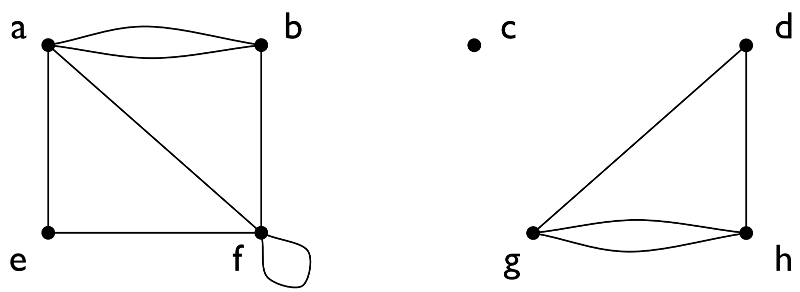
Teoria dos Grafos
Unidade 4: Conexidade

Prof. Dr. Paulo César Rodacki Gomes
paulo.rodacki@blumenau.ifc.edu.br

INSTITUTO FEDERAL
CATARINENSE
Câmpus Blumenau

Introdução

- quantos grafos há na figura abaixo?



Definições

- Um grafo não dirigido é "conexo" se para cada par de vértices u e v ∈ V, existe caminho entre u e v;
- caso contrário, o grafo é "não-conexo";
- Cada subgrafo conexo de um grafo não conexo é chamado de "componente conexa" do grafo.

Conexidade

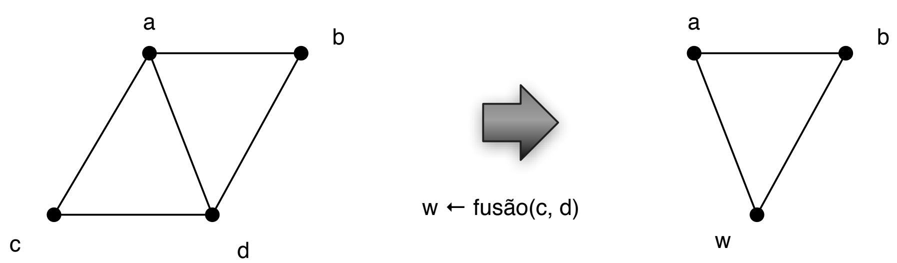
- Algoritmos para avaliação de conexidade em grafos não dirigidos:
  - busca em largura
  - busca em profundidade
  - algoritmo de Goodman (Rabuske, 1992)
  - estruturas de conjuntos (Cormen,

Algoritmo de Goodman

- o algoritmo realiza uma redução seqüencial do grafo, por meio de fusão de vértices, até que cada componente conexa seja reduzida a um único vértice
- na fusão de dois vértices adjacentes u e v, a aresta (u,v) é eliminada, é criado um novo vértice w, adjacente a todos os vértices adjacentes a u e v antes da fusão. Os vértices u e v são removidos

Algoritmo de Goodman

- Fusão de vértices:



Algoritmo de Goodman

P1) H←G; C←0;

P2) enquanto H ≠∅, faça

selecione um vértice w ∈ H;

enquanto w for adjacente a algum vértice u ∈ H, faça w ← fusão (w, u);

remova w, isto é, faça H←H-w; e C←C+1;

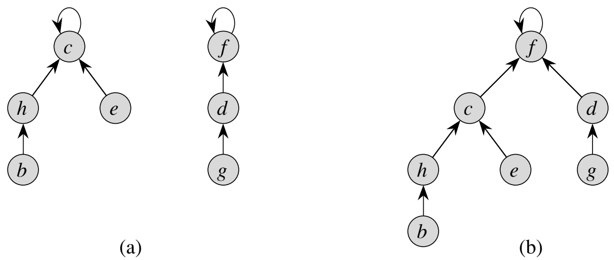
P3) se C > 1, então G é não-conexo.

Estrutura de conjuntos disjuntos

- $MAKE-SET(v)$: cria um conjunto contendo o vértice v, e atribui um identificador único a este conjunto;
- $FIND-SET(v)$: retorna o identificador do conjunto que contém o vértice v;
- $UNION(u,v)$: une os conjuntos que contém os vértices u e v, e atribui um identificador ao conjunto resultante.

