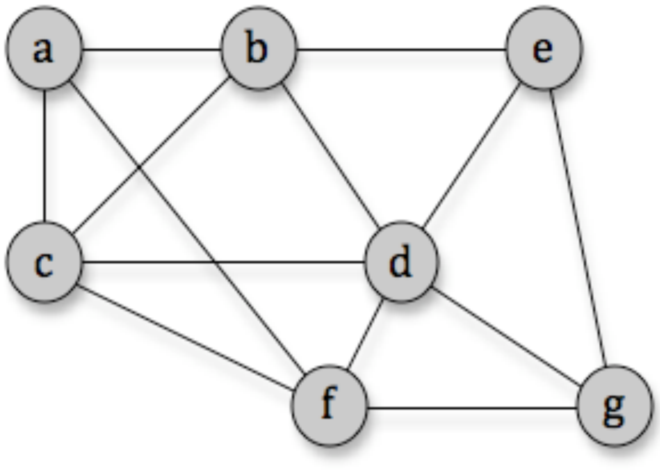
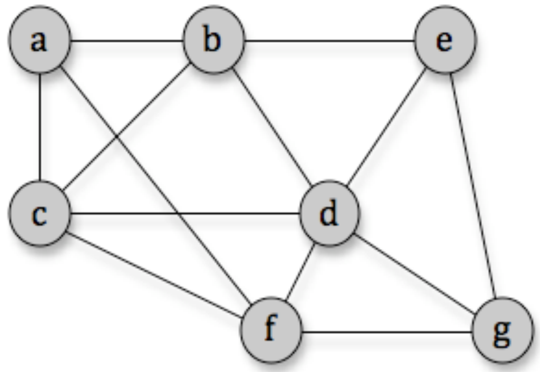
Árvores de espalhamento (*spanning trees*)

- Considere que os nós do grafo abaixo representam bairros de uma cidade e as arestas representam ruas que o prefeito cogita em asfaltar. Imagine que, por razões econômicas, devem ser asfaltadas o menor número possível de ruas, desde que seja possível ir de um bairro a outro, percorrendo somente ruas asfaltadas. Quais ruas devem ser asfaltadas?

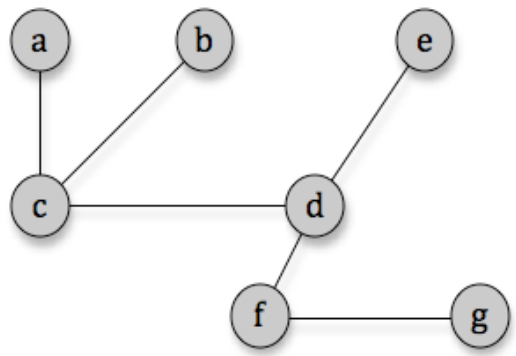


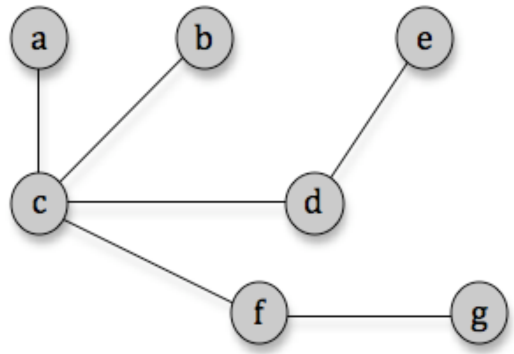
- A solução do problema é uma árvore de espalhamento.

Árvores de espalhamento (*spanning trees*)



- Duas soluções (com 6 arestas):





- Note que se removermos mais uma aresta em qualquer dessas duas soluções, pelo menos um vértice vai ficar isolado. Note também que se incluirmos mais uma aresta, será criado um ciclo.

Árvores de espalhamento (*spanning trees*)

- No caso de grafos ponderados, o problema é encontrar uma **árvore de espalhamento mínima** (AEM), ou seja, uma árvore de espalhamento na qual a soma dos pesos das arestas é mínima (o problema anterior é um caso particular, em que todas as arestas têm o mesmo peso).
- Os algoritmos já propostos para encontrar a AEM podem ser divididos nas seguintes categorias:
  - Criar e expandir muitas árvores ao mesmo tempo, que serão fundidas em árvores maiores (Boruvka);
  - Expandir várias árvores para formar uma árvore de espalhamento (Kruskal);
  - Expandir só uma árvore e adicionar novos ramos (Prim);
  - Expandir só uma árvore, adicionar novos ramos e possivelmente remover ramos (Dijkstra).

Algoritmo de Boruvka

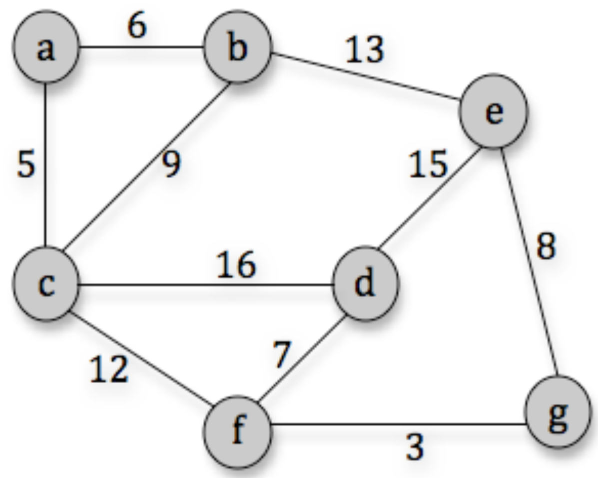
- Começa com |V| árvores de 1 vértice. Para cada árvore v, procura uma aresta (vw) de peso mínimo e inclui esta aresta para formar árvores de 2 vértices. Em seguida, procura por arestas de peso mínimo para conectar as árvores já existentes em árvores maiores. O processo termina quando uma árvore de espalhamento é criada.

```
AEM_Boruvka(digrafo G)
{
  tornar cada v ∈ V a raiz de uma árvore de 1 nó;
  while (número de árvores > 1)
  {
    for (cada árvore t)
    {
      a = aresta de peso mínimo (vu), onde v ∈ t e u ∉ t;
      criar uma árvore combinando t e a árvore que contém u;
    }
  }
}
```

