MINISTÉRIO DA CIÊNCIA E TECNOLOGIA

LABORATÓRIO ASSOCIADO DE COMPUTAÇÃO E MATEMÁTICA APLICADA

ESTUDO E IMPLEMENTAÇÃO DE ALGORITMOS PARA RESOLUÇÃO DE PROBLEMAS COMBINATÓRIOS EM GRAFOS

Ralphy Antonio Martin Castilho

Bolsista PIBIC

Orientador: Dr. Horacio Hideki Yanasse

Relatório Final

SÃO JOSÉ DOS CAMPOS, SP - BRASIL
FEVEREIRO DE 2001

SUMÁRIO

ÍNDICE DE FIGURAS

ÍNDICE DE TABELAS

LISTA DE ABREVIATURAS

PRV	Problemas de Roteamentos de Veículos
PT	Problema dos Transportes
PCC	Problema do Carteiro Chinês
SIG	Sistemas de Informações Geográficas
ESRI	Enyironmental Systems Research Institure

1. Introdução

O trabalho em si é um estudo sobre grafos, suas propriedades e aplicações. A teoria dos grafos pode apresentar algumas desvantagens, em que algoritmos motivados por teoria dos grafos são considerados geralmente muito mais eficientes do que os tradicionais programas matemáticos (no caso do Método Simplex), porém, problemas grandes podem apresentar dificuldades nas programações dos algoritmos; o fator de maior importância na teoria dos grafos é seu contexto em resolver problemas de redes e problemas de procedimento simples; para aplicações em sistemas de serviços urbanos, em que de acordo com a teoria dos grafos, o percurso será transformado em rede.

Para a realização deste trabalho, inicialmente foram estudados diversos problemas combinatórios conhecidos para aplicação em grafos e utilizados na otimização de problemas de roteamentos de veículos (PRV).

Vários PRV já foram estudados e algoritmos especializados para sua solução estão disponíveis na literatura. Então foram escolhidos os problemas principais os quais foram estudados detalhadamente para posterior implementação.

Os algoritmos em questão consideram apenas espaços urbanos bidimensionais não totalmente contínuos, mas normalmente muito discretizados. Utiliza-se como estratégia analisar os efeitos computacionais causados por distúrbios em termos contínuos.

Em seguida, para a implementação, foi preciso um estudo mais elaborado em linguagem de programação C/C++, com ênfase em orientação a objetos e alocação dinâmica de memória.

Foi procurado obter conhecimento sobre otimização de algoritmos, pois como deseja-se aplicar manipulação de grandes matrizes é de primeira necessidade códigos que trabalhem com alocação dinâmica de memória.

Outros algoritmos para resolução destes problemas têm sido propostos ao longo dos anos por estudiosos e pesquisadores especializados nesta área, porém, todos estes algoritmos aqui estudados são de grande praticidade e de resolução simples.

Pode-se notar, portanto, que alguns algoritmos apresentados não são necessariamente os mais eficientes, falando-se em termos computacionais.

1. Definição de Termos e Notações

O termo grafo (ou rede) é usado para referenciar uma entidade G(N, A) que consiste de um conjunto finito, **N**, de nós (ou vértices) e um conjunto finito, **A**, de arcos (ou arestas) que conecta os pares de nós. Um arco que conecta os nós $i \in N$ e $j \in N$ é denominado como (i, j).

Um arco (i, j) é chamado de arco orientado quando possui um sentido (direção) que vai de i para j. Um grafo que possui todos os seus arcos orientados é denominado de grafo orientado, se não possui nenhum arco orientado é denominado de grafo não orientado e se possui ambos os tipos de arcos é denominado de grafo misto.

Um caminho é a seqüência de nós e arcos adjacentes, em que o nó final de um arco é o nó inicial do próximo arco, respeitando a orientação dos arcos caso houver. Um caminho é simples quando cada arco é percorrido uma única vez na seqüência e o caminho é elementar quando cada nó é percorrido uma única vez na seqüência. Um ciclo (ou circuito) é um caminho em que o nó inicial e o final são coincidentes (os mesmos).

A figura 2.1 ilustra um grafo dado pela seqüência de nós {A,B,C,D,E,F,,G,H}, e pelo conjunto de arcos {(A,B), (A,D), (A,C), (C,D), (C,F), (E,G), (A,A)}.

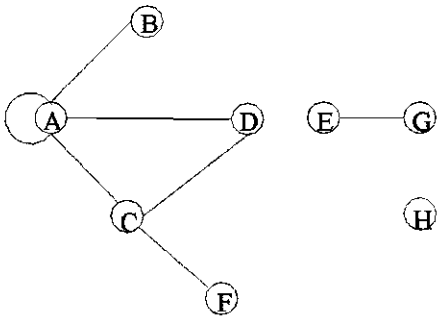

Figura 2.1 – Seqüência de nós e arcos de um grafo

Se os pares de nós que formam os arcos forem pares ordenados, diz-se que o grafo é um grafo orientado (ou dígrafo). As Figuras 2.2-a, 2.2-b e 2.2-c ilustram três dígrafos. As setas entre nós representam arcos. A ponta de cada seta representa o segundo nó do par ordenado de nós que forma um arco, e o final de cada seta

representa o primeiro nó do par ordenado. O conjunto de arcos do grafo da figura 2.2-a é {(A,B), (A,C), (A,D), (C,D), (F,C), (E,G), (A,A)}. Usando parênteses para indicar um par ordenado.

Uma árvore é um subgrafo com todos os nós conectados aos arcos e que não possui ciclos. Observar que um grafo não precisa ser uma árvore (Figuras 2.1, 2.2-a e 2.2-b), mas uma árvore tem de ser um grafo (Figura 2.2-c). Observar também que um nó não precisa ter arcos associados a ele (nó H nas Figuras 2.1 e 2.2-a).

Um nó n incide em um arco x se n for um de seus dois nós no par ordenado de nós que constituem x. (Diz-se também que x incide em n). O grau de um nó é o número de arcos incidentes nesse nó. O grau de entrada de um nó n é o número de arcos que tem n como cabeça, e o grau de saída de n é o número de arcos que tem n como terminação da seta. Por exemplo, o nó A na Figura 2.1.2-c tem grau de entrada 1, grau de saída 2 e grau 3.

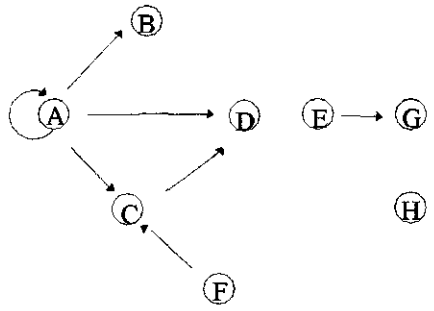

Figura 2.2-a – Grafo orientado

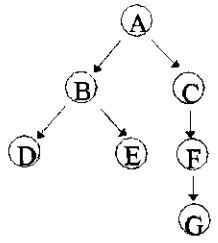

Figura 2.2-b – Árvore

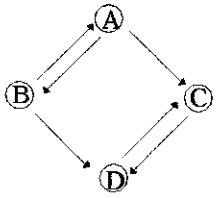

Figura 2.2-c – Grau de entrada e de saída de nós

Uma relação R num conjunto A é uma seqüência de pares ordenados de elementos de A. Por exemplo, se A = {3,5,6,8,10,17}, o conjunto R = {(3,10), (5,6), (5,8), (6,17), (8,17), (10,17)} será uma relação. Se (x, y) for um membro de uma relação R, diz-se que x está relacionado a y em R. A relação R anterior pode ser descrita dizendo-se que x está relacionado com y se x for menor que y e o resto obtido a partir da divisão de y por x for ímpar. (8,17) é um membro dessa relação porque 8 é menor que 17 e o resto da divisão de 17 por 8 é 1, um número ímpar.

Uma relação pode ser representada por um grafo no qual os nós representam o conjunto básico e os arcos representam os pares ordenados da relação. A Figura 2.3-a ilustra o grafo que representa a relação anterior. Um número pode ser associado a cada arco de um grafo, como na Figura 2.3-b. Nessa figura o número associado a cada arco é o resto obtido da divisão do inteiro posicionado na cabeça do arco pelo inteiro posicionado a cada arco, é chamado grafo ponderado ou rede. O número associado a um arco é chamado de peso.

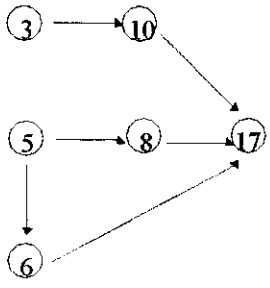

Figura 2.3-a – Conjunto de nós e arcos (pares ordenados)

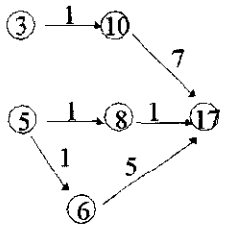

Figura 2.3-b – Grafo ponderado (arcos associados a números)

Um caminho de comprimento k do nó a ao nó b é definido como uma seqüência de k+1 nós n_1, n_2, ..., n_{k+1}, tal que n_1 = a, n_{k+1} = b. Se para algum inteiro k, existir um caminho de comprimento k entre a e b, existirá um caminho de a até b. Um caminho de um nó para si mesmo é chamado ciclo. Se um grafo contiver um ciclo, ele será cíclico; caso contrário, será um grafo acíclico, de acordo com HARARY et al. (1969), ORE et al. (1962-63).

A figura 2.4-a ilustra um grafo G não orientado, com seis nós e nove lados. Os nós a e b são adjacentes, já os nós a e c não são. O grau do nó *e* é 3, sendo os lados (b, e), (a, e) e (f, e); este caminho é simples e elementar. O caminho {a, b, e, f, d, b, c} contém um ciclo, e G está conectado.

A figura 2.4-b ilustra um subgrafo que também é uma árvore. A menor distância entre os nós a e c é igual a 7 unidades. A menor distância entre os pontos x $\in$ (a, f) e y$\in$ (b, d) é de 4 unidades.

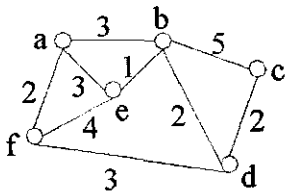

Figura 2.4-a – Grafo não orientado

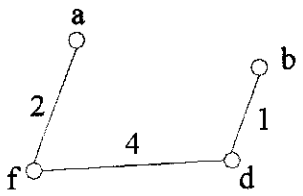

Figura 2.4-b – Subgrafo não orientado

A figura 2.5-a ilustra um grafo G orientado, onde os nós *f* e *e* são adjacentes, e os nós *a* e *d* não são. O caminho {a, f, e, d, g, f, b} é simples e não elementar, desde que ele contenha um ciclo. G é conexo, porém não fortemente conexo.