Estrutura de dados para conjuntos disjuntos

Operações de criação, consulta e união devem ser eficientes (MAKE-SET, FIND-SET, UNION)

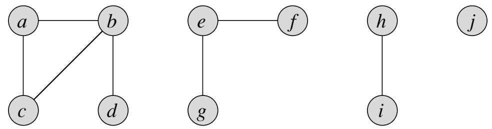


2 conjuntos UNION(h,g)


```
MAKE-SET(x)
   p[x] ← x
   rank[x] ← 0

UNION(x, y)
   LINK(FIND-SET(x), FIND-SET(y))

LINK(x, y)
   if rank[x] > rank[y]
     then p[y] ← x
     else p[x] ← y
          if rank[x] = rank[y]
            then rank[y] ← rank[y] + 1

FIND-SET(x)
   if x ≠ p[x]
     then p[x] ← FIND-SET(p[x])
   return p[x]
```



```
CONNECTED-COMPONENTS(G)
   for each vertex v ∈ V[G]
        do MAKE-SET(v)
   for each edge (u, v) ∈ E[G]
        do if FIND-SET(u) ≠ FIND-SET(v)
              then UNION(u, v)

SAME-COMPONENT(u, v)
   if FIND-SET(u) = FIND-SET(v)
     then return TRUE
     else return FALSE
```

(a)


```
CONNECTED-COMPONENTS(G)
   for each vertex v ∈ V[G]
        do MAKE-SET(v)
   for each edge (u, v) ∈ E[G]
        do if FIND-SET(u) ≠ FIND-SET(v)
              then UNION(u, v)

SAME-COMPONENT(u, v)
   if FIND-SET(u) = FIND-SET(v)
     then return TRUE
     else return FALSE
```


Edge processed	Collection of disjoint sets									
initial sets	{a}	{b}	{c}	{d}	{e}	{f}	{g}	{h}	{i}	{j}
(b,d)	{a}	{b,d}	{c}		{e}	{f}	{g}	{h}	{i}	{j}
(e,g)	{a}	{b,d}	{c}		{e,g}	{f}		{h}	{i}	{j}
(a,c)	{a,c}	{b,d}			{e,g}	{f}		{h}	{i}	{j}
(h,i)	{a,c}	{b,d}			{e,g}	{f}		{h,i}		{j}
(a,b)	{a,b,c,d}				{e,g}	{f}		{h,i}		{j}
(e,f)	{a,b,c,d}				{e,f,g}			{h,i}		{j}
(b,c)	{a,b,c,d}				{e,f,g}			{h,i}		{j}

(b)

Problema - Stammtich

Em junho, ocorreu em Rio do Sul o Stammtisch. Originalmente, o Stammtisch foi criado com o intuito de confraternização entre grupos de amigos, e esta característica se mantém até hoje. Nas barracas montadas ao longo da rua Aristiliano Ramos, os grupos expõem algum atrativo para chamar a atenção dos visitantes. Estes são dos mais variados, como jogos de cartas, comidas típicas alemãs, bebidas, etc. Entre as bebidas, certamente a mais cobiçada é o chopp.

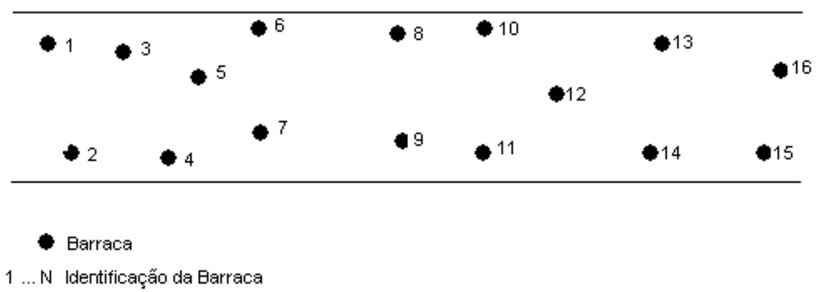
Dentre os visitantes, encontram-se também os tipos mais variados possíveis. Há aquele que vai apenas para uma visitinha, experimentar as comidas típicas alemãs, beber. Mas o que nos interessa neste momento, é um visitante especial. Seu nome é Hans Helmont Volksweigainkrollshmit, natural de Ituporanga, já com uma certa experiência de vida, e que não dispensa em hipótese alguma o chopp. Em todos os encontros ou festas onde existe chopp, Hans marca presença. Munido de sua caneca de 500*ml*, Hans se fez presente em todos os encontros Stammtisch.

Hans é um bebedor nato, e sua capacidade de beber chopp pode ser mensurada em chopp *por metro percorrido*. Hans, nos encontros de Stammtisch, estabelece a marca de 100*ml* de chopp a cada 5 metros percorridos, ou seja, seu copo de 500*m*l cheio, dura, exatamente 25 metros. Conseqüentemente, se a distância entre grupos que distribuem chopp for superior a 25 metros, Hans ficará sem beber.

Problema - Stammtich

Com o objetivo de agradar os visitantes do tipo de Hans (não os deixando um instante sem beber), a organização do evento decidiu dispor os grupos de forma estratégica ao longo da rua Aristiliano Ramos, ou seja, a distância entre as barracas dos grupos que distribuirão cerveja, não pode ser superior a 25 metros. A sua tarefa, é, dado uma distribuição de barracas ao longo da rua, determinar quantas vezes Hans ficará sem beber.

