Resolução – Lista 5 (Projeto e Análise de Algoritmos)

Fevereiro - 2023 / Leticia Bossatto Marchezi – 791003

Questão 1

Mostre que todo subcaminho de um caminho ótimo é sempre ótimo, ou seja, que o problema do caminho mínimo tem subestrutura ótima. Porque isso é importante?

Resolução:

Uma solução ótima para o caminho mínimo é uma solução com valor mínimo da soma dos pesos do caminho total.

Tomando um grafo com caminho mínimo ótimo P composto por 3 subcaminhos P_{ab}, $P_{b,c}$ e $P_{c,d}$. Ou seja, isso implica que a soma dos pesos

$$w(P) = w(P'_{a,b}) + w(P_{b,c}) + w(P_{c,d}) \tag{1.1}$$

é o mínimo entre os caminhos possíveis.

Supondo que haja um caminho $P'_{a,b}$ tal que $w(P'_{a,b}) < w(P_{a,b})$. Por consequência, o caminho mínimo é na verdade com caminho $P'_{a,b}$ de tal forma que o peso total é minimizado para um caminho P':

$$w(P) = w(P'_{a,b}) + w(P_{b,c}) + w(P_{c,d}) \tag{1.2}$$

Dessa forma, contradiz-se que a solução mínima é P, já que a somatória dos pesos de P tem valor superior ao caminho P'. Assim, um caminho ótimo precisa ter subcaminhos ótimos.

Questão 2

Explique como o algoritmo de Dijkstra utiliza programação dinâmica para resolver o problema de encontrar caminhos mínimos em grafos

Resolução:

O algoritmo de Dijkstra pode ser utilizado para resolver o problema de menores caminhos em grafos utilizando programação dinâmica por três fatores:

- O algoritmo não recalcula os caminhos mínimos em toda iteração, esses valores são armazenados em uma fila de prioridades.
- O algoritmo constrói a solução ótima a partir das soluções dos subproblemas que são menores.
- Utilizando a estratégia Bottom-Up é possível realizar a reversão de recursão.

Todos esses fatores possibilitam a implementação do algoritmo no paradigma da programação dinãmica.

Questão 3

Mostre o passo a passo do algoritmo de Dijkstra para encontrar a árvore de caminhos mínimos no grafo a seguir. Mostre a fila Q em cada iteração, bem como a ordem de acesso aos vértices e a atualização dos valores de $\lambda(v)$.

Lembre-se que G é um dígrafo, ou seja, A é vizinho de B, mas B não é vizinho de A (por causa da aresta direcionada).

Resolução:

Solução usando métodos de iteração abaixo seguindo Fila, Ordem de acesso e Mapa de predecessores.

Fila de prioridade

	s	a	b	c	d	t
λ^0(v)	0	∞	∞	∞	∞	∞
λ^1(v)		7	**4**	9	7	∞
λ^2(v)		**5**		7	7	∞
λ^3(v)				**7**	7	11
λ^4(v)					**7**	10
λ^5(v)						**10**

Figure 1: Fila de prioridade

Ordem de acesso aos vértices

u	V'	λ(v)	π(v)
s	{a,b,c,d}	λ(a) = min(λ(a),λ(s) + w(s,a)) = min {∞, 7} = 7	π(a) = s
		λ(b) = min(λ(b),λ(s) + w(s,b)) = min {∞, 4} = 4	π(b) = s
		λ(c) = min(λ(c),λ(s) + w(s,c)) = min {∞, 9} = 9	π(c) = s
		λ(d) = min(λ(d),λ(s) + w(s,d)) = min {∞, 7} = 7	π(d) = s
b	{a,c}	λ(a) = min(λ(a),λ(b) + w(b,a)) = min {7, 5} = 5	π(a) = b
		λ(c) = min(λ(c),λ(b) + w(b,c)) = min {9, 7} = 7	π(c) = b
a	{t}	λ(t) = min(λ(t),λ(a) + w(a,t)) = min {∞, 11} = 11	π(t) = a
c	{t}	λ(t) = min(λ(t),λ(c) + w(c,t)) = min {11, 10} = 10	π(t) = c
d	{c,t}	λ(c) = min(λ(c),λ(d) + w(d,c)) = min {7, 8} = 7	--
		λ(t) = min(λ(t),λ(d) + w(d,t)) = min {10, 11} = 10	--
t	{}	--	--

Figure 2: Ordem de acesso aos vértices

Mapa de predecessores

v	s	a	b	c	d	t
π(v)	--	b	s	b	s	c
λ(v)	0	5	4	7	7	10

Figure 3: Mapa de predecessores

Questão 4

Analise a complexidade do algoritmo de Dijkstra com estruturas de dados estáticas

Resolução:

Para uma estrutura de dados estática temos um vetor de estático de N elementos e uma matriz de adjacências. Com isso pode-se construir o grafo G = (V,E).

Seguindo a iteração explicada no exercício 3 acima e no livro texto disponibilizado pelo professor, temos que:

- Inicialização dos $\lambda(v) = \mathcal{O}(n)$.
- Inserção dos vértices na fila é $\mathcal{O}(n)$.
- Loop While executado n vezes.

