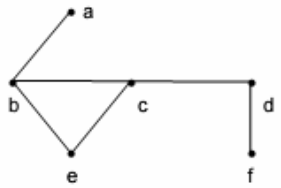
Material retirado de: GUIMARÃES, J. O. Teoria dos Grafos. UFSCar.

1 Introdução

1.1 Conceitos Básicos

Um grafo G = (V, E) é um conjunto V de vértices e um conjunto E de arestas (edges em Inglês) onde cada aresta é um par de vértices (Ex.: (v, w)). Um grafo é representado graficamente usando bolinhas para vértices e retas ou curvas para arestas. Exemplo:



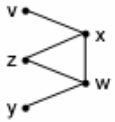
Este grafo possui V = {a, b, c, d, e, f} e E = {(a, b), (b, c), (b, e), (c, e), (c, d), (d, f)} onde (a, b) é uma aresta entre vértice a e b. Normalmente, arestas do tipo (a, a) não são permitidas.

Um grafo pode ser dirigido ou não dirigido. Em um grafo dirigido, a ordem entre os vértices de uma aresta (v, w) é importante. Esta aresta é diferente da aresta (w, v) e é representada com uma flecha de v para w:

V = {v, w, z} E = {(v, w), (v, z), (w, z)}

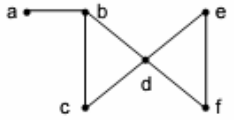
Em um grafo não dirigido, (v, w) = (w, v).

Um *caminho* (path) é uma seqüência de vértices $v_1, v_2, \ldots, v_n$ conectados por arestas (v_1, v_2), (v_2, v_3), ... (v_{n-1}, v_n). As arestas são também consideradas como parte do caminho. Ex.:



caminhos: v, x, w, y
v, x
y, w, z

Um *circuito* é um caminho onde $v_1 = v_n$, como b, c, d, e, f, d, b.



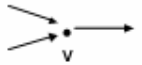
Um circuito será simples se nenhum vértice aparecer mais de uma vez, exceto o primeiro e o último. Um circuito simples é chamado de *ciclo*.

Definição: Dado um grafo, dizemos que vértice v é adjacente a vértice w (ou aresta E) se existe aresta (v, w) no grafo (ou e = (v, w)).

Definição: Um grafo é conectado se existe um caminho entre dois vértices quaisquer do grafo.

Definição: Dígrafo é um grafo dirigido.

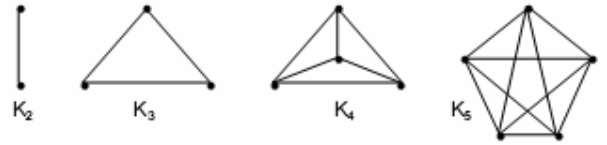
Definição: O grau de um vértice é o número de arestas adjacentes a ele. Em um grafo dirigido, o grau de entrada de um vértice v é o número de arestas (w, v) e o grau de saída é o número de arestas (v, w). Exemplo:



v possui grau de entrada 2 e grau de saída 1.

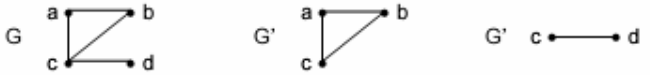
Definição: Uma fonte é um vértice com grau de entrada 0 e grau de saída ≥ 1. Um sumidouro é um vértice com grau de saída 0 e grau de entrada ≥ 1.

Definição: Um grafo é completo quando existe uma aresta entre dois vértices quaisquer do grafo. O grafo completo de n vértices é denotado por K_n. Exemplos:



(a) (b) (c) (d)

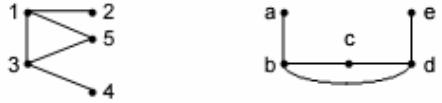
Definição: Um subgrafo G' = (V', E') de um grafo G = (V, E) é um grafo tal que $V' \subseteq V$ e $E' \subseteq E$. Exemplos:



Definição: Um grafo G = (V, E) é bipartido se V pode ser dividido em dois conjuntos V_1 e V_2 tal que toda aresta de G une um vértices de V_1 a outro de V_2. Exemplos:

Um grafo bipartido completo (GBC) possui uma aresta ligando cada vértice de V_1 a cada vértice de V_2. Se $n_1 = |V_1|$ e $n_2 = |V_2|$, o GBC é denotado por $K_{n1, n2}$.