A figura 2.5-b ilustra um subgrafo de G que também é uma árvore orientada rotacionada no nó a.

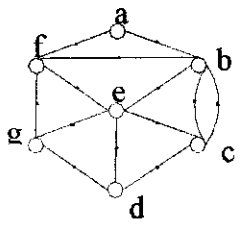

Figura 2.5-a – Grafo orientado

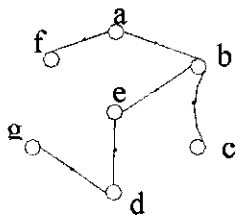

Figura 2.5-b – Subgrafo orientado

Um caminho Euleriano é um caminho que percorre todos os arcos do caminho exatamente uma vez. Um ciclo Euleriano é um ciclo que percorre todos os arcos do grafo exatamente uma vez, com início e término no mesmo nó. Pelo Teorema de Euler, um grafo não orientado pode possuir um ciclo Euleriano se e somente se não possuir nenhum nó de grau ímpar ou pode possuir um caminho Euleriano se e somente se

possuir exatamente dois nós de grau ímpar. No caso de um grafo orientado, o Teorema de Euler exige que o grau de entrada (*indegree)* e o grau de saída (*outdegree)* de cada nó sejam iguais para o ciclo Euleriano existir.

As figuras 2.6-a e 2.6-b ilustram grafos que não possuem nós de grau ímpar, e portanto, possuem ciclo Euleriano. Cada ciclo tem o circuito {B,C,D,A,C,A,B} para o grafo da figura 2.6-b (nota-se que a parte A-C-A do ciclo pode ser percorrida em cada um dos dois caminhos). Os grafos das figuras 2.6-c e 2.6-d possuem exatamente dois nós em cada grau ímpar. Eles devem processar um caminho de Euler de acordo com o teorema. Por um instante {A,B,D,E,A,F,C,B,E,F} é um caminho Euleriano do grafo da figura 2.6-d. Percebe-se que os caminhos Eulerianos são simples, mas não começam em A e terminam em F, ou começam em F e terminam em a.

Finalmente, os grafos das figuras 2.6-e e 2.6-f possuem quatro nós de grau ímpar, e eles não possuem um ciclo Euleriano e nem um caminho formado em cada um. Nota-se que o grafo da figura 2.6-f é um modelo de rede do problema de Konigsberg com sete vértices.

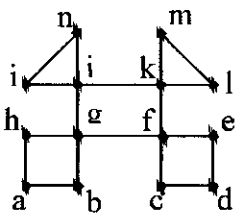

Figura 2.6-a – Ciclo Euleriano

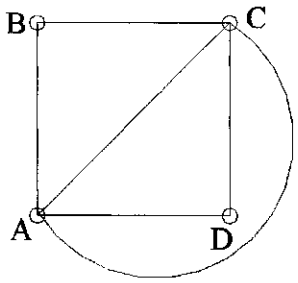

Figura 2.6-b – Ciclo Euleriano

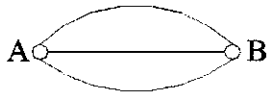

Figura 2.6-c – Caminho Euleriano

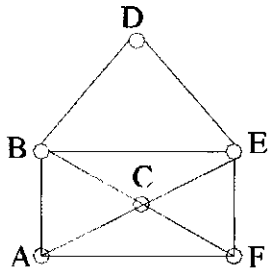

Figura 2.6-d – Caminho Euleriano

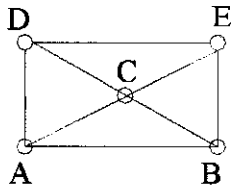

Figura 2.6-e – Grafo sem Ciclo ou Caminho Euleriano

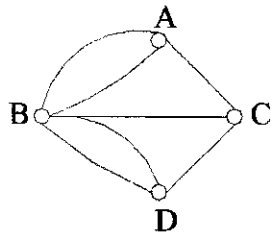

Figura 2.6-f – Grafo sem Ciclo ou Caminho Euleriano

Figura 2.6 - Ilustração do teorema de Euler

Um grafo não orientado é conexo quando existir um caminho ligando todos os pares de nós i, j $\in$ N. Um grafo orientado é conexo quando for conexo o grafo, associado à ele, resultante no caso da orientação ser removida dos arcos. Notando-se

que em um grafo orientado pode não existir um caminho percorrendo do nó i para j. De qualquer modo, no caso de um grafo orientado fortemente conexo, existe caminho entre todos os nós i $\in$ N para todos os nós j $\in$ N, com i $\neq$ j, e vice-versa.

Um subgrafo G'(N', A') de um grafo G(N, A) é um grafo em que o conjunto N'$\subset$ N e que o conjunto A'$\subset$ A. Para grafos não orientados, uma árvore (tree) é um subgrafo conexo que não possui ciclos e uma árvore de cobrimento (spanning tree) de um grafo G(N, A) possui todos os nós do conjunto N. Para grafos orientados, a árvore é orientada e possui um nó raiz e um único caminho de um nó para os outros demais nós da árvore, uma arborescência de um grafo orientado G(N, A) é uma árvore orientada que possui todos os nós do conjunto N.

Em um grafo G(N, A), o número de nós e as características dos arcos são normalmente especificadas. Quase sempre é associado um tamanho (ou comprimento) ao arco, sendo este a indicação de uma unidade como distância, tempo, custo, segurança, entre outros. O tamanho (peso) do arco (i, j) será l (i, j), sabendo que (i,j) $\in$ A. O tamanho de um caminho, S, entre dois nós de G é dado por L(S) = $\sum_{(i,j)\in S} l(i,j)$. Usa-se dist(i, j) para indicar o caminho mínimo entre dois nós i e j $\in$ N de G. (LARSON; ODONI, 1981)

A figura 2.7-a ilustra um exemplo de representação de aplicação de grafos.

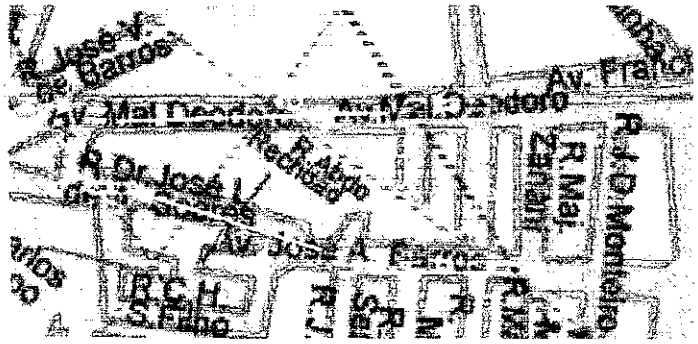

Figura 2.7-a – Aplicação prática de um grafo

A figura 2.7-b ilustra um exemplo de grafo extraído da representação ilustrada pela figura 2.7-a.

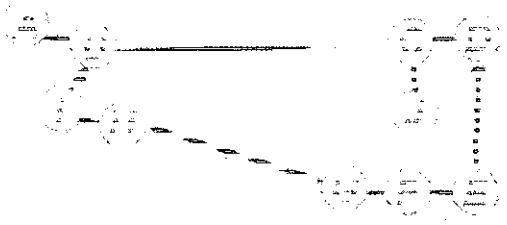

Figura 2.7-b – Grafo extraído da figura 2.7-a

2. Caminho Mínimo – (*Shortest Path*)

A questão mais comum que afronta um motorista que precisa dirigir de uma localidade em uma cidade à outra é a escolha da rota a ser seguida. Entre as alternativas de rotas existentes, uma deve ser escolhida com base em critérios como distância da viagem, tempo de viagem, custo de viagem, segurança e confiabilidade da rota e outros que poderão surgir.

Tratando de problemas de menor caminho, serão estudados eficientes métodos para a determinação de caminhos que minimizem a distância (ou custo) da viagem entre quaisquer duas localidades (pontos) ou entre conjunto de pares de localidades em uma rede de transporte (grafo).

Enquanto a aplicabilidade dessas técnicas de solução para problemas de motoristas, correspondências ou serviços de emergências ser óbvia, existe um motivo mais importante para discussão: a determinação de um caminho mínimo aparece constantemente como um subproblema em grafos e são usados para a construção de blocos em algoritmos que resolvem problemas mais complexos.

Um grande número de algoritmos tem sido propostos durante anos para a resolução de problemas de caminhos mínimos. Entretanto, todos podem ser considerados como variações dos poucos temas básicos e os dois algoritmos que irão ser descritos neste trabalho ilustra esses temas. O primeiro dos algoritmos, o algoritmo de Dijkstra, irá determinar o caminho mínimo entre um dado nó e todos os outros nós do grafo, enquanto o segundo, o algoritmo de Floyd, obtém simultaneamente o caminho mínimo entre todos os pares de nós do grafo.

Ambos os algoritmos podem ser usados com grafos direcionados, não direcionados ou mistos, e todos assumem que os arcos possuem valores (tamanhos) não-negativos. Enquanto essa última suposição é razoável no contexto de sistemas de serviços urbanos, em que os tamanhos dos arcos usualmente representam distância ou tempo de viagem, este pode não ser o caso de outras aplicações em que o tamanho de arcos podem, por exemplo, representar custos e então tamanhos negativos iriam implicar um lucro par travessia do arco correspondente.

Existem algoritmos para a determinação de caminhos mínimos em grafos com arcos negativos e estes podem ser considerados como simples extensões dos dois algoritmos que irão ser descritos a seguir.

A preferência de o assunto abordado ser sobre grafos com arcos não negativos se dá pelo fato de serem estes os observados na maioria dos problemas combinatoriais práticos em grafos.

2.1. CAMINHO MÍNIMO ENTRE UM DADO NÓ E OS OUTROS NÓS DO GRAFO

Uma necessidade comum em uma empresa de ônibus interurbanos que faz diversas rotas como, por exemplo, São Paulo – Rio de Janeiro, São Paulo – Belo Horizonte, São Paulo – Vitória e São Paulo – Curitiba é saber qual o caminho, dos diversos existentes para cada destino, possui a menor distância.

