O Problema do Carteiro Chinês

Trabalho de conclusão de curso
Universidade de São Paulo
Bacharelado em Ciência da Computação

Gabriel Fernandes de Oliveira
Orientador: Carlos Eduardo Ferreira

São Paulo, 2020

Resumo

O Problema do Carteiro Chinês, enunciado pelo matemático chinês Meigu Guan em 1962, consiste em encontrar um trajeto fechado de custo mínimo que percorre todas arestas de um grafo ao menos uma vez.

Essa generalização do problema dos circuitos eulerianos possui muitas aplicações na área de logística, podendo ser usado, por exemplo, para reduzir a distância percorrida na rota de carteiros, caminhões de lixo, ou mesmo de veículos de remoção de neve.

Este trabalho tem como objetivos explicar e implementar a solução do Problema do Carteiro Chinês e de algumas de suas variações (*e.g.* o Problema aplicado a grafos mistos e o Problema do Carteiro Rural).

Palavras-chave: grafos, circuito euleriano, problema do carteiro chinês

Abstract

The Chinese Postman Problem, originally studied by the Chinese mathematician Meigu Guan in 1962, consists of finding a minimum cost closed path that passes every edge in a graph at least once.

This generalization of the eulerian circuit problem has many applications in logistics. Some of its applications are optimizing routes for mail delivery, waste collection, and even snow plowing.

The goals of this project are to explain and implement solutions for the Chinese Postman Problem and some of its variations (e.g. Mixed and Rural Postman Problem).

Keywords: graphs, eulerian circuit, chinese postman problem

Sumário

Capítulo 1

Grafos eulerianos

1.1 As sete pontes de Königsberg

O problema das sete pontes de Königsberg foi descrito e solucionado pelo matemático Leonhard Euler em 1736. O problema consistia em decidir se seria possível traçar no mapa de Königsberg um trajeto que percorresse cada uma de suas 7 pontes uma única vez, sem repetições.

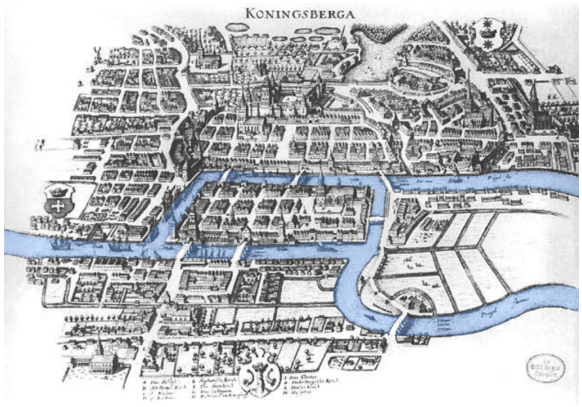


Figura 1.1: Representação das sete pontes de Königsberg

Euler resolveu esse problema do seguinte modo: Primeiramente, ele identificou cada uma das massas de terra do mapa com as letras A, B, C e D.

Em seguida, ele definiu que um trajeto nesse mapa seria descrito por uma sequência dessas letras: por exemplo, “ACD” indicaria o trajeto que se inicia na massa de terra A, move-se para a massa de terra C, usando uma das pontes, e termina na massa D.

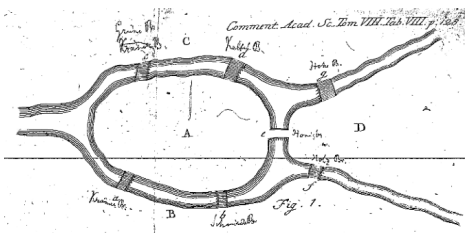


Figura 1.2: Representação de Euler

Euler então começou a definir algumas restrições, assumindo que o problema possuiria alguma solução:

Como o trajeto final deverá passar por todas as 7 pontes exatamente uma vez, isso implica que a sequência de letras que o representa deverá ter tamanho 8.

Além disso, como a massa de terra A possui 5 pontes, necessariamente a letra A aparecerá exatamente 3 vezes na sequência. A massa B, C e D, no entanto possuem 3 pontes, portanto, suas letras correspondentes deverão aparecer apenas 2 vezes na sequência.

Chegamos assim a um absurdo, pois provamos que a sequência de letras que representa uma solução deveria ter tamanho 8 e 9. Provando assim que não existe um

trajeto como o pedido no enunciado do problema.

Essa observação que, aos olhos de hoje, parece muito simples, teve um impacto profundo nas áreas da Matemática e Computação. A modelagem de Euler, tratando as massas de terra e pontes de forma abstrata foi absolutamente inovadora e é o embrião da área da teoria dos grafos.

1.2 Grafos não direcionados

Antes de voltar ao problema de Euler, realizaremos algumas definições de teoria dos grafos.

Define-se como **passeio** em um grafo uma sequência finita não vazia $P = \{v_0, v_1, \ldots, v_k\}$, cujos termos são vértices v_i tais que, para todo i, $0 \leq i < k$, os vértices v_i e v_{i+1} são ligados por uma aresta. Os vértices v_0 e v_k são a origem e o término de P, respectivamente; e os vértices $v_1, v_2, \ldots, v_{k-1}$ são chamados vértices internos de P.

Uma **trilha** é um passeio sem arestas repetidas enquanto que um **caminho** é um passeio sem vértices repetidos. Definimos o comprimento de um passeio como o número de arestas do mesmo.

Um passeio é considerado **fechado** se sua origem e término são iguais.

Uma trilha fechada é um **circuito**.

Como homenagem às contribuições de Euler, definiu-se **trilha euleriana** como uma trilha que passa por todas arestas de um grafo, **circuito euleriano** como um circuito que passa por todas arestas de um grafo e **grafo euleriano** como um grafo que possui um circuito euleriano.

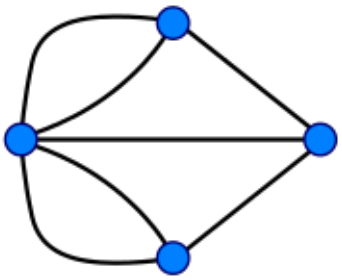


Figura 1.3: Grafo de Königsberg

Euler modelou essa área da cidade como um grafo, representado na figura 1.3, tratando as pontes como arestas e as massas de terra como vértices.

A partir de tal modelagem e das definições feitas, o problema das pontes de Königsberg consiste, em definir se o grafo que representa a cidade possui ou não uma trilha euleriana.

Apesar de ter recebido grande parte do crédito histórico pelas suas implicações, Euler não provou que qualquer grafo conexo com vértices de grau par é euleriano, essa prova só foi publicada mais de cem anos depois, em 1873, por Carl Hierholzer [Hie73], no que se tornou o conhecido Teorema de Euler.

Teorema 1.2.1 (Teorema de Euler ou de Euler-Hierholzer)**.** *Um grafo é euleriano se, e somente se, é conexo e todos seus vértices possuem grau par.*

Para suportar a prova de tal teorema, primeiramente apresenta-se o seguinte lema:
Seja $\delta(G)$ o menor grau de um vértice pertencente a G.

Lema 1. *Se G é um grafo tal que $\delta(G) \geq 2$, então G possui um circuito.*

Demonstração. Seja P um caminho de comprimento maximal pertencente a G, denominamos v um dos extremos de P.

Como $\delta(G) \geq 2$, então v possui ao menos dois vértices adjacentes u, w.

Por P ser maximal é impossível que v possua uma adjacência que o ligue a um vértice não pertencente a P. Implicando assim que $u, w \in P$.

Consequentemente garante-se a existência de um circuito em G, basta tomar as arestas vu, wv e o subcaminho de P que liga u a w, fechando assim um circuito.

Demonstra-se assim que se $\delta(G) \geq 2$, então G deverá possuir ao menos um circuito. □

Provado tal lema, podemos agora provar o teorema 1.2.1:

Demonstração. Seja G o grafo em questão.

($\Rightarrow$) Começamos provando que se um grafo é euleriano então todos seus vértices possuem grau par.

Seja T um circuito euleriano, cuja existência é garantida já que G é euleriano. Analisaremos o grau de um vértice qualquer v de G.

Se v não for um vértice extremo de T, então sempre que o mesmo aparecer em T ele deverá ser precedido e sucedido de arestas, indicando assim que v deverá possuir um grau par.

Do contrário, se v for o vértice extremo de T ele necessariamente possui duas arestas, uma ligando-o ao segundo vértice do circuito, e outra o ligando ao último vértice de T que não é v. Além disso, cada aparição de v como vértice interno de T contabiliza mais duas arestas ao vértice em questão, de modo que v possuirá um grau par ao final desta contagem.

Sendo assim, se G é euleriano, então todos seus vértices (extremos ou não) possuem grau par, como queríamos demonstrar.

($\Leftarrow$) Agora, prova-se por indução no número de arestas que se G for conexo e se todos seus nós possuem grau par, então ele é euleriano.

O caso base da indução é quando não há arestas em G. O único grafo conexo que respeita tal condição é o grafo que possui apenas um vértice v. Neste exemplo, $\{v\}$ é o circuito euleriano do grafo.

A hipótese de indução é que todo grafo simples, conexo, que possui até $k-1$ arestas e cujos vértices têm grau par é euleriano. Seja G um grafo conexo, de vértices de grau par e que possua $k \geq 1$ arestas, prova-se a seguir que G também deverá ser euleriano.

Como G é conexo e possui ao menos uma aresta, vale que $\delta(G) \geq 1$. Além disso, todos nós de G têm grau par então podemos afirmar que $\delta(G) \geq 2$. Sendo assim, pelo lema 1, G deverá possuir um circuito C.

Se C possui todas arestas de G, então C é um circuito euleriano do grafo, finalizando a prova.

Do contrário, aplicamos o seguinte procedimento para construir um circuito euleriano $\mathcal{C}$ de G:

Retira-se de G as arestas pertencentes a C, resultando assim em um grafo G'.

Possivelmente G' será desconexo, por isso definimos que G' será a união de k componentes conexas $G'_1, G'_2, \ldots, G'_k$ disjuntas entre si.

O grau dos vértices dessas componentes G'_i deverá ser par, já que, ao retirar todas as arestas de um circuito do grafo G, diminuímos o grau de um vértice qualquer v em duas vezes o número de aparições do mesmo no circuito. mantendo assim a paridade dos graus.

Além disso, cada componente conexa de G' possuirá uma quantidade de arestas menor do que k. Portanto, pela hipótese da indução, cada uma dessas componentes deverá possuir um circuito euleriano próprio. Chamaremos de C_i o circuito euleriano da componente conexa G'_i.

Ao longo do algoritmo, os circuitos C_i serão adicionados, um a um, ao circuito resultante $\mathcal{C}$. Definimos $\mathcal{T}$, como o conjunto dos circuitos eulerianos C_i que ainda não fazem parte de $\mathcal{C}$. Inicialmente, $\mathcal{T}$ é igual ao conjunto $\{C_1, C_2, \ldots, C_k\}$.

$$C = \{v, v_2, \ldots, v_n, v\}$$

Para cada vértice u de C devemos realizar as seguintes verificações:

Montando um circuito euleriano $\mathcal{C}$ usando C e $\mathcal{T}$

Adicionam-se um a um os vértices de C em $\mathcal{C}$.
Seja u o próximo vértice de C a ser adicionado. Enquanto houver um circuito euleriano $C_i \in \mathcal{T}$ do qual u faz parte, fazemos o seguinte:

1. Representamos C_i como:

$$C_i = \{u, u_2, \ldots, u_l, u\}$$

2. Adicionamos ao final de $\mathcal{C}$ o circuito C_i como representado, exceto pelo primeiro vértice u.

3. Removemos de $\mathcal{T}$ o circuito C_i, indicando que o mesmo já foi adicionado a $\mathcal{C}$.

Deste modo as arestas do circuito C_i são adicionadas a $\mathcal{C}$, removidas de $\mathcal{T}$, e $\mathcal{C}$ continua sua construção como um passeio válido.
Repetem-se então as mesmas verificações para os próximos vértices de C.

Ao final desse procedimento, o conjunto $\mathcal{T}$ deverá ser vazio, já que toda trilha C_i possui pelo menos um vértice em C. Além disso, toda trilha C_i deverá ter sido adicionada à $\mathcal{C}$ uma única vez, já que logo após adicionar uma trilha à $\mathcal{C}$ já a removíamos de $\mathcal{T}$, impedindo que ela fosse adicionada outra vez na trilha euleriana final.

Comprovado o passo da indução, finalizamos a prova do Teorema de Euler por indução.

□

Corolário 1. *Um grafo possui uma trilha euleriana se, e somente se, é conexo e possui apenas zero ou dois vértices de grau ímpar.*

Demonstração. Desconsideram-se aqui grafos que possuam vértices de grau 0. Se um

grafo não é conexo, então é impossível encontrar uma trilha euleriana pro mesmo, já que dependendo de qual vértice a trilha comece, existirão sempre arestas que não serão alcançáveis.

Seja, portanto, G um grafo conexo qualquer.

Analisaremos a quantidade de vértices de grau ímpar de G:

1. G não possui vértices de grau ímpar. Neste caso, G possui, segundo o teorema 1.2.1, um circuito euleriano, e portanto, uma trilha euleriana fechada.

2. G possui apenas um vértice de grau ímpar. Este caso é impossível, já que a soma do grau de todos vértices deve ser par.

3. G possui dois vértices de grau ímpar.

   Sejam u e v os únicos vértices de G que possuem grau ímpar. Define-se um grafo G' igual a G com uma aresta artificial acrescida entre os vértices u e v. Devido à nova aresta uv todos vértices de G' possuirão grau par.

   Além disso, vale que G' é conexo, pois faz parte da premissa que o grafo original G era conexo.

   Sendo assim, podemos aplicar o teorema 1.2.1, provando a existência de um circuito euleriano G', que chamaremos de C. Por ser euleriano, C deverá percorrer a aresta artificial uv. Representando C com u e v lado a lado, do seguinte modo:

   $$C = \{u, v, w_1, w_2, \ldots, w_k, u\}$$

   Tome, agora, T igual ao circuito C sem seu vértice inicial:

   $$T = \{v, w_1, w_2, \ldots, w_k, u\}$$

   Tal procedimento retira do circuito C a aresta artificial uv, transformando-o em uma trilha T, que percorre todas arestas de G‘ exceto uv, sendo portanto uma trilha euleriana do grafo G.

4. G possui três ou mais vértices de grau ímpar.

   Assuma que existe uma trilha euleriana T para G. Neste caso, como pelo menos 3 vértices possuem grau ímpar, necessariamente existirá um vértice v que não é nem o primeiro nem o último vértice de T. Isso implica que todas aparições de v em T são internas ao caminho, ou seja, toda aparição de v será precedida e sucedida de arestas ligadas a v. Como estamos tratando de uma trilha euleriana, sabemos que todas arestas adjacentes a v estão presentes em T uma única vez. Mas como todas arestas de v devem aparecer em pares (precedendo e sucedendo v), isso implica que o grau de v deverá ser par. Contradizendo a premissa.

   Por essa contradição provamos que G não possuirá trilha euleriana se tiver três ou mais vértices de grau ímpar.

□

1.3 Grafos direcionados

Até então tratamos apenas de grafos não direcionados, mas podemos expandir esses mesmos conceitos para grafos direcionados, os **digrafos**.

Um grafo é direcionado quando suas arestas são orientadas, chamaremos de **arcos** tais arestas.

Numa analogia com trânsito, uma aresta é como uma estrada de mão dupla, pode-se percorrê-la nos dois sentidos, enquanto isso, um arco é como uma rua de mão única, em que apenas um sentido é permitido.

Quanto tratamos de grafos direcionados (ou digrafos), é necessário refinar a noção de grau de um vértice: Por isso, denominamos **grau de saída** de um vértice v, $\delta^-(v)$, como o número de arcos que têm origem em v, e **grau de entrada**, $\delta^+(v)$, como o número de arcos que têm seu destino em v.

Definimos como **fortemente conexo** um digrafo que possui um caminho entre todo par de vértices, e como **fracamente conexo** um digrafo que possui para todo par de vértices u e v um caminho de u a v ou de v a u.

Dadas as devidas definições, apresentamos agora o teorema de Euler para o caso de digrafos:

Teorema 1.3.1 (Teorema de Euler para digrafos)**.** *Seja G um digrafo fortemente conexo. G é euleriano se, e somente se, todos seus vértices têm valores de grau de entrada e saída iguais, ou seja, se para todo vértice v de G vale que $\delta^+(v) = \delta^-(v)$.*

Demonstração. ($\Rightarrow$) Seja G um digrafo euleriano e C o circuito euleriano do mesmo.

Vamos assumir, contrariando o teorema, que G possui um vértice v que tem valores diferentes de grau de entrada e saída.

Assumimos, sem perda de generalidade, que $\delta^-(v) > \delta^+(v)$.

Como C é euleriano, todo arco que sai de v aparece uma vez no circuito C, então v aparece $\delta^-(v)$ vezes em C.

Porém, precedendo toda aparição de v deverá ter um arco que entra em v. Como o número de arcos que entram em v é menor que os que saem, se torna impossível que isso seja verdade sem que C possua repetição de arcos. Contradizendo assim a condição inicial de que C é um circuito euleriano.

Por absurdo, provamos que todos vértices de um digrafo euleriano devem possuir o mesmo valor de grau de entrada e saída.

($\Leftarrow$) Provaremos a volta por indução no número de arcos de G. A prova a seguir é similar à utilizada na prova do teorema 1.2.1 caso de grafos não direcionados.

O caso base desta indução é o digrafo G conexo sem arcos. Neste caso G consistirá de um único vértice v, sendo assim $\{v\}$ será um circuito euleriano válido.

A hipótese de indução é que todo digrafo fortemente conexo, que possui vértices com grau de entrada e saída iguais e que possui até $k-1$ arcos é euleriano.

Seja G, portanto, um digrafo fortemente conexo, em que todos vértices têm grau de entrada igual ao de saída mas que possui $k \geq 1$ arcos.

Inicialmente provaremos que G deve possuir um caminho fechado:

Lema 2. *Todo digrafo G fortemente conexo com igualdade de grau de entrada e saída para todos vértices possui um caminho fechado.*

Demonstração. Tome um caminho maximal P de G, sendo v o último vértice de tal caminho e vw um arco que sai de v.

Como P é maximal, podemos concluir w já deve pertencer a P, fechando assim ao menos um caminho fechado, formado pelo arco vw e o subcaminho de w a v em P. □

Provada a existência de um caminho fechado C em G temos dois casos a analisar:

1. C **percorre todos arcos de** G. Neste caso, C é o circuito euleriano, finalizando a prova.

2. C **não percorre todos arcos de** G.

   Definimos então G' como o grafo G sem os arcos pertencentes a C.

   Como estamos retirando do grafo G um caminho fechado, todo vértice em C perderá apenas um arco de entrada e de saída, mantendo em G' a característica de igualdade do grau de entrada e saída dos vértices.

   Porém, não é garantido que G' é fortemente conexo.

   Seja G a união de k componentes fortemente conexas $G'_1, G'_2, \ldots, G'_k$ disjuntas entre si.

   Sabemos que cada componente G'_i possuirá uma quantidade de arcos menor que k. Portanto, pela hipótese de indução definida. cada componente de G' é euleriana. Chamaremos de C_i o circuito euleriano da componente G'_i.

   Mostraremos agora um algoritmo que construirá um circuito euleriano $\mathcal{C}$ de G a partir de C e dos circuitos eulerianos C'_i:

   Definimos $\mathcal{T}$, como o conjunto dos circuitos eulerianos C_i que ainda não fazem parte de $\mathcal{C}$. Inicialmente, $\mathcal{T}$ é igual ao conjunto $\{C_1, C_2, \ldots, C_k\}$.

   Para cada vértice u de C devemos realizar as seguintes verificações:

   > Começamos adicionando u ao circuito euleriano $\mathcal{C}$ que estamos construindo. Enquanto houver um circuito euleriano C_i pertencente a $\mathcal{T}$ do qual u faz parte, fazemos o seguinte:
   >
   > (a) Representamos C_i como:
   >
   > $$C_i = \{u, w, \ldots, v, u\}$$
   >
   > (b) Adicionamos ao final de $\mathcal{C}$ o circuito C_i como representado, exceto pelo primeiro vértice u.
   >
   > (c) Removemos de $\mathcal{T}$ o circuito C_i, indicando que o mesmo já foi adicionado a $\mathcal{C}$.
   >
   > Repetem-se então as mesmas verificações para os próximos vértices de C.

Ao final desse procedimento, o conjunto $\mathcal{T}$ deverá ser vazio, já que todo circuito C_i possui pelo menos um vértice em C. Além disso, todo circuito C_i deverá ter sido adicionado a $\mathcal{C}$ uma única vez, já que logo após adicionar um circuito a $\mathcal{C}$ já o removíamos de $\mathcal{T}$, impedindo que ele fosse adicionado outra vez no circuito euleriano final.

Como é possível realizar a construção de um circuito euleriano $\mathcal{C}$ provamos que qualquer digrafo fortemente conexo com vértices possuindo valores iguais de grau de entrada e saída é euleriano.

□

Corolário 2. *Um digrafo G possui uma trilha euleriana se, e somente se, é fracamente conexo, possui no máximo um par de vértices u, v tal que $\delta^-(v) - \delta^+(v) = 1$, $\delta^+(u) - \delta^-(u) = 1$ e todos outros vértices w de $V(G) \setminus \{u, v\}$ possuem $\delta^+(w) = \delta^-(w)$.*

Demonstração. ($\Rightarrow$) Começaremos provando que G deverá ser fracamente conexo.

Se um digrafo G possui uma trilha euleriana T, podemos derivar um caminho entre qualquer par de vértices u, v a partir de T, como mostra-se a seguir:

Como T é euleriana, ela deve conter os vértices u e v, portanto podemos representar T de um dos seguintes modos:

$$T = \{w_1, \ldots, w_i = u, w_{i+1} \ldots, w_{j-1}, w_j = v, \ldots\}$$

$$T = \{w_1, \ldots, w_i = v, w_{i+1} \ldots, w_{j-1}, w_j = u, \ldots\}$$

Para ambos casos, a subtrilha de $w_i \ldots w_j$ representa uma trilha entre os vértices u e v. A partir de uma trilha entre quaisquer dois vértices, podemos derivar um caminho, seguindo o seguinte método:

Método para derivar um caminho de um passeio qualquer
Seja P um passeio que liga dois vértices quaisquer, mostraremos como construir um caminho ligando esses mesmos vértices a partir de P.
Como P é um passeio, possivelmente ele percorre um mesmo vértice mais que uma vez. Do contrário, já podemos considerar P um caminho, finalizando o método.
Enquanto houver um vértice v percorrido mais que uma vez por P executa-se o seguinte passo:
Retiramos de P todos os vértices entre a primeira e a última aparição do vértice v, mantendo apenas tal vértice:
Portanto um passeio P qualquer com repetição do vértice v, como o seguinte:

$$P = \{u_1, \dots u_i, v, \dots, v, u_j, \dots u_n\}$$

Passa a ser:

$$P = \{u_1, \dots u_i, v, u_j, \dots u_n\}$$

P continua sendo um passeio, já que é garantido que u_i liga-se a v e v liga-se a u_j, porém agora P possui apenas uma aparição do vértice v.
Repete-se tal passo até que P não percorra um mesmo vértice duas vezes, se tornando assim, por definição, um caminho.

Esse procedimento pode ser utilizado para encontrar um caminho entre quaisquer dois vértices a partir de T. Como T é euleriano, por definição ele passa por todos vértices de G, provando-se assim que G é fracamente conexo, isto é, a partir de T gera-se um caminho entre qualquer par de vértices de G.

Basta, portanto, provar que G possui todos os vértices w tal que $\delta^+(w) = \delta^-(w)$ ou que existe um par de vértices u, v tal que $\delta^+(u) - \delta^-(u) = 1$, $\delta^-(v) - \delta^+(v) = 1$ e $\delta^+(w) = \delta^-(w)$ para todo $w \in V(G) \setminus \{u, v\}$.

Analisaremos dois casos em relação à trilha euleriana T de G:

- Se T é um circuito:

  $$T = \{v, w_1, w_2, \dots, w_n, v\}$$

  O vértice v é o vértice inicial e final de T, mas também pode estar presente internamente em T. Considere os vértices w_i como vértices diferentes de v, internos a T.