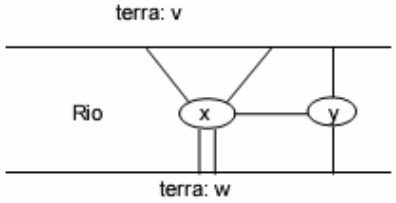
Definição: Dados dois grafos $G_1 = (V_1, E_1)$ e $G_2 = (V_2, E_2)$, dizemos que G_1 é isomorfo a G_2 se e somente se existe uma função f: $V_1 \rightarrow V_2$ tal que $(v, w) \in E_1$ se $(f(v), f(w)) \in E_2$, para todo $v, w \in V_1$. Exemplo:



f = {(1, b), (5, c), (3, d), (2, a), (4, e)}

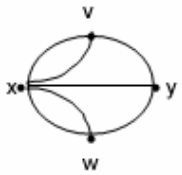
1.2 História

O primeiro problema de teoria dos grafos foi o das pontes de Könisberg. Como no desenho:



Esta cidade possuía um rio com duas ilhas conectadas por sete pontes como mostra o desenho acima. O problema é saber se é possível caminhar de um ponto qualquer da cidade e retornar a este ponto passando por cada ponte exatamente um vez.

Euler resolveu este problema criando um grafo em que terra firme é vértice e ponte é aresta:



Quando caminhamos por um vértice, nós temos que entrar e sair dele (ou vice-versa, no caso do ponto inicial), o que significa que usamos um número par de arestas cada vez que passamos por um vértice.

Como o grafo acima possui vértices com número ímpar de arestas, a resposta para o problema é NÃO.

1.3 Aplicações de Teoria dos Grafos

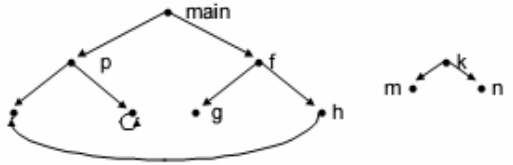
Nos itens abaixo são detalhados alguns problemas que podem ser resolvidos utilizando teoria dos grafos.

- Existem funções inúteis no programa?

Neste exemplo utilizaremos a linguagem C. Considere que funções são vértices e existe aresta de f para g se existe chamada a g no corpo de f:

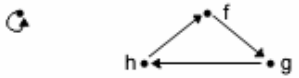
```
void f (int n)
{ if   (n > 5)
     g ( );
  …
}
```

Monta-se um grafo de todo o programa:



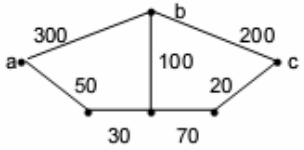
A execução do programa começa na função main que pode chamar as funções p e f. A função f pode chamar g e h. Claramente, funções k, m e n nunca serão chamadas e podem ser removidas na ligação do programa.

- Usando a mesma representação, podemos descobrir se um programa possui recursão direta ou indireta. Pode existir recursão se existe ciclo no grafo:



- Um vendedor deve passar por várias cidades e retornar ao ponto inicial. Qual o trajeto de menor distância possível ?

- Qual a menor distância entre duas cidades a e c ?



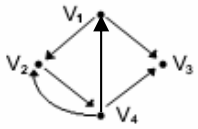
- Como ir da cidade "a" a "c" passando pelo número mínimo de cidades?

4 Estruturas de Dados para Grafos

Um grafo G = (V, E) é usualmente representado por uma matriz ou lista de adjacências.