Com isso temos como primeiro problema a busca do caminho mínimo entre um dado nó e todos os outros nós de um dado grafo G(**N**, A), assumindo que o tamanho de todos os seus arcos l(i, j) são não negativos.

De acordo com DIJKSTRA (1959), o algoritmo consiste basicamente em percorrer todos os nós do grafo, a partir de um nó fonte, e então procurar sucessivamente o nó mais próximo, o segundo mais próximo, o terceiro mais próximo, e assim por diante, um de cada vez, até todos os nós terem sido visitados.

O algoritmo é de fácil execução em formato de tabelas e razoavelmente simples para implementação computacional. Este algoritmo, ou mais exatamente, a menor variação dele é geralmente atribuída a Dijkstra [DIJKSTRA, 1959], mas muitos algoritmos similares têm sido propostos durante todos os anos.

O procedimento é realizado pelo uso de dois vetores como saída com elementos dist[j] e cam[j], para cada nó j, em que dist[j] representa o tamanho do caminho mínimo do nó fonte ao nó j e cam[j] representa o nó predecessor imediato para ir ao nó j no caminho mínimo.

Sendo mat[i][j] a matriz de distâncias entre todos os nós i à j do grafo em que é atribuído um valor infinito para os arcos inexistentes, infinito um valor de um ciclo máximo do grafo sem a orientação, visita[j] o vetor que sinaliza os nós já visitados, nos o número de nós do grafo, fonte o nó fonte escolhido, atual o nó que está sendo visitado.

2.1.1. *Algoritmo de Dijkstra*

PASSO 1:

Logo após ser escolhido o nó fonte e inicializar a variável especifica (fonte) com o índice do nó fonte, inicializa-se os elementos do vetor de distâncias do nó fonte até todos os outros nós do grafo com os valores dos elementos da matriz de distâncias do grafo referentes a distância do nó fonte até os demais nós, dist[j]=mat[fonte][j] variando j de 0 à nnos-1, sendo nnos o número de nós do grafo, os elementos do vetor que sinaliza os nós visitados com um valor nulo (nó não visitado), visita[j]=0 variando j de 0 à nnos-1, a variável que auxilia a execução do algoritmo com o valor fonte, atual = fonte;

Inicializa-se o elemento do vetor de distâncias do nó fonte à todos os outros nós do grafo referente à distância do nó fonte ao nó fonte com um valor nulo, dist[fonte]=0, o elemento do vetor da seqüência de nós do caminho mínimo referente ao nó antecessor ao nó fonte com –1 (nó inexistente), cam[fonte] = -1, o elemento do vetor que sinaliza os nó já visitados referente ao nó fonte com 1 (nó visitado), visita[fonte]=1;

PASSO 2:

Verifica-se se o nó em questão, nó j, já foi visitado ou não. Se já foi visitado, verifica-se o nó seguinte. Se não foi visitado, atualiza-se o elemento do vetor de distâncias do nó fonte aos demais nós (referente ao nó j) com o menor valor da comparação entre os elementos do vetor de distâncias do nó fonte aos demais nós (referente ao nó j) e a soma do elemento do vetor de distâncias do nó fonte aos demais nós (referente ao nó com índice da variável auxiliar, nó atual) e o elemento da matriz de distâncias do grafo (referente a distância do nó atual ao nó j), ou seja, se visita[j]=0, então dist[j] = menor (dist[j], dist[atual]+mat[atual][j]), sendo *menor* uma função que retorna o menor valor da comparação entre dois valores;

PASSO 3:

Para se escolher o próximo nó a visitar, calcula-se um valor que seja maior que qualquer elemento do vetor de distâncias do nó fonte e os demais, chamando-o de infinito. Verifica-se se o nó em questão, nó j, não foi visitado e se o elemento do vetor de distâncias do nó fonte aos demais nós (referente ao nó j) é menor que o valor de

infinito. Se as condições não forem satisfeitas, verifica-se o nó seguinte. Se as condições forem satisfeitas, atualiza-se o valor de infinito com o valor do elemento do vetor de distâncias do nó fonte aos demais nós (referente ao nó j) e a variável auxiliar (atual) com o índice j, ou seja, se visita[j]=0 e dist[j] < infinito, infinito = dist[j] e atual=j;

PASSO 4:

Para se achar o nó precedente ao nó em questão, nó j, verifica-se se o nó j já foi visitado e se o elemento do vetor de distâncias do nó fonte aos demais nós (referente ao nó j) é igual diferença entre o elemento do vetor de distâncias do nó fonte aos demais nós (referente ao nó com índice da variável auxiliar, nó atual) e o elemento da matriz de distâncias do grafo (referente a distância do nó j ao nó atual). Se as condições não forem satisfeitas, verifica-se o nó seguinte. Se forem satisfeitas, atualiza-se o elemento do vetor da seqüência de nós do caminho mínimo (referente ao nó atual) com o valor de j, ou seja, se visita[j])=1 e dist[j]=dist[atual]-mat[j][atual], cam[atual]=j;

PASSO 5:

Atualiza-se o elemento do vetor que sinaliza os nós visitados (referente ao nó atual) com o valor 1 (nó visitado), ou seja, visita[atual]=1;

Execute do PASSO 2 ao PASSO 5 mais nnos-2 vezes, sendo nnos o número de nós do grafo.

2.1.2. Exemplificação do Algoritmo de Dijkstra

No primeiro passo, faz-se a inicialização das variáveis. No segundo passo, atualizam-se as distâncias mínimas do nó fonte aos nós não visitados. No terceiro passo, encontra-se o nó não visitado com a menor distância entre o nó fonte e ele. No quarto passo, faz-se uma busca entre os nós visitados e verifica se a distância mínima entre o nó fonte e o nó visitado e é igual a distância mínima entre o nó fonte e o nó achado no terceiro passo subtraído da distância entre o nó visitado e o nó achado no terceiro passo, então atualiza-se o vetor de nós precedentes correspondente ao nó achado no terceiro passo com o nó visitado. No quinto passo, sinaliza-se o nó achado no terceiro passo como visitado. Se todos os nós já forem visitados, então para-se, caso contrário volta-se ao segundo passo.

A figura 3.1.2.1 ilustra um exemplo para execução do algoritmo de Dijkstra.

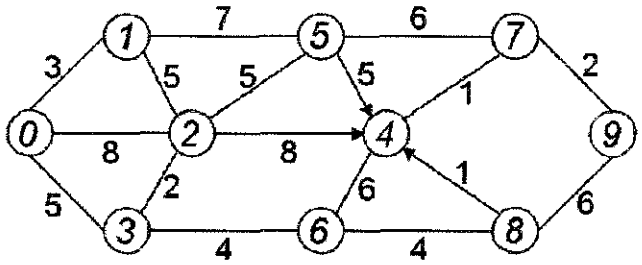

Fig. 3.1.2.1 – Exemplo para execução do Algoritmo Dijkstra

Inicialização: seja o nó 0 o nó fonte. Sabendo que nnos=10, inicializa-se fonte=0, dist[0]=0, cam[0]= -1, visita[0]=1, atual=0, visita[j]=0 e dist[j]=mat[fonte][j], com j variando de 0 a nnos-1 e j≠0 (fonte), ficando dist[j]={0,3,8,5, ∞,∞,∞,∞,∞,∞} e visita={1,0,0,0,0,0,0,0,0,0};

A Tabela 3.1.2.1 representa a matriz de distancias da figura 3.1.2.1 e a Tabela 3.1.2.2 representa o Quadro de Iterações do Algoritmo de Dijkstra.

Matriz de Distâncias (mat)										
↱	0	1	2	3	4	5	6	7	8	9
0	0	3	8	5	∞	∞	∞	∞	∞	∞
1	3	0	5	∞	∞	7	∞	∞	∞	∞
2	8	5	0	2	8	5	∞	∞	∞	∞
3	5	∞	2	0	∞	∞	4	∞	∞	∞
4	∞	∞	∞	∞	0	∞	6	1	∞	∞
5	∞	7	5	∞	5	0	∞	6	∞	∞
6	∞	∞	∞	4	6	∞	0	∞	4	∞
7	∞	∞	∞	∞	1	6	∞	0	∞	2
8	∞	∞	∞	∞	1	∞	4	∞	0	6
9	∞	∞	∞	∞	∞	∞	∞	2	6	0

Tabela 3.1.2.1 – Matriz de distancias do grafo da figura 3.1.2.1

A Tabela 3.1.2.2 representa os Quadros das iterações passo a passo da execução do Algoritmo de Dijkstra.

1ª Iteração	j=0	j=1	j=2	J=3	j=4	j=5	j=6	j=7	j=8	j=9	Atual	Cam(Atual)
dist[j]	0	3	8	5	∞	∞	∞	∞	∞	∞	1	0
visita[j]	1	0	0	0	0	0	0	0	0	0		

2ª Iteração	j=0	j=1	j=2	J=3	j=4	j=5	j=6	j=7	j=8	j=9	Atual	Cam(Atual)
dist[j]	0	3	8	5	∞	10	∞	∞	∞	∞	3	0
visita[j]	1	1	0	0	0	0	0	0	0	0		

3ª Iteração	j=0	j=1	j=2	J=3	j=4	j=5	j=6	j=7	j=8	j=9	Atual	Cam(Atual)
dist[j]	0	3	7	5	∞	10	9	∞	∞	∞	2	3
visita[j]	1	1	0	1	0	0	0	0	0	0		

4ª Iteração	j=0	j=1	j=2	J=3	j=4	j=5	j=6	j=7	j=8	j=9	Atual	Cam(Atual)
dist[j]	0	3	7	5	15	10	9	∞	∞	∞	6	3
visita[j]	1	1	1	1	0	0	0	0	0	0		

5ª Iteração	j=0	j=1	j=2	J=3	j=4	j=5	j=6	j=7	j=8	j=9	Atual	Cam(Atual)
dist[j]	0	3	7	5	15	10	9	∞	13	∞	5	1
visita[j]	1	1	1	1	0	0	1	0	0	0		

6ª Iteração	j=0	j=1	j=2	J=3	j=4	j=5	j=6	j=7	j=8	j=9	Atual	Cam(Atual)
dist[j]	0	3	7	5	15	10	9	16	13	∞	8	6
visita[j]	1	1	1	1	0	1	1	0	0	0		

7ª Iteração	j=0	j=1	j=2	J=3	j=4	j=5	j=6	j=7	j=8	j=9	Atual	Cam(Atual)
dist[j]	0	3	7	5	14	10	9	16	13	19	4	8
visita[j]	1	1	1	1	0	1	1	0	1	0		

8ª Iteração	j=0	j=1	j=2	J=3	j=4	j=5	j=6	j=7	j=8	j=9	Atual	Cam(Atual)
dist[j]	0	3	7	5	14	10	9	15	13	19	7	4
visita[j]	1	1	1	1	1	1	1	0	1	0		

9ª Iteração	j=0	j=1	j=2	J=3	j=4	j=5	j=6	j=7	j=8	j=9	Atual	Cam(Atual)
dist[j]	0	3	7	5	14	10	9	15	13	17	9	7
visita[j]	1	1	1	1	1	1	1	1	1	0		

Tabela 3.1.2.2 – Quadros de Iterações do Algoritmo de Dijkstra

Depois de executado o algoritmo, os dist[j] indicam o tamanho do caminho mínimo entre o nó fonte e o nó j, enquanto os cam[j] indicam os nós precedentes ao nó j no caminho mínimo.

