Lista 2

Projeto e Análise de Algoritmos (INE410104) – 22 set 2020
Aluno: Matheus Cardoso de Almeida

Questão 1

a) Algoritmo

O código fonte para a implementação está no arquivo ***l2q1.py***, junto dos demais anexos. O algoritmo tomou como base a Busca em Largura para grafos (página 87, Algoritmo 19). Para isso, considerou-se a seguinte abstração:

- Grupo com identificadores dos centros de distribuição:
  - $C = \{c_1, c_2, ..., c_n\}$
  - Ex.: `distribution_centers = ['A', 'B', 'C', 'D', 'E']`
- Grupo com pares de centros de distribuição que trocam encomendas:
  - $L = \{\{c_a, c_b\}, \{c_c, c_d\}, ...\}$
  - Ex.:
```
relations = [['A', 'B'], ['C', 'E'],
             ['A', 'D'], ['B', 'E']]
```
- Sistema de distribuição:
  - Grafo não-direcionado e não-ponderado composto por vértices C e arestas L.
  - $G = \{C, L\}$
  - Foi representado na forma de lista de adjacências (na forma de um dict).
  - Ex.:
```
adjacency_list = {'A': ['B', 'D'],
                  'B': ['A', 'E'],
                  'C': ['E'],
                  'D': ['A'],
                  'E': ['C', 'B']}
```

b) Análise de complexidade

Considerando a abstração de grafo utilizada e a implementação com base na Busca em Largura, tem-se a complexidade dada por $O(|V|+|E|)$. As parcelas de código adicionados não alteram esse valor, uma vez que afetam o algoritmo com custos menores, ou iguais, a $O(|V|+|E|)$.

- A função `create_adjacency_list` itera sobre $|V|$ e $|E|$, assim também assumindo uma complexidade de $O(|V|+|E|)$.
- As etapas de configuração dos vértices itera sobre $|V|$, assim, assumindo uma complexidade de $O(|V|)$.

- As etapas de configuração do vértice de origem e de preparação da fila de visitas possuem custos constantes $O(C)$.
- A propagação das visitas é limitado por $|V|$ mais a varredura de vizinhos não visitados (limitado por $|E|$), conforme explanado no tópico 5.3.1.1 da apostila. Dessa forma, assume-se uma complexidade de $O(|V|+|E|)$.
- A etapa de construção do caminho que leva da origem ao destino varre o vetor de ancestrais, sendo limitado a $|V|$, portanto, $O(|V|)$.

Questão 2

a) Algoritmo

O código fonte para a implementação está no arquivo ***l2q2.py***, junto dos demais anexos. O código tomou como base o algoritmo de Dijkstra (página 121, Algoritmo 33). Para isso, considerou-se a seguinte abstração:

- Grupo com identificadores de locais/cidades:
  - $V = \{v_1, v_2, ..., v_n\}$
  - Ex.: `cities = ['A', 'B', 'C', 'D', 'E']`
- Grupo com pares representando as estradas entre locais/cidades:
  - $A = \{(v_1, v_2), (v_3, v_4), ...\}$
  - Ex.:
```
roads = [('A', 'B'), ('B', 'C'),
         ('C', 'D'), ('C', 'E'),
         ('B', 'E')]
```
- Função para pesos/custo das estradas:
  - $w = d(v).c/a + p(v)$
  - $d(v)$: função de distâncias para estradas (km).
  - c: preço do combustível ($R\$/l$).
  - a: autonomia de combustível do veículo (km/l).
  - $p(v)$: função de pedágio para cidades de destino ($R\$$).
  - Representado pela função `cost`.
- Sistema de distribuição:
  - Grafo direcionado e ponderado composto por vértices V e arcos A e pesos definidos por w.
  - $G = \{V, A, w\}$
  - Foi representado na forma de lista de adjacências (na forma de um dict).
  - Ex.:
```
adjacency_list = {'A': [(1.0, 'B')], 'B': [(1.0, 'C'),
                                           (10.0, 'E')], 'C': [(1.0, 'D'),
                                           (1.0, 'E')], 'D': [], 'E': []}
```

b) Justificativa de corretude após parada

Tendo em mente que a implementação em questão tem em sua base o algoritmo de Dijkstra (linhas de 75 a 101 do código), assume-se o Corolário 6.1.4 (apresentado na apostila).

- **Corolário 6.1.4.** Ao executar algoritmo de Dijkstra (Algoritmo 33) sobre o grafo $G = (V, E, w)$ ponderado orientado ou não, sem ciclos de peso negativo, para o vértice de origem $s \in V$, o subgrafo dos predecessores $G\pi$ será uma árvore de caminhos mínimos em s.