  Todo vértice w_i possui, em cada uma de suas aparições em T, um vértice que o precede e um vértice que o sucede, consequentemente em cada aparição de w_i em T, dois arcos ligados a w_i podem ser contabilizados: um entrando em w_i e outro saindo do mesmo. Isso implica que os arcos que entram e saem de um vértice interno a T são sempre contabilizados em pares, mantendo assim a igualdade $\delta^+(w_i) = \delta^-(w_i)$.

  Analisaremos o vértice v a parte:

  No início de T, há uma aparição de v em que contabilizamos uma única aresta saindo de v, o que adiciona uma unidade ao grau de saída do mesmo $\delta^-(v)$.

As aparições de v como vértice interno de T contribuem igualmente para o grau de entrada e saída do mesmo, similarmente ao que ocorre com os vértices w_i.

No fim de T, há outra aparição de v, que contabiliza uma única aresta entrando em v, adicionando uma unidade ao grau de entrada do vértice $\delta^+(v)$.

Finalmente, nota-se que apesar de v ser um vértice extremo, seus graus de saída e entrada também se igualam, já que v é tanto o vértice inicial quanto o vértice final de T.

Provando assim que todos vértices de G possuem grau de saída e entrada iguais, satisfazendo a implicação do corolário.

- Devemos analisar agora o caso em que T não é um circuito.

$$T = \{u, w_1, w_2, \ldots, w_n, v\}$$

A diferença neste caso é que os vértices extremos são diferentes. O vértice u, do início de T, possuirá exatamente uma unidade a mais de grau de saída em relação ao seu grau de entrada $\delta^-(u) - \delta^+(u) = 1$, enquanto que v, o vértice final de T, possuirá uma unidade a mais de grau de entrada em relação ao grau de saída, $\delta^+(v) - \delta^-(v) = 1$. Todos vértices w_i diferentes de u e v possuirão um valor igual de grau de entrada e saída, já que são apenas vértices internos de T.

Este caso também é valido, condizendo com a implicação do corolário.

($\Leftarrow$) Seja G um digrafo fracamente conexo, com vértices u, v tal que $\delta^+(u) - \delta^-(u) = 1$, $\delta^-(v) - \delta^+(v) = 1$ e $\delta^+(w) = \delta^-(w)$ para todo $w \in V(G) \setminus \{u, v\}$, provaremos que deverá existir uma trilha euleriana em G.

Seja G' uma cópia do grafo G em que adiciona-se um arco de u a v. A adição de tal arco aumenta o grau de saída de u e o grau de entrada de v em uma unidade, fazendo com que $\delta^+(u) = \delta^-(u)$ e $\delta^+(v) = \delta^-(v)$.

Deste modo, todos vértices de G' possuem o mesmo grau de entrada e saída e G' continua sendo fracamente conexo.

Provaremos inicialmente que G' deverá ser fortemente conexo. Sejam w_i, w_j dois vértices quaisquer de G', como este grafo é fracamente conexo podemos assumir, sem perda de generalidade, que existe ao menos um caminho de w_i a w_j. Provaremos que também deverá existir um caminho no sentido inverso, de w_j a w_i.

Seja T uma trilha maximal que percorre um dos caminhos de w_i a w_j:

$$T = \{w_1, w_2, \ldots, w_i, \ldots, w_j, \ldots w_n\}$$

Como T é maximal, podemos afirmar que todos os arcos que saem do vértice w_n são percorridos por T, já que do contrário T não seria maximal.

Além disso, como todos os vértices possuem grau de saída e entrada iguais em G', deve valer que $w_1 = w_n$, já que do contrário $\delta^-(w_1) - \delta^+(w_1) = 1$ e $\delta^+(w_n) - \delta^-(w_n) = 1$. Consequentemente podemos reescrever a trilha T de outro modo, que chamaremos de T', evidenciando a existência de uma trilha de w_j a w_i:

$$T' = \{w_j, w_{j+1}, \ldots, w_n = w_1, w_2, \ldots, w_i\}$$

A partir da trilha T' é possível derivar um caminho de w_j a w_i, como realizado anteriormente na demonstração deste corolário.

Prova-se assim que todo par de vértices w_i, w_j possui um caminho em G' de w_i a w_j e de w_j a w_i. G', pela definição, é um grafo fortemente conexo.

Além disso, pelo teorema 1.3.1, G' é euleriano, possuindo assim um circuito euleriano C:

$$C = \{w_1, w_2, \ldots, w_k = u, w_{k+1} = v, \ldots, w_n, w_1\}$$

Podemos reescrever C do seguinte modo, tornando o arco artificial, de u a v, o último arco percorrido em C:

$$C = \{w_{k+1} = v, \ldots, w_n, w_1, w_2, \ldots, w_{k-1}, w_k = u, w_{k+1} = v\}$$

A partir de C podemos derivar uma trilha T, removendo apenas o arco artificial de C:

$$T = \{w_{k+1} = v, \ldots, w_n, w_1, w_2, \ldots, w_{k-1}, w_k = u\}$$

Ao remover de C o único arco que não pertencia ao grafo original G, criamos uma trilha euleriana T em relação a G, provando assim a volta do corolário.

□

1.4 Algoritmo de Hierholzer

O algoritmo de Hierholzer foca em encontrar e unir um conjunto de circuitos C_1, C_2,- $\ldots, C_k$, tal que não existe aresta pertencente a dois circuitos diferentes, e que toda aresta pertence ao menos a um circuito. Tal união gera um grafo conexo euleriano.

O algoritmo desenvolvido se baseia em percorrer o grafo euleriano com uma busca em profundidade, apagando todas as arestas percorridas nesta busca. Quando o vértice atual da busca não possui mais arestas para se percorrer, o mesmo é adicionado ao começo da trilha euleriana que está sendo construída.

Segue o pseudo código do algoritmo de Hierholzer para encontrar uma trilha euleriana dado um grafo euleriano.

Algoritmo 1 Solução de Hierholzer

```
1: função EULER_HIERHOLZER(u)
2:     para cada v em adj[u] faça
3:         apaga a aresta uv
4:         EULER_HIERHOLZER(v)
5:     Insere u no início de trilha_euleriana
```

Digamos que a função apresentada seja chamada com um vértice inicial v, pertencente a um grafo euleriano G. A busca segue visitando vértices e apagando arestas até chegar no primeiro vértice em que não existem mais arestas para se percorrer.

Em grafos que possuem um circuito euleriano, tal vértice será o próprio vértice inicial v, e podemos chamar o circuito formado de C_1. Enquanto que, grafos que

possuem apenas uma trilha euleriana, tal vértice será aquele cujo grau de entrada é maior que o grau de saída.

Em ambos casos, após encontrar tal vértice sem arestas, o algoritmo deve continuar percorrendo o grafo em uma busca em profundidade, ou seja voltando por todos os vértices percorridos anteriormente, procurando algum que ainda possua arestas não apagadas, para que ela possa encontrar um novo circuito C_i e juntar o mesmo ao circuito, ou trilha, inicial, unindo ambas estruturas.

Apesar da união dos circuitos ser um procedimento complexo e possivelmente custoso, não é necessário manter o resultado de todas uniões a serem realizados durante o algoritmo, já que apenas estamos interessados no resultado final, a trilha euleriana. Para manter tal trilha, basta adicionar a uma pilha todo vértice encontrado na busca quando o mesmo não possui mais arestas a serem percorridas.

Pode-se observar tal procedimento em prática com o seguinte exemplo:

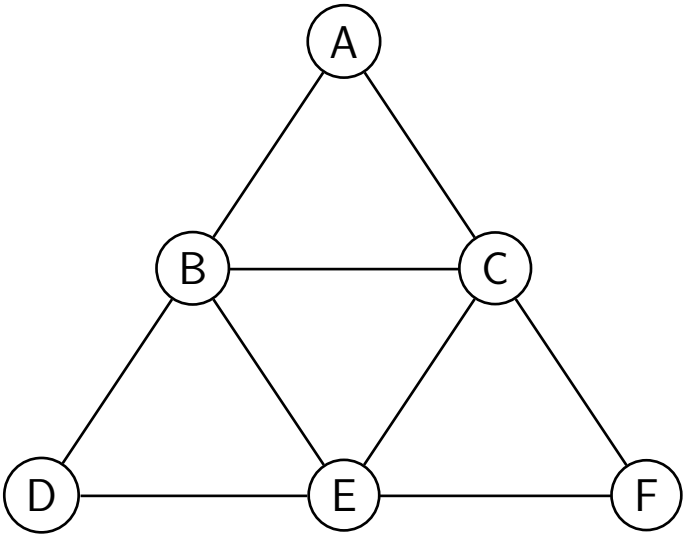


Figura 1.4: Grafo G, foco do exemplo do algoritmo de Hierholzer

O grafo apresentado possui uma grande quantidade de circuitos, por isso encontrar uma partição do mesmo em circuitos $C_1, C_2, \ldots, C_k$ não é uma tarefa simples. Uma possibilidade para este grafo é tomar $C_1 = \{A, B, C, A\}, C_2 = \{B, D, E, B\}, C_3 = \{C, E, F, C\}$, outra possibilidade é separar G em dois circuitos apenas $C_1 = \{A, B, D, E, F, C, A\}, C_2 = \{B, E, C, B\}$.

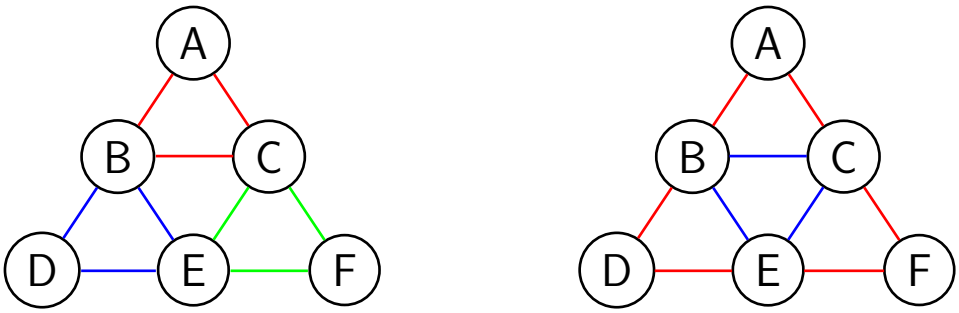


Figura 1.5: Dois exemplos de partição de G em circuitos. Em vermelho representou-se C_1, em azul C_2 e em verde C_3.

Felizmente, não é necessário particionar o grafo nos circuitos para se determinar o circuito euleriano de G.

Simularemos nos próximos passos a busca em profundidade que o algoritmo realiza em G, tomando o vértice A como o vértice inicial:

Neste exemplo, digamos que o trajeto inicial percorrido na busca é $\{A, B, C, A\}$.

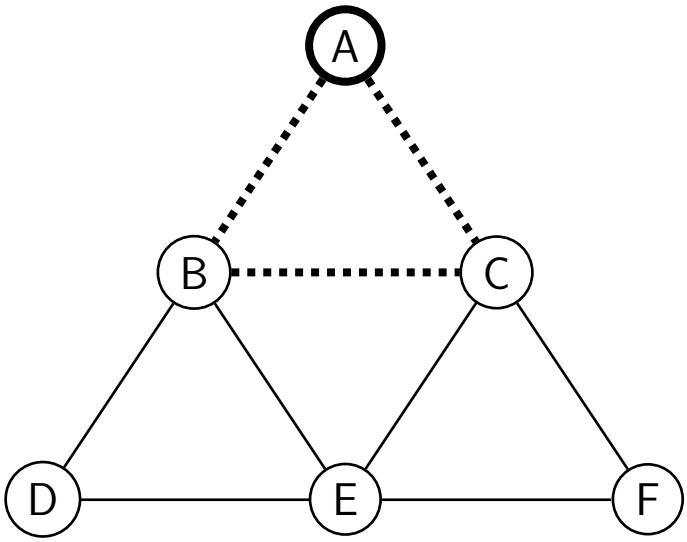


Figura 1.6: Representam-se pontilhadas as arestas percorridas na busca e em negrito o vértice atual da busca.

Após percorrido o caminho $\{A, B, C, A\}$ as arestas AB, BC, BA são apagadas e a busca não consegue progredir para novas arestas a partir do vértice A.

Seja T uma trilha inicialmente vazia. Como A não possui mais caminhos a se percorrer, adiciona-se o mesmo ao início da trilha T, e a busca continua recursivamente com o vértice C.

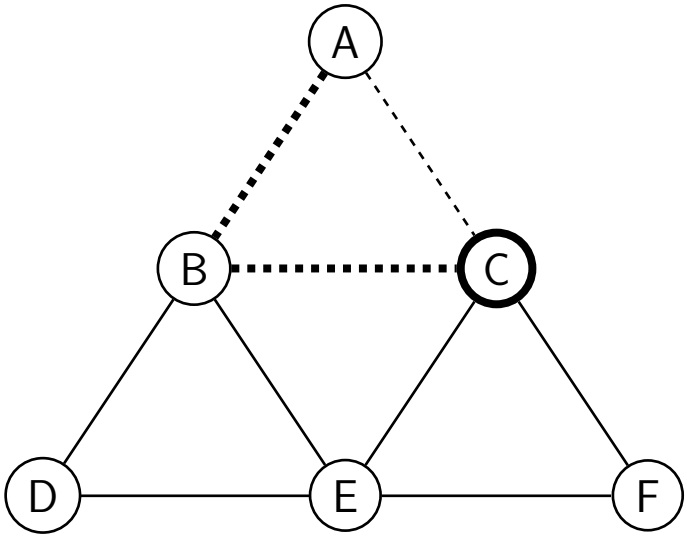
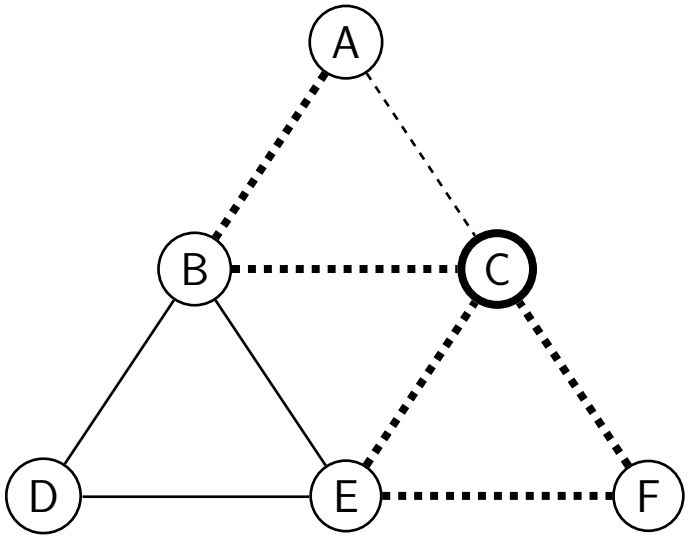
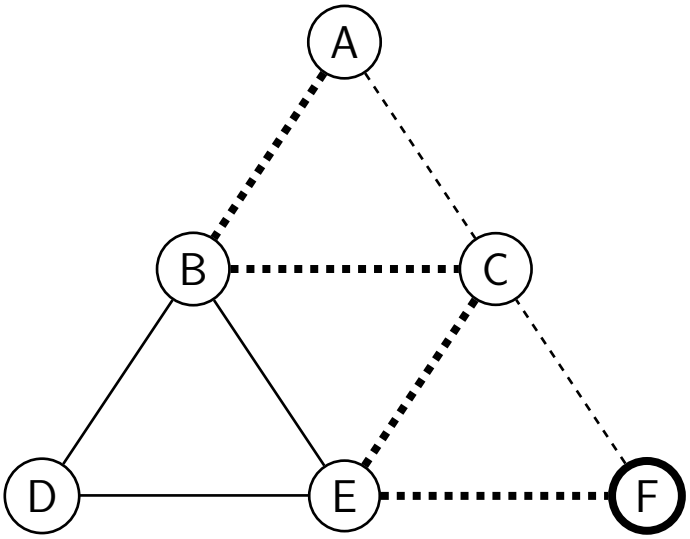


Figura 1.7: Representam-se em negrito as arestas que ainda fazem parte da pilha de recursão da busca.

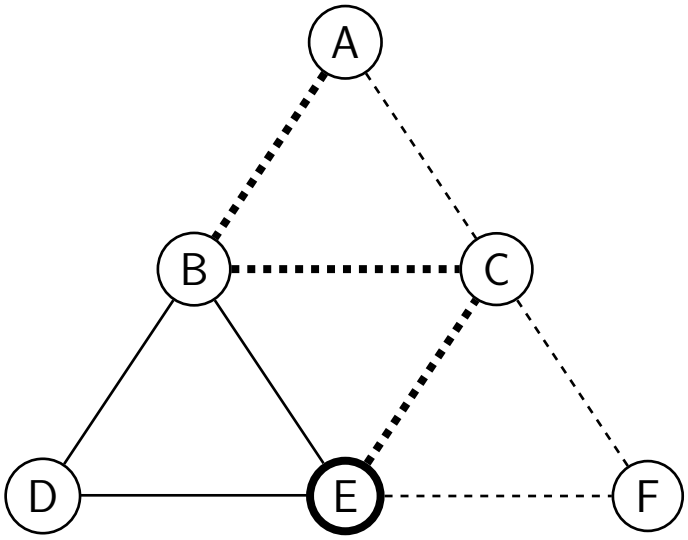
Seja $\{C, E, F, C\}$ o caminho percorrido na continuação da busca:



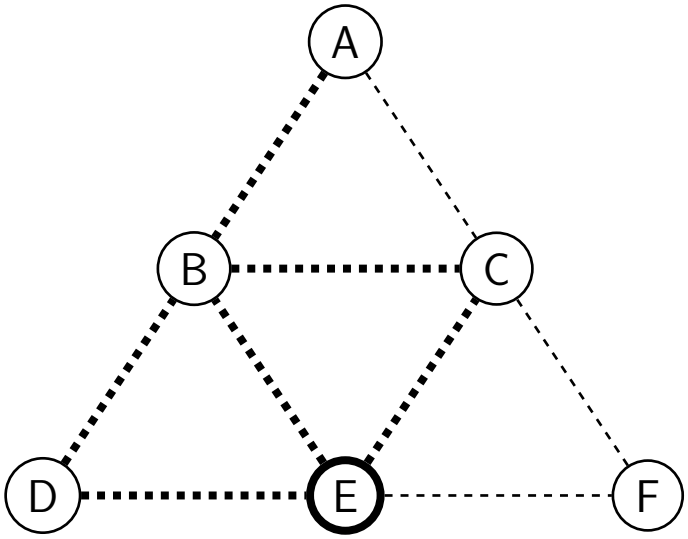
De modo similar ao primeiro passo, adicionamos o vértice C ao início de T e voltamos, recursivamente ao vértice F.



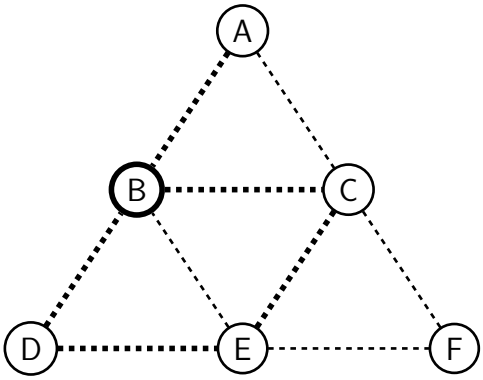
Como F também não possui arestas, ele é adicionado ao início de T, e a busca volta ao vértice E. Até o momento, $T = \{F, C, A\}$

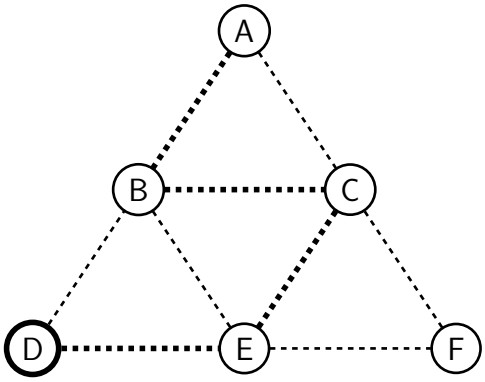


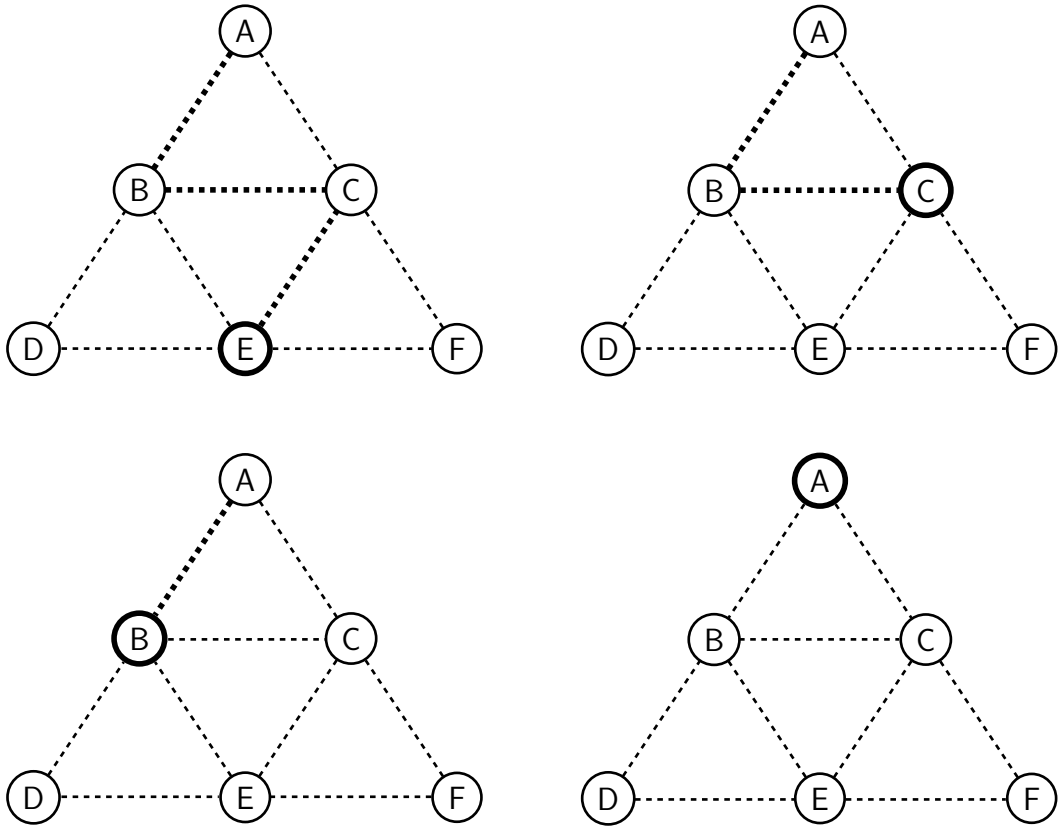
A partir de E a busca continua no caminho, por exemplo, $\{E, D, B, E\}$.



Como todas as arestas de G foram percorridas, a busca realiza, nos próximos passos, uma série de retornos, sempre adicionando à T o vértice atual da busca.







Nesta série de retornos são adicionados ao início de T os vértices E, B, D, E, C, B, A, um a um.

Finalizando-se a simulação do algoritmo temos a seguinte trilha T:

$$T = \{A, B, C, E, D, B, E, F, C, A\}$$

Como pode-se conferir, a trilha T construída é um circuito euleriano de G.

Segue agora a implementação do algoritmo de Hierholzer em C++ para digrafos (linhas 13 a 21 do algoritmo implementado em `euler-digrafo.cpp`):

```
void euler_hierholzer(int u){
    while(!_adj[u].empty()){
        Aresta e = _adj[u].back();
        int v = e.prox;
        _adj[u].pop_back();
        euler_hierholzer(v);
    }
    trilha.push(u);
}
```

Optou-se nesta implementação pelo uso de um `vector` para o armazenamento da lista de adjacências *adj*, pois assim é possível apagar facilmente os arcos já percorridas na busca, já que essa estrutura possui funções de acesso inserção e remoção de seu último elemento.

Para que tal função retorne corretamente uma trilha euleriana em grafos que não possuem circuito euleriano, é necessário que a primeira chamada de *trilha_euleriana*

tenha como parâmetro u o vértice cujo grau de saída é maior que seu grau de entrada, ou seja o vértice que será necessariamente o início da trilha euleriana.