Então, por exemplo, para verifica-se a seqüência de nós do caminho mínimo do nó fonte (0) ao nó 9, retrocedendo a partir do nó destino, cam[9]=7, cam[7]=4, cam[4]=8, cam[8]=6, cam[6]=3 e cam[3]=0, portanto conclui-se que o caminho é $0 \rightarrow 3 \rightarrow 6 \rightarrow 8 \rightarrow 4 \rightarrow 7 \rightarrow 9$ e seu tamanho é 17 unidades (dist[9]=17).

2.2. CAMINHO MÍNIMO ENTRE TODOS OS PARES DE NÓS

Algoritmo de Floyd (FLOYD, 1962)

É muito comum o caso de ser necessário o caminho mínimo entre todos os pares de nós de um grafo. Um exemplo óbvio é a preparação de tabelas indicando distâncias entre todas as maiores cidades em mapas rodoviários de estados ou regiões, obtenção do caminho mínimo de um nó inicial, passando por determinada ordem de nós prioritários, chegando em um nó final, entre outras aplicações como entrega de encomendas com hora marcada por empresas distribuidoras de mercadorias em pronta entrega.

Uma solução óbvia é repetir a aplicação do algoritmo de Dijkstra, já descrito, sucessivamente para cada um dos nós do grafo. Outra solução, mais limpa, eficiente e simples de implementar é o algoritmo de Floyd [FLOYD, 1962].

O algoritmo de Floyd é simples de descrever, porém sua lógica é um pouco mais complexa para se entender. O procedimento desse algoritmo é atualizar a matriz de distâncias n vezes, sendo n o número de nós do grafo, buscando na atual-ésima iteração por menores distâncias entre os pares de nós que passem pelo nó atual. Isto é realizado pelo uso de três matrizes com elementos mat[i][j], dist[i][j] e cam[i][j], em que mat[i][j] representa a distância do nó i ao j, dist[i][j] a distância mínima do nó i ao j, cam[i][j] representa o nó predecessor imediato ao nó j no caminho mínimo do nó i ao j.

Inicialmente, enumera-se os nós do grafo G(N, A) com inteiros positivos 1, 2, ...n. A Tabela 3.2.1 representa os elementos das matrizes dist e cam.

dist[i][j]=	mat[i][j], se caminho de i à j existir 0, se i = j ∞ (infinito), se caminho de i à j não existir
cam[i][j]=	i, se $i \neq j$ -1, se i = j

Tabela 3.2.1 – Elementos de dist e cam

As figuras 3.2.1-a, 3.2.1-b, 3.2.1-c e 3.2.1-d ilustram a demonstração da execução das iterações do algoritmo de Floyd.

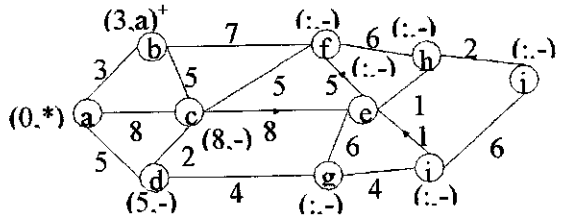

Figura 3.2.1-a

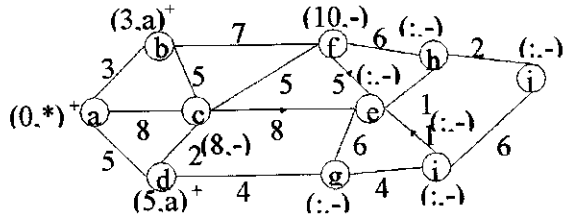

Figura 3.2.1-b

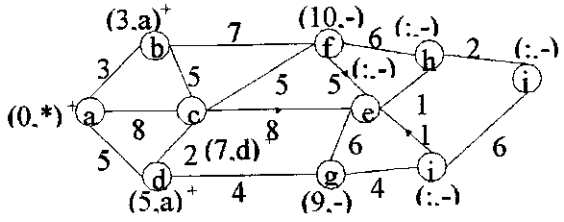

Figura 3.2.1-c

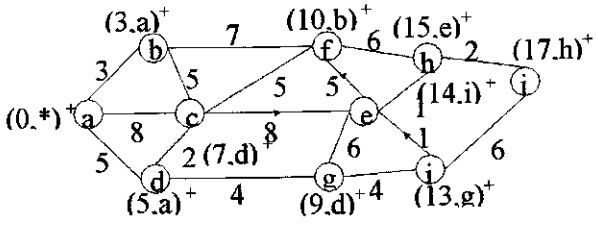

Figura 3.2.1-d

A figura 3.2.2 ilustra o grafo resultante do Algoritmo de Floyd.

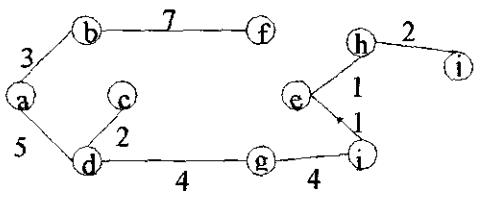

Figura 3.2.2 – Menor caminho entre todos os pares de nós do grafo

2.2.1. Algoritmo de Floyd

PASSO 1:

Inicializa-se os elementos de cada linha da matriz da seqüência de nós do caminho mínimo com o índice referente a linha quando o elemento for referente ao caminho entre nós diferentes, cam[i][j]=i para i≠j, e com –1 (não existe nó antecessor) quando o elemento for referente a um caminho inexistente (entre os mesmos nós), cam[i][j]= -1 para i=j; Lembrando que na matriz de distâncias do grafo, caminhos inexistentes (entre nós os mesmos nós) são inicializados com um valor, chamado infinito, que seja maior que qualquer elemento da matriz de distâncias do grafo, mat[i][j]=infinito para i=j;

Inicializa-se a variável que auxilia a execução do algoritmo com um valor nulo, atual=0; Chama-se nó atual o nó de índice igual ao valor da variável atual;

PASSO 2:

Atualiza-se os elementos da matriz de distâncias mínimas do nó de índice i, nó i, ao nó de índice j, nó j, com o menor valor da comparação entre o elemento da matriz de distâncias do nó i ao nó j e a soma do elemento da matriz de distâncias do nó i ao nó atual com o elemento da matriz de distâncias do nó atual ao nó j, ou seja, dist[i][j] = Menor (mat[i][j], mat[i][atual] + mat[atual][j]);

PASSO 3:

Se os elementos da matriz de distâncias mínimas do nó i ao nó j, dist[i][j], forem diferentes dos elementos da matriz de distâncias do nó i ao nó j, mat[i][j], atualiza-se os elementos da matriz de seqüência de nós do caminho mínimo do nó i ao nó j com os valores dos elementos da matriz de seqüência de nós do caminho mínimo do nó atual ao nó j, cam[i][j] = cam[atual][j];

PASSO 4:

Atualiza-se os elementos da matriz de distâncias do nó i ao nó j, mat[i][j], com os valores dos elementos da matriz de distâncias mínimas do nó i ao nó j, dist[i][j];

Se a variável atual for menor que o número de nós do grafo, incrementa-se a variável atual, ou seja, se atual<nnos faz-se atual=atual+1. Caso atual seja igual ao número de nós do grafo, atual=n, então para-se.

2.2.2. *Exemplificação do Algoritmo de Floyd*

A figura 3.2.2.1 ilustra um exemplo de grafo misto para a execução do Algoritmo.

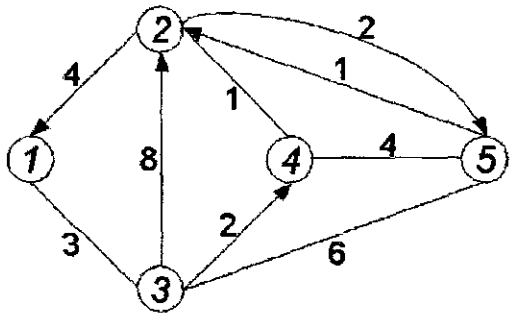

Figura 3.2.2.1 – Grafo misto

No primeiro passo, inicializa-se dist, cam e atual=1, este último conta as iterações e auxilia na atualização do caminho mínimo. No segundo passo, atualiza-se os elementos dist[i][j] com o menor da comparação entre mat[i][j] e mat[i][atual]+mat[atual][j]. No terceiro passo, atualiza-se os elementos cam[i][j] com cam[atual][j] se dist[i][j]≠mat[i][j]. Por fim, no quarto passo, verifica-se se atual<nnos, se for atualiza-se mat[i][j]=dist[i][j] e incrementa atual de 1. Caso contrário, para-se.

No término da execução do algoritmo, o elemento dist[i][j], representará o tamanho do caminho mínimo do nó i para o nó j, enquanto a matriz de predecessores cam, ou matriz de marcação do caminho mínimo, torna possível, traçando em retrocesso nó a nó, o conhecimento da rota do caminho mínimo.

A Tabela 3.2.2.1-a e 3.2.2.1-b representam as matrizes de distancias e de seqüência de caminhos respectivamente, ambas iniciais.

