Documentação do Trabalho Prático III da disciplina de Algoritmos I

Cecília Kind - 2019054420

1. Introdução:

Esta documentação se refere ao Trabalho Prático III da disciplina de Algoritmos I. A proposta do trabalho é desenvolver um programa para auxiliar na distribuição de depósitos nas trilhas entre vilas, de forma a cobrir todos os caminhos com o menor número de depósitos possíveis. O trabalho foi implementado em C + + seguindo os princípios da programação orientada a objetos. A solução do problema utilizou os conceitos de programação dinâmica, do problema da cobertura de vértices e da resolução de problemas por meio de grafos. As demais seções presentes neste documento abordam a modelagem do programa (seção 2), as estruturas de dados e algoritmos (seção 3), a complexidade assintótica dos algoritmos (seção 4) e a avaliação experimental (seção 5).

2. Modelagem:

O problema foi modelado por meio de um grafo não direcionado sem pesos em que cada vértice representa uma vila e uma aresta de um vértice P para um vértice Q indica que há uma estrada direta de P para Q.

O problema consiste em determinar o número mínimo de depósitos a serem construídos de forma que cada caminho em uma trilha seja incidente em pelo menos um depósito. No contexto de grafos, o problema consiste em determinar a cobertura de vértices mínima para o grafo que representa a trilha.

A solução do problema da cobertura de vértices pertence ao conjunto NP, que ainda não há solução polinomial disponível. Para a tarefa 1, o problema pode ser solucionado uma vez que a trilha não possui ciclos, podendo ser resolvida recursivamente por meio da programação dinâmica. A solução implementada inicia com a construção de uma árvore a partir do grafo original, para definir uma ordem em que cada vila será avaliada para ter ou não ter um depósito. A ideia consiste em começar a decisão a partir das folhas e subir na árvore de forma que cada vértice depende do resultado dos seus descendentes. Se uma vila é definida sem um depósito, então obrigatoriamente seus descendentes devem ter depósitos. Por outro lado, se uma vila possui um depósito, seus descendentes podem ou não ter um depósito. Os dois casos são avaliados de baixo para cima na árvore buscando sempre o menor número de depósitos possíveis.

Para a segunda tarefa era necessário definir um algoritmo alternativo que permitisse obter uma cobertura de vértices sem extrapolar o resultado ótimo em duas vezes. Como há ciclos no segundo caso, o algoritmo anterior não seria válido, e uma nova estratégia foi abordada. A solução implementada para o problema segue os seguintes passos: uma aresta (u,v) é escolhida aleatoriamente e os vértices u e v são incluídos na solução. Todas as arestas incidentes a u e v são então removidas do grafo e o processo se repete até que não haja mais arestas disponíveis. O algoritmo foi resolvido em tempo polinomial e atendeu ao limite superior imposto. As provas e a complexidade serão abordadas posteriormente.

3. Estruturas de dados e algoritmos

Tarefa 1

- As principais estruturas utilizadas:
  lista de adjacência - vector<int> g[n]: utilizada para armazenar o grafo original
  lista de adjacência - vector<int> tree[n]: utilizada para armazenar o grafo como árvore.
  Vetor auxiliar - visited[n]: utilizados para registrar os nós que foram visitados,
  Vetores auxiliares - inc[n], exc[n]: utilizado para armazenar o valor mínimo de depósitos até aquele ponto no caso em que o depósito é incluído ou excluído na vila em questão.

- Algoritmos principais:

  graphToTree (grafo original, árvore, vértice origem, número de vértices)
  Cria uma fila para armazenar os vértices
  Seleciona um vértice V ainda não visitado e o coloca na fila

  Enquanto a fila não está vazia:
  Remove o primeiro vértice da fila: V
  Para cada vértice W alcançado diretamente de V ainda não visitado:
  Coloca W na fila
  Marca W como visitado
  Adiciona W na árvore como filho de V

  opt (árvore, índice do vértice V, vetoes inc, exc, visited)
  Marca o vértice V como visitado
  Para cada filho de V:
  Chama opt recursivamente

  Se o vértice é uma folha:
  inc[v] = 1: Se inclui um depósito, o número de depósitos é 1
  exc[v] = 0 Se exclui, é 0