Como em cada iteração deste algoritmo um arco é deletado, o número máximo de iterações que ele pode realizar é limitado pelo número de arcos de um digrafo, sendo assim o mesmo possui complexidade $\mathcal{O}(|E|)$ tanto em tempo quanto memória.

Um exemplo da utilização deste algoritmo em um problema de competições de programação pode ser encontrado no apêndice A.1.

Pode-se adaptar este algoritmo para grafos não direcionados mudando o modo como arestas são apagadas (vide código `euler-grafo.cpp`).

A função mostrada não funciona para grafos não direcionados pois trata cada sentido de uma aresta uv como arcos separados, $u \to v$ e $v \to u$. Sendo assim, quando uma aresta é deletada, digamos $u \to v$, a mesma aresta com orientação reversa $v \to u$, não é deletada automaticamente.

Para corrigir isto, pode-se criar um identificador único para cada aresta (igual para os dois sentidos de uma mesma aresta) que pode ser utilizado para marcar as arestas já percorridas na busca em profundidade.

Capítulo 2

O problema do Carteiro Chinês

Com o passar dos anos, a área de teoria dos grafos se desenvolveu muito, tratando dos mais variados tipos de problemas.

Em 1962, mais de 200 anos após Euler descrever sua solução para o problema de Königsberg, o matemático chinês Meigu Guan publicou um estudo [Gua62] que generalizava ainda mais o problema dos grafos eulerianos. Esse problema foi denominado Problema da Inspeção de Rotas, ou, como também é conhecido hoje: Problema do carteiro chinês (PCC), nomeado assim em homenagem a Guan.

O problema consiste em encontrar um passeio fechado que visite toda aresta de um grafo conexo pelo menos uma vez e que possua menor custo. A grande diferença aqui é que as arestas podem ser repetidas, ou seja, usadas mais de uma vez no passeio final.

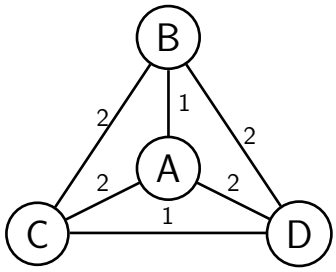


Figura 2.1: Exemplo

O nome do problema está relacionado ao problema do planejamento de rotas de carteiros: dada uma cidade com várias ruas de diferentes comprimentos e um posto de carteiros, encontrar a menor rota que um carteiro deve percorrer de modo a passar por todas ruas da cidade entregando cartas e voltar ao posto de carteiros no fim de sua rota.

Entre outras aplicações do problema estão o planejamento de rotas de coletadores de lixo e removedores de neve[1].

Por exemplo, para a figura 2.1, um passeio fechado de custo ótimo seria:

$$\{A, B, D, C, A, D, C, B, A\}$$

Sendo este um caminho fechado de custo 12 que percorre todas arestas do grafo. Neste exemplo foi necessário que as arestas AB e DC fossem percorridas duas vezes no passeio, porém nem sempre é necessária esta repetição.

No caso dos grafos eulerianos, a resposta para o problema do carteiro chinês é justamente o circuito euleriano. Nos outros casos, o procedimento será realizar a cópia de algumas arestas do grafo de modo a torná-lo euleriano.

[1] Na cidade de Morris, em Minnesota, Estados Unidos, foi realizado um estudo para otimizar as rotas de removedores de neve baseado no Problema do Carteiro Chinês [Ng96]

A solução do PCC para um grafo G terá duas partes: Tornar G euleriano com o menor custo possível e então encontrar o ciclo euleriano do grafo G modificado.

Uma generalização do Problema do Carteiro Chinês é o **problema das T-junções**. Este problema consiste em, dado um grafo (V, E) e um conjunto de vértices $T \subseteq V$, definir um conjunto de arestas J de custo mínimo tal que todo vértice do conjunto T possui grau ímpar no grafo (V, J) e todo vértice em $V \setminus T$ possui grau par no mesmo grafo.

Se tomamos, por exemplo, T como o conjunto de vértices de grau ímpar em um grafo não direcionado, um T-join J de menor custo representa exatamente o conjunto de arestas que, se adicionadas a um grafo qualquer torna-o euleriano. Sendo assim, o problema das T-junções resolve a primeira parte do PCC: tornar um grafo euleriano com o menor custo possível. Por esse motivo podemos dizer que o problema das T-junções é uma generalização do problema do carteiro chinês.

Discutiremos nas seções a seguir a solução para o problema para diferentes tipos de grafo quanto à orientação de suas arestas.

2.1 Grafos não direcionados

Analisaremos o caso em que o problema é modelado a partir de um grafo $G(V, E)$ simples, conexo e não direcionado.

Uma solução qualquer T do problema do carteiro chinês deverá percorrer cada aresta de G pelo menos uma vez, sendo assim uma trilha fechada.

Se G é euleriano, então as soluções ótimas para o PCC são os circuitos eulerianos do grafo. Do contrário, será necessário que T percorra uma mesma aresta duas vezes.

Seja $1 + x_e$ o número de vezes que uma aresta $e \in E$ é percorrida em T. Definimos G' como o grafo formado por G adicionado de x_e cópias de cada aresta $e \in E$. Isto é, cada aresta e de G aparecerá em G' $1 + x_e$ vezes.

Além disso, podemos modelar a trilha fechada T (solução do PCC em G) como uma trilha euleriana do grafo G', já que agora cada aresta repetida em T tem uma cópia correspondente no grafo G'.

Em outras palavras, o grafo G' será euleriano, e uma trilha euleriana de G' corresponderá a T, solução do PCC para o grafo G.

Sendo assim, podemos separar a solução do PCC em duas partes: Encontrar o valor ótimo de cópias x_e e, em seguida, encontrar um circuito euleriano no grafo G' construído.

Resolve-se a primeira parte do problema com um algoritmo de emparelhamento.

Lema 3. *Para todo grafo simples e conexo, existe uma solução ótima do PCC em que cada aresta é copiada no máximo 1 vez, ou seja* $x_e \in \{0, 1\}$.

Demonstração. Seja T uma solução ótima do PCC para um grafo G e x um vetor indexado pelas arestas indicando a quantidade de cópias necessárias de cada aresta para a formação de um supergrafo euleriano G'.

Por definição, x deverá possuir valores inteiros não negativos. Além disso, por x ser derivado de uma solução ótima, sabe-se que o valor de $\sum_e c_e x_e$ será o mínimo possível.

Seja x^* um vetor definido do seguinte modo:

$$x_e^* = x_e \bmod 2$$

Como o grafo G' induzido por x é euleriano, o grafo G^* induzido por x^* também deverá ser euleriano, pois a paridade dos graus de vértices em ambos os grafos é a mesma. Além disso, G' e G^* são conexos, pois possuem G como subgrafo.

Pela definição de x^*, $x_e^* \leq x_e, \forall e \in E$, garantindo assim que $\sum_e c_e x_e^* \leq \sum_e c_e x_e$.

No entanto, como x foi derivado da solução ótima T, sabemos que $\sum_e c_e x_e$ deverá ser mínimo, e portanto, $\sum_e c_e x_e^* = \sum_e c_e x_e$.

Sendo assim, os valores de x^* serão apenas 0 ou 1 e T^* consistirá em uma solução ótima, assim como T, provando o lema.

□

Lema 4. *Para todo grafo G simples e conexo, existe uma solução ótima do PCC cujo conjunto de arestas duplicadas consiste na união de caminhos aresta-disjuntos entre vértices de grau ímpar.*

Demonstração. Realizaremos uma prova por indução no número de arestas duplicadas (indicada pelo vetor x) em uma solução ótima de G, ou seja, em $\sum_{e \in E} x_e$.

O **caso base** dessa indução é quando $\sum_{e \in E} x_e = 0$. Neste caso, o grafo em questão é euleriano e o conjunto de arestas duplicadas será vazio, valendo então a propriedade do lema.

Na figura 2.2 mostra-se um exemplo deste caso. Como o grafo já possui um circuito euleriano, a solução ótima do problema não deve duplicar aresta alguma, ou seja, $x = \{0, 0, \ldots, 0\}$ induz uma solução ótima para o PCC.

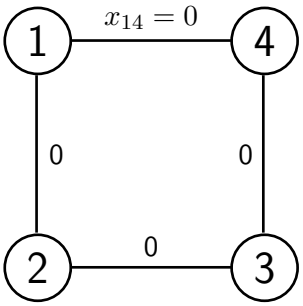


Figura 2.2: Para um grafo euleriano a solução $\{x_e = 0 : \forall e \in E\}$ é ótima.

A **hipótese de indução** é que a propriedade do lema vale para uma soma $\sum_e x_e < k$, com $k > 0$.

Trataremos agora de um caso de um grafo qualquer $G(V, E)$ em que sua solução ótima é induzida por um vetor de cópias x com $\sum_{e \in E} x_e = k$ e tal que cada aresta $e \in E$ é copiada no máximo 1 vez. A existência de tal solução ótima é garantida pelo lema 3.

Sejam $v \in V$ um vértice de grau ímpar e C um caminho de tamanho maximal que começa em v e percorre apenas arestas duplicadas.

Se C forma um circuito, pode-se simplesmente retirar as arestas duplicadas que formam C ($x_e = 0 \ \forall e \in C$) da solução, que a paridade dos vértices de G não mudaria,

e a nova solução induzida por x sem tais arestas seria coberta pela hipótese de indução, já que após essa mudança $\sum_{e \in E} x_e < k$.

Assume-se que C não forma um circuito daqui em diante.

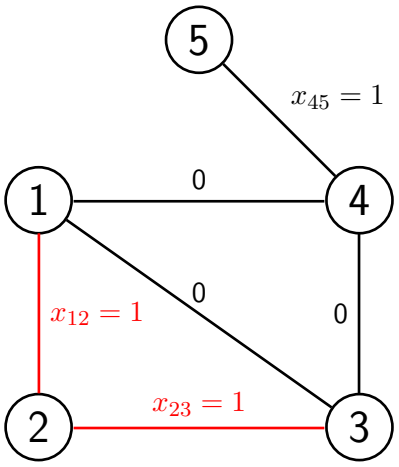


Figura 2.3: Exemplo de caminho C escolhido em vermelho, para caso com $k = 3$

Define-se $G' := (V, E \setminus C)$ como o grafo, possivelmente desconexo, formado por G decrescido das arestas de C.

Todos os vértices em G e G' terão a mesma paridade de grau, exceto pelos vértices extremos de C. Como estes eram vértices de grau ímpar em G, ambos terão grau par em G'.

Sendo assim, podemos definir um vetor x' que induz uma solução ótima do PCC para as componentes conexas de G' do seguinte modo:

$$x'_e = \begin{cases} 0 & e \in C \\ x_e & e \notin C \end{cases}$$

Isso porque as arestas do caminho C foram removidas, portanto não há mais necessidade de se duplicar as mesmas na solução ótima do PCC para G'.

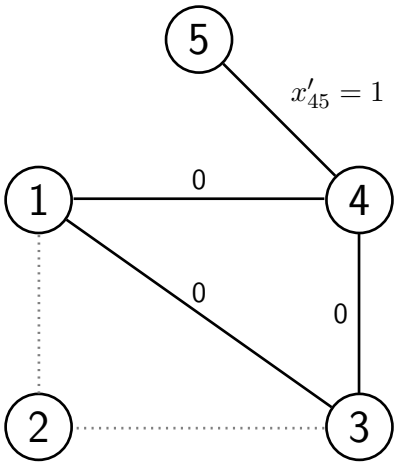


Figura 2.4: Grafo G' definido pelo grafo da figura 2.3

Segue da definição de x':

$$\sum_{e \in E \setminus C} x'_e = \sum_{e \in E} x_e - |\mathcal{C}|$$
$$\sum_{e \in E \setminus C} x'_e < \sum_{e \in E} x_e$$
$$\sum_{e \in E \setminus C} x'_e < k$$

Por consequência, vale a hipótese de indução para as componentes conexas do grafo G', ou seja, existe um conjunto $\mathcal{C}$ de caminhos aresta-disjuntos entre vértices de grau ímpar composto apenas por arestas duplicadas, indicadas por $x'_e = 1$.

Como o grafo G' não possui as arestas do caminho C, temos que todo caminho de $\mathcal{C}$ deverá ser aresta-disjunto com C.

Deste modo, o conjunto de caminhos aresta-disjuntos entre vértices de grau ímpar $\mathcal{C} \cup C$ induzirá a solução ótima para o PCC em G. Provando assim o passo da indução e finalizando a prova do lema.

□

Corolário 3. *Para todo grafo G simples e conexo, existe uma solução ótima do PCC cujo conjunto de arestas duplicadas é a união de caminhos de* ***custo mínimo*** *entre vértices de grau ímpar.*

Demonstração. Seja S o conjunto de arestas duplicadas em uma solução ótima do PCC para o grafo G, que consiste de uma união de caminhos aresta-disjuntos entre vértices de grau ímpar, segundo argumentado no lema 4.

Imagine que um caminho C pertencente a S, que liga os vértices quaisquer u e v, não é o caminho mínimo entre os vértices que liga. Isto é, existe um caminho C' de custo menor que C que liga u e v.

Neste caso, poderíamos retirar de S as arestas do caminho C e adicionar ao conjunto as arestas de C'. Deste modo, o conjunto S possuiria um custo menor que o original, nos permitindo derivar de S uma solução para o PCC de custo menor que a solução original.

Porém, fazia parte da hipótese que a solução original era ótima, nos levando assim a uma contradição.

Provamos assim o lema, garantindo a existência de uma solução para o PCC que consiste de uma união de caminhos mínimos entre vértices de grau ímpar.

□

Trataremos agora de explicar um algoritmo que resolve o PCC em grafos não direcionados.

A ideia principal de tal algoritmo é criar um novo grafo completo $G'(V', E')$. Em que V' é definido como o subconjunto de vértices de V que possuem um grau ímpar, e o custo de uma aresta de E' entre dois vértices u e v quaisquer será o custo de um caminho mínimo entre u e v no grafo original G.

Pelo corolário 3, uma solução ótima do PCC em G possui um conjunto de arestas duplicadas que pode ser representado como a união de caminhos de custo mínimo

entre vértices de grau ímpar. Como V' é o conjunto dos vértices de grau ímpar em G e cada aresta de E' representa um caminho de custo mínimo entre dois vértices de V', podemos reduzir o problema original no grafo G a um problema de emparelhamento perfeito de custo mínimo no grafo G'.

Dado um emparelhamento perfeito de custo mínimo M em G', consegue-se descobrir exatamente que arestas de G devem ser copiadas para gerar um supergrafo euleriano G^*. Por sua vez, um circuito euleriano de G^* representará uma solução ótima do PCC no grafo original G.

Para construir o grafo G^* a partir do emparelhamento M pode-se seguir o seguinte procedimento:

Considere inicialmente $G^*(V, E)$. Para cada aresta $e = uv \in M$, tome C um caminho de custo mínimo de u a v em G e duplique as arestas de C em G^*.

A duplicação das arestas de um caminho modifica apenas a paridade de grau dos vértices extremos. Como, no procedimento acima, estamos duplicando as arestas de todos caminhos derivados do emparelhamento perfeito M, isso implica que modificamos as paridades dos graus de todos os vértices de G^*, ou seja, modificamos apenas as paridades dos graus dos vértices de grau ímpar de G. Comprovando assim que o grafo G^* resultante é euleriano.

Após a construção G^* basta derivar um circuito euleriano do mesmo que chega-se a uma solução para o problema do PCC no grafo original G.

Segue no algoritmo 2 uma versão em pseudocódigo do algoritmo descrito.

Algoritmo 2 Solução do PCC em grafos não direcionados

```
1:  função PCC(G)
2:      (V, E, dist) ← G                    ▷ dist representa o custo de cada aresta de E
3:      V' ← {v ∈ V : δ[v] é impar}
4:      distMin ← FLOYD_WARSHALL(V, E, dist)
5:      E' ← V' × V'
6:      G' ← (V', E', distMin)
7:      M ← EMPARELHAMENTO_PERFEITO_MINIMO(G')
8:      E* ← E
9:      para uv ∈ M faça
10:         C ← EXPANDE(uv, E, dist, distMin)
11:         E* ← E* ∪ C
12:     G* ← (V, E*)
13:     devolve CIRCUITO_EULERIANO(G*)
14: função EXPANDE(uv, E, dist, distMin)    ▷ descomprime arestas de M
15:     se u = v então devolve ∅
16:     para w : uw ∈ E faça
17:         se distMin[u][v] = dist[u][w] + distMin[w][v] então
18:             devolve {uw} ∪ EXPANDE(wv, E, dist, distMin)
```

Escolheu-se o algoritmo de Floyd-Warshall para encontrar as distâncias mínimas entre todo par de vértices de G. Apesar de tal algoritmo possuir complexidade $\mathcal{O}(|V|^3)$, ele possui uma simples implementação.

Se G não possuir arestas de custo negativo, uma alternativa ao algoritmo de

Floyd-Warshall é utilizar o algoritmo de Dijkstra $|V'|$ vezes, calculando a distância mínima de cada vértice de grau ímpar a todos outros vértices do grafo, possuindo assim uma complexidade $\mathcal{O}(|V'||E| \log |V|)$.

Porém, independente do algoritmo escolhido, a complexidade geral do algoritmo ainda é dominada pela complexidade do algoritmo de emparelhamento.

Uma solução para se encontrar o emparelhamento perfeito mínimo de um grafo não direcionado com custos é usar o algoritmo Blossom de Edmonds [Edm61]. Encontra-se nas referências deste trabalho uma implementação do algoritmo de Edmonds intitulada "Blossom V" e publicada por Vladimir Kolmogorov em 2009 [Kol09].

A implementação do algoritmo completo desenvolvida neste trabalho tem complexidade $\mathcal{O}(|E||V|^2)$, entra-se em mais detalhes de implementação no capítulo 3.

Nos casos de grafos com um número pequeno de vértices ($\leq$ 24 vértices), é possível desenvolver uma solução usando programação dinâmica para encontrar tal emparelhamento, esse caso é ilustrado na explicação do problema tratado no apêndice A.3.

Para ilustrar um uso do PCC que foge um pouco da área de grafos, explica-se no apêndice A.2 o desafio "Sereja and the Arrangement of Numbers", do site Codeforces, que requer uma aplicação interessante do PCC para montar uma sequência de letras com uma característica especial.

2.2 Grafos direcionados

Analisaremos agora o problema do carteiro chinês aplicado a grafos direcionados, ou digrafos.

Lema 5. *Um digrafo G possui solução para o problema do carteiro chinês se, e somente se, é fortemente conexo.*

Demonstração. ($\Rightarrow$) Começamos provando que para que um digrafo G só possui solução para o problema do carteiro chinês se é fortemente conexo.

Vamos assumir, por absurdo, que G possui uma solução para o PCC mas não é fortemente conexo.

Como G possui uma solução para o PCC, então existe um passeio fechado T que passa por todos arcos de G.

Além disso, como T é um passeio fechado, para cada par de vértices u, v de G, sabemos que T passa por u e v. Assim, $T = \{v_0, v_1, \ldots, v_i = u, \ldots, v_j = v, \ldots\}$.

A partir de um passeio entre dois vértices sempre é possível derivar um caminho entre os mesmos seguindo o seguinte método:

Método para derivar um caminho de um passeio qualquer
Seja P um passeio que liga dois vértices quaisquer, mostraremos como construir um caminho ligando esses mesmos vértices a partir de P.
Como P é um passeio, possivelmente ele percorre um mesmo vértice mais que uma vez. Do contrário, já podemos considerar P um caminho, por definição.
Enquanto houver um vértice v percorrido mais que uma vez por P retira-se de P todos os vértices entre a primeira e a última aparição de v, mantendo apenas tal vértice no lugar.
Portanto um passeio P qualquer com repetição do vértice v, como o seguinte:

$$P = \{u_1, \dots u_i, v, \dots, v, u_j, \dots u_n\}$$

Passa a ser, após apagarmos todos elementos entre o primeiro e último v:

$$P = \{u_1, \dots u_i, v, u_j, \dots u_n\}$$

Repete-se tal passo até que P não percorra um mesmo vértice duas vezes, se tornando assim, por definição, um caminho.

Sendo assim, podemos afirmar que para todo par de vértices $u, v \in T$ existe um caminho entre u e v.

Já que não estamos considerando neste trabalho digrafos que possuem vértices isolados, cada vértice de G deve ser incidente ao menos a um arco.

Além disso, como todos arcos são percorridos em T (por ser solução do PCC), todos os vértices deverão estar presentes no passeio fechado T.

Por consequência para todo par de vértices de G existe um caminho entre eles, definindo assim G como um grafo fortemente conexo, contradizendo a hipótese inicial.

Provando assim que um digrafo que possua solução para o PCC é necessariamente fortemente conexo.

($\Leftarrow$) Provaremos agora que todo grafo fortemente conexo G possui uma solução para o PCC.

Seja v um vértice qualquer de G, usaremos este como um pivô para construir uma solução do PCC para G.

Para todo arco uw de G definimos C_{uw} como um passeio fechado que passa por uw e pelo vértice v. Uma possível construção de C_{uw} é concatenar o caminho de v a u, o arco uw e o caminho de w a v.

A existência de um caminho entre quaisquer dois vértices é garantida, já que G é fortemente conexo, garantindo assim a existência de C_{uw}.

Para construir a solução P, finalmente, basta realizar a concatenação dos arcos de C_e para todo arco e presente em G.

Deste modo garantimos que toda aresta de G é percorrida pelo menos uma vez, provando assim a volta do lema: todo digrafo fortemente conexo possui uma solução para o problema do carteiro chinês.

□

O algoritmo que soluciona o PCC para digrafos segue o mesmo modelo geral da solução para grafos: multiplicar arcos do grafo original até que o mesmo se torne

euleriano.

O circuito euleriano desse digrafo modificado será a solução do problema do carteiro chinês para o digrafo original.

Uma grande diferença na solução do PCC é que, para digrafos, não vale o lema 3, que garante a existência de uma solução para todo PCC em que cada aresta do grafo é percorrida no máximo duas vezes. Um contra exemplo disso é o caso ilustrado na figura 2.5.

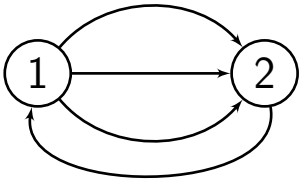


Figura 2.5: Contra exemplo do lema 3 para os digrafos

Para que um passeio fechado percorra os três arcos de 1 a 2 da figura 2.5, é necessário que esse mesmo passeio percorra três vezes o arco de 2 a 1.

O lema 3 é utilizado para justificar a utilização de um algoritmo de emparelhamento perfeito para definir quais arestas duplicar na solução do PCC. Como tal lema não vale para digrafos, a escolha dessas arestas deverá ser feita de um modo diferente, utilizando uma modelagem do problema de transporte[2].

A solução do PCC para um digrafo G fortemente conexo pode ser dividida em 4 passos:

1º Define-se F como o conjunto dos vértices que possuam grau de entrada (δ^+) maior que o grau de saída (δ^-) e S como o conjunto dos vértices que possuam grau de saída maior que o grau de entrada.

Calcula-se o custo do menor caminho entre todos os pares de vértices u, v onde $u \in F$ e $v \in S$.

2º Modela-se um problema de transporte: Seja b uma função demanda definida para todo vértice como $b(v) = \delta^-(v) - \delta^+(v)$.

Os vértices de F serão vértices de oferta (com $b < 0$), cada vértice $u \in F$ terá que escoar $-b(u)$ unidades de fluxo.

Já os vértices de S serão vértices de demanda (com $b > 0$), cada vértice $v \in S$ terá que receber $b(v)$ unidades de fluxo.

Todo par de vértices u, v com $u \in F$ e $v \in S$ será ligado por um arco que pode transportar qualquer quantidade de fluxo, mas que tem custo igual a um menor caminho de u a v.

Deve-se resolver tal problema de transporte no grafo F, S-bipartido minimizando o custo total e respeitando a função demanda definida.

[2]Problema de fluxo máximo que determina uma maneira de transportar produtos de certos vértices de origem até outros vértices de destino, respeitando ofertas e demandas de cada vértice, capacidade de arestas e minimizando o custo de transporte.

A solução do problema de transporte descrito será dado por uma função $f : F \times S \to \mathbb{N}$ que determina a quantidade de fluxo transportada entre cada par de vértices de F e S.