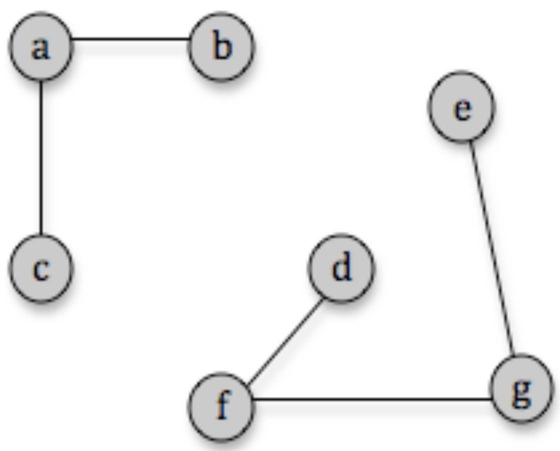
A cada iteração, cada uma das **k** árvores existentes é unida a pelo menos uma árvore. No pior caso, k/2 árvores serão geradas. Na iteração seguinte, k/4 árvores, e assim sucessivamente. Portanto, no pior caso, são necessárias **log(|V|)** iterações para construir a AEM.

Algoritmo de Boruvka

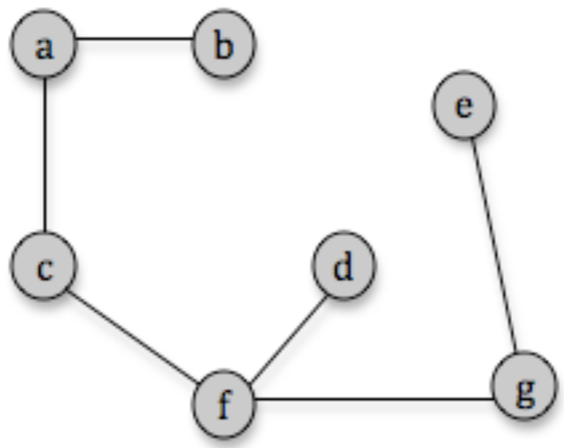
Exemplo:



Iteração 1



Iteração 2



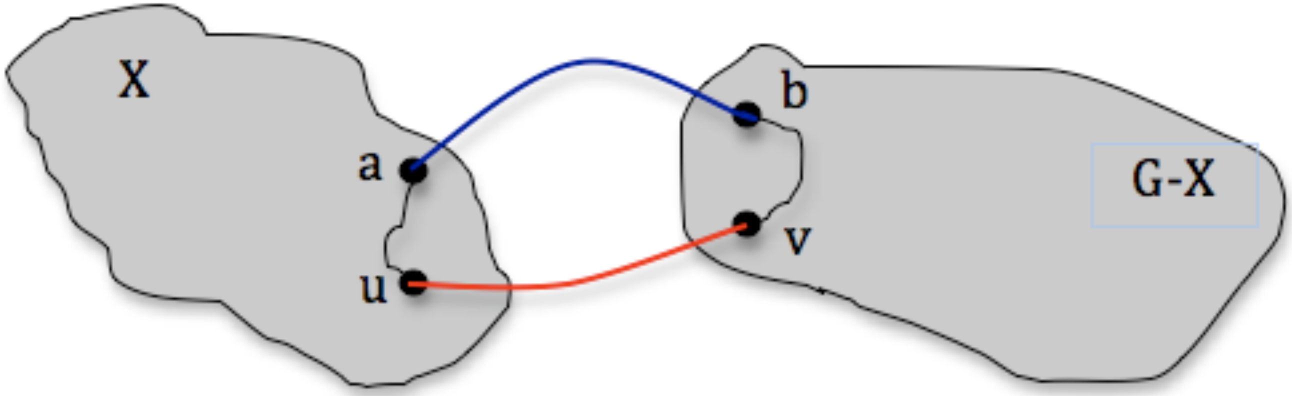
Algoritmo de Boruvka

- Por que, no algoritmo de Boruvka, escolhendo-se a cada passo uma aresta de peso mínimo garante-se obter uma árvore de espalhamento mínima?
- **Teorema**: Seja X um subconjunto de vértices de G, e seja ***x*** a aresta de peso mínimo conectando X a G-X. Então ***x*** é parte da árvore de espalhamento mínima de G.
- **Prova**: Vamos supor que existe uma árvore de espalhamento T que não contém ***x***. Vamos mostrar que T não é uma árvore de espalhamento mínima.

  Seja ***x*** = (*u*, *v*), com $u \in$ X e $v \notin$ X. Então, como T é uma árvore de espalhamento, T contém um único caminho de *u* a *v*, que juntamente com ***x*** forma um ciclo em G.

  Este caminho precisa incluir uma outra aresta ***y*** conectanto X a G-X.

Algoritmo de Boruvka



y = (a, b)

Então, T+***x***-***y*** é uma outra árvore de espalhamento (note que essa árvore tem o mesmo número de arestas que T e continua conectada pois pode-se trocar qualquer caminho contendo ***y*** pelo outro que vai no sentido contrário do ciclo). Portanto, T+***x***-***y*** tem peso menor que T, pois ***x*** tem peso menor do que ***y***. Logo, T não é mínima.

Algoritmo de Kruskal

- No algoritmo de Kruskal, as arestas são ordenadas pelo peso. Em seguida, cada uma das arestas desta sequência ordenada é verificada se pode ser incluída na árvore em construção, ou seja, se após sua inclusão nenhum ciclo é formado.