Dist					
↱	1	2	3	4	5
1	0	∞	3	∞	∞
2	4	0	∞	1	2
3	3	8	0	2	6
4	∞	1	∞	0	4
5	∞	1	6	4	0

Tabela 3.2.2.1-a: Matriz de Distancias

Cam					
↱	1	2	3	4	5
1	-1	1	1	1	1
2	2	-1	2	2	2
3	3	3	-1	3	3
4	4	4	4	-1	4
5	5	5	5	5	-1

Tabela 3.2.2.1-b: Matriz de Caminhos

Após a variável atual variar de 1 à 5, ou seja, após n iterações, a Tabela 3221-c e Tabela 3.2.2.1-d representa a Matriz de Distâncias Mínimas e a Matriz de Caminhos final respectivamente.

dist					
↱	1	2	3	4	5
1	0	6	3	5	8
2	4	0	7	1	2
3	3	8	0	2	6
4	5	1	8	0	3
5	5	1	6	2	0

Cam					
↱	1	2	3	4	5
1	-1	4	1	3	2
2	4	-1	1	2	2
3	3	4	-1	3	2
4	2	4	1	-1	2
5	2	5	5	2	-1

Tabela 3.2.2.1-c: Matriz de Distancias Mínimas Tabela 3.2.2.1-d: Matriz de Caminhos

Logo após as iterações, as matrizes dist e cam indicam, respectivamente, a matriz de caminhos mínimos e a matriz de predecessores, observando-se, por exemplo, que o tamanho do caminho mínimo do nó 1 ao nó 5 é dist[1][5]=8 e seu caminho, traçado em retrocesso é cam[1][5]=2, cam[1][2]=*4*, cam[1][4]=*3* e cam[1][3]=*1*, resultando em $1 \rightarrow 3 \rightarrow 4 \rightarrow 2 \rightarrow 5$.

3. Árvores

3.1. Árvore de Cobrimento Mínimo (*Minimum Spanning Tree*)

Uma árvore de cobrimento de um grafo G(N, A) não orientado como já foi definido como uma árvore do grafo G que contém o conjunto completo de nós N de G. O problema da árvore de cobrimento mínimo é procurar uma entre todas as possíveis árvores de cobrimento do grafo com a soma total de tamanhos de arcos que seja mínima. Se o número de nós do conjunto N é n, então todas as árvores de cobrimento de G possuirão n-1 arcos.

Exatamente como no caso dos problemas de caminho mínimo, o problema da árvore de cobrimento mínimo tem aplicações diretas, como no desenvolvimento de rede de transportes e também pode constituir um importante bloco na construção de soluções aproximadas (métodos heurísticos) para problemas mais complexos como o problema de cobertura de nós ou o chamado Caixeiro Viajante, de acordo com Papadimitriou et al. (1977).

Este problema aparece, por exemplo, no seguinte contexto: dado um mapa de n cidades rurais com uma matriz listando as distâncias euclidianas entre todos os pares possíveis de cidades, deseja-se obter o menor comprimento de rodovias necessárias para ligar todas estas cidades.

Uma solução para esse problema pode ser derivada, por exemplo, de uma idéia básica: escolhendo um nó arbitrário inicialmente, visita-se todos os nós do grafo escolhendo como próximo a ser visitado o nó mais próximo de um dos nós já visitados. Assumindo-se que o conjunto N de nós do grafo G possua um número de elementos maior que 1, ou seja, n>1.

3.1.1. O Algoritmo da Árvore de Cobrimento Mínimo

PASSO 1:

Inicializa-se uma variável que auxilia o calculo da menor árvore construída com um valor maior que qualquer elemento da matriz de distâncias chamado de infinito e uma variável de contagem de laços com um valor nulo, c=0;

PASSO 2:

Inicializa-se os elementos do vetor que sinaliza os nós i, nós de índice i com i variando de 0 até nnos (número de nós do grafo), já visitados com um valor nulo, visit[i]=0, e vetor da seqüência de nós do caminho mínimo com o valor –1 (sem nó antecessor), cam[i] = -1;

Inicializa-se o elemento do vetor de nós já visitados com índice c com o valor 1 (nó já visitado) e uma variável que auxilia na determinação da árvore de cobrimento mínimo, chamada soma, com um valor nulo, soma=0;

Inicializa-se outra variável de contagem de laços com um valor nulo, k=1;

PASSO 3:

Inicializa-se uma variável que auxilia na determinação do nó mais próximo ao nó i com o valor maior que o de infinito, ou seja, max = infinito+1, lembrando que os elementos da matriz de distâncias referentes a arcos inexistentes possuem valor igual a infinito; Inicializa-se uma variável que habilita construção da árvore com um valor nulo, ok = 0;

Busca-se o nó j mais próximo ao nó i verificando-se se o nó i já foi visitado, o nó j não foi visitado e o elemento da matriz de distâncias do nó i ao nó j for menor que a variável max, então atualiza-se max com o valor d o elemento da matriz de distâncias do nó i ao nó j, a variável x com o valor de i e a variável y com o valor de j e a variável ok com o valor 1, ou seja, se visit[i]=1, visit[j]=0 e mat[i][j]<max, então faça max=mat[i][j], x=i, y=j, e ok=1;

Se o nó j for achado então a variável ok é igual a 1 então, atualizando a variável soma, soma-se o elemento da matriz de distâncias do nó de índice x ao nó de índice y à variável soma, ou seja, soma = soma + mat[x][y]; atualiza-se o elemento do vetor de seqüência de nós do caminho mínimo referente ao nó y com o valor de x (nó

antecessor a y), cam[y] = x, e sinaliza-se o elemento do vetor de nós já visitados referente ao nó y com o valor 1 (nó já visitado), visit[y] = 1; Faça k = k + 1; se k for menor que nnos (número de nós do grafo), volta-se ao PASSO 3; se não, continua-se;

PASSO 4:

Se a última árvore de cobrimento mínimo construída for a menor até então, ou seja, se soma < maior, então atualiza-se a variável maior com o valor de soma, uma variável chamada fonte com o valor de c e faz-se uma cópia do vetor da sequencias de nós do caminho mínimo chamado tree;

Incrementa-se a variável c de 1, c = c + 1; Se c for menor que o número de nós do grafo, então volta-se ao PASSO 2; caso contrário, para-se.

3.1.2. Exemplificação da Árvore de Cobrimento Mínimo

A figura 4.1.2.1 ilustra um grafo G como exemplo para a execução dos passos do algoritmo da Árvore de cobrimento mínimo.

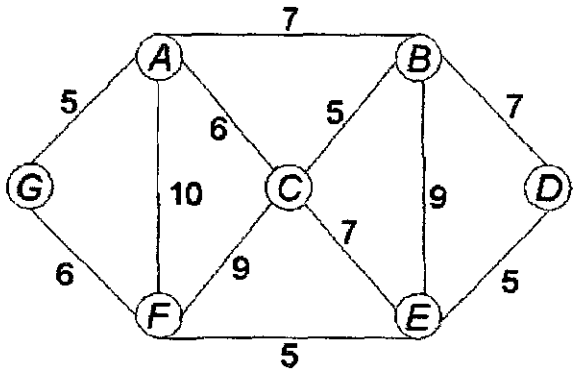

Figura 4.1.2.1 – Exemplificação de árvore

Arbitrando o nó A como o nó inicial e seguindo os passos do algoritmo. A Tabela 4.1.2.1 representa os resultados da execução do algoritmo.

ARCOS INTRODUZIDOS	NÓS CONECTADOS	NÓS NÃO CONECTADOS
-	A	B – C – D – E – F - G
(A, G)	A – G	B - C - D - E - F
(A, C)	A – C - G	B – D – E - F
(C, B)	A – B – C - G	D – E - F

(G, F)	A – B – C – F - G	D - E
(F, E)	A – B – C – E – F - G	D
(E, D)	A – B – C – D – E – F - G	-

Tabela 4.1.2.1 – Execução do algoritmo da Árvore do Cobrimento Mínimo

De acordo com a Tabela 4.1.2.1 da execução do algoritmo, pode-se afirmar que a Árvore de Cobrimento Mínimo é representada pelo grafo da figura 4.1.2.2.

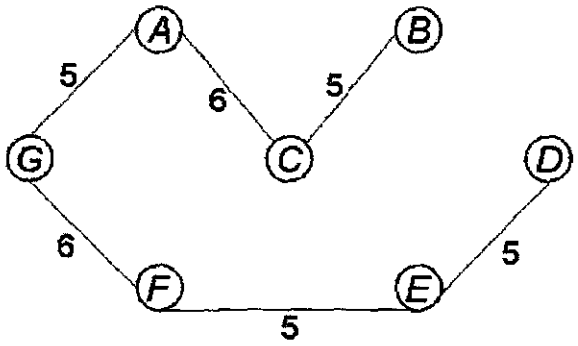

Figura 4.1.2.2 – Árvore do Cobrimento Mínimo

Tendo a árvore de cobrimento mínimo gerada, pode-se afirmar que a árvore é formada pelos arcos da coluna "Arcos Introduzidos" da Tabela 4.1.2.1 e pelos nós do grafo ilustrado pela figura 4.1.2.1.

4. Problema do Carteiro Chinês

Para resolver estes problemas em grafos orientados, de acordo com BELTRAMI et al. (1974), inicia-se equacionando versões do teorema de Euler que são aplicados a grafos direcionados: um grafo orientado conexo possui um ciclo Euleriano se e somente se o grau de entrada (*indegree*) e o grau de saída (*outdegree*) de cada um dos nós forem iguais.

A prova deste teorema é completamente análoga e a prova do teorema de Euler para grafos não direcionados (Pode-se refazer os passo conforme a vontade do usuário).

Para resolver o problema do carteiro chinês em qualquer grafo direcionado, é necessário ter-se a seguinte definição:

Pi = "Polaridade" de i = (*indegree* de i) – (*outdegree* de i)

Um nó i para cada Pi>0 (Pi<0) é chamado de nó de oferta ou "demanda". Indica-se o conjunto de nós de ofertas e demandas como S e D, respectivamente:

$$\sum_{i \in N} Pi = 0$$

O algoritmo a seguir resolve o problema em questão para grafos direcionados.

4.1. ALGORITMO DO CARTEIRO CHINÊS

PASSO 1:

Identifica-se os nós de oferta e de demanda e calcula-se as polaridades de cada nó e minimiza-se as distâncias d(i, j) de todos os nós $i \in j$ para todos o nós $j \in D$.

PASSO 2:

Resolve-se o problema dos transportes para encontrar as combinações ótimas dos nós de oferta com os de demanda. Este PT é:

Minimize: $\sum_{i \in j} \sum_{j \in D} d(i,j)xij$

Sujeito à:

$\sum_{j \in D} x_{ij} = Pi$ para todos os nós $i \in S$

$\sum_{i \in S} x_{ij} = -P_j$ para todos os nós $j \in D$

Com $x_{ij} \geq 0$. (Lembrando que $P_j < 0$ para $j \in D$ então - P_j é positivo)

PASSO 3:

Para cada $x_{ij} > 0$ na solução do PT. Adiciona-se à G, x_{ij} réplicas do caminho mínimo de $i \in S$ à $j \in D$. Resultando em um grafo G' que possui seus Pi=0 para todos os nós.