- A função ExtractMin tem custo $\mathcal{O}(n)$.
- for loop para atualizar os valores de $\lambda(v) = \mathcal{O}(n)$.

Sendo assim, com os custos $\mathcal{O}()$ acima conseguimos calcular a complexidade do algoritmo.

$$T(n) = \mathcal{O}(n) + \mathcal{O}(n^2) + \mathcal{O}(1) * \mathcal{O}(m) \tag{4.1}$$

Dessa maneira, percebemos que para uma estrutura de dados estáticas a complexidade do algoritmo é $\mathcal{O}(n)$.

Questão 5

Analise a complexidade do algoritmo de Dijkstra com estruturas de dados dinâmicas.

Resolução:

Para uma estrutura de dados dinâmica temos uma fila de prioridades Q representada por um heap dinâmico, assim temos o grafo G = (V,E).

Com isso, os passos para o algoritmo ocorrer são:

- Inicialização do $\lambda(n) = \mathcal{O}(n)$.
- Inserção dos vértices na fila Q = $\mathcal{O}(\log n)$
- Loop while executado N vezes $\mathcal{O}(n)$.
- Função ExtractMin $\mathcal{O}(\log n)$.
- Atualização do valor de $\lambda(v) = \mathcal{O}(1)$ executado $d(u)$ vezes.
- Decrease é $\mathcal{O}(\log n)$

Assim, temos que custos $\mathcal{O}$ possibilitando calcular a complexidade do algoritmo:

$$T(n) = \mathcal{O}(n) + \mathcal{O}(\log n) + \mathcal{O}(n \log n) + \mathcal{O}(m \log n) \Rightarrow \tag{5.1}$$

$$T(n) = \mathcal{O}((n + m) \log n)). \tag{5.2}$$

Como em grafos conexos $m > n - 1$ temos que a complexidade do algoritmo é igual a $\mathcal{O}(m \log n)$.

Questão 6

Demonstre que o algoritmo de Dijkstra é ótimo, ou seja, sempre retorna os caminhos mínimos em grafos com pesos não negativos.

Resolução:

O algoritmo de Dijkstra se baseia no teorema em que $\lambda(v) = d(s, v)$ sendo d a menor distância possível de s até v. Adicionando um vértice a tal que $\lambda(a) \neq d(s, a)$.

Deve existir um caminho P_{sa}, caso contrário $\lambda(a) = \infty$. Logo, existe um caminho mínimo P^*_{sa}.

Assumindo que há um vértice y, sendo o primeiro vértice de P^*_{sa} e seja x seu predecessor.

Assim, $\lambda(y) = d(s, x) + w(x, y) = d(s, y)$. Como y antecede a, então vale $d(s, y) \leq d(s, a)$. Mas, por ter

escolhido a para entrar em S, então $\lambda(a) \leq \lambda(y)$. Logo, $\lambda(a) = \lambda(y)$.

Porém, chegamos na contradição

$$\lambda(y) = d(s, y) = d(s, a) = \lambda(a) \tag{6.1}$$

Assim, o algoritmo mantém sua corretude quando suas propriedades são mantidas.

Questão 7

Explique como o algoritmo Floyd-Warshall utiliza programação dinâmica para encontrar caminhos mínimos entre todos os pares de vértices em um grafo e porque essa estratégia funciona.

Resolução:

O algoritmo de Floyd-Marshall utiliza programação dinâmica ao construir uma matriz de distância entre vértices que são atualizadas a cada iteração do algoritmo, buscando os melhores caminhos armazenando os cálculos das distâncias mínimas e os predecessores dos vértices, assim evitando a necessidade de recalcular trajetórias inteiras a partir de um ponto, já que para novas iterações são comparados o custo passado do caminho com o custo da nova aresta mais o percurso até seu vértice.

Questão 8

Dada a matriz de custos a seguir, implemente o algoritmo Floyd-Warshall para obter as distâncias mínimas entre cada par de vértices (pode ser em qualquer linguagem de programação).

Resolução:

A função FloydWarshall implementa o algoritmo para cálculo de distância mínima entre vértices e o resultado pode ser conferido no output:

```python
graph = [[0, 2, float('inf'), 1, 8],
    [6, 0, 3, 2, float('inf')],
    [float('inf'),float('inf'), 0, 4, float('inf')],
    [float('inf'), float('inf'), 2, 0, 3],
    [3, float('inf'),float('inf'), float('inf'), 0]]

def FloydWarshall(W):
    n = len(W)
    D=[[0]*n]*n
    D = W
    for k in range(n):
      for i in range(n):
        for j in range(n):
          D[i][j] = min(D[i][j], D[i][k]+D[k][j])
    return D

print("Matriz de distância mínima entre os vértices:")
FloydWarshall(graph)
```

```
Matriz de distância mínima entre os vértices:
[[0, 2, 3, 1, 4],
 [6, 0, 3, 2, 5],
 [10, 12, 0, 4, 7],
 [6, 8, 2, 0, 3],
 [3, 5, 6, 4, 0]]
```

Figure 4: Algoritmo em Python para algoritmo Floyd Warshall