Dessa forma, tem-se:

- como peso, os valores calculados pela função `cost`.
- como condição de parada do algoritmo, o momento em que a árvore visita efetivamente o vértice de destino, assim definindo o menor percurso partindo da origem.

Faz-se uso do algoritmo de Dijkstra para definir o menor caminho até o vértice de destino e, então, usa-se os os antecessores de cada vértice para definir a sequência de cidades até a origem.

Portanto, aproveita-se da justificativa de corretude do algoritmo de Dijkstra para justificar os resultados do algoritmo implementado.

Questão 3

a) Algoritmo

O código fonte para a implementação está no arquivo ***l2q3.py***, junto dos demais anexos. O código tomou como base a solução em programação dinâmica para o problema da mochila (seção 7.2). Para isso, considerou-se a seguinte abstração:

- Grupo $C = \{c_1, c_2, ..., c_n\}$ indicando a capacidade para cada um dos caminhões.
- Grupo $W = \{w1, w_2, ..., w_n\}$ indicando os pesos de cada item.
- Grupo $V = \{v_1, v_2, ..., v_n\}$ indicando os valores de cada produto.

b) Justificativa

Similar à lógica aplicada na solução do problema da Mochila, iteira-se sobre cada um dos itens e verifica-se em quais caminhões ele poderia ser adicionado, calculando um resultado parcial de valor (que é passado adiante) e deduzindo o peso do item de sua capacidade. O algoritmo pára quando todos os itens tenham sido avaliados, ou não haja mais capacidade disponível em nenhum dos caminhões.

A lógica em questão pode ser resumida na forma da seguinte função OPT:

- Para todo caminhão t cujo a capacidade (c_t) atual suporta o peso do item em análise ($w(i)$):
    - $parcial = v(i) \ + \ OPT(i-1, \ c_1, ..., c_t - w(i), ..., c_n)$
- A manutenção do valor anterior:
    - $parcial = OPT(i-1, \ c1, ..., cj, ..., cn)$
- Por fim, o valor resultante de OPT é dado pelo valor máximo do vetor composto por todas as parciais.

O resultado calculado pela função OPT representa o máximo valor de produto alcançado para a organização definida para as entradas. Dessa forma, subtrai-se isso da soma dos valores de todos os itens, assim obtendo apenas o valor dos produtos que não foram entregues.

Questão 4

a) O problema do Caixeiro Viajante de decisão é um problema considerado NP-Completo, pois pode ser verificado em tempo polinomial e resolve outro problema NP-Completo através da transformação polinomial. O que aconteceria se alguém descobrisse um algoritmo resolvido em tempo polinomial para este problema?

Caso esse algoritmo seja determinístico, sua existência representaria a equivalência das classes P e NP, portanto, todos os demais problemas de tipo NP também poderiam ser resolvidos por meio de um processo de transformação polinomial. Isso acontece devido a classe NP-Completo ser um sub-grupo restrito e capaz de resolver toda a classe NP.

Caso esse algoritmo seja não-determinístico, nada mudaria, pois apenas reafirma o problema como pertencente à classe NP.

b) Para demonstrar que um problema é difícil de ser resolvido, muitos pesquisadores utilizam o problema em questão para resolver um NP-Difícil através da transformação polinomial. Por quê?

É uma maneira de demonstrar que o problema em questão, de alguma forma, pertence à classe NP-Difícil. Ou seja, não se conhece uma resolução polinomial determinística (em alguns casos, até mesmo não-determinística) para o problema, além de ser capaz de resolver todos os demais problemas da classe NP.

c) Qual a diferença entre verificação ou resolução em tempo polinomial?

A expressão “resolução em tempo polinomial” faz referência ao processo de determinação de uma resposta SIM ou NÃO para uma instância de problema de decisão, tendo como base um algoritmo de tempo polinomial.

Por outro lado, a expressão “verificação em tempo polinomial” trata de validar/corrigir um único certificado/prova associada à instância do problema, tendo como base um algoritmo em tempo polinomial.

d) Defina o que são algoritmos de aproximação e heurísticas?

Ambos tratam de resoluções que não garantem o resultado ótimo para uma determinada instância de problema. No entanto, enquanto algoritmos aproximados possuem uma garantia de aproximação de seus resultados em relação às soluções ótimas (com base em uma razão que descreva essa proximidade), heurísticas não apresentam nenhuma garantia formal para suas soluções e usam características do problema para alcançar resultados estatisticamente aceitáveis.