PASSO 4:

Encontra-se um ciclo Euleriano em G'. Este ciclo é a solução do PCC em G.

4.2. EXEMPLIFICAÇÃO DO ALGORITMO DO CARTEIRO CHINÊS

Considere o caso de um carteiro responsável pela correspondência em uma área mostrada pelo grafo da figura abaixo:

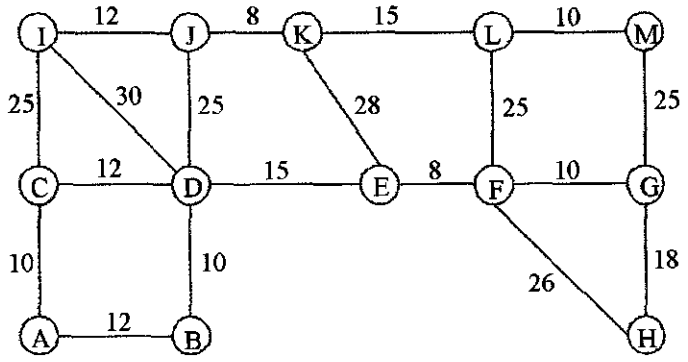

Figura 6.2.1– Área de cidade de responsabilidade do carteiro

O carteiro deverá sempre iniciar sua rota de entrega no nó A onde está localizado o Correio, deverá passar por todas as ruas (arcos) de sua área e retornar ao nó inicial A.

A questão de como deverá ser a rota a fim de que o carteiro passe todas as ruas ao menos uma vez, minimizando a distância total que ele percorre. Este prolema é conhecido como Chinese Postman Problem, nome derivado do fato de ter sido no jornal Chinese Mathematics, em 1952, a primeira vez em que esse problema foi discutido.

A história desse problema é muito interessante. No século XVIII, os moradores da cidade russa de Konigsberg (hoje Kaliningrad, cidade onde o filósofo prussiano Immanuel Kant nasceu (em 1724) e passou toda a sua vida) queriam realizar um desfile que pudesse passar pelas sete pontes sobre o rio Prevel apenas uma vez e deram o problema para o matemático suíço Leonhard Euler resolver.

Euler provou em 1736 que não existe solução para esse problema. Ele também obteve alguns resultados gerais que possibilitaram a motivação para as soluções do problema do carteiro chinês. Um deles bastante importante, nos diz que o número de nós de grau ímpar de um grafo não-orientado G é sempre par, visto que a soma dos graus de todos os nós em G é um número par pois cada arco é incidente a dois nós.

Outro resultado bastante importante, o teorema de Euler, nos diz que para o carteiro efetuar um "Ciclo Euleriano", devem existir ZERO nós de grau ímpar no grafo. Em outras palavras, se quiser realizar um ciclo euleriano em um grafo qualquer, este deverá ser modificado de modo a tornar de grau par todos os seus nós de grau ímpar, sem mudanças em sua estrutura.

Seja um grafo G(N, A) e $M \subset N$ o subconjunto dos nós de grau ímpar. Então como foi citado anteriormente, a cardinalidade (número de elementos) de M é par e esse valor chama-se de m. Realizando m/2 ligações dois a dois entre os elementos de M, aumenta-se em uma unidade o grau desses nós, tornando-os de grau par.

Por exemplo, para o grafo da figura 6.2.1, N = {A, B, C, D, E, F, G, H, I, J, K, L, M}, M = {C, D, E, F, G, I, J, K, L} e m = 8 (um número par).

Define-se a distância de um par de nós, de um grafo conexo não-orientado, não necessariamente interligados por um arco, como sendo o Comprimento do menor caminho entre os nós desse par. No grafo da figura anterior, por exemplo, a distância do par [E-L] equivale a 33 unidades.

5. Problema de Localização de Facilidades

São voltados para uma outra categoria extensiva a sistemas de problemas urbanos, de acordo com determinadas localizações de serviços de veículos a procura de maiores facilidades.

Estes problemas também possuem uma rotina de serviços de emergência, mas os objetivos são usualmente diferentes nos dois casos.

Algo facilmente suspeito é que, a melhor das localizações depende da proporção dos começos mais efetivos utilizados. Pegando um exemplo extremo, o centro da cidade pode ser considerado uma localização ideal para um departamento de correios, mas certamente seria uma escolha muito pobre para um ponto de incinerador de lixo. No caso de um departamento de correios, o objetivo razoável pode ser minimizar a distância do caminho médio para edifícios na cidade residentes no centro da cidade. Para um ponto de incinerador de lixo, o objetivo mais apropriado poderia ser maximizar a distância mínima entre o ponto escolhido e qualquer casa ou edifício da cidade.

Para resolver problemas de localizações de facilidades utilizaremos os problemas das medianas.

5.1. Problema das Medianas

O número de facilidades deve ser localizado minimizando a distância média (ou o tempo médio ou o custo médio de viagem) das facilidades para a população e seus usuários. O problema das medianas oferece um conceito de construção de facilidades para entregas domiciliares e serviços de emergências. (departamentos de correios, pontos de transportes, agências telefônicas, pequenas salas comerciais, agências de relações governamentais extensivos ao público, etc.).

Modelos de redes em centros urbanos ou metropolitanos são particularmente convenientes para a discussão de problemas de localizações de facilidades. Chamamos para representar vários problemas de transportes principais como os links na rede e suas intersecções de nós. Além disso, a suposição será que a demanda para serviços somente será generalizada se o número de pontos finitos, também for designado ao conjunto de nós na rede. A suposição pode ser inicializada utilizando

limites para potencial usualmente nos modelos. Entretanto, um pode sempre substituir muitos nós ao longo dos links na rede para representar pontoes de demanda, e os modelos podem ser feitos com detalhes realísticos e chamados por casos manuais. Podemos utilizar a carga da demanda h_j para indicar a intensidade que cada demanda origina por serviço no nó j.

Considerando um grafo não direcionado G(N, A) com n nós. Sendo k um número inteiro positivo (k=1, 2, 3...) e escolhendo k como números distintos no grafo G a ser indicado como $X_k = \{x_1, x_2, ...x_{k-1}, x_k\}$. São indicados por $d(X_k, j)$ a menor distância entre qualquer um dos pontos $x_1 \varepsilon x_k$ e o nó j em G. Logo tem-se:

$$d(X_k, j) = \text{Min}x_1 \varepsilon x_k \, d(x, j)$$

Temos por definição que um conjunto de pontos k em X_k^* em G é um conjunto de medianas de k de G se, $X_k \varepsilon G$, da seguinte forma:

$$J(X_k^*) \leq J(x_k)$$

Onde:

$$J(X_k) = \sum_{j=1}^{n} h_j d(X_k, j)$$

Se os pontos k em X_k são os pontos onde as k facilidades executam um dado serviço que será localizado e se h_j, que é o valor da demanda do nó j, é igualado a fração de todas as demandas de serviço em questão, que origina de $j(\sum_{j=1}^{n} hj = 1)$, então buscando as k medianas, X_k^*, de G somado para encontrar o conjunto de k locações que minimizam a distância média para (ou das) facilidades por usuários de serviços. Note bem que a suposição implícita de todos acima é que as demandas originadas em algum dado nó j irão servir exclusivamente para facilidade que é fechada em j.

Pelo menos um conjunto de k medianas existe exclusivamente nos nós de G. O significado prático deste teorema é sua eficácia. Os estados em efeito que procuram por k de locações ótimas para as k facilidades podem ser limitados no conjunto de nós

de G (total de n pontos). Em vez de números infinitos de pontos que se encontram nos links de G.

A validade do teorema é óbvia para um caso trivial onde o número de facilidades k requeridas é então satisfatório ou igual ao número de nós n. Então temos somente que localizar uma facilidade em cada nó para reduzir a média de distância de viagem para zero.

Assim, provamos o teorema para um caso especial de facilidade singular (k=1). A linha de aproximação similar para uma linha abaixo pode ser usada para provar um caso mais geral $(k \geq 1)$.

Prova (para uma facilidade singular, 1 mediana): supondo que a locação ótima para uma facilidade singular é um ponto x que encontra-se no link (p, q) entre os nós p e q. A distância d(x, j) entre x e algum vértice j ε N pode ser escrito como:

$$d(x, j) = Min(d(*, p) + d(p, j); d(x, q) + d(q, j))$$

Isto é, algum nó j ε N será alcançado de X ε (p, q) ou conexo a pe ou a q. Deixando agora P ser o conjunto de nós que o ponto x alcança mais eficientemente a conexão ao nó p e Q o conjunto de nós que alcança mais eficientemente a conexão ao nó q (os nós podem ser quebrados arbitrariamente). P e Q são ambos mutuamente conjuntos exclusivos cuja união é igual a N.

Assumindo agora sem menos generalidade que os mais usados são alcançados conectando mais p do que q, isto é:

$$\sum_{j \in P} h_j \geq \sum_{j \in Q} h_j$$

Se o oposto for verdadeiro, o argumento reverso é simples. Então teremos:

$$J(x) = \sum_{j \in N} h_j d(x, j)$$

$$J(x) = \sum_{j \in P} h_j (d(x, p) + d(p, j)) + \sum_{j \in Q} h_j (d(x, q) + d(q, j))$$

$$J(x) = \sum_{j \in P} h_j (d(x, p) + d(p, j)) + \sum_{j \in Q} h_j (d(x, q) - d(x, p) + d(q, j))$$

$$J(x) = d(x, p)(\sum_{J \in P} h_j - \sum_{j \in Q} h_j) + \sum_{j \in P} h_j d(p, j) + \sum_{j \in Q} h_j (d(p, q) + d(q, j))$$

Para a definição de distância d(p, j), temos:

$d(p, j) \leq d(p, q) + d(q, j)$ para todo j ε N

Usando essa definição anterior na expressão para J(x) temos:

$$J(x) \geq d(x, p)(\sum_{j \in P} h_j - \sum_{j \in Q} h_j) + \sum_{j \in P} h_j d(p, j) + \sum_{j \in Q} h_j d(p, j)$$

$$J(x) = d(x, p)(\sum_{j \in P} h_j - \sum_{j \in Q} h_j) + \sum_{j \in N} h_j d(p, j)$$

$$J(x) = d(x, p)(\sum_{j \in P} h_j - \sum_{j \in Q} h_j) + J(p)$$

Mas o termo $d(x, p)(\sum_{j \in P} h_j - \sum_{j \in Q} h_j)$ é, por associação melhor resultado ou igual a zero. Com isso concluímos que $J(x) \geq J(p)$, com contradições de associação que uma mediana está localizada no ponto interior do link (p, q). Diferentemente, temos provado que podemos pelo menos mover tão bem a facilidade de x no nó p. Isso também completa nossa teoria.