1. Matriz de adjacência.

Sendo n o número de vértices de G, uma matriz de adjacência para G é uma matriz A = (a_{ij}) n × n tal que a_{ij} = 1 se $(v_i, v_j) \in E$.



$$A = \begin{array}{c} \\ V_1 \\ V_2 \\ V_3 \\ V_4 \end{array} \begin{array}{c} \begin{array}{cccc} V_1 & V_2 & V_3 & V_4 \end{array} \\ \begin{pmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 \end{pmatrix} \end{array}$$

A desvantagem desta representação é que ela ocupa muito espaço se há poucas arestas. Neste caso, a maior parte da matriz é inútil. A vantagem é que podemos saber se uma aresta existe ou não em tempo constante.

2. Lista de Adjacências.

Há um vetor de n posições cada uma apontando para uma lista. A posição i do vetor aponta para uma lista contendo números j tal que $(v_i, v_j) \in E$. Para o grafo anterior temos:

7 Conectividade, Caminhos e Ciclos

Definição: Um grafo não dirigido é bi-conectado se existe pelo menos dois caminhos disjuntos em vértices ligando dois vértices quaisquer do grafo.

Exemplos:

Se vértices são computadores (ou processadores) e as arestas são ligações entre eles, um computador pode falhar e ainda sim os outros serão capazes de conversar entre si.

Como caminhos disjuntos em vértices implica em caminhos disjuntos em arestas, uma ligação pode ser interrompida e ainda assim todos os computadores serão capazes de se comunicar entre si.

Grafos bi-conectado possuem um alto grau de conectividade. Situação oposta acontece com árvores. Elas são conectadas mas a remoção de qualquer vértice que não seja folha (ou mesmo uma aresta) as desconecta.

Então, se quisermos ligar vários computadores utilizando o menor número de ligações possível, mas de tal forma que todos possam conversar entre si, devemos usar uma árvore.

7.1 Ciclos

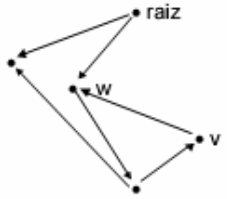
Problema: Faça um algoritmo que retorna true se um grafo não dirigido G (V, E) possui um ciclo, false caso contrário.

```
Algorithm HaCiclo (G (V, E), v): boolean
  { Faz uma busca em profundidade no grafo }
  Entrada: um grafo G e um vértice v de G

begin
marca v
Seja S o conjunto de arestas (v, w) tal que w não foi marcado
for each aresta (v, w) de S do
  if w não foi marcado
  then
    if HaCiclo (G, w)
    then
      retorne true;
    endif
  else
      { achou um ciclo - não é necessário continuar
        a busca em vértices adjacentes a v }
    return true;
  endif
return false;
```

```
end
```

O algoritmo `HaCiclo` para grafos dirigidos, `HaCicloDirig`, faz uma busca em profundidade e retorna true (há ciclo) se há aresta do vértice corrente (v) para um outro vértice w que está na pilha construída pela busca em profundidade. O ciclo encontrado é formado pelo caminho de w até v e a aresta (v, w):




```
Algorithm HaCicloDirig( G(V, E), v, S) : boolean

    Entrada: um grafo G(V, E), um vértice v de G e uma pilha S
    contendo os vértices sendo visitados.
    Saída: retorna true se houver ciclo cujos vértices são
    alcançáveis a partir de v.

begin
marque v
empilhe v na pilha S
for each aresta (v, w) ∈ V do
  if w não foi marcado
  then
    if HaCicloDirig( G, w, S)
    then
      return true;
    endif
  else
    if w está na pilha S
    then
      return true;
    endif
  endif
desempilhe v de S
return false;
end
```

Este algoritmo só funcionará se o ciclo do grafo (se existir) é alcançável a partir do vértice v. Ao chamar este algoritmo, passa-se uma pilha vazia como parâmetro S.

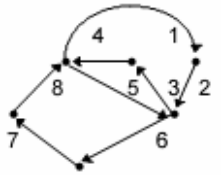
7.3 Caminhos

Definição: Um grafo G não dirigido é Eureliano se existe um caminho fechado (circuito) que inclui cada aresta de G. Este caminho é chamado de "caminho Eureliano".

Arestas em um circuito não se repetem. Assim, um caminho Eureliano contém cada aresta do grafo exatamente uma vez.