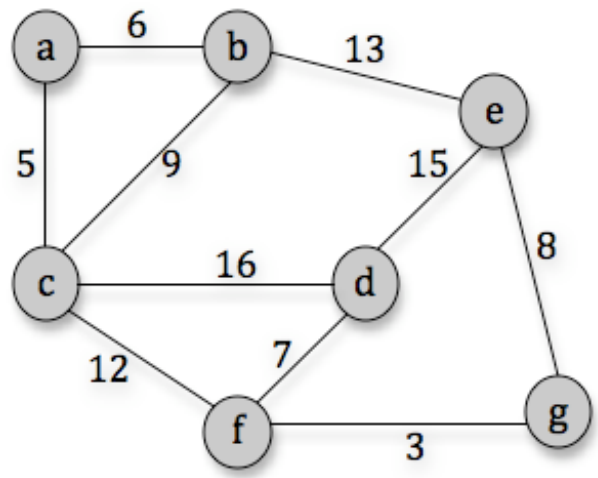
```
AEM_Kruskal(digrafo G)
{
  arvore = ∅;
  aresta = sequência de arestas de A ordenadas pelo peso;
  for (i = 0; i < |A| && |arvore| < |V|-1; i++)
  {
    if (aresta[i] não forma um ciclo em arvore)
      arvore = arvore ∪ aresta[i];
  }
}
```

A complexidade do algoritmo de Kruskal é **O(|A| log|V|)**.

Ver: DROZDEK, A. *Estruturas de Dados e Algoritmos em C++*, São Paulo: Thomson, 2002.

Algoritmo de Kruskal

Exemplo:



arestas ordenadas:

(fg), (ac), (ab), (df), (eg), (cb), (cf), (be), (de), (cd)

- Portanto, a árvore de espalhamento mínima será construída incluindo-se as arestas:

  (fg), (ac), (ab), (df), (eg) e (cf)

Algoritmo de Prim

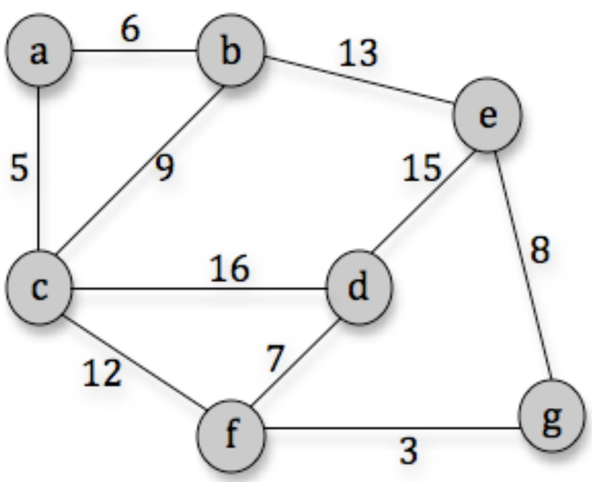
- No algoritmo de Prim, todas as arestas também são ordenadas pelo peso, inicialmente. A aresta candidata a ser incluída na árvore é a aresta desta sequência ordenada que não acarreta um ciclo e que é incidente a um vértice já presente na árvore.

```
AEM_Prim(digrafo G)
{
  arvore = ∅;
  aresta = sequência de arestas de A ordenadas pelo peso;
  for (i = 0; i < |V|-1; i++)
    for (j = 0; j < |A|; j++)
      if ((aresta[j] não forma um ciclo em arvore) &&
          (aresta[j] é incidente a um vértice de arvore))
      {
        arvore = arvore ∪ aresta[j];
        break;
      }
}
```

Algoritmo de Prim

- Note que no algoritmo de Prim, uma determinada aresta pode ser examinada diversas vezes como candidata a ser incluída na árvore. No algoritmo de Kruskal, cada aresta é examinada apenas uma vez porque se uma aresta provoca um ciclo em uma iteração do algoritmo, esta aresta também provocaria um ciclo em iterações posteriores. Portanto, o algoritmo de Kruskal é mais eficiente.

Exemplo:



Arestas ordenadas:

(fg), (ac), (ab), (df), (eg), (cb), (cf), (be), (de), (cd)

Árvore de espalhamento mínima construída com as arestas:

(fg), (df), (eg), (cf), (ac) e (ab).

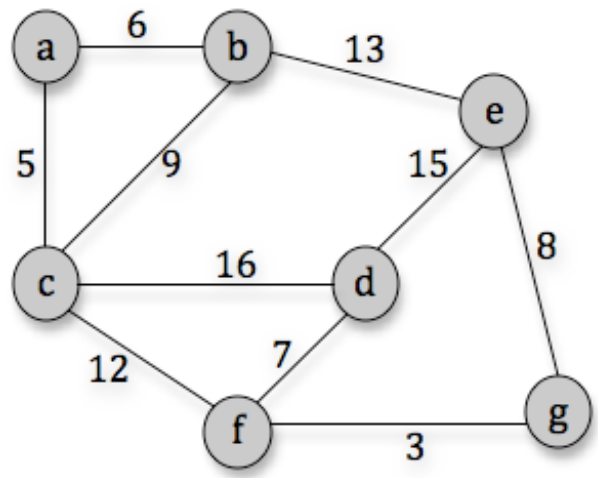
Algoritmo de Dijkstra

- Os algoritmos de Kruskal e de Prim exigem que as arestas do grafo sejam ordenadas. No algoritmo de Dijskstra isto não é necessário.

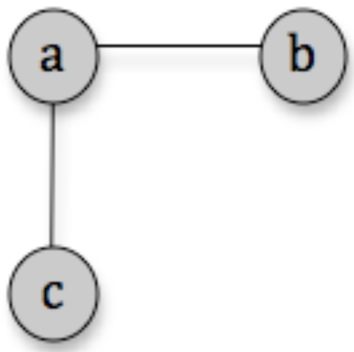
```
AEM_Dijkstra(digrafo G)
{
  arvore = ∅;
  aresta = sequência de arestas de A;
  for (j = 0; j < |A|; j++)
  {
    arvore = arvore ∪ aresta[j];
    if (existe um ciclo em arvore)
      remover a aresta de peso máximo do ciclo;
  }
}
```

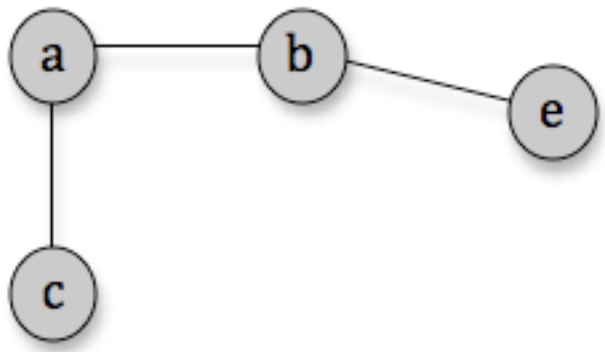
Algoritmo de Dijkstra

Exemplo:



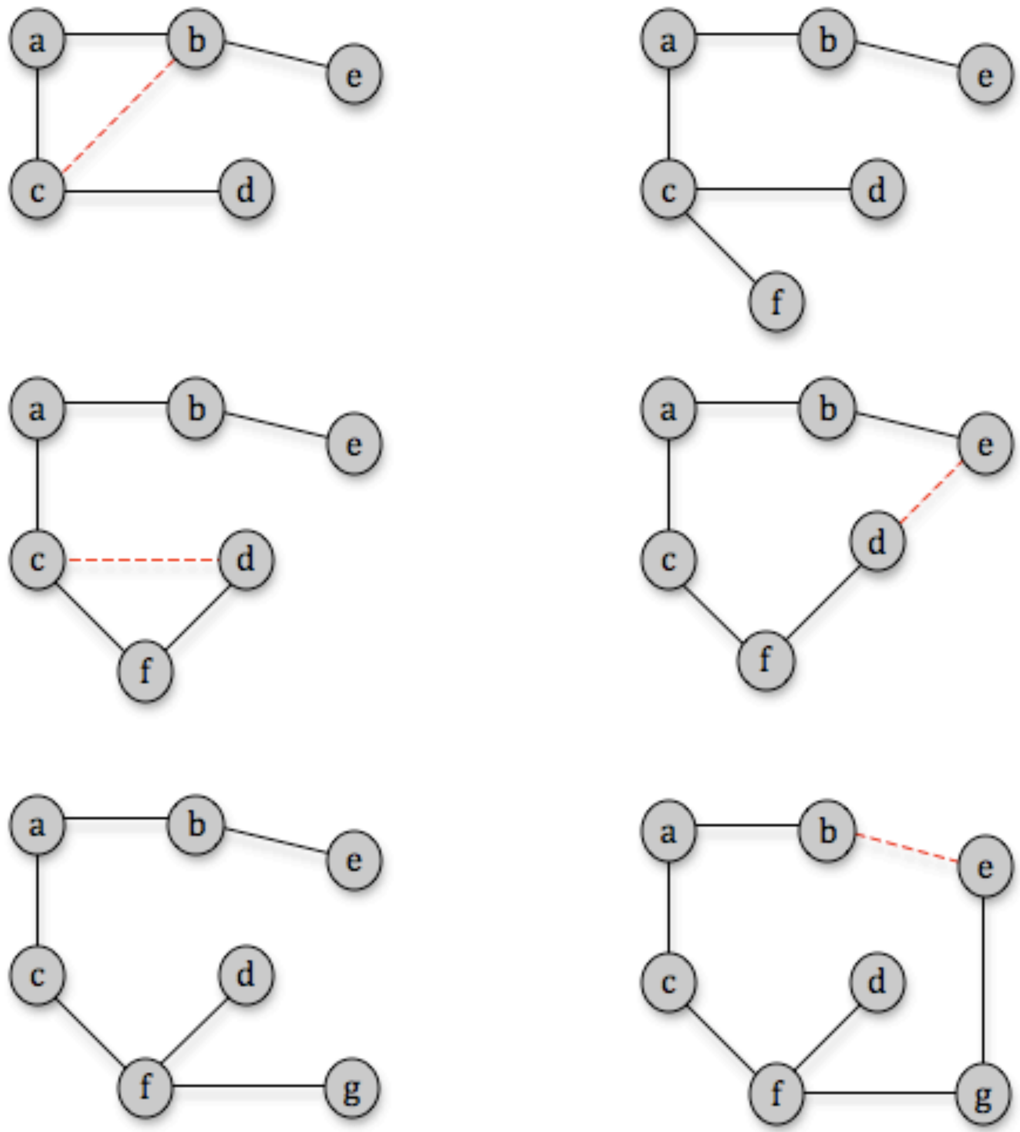
- Considere as arestas na sequência: (ab), (ac), (be), (cb), (cd), (cf), (df), (ed), (fg) e (ge)
- A aplicação do algoritmo de Dijkstra resulta nas árvores:





Algoritmo de Dijkstra

- Arestas: (ab), (ac), (be), (cb), (cd), (cf), (df), (ed), (fg) e (ge)

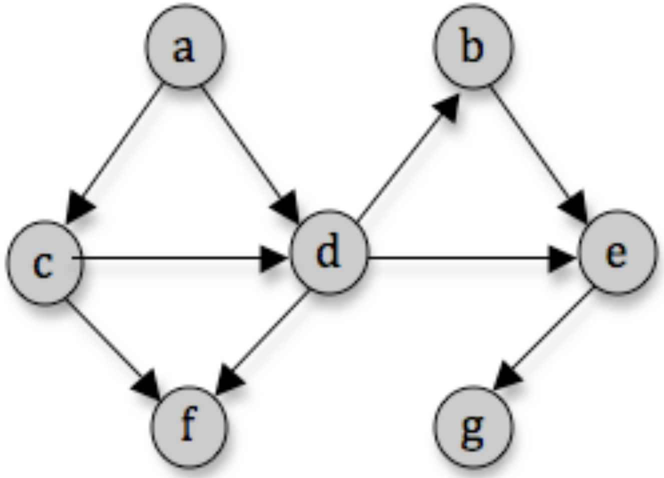


Ordenação topológica

- Em alguns problemas, um conjunto de tarefas devem ser realizadas. Para alguns pares de tarefas, a ordem de execução é relevante, no sentido de que uma tarefa deve necessariamente ser feita antes da outra.
- Estas dependências entre tarefas podem ser representadas por um digrafo.
- Uma **ordenação topológica** de um digrafo G corresponde a uma ordenação de seus vértices tal que, se G contém uma aresta (uv), então u aparece antes de v na ordenação (notar que o contrário não necessariamente precisa ocorrer).
- Evidentemente, se o digrafo contém um ciclo, uma ordenação topológica é impossível.