5.1.1. *O Algoritmo da Mediana Singular*

PASSO 1:

Obtenha a matriz de distâncias mínimas para os nós de G.

PASSO 2:

Multiplique a j-ésima coluna da matriz de distâncias mínimas pelo valor da demanda L_j (j=1, 2, 3, ... n) para obter a matriz $[L_j.d(i, j)]$.

PASSO 3:

Para cada linha i da matriz $[L_j.d(i, j)]$, faça a soma de todos os termos da linha. O nó que corresponde a linha com a menor soma dos termos é a localidade de 1 mediana.

Este algoritmo pode também ser usado, em princípio, para obter as k medianas para algum valor de $k \geq 1$. Somente o passo 3 deve ser modificado para proporcionar por consideração dos conjuntos das k linhas (melhor que linhas singulares).

Infelizmente as combinações e suas conseqüências requerem comparações no passo 3 – inicia-se também de muitas formas manuais ou computacionais tão logo que o número de nós, n, e o número de facilidades de localizações requeridas, k, alcançam um valor moderado.

A princípio, para n=100 e k=5, existem 75.000.000 possibilidades de combinações de 5 medianas, computacionando as distâncias de cada um com 500 comparações no passo 3. Ambos possuem uma aproximação por indução para k>1 e aproximações mais sofisticadas devem ser buscadas.

Alguns algoritmos têm sido propostos para problemas de k medianas [Revê 70, Garf 74]. Basicamente, estes algoritmos atendem eficientemente programando formulações inteiras no problema. Melhor conhecido é o conceito simples do algoritmo Heurístico [Teit 68], entretanto, ele nem sempre termina com a solução de k medianas ótimas. Descrevemos abaixo uma versão otimizada deste algoritmo, que começa encontrando uma mediana no grafo e depois incrementa o número de medianas nos passos de um time, ele começa igualando o número requerido, k(k>1). Pois o teorema de Hakimi deve ser somente de acordo com as locações nos nós. Devemos utilizar S para indicar o conjunto de nós, enquanto as medianas têm sido localizadas em algum

dado estágio na execução do algoritmo e m para indicar o número de nós em S. Durante a execução do algoritmo m será incrementado de 1 para k.

5.1.2. O Algoritmo Heurístico das Multimedianas

PASSO 1:

Faça m=1. Ache a mediana 1 do grafo G(N, A) usando o algoritmo da Mediana Singular. Faça a mediana 1 ser o nó i. Conjunto S={i}.

PASSO 2:

(Adição de Facilidade): Adiciona uma nova facilidade para o fluxo associado ao conjunto S pela escolha de uma localização entre os nós de N-S, os nós que não estão em S, que produzem o máximo possível de melhorias na função objetivo como o número de medianas incrementadas por 1. Faça m=m+1.

PASSO 3:

(Solução Otimizada): Tentando otimizar a função objetivo pela substituição de modo sistema. Tira um por vez, um dos nós em S com um nó que está em N-S. Toda vez que uma solução otimizada é obtida, use-a como uma nova solução, S, e repita o Passo 3. Quando todas as possibilidades de substituição de nós singulares para o conjunto S terem sido testadas sem otimizar a função objetivo, vá para o passo 4.

PASSO 4:

Se m=k, pare. Caso contrário vá para o passo 2.

O algoritmo Heurístico das Multimedianas é típico de um número de algoritmos Heurísticos em grafos que usam o método de substituição para otimizar as soluções iniciais.

Por exemplo, um dos mais conhecidos algoritmos para o problema do Caixeiro Viajante [Lin 73] inicia-se com um ciclo inicial e otimiza este ciclo pela substituição de um arco de um ciclo inicial com outro arco que não irá de um ciclo inicial. A solução obtida para estes algoritmos é algumas vezes referida a uma solução ótima, pois eles não podem ser otimizados substituindo algum número singular (nó, arco ou qualquer

outra coisa) no final da solução do conjunto. O algoritmo Heurístico de multimedianas pode ser facilmente modificado para ser dois (ou três, etc) soluções ótimas, por permitir a substituição no passo 3 em cima do dois (ou três, etc) nós de S por vez (em vez de exatamente um) com nós em N-S. A solução deles poderia ser ao menos boa ou melhor que uma solução ótima. Entretanto, uma poderia resolver o custo para duas soluções ótimas no algoritmo para ser consideravelmente maior para grandes problemas desde que o número de substituições dele esteja atendendo consideravelmente os incrementos do passo 3. Sendo assim, temos mostrado que uma versão da solução ótima do algoritmo em questão é efetiva. Ela é simples e fácil de programar, muito rápido e usualmente produz soluções que fecham a solução ótima.

6. Conclusão

Neste Trabalho de Iniciação Científica estudou-se os algoritmos clássicos para resolução de problemas combinatórios em grafos dando enfoque a aplicação aos (PRV), sendo implementados cinco algoritmos de grande importância que são os de Caminho Mínimo de um dado nó e todos os outros, Caminho Mínimo entre todos os Pares de Nós, Árvore de Cobrimento Mínimo, Carteiro Chinês e Problema das Medianas.

Vale salientar que os algoritmos foram construídos (ou adaptados) para execução em grafos orientados, em que supri a necessidade real devido à existência de rodovias de "mão única", de "mão dupla" com "mãos" de custos diferentes.

A idéia original seria anexar os algoritmos à um software de Sistemas de Informações Geográficas (SIG) chamado ARC-INFO do Environmental Systems Research Institute (ESRI), sendo esta desconsiderada devido a falta de informações sobre este software.

Este trabalho é inicial e possui classes em que se pode incluir novos algoritmos. Como sugestão para novos trabalhos, os algoritmos a seguir são fundamentais para estudo :

- Caixeiro Viajante
- Carteiro Chinês para Grafos Orientados com introdução da Capacidade da Mochila

Existem outros critérios interessantes que podem ser abordados e que aproximam ainda mais as simulações das situações reais. Os critérios sugeridos são:

Alteração dos custos dos arcos conforme o fluxo em determinado horário do dia, ou seja, para horário de congestionamento aumenta-se o custo e para horário de menor fluxo atribui-se o custo normal

Inclusão do consumo do veículo em questão (quilômetro por litro), assim como valor do combustível

7. Bibliografia

1. Jamsa, Ph.D. K.; Klander, L. **Programando em C/C++ "A Bíblia".** Makron Books, 1999.
2. Larson, R. C.; Odoni, A. R. **Urban Operations Research.** Prentice Hall, 1981.
3. Mizrahi, V. V. **Treinamento em Linguagem C Módulo 1.** McGraw-Hill, 1990.
4. Mizrahi, V. V. **Treinamento em Linguagem C Módulo 2.** McGraw-Hill, 1990.
5. Pappas, C. H.; Murray, W. H. **Turbo C++ Completo e Total.** McGraw-Hill & Makron Books, 1991.
6. Puccini, A. L.; Pizzolato, N. D. **Programação Linear.** Editora Livros Técnicos e Científicos, 1989.
7. Sampaio, R. M. **Estudo e Implementação de Algoritmos de Roteamento.** Trabalho de Graduação CTA/ITA, 1998.
8. Tenenbaum, A. M.; Langsam, Y.; Augestein, M. J. **Estruturas de Dados usando C.** Makron Books, 1995.

8. Anexo

8.1. Código do Algoritmo de Dijkstra

```
ROTA::dijkstra(int fonte, float **mat){  //Fonte: nó fonte escolhido; **mat: matriz de distâncias
 /*Declaração da variáveis locais:
 *dist: vetor que armazena as distâncias mínimas do nó fonte aos demais
 aux: auxilia na procura da menor distância de um nó à outro
 infinito: valor da somatória de todas as distâncias do grafo
 *cam: vetor que armazena a sequência de nós dos caminhos mínimos
 *visita: vetor que sinaliza os nós que foram visitados ou não
 atual: indice do nó sendo visitado
 i e j: contadores dos laços  */
 float *dist, aux, infinito;
 int *cam, *visita, atual, i, j;

 /*Alocação dinâmica dos Vetores dist, cam e visita*/
 dist = new float [nnos];
 cam = new int [nnos];
 visita = new int [nnos];

 /*Inicializações do algoritmo*/
 cam[fonte] = -1;
 atual = fonte;

 //Cálculo do valor infinito
 infinito=infinit(nnos,mat);

 //Atualiza cópia da matriz de distâncias com os valores infinito
 **mat=infinitmat(nnos,infinito,mat);  //Função Infinitmat atualiza matriz de acordo com
                                       critérios pré estabelecidos
```

```
//Inicializa vetores dist e visita
for(j=0; j<nnos; j++){
  dist[j] = mat[fonte][j];
  visita[j] = 0;
}
dist[fonte] = 0.0;
visita[fonte] = 1;  //Sinaliza nó fonte como visitado

/*Laço principal do algoritmo*/
for(i=0; i<nnos-1; i++){
  for(j=0; j<nnos; j++)
    if(!visita[j])  //Se nó j não foi visitado, faça:
      //Função Min(A, B) retorna o menor da comparação entre A e B
      dist[j]=Min(dist[j],dist[atual]+mat[atual][j]);
  aux = infinito;  //Atualiza aux a cada loop de i
  for(j=0; j<nnos; j++)
    //Se nó j não foi visitado e distância de i à j for a menor no momento
    if((!visita[j])&&(dist[j]<aux)){
      aux = dist[j];
      atual = j;
    }
  for(j=0; j<nnos; j++)
    //Se j é nó visitado e dist[j] é igual a dist[atual]-mat[j][atual] faça
    if((visita[j])&&(dist[j]==dist[atual]-mat[j][atual]))
      cam[atual] = j;  //Marca nó predecessor ao atual
  visita[atual] = 1;  //Sinaliza nó atual como visitado
 }

 /*Desalocação dos vetores dist, cam e visita*/
 delete [] dist;
 delete [] cam;
 delete [] visita;
}
```