3º Usa-se a solução do problema de transporte para derivar um digrafo euleriano G^*:

Inicialmente $G^* = G$.

Se um arco uv é escolhido para transportar uma quantidade $f(uv)$ positiva de fluxo, deve-se criar $f(uv)$ cópias de todos arcos de um menor caminho de u a v em G^*, chamaremos esse processo de expansão de uv.

Após a duplicação dos arcos de um caminho entre dois vértices u e v a diferença absoluta dos graus de entrada e saída de u e v diminuirá em uma unidade, essa mesma diferença de graus se manterá constante para vértices internos ao caminho.

Ao final da expansão de todos arcos que possuem valor de f positivo, todos vértices de G^* possuirão um mesmo valor de grau de entrada e saída, tornando G^* euleriano (teorema 1.3.1).

4º Encontrar o circuito euleriano de G^*, que será também a solução do problema do carteiro chinês para o digrafo original G.

O algoritmo 3 é a implementação em pseudocódigo da solução descrita acima.

Na função $PROBLEMA\_DE\_TRANSPORTE$ define-se uma redução do problema para o problema do fluxo máximo de custo mínimo. Para se resolver esta redução, pode-se utilizar um algoritmo de Simplex para Redes como discutido por Orlin [Orl95], de complexidade $\mathcal{O}(VE \log V \log(VC))$, sendo C a capacidade máxima dos arcos da rede.

Outra solução para a redução é usar uma variação do algoritmo de Edmonds-Karp [Edm72] para fluxo de custo mínimo de complexidade $\mathcal{O}(V^2E^2)$. A explicação e implementação dessa variação estão disponíveis online, no site E-maxx [E-m14].

Neste trabalho seguiu-se o modelo da implementação de Edmonds-Karp, como se descreve em mais detalhes no capítulo 3 de implementações.

Apresentado o pseudocódigo deste algoritmo, trataremos em seguida de aplicar o algoritmo explicado em um exemplo, seguindo os quatro passos da explicação.

Algoritmo 3 Solução do PCC em digrafos

```
1: função PCC(G)
2:     (V, E, dist) ← G
3:     dist' ← FLOYD_WARSHALL(V, E, dist)
4:     para v ∈ V faça
5:         b(v) ← δ⁻(v) − δ⁺(v)                    ▷ Calcula a função demanda
6:     F ← {v ∈ V : b(v) < 0}
7:     S ← {v ∈ V : b(v) > 0}
8:     f ← PROBLEMA_DE_TRANSPORTE(F, S, dist, b)
9:     E* ← E                                       ▷ Cria uma cópia de E
10:    para uv ∈ E faça
11:        C ← EXPANDE(uv, E, dist, dist')
12:        para i de 1 a f(uv) faça
13:            E* ← E* ∪ C                          ▷ Duplica f(uv) vezes os arcos de C
14:    G* ← (V, E*)
15:    devolve CIRCUITO_EULERIANO(G*)
16: função EXPANDE(uv, E, dist, dist')
17:    se u = v então devolve ∅
18:    para w : uw ∈ E faça
19:        se dist'[u][v] = dist[u][w] + dist'[w][v] então
20:            devolve {uw} ∪ EXPANDE(wv, E, dist, dist')
21: função PROBLEMA_DE_TRANSPORTE(F, S, custo, b)
22:    Define dois vértices artificiais, uma fonte s e um sorvedouro t
23:    E ← F × S
24:    para uv ∈ E faça
25:        cap(uv) = ∞
26:    para u ∈ F faça
27:        cap(su) = −b(u)
28:        custo(su) = 0
29:        E = E ∪ {su}
30:    para v ∈ S faça
31:        cap(vt) = b(v)
32:        custo(vt) = 0
33:        E = E ∪ {vt}
34:    devolve FLUXO_CUSTO_MINIMO(s, t, E, custo, cap)
```

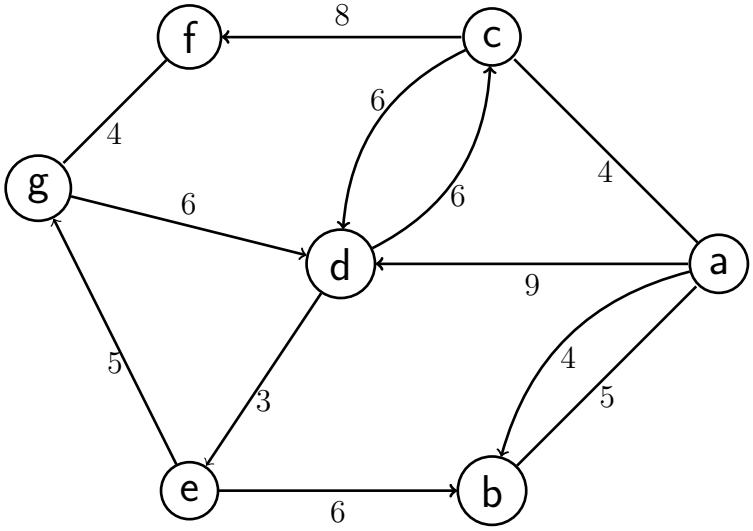


Figura 2.6: Digrafo retirado de exercício do site do MIT [LO81]

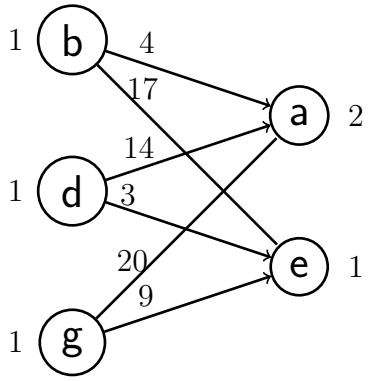
1º Definimos então os conjuntos F e S:

$$F = \{b, d, g\}$$
$$S = \{a, e\}$$

Computamos também a distância mínima entre os pares de vértices dos dois conjuntos:

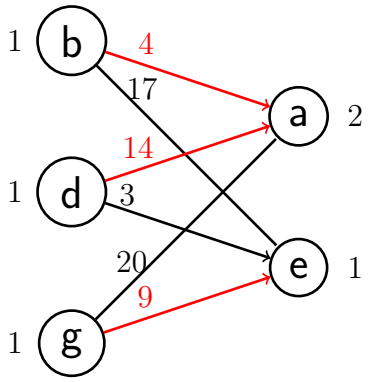
	a	e
b	4	17
d	14	3
g	20	9

2º Modelamos então o problema de transporte, os vértices b, d, g escoando uma unidade de fluxo cada e os vértices a, b com demandas 2 e 1, respectivamente.



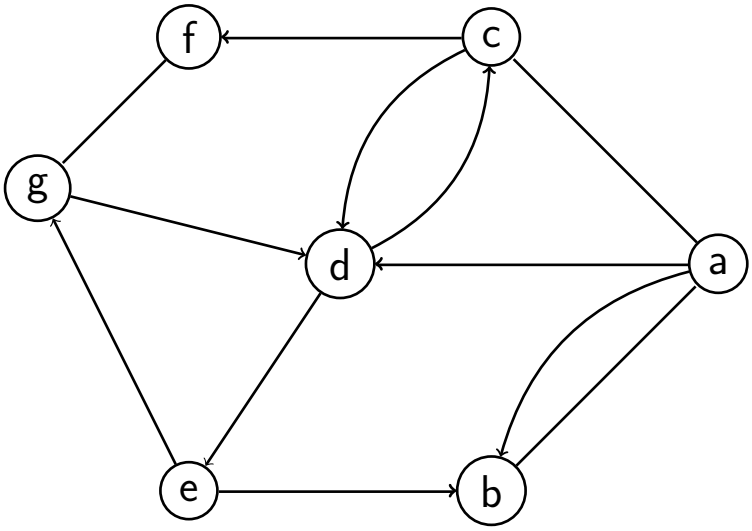
Os arcos definidos entre os vértices de F e S possuem capacidade infinita e custo igual à distância mínima dos mesmos vértices em G, como calculado e apresentado na tabela do passo anterior.

Uma solução ótima para o problema apresentado consiste em transportar uma unidade de fluxo pelos arcos de custo $4, 14$ e 9, escoando os vértices b, d e g e suprindo a demanda dos vértices a e e.

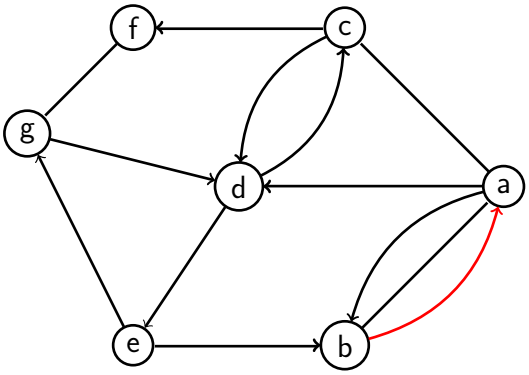


3º Agora devemos realizar a duplicação dos caminhos representados pelos arcos ba, da e ge.

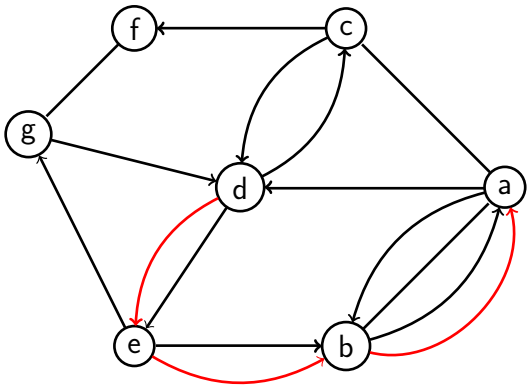
Representamos o estado inicial do digrafo G, sem os pesos nos arcos:



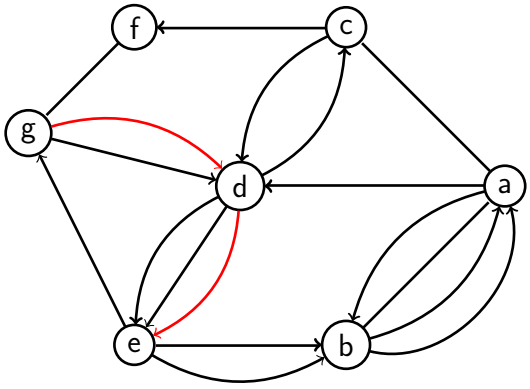
Faremos tal duplicação passo a passo, começando a duplicar os arcos do caminho mínimo de b a a. Representamos os arcos adicionados em vermelho.



Agora duplicaremos os arcos do caminho mínimo de d a a:



Finalmente, duplicamos os arcos do caminho mínimo de g a e:



As sucessivas duplicações de caminho realizadas em G fazem com que todos os vértices possuam grau de saída e entrada iguais, tornando o novo grafo euleriano (teorema 1.3.1).

No quarto passo deriva-se um circuito euleriano para o digrafo construído.

4º Um possível circuito euleriano $\mathcal{C}$ que se inicia no vértice a é o seguinte:

$$\mathcal{C} = \{a, c, f, g, d, c, d, e, g, d, e, b, a, d, e, b, a, b, a\}$$

Representamos tal circuito no grafo abaixo, indicando em cada arco a ordem em que o mesmo é percorrido:

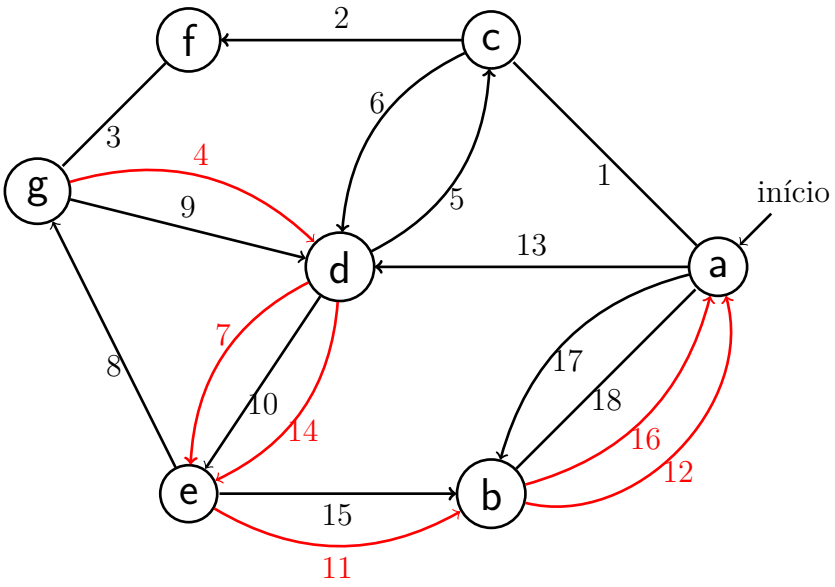


Figura 2.7: Representam-se em preto os arcos presentes no grafo G original, e em vermelho os arcos resultantes das duplicações realizadas no terceiro passo da solução.

O circuito $\mathcal{C}$ percorrido no digrafo original G representa o passeio fechado que soluciona o Problema do Carteiro Chinês, já que percorre todo arco de G ao menos uma vez.

Analisando o peso dos arcos de G concluímos que a solução $\mathcal{C}$ tem custo $66+27 = 93$, 66 é o custo de se percorrer todas os arcos de G, representados em preto, uma única vez, e 27 é o custo de se percorrer os arcos em vermelho.

2.3 Grafos mistos

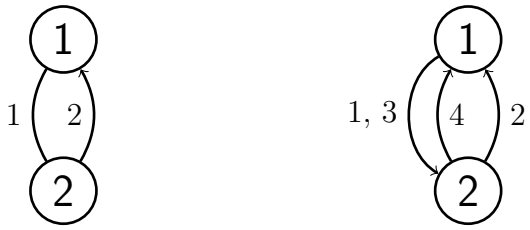
Já analisamos o Problema do Carteiro Chinês aplicado a grafos e digrafos, agora discutiremos o mesmo problema aplicado a grafos mistos, ou seja, grafos que tenham tanto arestas quanto arcos.

Ao contrário do que uma análise breve pode sugerir, não é possível reduzir o PCC para grafos mistos ao caso de grafos direcionados simplesmente transformando as arestas não direcionadas em dois arcos de sentidos opostos.

Um contra exemplo para este pensamento é o seguinte:

Na esquerda temos um grafo misto, e na direita temos o mesmo grafo com sua aresta substituída por dois arcos em sentidos opostos.

Apesar da similaridade dos grafos, suas soluções para o problema do carteiro chinês diferem. Uma solução ótima do problema para o primeiro grafo é $\{1, 2, 1\}$, enquanto que uma solução ótima possível para o segundo grafo é $\{1, 2, 1, 2, 1\}$, como representado visualmente a seguir.



Apesar de conseguirmos encontrar uma solução para o caso direcionado derivado, não há um modo trivial de transformar uma solução para tal caso em uma solução do caso não direcionado.

Diferentemente do problema aplicado a grafos ou digrafos, o PCC aplicado a grafos mistos é um problema NP-difícil.

Trataremos, portanto, de uma 2-aproximação para solucionar este problema, publicado em 1973 por Edmonds e Johnson [EJ73] e posteriormente explicado em mais detalhes por Frederickson [Fre79].

O algoritmo é composto por três passos principais:

1. Adicionar cópias de arestas ao grafo de modo a tornar o grau de todo vértice par, considerando arcos como arestas sem orientação

2. Igualar o grau de entrada e saída de todo vértice, adicionando cópias de arcos e orientando arestas existentes, mantendo par o grau de cada vértice

3. Encontrar um circuito euleriano no digrafo construído

Apresenta-se no algoritmo 4 um pseudocódigo adaptado de Frederickson [Fre79] que segue esses passos. Seja E o conjunto de arestas não direcionadas, A o conjunto de arcos e c a função de custo destas arestas e arcos.

Precisamos também fazer algumas redefinições das funções δ que definem o grau de um vértice em um grafo misto:

Tome $\delta(v)$ como a quantidade de arestas que incidem no vértice v, $\delta^+(v)$ como o grau de entrada de v, isto é, a quantidade de arcos que apontam a v e $\delta^-(v)$ como o grau de saída de v, o número de arcos que saem de v.

Define-se como **grau total** de um vértice v, $\delta_t(v)$, a soma do grau de arestas não direcionadas, grau de entrada e saída, ou seja $\delta_t(v) := \delta(v) + \delta^+(v) + \delta^-(v)$.

Para exemplificar o funcionamento do algoritmo, usaremos o seguinte grafo:

Algoritmo 4 Função principal do algoritmo sugerido por Frederickson

```
1: função MISTO(G, c)
2:     (V, A, E) ← G
3:     G' ←GRAU_TOTAL_PAR(G, c)                          ▷ Torna δ_t par
4:     M, U ←IGUALA_GRAU_DIR(G', c)     ▷ Iguala δ+ e δ− no grafo (V, M, U)
5:     M', U' ←GRAU_PAR(G', M, U)          ▷ δ par e δ+ = δ− em (V, M', U')
6:     G_e ← (V, U', M')
7:     devolve CIRCUITO_EULERIANO(G_e)
```

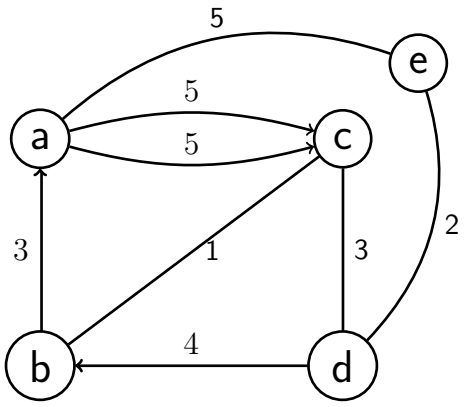


Figura 2.8: Exemplo de grafo misto G

A função auxiliar "GRAU_TOTAL_PAR" apresentada no algoritmo 5 duplica arcos e arestas de G, criando um outro grafo G' em que todos vértices possuem grau total par.

Os arcos e arestas duplicados são escolhidos de modo a minimizar o custo total dos arcos e arestas de G'.

Algoritmo 5 Função auxiliar GRAU TOTAL PAR

```
1: função GRAU_TOTAL_PAR(G,c)
2:     (V, E, A) ← G
3:     V' ← {v ∈ V : δ_t(v) é impar}
4:     custMin ← FLOYD_WARSHALL(G, c)   ▷ Ignora o sentido dos arcos de A
5:     K ← (V', V' × V', custMin)                    ▷ Grafo completo de V'
6:     M ← EMPARELHAMENTO_PERFEITO_MINIMO(K)
7:     Expande cada aresta de M nos arcos e arestas correspondentes do grafo original
8:     Define o multiconjunto A' dos arcos expandidos
9:     Define o multiconjunto E' das arestas expandidas
10:    devolve (V, E + E', A + A')          ▷ Devolve grafo com δ_t(v) par ∀v ∈ V
```

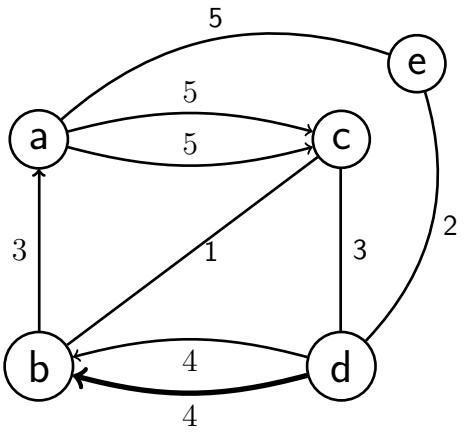


Figura 2.9: Em destaque o arco duplicado em GRAU_TOTAL_PAR (algoritmo 5)

Por sua vez, a função "IGUALA_GRAU_DIR" devolve dois multiconjuntos M e U, de arcos e arestas, respectivamente, de menor custo tal que todo vértice do grafo (V, U, M) possui grau de entrada e saída iguais.

O multiconjunto M será formado por ao menos uma cópia de todos arcos de A e cópias de arestas de E direcionadas. Já U será composto pelas arestas de E que não foram direcionadas e adicionadas a M.

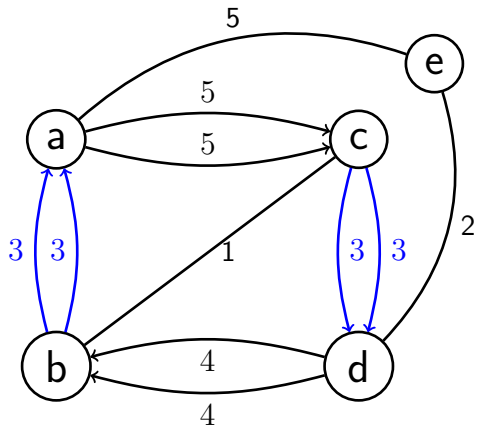


Figura 2.10: O arco ba e a aresta orientada cd duplicados pela função IGUALA_GRAU_DIR (algoritmo 6)

Apesar do algoritmo 6 devolver multiconjuntos M, U que igualam o grau de entrada e saída dos vértices de V, ele não garante que o grau total desses vértices continua par, como pode-se notar no exemplo da figura 2.10.

Por esse motivo é necessária a função GRAU_PAR, do algoritmo 7, que deriva de M, U outros multiconjuntos M' e U' que tornam δ_t par, mantêm a igualdade de δ^+ e δ^- e por consequência, tornam δ par para todo vértice do grafo (V, M', U').

Algoritmo 6 Função auxiliar IGUALA GRAU DIR

função IGUALA_GRAU_DIR(G, c)
    ▷ Devolve dois multiconjuntos M e U, tal que M contém ao menos uma cópia de cada arco de A e algumas arestas de E orientadas, e $U \subseteq E$ contém apenas arestas não orientadas e não pertencentes a M.
    $(V, E, A) \leftarrow G$
    $custMin \leftarrow$ FLOYD_WARSHALL(V, E, A, c)
    para $v \in V$ **faça**
        $b(v) \leftarrow \delta^-(v) - \delta^+(v)$ ▷ Calcula a função demanda
    $F \leftarrow \{v \in V : b(v) < 0\}$
    $S \leftarrow \{v \in V : b(v) > 0\}$
    $f \leftarrow$ PROBLEMA_DE_TRANSPORTE$(F, S, custMin, b)$
    Seja $f(a)$ a quantidade de fluxo que passa por cada arco $a \in A$
    Sejam $f(uv)$ e $f(vu)$ as quantidades de fluxo transportadas pelas duas orientações de uma aresta $uv \in E$
    $M, U \leftarrow \emptyset$
    para $a \in A$ **faça**
        Insere $f(a) + 1$ cópias do arco a em M
    para $uv \in E$ **faça**
        se $f(vu) \geq 1$ ou $f(uv) \geq 1$ **então** ▷ Se $custMin > 0$ é impossível que $f(uv) \geq 1$ e $f(vu) \geq 1$ ao mesmo tempo
            Insere $f(uv)$ cópias de uv e $f(vu)$ cópias de vu em M
        senão
            Insere uv em U
    devolve M, U

Algoritmo 7 Função auxiliar GRAU PAR

```
1: função GRAU_PAR(M, U)
2:     U' ← U
3:     O multiconjunto M' ⊆ M deverá possuir apenas as arestas direcionadas e as
       cópias de arcos criadas em GRAU_DIR.
4:     Seja V' = {v ∈ V : δ+ + δ− + δ é ímpar }
5:     enquanto V' não é vazio faça
6:         Seleciona um vértice qualquer v_ini de V'
7:         v ← v_ini
8:         enquanto v ∈ V' faça
9:             V' ← V' \ v
10:            repita
11:                Seja a ∈ M', um arco da forma vw ou wv
12:                se a é o arco vw então
13:                    Insere outra cópia de a em M'
14:                senão
15:                    Apague uma cópia de a de M'
16:                v ← w
17:            enquanto v ∉ V'
18:            V' ← V' \ v
19:            repita
20:                Remove uma aresta vw de U'
21:                Orienta tal aresta no sentido de v para w e a adiciona em M'
22:                v ← w
23:            enquanto v ∉ V' e v ≠ v_ini
       devolve M', U'
```

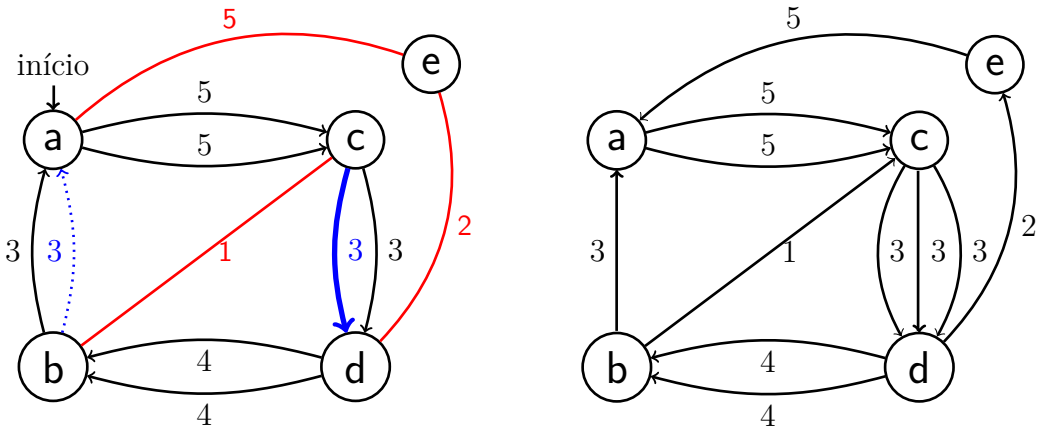


Figura 2.11: Criação de M', U', a partir de M, U pelo algoritmo 7

No exemplo 2.11 o algoritmo 7 toma o circuito $\{a, b, c, d, e, a\}$ para realizar a construção de M', U'. Em azul destacam-se os arcos de M e em vermelho as arestas de U escolhidos.