Exemplo:

Caminho Eureliano:



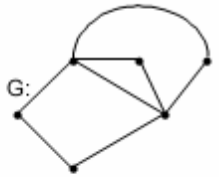
Teorema: Um grafo conectado G é Eureliano se e somente se o grau de cada vértice de G é par.

A prova $\Rightarrow$ é feita por contradição: admita que um vértice possui grau ímpar. O caminho Eureliano passará por ele um certo número de vezes e, a cada vez, utilizará duas

arestas (uma para entrar e outra para sair). Então sobrará uma única aresta que não poderá ser utilizada no caminho: contradição, o que prova que a hipótese $\Rightarrow$ está correta. A prova $\Leftarrow$ é dada pelo algoritmo para encontrar um caminho Eureliano, que é dado a seguir.

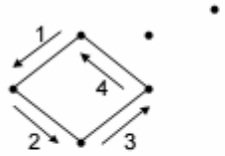
A partir de um vértice v qualquer, comece a percorrer o grafo sem passar por nenhuma aresta duas vezes. Como o grau de cada vértice é par, podemos sempre entrar em um novo vértice por uma aresta e sair por outra:

Como o número de vértices de G é finito, em algum momento retornaremos a v. Como nenhuma aresta foi usada duas vezes, temos um circuito P. Note que este circuito pode não conter todos os vértices do grafo:



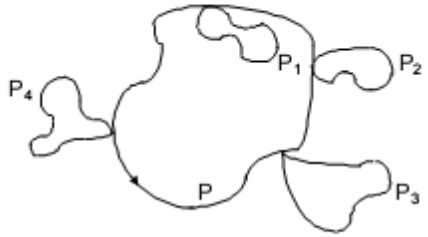
Circuito:

P

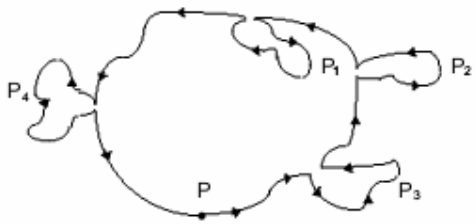


Agora construímos G' retirando de G as arestas do circuito encontrado acima. O grau de cada vértice em G' é par pois o número de arestas removidas de cada vértice é par. G' pode não ser conectado. Sejam $G'_1, G'_2, \ldots G'_k$ os componentes conexos de G'. Cada grafo G'_i é conectado e o grau de cada vértice é par (pois o grau de cada vértice de G' é par). Sejam $P_1, P_2, \ldots P_k$ os circuitos Eurelianos encontrados pela aplicação recursiva deste algoritmo a $G'_1, \ldots G'_k$.

Para encontrar um Caminho Eureliano para G, começamos a percorrer o circuito P a partir de um vértice qualquer. Quando encontramos um vértice v que pertence a um caminho P_i, percorremos P_i, retornamos a v e continuamos a percorrer P novamente. Deste modo, quanto chegarmos ao vértice inicial de P, teremos percorrido P, $P_1, P_2, \ldots P_k$. Isto é, teremos um circuito Eureliano. Veja a ilustração abaixo.



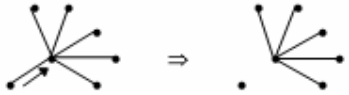
O desenho acima mostra o circuito P (grande) e os circuitos P_1, P_2, P_3 e P_4. obtidos após a remoção de P do grafo. O desenho abaixo mostra como todos estes circuitos podem ser conectados para formar um caminho Eureliano para o grafo.



Definição: Um circuito Hamiltoniano em um grafo conectado G é um circuito que inclui cada vértice de G exatamente uma vez.

Encontrar um circuito Hamiltoniano (CH) é claramente mais difícil que encontrar um caminho Eureliano (CE), pois cada vez que atingimos (num percurso) um vértice v a partir de uma aresta "e", nunca mais podemos usar v e quaisquer de suas arestas. Em um CE, não poderíamos usar "e", mas poderíamos usar v e suas outras arestas.