8.2. CÓDIGO DO ALGORITMO DE FLOYD

```
ROTA::floyd(float **mat){  //**mat: cópia da matriz de distâncias
 /*Declaração da variáveis locais:
 **dist: matriz que armazena as distâncias mínimas entre os pares de nós
 **cam: matriz que armazena a sequência de nós dos caminhos mínimos
 infinito: valor da somatória de todas as distâncias do grafo
 atual: nó sendo processado no momento
 i e j: contadores dos laços  */
 float **dist, **cam, infinito;
 int atual, i, j;

 /*Alocação dinâmica das matrizes dist e cam
 dist=new float *[nnos];
 cam=new float *[nnos];
 for(i=0; i<nnos; i++){
  cam[i]=new float[nnos];
  dist[i]=new float[nnos];
 }

 /*Inicialização da matriz cam*/
 for(i=0; i<nnos; i++)
  for(j=0; j<nnos; j++){
   if(i==j)  //Se i for igual a j, faça:
    cam[i][j]=-1;
   else  //Caso contrário, faça:
    cam[i][j]=i;
  }
 infinito=infinit(nnos,mat);  //Cálculo de valor infinito
 //Atualiza cópia da matriz de distâncias com os valores infinito
 **mat=infinitmat(nnos,infinito,mat);  //Função Infinitmat atualiza matriz de acordo com
                                       critérios pré estabelecidos
```

```
 /*Laço principal do algoritmo*/
 for(atual=0; atual<nnos-1; atual++){
  for(i=0; i<nnos; i++)
   for(j=0; j<nnos; j++)
    //Função Min(A, B) retorna o menor da comparação entre A e B
    dist[i][j]=Min(mat[i][j],mat[i][atual]+mat[atual][j]);
  for(i=0; i<nnos; i++)
   for(j=0; j<nnos; j++)
    if(dist[i][j]!=mat[i][j])  //Se distância mínima de i à j for diferente de distância de i à j,
faça:
      cam[i][j]=cam[atual][j];
  for(i=0; i<nnos; i++)
   for(j=0; j<nnos; j++)
    mat[i][j]=dist[i][j];  //Atualiza matriz mat para ser usada em novo loop
 }

 /*Desalocação das matrizes cam e dist*/
 delete [] cam;
 delete [] dist;
}
```

8.3. CÓDIGO DO ALGORITMO DA ÁRVORE DE COBRIMENTO MÍNIMO

```
ROTA::tree(float **mat){  //**mat: cópia da matriz de distâncias
 /*Declaração da variáveis locais:
 *cam: vetor que armazena a sequência de nós dos caminhos da árvore atual
 *final: vetor que armazena a sequência de nós dos caminhos da árvore de cobrimento
mínimo
 *visit: vetor que sinaliza os nós visitados ou não
 indicei, indicej: armazena indices de nó visitado e nó não visitado respectivamente
 infinito: valor da somatória de todas as distâncias do grafo
 ok: auxiliar de tomada de decisão
 fonte: nó fonte da árvore de cobrimento mínimo
 aux: auxilia na procura da menor distância de i à j
 soma: somatória dos arcos da árvore corrente
 menor: auxilia na procura da árvore de cobrimento mínimo
 i, j, k: contadores dos laços  */
 int i, j, k, l, ok, fonte, *visit, indicei, indicej, *cam, *final;
 float infinito, aux, soma, menor;
 /*Alocação dinâmica dos vetores visit, cam, final*/
 visit = new int [nnos];
 cam = new int [nnos];
 final = new int [nnos];

 /*Inicialização de variáveis*/
 infinito = infinit(nnos,mat);  //Cálculo de valor infinito
 menor = infinito;
 for(l=0; l<nnos; l++){
  for(i=0;i<nnos;i++){
    visit[i] = 0;
    cam[i] = -1;
  }
  visit[l]=1;  //Sinaliza nó l (letra L) como visitado
  soma = 0.0;  //Atualiza soma para próximo loop
```

```
  for(k=0; k<nnos-1; k++){
    aux = infinito;  //Atualiza aux para próximo loop
    ok=0;  //Atualiza ok para próximo loop
    for (i=0;i<nnos;i++)
     for(j=0;j<nnos;j++)
      //Se nó i for um nó visitado, se distância de i à j for maior que zero, se nó j não for
um nó
        visitado e se distância de i à j for menor que aux, faça:
      if((visit[i])&&(mat[i][j]>0)&&(!visit[j])&&(mat[i][j]<aux)){
        aux=mat[i][j];  //Atualiza aux com menor valor (mat[i][j])
        indicei=i;  //Armazena nó i (visitado) que liga o nó j à árvore
        indicej=j; //Armazena nó j (não visitado) que é o novo nó da árvore
        ok=1;  //Habilita próxima condição pois foi incluso novo nó à arvore
      }
    if(ok){  //Se novo nó foi incluso, faça:
      soma + = mat[indicei][indicej];  //Somatório do tamanhos dos arcos da árvore
      cam[indicej]=indicei;  //Marca nó predecessor do novo nó da árvore
      visit[indicej]=1;
    }
  }
  if((soma)&&(soma<menor)){  //Se soma for maior que zero e menor que menor
    menor = soma;  //Atualiza menor para próximo loop de l
    fonte = l;  //Atualiza fonte com nó fonte da menor árvore calculada
    for(i=0; i<nnos; i++)
     final[i] = cam[i];  //Atualiza o vetor final com o vetor de nós predecessores da
menor árvore
                          calculada até o momento
  }
 }
 /*Desaloca vetores visit, cam e final*/
 delete [] visit;
 delete [] cam;
 delete [] final;
}
```

8.4. CÓDIGO DO CARTEIRO CHINÊS

```
ROTA::cchines(float **mat){
        int i, j, in, out, aux, auxi, auxj,*polarity, *uf, *vf;
        float infinito, max, k, **tmp, **dist, **cust, *oferta, *demanda, *u, *v, **x;

        polarity=new int [nnos];
        uf=new int [nnos];
        vf=new int [nnos];
        oferta=new float [nnos];
        demanda=new float [nnos];
        u=new float [nnos];
        v=new float [nnos];
        dist=new float *[nnos];
        cust=new float *[nnos];
        x=new float *[nnos];
        for(i=0; i<nnos; i++){
                dist[i]=new float [nnos];
                cust[i]=new float [nnos];
                x[i]=new float [nnos];
        }

        //calcula as polaridades dos n¢s
        for(i=0; i<nnos; i++){
                in=0;
                out=0;
                for(j=0; j<nnos; j++){
                        if(mat[i][j]>0) in+=1;
                        if(mat[j][i]>0) out+=1;
                }
                polarity[i]=in-out;
        }
        for(i=0; i<nnos; i++)
```

```
//inicializa vetor oferta, demanda e matriz custo m¡nimo com valores nulos
for(i=0; i<nnos; i++){
        oferta[i]=0.0;
        demanda[i]=0.0;
        for(j=0; j<nnos; j++)
                cust[i][j]=0.0;
}

//calcula valores de oferta e demanda
for(i=0; i<nnos; i++)
        for(j=0; j<nnos; j++){
                if(mat[i][j]>0.0)
                        oferta[i]+=mat[i][j];
                if(mat[j][i]>0.0)
                        demanda[i]+=mat[j][i];
        }

//calcula matriz de custos m¡nimos entre todos os pares de nos
tmp=floyd(1,mat);

//calcula um valor "infinito"
infinito=infinit(nnos, mat);

//inicializa (com adapta‡äeas) a matriz de custo m¡nimo
for(i=0; i<nnos; i++)
        for(j=0; j<nnos; j++){
                if(i==j)
                        dist[i][j]=infinito;
                else
                        dist[i][j]=tmp[i][j];
        }
```

```
//calcula a m,todo do custo m¡nimo
do{
        max=infinito+1;
        for(i=0; i<nnos; i++)
                for(j=0; j<nnos; j++)
                        if((oferta[i]>0.0)&&(demanda[j]>0.0)&&(cust[i][j]==0.0))
                                if(dist[i][j]<max){
                                        max=dist[i][j];
                                        auxi=i;
                                        auxj=j;
                                }
        cust[auxi][auxj]=(Menor(oferta[auxi],demanda[auxj]));
        oferta[auxi]-=cust[auxi][auxj];
        demanda[auxj]-=cust[auxi][auxj];
        k=0.0;
        for(i=0; i<nnos; i++)
                if((oferta[i]>0.0)||(demanda[i]>0.0)){
                        k=1.0;
                        break;
                }
}while(k);

//busca vari vel b sica que mais "aparece"
max=0.0;
for(i=0;i<nnos;i++){
        k=0.0;
        uf[i]=0;
        vf[i]=0;
        u[i]=0.0;
        v[i]=0.0;
        for(j=0;j<nnos;j++)
                if(cust[i][j]>0.0)
                        k++;
        if(k>max){
```

```
                max=k;
                aux=i;
        }
}

//arbitra valor nulo para a vari vel b sica
u[aux]=0.0;
uf[aux]=1;

//calcula demais vari veis b sicas
do{
        for(i=0;i<nnos;i++)
                for (j=0;j<nnos;j++)
                        if(cust[i][j]>0.0){
                                if((uf[i]==1)&&(vf[j]==0)){
                                        vf[j]=1;
                                        v[j]=dist[i][j]-u[i];
                                }
                                if((vf[j]==1)&&(uf[i]==0)){
                                        uf[i]=1;
                                        u[i]=dist[i][j]-v[j];
                                }
                        }
        k=0.0;
        for(i=0; i<nnos; i++)
                for(j=0; j<nnos; j++)
                        if(cust[i][j]>0.0)
                                if((uf[i]==0)||(vf[j]==0)){
                        k=1.0;
                        break;
                }
}while(k);

//calculo das variaveis nao basicas
```

```
        for(i=0;i<nnos;i++)
                for (j=0;j<nnos;j++)
                        if(cust[i][j]==0.0)
                                x[i][j]=dist[i][j]-u[i]-v[j];

        //busca da menor variavel nao basica
        k=infinito;
        for(i=0;i<nnos;i++)
                for (j=0;j<nnos;j++)
                        if(cust[i][j]==0.0)
                                if(x[i][j]<k){
                                        k=x[i][j];
                                        auxi=i;
                                        auxj=j;
                                }
}
```