Como o arco $ab \in M$ é percorrido pelo circuito no sentido contrário, o mesmo é deletado do conjunto M' (linha 15), já o arco $cd \in M$, percorrido no sentido correto, é duplicado (linha 13) em M'.

As arestas do circuito são todas orientadas no sentido em que são percorridas: $bc, de, ea \in E$ (linha 21).

Deste modo conseguem-se multiconjuntos M', U' de mesmo custo que M, U e que mantêm $\delta^+ = \delta^-$ e δ par para todo vértice de G.

Teorema 2.3.1. *Um grafo misto fortemente conexo que possui $\delta(v)$ par e $\delta^+(v) = \delta^-(v)$ para todo vértice v, possui um circuito euleriano.*

Demonstração. Seja $G(V, E, A)$ um grafo que respeita as hipóteses do teorema, sendo E seu multiconjunto de arestas e A seu multiconjunto de arcos.

Definem-se G_e como o grafo induzido pelas arestas de E, e G_a como o grafo induzido pelos arcos de A. Apesar de não serem necessariamente conexos, valerá que todo vértice de G_e possui um δ par e que todo vértice de G_a tem $\delta^+ = \delta^-$, já que o grafo G, base de ambos subgrafos, respeita a hipótese do teorema.

Por consequência, G_e será composto por diferentes componentes conexas eulerianas (já que δ é par para todos seus vértices), pelo teorema 1.2.1. Denominam-se $C_1, C_2, \ldots, C_k$ os circuitos eulerianos das k componentes de G_e.

A mesma consequência segue para o grafo G_a: cada uma de suas componentes fortemente conexas possuirá grau de entrada e saída iguais, sendo assim eulerianas pelo teorema 1.3.1. Denominam-se $C'_1, \ldots, C'_l$ os circuitos eulerianos dessas l componentes.

Pode-se definir um procedimento de união de dois circuitos em um único circuito se ambos possuem ao menos um vértice em comum:

Digamos que dois circuitos $\mathcal{C}_i$ e $\mathcal{C}_j$ possuem um vértice v em comum. Podemos descrever ambos circuitos por seus arcos ou arestas do seguinte modo:

$$\mathcal{C}_i = \{vw_1, w_1w_2, \ldots, w_nv\}$$
$$\mathcal{C}_j = \{vu_1, u_1u_2, \ldots, u_mv\}$$

Se representados desse modo, a união dos circuitos será equivalente à justaposição de ambos:

$$\mathcal{C}_i \cup \mathcal{C}_j = \{vw_1, w_1w_2, \dots w_nv, vu_1, u_1u_2, \dots, u_mv\}$$

Usaremos este método de união de circuitos para unir os circuitos $C_1, \dots C_k$ e $C'_1, \dots, C'_l$ formando assim um circuito euleriano do grafo misto original G.

Como o grafo original é fortemente conexo, sempre é possível realizar a união de todos circuitos.

Do contrário, se não houver meio de realizar tal união deverá necessariamente existir um circuito C_i que não possui vértice em comum com nenhum outro circuito, implicando assim que $\bigcup C_i, C'_i = G$ não seria fortemente conexo, contradizendo a hipótese do teorema.

Prova-se assim a existência de um circuito euleriano dado que um grafo misto é fortemente conexo e possui δ par e $\delta^+ = \delta^-$ para todos seus vértices. □

Como o exemplo apresentado é transformado em um digrafo euleriano após a execução de 7, basta tomar o circuito euleriano do mesmo para derivar uma solução do grafo misto original, como mostrado na figura 2.12.

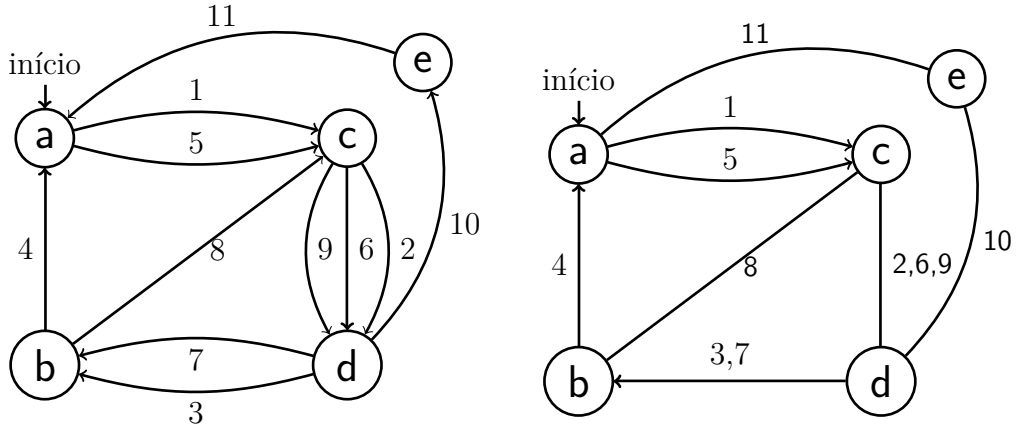


Figura 2.12: Circuito euleriano de G_e e solução do PCC de G, respectivamente

No caso geral em que o grafo final é um grafo misto, e não um digrafo ou grafo normal, pode-se usar uma versão especial do algoritmo de Hierholzer[3] explicado na seção 1.4.

Lema 6. *Seja $C(M, U)$ o custo dos multiconjuntos M, U, ou seja, a soma do custo dos arcos e arestas que compõem M e U.*

Vale que $C(M', U') = C(M, U)$. Isto é, os multiconjuntos M', U' retornados pela função GRAU PAR (algoritmo 7) possuem o mesmo custo que os multiconjuntos M, U, retornados pela função IGUALA GRAU DIR (algoritmo 6).

Demonstração. Como os conjuntos M, U foram gerados resolvendo um problema de transporte, possuímos a garantia que eles são os multiconjuntos de menor custo

[3] O repositório do projeto possui todas implementações de algoritmos tratados nesta monografia, incluindo essa implementação mencionada de caminhos eulerianos em grafos mistos.

que igualam o grau de entrada e saída de todos vértices do grafo (V, M, U). Por consequência, vale que $C(M', U') \geq C(M, U)$.

Para mostrar que $C(M', U') = C(M, U)$ chegaremos a uma contradição assumindo que $C(M', U') > C(M, U)$:

Os multiconjuntos M' e U' são construídos pelo algoritmo 7 a partir de M e U. Como exemplificado na figura 2.11 na construção de M', U' podem ocorrer três tipos de eventos:

1. Um arco $a \in M$ é duplicado em M'
2. Um arco $a \in M$ é deletado em M'
3. Uma aresta $e \in U$ é orientada e adicionada a M'

Dos três tipos de eventos listados, apenas os dois primeiros mudam o custo dos multiconjuntos M', U' em relação a M, U. Um evento do tipo 1 aumenta o custo de M', U' pelo valor do custo do arco a, assim como um evento do tipo 2 diminui o custo de M', U' pelo custo de a, em relação ao custo de M, U.

Sendo assim, podemos definir $C(M', U')$ como $C(M', U') = C(M, U) + C_+ - C_-$, sendo C_+ a soma do custo dos arcos duplicados (evento 1) e C_- a soma do custo dos arcos deletados no algoritmo (evento 2).

Se $C(M', U') > C(M, U)$, como assumimos, vale que:

$$\begin{aligned} C(M, U) + C_+ - C_- &= C(M', U') \\ C_+ - C_- &= C(M', U') - C(M, U) \\ C_+ - C_- &> 0 \end{aligned}$$

Vamos tomar agora outros multiconjuntos M'', U'', em que $U'' = U'$ e que M'' será definido a partir de M com três eventos, de modo análogo a M':

1. Se um arco $a \in M$ foi duplicado em M', o mesmo será deletado de M''.
2. Se um arco $a \in M$ foi deletado em M', o mesmo será duplicado em M''.
3. Se uma aresta $e \in U$ foi orientada da forma uv e adicionada a M', a mesma será adicionada a M'' com a orientação oposta: vu.

Para exemplificar essa definição, na figura 2.13 comparam-se os conjuntos M', U' e M'', U'' do exemplo tratado nesta seção.

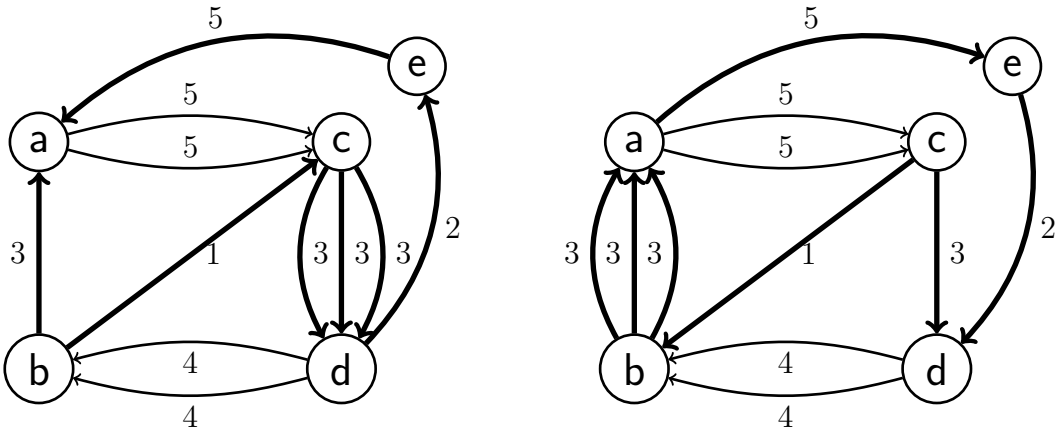


Figura 2.13: Diferenças entre M', U' e M'', U'', respectivamente

Como M'', U'' também igualam os graus de entrada e saída de G, então vale que $C(M'', U'') \geq C(M, U)$, já que M, U foram gerados usando um algoritmo exato e possuem custo mínimo, como argumentado anteriormente. Além disso, pela definição de M'', U'', podemos escrever o custo desses multiconjuntos como $C(M'', U'') = C(M, U) + C_- - C_+$.

Sendo assim:

$$\begin{aligned} C(M, U) + C_- - C_+ &= C(M'', U'') \\ C_+ - C_- &= C(M, U) - C(M'', U'') \\ C_+ - C_- &\leq 0 \end{aligned}$$

Como havíamos provado anteriormente que $C_+ - C_- > 0$, chega-se a uma contradição. Provando, por absurdo, que $C(M', U') = C(M, U)$. □

Adapta-se a seguir a prova feita por Frederickson [Fre79] da razão de aproximação do algoritmo.

Teorema 2.3.2. *O algoritmo apresentado é uma 2-aproximação.*

Em outras palavras, seja C o custo de uma solução ótima do PCC em um grafo misto G, e C' o custo da solução construída pelo algoritmo descrito, vale que:

$$C'/C \leq 2$$

Demonstração. Assume-se que todas arestas e arcos de G tenham um custo positivo.

Define-se G^* como o grafo G com todos arcos e arestas duplicados e C^* o custo da solução dada pelo algoritmo MISTO (algoritmo 4) ao grafo G^*.

Como o grau total de todos vértices de G^* é par, as únicas funções que modificarão o grafo G^* do algoritmo serão 6 e 7. Além disso, como a função 6 encontra multiconjuntos de arcos e arestas que igualam os graus de entrada e saída de G^* com o menor custo possível (solucionando o problema de transporte modelado) e como a função 7 modifica os multiconjuntos sem modificar o seu custo (lema 6), podemos dizer que C^* é o custo ótimo para a solução do PCC para o grafo misto G^*.

Sendo assim, como a solução ótima de G, de custo C, pode ser duplicada para encontrar uma solução de G^* também deve valer que $C^* \leq 2C$.

Definimos agora G' como o grafo retornado pela função 5 com parâmetro G. A função 5 constrói G' duplicando arcos e arestas de caminhos entre vértices de grau total ímpar de G, minimizando o custo de tal construção.

Pode-se afirmar que não existirá em G' um arco ou aresta uv que foi duplicado mais que uma vez na função 5, pois se existisse poderia-se remover de G' duas cópias de uv que teríamos outro grafo com mesma paridade total de vértices e custo menor.

Assim, como cada aresta e arco de G é duplicado no máximo uma vez em G', podemos afirmar que os multiconjuntos de arcos e arestas de G' estarão contidos nos multiconjuntos de G^* e por isso, o custo C' da solução dada pelo algoritmo MISTO ao grafo G deverá ser menor ou igual a C^*.

$$C' \leq C^* \leq 2C$$
$$C'/C \leq 2$$

Provando assim que o algoritmo MISTO é uma 2-aproximação para o problema do carteiro misto.

□

2.4 Problema do carteiro rural

Nesta versão mais geral do PCC devemos realizar o planejamento de um circuito de custo mínimo que passa ao menos uma vez por toda aresta pertencente a um subconjunto $R \subseteq E$. Isto é, não há a necessidade de se percorrer na solução todas arestas do grafo, apenas as pertencentes ao subconjunto R.

Denomina-se esta variação de Problema do Carteiro Rural (PCR).

Esse problema possui aplicações mais gerais que a sua versão original, por exemplo: pode-se modelar com ele a rota de um ônibus escolar, que deve percorrer as ruas dos alunos que ele atende, mas não necessariamente por todas ruas de um bairro.

O PCR é um problema NP-completo, como provado por Lenstra e Kan [LK76].

De modo similar aos outros casos do PCC, a solução aproximada para este problema consiste em estender G adicionando cópias de arestas de modo que o mesmo se torne euleriano e em sequência construir uma solução a partir do circuito euleriano do grafo estendido.

Para realizar a primeira parte da solução do problema, ou seja, criar uma extensão euleriana de G, serão apresentadas duas heurísticas, uma para cada grafos não direcionados e outra para digrafos.

As explicações e heurísticas tratadas a seguir foram baseadas no artigo de Eiselt de 1994 [EGL95].

2.4.1 Grafos não direcionados

Em 1979 Christofides [Chr76] sugeriu um algoritmo aproximado para o problema do caixeiro viajante que possuiu o melhor fator de aproximação para grafos gerais

por 30 anos. Até que, em junho de 2020, uma aproximação sutilmente melhor foi desenvolvida por Karlin, Klein e Gharan [KKG20].

Será apresentada uma adaptação do algoritmo de aproximação de Christofides para o PCR, sugerida por Frederickson [Fre79] e denominada **heurística da árvore geradora mínima**.

O algoritmo que será apresentado é uma $\frac{3}{2}$-aproximação polinomial para o PCR em grafos cuja função de custo de suas arestas respeita a desigualdade triangular. Chamaremos os grafos que respeitam essa característica quanto ao custo de grafos métricos.

Para resolver o PCR em grafos não direcionados define-se um grafo $G'(V', E')$, onde $V' = \{u \in V : (u, v) \in R \text{ para algum } v \in V\}$, e o novo conjunto de arestas E' é definido em dois passos:

1. Inicialmente adiciona-se a E' as arestas (v_i, v_j) para todo par $v_i, v_j \in V'$ cujo custo é igual ao custo do menor caminho de v_i a v_j em G

2. Retiram-se de E' as arestas $uv \in E' \setminus R$ se já existir outra aresta entre u e v de mesmo custo ou se existir um vértice w tal que $c_{uv} = c_{uw} + c_{wv}$ para algum vértice $w \in V'$, isto é, o custo da aresta uv é igual a soma do custo de outras duas arestas uw e wv.

Este procedimento em um grafo métrico remove vértices que não possuem adjacências em R e cria arestas condensadas que representam os caminhos apagados pela remoção de vértices. Por exemplo, na figura 2.15 o vértice d, do grafo original da figura 2.14, é apagado e substituído por duas arestas, de custos 3 e 5, representando os caminhos que passavam por d.

Pode-se visualizar um exemplo de tal separação a seguir:

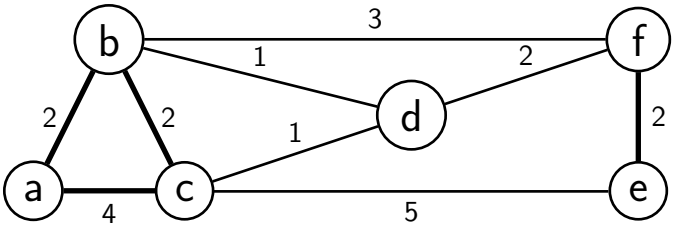


Figura 2.14: Exemplo de grafo G, destacam-se no mesmo as arestas pertencentes a R

Define-se o conjunto de vértices de G': $V' = \{a, b, c, e, f\}$. Exclui-se de tal conjunto apenas o vértice d, que não possui ligação com alguma aresta de R.

O conjunto E' é definido em duas etapas: primeiramente criam-se novas arestas entre os vértices de V' com custo igual ao custo da menor distância entre os mesmos vértices em G, tais arestas são representadas em azul na imagem 2.15.

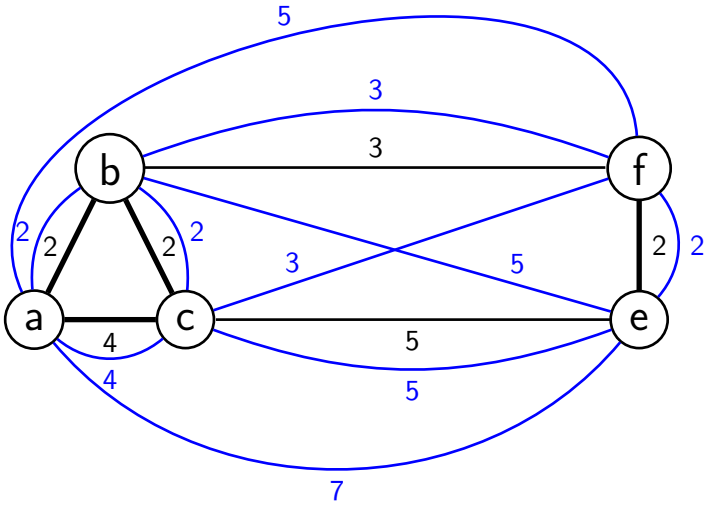


Figura 2.15: Grafo intermediário na transformação de G em G' (após o passo 1.)

Após a criação das novas arestas são deletadas do grafo todas arestas $uv \notin R$ tal que existe $w \in V'$ para que $c_{uv} = c_{uw} + c_{wv}$, assim como as arestas paralelas não pertencentes a R de mesmo custo.

No exemplo são retiradas de E' uma cópia das arestas ab, ac, bf, ce, ef pois já existem arestas paralelas a estas de mesmo custo. Retiram-se também as arestas: af já que $c_{af} = c_{ab} + c_{bf}$, be já que $c_{be} = c_{bf} + c_{fe}$ e ae já que $c_{ae} = c_{af} + c_{fe}$, sobrando assim apenas a aresta cf de custo 3.

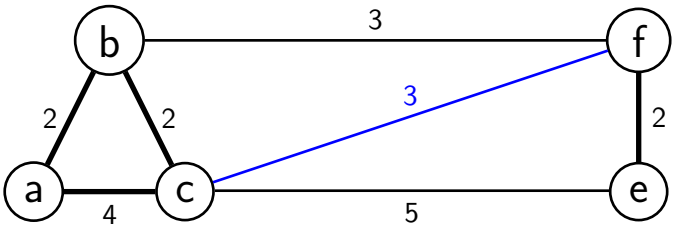


Figura 2.16: Grafo transformado G'

Considere o subgrafo de G' induzido pelas arestas de R. Este grafo possui k componentes conexas, que são denominadas $G_1, G_2, \ldots, G_k$.

No exemplo tal separação gera dois subgrafos G_1 e G_2, representadas na figura 2.17, cujos conjuntos de vértices são $V_1 = \{a, b, c\}$ e $V_2 = \{f, e\}$ respectivamente.

Figura 2.17: Separação induzida pelas arestas de R em G'

Segue agora uma descrição da heurística de Frederickson como apresentado por Gendreau e Laporte [EGL95].

Passo 1. Construir uma árvore geradora mínima T em G' que conecte os k subgrafos G_i induzidos por R.

No exemplo, uma árvore de custo mínimo que liga os subgrafos G_1 e G_2 tem custo 3, como se vê nas figuras 2.18 e 2.19, sendo composta pelos vértices c, f e a aresta entre os mesmos.

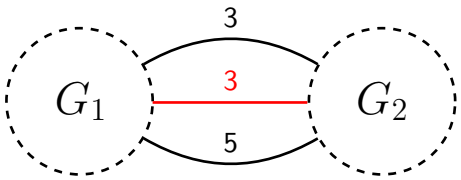


Figura 2.18: Árvore geradora mínima T representada em vermelho no grafo G' condensado

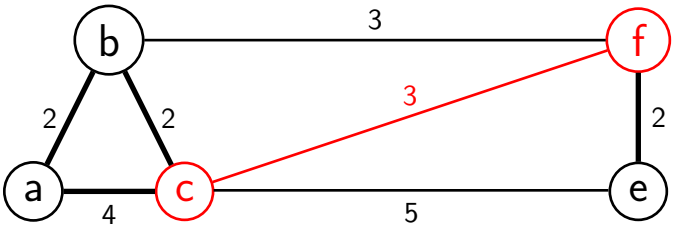


Figura 2.19: Representação de T em G'

Para a implementação desse passo, usou-se o algoritmo de Union-Find para comprimir as componentes G_i e Kruskal para encontrar a árvore geradora mínima do grafo comprimido.

Passo 2. Encontrar um emparelhamento perfeito M de custo mínimo entre os vértices de grau ímpar do grafo induzido por $R \cup T$.

No exemplo, os únicos vértices de grau ímpar em $R \cup T$ são os vértices c e e.

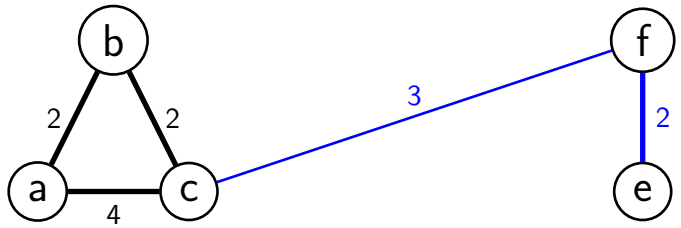


Figura 2.20: Grafo induzido por $R \cup T$ evidenciando o emparelhamento dos vértices c e e de custo mínimo

Passo 3. Encontrar um circuito euleriano no grafo induzido por $R \cup T \cup M$. A partir de tal circuito é possível derivar-se um circuito de mesmo custo no grafo G original que resolve o PCR.