  Caso contrário, devemos analisar a inclusão e exclusão separadamente:
  Se incluir: inc[v] = soma do mínimo entre inc e exc de cada um dos filhos de v
  Se excluir: exc[v] = soma de inc para cada um dos filhos de v

  O valor retornado é o menor entre inc[v] e exc[v]

- Corretude do algoritmo:

  Para compreender que o algoritmo é correto podemos analisá-lo passo a passo. Para cada vila, precisamos determinar se ela possui ou não possui um depósito. Se um vértice (vila) não possui descendentes, então não temos restrições. Se decidirmos não ter um posto, então o número de depósitos é 0. Caso contrário, o número é 1. Caso um vértice tenha descendentes, devemos estar atento a duas

condições possíveis: se o depósito é incluído, pode haver ou não depósitos dos descendentes, e a escolha é feita buscando o valor mínimo registrado previamente sobre o custo entre incluir ou não um depósito na vila anterior. Por outro lado, se a escolha é de excluir o depósito, então obrigatoriamente os descendentes devem incluir os depósitos, e o custo também já foi computado. Assim, o resultado mínimo naquele ponto é o mínimo obtido entre as decisões de incluir ou excluir o posto, que é computado corretamente pois constrói as decisões de forma ótima de baixo para cima, garantindo sempre o menor valor necessário até cada ponto da trilha.

Tarefa 2

- As principais estruturas utilizadas:
  lista de adjacência - vector<int> g[n]: utilizada para armazenar o grafo original
  Vetor auxiliar - vector<int> solution_set: armazena os vértices da solução

- Algoritmos principais:

  simpleVertexCover (grafo original, número de vértices)
  cria vetor solução vazio
  cria vetor visited como falso para todas as posições

  Para cada vértice V não visitado:
  Para cada vértice W adjacente não visitado:
  Adiciona V e W na solução
  marca W como visitado
  soma dois no número de vértices da solução final
  marca V como visitado
  Imprime os vértices da solução

- Corretude do algoritmo:

  A lógica seguida para a implementação é a seguinte: uma aresta é selecionada do grafo e os dois vértices que a compõem (u e v) são adicionados ao conjunto solução. Em seguida, todas as arestas incidentes a u ou v são removidas e outra aresta é escolhida, até que não haja mais arestas disponíveis. O algoritmo obtém uma cobertura de vértice pois o loop só termina quando todas as arestas foram cobertas, ou seja, pelo menos um vértice na solução consiste em uma das extremidades da aresta. A restrição de tempo será abordada na próxima seção.

- Prova da aproximação:
  Uma condição para a elaboração da heurística para o problema da cobertura de vértices é que o conjunto solução deveria ser no máximo duas vezes maior do que o resultado ótimo. O algoritmo implementado satisfaz a condição da seguinte forma: Em cada iteração o algoritmo seleciona uma aresta, que juntas compõem um conjunto de tamanho A. Neste conjunto, não há arestas com vértices repetidos, uma vez que após a seleção de uma aresta todas as outras que são incidentes em um dos dois vértices são removidas como opção. Assim, para realizar uma cobertura de vértices que cobre todas as A arestas, são necessários pelo menos A vértices. Assim, a solução ótima O é menor ou igual a A. Agora, comparando com o algoritmo

implementado, temos que para cada uma das A arestas escolhidas acrescentamos 2 vértices à solução, então temos que o resultado O* é igual a 2 x A. Assim, temos que O* = 2A e O >= A, então provamos que O* <= 2O. Na seção da avaliação experimental alguns exemplos serão ilustrados para reforçar esta prova.

4. **Análise de complexidade:**

Tarefa 1:
Seja N o número de vértices e A o número de arestas:

Complexidade de espaço:
Principais estruturas:

- Lista de adjacência para representar o grafo e a árvore: O(N + A)

- Vetores auxiliares: O(N - todos os vetores auxiliares tem tamanho igual ao número de vértices.

