UNIFEI - UNIVERSIDADE FEDERAL DE ITAJUBÁ

CURSO DE GRADUAÇÃO EM SISTEMAS DE INFORMAÇÃO

SIN110 - ALGORITMOS E GRAFOS

RESOLUÇÃO DOS EXERCÍCIOS E07 DO DIA 02/10/2015

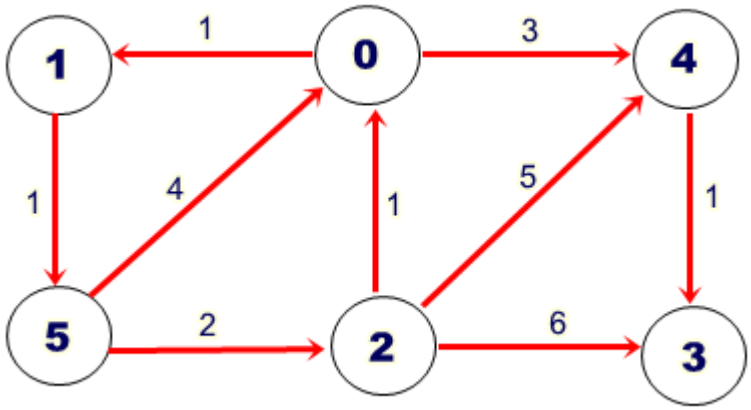
ITAJUBÁ

2015

Exercícios E07 – 02/10/15

Aluna: Karen Dantas **Número de matrícula:** 31243

1) Dígrafo:

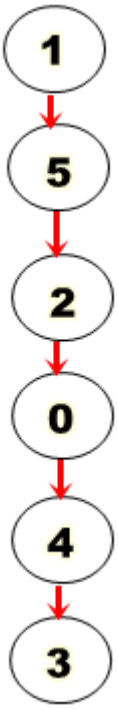


Algoritmo de Dijkstra com início no vértice 1:

Filas Q: 1 – 0 - 2 - 3 - 4 - 5
5 - 0 - 2 - 3 - 4
2 - 0 - 3 - 4
0 - 4 - 3
4 - 3
3

Vértice	Predecessores	d[v]
0	~~5~~ / 2	~~∞~~ / ~~5~~ / 4
1	-	0
2	5	~~∞~~ / 3
3	~~2~~ / 4	~~∞~~ / ~~9~~ / 8
4	~~2~~ / 0	~~∞~~ / ~~8~~ / 7
5	1	~~∞~~ / 1

Árvore de caminhos mínimos com origem 1:



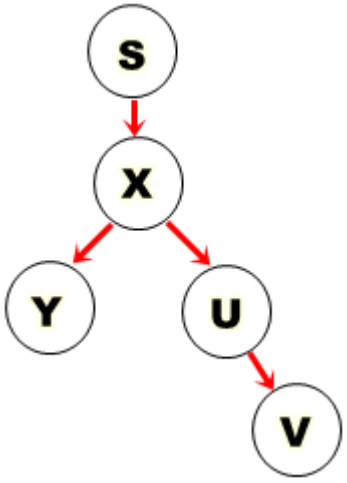
---

2) a) Algoritmo de Dijkstra com início no vértice S:

Filas Q: S - U – X – V – Y
X – U - V – Y
Y - U - V
U - V
V

Vértice	Predecessores	d[]
S	-	0
U	~~S~~ / X	~~∞~~ / ~~10~~ / 8
X	S	~~∞~~ / 5
V	~~X~~ / ~~Y~~ / U	~~∞~~ / ~~14~~ /~~13~~ / 9
Y	X	~~∞~~ / 7

Árvore de caminhos mínimos com origem S:



b) O peso da aresta (X, U) passa a ser -3.

Algoritmo de Bellmann-Ford com início no vértice S:

A tabela abaixo possui todos os arcos presentes no dígrafo e seus respectivos pesos e também representa a ordem em que o dígrafo será percorrido.

Arco (u, v)	Peso
S-U	10
S-X	5
U-X	2
U-V	1
X-U	-3
X-V	9
X-Y	2
Y-S	7
Y-V	6
V-Y	4

1ª iteração		
Vértice	**Predecessores**	**d[v]**
S	-	0
U	~~S~~ / X	~~∞~~ / ~~10~~ / 2
X	S	~~∞~~ / 5
V	U	~~∞~~ / 11
Y	X	~~∞~~ / 7

2ª iteração		
Vértice	**Predecessores**	**d[v]**
S	-	0
U	X	~~2~~ / 1
X	~~S~~ / U	~~5~~ / 4
V	U	~~11~~ / 3
Y	X	~~7~~ / 6

3ª iteração		
Vértice	**Predecessores**	**d[v]**
S	-	0
U	X	~~1~~ / 0
X	U	~~4~~ / 3
V	U	~~3~~ / 2
Y	X	~~6~~ / 5