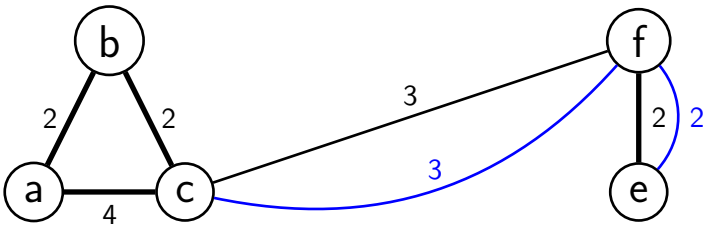


Figura 2.21: Grafo induzido por $R \cup T \cup M$

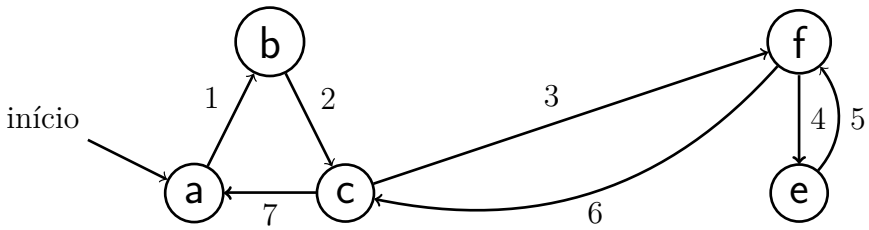


Figura 2.22: Circuito euleriano no grafo induzido

Segue agora o caminho que resolve o PCR no grafo original baseado no circuito euleriano da figura 2.22:

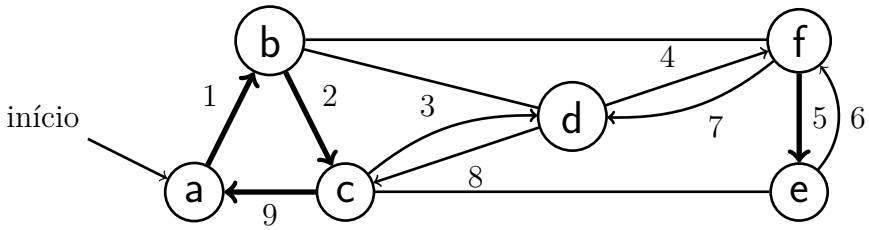


Figura 2.23: Solução do PCR encontrada pela heurística de Christofides

Totalizando um circuito que passa por todas arestas de R e tem custo igual a 18, o valor ótimo para uma solução do PCR no grafo de exemplo.

A solução descrita é polinomial, e foi implementada com complexidade $\mathcal{O}(|E||V|^2)$, como comentado em mais detalhes no capítulo 3 de implementação.

2.4.2 Grafos direcionados

Para tratar sobre a versão direcionada do PCR, tomaremos $G(V, A)$ como o digrafo original e um conjunto $R \subseteq A$ de arcos que devem ser percorridos no circuito que resolve o PCR para G.

Assim como no PCR para grafos não direcionados, a solução deste caso se baseia em estender o digrafo G em um digrafo euleriano, cujo circuito euleriano representa uma solução para o PCR do digrafo original.

Será apresentada uma heurística que resolve esse problema, proposta por Christofides [Chr+86], muito semelhante à heurística apresentada para o caso de grafos não direcionados. Nesse caso a heurística é denominada **heurística da arborescência geradora mínima**.

Para guiar a explicação toma-se, como exemplo, o digrafo G, a seguir:

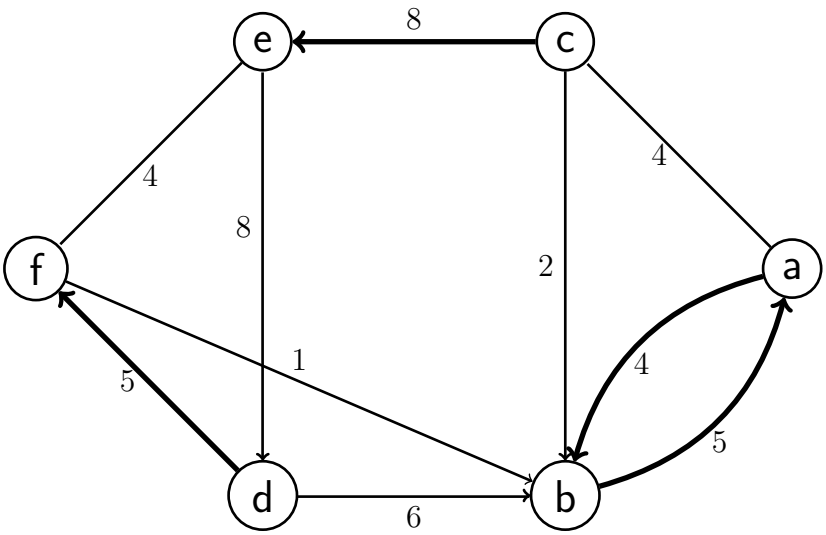


Figura 2.24: Digrafo G, o conjunto R corresponde aos arcos em negrito

A extensão de G em um digrafo euleriano $G'(V', A')$ se dá de modo similar àquela do caso com grafos não direcionados:

O conjunto de vértices possui a mesma definição, $V' = \{u \in V : uv \in R \text{ para algum } v \in V\}$.

Como no grafo exemplo da figura 2.24 todos vértices possuem ao menos um arco pertencente a R, define-se $V' = V$. No caso geral em que $V' \neq V$ é necessário criar novas arestas no grafo para representar os caminhos que passavam pelos vértices apagados, exatamente como foi feito no exemplo do PCR para grafos quando o vértice d foi deletado (vide figuras 2.14 e 2.15).

O conjunto de arcos A' será inicialmente acrescido dos arcos v_iv_j para todo par $v_i, v_j \in V'$, de custo igual ao custo do menor caminho de v_i a v_j. Posteriormente remove-se de A' todo arco que pertence também a $A \setminus R$ e cujo custo c_{uv} é igual a

$c_{uw} + c_{wv}$ para algum vértice w, removem-se também os arcos paralelos de mesmo valor que pertencem a $A \setminus R$.

Realizando tal extensão no exemplo sugerido, teremos o digrafo G' representado na figura 2.25.

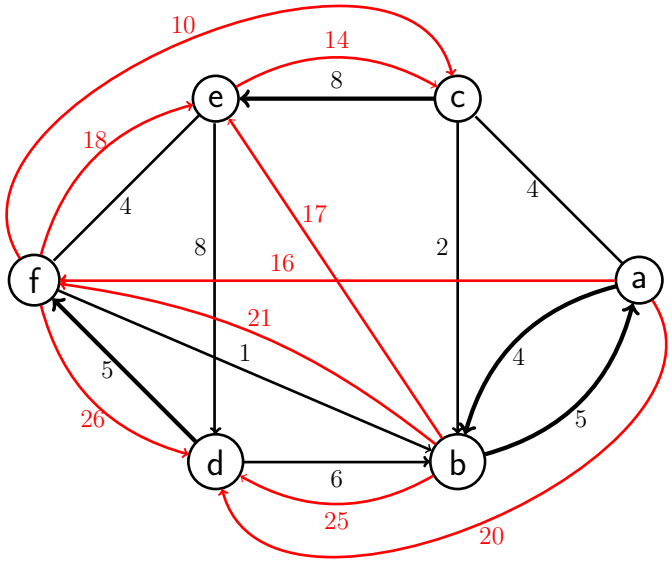


Figura 2.25: São vermelhas as arestas criadas na extensão G'

O grafo G' será composto por k conjuntos conexos $G_1, G_2, \ldots, G_k$, induzidos por R. Isto é, cada conjunto G_i será composto apenas por arcos pertencentes a R e será conexo desconsiderando a orientação dos arcos.

No exemplo apresentado, G possui três componentes, G_1 composta pelos vértices a, b, G_2 composta por c, e e G_3 composta por d, f, representadas na figura 2.26.

Passo 1. Encontrar uma arborescência geradora T de custo mínimo que conecte os subgrafos $G_1, \ldots, G_k$ e tem raiz em um vértice qualquer.

Para encontrar tal arborescência, pode-se utilizar o algoritmo proposto independentemente pela dupla Yoeng-Jin Chu e Tseng-Hong Liu e por Edmonds [Edm67], sendo, portanto, chamado de algoritmo de Chu-Liu/Edmonds.

Condensando o digrafo G' em suas componentes G_1, G_2, G_3 temos o grafo representado na figura 2.26.

O primeiro passo do algoritmo de Chu-Liu/Edmonds consiste em definir um vértice raiz r para a arborescência. Como a heurística de Christofides não requer que um vértice específico seja tal raiz, podemos escolher o vértice condensado G_1 para cumprir tal função ($r = G_1$).

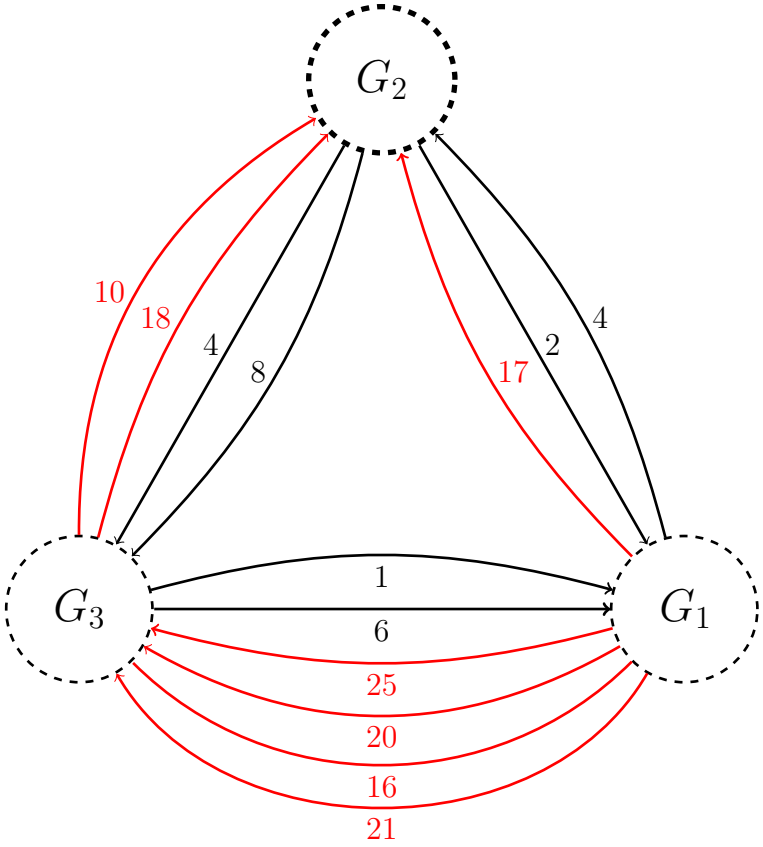


Figura 2.26: Digrafo G' condensado em suas componentes induzidas por R

Definida a raiz r, deve-se retirar do digrafo analisado todos arcos que tem como destino r. Além disso, pode-se também substituir qualquer conjunto de arcos paralelos por um único arco de menor custo. Pode-se visualizar o efeito de tais modificações em G' na figura 2.27

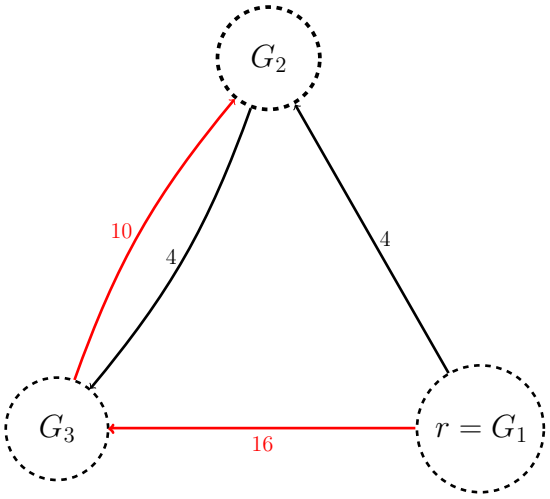


Figura 2.27: Digrafo G' após remoção de arcos do Passo 1

Para todo vértice v do grafo condensado diferente da raiz ($v \neq r$) encontra-se o

arco de menor custo que chega em v. Define-se como $\pi(v)$ o vértice origem de tal arco.

Se o conjunto de arcos $T = \{\pi(v)v \ \forall v \in \{G_1, \ldots, G_k\} : v \neq r\}$ não contém circuitos, então T é uma arborescência de custo mínimo enraizada em r.

Do contrário, realiza-se uma contração dos circuitos existentes em T, atualizam-se os custos dos arcos e repete-se recursivamente o mesmo procedimento de criação de T.

Pela definição do algoritmo, quando se contrai um circuito $C \subseteq T$ em um vértice $\mathcal{C}$ a atualização de custos para um arco uv segue o seguinte padrão:

- Se $v \in C$ e $u \notin C$, então define-se o arco $u\mathcal{C}$ com custo $c_{uv} - c_{\pi(v)v}$
- Se $u \in C$ e $v \notin C$, então define-se o arco $\mathcal{C}v$ de custo c_{uv}
- Se $u, v \notin C$, então o arco uv não é modificado
- Se $u, v \in C$ então uv é condensado na criação do vértice $\mathcal{C}$

No exemplo analisado, o conjunto T consiste nos arcos de G_1 a G_2 e G_2 a G_3, ambos de custo 4.

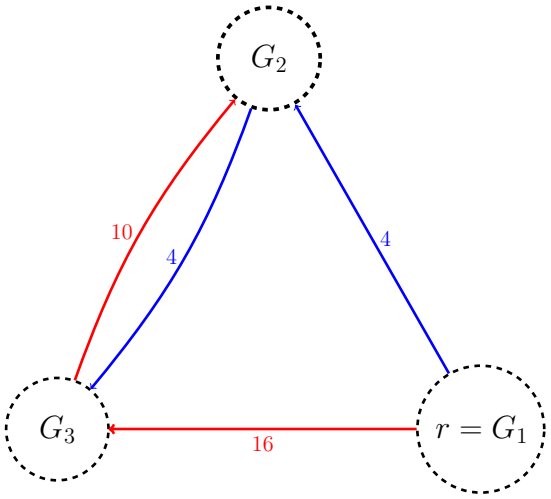


Figura 2.28: Em azul representa-se o conjunto de arcos T

Como T não possui circuitos, não é necessário realizar uma nova iteração do algoritmo. T é o conjunto de arcos que induz a arborescência de custo mínimo enraizada em G_1.

No próximo passo, tratamos de tornar o digrafo induzido pelos arcos de $R \cup T$ euleriano.

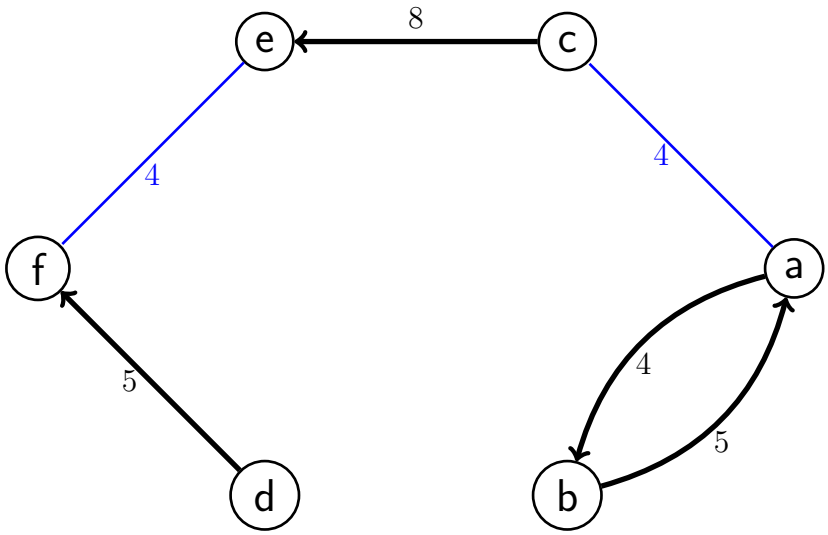


Figura 2.29: Digrafo induzido por $R \cup T$, com os arcos de T em azul

Passo 2. Encontrar um multiconjunto M de menor custo composto por arcos de A' que torna o digrafo induzido por $R \cup T$ euleriano, ou seja iguala os graus de entrada e saída de todos vértices no digrafo induzido por $R \cup T \cup M$.

Pode-se determinar M a partir da resolução de um problema de transporte: cuja base é o digrafo G', e cujas funções de oferta e demanda são definidas a partir dos graus dos vértices no digrafo induzido por $R \cup T$.

Um vértice $v \in V(G')$ cujo grau de entrada é maior que seu grau de saída (em relação ao grafo induzido por $R \cup T$) possuirá uma oferta igual ao valor absoluto da diferença de seus graus, do contrário, o valor absoluto da diferença representará a demanda do vértice v.

No exemplo abordado (figura 2.29) o vértice f possui uma oferta de valor 2, os vértices a e d possuem uma demanda de valor 1 e os vértices restantes já se encontram em igualdade de grau de entrada e saída.

As arestas u, v do problema de transporte modelado terão capacidade infinita e custo igual ao custo de menor caminho entre u e v no grafo original.

A resolução do problema de transporte modelado gera um conjunto de caminhos, representando os arcos que devem ser duplicados para criar um grafo euleriano, juntamente com os arcos de R e T.

Voltando ao exemplo, a solução do problema de transporte consiste na utilização dos caminhos de menor custo de f a a e de f a d, escoando uma unidade por cada um desses caminhos. Esta solução supre a oferta de duas unidades do vértice f assim como a demanda de uma unidade dos vértices d e a.

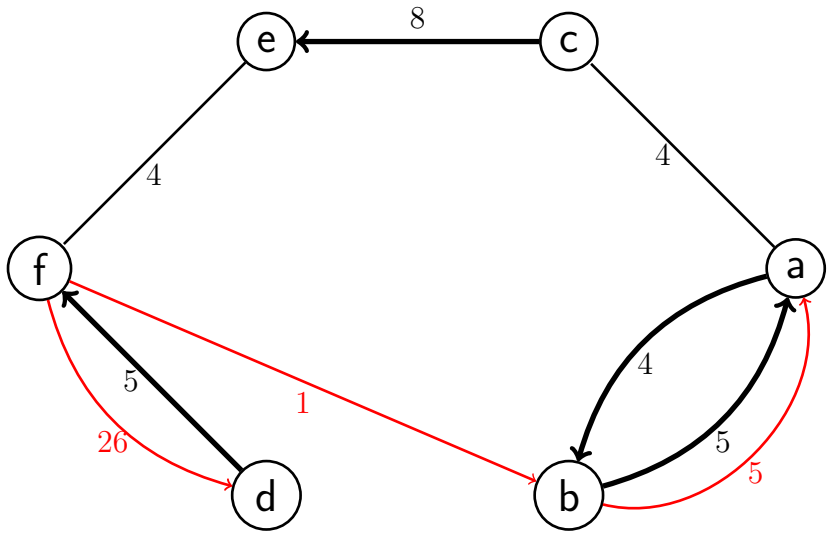


Figura 2.30: Digrafo euleriano $R \cup T \cup M$ com arcos de M representados em vermelho.

Na figura 2.30 pode-se visualizar o digrafo induzido pelos arcos $R \cup T \cup M$. Note que o arco de b a a de custo 5 é representado duas vezes, isso pois ele pertence tanto a R quanto a M.

Passo 3. O digrafo induzido pelo multiconjunto de arcos $R \cup T \cup M$ é, pela escolha de M, euleriano.

Um possível circuito euleriano para o exemplo apresentado é representado na figura 2.31 a seguir.

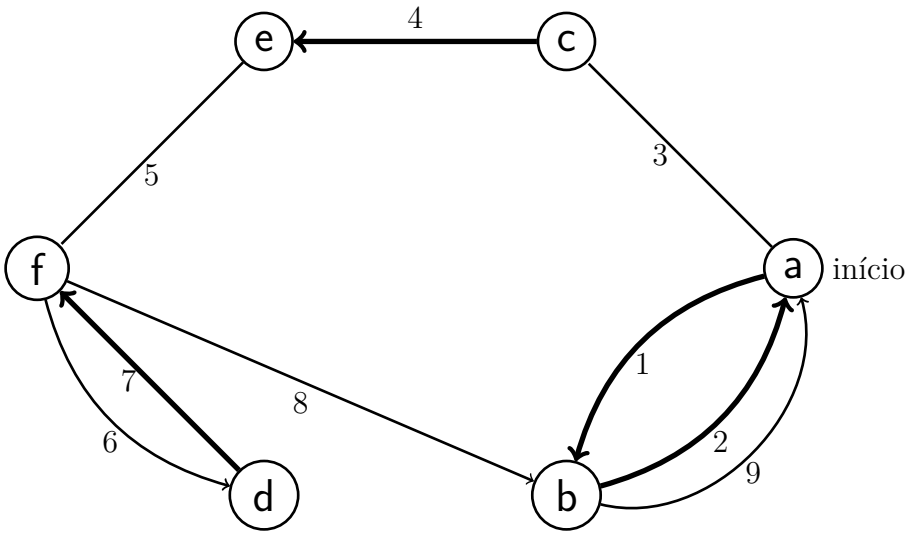


Figura 2.31: Circuito euleriano de $R \cup T \cup M$.

A partir de tal circuito pode-se encontrar uma solução para o PCR do digrafo original G expandindo os arcos contraídos na construção de G' e removendo as duplicatas de arcos artificiais da criação de um supergrafo euleriano. Por exemplo,

o arco do nó f ao d de custo 26 pertencente a M consiste na condensação do caminho mínimo de f a d: $\{f, b, a, c, e, d\}$.

Após a transformação do circuito euleriano em uma solução do PCR no digrafo original temos o seguinte passeio fechado (figura 2.32):

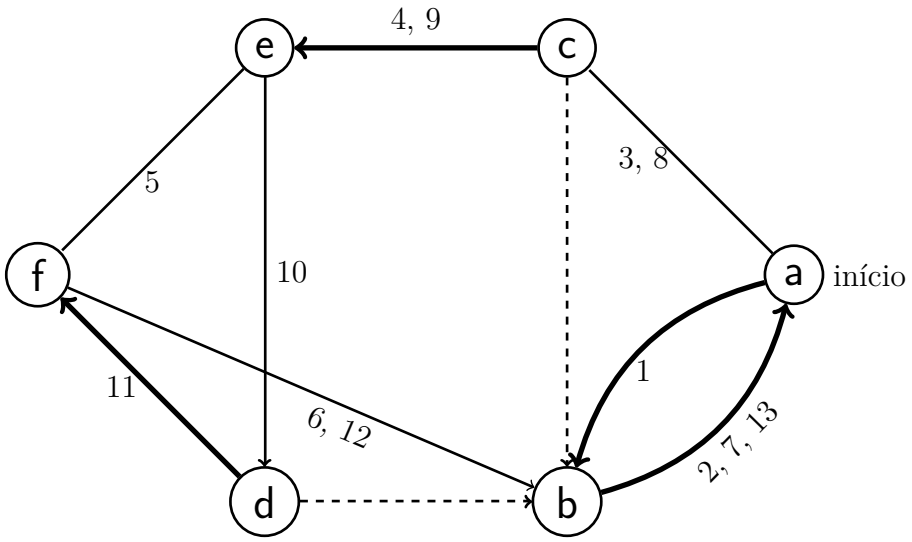


Figura 2.32: Circuito que resolve o PCR do grafo G. Representam-se em pontilhado os arcos de G não percorridos nesta solução.

Finaliza-se assim a execução da heurística da arborescência geradora mínima, que gera uma solução para o PCR de G com custo 62, um valor um pouco acima do custo da solução ótima 40, como demonstrado abaixo na figura 2.33:

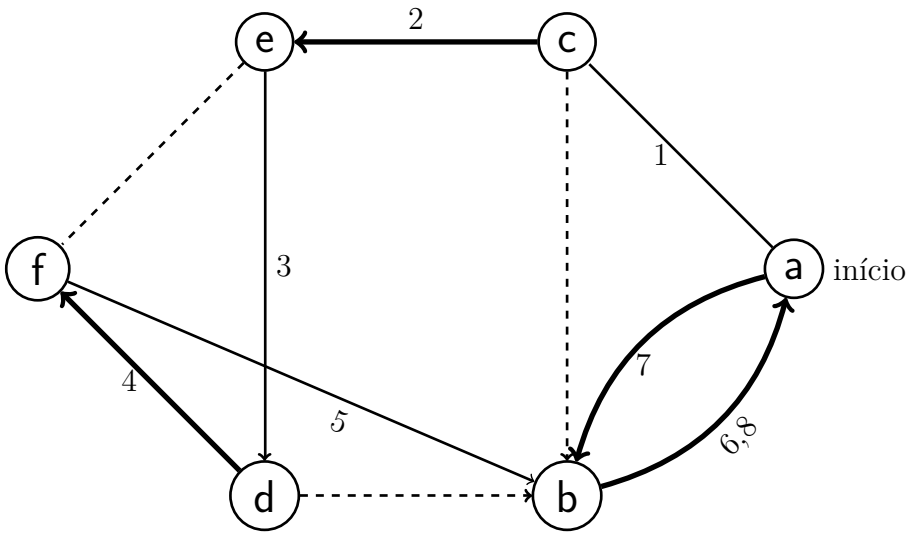


Figura 2.33: Uma solução ótima para o PCR no grafo do exemplo

2.5 Problema do carteiro chinês com vento

Considere o problema em que cada aresta tem um custo diferente dependendo do sentido em que é percorrida. Este problema é conhecido como o Problema do Carteiro Chinês com Vento (PCV).