Complexidade de tempo:
Principais funções:

- graphToTree: o algoritmo tem complexidade semelhante a uma BFS, que percorre todos os vértices no grafo por camadas, e possui complexidade O(N + A)
- opt: o algoritmo calcula o valor mínimo para cada vértice em tempo constante em função dos valores calculados para os vértices descendentes a ele. Assim, a complexidade é linear: O(N)

Tarefa 2:
Seja N o número de vértices e A o número de arestas:

Complexidade de espaço:
Principais estruturas:

- Lista de adjacência para representar o grafo e a árvore: O(N + A)

- Vetores auxiliares: O(N - todos os vetores auxiliares tem tamanho igual ao número de vértices.

Complexidade de tempo:
Principais funções:

- simpleVertexCover: O algoritmo percorre a lista de adjacência do grafo e realiza operações constantes em cada iteração, sendo assim linear: O(N + A).

5. Avaliação experimental

Tarefa 1

Foram realizados 4 testes diferentes com diferentes configurações. O tempo de execução em segundos para 5 testes, a média e o desvio padrão são exibidos a seguir.

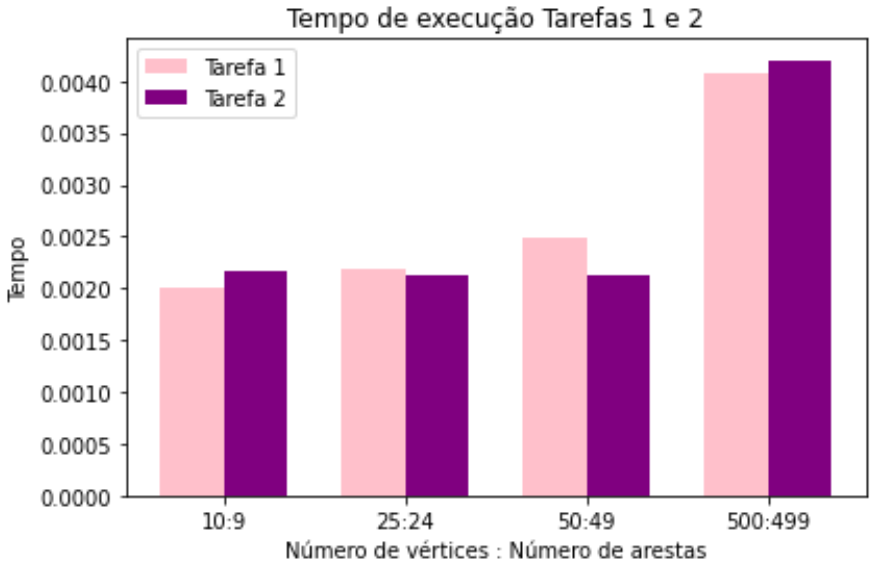
Entrada	Teste 1	Teste 2	Teste 2	Teste 4	Teste 5	Média	Desvio
N = 10 A = 9	0.0019	0.0021	0.002	0.0023	0.0017	0.002	0.000223607
N = 25 A = 24	0.0024	0.0021	0.0019	0.0019	0.0026	0.00218	0.000311448
N = 50 A = 49	0.0022	0.0025	0.0027	0.0023	0.0027	0.00248	0.000228035
N = 500 A = 499	0.0037	0.0040	0.0044	0.0038	0.0045	0.00408	0.000356371

Tarefa 2:

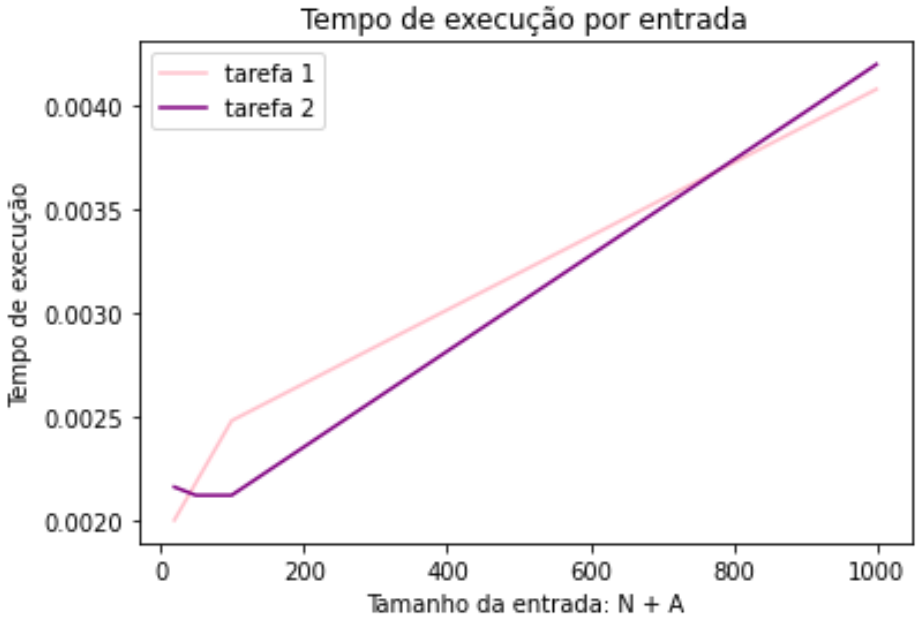
Os mesmos testes utilizados anteriormente foram utilizados na primeira parte. Os resultados são mostrados a seguir.

Entrada	Teste 1	Teste 2	Teste 2	Teste 4	Teste 5	Média	Desvio
N = 10 A = 9	0.0019	0.0021	0.0025	0.0020	0.0023	0.00216	0.000240832
N = 25 A = 24	0.0025	0.0021	0.0020	0.0021	0.0019	0.00212	0.000228035
N = 50 A = 49	0.0019	0.0021	0.0026	0.0019	0.0021	0.00212	0.000286356
N = 500 A = 499	0.0037	0.0041	0.004	0.0041	0.0052	0.0042	0.000571839

O gráfico a seguir apresenta uma comparação entre o tempo médio nos 4 testes para as duas tarefas. Como foi demonstrado na seção anterior, os dois algoritmos foram implementados em tempo linear, o que justifica os resultados próximos para o tempo de execução nas duas tarefas. Nenhum dos casos ultrapassou o limite de dois segundos.



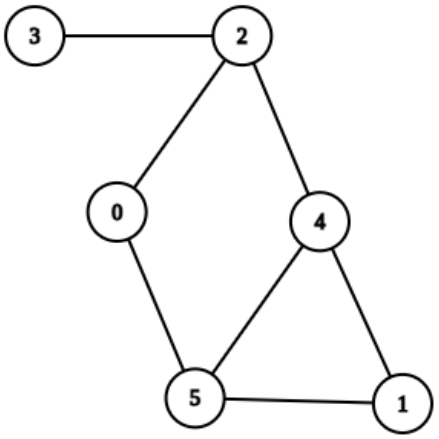
A seguir, os resultados são apresentados em um outro grafo para as duas tarefas. Mesmo com o aumento da entrada o tempo de execução cresce pouco, e novamente confirma-se a complexidade linear dos algoritmos.



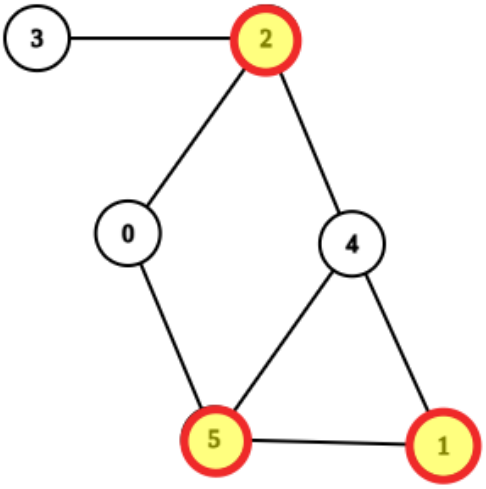
Testes para a tarefa 2

Como foi dito anteriormente, era exigido que o algoritmo implementado para a tarefa 2 não tivesse resultado pior que 2x o resultado ótimo. A prova já foi demonstrada na seção 3, mas alguns exemplos serão ilustrados para demonstrar que o algoritmo não extrapola o limite superior.