CH



CE

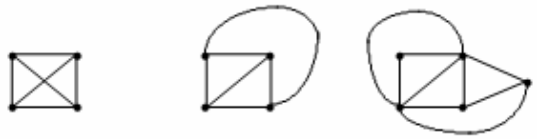
O problema "Encontre um CH para um grafo G" é NP-completo (Veja Seção 16) e portanto nenhum algoritmo será estudado.

O Problema do Caixeiro Viajante (The Traveling Salesman Problem)

Um vendedor quer visitar um grupo de cidades começando e terminando em uma mesma cidade de tal forma que o custo da viagem (distância percorrida) seja o menor possível e que cada cidade seja visitada apenas uma vez. Ou seja, o problema é encontrar um circuito Hamiltoniano de custo mínimo. Naturalmente, este problema é NP-completo.

9 Planaridade

Um grafo planar é aquele que pode ser desenhado no plano de tal forma que duas arestas quaisquer não se interceptam. Exemplo:



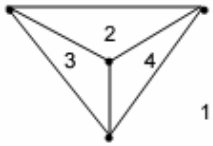
(a) (b) (c)

Observe que, apesar de duas arestas de (a) cruzarem, este grafo é planar porque ele pode ser transformado no desenho (b).

Teorema de Jordan: Dada uma curva fechada ψ no plano e dois pontos, um interior e outro exterior a ela, qualquer curva ligando os dois pontos intercepta ψ.

Teorema: k_5 e $k_{3,3}$ não são planares.

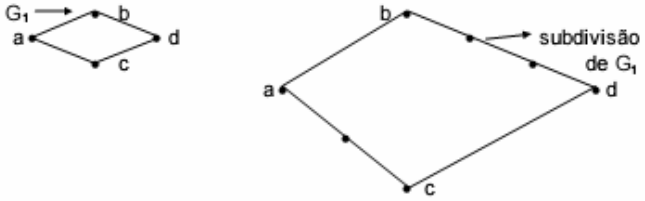
Prova: Provaremos apenas que k_5 não é planar. Usando a representação



o plano fica dividido em quatro regiões. Em qualquer região que coloquemos o quinto vértice, uma aresta que o liga a algum outro vértice cruzará outra aresta (pelo teorema de Jordan).

Definição: A subdivisão de uma aresta é uma operação que transforma a aresta (v, w) em um caminho v, $z_1, z_2, \ldots z_k$ w sendo $k \geq 0$, onde os z_i são vértices de grau 2 adicionados ao grafo.

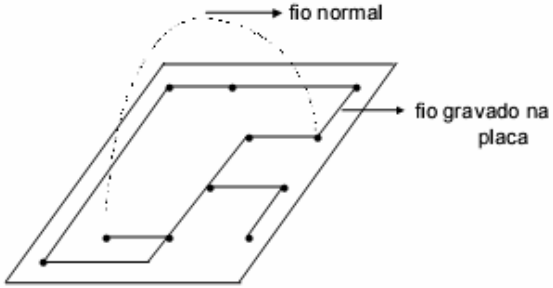
Um grafo G_2 será uma subdivisão do grafo G_1 quando G_2 puder ser obtido de G_1 através de uma seqüência de arestas de G_1. Exemplo:



Teorema de Kuratowski: Um grafo é planar se e somente se ele não contém nenhum subgrafo que é uma subdivisão de K_5 ou $K_{3,3}$.

A prova deste teorema está além do alcance deste curso.

Um circuito eletrônico pode ser considerado um grafo onde as junções são vértices e as arestas são os fios ligando as junções. Se o grafo correspondente ao circuito é planar, todos os fios podem ser gravados na própria placa. Se o grafo não é planar por causa de apenas uma aresta, esta é um fio normal que deve passar por cima da placa. Isto equivale a colocar esta aresta acima do plano contendo o restante do grafo:

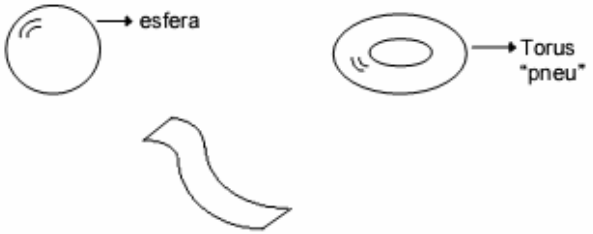


Teorema: Todo grafo planar admite uma representação plana em que todas as linhas são retas.

