Algoritmos com Implementações em Java e C++*. Cengage Learning.