Este problema possui várias aplicações, por exemplo, podem existir estradas com uma corrente de vento que o carteiro deve percorrer. A corrente de vento tanto pode ajudar o carteiro se ele anda no mesmo sentido que a corrente, quanto pode atrapalhar se ele anda no sentido contrário do vento.

Outra ilustração possível para esta variação é imaginar que uma estrada pode ser íngreme, sendo assim mais difícil subir tal estrada do que descê-la.

Vamos denominar um grafo que possui arestas com diferentes custos dependendo do sentido em que são percorridas de um grafo íngreme.

Esta variação do problema do carteiro chinês também é NP-difícil. Porém, como provado por Meigu Guan [Gua83], se todo circuito de um grafo íngreme G possui o mesmo custo independente do sentido em que é percorrido, então é possível encontrar uma solução para o PCV usando um algoritmo polinomial.

Seja c_{ij} o custo de se percorrer a aresta ij de i a j e c_{ji} o custo de percorrer a mesma aresta no sentido contrário.

O algoritmo sugerido por Guan se baseia em criar uma nova função de custo c' para o problema, tal que $c'_{ij} = c'_{ji} = \frac{c_{ij}+c_{ji}}{2}$ para toda aresta ij. Essa transformação reduz o problema a um PCC em um grafo não direcionado, que possui solução exata polinomial, vide seção 2.1.

Enuncia-se a seguir o teorema 2.5.1 que garante que uma resposta ótima do PCC no grafo (G, c') equivale a uma resposta ótima, de mesmo custo, do PCV no grafo íngreme original (G, c).

Lema 7. *Seja G um grafo cujos circuitos possuem os mesmos custos independente do sentido em que são percorridos. Seja C um circuito do grafo G, vale que:*

$$\sum_{ij \in C} c'_{ij} = \sum_{ij \in C} c'_{ji} = \sum_{ij \in C} c_{ij} = \sum_{ij \in C} c_{ji}$$

Demonstração. Como G respeita a propriedade que todo circuito possui custo igual, não importando a ordem em que o mesmo é percorrido, pode-se afirmar que:

$$\sum_{ij \in C} c_{ij} = \sum_{ij \in C} c_{ji}$$

Além disso, por definição $c'_{ij} = c'_{ji}$, valendo assim que:

$$\sum_{ij \in C} c'_{ij} = \sum_{ij \in C} c'_{ji}$$

Finalmente, pela definição de c':

$$\sum_{ij \in C} c'_{ij} = \sum_{ij \in C} \frac{c_{ij} + c_{ji}}{2}$$
$$\sum_{ij \in C} c'_{ij} = \frac{1}{2} \left(\sum_{ij \in C} c_{ij} + \sum_{ij \in C} c_{ji} \right)$$
$$\sum_{ij \in C} c'_{ij} = \sum_{ij \in C} c_{ij}$$

Finalizando assim a prova da igualdade dos quatro somatórios apresentados.

□

Teorema 2.5.1. *Seja G um grafo conexo não direcionado cujos circuitos possuem o mesmo custo, independente do sentido que são percorridos.*

Qualquer solução ótima para o PCC em (G, c') também é uma solução ótima do PCV no grafo íngreme (G, c).

Para exemplificar a prova deste teorema, usaremos o seguinte grafo G:

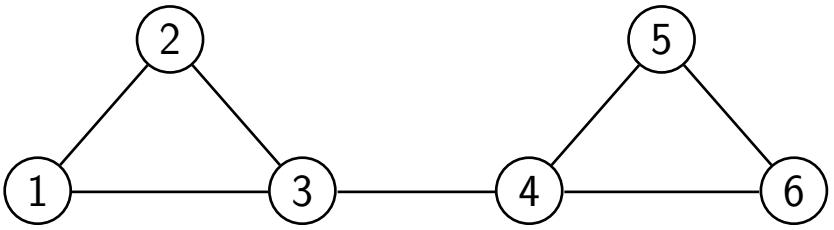


Figura 2.34: Grafo G, a função custo das arestas é definida na tabela 2.1

c	Destino					
Origem	1	2	3	4	5	6
1		2	3			
2	5		3			
3	4	1		7		
4			1		1	1
5				3		4
6				2	3	

Tabela 2.1: O custo de se percorrer a aresta de i a j é o valor da linha i e coluna j

Para melhor visualização destes custos, podemos representar o grafo original usando pares de arcos para cada aresta existente:

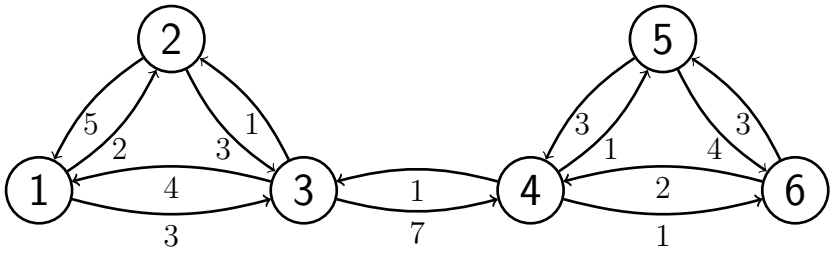


Figura 2.35: Representação direcionada do grafo G

Note que os dois circuitos dessa imagem possuem o mesmo custo independente do sentido em que são percorridos:

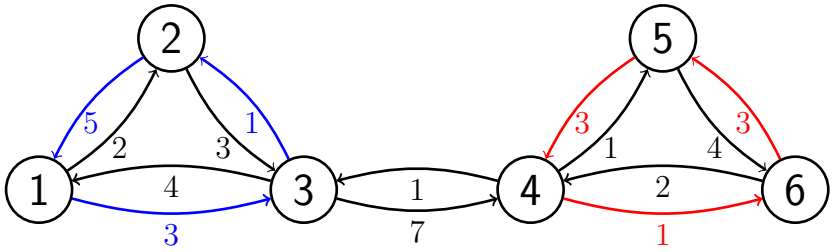


Figura 2.36: Circuitos com custo 9 e 7 respectivamente

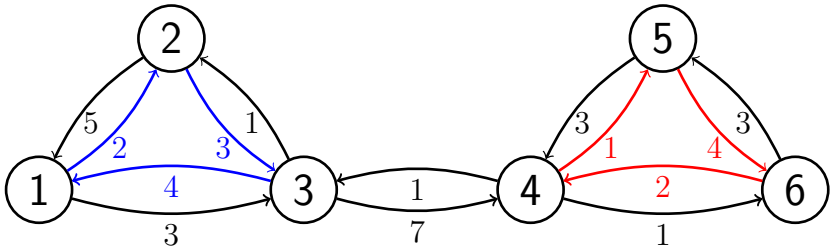


Figura 2.37: Circuitos no sentido contrário com mesmo custo, 9 e 7

Lembrando que os arcos de custo 1 e 7 entre os vértices 3 e 4 representam uma única aresta, por isso não representam um circuito.

Realizando a transformação da função do custo de toda aresta ij de c_{ij} para $c'_{ij} = \frac{c_{ij}+c_{ji}}{2}$ temos a seguinte representação de (G, c'):

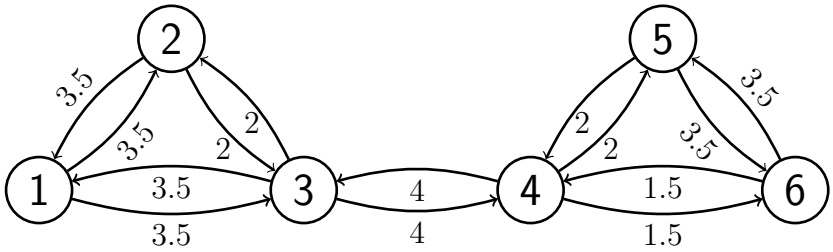


Figura 2.38: Representação direcionada do grafo G com custos c'

Definido o exemplo que seguiremos, podemos seguir com a prova do teorema 2.5.1

Demonstração. Seja T uma solução para o PCC em (G, c'). T será um passeio fechado composto por arestas orientadas de G.

Pode-se dividir T em k circuitos de arcos $C_1, C_2, \ldots, C_k$. Podemos expressar o custo da solução T como a soma dos custos de cada arco dos circuitos $C_1, C_2, \ldots, C_k$.

Provaremos por indução, que para um circuito qualquer C_i vale que:

$$\sum_{ij \in C_i} c_{ij} = \sum_{ij \in C_i} c'_{ij}$$

Seja $T = \{1, 2, 3, 4, 5, 6, 4, 3, 1\}$ uma solução ótima do PCV do exemplo da figura 2.38. Por si só, T é um circuito de arcos, já que não percorre um mesmo arco duas vezes, mas para exemplificar melhor os casos a seguir decompomos T em dois circuitos: $C_1 = \{1, 2, 3, 4, 3, 1\}, C_2 = \{4, 5, 6, 4\}$.

A indução é realizada com base no tamanho do circuito C_i analisado.

Há dois casos base desta indução:

- Quando o circuito analisado C_i é vazio.
- Quando C_i é um circuito presente no grafo original G. Isto é, C_i não percorre uma mesma aresta de G mais que uma vez. O lema 7 garante a hipótese de igualdade neste caso.

No exemplo que estamos tratando, o circuito C_2 se enquadra no segundo tipo de caso base, já que $C_2 = \{4, 5, 6, 4\}$ está no grafo original 2.34

Assume-se que a hipótese de igualdade de custos, $\sum_{ij \in C_i} c_{ij} = \sum_{ij \in C_i} c'_{ij}$, vale para circuitos com até k arcos.

Seja C_i um circuito que não faz parte do caso base e que possui $k + 1$ arcos.

Pela definição de circuitos, C_i não pode possuir arcos duplicados. Sendo assim, para C_i não estar presente em G, deverá existir uma aresta uv de G tal que $uv \in C_i$ e $vu \in C_i$, como representa-se a seguir:

$$C_i = \{w_1 w_2, \ldots, w_i u, uv, v w_{i+1}, \ldots, w_j v, vu, u w_{j+1}, \ldots w_k w_1\}$$

É possível, a partir de tal representação retirar de C_i os arcos uv e vu, separando-o em dois circuitos direcionados de tamanho menor:

$$C'_i = \{w_1 w_2, \ldots w_i u, u w_{j+1}, \ldots, w_k w_1\}$$
$$C''_i = \{v w_{i+1}, \ldots, w_j v\}$$

Como ambos novos circuitos tem tamanho menor que $|C_i| = k + 1$, vale a hipótese para ambos:

$$\sum_{ij \in C'_i} c_{ij} = \sum_{ij \in C'_i} c'_{ij} \tag{2.1}$$

$$\sum_{ij \in C''_i} c_{ij} = \sum_{ij \in C''_i} c'_{ij} \tag{2.2}$$

Além disso, vale pela definição da função c', que:

$$c_{uv} + c_{vu} = c'_{uv} + c'_{vu} \tag{2.3}$$

Somando as três igualdades apresentadas temos a representação do custo do circuito original C_i:

$$\sum_{ij \in C'_i} c_{ij} + \sum_{ij \in C''_i} c_{ij} + c_{uv} + c_{vu} = \sum_{ij \in C'_i} c'_{ij} + \sum_{ij \in C''_i} + c'_{ij} c'_{uv} + c'_{vu}$$
$$\sum_{ij \in C_i} c_{ij} = \sum_{ij \in C_i} c'_{ij}$$

Provando assim que a hipótese de indução também vale para um circuito de tamanho $k + 1$.

Finalmente, temos que o custo da solução T do PCV de (G, c) será igual ao custo da solução do PCC em (G, c'):

$$\begin{aligned} \sum_{ij \in T} c_{ij} &= \sum_{1 \leq u \leq k} \sum_{ij \in C_u} c_{ij} \\ &= \sum_{1 \leq u \leq k} \sum_{ij \in C_u} c'_{ij} \\ &= \sum_{ij \in T} c'_{ij} \end{aligned}$$

Para ilustrar os últimos passos desta prova, aplicaremos o argumento de quebra ao circuito C_1 do exemplo.

Apesar de C_1 ser um circuito válido na representação direcionada de G (figura 2.35), o mesmo não pode ser considerado um circuito do grafo não direcionado G pois percorre a mesma aresta $(3 - 4)$ duas vezes, devemos portanto aplicar o passo da indução a C_1.

Representando C_1 como um conjunto de arcos percorridos:

$$C_1 = \{3 \to 4, 4 \to 5, 5 \to 6, 6 \to 4, 4 \to 3\}$$

Realizando a quebra de C_1 a partir dos arcos $3 \to 4$ e $4 \to 3$ temos:

$$C'_1 = \{4 \to 5, 5 \to 6, 6 \to 4\}$$
$$C''_1 = \{\}$$

Como C'_1 é um circuito presente no grafo original e C''_1 é um conjunto vazio, ambos são cobertos pelos casos base estabelecidos.

Finalmente, pode-se comparar o custo da solução T do PCV de G com as duas funções de custo definidas: c e c'.

$$
\begin{aligned}
T &= \{1, 2, 3, 4, 5, 6, 4, 3, 1\} \\
\sum_{ij \in T} c_{ij} &= c_{12} + c_{23} + c_{34} + c_{45} + c_{56} + c_{64} + c_{43} + c_{31} \\
&= 2 + 3 + 7 + 1 + 4 + 2 + 1 + 4 \\
&= 24 \\
\sum_{ij \in T} c'_{ij} &= c'_{12} + c'_{23} + c'_{34} + c'_{45} + c'_{56} + c'_{64} + c'_{43} + c'_{31} \\
&= 3.5 + 2 + 4 + 2 + 3.5 + 1.5 + 4 + 3.5 \\
&= 24
\end{aligned}
$$

Como provado, ambas funções de custo são equivalentes para o cálculo do custo da solução T. □

Implementou-se este algoritmo utilizando o algoritmo que resolve o PCC em grafos como sub-rotina, o que simplificou muito esta implementação.

Implementou-se também uma função de complexidade $\mathcal{O}(|V| + |E|)$ baseada em uma busca em profundidade que checa se todos circuitos de grafo íngreme possuem o mesmo custo independente do sentido em que são percorridos.

Capítulo 3

Código desenvolvido

Estão disponíveis no GitHub as implementações em `C++17` para todos os algoritmos apresentados na monografia, com comentários explicando cada estrutura e método desenvolvidos.

Para acessá-las, basta navegar à pasta *code* do repositório.

Nesse capítulo explica-se brevemente a organização do código produzido e detalhes das implementações realizadas.

3.1 Estruturas de dados

São disponibilizados códigos gerais para estruturas de dados, focados na área de grafos:

- `aresta.hpp`

  Define a `struct Aresta`, usada para modelar tanto aresta direcionadas quanto não direcionadas.

  Toda aresta possui um valor `prox` indicando o vértice com o qual ela se liga, um identificador `id` e um custo real `cus`.

- `aresta-ingreme.hpp`

  Define a `struct ArestaIngreme`, modelando uma aresta que possui custos diferentes dependendo do sentido em que é percorrida.

  Além de armazenar os mesmos valores `prox` e `id` de uma aresta comum, também armazena `dirCus` e `invCus`, custos para se percorrer uma aresta na direção do vértice `prox` e no sentido inverso, respectivamente.

- `grafo.hpp`

  Define a `struct Grafo`, com `n` vértices e `m` arestas, armazenadas em uma lista de adjacências `adj`.

  Utiliza a `struct Aresta` para representar as arestas do grafo.

- `digrafo.hpp`

Analogamente, define a `struct Digrafo`, com `n` vértices e `m` arcos, armazenados em uma lista de adjacências `adj`. Além disso, também possui como parâmetros os vetores `grauEntrada` e `grauSaida` que indicam o grau dos vértices do digrafo.

Assim como `grafo.hpp`, utiliza a `struct Aresta` para representar os arcos do digrafo.

A `struct` possui também um método chamado `countSCC` que conta o número de componentes fortemente conexas do digrafo em questão, utilizando o algoritmo de Tarjan, de complexidade $\mathcal{O}(|V| + |E|)$.

- `grafo-misto.hpp`

  Define a `struct Misto` que representa grafos que possuem tanto arestas quanto arcos.

  Além das propriedades `n, m, adj`, esta estrutura contem um contador `nArestas` que conta a quantidade de arestas presentes no grafo, além de auxiliar na identificação de arcos e arestas pelos seus identificadores.

  Para facilitar a manipulação desta estrutura, a `struct Misto` contêm métodos auxiliares como:

  - Devolver o grau total (`grauTotal(v)`), grau de entrada (`grauEntrada(v)`) e saída (`grauSaida(v)`) de todo vértice
  - Contar o número de componentes fortemente conexas, usando o algoritmo de Tarjan de modo similar à `struct Digrafo`
  - Checar se uma aresta, dada um identificador é arco (`arco(id)`) ou aresta (`aresta(id)`)

- `grafo-ingreme.hpp`

  Define a `struct GrafoIngreme` usada na modelagem do problema do carteiro com vento, que possui os parâmetros básicos `n, m, adj` representando o número de vértices, arestas e a lista de adjacências do grafo.

  Este grafo utiliza a estrutura `ArestaIngreme` para representar suas arestas.

- `union-find.cpp`

  Essa implementação da estrutura "Union Find" utiliza as otimizações de compressão de caminhos, encurtando as relações de ancestralidade sempre que possível, e a união das componentes de menor tamanho sob as componentes maiores. Atingindo assim uma complexidade amortizada $\mathcal{O}(\log^* |V|)$ por operações de busca e união.

  O algoritmo pode ser utilizado por meio da `struct UnionFind` definida neste arquivo, que possui os métodos `raiz(v)` que encontra a raiz da componente de um vértice `v` e `join(u, v)` que une as componentes dos vértices `u` e `v`.

3.2 Algoritmos auxiliares

- `mst.cpp`

  Implementa o algoritmo de Kruskal para encontrar a árvore geradora mínima de um grafo não direcionado, com complexidade $\mathcal{O}(|E| \log |E|)$.

- `floyd-warshall.cpp`

  O algoritmo de Floyd-Warshall foi utilizado amplamente para calcular a menor distância entre todos pares de vértices de um grafo.

  Neste mesmo arquivo estão dispostas três implementações do método `floyd_warshall`, uma para grafos, outra para digrafos e a última para grafos mistos.

  Todas essas possuem complexidade $\mathcal{O}(|V|^3)$, como definido é característico deste algoritmo.

- `problema-transporte.cpp`

  Esse programa modela a `struct ProblemaTransporte` que resolve o problema de distribuir uma oferta de vértices em um conjunto `F` para vértices de outro conjunto `S` que possuem requisições de demanda.

  O grafo do problema de transporte deve ser `F,S`-bipartido e completo, com custos definidos para cada aresta do mesmo.

  A solução deste problema é um conjunto de arestas escolhidas para realizar o transporte, juntamente com a quantidade de fluxo que é passada por cada aresta.

  Esta modelagem consiste em uma simples abstração usada em cima do problema do fluxo máximo de custo mínimo, por isso possui a mesma complexidade que tal algoritmo, $\mathcal{O}(|E|^2|V|^2)$.

- `min-cost-flow.cpp`

  Para resolver o problema do fluxo de custo mínimo usou-se o algoritmo de Edmonds-Karp, que possui complexidade $\mathcal{O}(|E|^2|V|^2)$. A implementação disponibilizada se baseou no artigo [E-m14] do site cp-algorithms.com

Para facilitar o desenvolvimento, usaram-se também adaptações de implementações já prontas dos seguintes algoritmos:

- `ssa.cpp`

  O algoritmo de Chu-Liu/Edmonds que encontra uma arborescência geradora mínima foi implementado no arquivo `ssa.cpp`, que referencia o nome em inglês deste problema: *shortest spanning arborescence*.

  Adaptou-se o código disponibilizado pela equipe argentina universitária *el-vasito* em seu repositório do GitHub para implementar a `struct ChuLiu` deste arquivo. Esta implementação possui complexidade $\mathcal{O}(|E||V|)$

- `min-cost-matching/`

  Esta pasta é um submódulo do GitHub para um repositório próprio baseado (usou-se um "fork" no GitHub) na implementação de Dilson Lucas Pereira, estudante da Universidade Federal de Lavras.

  São utilizados desta subpasta códigos que resolvem os problemas:

  – Emparelhamento perfeito de custo mínimo
  – Emparelhamento de cardinalidade máxima

  Ambos problemas são resolvidos usando uma combinação de uma heurística de geração de emparelhamento aliada ao algoritmo Blossom de Edmonds, que tem complexidade de pior caso $\mathcal{O}(|E||V|^2)$.

3.3 Implementações

Esta seção apresenta os códigos implementados que resolvem os problemas tratados na monografia.

O padrão de implementação que foi seguido é criar uma `struct` que contém os métodos e parâmetros necessários para se resolver o problema.

Geralmente, o método `solve` ou `solveById` de cada `struct` é aquele que devolve a solução desejada. Como a solução de tanto do problema de Euler quanto do Carteiro Chinês são passeios em um grafo, usou-se dois meios de se representar tal passeio

- Um vetor de vértices, retornado geralmente pelo método `solve`
- Um vetor de identificadores de arestas do passeio, retornado geralmente pelo método `solveById`

Apesar de ser mais intuitiva a representação de um passeio por vértices pode ser ambígua nos casos de arestas paralelas, por isso opta-se preferencialmente pelo uso do método `solveById`.

3.3.1 Euler

Foram implementadas versões do algoritmo de Hierholzer, explicado na seção 1.4 para grafos não direcionados, direcionados e mistos, respectivamente:

- `euler-grafo.cpp`
- `euler-digrafo.cpp`
- `euler-misto.cpp`

Os algoritmos implementados possuem complexidade da ordem $\mathcal{O}(|V| + |E|)$.

3.3.2 Problema do Carteiro Chinês

- `pcc-grafo.cpp`

  Implementa a `struct PCC` que resolve o problema do carteiro chinês em grafos não direcionados.

  O algoritmo implementado tem complexidade $\mathcal{O}(|E||V|^2)$, devido a necessidade de utilizar o algoritmo do emparelhamento perfeito de custo mínimo, além do algoritmo Floyd-Warshall.

- `pcc-digrafo.cpp`

  Também implementa a `struct PCC`, porém neste caso resolve o problema do carteiro chinês para digrafos.

  Utiliza o algoritmo que resolve o problema de transporte, implementado em `problema-transporte.cpp`, por isso possui complexidade $\mathcal{O}(|E||V|^2)$.

- `pcc-misto.cpp`

  Implementa a 2-aproximação descrita por Frederickson [Fre79] que resolve o problema do carteiro chinês em grafos mistos com complexidade $\mathcal{O}(|V|^4)$, devido a utilização do algoritmo que resolve o problema de transporte em um grafo bipartido completo com $\mathcal{O}(|V|)$ vértices.

- `pcr-grafo.cpp`

  Implementa a $\frac{3}{2}$-aproximação que resolve o problema do carteiro rural em grafos métricos, isto é, cuja função custo respeita a desigualdade triangular. Tal solução foi desenvolvida a partir de um artigo de Eiselt [EGL95] e originalmente sugerida por Frederickson [Fre79].

  O algoritmo desenvolvido é polinomial, com complexidade dominada pelo algoritmo de emparelhamento perfeito de custo mínimo $\mathcal{O}(|E||V|^2)$.

  Também se usam se na implementação o algoritmo de Floyd-Warshall, a estrutura de "Union Find" para comprimir as componentes induzidas por R (conjunto de arestas essenciais do problema), o algoritmo de Kruskal para encontrar uma árvore geradora mínima e, finalmente, o algoritmo de Hierholzer, que encontra o circuito euleriano que serve como base da solução do problema.