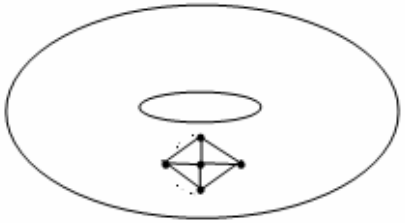
Definição: Um grafo pode ser embebido em uma superfície S se ele pode ser colocado em S de tal forma que quaisquer duas de suas arestas não se cruzam.

Teorema: Para cada superfície S, existe um grafo que não pode ser embebido em S.

Definição: Uma superfície é uma curva descrita por 2 dimensões, não necessariamente no plano. Ex.:



Nota: K_5 pode ser embebido no Torus:



$K_{3,3}$ pode ser embebido na fita de Möbius

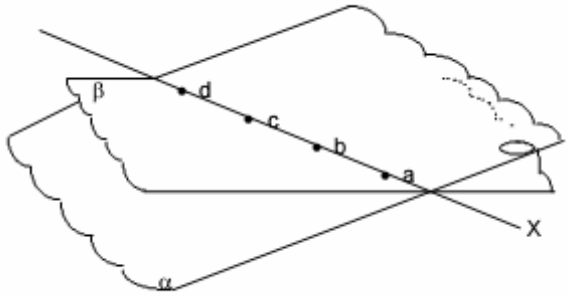
Teorema: Um grafo pode ser embebido na esfera S se ele pode ser embebido no plano.

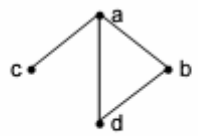
Existe um algoritmo O (n) para determinar se um grafo é planar ou não, feito por Hopcroft e Tarjan.

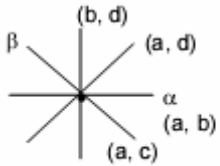
Teorema: Qualquer grafo pode ser colocado em um espaço de três dimensões.

Prova: Coloque os vértices do grafo em uma reta X. Então, para cada aresta, faça um plano que contém X. Arestas distintas devem corresponder a planos distintos.

Para cada aresta desenharemos um semicírculo ligando os dois vértices. As arestas não se interceptarão porque elas estarão em planos diferentes.



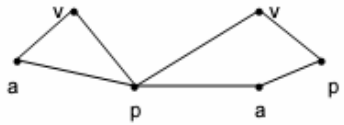




15 Coloração

Seja G (V, E) um grafo e C = $\{C_i \mid 1 \leq i \leq n\}$ um conjunto de cores. Uma coloração de G é a atribuição de cores de C para todos os vértices de G de tal forma que vértices adjacentes tenham cores diferentes. Ex.:

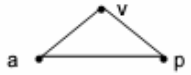
C = {v (ermelho), a (zul), p (reto)}



Uma K-coloração é uma coloração que utiliza K cores.

Definição: O número cromático de um grafo G, indicado por X (G), é o menor número de cores K para o qual existe uma K - coloração de G.

No grafo acima, X (G) = 3, pois o "triângulo"

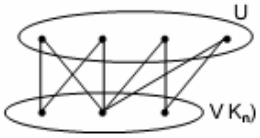


impede que X (G) seja 2.

Um grafo completo de n vértices, também conhecido por K_n necessita de n cores, já que cada vértice está ligado a todos os outros:

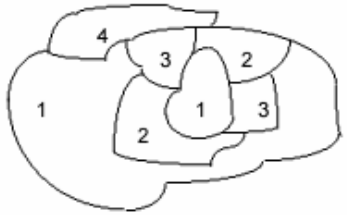
Então, X (K_n) = n. Obviamente, se um grafo G possuir K_n como subgrafo, então X (G) ≥ X (K_n) ou X (G) ≥ n.