Fig. 1: exemplo da disposição das barracas ao longo da rua Aristiliano Ramos



Problema - Stammtich

A entrada de dados será composta por:

- Um inteiro A representando o ano de realização do evento.
- Um inteiro Q que representará a quantidade de grupos participantes, sendo
- As próximas linhas, contém as distâncias entre as barracas, e são apresentadas da seguinte forma: **O D M**, sendo: **O** e **D** barracas e **M** a distância entre elas.

Exemplo:

```
1990      2 5 40
6         2 6 60
1 2 20    3 4 30
1 3 10    3 5 40
1 4 30    3 6 50
1 5 40    4 5 10
1 6 50    4 6 20
2 3 10    5 6 15
2 4 40    0
```


Problema I
Ir e Vir

Nome do arquivo fonte: `ir.c`, `ir.cpp` *ou* `ir.java`

Numa certa cidade há N intersecções ligadas por ruas de mão única e ruas com mão dupla de direção. É uma cidade moderna, de forma que muitas ruas atravessam túneis ou têm viadutos. Evidentemente é necessário que se possa viajar entre quaisquer duas intersecções, isto é, dadas duas intersecções V e W, deve ser posssível viajar de V para W e de W para V.

Sua tarefa é escrever um programa que leia a descrição do sistema de tráfego de uma cidade e determine se o requisito de conexidade é satisfeito ou não.

Entrada

A entrada contém vários casos de teste. A primeira linha de um caso de teste contém dois números inteiros N e M, separados por um espaço em branco, indicando respectivamente o

Saída

Para cada caso de teste seu programa deve imprimir uma linha contendo um inteiro G, onde G é igual a um se o requisito de conexidade está satisfeito, ou G é igual a zero, caso contrário.

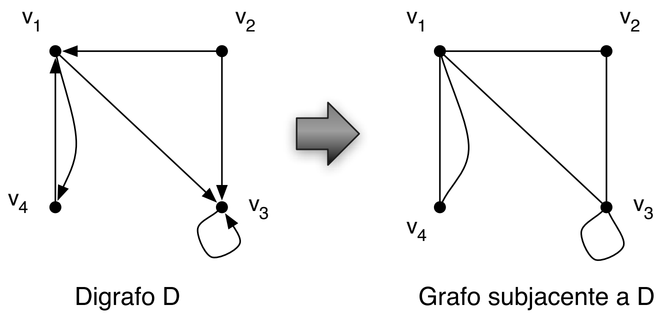
dupla, liga V e W. Não existe duas ruas ligando as mesmas intersecções.

O último caso de teste é seguido por uma linha que contém apenas dois números zero separados por um espaço em branco.

Digrafos - Conexidade

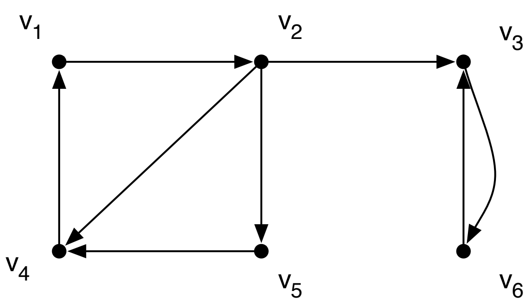
Definição 3.3 (Grafo subjacente). *Dado um digrafo D, seu grafo subjacente é um grafo não dirigido obtido pela substituição de todas as arestas dirigidas de D por arestas não dirigidas.*

Definição 3.4 (Digrafo conexo). *Um digrafo D é conexo se seu grafo subjacente for conexo.*



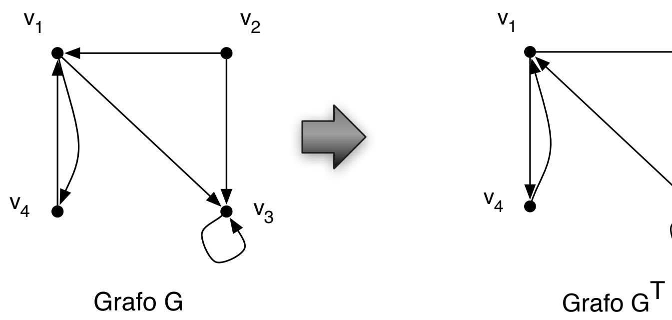
Componentes fortemente conexas

Definição 3.5 (Componente fortemente conexa). *Uma componente fortemente conexa de um digrafo G=(V,E) é é um conjunto maximal de vértices $C \subseteq V$ tal que para cada par de vértices u e v em C, existe caminho do vértice u para o vértice v e vice-versa (de v para u).*