4ª iteração		
Vértice	**Predecessores**	**d[v]**
S	-	0
U	X	~~0~~ / -1
X	U	~~3~~ / 2
V	U	~~2~~ / 1
Y	X	~~5~~ / 4

Verifica existência de ciclo negativo			
Arco (u, v)	**Peso**	**d[v] > d[u] + w(u, v)**	**Resposta**
S-U	**10**	-1 > 10	Não
S-X	**5**	2 > 5	Não
U-X	**2**	2 > 1	Sim
U-V	**1**	1 > 0	Sim
X-U	**-3**	-1 > -1	Não
X-V	**9**	1 > 11	Não
X-Y	**2**	4 > 4	Não
Y-S	**7**	0 > 11	Não
Y-V	**6**	1 >10	Não
V-Y	**4**	4 > 5	Não

A partir da análise da tabela acima verificou-se que foram encontrados dois ciclos negativos nos arcos (U, X) e (U, V). Vale ressaltar, que o algoritmo ao identificar o primeiro ciclo negativo em (U, X) ele terminaria sua execução, mas continuei a verificação para ver em quais outros arcos seriam identificados outros ciclos negativos caso a execução continuasse.

Como foi identificado ciclo negativo, quer dizer que não há solução e, consequentemente, não é possível produzir a árvore de caminhos mínimos.

3) Algoritmo:

```
RotaTotal_MenorCusto (n, m, c, k, R)
1.    G ← ConstroiDigrafo (R, 3, m, n)
2.    IniciaOrigemÚnica (G, 0)
3.    Q ← Cria-Fila(c)
4.    enquanto Q ≠ Ø faça
5.          u ← Retire-Minimo (Q)
6.          para cada v em Adj[u] faça
7.                Relaxa (u, v, w)
8.    devolve MenorCusto_CidadeConserto (pred, c, k, G)

MenorCusto_CidadeConserto (pred, c, k, G)
1.    total ← 100000
2.    para cada v em Adj[k] faça
3.          se total < d[v] + w (v, k)
4.                total ← d[v] + w (v, k)
5.                vert_custo_menor ← v
6.    encontrou ← -1
7.    para i ← 0 até c-1 faça
8.          se vert_custo_menor = pred [i]
9.                encontrou ← 1
10.               custo ← Calcula_CustoRota (vert_custo_menor, G)
11.   se encontrou =1
12.         devolve custo
13.   Relaxa (k, vert_custo_menor, w)
14.   devolve MenorCusto_CidadeConserto (pred, c, vert_custo_menor, G)
```

Os algoritmos acima funcionam da seguinte forma: Primeiramente, é executado o algoritmo RotaTotal_MenorCusto no qual, na linha 1, é chamada a função ConstroiDigrafo que transforma a matriz R em um dígrafo que possui todas as rotas possíveis e seus custos quais são representados pelos pesos dos arcos entre os vértices. O número 3 que é passado para essa função se refere ao número de colunas da matriz. Na linha 2, é chamada a função IniciaOrigemÚnica que cria a origem da rota e é passado para o ela o dígrafo gerado e o vértice de origem que é 0. Na linha 3, é criada uma fila com as cidades na rota de serviço. Nas linhas 4, 5, 6, 7, é descoberto o caminho que percorre todas as cidades e com menor custo definindo assim, a rota. Na linha 8, é retornado o resultado da função MenorCusto_CidadeConserto.

Na função MenorCusto_CidadeConserto, na linha 1 a variável ‘total’ é iniciada com um número muito grande para quando ela for ser utilizada, seu valor não interfira na descoberta do menor peso. Na linha 2, é realizado um *loop* que irá percorrer todos os vértices adjacentes ao vértice ‘k’ que representa a cidade (vértice) onde o veículo foi consertado. Nesse *loop* é buscado um vértice adjacente ‘vert_custo_menor’ o qual, seu peso somado ao peso da aresta (k, v), possui menor total

em relação aos outros vértices adjacentes de ‘k’. Na linha 7, é realizado outro *loop* no qual irá ser percorrido o vetor de predecessores ‘pred’, que foi gerado na função RotaTotal_MenorCusto, e irá ser verificado se o vértice ‘vert_custo_menor’ está presente nesse vetor. Se estiver, quer dizer que a cidade onde o veículo foi consertado pertence a rota, assim, é chamada a função Calcula_CustoRota a qual calculará o custo gerado pelo percurso a partir do vértice ‘vert_custo_menor’ até o destino.

Caso contrário, se o vértice ‘vert_custo_menor’ não estiver presente em ‘pred’ significa que a cidade onde o veículo foi consertado está fora da rota. Assim, é chamada a função Relaxa para atualizar o peso do vértice ‘k’. Logo após, a função MenorCusto_CidadeConserto é chamada recursivamente passando o ‘vert_custo_menor’ como parâmetro e para ele será verificado o menor caminho para seus adjacentes e se o adjacente encontrado faz parte da rota e, assim, sucessivamente.