Um grafo bipartido pode ser dividido em dois conjuntos U e V tal que não existam arestas dentro de cada conjunto:



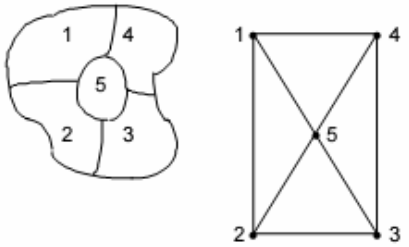
Então, todo grafo bipartido pode ser colorido com apenas duas cores.

O Problema das Quatro Cores: Dado um mapa qualquer (no plano), podemos colori-lo com apenas quatro cores? Por colorir queremos dizer que regiões adjacentes são coloridas com cores diferentes:



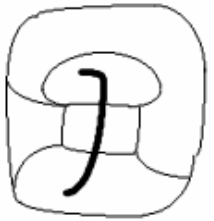
De fato, quatro cores podem colorir qualquer mapa no plano, o que é conhecido por cartógrafos a séculos. Contudo, este teorema só foi provado em 1976 usando teoria dos grafos e um computador.

Este problema pode ser transformado em um problema de grafos associando-se cada região a um vértice. Existe uma aresta entre dois vértices se as duas regiões correspondentes forem adjacentes (fazem fronteira) no mapa. Por exemplo, o mapa da esquerda é transformado no grafo da direita.

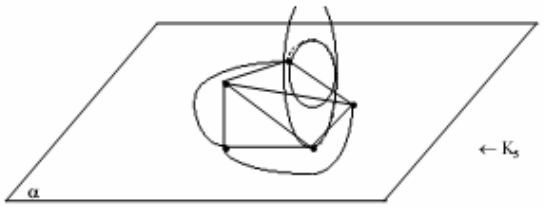


O problema agora é provar que qualquer grafo construído desta forma pode ser colorido com no máximo quatro cores.

Todo grafo obtido de um mapa é planar. Se não fosse, haveria fronteira entre duas regiões que não fazem fronteira no plano — seria uma fronteira no espaço. Veja figura abaixo, onde a ligação preta indica uma ponte acima do papel.



Lembre-se de que um grafo não planar pode ser colocado em um plano desde que algumas arestas liguem alguns vértices no espaço:



a aresta pontilhada é uma aresta no espaço.

13 Compressão de Dados — Algoritmo de Huffman

Problema: Temos um texto usando caracteres ASCII e queremos comprimi-lo para uma seqüência menor de caracteres.

A técnica que utilizaremos é representar cada caráter por uma seqüência de bits. Então tomaremos o texto original substituindo cada caráter pela seqüência de bits correspondente.

Como existe 128 caracteres ASCII, precisamos de 7 bits para cada caráter se representarmos todos eles usando o mesmo número de bits.

O algoritmo de Huffman representa os caracteres que aparecem mais no texto por uma seqüência menor e os que aparecem mais por uma seqüência maior. Assim ele comprime o texto. Ex.:

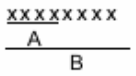
Comprimir DABAABCAA assumindo que só usamos caracteres A, B, C e D.
A → 1
B → 00
C → 010
D → 011

DABAABCAA → 011100110001000 → 15 bits

Se representássemos cada caráter com 2 bits (A = 00, B = 01, C = 10, D = 11) usaríamos 9 × 2 = 18 bits.

Poderia haver ambigüidades na representação. Por exemplo, se A = 1 e B = 10, não saberíamos se 1010 é AC ou BB.

Em geral, o código de um caráter não pode ser prefixo de outro.



Esta restrição implica que colocar menos bits para alguns caracteres significa mais bits para outros.

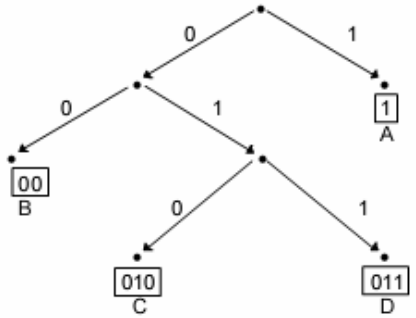
A compressão de dados utilizando esta técnica implica em encontrar uma relação caracteres/bits que satisfaça a restrição de prefixos e que minimize o número total de bits para codificar o texto.