Componentes fortemente conexas

Definição 3.6 (Grafo transposto). *Dado um grafo dirigido G=(V,E), seu grafo transposto é definigo por $G^T = (V, E^T)$, onde $E^T = \{(u,v) : (v,u) \in E\}$.*



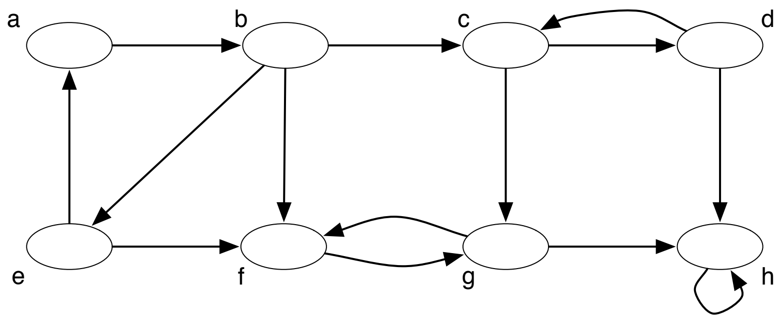
Algoritmo

STRONGLY-CONNECTED-COMPONENTS(G)

1 call DFS(G) to compute finishing times $f[u]$ for each vertex u
2 compute G^{T}
3 call DFS(G^{T}), but in the main loop of DFS, consider the vertices in order of decreasing $f[u]$ (as computed in line 1)
4 output the vertices of each tree in the depth-first forest formed in line 3 as a separate strongly connected component

STRONGLY-CONNECTED-COMPONENTS(G)

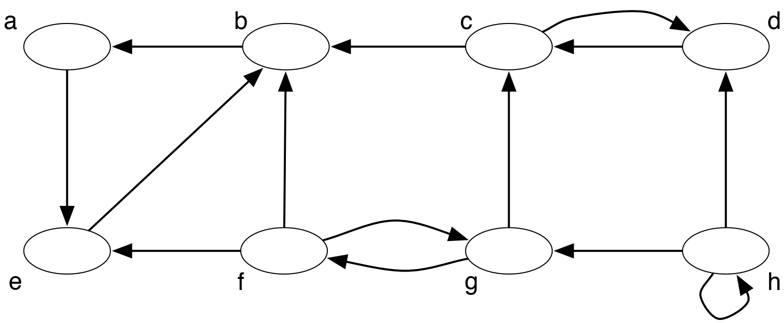
1 call DFS(G) to compute finishing times $f[u]$ for each vertex u
2 compute G^{T}
3 call DFS(G^{T}), but in the main loop of DFS, consider the vertices
in order of decreasing $f[u]$ (as computed in line 1)
4 output the vertices of each tree in the depth-first forest formed in line 3 as a
separate strongly connected component



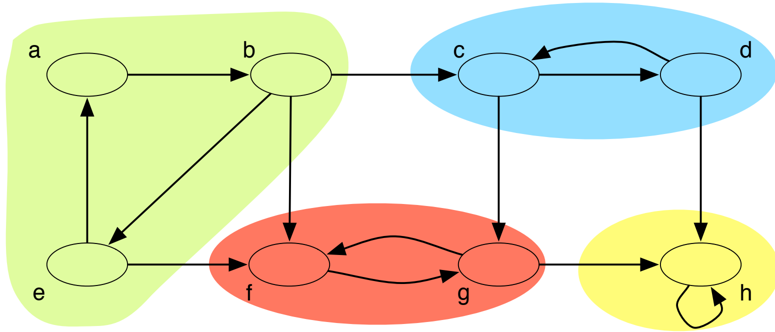
Grafo G

STRONGLY-CONNECTED-COMPONENTS(G)

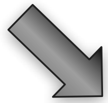
1 call DFS(G) to compute finishing times $f[u]$ for each vertex u
2 compute G^{T}
3 call DFS(G^{T}), but in the main loop of DFS, consider the vertices
in order of decreasing $f[u]$ (as computed in line 1)
4 output the vertices of each tree in the depth-first forest formed in line 3 as a
separate strongly connected component

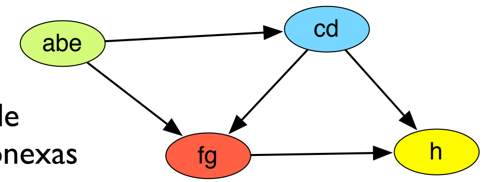


Grafo G^{T}



Grafo G





Grafo Acíclico Dirigido de
Componentes fortemente conexas

Blibliografia

- Márcia A. Rabuske. **Introdução à Teoria dos Grafos**. Editora da UFSC. 1992.
- Joan M. Aldous, Robin J. Wilson. **Graphs and Applications**: as introductory approach. Springer. 2001
- Thomas Cormen et al. **Algoritmos**: teoria e prática. Ed. Campus. 2004.