Ordenação topológica

Exemplo:



Uma ordenação topológica possível: **a, c, d, f, b, e, g**

```
void OrdenacaoTopologica(digrafo G)
{
  for (i = 0; i < |V|; i++)
  {
    v = um vértice mínimo de G;
    empilhar(v);
    remover v de G e todas as arestas incidentes a v;
  }
  mostrarPilha();
}
```

Note que o algoritmo procura encontrar vértices dos quais **não saem arcos**. Estes vértices são denominados **vértices mínimos**.

Grafos bipartidos

- Imagine que cinco tarefas (a, b, c, d, e) devem ser realizadas e existem cinco máquinas (1, 2, 3, 4, 5) capazes de realizar estas tarefas. Cada tarefa deve ser realizada em apenas uma máquina, mas nem todas as máquinas estão habilitadas a realizar todas as tarefas. Por exemplo:

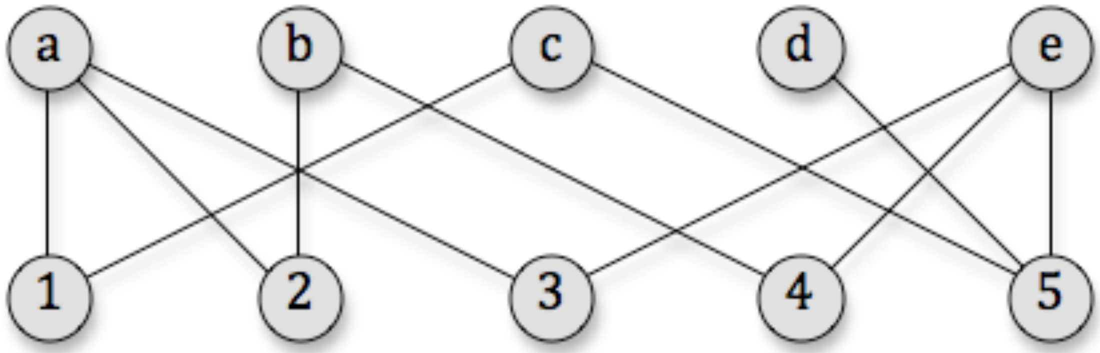
Tarefas	**a**	**b**	**c**	**d**	**e**
Máquinas	1, 2, 3	2, 4	1, 5	5	3, 4, 5

- O problema é encontrar uma máquina para cada tarefa, ou seja, como casar as tarefas com as máquinas.

- Este tipo de problema pode ser representado por um **grafo bipartido**. Um grafo bipartido é um grafo $G = (V, A)$ no qual $V = V_1 \cup V_2$, com $V_1 \cap V_2 = \emptyset$, tal que para qualquer aresta $(uv) \in A$, $u \in V_1$ e $v \in V_2$.

O problema de casamento

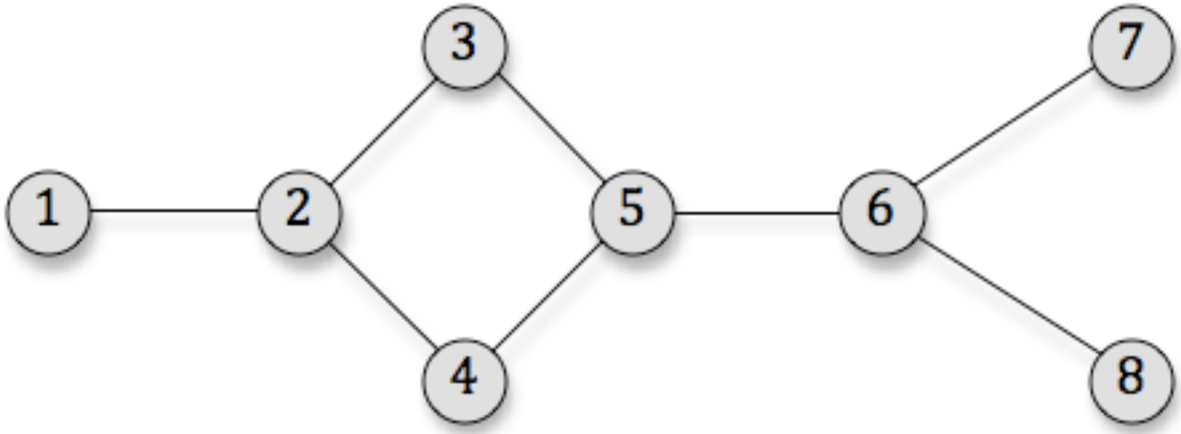
Tarefas	a	b	c	d	e
Máquinas	1, 2, 3	2, 4	1, 5	5	3, 4, 5



- Um **casamento** (ou *matching*) M em um grafo G = (V, A) é um subconjunto de arestas, M ⊆ A, tal que não existem arestas adjacentes (arestas compartilhando um mesmo vértice).
- Um **casamento máximo** é um casamento que contém o maior número possível de arestas, ou seja, em que o número de vértices não-casados seja o menor possível.

O problema de casamento

Exemplo:

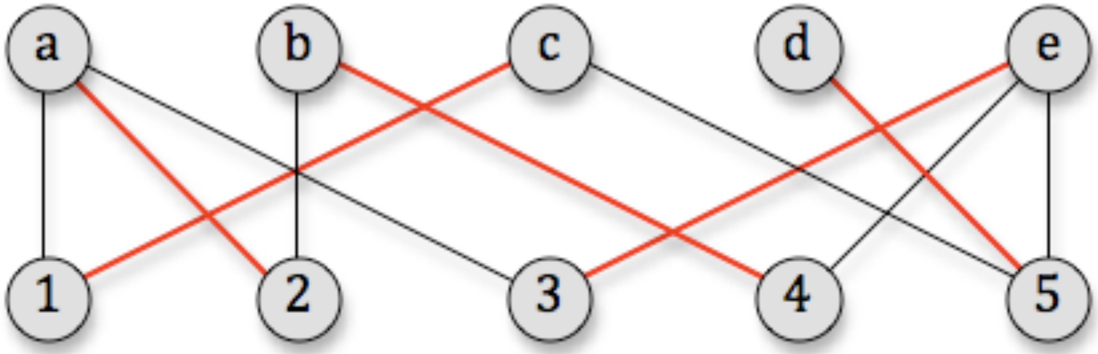


- M_1 = {(12), (56)} e M_2 = {(12), (45), (67)} são casamentos.
- M_2 é um casamento máximo.
- Um **casamento perfeito** é um casamento em que todos os vértices participam (ou seja, em que não existem vértices não-casados).
- Note que M_2, embora seja um casamento máximo, não é um casamento perfeito.
- Existe casamento perfeito para este grafo?