- `pcr-digrafo.cpp`

  Implementa a heurística que resolve de maneira aproximada o problema do carteiro rural em digrafos.

  Utiliza como sub-rotinas o algoritmo de Chu-Liu do arquivo `ssa.cpp`, que possui complexidade $\mathcal{O}(|E||V|)$ e o algoritmo do Problema de Transporte, $\mathcal{O}(|E|^2|V|^2)$.

  Assim como a solução para o caso rural em grafos não direcionados, esta solução foi baseada no artigo de Eiselt [EGL95] e originalmente desenvolvida por Christofides et al. [Chr+86].

- `pcv-guan.cpp`

  Implementa o algoritmo descrito por Guan [Gua83] que resolve o problema do carteiro chinês com vento no caso em que todos circuitos possuem o mesmo custo independente do sentido em que são percorridos.

  Esta solução é polinomial e exata, com complexidade igual à do problema do carteiro em grafos direcionados $\mathcal{O}(|E||V|^2)$.

  O código implementado também checa se o grafo em questão respeita a característica especial do custo de seus ciclos por meio de uma busca em profundidade, tendo assim complexidade $\mathcal{O}(|V| + |E|)$, vide método `checkCyclesCost`.

3.4 Testes

Todos os testes estão disponíveis na pasta `test` do repositório, organizados do mesmo modo que os arquivos de código. Por exemplo, o arquivo `test/pcc-grafo.cpp` implementa testes para o arquivo homônimo da pasta `code`.

Utilizou-se o framework googletest para o desenvolvimento dos testes deste projeto. A configuração de compilação de código e execução de testes é automatizada com o `Makefile` da pasta `test`, basta rodar `make` que todos os testes são executados, como se demonstra na figura 3.1.

```
~/chinese-postman/test$ make

g++ -isystem gtest/googletest/include -g -Wall -Wextra -pthread -std=c++17 -fprofile-arcs -ftest-coverage -lgcov
-lpthread min-cost-matching/MCM.cpp gtest_main.a -o min-cost-matching/MCM
./min-cost-matching/MCM --gtest_color=yes
Running main() from gtest/googletest/src/gtest_main.cc
[==========] Running 1 test from 1 test suite.
[----------] Global test environment set-up.
[----------] 1 test from MCPM
[ RUN      ] MCPM.SimpleMatching
[       OK ] MCPM.SimpleMatching (0 ms)
[----------] 1 test from MCPM (0 ms total)

[----------] Global test environment tear-down
[==========] 1 test from 1 test suite ran. (0 ms total)
[  PASSED  ] 1 test.
g++ -isystem gtest/googletest/include -g -Wall -Wextra -pthread -std=c++17 -fprofile-arcs -ftest-coverage -lgcov
-lpthread pcr-grafo.cpp gtest_main.a -o pcr-grafo
./pcr-grafo --gtest_color=yes
Running main() from gtest/googletest/src/gtest_main.cc
[==========] Running 9 tests from 1 test suite.
[----------] Global test environment set-up.
[----------] 9 tests from PCRGrafo
[ RUN      ] PCRGrafo.Simples
...
```

Figura 3.1: Rodando os testes usando o comando `make`

Além disso, utiliza-se o site `codecov.io` para manter atualizadas as últimas informações de cobertura de testes.

Acessando o codecov é possível ver informações como: a porcentagem de linhas de código cobertas por testes (figura 3.2), quantas vezes uma linha específica é executada por testes desenvolvidos e linhas de código que ainda não são cobertas por nenhum teste (linhas em vermelho na figura 3.3).

Essa ferramenta foi extremamente útil para identificar possíveis casos de borda que não eram testados pelos algoritmos que implementei, pois apontava claramente quais linhas de código não tinham sido ainda testadas. Cool!

```
292      /// Conta o número de componentes fortemente conexas em 'digrafo'
293      /// Implementação do algoritmo de Tarjan
294  1   int countSCC() {
295  1       vis.clear();
296  1       vis.resize(n, false);
297  1       cy.clear();
298  1       cy.resize(n, -1);
299  1       id.resize(n);
300  1       mn.resize(n);
301  1       nc = 0;
302  1       idd = 0;
303  1       while(!q.empty())
304              q.pop();
305  1       for (int ini = 0; ini < n; ini++) {
306  1           if (!vis[ini])
307  1               tarjan(ini);
308          }
309  1       return nc;
310      }
```

Figura 3.2: Visualização antiga do codecov para o arquivo `digrafo.hpp`

Além disso, usou-se a ferramenta Travis CI atrelada ao GitHub para automatizar as seguintes tarefas a cada "push" realizado no repositório:

- Testar a compilação dos arquivos latex da monografia
- Rodar todos testes implementados, checando seu funcionamento, assim como é mostrado na figura 3.1
- Enviar o relatório dos testes executados ao codecov, atualizando assim as informações de cobertura mencionadas

Todas estas configurações estão definidas no arquivo `.travis.yml` do repositório.

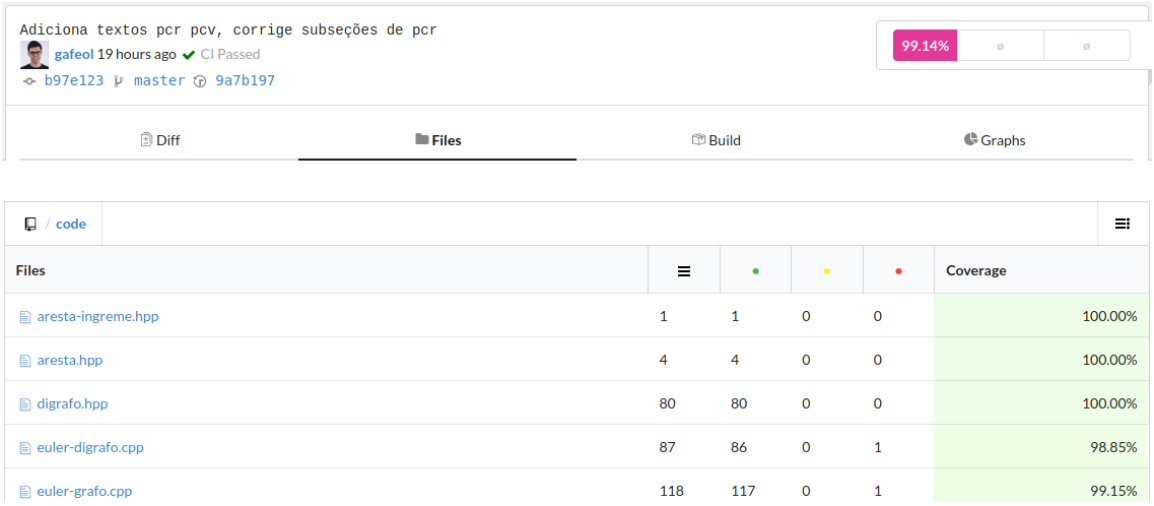


Figura 3.3: Visualização da porcentagem de cobertura

Apêndice

Apêndice A

Aplicações em problemas

Esta seção se dedica a discutir aplicações do Problema de Euler e do Problema do Carteiro Chinês em competições de programação, como a Maratona de Programação ou a ICPC.

As soluções implementadas estão disponibilizadas na pasta `probs` do repositório do GitHub.

A.1 Tanya and Password

URL do problema: codeforces.com/contest/508/problem/D

Solução: github.com/gafeol/chinese-postman/blob/master/probs/cf/508D.cpp

Este problema fornece um multiconjunto $\mathcal{C}$ de n strings de tamanho 3 e pede para que se construa, se possível, uma string S de tamanho $n+2$ com a restrição de que o conjunto das substrings de tamanho 3 da string S deve corresponder a $\mathcal{C}$.

Uma instância desse problema é a seguinte, para $n = 5$:

$$\mathcal{C} = \{aca, aba, aba, cab, bac\}$$

Uma string S que resolve este exemplo é *abacaba*, já que existe uma bijeção entre toda substring de tamanho 3 de S e $\mathcal{C}$. A string *abacab*, no entanto, não satisfaz a restrição do problema já que a mesma não possui duas substrings *aba*, como ocorre em $\mathcal{C}$.

A solução que será apresentada para este problema envolve a teoria de caminhos eulerianos apresentada.

Iniciaremos por explicar a modelagem realizada: Cada vértice do digrafo que construiremos representará um conjunto de duas letras.

Representaremos cada string $w \in \mathcal{C}$ com um arco uv. O vértice u representa os dois primeiros caracteres de w, enquanto que o vértice v representa os dois últimos caracteres de w.

Para exemplificar tal procedimento tome $w = aca$. Criam-se primeiramente dois vértices, um representando a string ac e outro representando ca, e liga-se ambos com um arco, como representado na figura A.1.

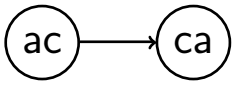
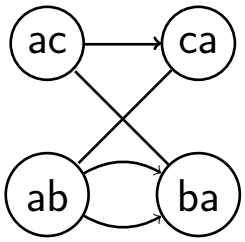


Figura A.1: Exemplo da modelagem usada na solução

Deve repetir-se tal procedimento para toda string de $\mathcal{C}$. Segue a modelagem completa, que chamaremos de G, do exemplo inicial ($\mathcal{C} = \{aca, aba, aba, cab, bac\}$):



A partir da correspondência entre arcos e strings podemos representar um passeio, P, em G como uma sequência de strings, seq, subconjunto de $\mathcal{C}$:

O passeio $P = \{ba, ac, ca, ab\}$, por exemplo, percorre os arcos que correspondem à sequência $seq = \{bac, aca, cab\}$.

Por sua vez, a partir de seq é possível montar uma string S que possua todas strings de seq como substrings:

$$seq = \{bac, aca, cab\} \rightarrow S = \text{“}bacab''$$

Tal procedimento permite que, a partir de um passeio P em G construa-se uma string S tal que:

- O tamanho de S é igual ao número de arcos percorridos em P mais 2;
- Se um arco e é percorrido em P, então a string de $\mathcal{C}$ que e representa será uma substring de S.

Como o problema pede que encontremos uma string de tamanho $n + 2$ que possua todas strings de $\mathcal{C}$ como substrings, basta encontrar, se existir, uma trilha que percorra todo arco de G uma única vez, isto é, uma trilha ou um circuito euleriano do grafo modelado.

Deste modo a solução do problema consiste em checar as propriedades necessárias para a existência de uma trilha euleriana, que são estabelecidas neste trabalho pelo corolário 2.

Se uma trilha ou circuito euleriano existir em G, podemos usar o algoritmo de Hierholzer para encontrar tal passeio.

A.2 Sereja and the Arrangement of Numbers

URL do problema: https://codeforces.com/problemset/problem/367/C
Solução: github.com/gafeol/chinese-postman/blob/master/probs/cf/367C.cpp

No enunciado do problema define-se uma sequência S como "bonita" quando existe um inteiro i para cada par de valores distintos u, $v \in S$, tal que $S_i = u, S_{i+1} = v$ ou $S_i = v, S_{i+1} = u$.

Em outras palavras, uma sequência é bonita se todo par de valores distintos pertencentes à essa sequência aparecem na mesma lado a lado ao menos uma vez.

Define-se também um sistema de recompensas: para cada valor há uma recompensa positiva associada. A recompensa total de uma sequência é igual à soma das recompensas de seus valores distintos, ou seja, a recompensa não depende do número de vezes que cada valor aparece na sequência.

O problema consiste então desenvolver um algoritmo que calcule a maior recompensa atingível para uma sequência bonita de tamanho n que seja constituída apenas por valores definidos pela entrada.

A entrada do programa, portanto será dada pelos valores n, m e por m pares q_i, w_i, indicando que q_i é um valor que pode ser utilizado e que possui recompensa w_i, garante-se que os valores de q_i são distintos e que w_i é positivo.

Um exemplo de entrada para este problema é:

5 4	Neste exemplo $n = 5$ e $m = 4$
1 4	O primeiro valor é 1, e sua recompensa é 4
2 3	Segue o valor 2, com recompensa 3
4 2	Segue o valor 4, com recompensa 2
3 10	Finalmente temos o valor 3, de recompensa 10

Para este exemplo, uma solução ótima é a sequência $1, 2, 3, 1, 1$, que é bonita e possui recompensa total igual a 17. Não existe uma sequência bonita que possua os quatro valores do exemplo.

Como todo valor possui uma recompensa positiva, a solução final deverá usar a maior quantidade de valores distintos que for possível, priorizando a utilização dos valores de maior recompensa.

Começaremos discutindo um problema mais simples: Qual o tamanho da menor sequência bonita que utiliza x valores distintos?

Para resolver esse problema faremos uma modelagem do problema usando um grafo completo K_x, que possui x vértices, onde cada vértice representa um valor distinto e cada aresta possui custo unitário.

Todo passeio do grafo K_x pode ser representado como uma sequência dos valores dos vértices do grafo, as sequências definidas como bonitas são aquelas que representam um passeio que percorre todas arestas do grafo K_x.

Para facilitar a análise deste problema, assumiremos que pode existir em um passeio dois vértices de mesmo valor em sequência, mesmo sem haver um laço neste vértice, além disso, vamos assumir que os valores distintos são $1, 2, \ldots, x$.

Sendo assim, podemos interpretar uma sequência S que possua apenas valores entre 1 e x como um passeio em K_x, os valores da sequência representam os vértices do grafo K_x, e dois valores adjacentes u e v representam que a aresta uv está presente

no passeio derivado de S.

Para que a sequência S seja bonita, todo par de valores distintos u, v deverá aparecer ao menos uma vez lado a lado, o que implica que o passeio derivado de S deverá percorrer a aresta uv ao menos uma vez.

Concluímos então que um passeio derivado de uma sequência que seja bonita, deverá percorrer todas arestas do grafo K_x, sendo assim uma solução do problema do carteiro chinês para o mesmo.

Voltamos assim à pergunta inicial, a menor sequência bonita que utiliza x valores distintos, será aquela que representa uma solução ótima do problema do carteiro chinês para o grafo K_x, já que a solução que minimiza o custo do PCC também minimizará o número de arestas percorridas, e portanto, o tamanho da sequência bonita.

Vejamos agora alguns exemplos:

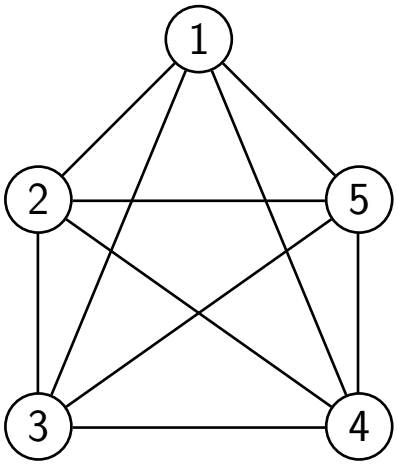


Figura A.2: Exemplo com $x = 5$

Para $x = 5$, o grafo induzido é o K_5, como representado na figura A.2. Como todos os vértices do K_5 possuem grau par, a solução ótima do Problema do Carteiro Chinês para este exemplo é também um circuito euleriano, como o seguinte:

$$P = \{1, 2, 3, 4, 5, 1, 3, 5, 2, 4, 1\}$$

Como discutido anteriormente, P, além de solução do PCC é também uma sequência bonita que utiliza os x valores definidos pelos vértices do grafo.

Sendo assim, para utilizar 5 valores distintos, é necessário ter uma sequência de tamanho no mínimo 11, valor este que é igual a $1 + |E(K_5)| = 1 + 10 = 11$.

Analisaremos agora outro exemplo, em que $x = 4$, do qual se deriva o grafo K_4 representado na figura A.3.

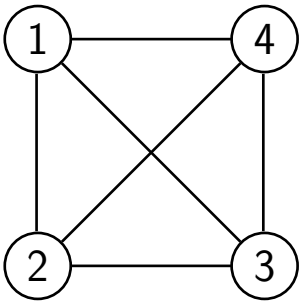


Figura A.3: Exemplo com $x = 4$

Neste exemplo, todos os vértices de K_4 possuem grau ímpar, por isso a primeira parte da solução do PCC se baseia em duplicar algumas arestas do grafo, tornando-o euleriano.

Como estamos tratando de um grafo completo, com arestas de custo unitário, não é necessário criar uma condensação do grafo K_x, e achar um emparelhamento perfeito de custo mínimo. Basta apenas duplicar uma aresta entre todo par de vértices de índices i e $i + 1$ para todo índice i ímpar, como representado na figura A.4 para o exemplo em questão.

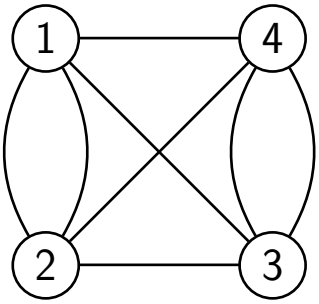


Figura A.4: K_4 com arestas duplicadas, tornando-se euleriano

A partir do grafo euleriano criado, é possível encontrar o caminho euleriano P, que também é solução do PCC para o grafo K_4, e, finalmente, sequência bonita de tamanho mínimo que usa 4 valores distintos.

$$P = \{1, 2, 3, 4, 1, 3, 4, 2, 1\}$$

Sendo assim, o tamanho mínimo de uma sequência bonita que possui 4 valores distintos é 9, valor este composto da seguinte forma: o número de arestas iniciais acrescido do vértice inicial, $1 + |E(K_4)| = 1 + 6 = 7$, somado ao número de arestas duplicadas para tornar o grafo euleriano $\frac{4}{2} = 2$.

Podemos assim abstrair uma fórmula geral para o tamanho mínimo de uma sequência bonita P que utilize k valores distintos em sua composição:

$$|P| = 1 + \frac{k(k-1)}{2} + \begin{cases} 0 & k \text{ ímpar} \\ \frac{k}{2} & k \text{ par} \end{cases}$$

Todo grafo completo com uma quantidade ímpar de vértices é euleriano, já que todos seus vértices possuem grau par, enquanto os grafos completos com quantidade

par de vértices não são eulerianos, necessitando assim de $\frac{k}{2}$ duplicações de aresta. Por isso a diferença na fórmula de $|P|$.

A partir dessa fórmula fechada, é possível descobrir com uma busca binária, em complexidade $\mathcal{O}(\lg n)$, qual o maior valor de k tal que $|P| \leq n$.

Encontrado tal valor k, o número máximo de valores distintos que a solução pode possuir para ser bonita, basta descobrir quais valores da entrada escolher para maximizar a recompensa.

Se possuirmos na entrada uma quantidade de valores, m, tal que $m \leq k$, então a solução ótima será a soma das recompensas de todos valores, já que é possível encontrar uma sequência bonita que irá conter todos valores disponíveis.

Do contrário, a solução consistirá em escolher os k valores de maior recompensa disponíveis. Para encontrar tal valor, basta manter uma árvore de busca binária balanceada com as k maiores recompensas dos valores disponíveis, o que custa tempo $\mathcal{O}(m \lg k) = \mathcal{O}(m \lg \sqrt{n})$.

A.3 Jogging Trails

URL do problema: onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&page=show_problem&problem=1237

Solução: github.com/gafeol/chinese-postman/blob/master/probs/uva/1237.cpp

Neste problema devemos desenvolver um programa que calcula o custo de uma solução ótima para o problema do carteiro chinês em um grafo não direcionado de até 15 vértices com pesos nas arestas.

Este problema é uma aplicação direta do problema descrito nesse trabalho. Entretanto, em virtude do limite do número de vértices das instâncias, é possível resolver o problema de maneira mais simples.

Um dos últimos passos do algoritmo que resolve o PCC em grafos não direcionados é realizar o emparelhamento perfeito de custo mínimo usando o algoritmo de Edmonds, para determinar o supergrafo euleriano de menor custo.

Porém, como o número de vértices é muito pequeno, é possível realizar a construção do emparelhamento perfeito usando, ao invés do algoritmo de Edmonds, um algoritmo de programação dinâmica com máscara de bits ("bitmask"):

Para esta solução devemos memoizar o custo mínimo de emparelhamento perfeito para todo subconjunto de vértices do grafo condensado.

Seja K o grafo completo condensado, S um subconjunto dos vértices de K, $dist$ uma função que retorna a distância entre dois vértices quaisquer de K e min_cost_S o menor custo de emparelhamento perfeito do subconjunto S de vértices, podemos definir a seguinte recorrência:

$$min\_cost_S = \begin{cases} 0 & \text{se } S = \emptyset \\ \min_{u,v \in S} min\_cost_{S-u-v} + dist(u,v) & \text{caso contrário} \end{cases}$$

Para representar os diversos subconjuntos de vértices pode-se utilizar uma máscara de bits, isto é, um valor até $2^n - 1$, em que cada bit ativo representa um vértice presente em S.

Bibliografia

[Chr+86] N. Christofides, V. Campos, A. Corberán e E. Mota. “An algorithm for the rural postman problem on a directed graph”. Em: (1986), pp. 155–166.

[Chr76] Nicos Christofides. *Algoritmo de Christofides*. 1976. URL: https://en.wikipedia.org/wiki/Christofides_algorithm.

[E-m14] E-maxx. *Minimum-cost flow - Successive shortest path algorithm*. 2014. URL: https://cp-algorithms.com/graph/min_cost_flow.html.

[EJ73] J. Edmonds e E. L. Johnson. “Matching, Euler tours and the Chinese postman”. Em: (1973), pp. 88–124.

[Edm61] Jack Edmonds. *Algoritmo Blossom*. 1961. URL: https://en.wikipedia.org/wiki/Blossom_algorithm.

[Edm67] Jack Edmonds. *Optimum Branchings*. 1967. URL: https://nvlpubs.nist.gov/nistpubs/jres/71B/jresv71Bn4p233_A1b.pdf.

[Edm72] Richard M. Edmonds Jack e Karp. “Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems”. Em: (1972). URL: https://web.eecs.umich.edu/~pettie/matching/Edmonds-Karp-network-flow.pdf.

[EGL95] H. A. Eiselt, Michel Gendreau e Gilbert Laporte. *Arc Routing Problems, Part II: The Rural Postman Problem*. 1995. URL: https://pubsonline.informs.org/doi/10.1287/opre.43.3.399.

[Fre79] Greg N. Frederickson. “Approximation Algorithms for Some Postman Problems”. Em: *Journal of the Association for Computing Machinery* 26 (3) (1979). URL: https://dl.acm.org/doi/10.1145/322139.322150.

[Gua62] Meigu Guan. “Graphic programming using odd or even points”. Em: *Acta Mathematica Sinica* (1962), pp. 273–277.

[Gua83] Meigu Guan. “On the Windy Postman Problem”. Em: *Discrete Applied Mathematics 9* (1983), pp. 41–46.

[Hie73] Carl Hierholzer. “Über die Möglichkeit, einen Linienzug ohne Wiederholung und ohne Unterbrechung zu umfahren”. Em: (1873).

[KKG20] Anna R. Karlin, Nathan Klein e Shayan Oveis Gharan. “An Improved Approximation Algorithm for TSP in the Half Integral Case”. Em: *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*. STOC 2020. Chicago, IL, USA: Association for Computing Machinery, 2020, pp. 28–39. ISBN: 9781450369794. DOI: 10.1145/3357713.3384273. URL: https://doi.org/10.1145/3357713.3384273.

[Kol09] Vladimir Kolmogorov. *Blossom V: A new implementation of a minimum cost perfect matching algorithm*. 2009.

[LO81] Richard C. Larson e Amedeo R. Odoni. "Urban Operations Research, exercício 6.6". Em: Prentice-Hall, 1981. Cap. 6. URL: http://web.mit.edu/urban_or_book/www/book/chapter6/problems6/6.6.html.

[LK76] J. K. Lenstra e A. H. G. Rinnooy Kan. "On general routing problems". Em: *Networks* 6 (3) (1976), pp. 273–280. DOI: 10.1002/net.3230060305. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/net.3230060305. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/net.3230060305.

[Ng96] Peh H. Ng. "Designing Efficient Snow Plow Routes: A Service-Learning Project". Em: (1996). URL: http://cda.morris.umn.edu/~pehng/ng_final-CH.pdf.

[Orl95] James B. Orlin. *A polynomial time primal network simplex algorithm for minimum cost flows*. 1995. URL: https://dspace.mit.edu/handle/1721.1/2584.