Para saber que caracteres aparecem mais ou menos no texto, calculamos inicialmente a freqüência de cada caráter. Assuma que, em um texto F, os caracteres c_1, c_2, ... c_n aparecem com freqüências f_1, f_2, ... f_n. Uma codificação E associa cada c_i a uma *string* S_i de bits cujo tamanho é s_i. O nosso objetivo é encontrar codificação E que satisfaça a restrição de prefixo e minimize

$$L(E, F) = \sum_{i=1}^{n} s_i \cdot f_i$$

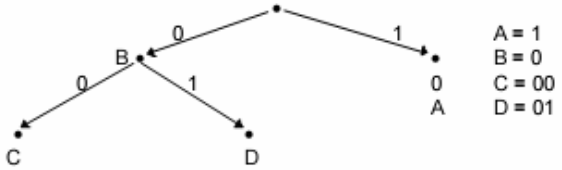
que é o tamanho do texto comprimido. No exemplo anterior, E = { (A, 1), (B, 00), (C, 010), (D, 011) } e $s_1 = 1$, $s_2 = 2$, $s_3 = 3$, $s_4 = 3$ e L (E, F) = 5 . 1 + 2 . 2 + 1 . 3 + 1 . 3 = 15

Considere uma árvore binária dirigida em que cada vértice possui grau de saída 0 ou 2 e as arestas da esquerda e direita estão associados os números 0 e 1, respectivamente:



Associamos as folhas aos caracteres e a seqüência de bits da raiz até a folha como a codificação do caráter. Para decodificar um texto codificado, percorremos a árvore até encontrar o 1.º caráter. Depois fazemos outra busca para o 2.º caráter e assim por diante. Observe que, como os caracteres são folhas, a restrição de prefixo é obedecida.

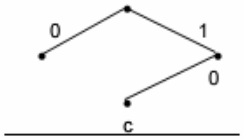
Uma seqüência de bits só poderia ser prefixo de outra se o caráter correspondente estivesse no meio da árvore:



Agora temos que construir uma árvore que minimize L (E, F). Usamos indução finita para reduzir o problema de n para n - 1 caracteres. O caso base é n = 2 e é trivial.

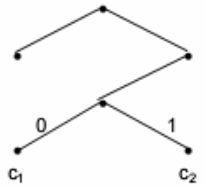
HI: Sabemos como construir a árvore descrita acima para n - 1 caracteres.

Para resolver o problema, tomaremos n caracteres e combinaremos 2 deles em um nó, resultado em n - 1 caracteres. Aplicamos a HI para n - 1 obtendo uma árvore na qual os dois caracteres combinados são introduzidos:



obtido p / n - 1
c = caracteres combinados

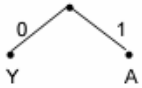
reintroduzindo os 2 caracteres



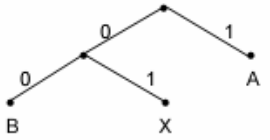
Os caracteres a serem combinados (c_1 e c_2) são os nós com menor freqüência. Nós podemos assumir que c_1 e c_2 são filhos de um mesmo vértice porque cada vértice possui um ou dois filhos.

Obs.: Não provamos que a árvore obtida desta forma minimiza L (E,F).

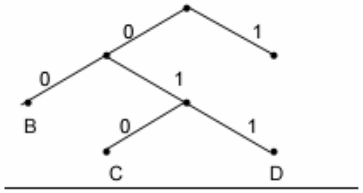
Como exemplo, codificaremos DABAABCAA, onde as seqüências são A = 5, B = 2, C = 1, D = 1.
Combinando C e D, obtemos X com freqüência 2 {A = 5 / B = 2 / X = 2 }. Combinando B e X, obtemos Y com freqüência 4 {A = 5 / Y = 4 }. Este é o caso base, que resulta na árvore



Desdobrando Y, temos



Desdobrando X, temos



Então, A = 1, B = 00, C = 010, D = 011

DABAABCAA = 011100110001011

Observe que este algoritmo não é um algoritmo de grafos, embora o uso de grafos facilite a sua compreensão.