O problema de casamento

- O problema de casamento é encontrar um casamento máximo para um dado grafo.

Exemplo: Problema de tarefas e máquinas



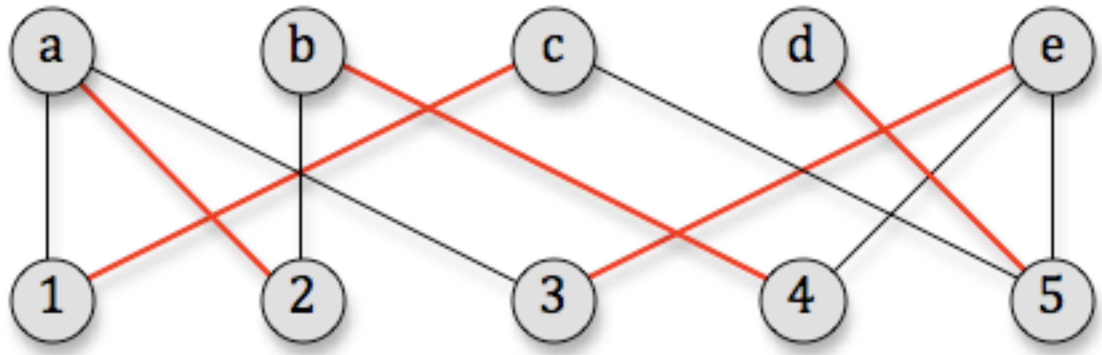
Notar que este é um **casamento perfeito**.

- Um **caminho alternante** para um casamento M é uma sequência de vértices C = (v_1, v_2, ..., v_k) tal que as arestas (v_1v_2), (v_2v_3), ..., (v_{k-1},v_k) pertencem, alternadamente, a M e a A-M. Seja C = (v_1, v_2, ..., v_k) um caminho alternante. Vamos definir como **arestas(C)** o conjunto:

$$\text{arestas}(C) = \{ (v_1v_2), (v_2v_3), \ldots, (v_{k-1},v_k) \}$$

O problema de casamento

Exemplo:



$M = \{(a2), (b4), (c1), (d5), (e3)\}$.

- $C = (3, a, 2, b, 4, e, 3)$ é um caminho alternante para M. Neste caso, arestas(C) = { (3a), (a2), (2b), (b4), (4e), (e3) }.

- Um **caminho de aumento** para um casamento M é um caminho alternante $(v_1, v_2, ..., v_k)$ em que aos vértices das extremidades (v_1 e v_k) não incidem arestas de M.

- A **diferença simétrica** entre dois conjuntos X e Y:

$$X \oplus Y = (X \cup Y) - (X \cap Y)$$

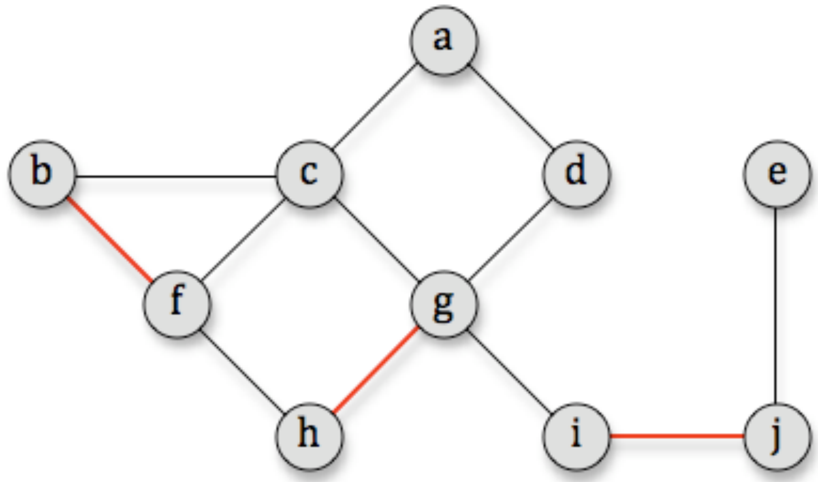
O problema de casamento

- **Lema**: Se M é um casamento e C é um caminho de aumento para M, então M ⊕ arestas(C) é um casamento de cardinalidade |M| + 1.

Exemplo:

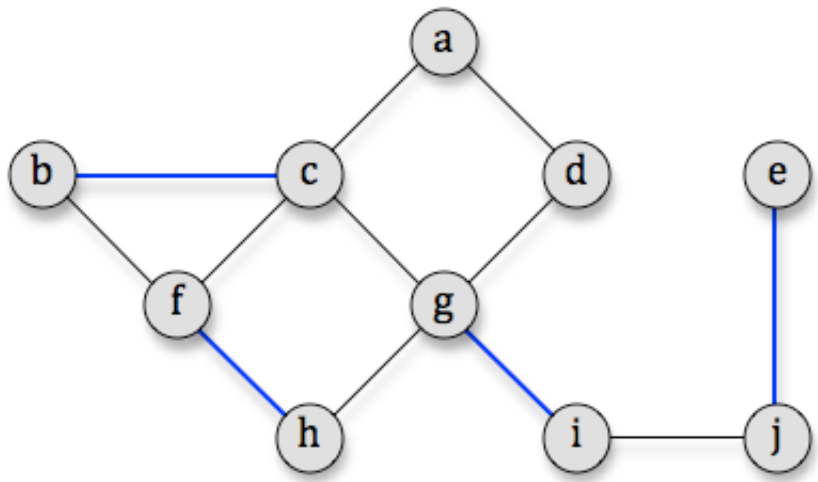
M = {(bf), (gh), (ij)}

C = {c, b, f, h, g, i, j, e}



arestas(C) =
{(cb), (bf), (fh), (hg), (gi), (ij), (je)}

M' = M ⊕ arestas(C) =
{(cb), (fh), (gi), (je)}



Não-casados(M') = {a,d}

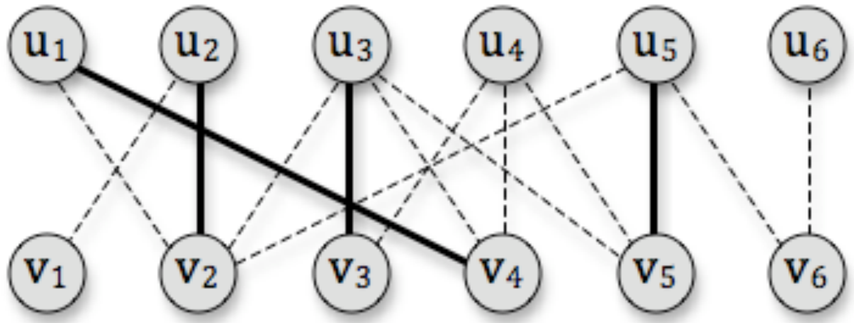
O problema de casamento

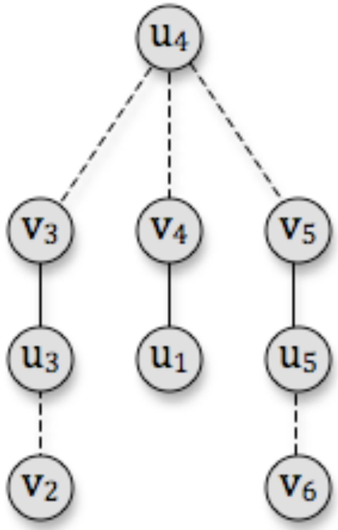
- **Teorema**. Um casamento M em um grafo G é máximo se não existe caminho de aumento para M que conecta dois vértices não-casados de G.
- Este teorema sugere que um casamento máximo pode ser encontrado partindo-se de um casamento inicial e, repetidamente, encontrando novos caminhos de aumento e com isto, construindo casamentos cada vez maiores, até que nenhum caminho de aumento possa ser encontrado.
- A busca em largura pode ser adaptada para encontrar caminhos de aumento em grafos bipartidos. A ideia é construir uma árvore com um vértice não-casado na raiz e incluir novos vértices de modo a constituir um caminho alternante. O procedimento termina assim que se encontre um vértice não-casado diferente da raiz (o que configura um caminho de aumento).

O problema de casamento

```
CasamentoMaximo(grafo-bipartido G)
{
  for (todo vértice não-casado)
  {
    ajustar o nível de todos os vértices em 0;
    ajustar o pai de todos os vértices em nulo;
    fila = ∅;
    ultimo = nulo;
    v = vertice não-casado;
    nivel(v) = 1;
    incluirNaFila(v);
    while (fila != ∅ && ultimo == nulo)
    {
      v = retirarDaFila();
      if (nivel(v) é impar)
      {
        for (todo vértice u adjacente a v tal que nivel(u) = 0)
        {
          if (u é não-casado)
          {
            pai(u) = v;
            ultimo = u;
            break;
          }
          else
          if (u é casado mas não com v)
          {
            pai(u) = v;
            nivel(u) = nivel(v) + 1;
            incluirNaFila(u);
          }
        }
      }
      else
      {
        incluirNaFila(vértice u casado com v);
        pai(u) = v;
        nivel(u) = nivel(v) + 1;
      }
    }
    if (ultimo != nulo)
    {
      for (u = ultimo; u != nulo; u = pai(pai(u)))
      {
        casadoCom(u) = pai(u);
        casadoCom(pai(u)) = u;
      }
    }
  }
}
```

Exemplo:





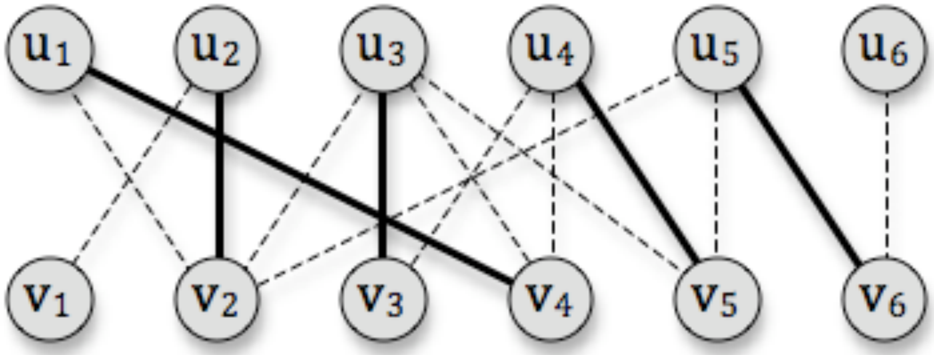
caminho de aumento:
$C = \{u_4, v_5, u_5, v_6\}$

O problema de casamento

$M_0 = \{ (u_1v_4), (u_2v_2), (u_3v_3), (u_5v_5) \}$

$\text{arestas}(C) = \{ (u_4v_5), (v_5u_5), (u_5v_6) \}$

$M_1 = M_0 \oplus \text{arestas}(C) = \{ (u_1v_4), (u_2v_2), (u_3v_3), (u_4v_5), (u_5v_6) \}$



- **Exercício**. Completar a execução do algoritmo para encontrar o casamento máximo.

A complexidade do algoritmo de casamento máximo é **O(|V||A|)**, pois:
a) Cada novo caminho aumenta o número de arestas do casamento em 1;
b) O número máximo de arestas no casamento é |V|/2, portanto:
c) O número máximo de iterações do **for** mais externo é |V|/2;
d) Encontrar um caminho de aumento exige O(|A|) passos.

O problema de atribuição ótima

- O problema de encontrar um casamento torna-se mais difícil no caso de grafos ponderados.
- Para este tipo de grafo interessa encontrar o casamento com peso total máximo. Neste caso, o problema é conhecido como **problema de atribuição**.
- O problema de atribuição para grafos bipartidos completos com dois conjuntos de vértices de mesmo tamanho é conhecido como **problema de atribuição ótima**.
- O **algoritmo de Kuhn-Munkres** (também conhecido como **método húngaro**) resolve o problema de atribuição ótima e tem complexidade $O(|V|^3)$.