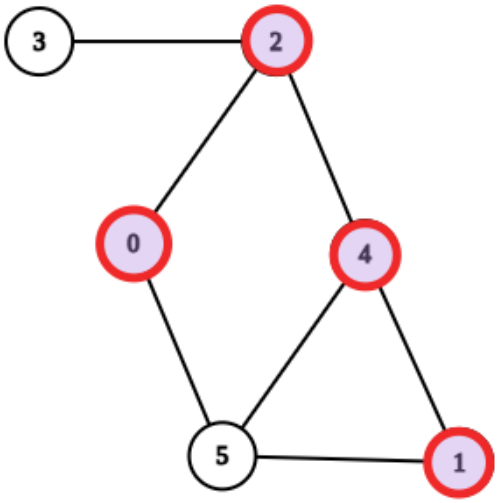
Exemplo 1: Grafo original



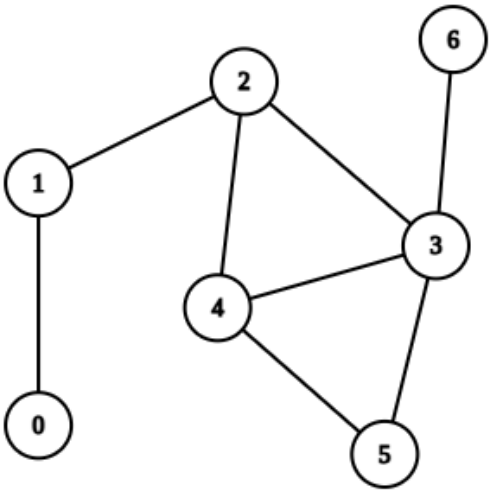
Solução ótima: Tamanho 3



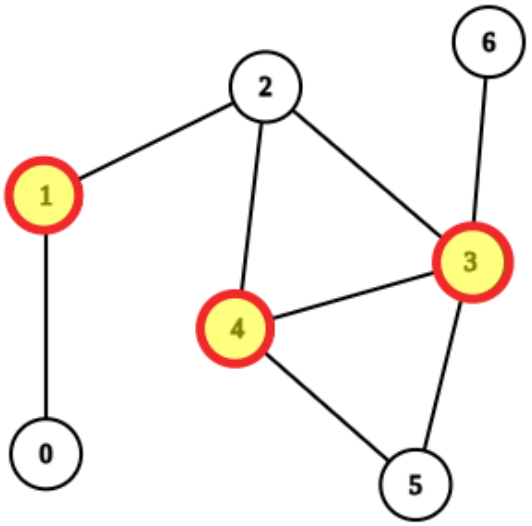
Solução do algoritmo: tamanho 4



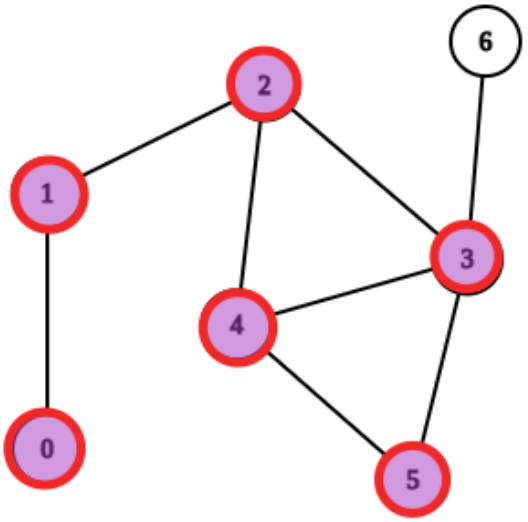
Exemplo 2: Grafo original



Solução ótima: Tamanho 3



Solução do algoritmo: Tamanho 6



6. Conclusão:

Este trabalho teve como base os conceitos de programação dinâmica, cobertura de vértice e problemas em NP. A partir destes conceitos e dos conhecimentos de grafos e árvores foi possível desenvolver uma solução simples para o problema de distribuição de depósitos proposto. Foi provado que a heurística implementada correspondeu aos critérios estabelecidos e os exemplos demonstrados reforçaram os resultados. A complexidade de tempo das duas tarefas é O(N + A) e de espaço O(N + A).