O problema de atribuição ótima

- Seja G = (V, A), um grafo bipartido com $V = X \cup Y$. Define-se a **função de rotulação** $f: X \cup Y \rightarrow R$ tal que para todos os vértices u e v, $f(u) + f(v) \geq peso(uv)$.

- Seja H o conjunto de arestas definido como:

$$H = \{ (uv) \in A \mid f(u) + f(v) = peso(uv) \}$$

O subgrafo $G_f = (V, H)$ é conhecido como o **subgrafo de igualdade** de G.

- O algoritmo de Kuhn-Munkres baseia-se no seguinte teorema:

- **Teorema**. Se f é uma função de rotulação e M é um casamento perfeito no subgrafo de igualdade G_f, então M é um casamento ótimo de G.

O problema de atribuição ótima

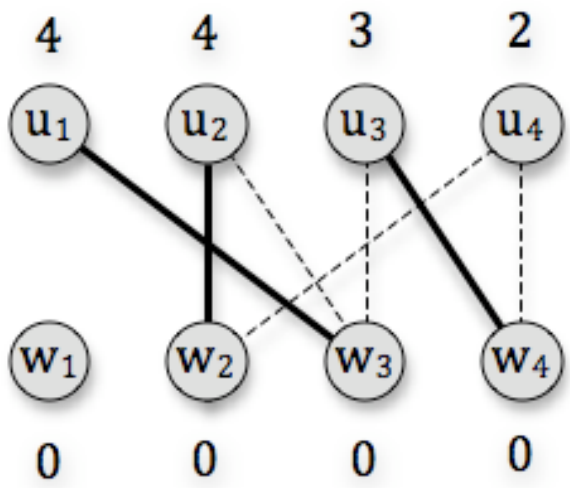
```
AlgoritmoKuhnMunkres(grafo-bipartido G)
{
  Gf = subgrafo de igualdade para f;
  M = casamento em Gf;
  S = {algum vértice não-casado u};
  T = ∅;
  while (M não é casamento perfeito)
  {
    A(S) = {v : ∃ u ∈ S tal que (uv) ∈ Gf};
    if (A(S) == T)
    {
      d = min{(f(u)+f(w)-peso((uw)) tal que u ∈ S e w ∉ T};
      for (cada vértice v)
      {
        if (v ∈ S)
          f(v) = f(v) - d;
        else
        if (v ∈ T)
          f(v) = f(v) + d;
      }
      construir um novo subgrafo de igualdade Gf;
      construir um novo casamento M;
    }
    else
    if (T ⊂ A(S))
    {
      w = um vértice de A(S)-T;
      if (w é não-casado)
      {
        C = caminho de aumento que termina em w;
        M = M ⊕ arestas(C);
        S = {algum vértice não-casado u};
        T = ∅;
      }
      else
      {
        S = S ∪ {vizinho de w em m};
        T = T ∪ {w};
      }
    }
  }
}
```

O problema de atribuição ótima

Exemplo:

- Considere $G = (\{u_1, u_2, u_3, u_4\} \cup \{w_1, w_2, w_3, w_4\}, A)$ um grafo bipartido completo com pesos dados pela matriz:

	w1	w2	w3	w4
u1	2	2	4	1
u2	3	4	4	2
u3	2	2	3	3
u4	1	2	1	2



Seja a rotulação:

f(u) = max(peso(uw)), para todo w;

f(w) = 0.

Neste ponto: $S = \{u_4\}$ e $T = \varnothing$.

Na primeira iteração fazemos:

$A(S) = \{w_2, w_4\}$ (vizinhos em G_f de u_4, que é único elemento de S).

Como $T \subset A(S)$, fazemos $w = w_2$ e como w é casado, fazemos:

$S = \{u_4\} \cup \{u_2, u_4\} = \{u_2, u_4\}$

$T = \varnothing \cup \{w_2\} = \{w_2\}$

O problema de atribuição ótima

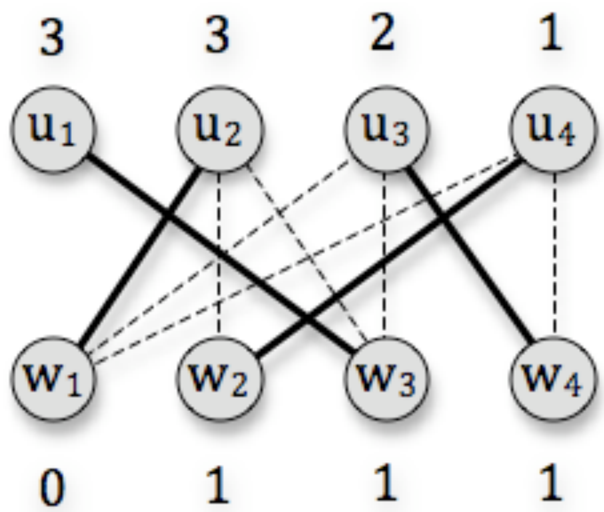
- A tabela resume os valores de S, A(S), w e T ao final das iterações do algoritmo:

Iteração	S	A(S)	w	T
0	$\{u_4\}$	$\varnothing$		$\varnothing$
1	$\{u_2, u_4\}$	$\{w_2, w_4\}$	w_2	$\{w_2\}$
2	$\{u_1, u_2, u_4\}$	$\{w_2, w_3, w_4\}$	w_3	$\{w_2, w_3\}$
3	$\{u_1, u_2, u_3, u_4\}$	$\{w_2, w_3, w_4\}$	w_4	$\{w_2, w_3, w_4\}$
4		$\{w_2, w_3, w_4\}$		

Neste ponto, como A(S) = T, calcula-se a distância d. Como w_1 é o único vértice que não está em T, d = min{(f(u) + f(w_1) – peso(uw_1)), para todo u $\in$ S}, ou seja:

d = min{(4+0-2), (4+0-3), (3+0-2), (2+0-1)} = 1.

Com isso, os rótulos dos vértices de S são aumentados de 1 e rótulos dos vértices de T são diminuídos de 1 e são construídos um novo subgrafo de igualdade e um novo casamento:



Como o novo casamento é um **casamento perfeito no subgrafo de igualdade**, este casamento corresponde à **atribuição ótima**.