Algoritmos e Estruturas de Dados II
Aula 11 – Árvores Geradoras de Custo Mínimo - Algoritmo de Kruskal

Prof. Luciano A. Digiampietri
digiampietri@usp.br
@digiampietri

Árvores Geradoras de Custo Mínimo

Uma **Árvore Geradora de Custo Mínimo** de um grafo G (também chamada de árvore geradora mínima ou *Minimum Spanning Tree - MST*) é uma árvore geradora do grafo G, cuja soma das arestas tenha peso/custo mínimo.

Grafo gerador de G é um subgrafo de G que possui todos os vértices de G.

Árvore é um tipo abstrato de dados composto por elementos conectados que possui um elemento especial chamado raiz que não possui pai e todos os demais elementos possuem um único pai. Ao olharmos a árvore como um grafo, temos um grafo conexo (ou conectado) e acíclico.

Árvores Geradoras de Custo Mínimo

Considerando que as arestas representam o **custo** para se conectar os diversos vértices...

... as árvores geradoras de custo mínimo correspondem a **forma mais barata** de se conectar todos os vértices em um único componente.

Por exemplo, **menor custo para se conectar todos os computadores de uma rede** (os computadores correspondem aos vértices e as conexões às arestas).

Árvores Geradoras de Custo Mínimo

Tipicamente são encontradas em **grafos conexos, ponderados e não direcionados**.

Podem existir **diversas árvores** geradoras de custo mínimo de um único grafo.

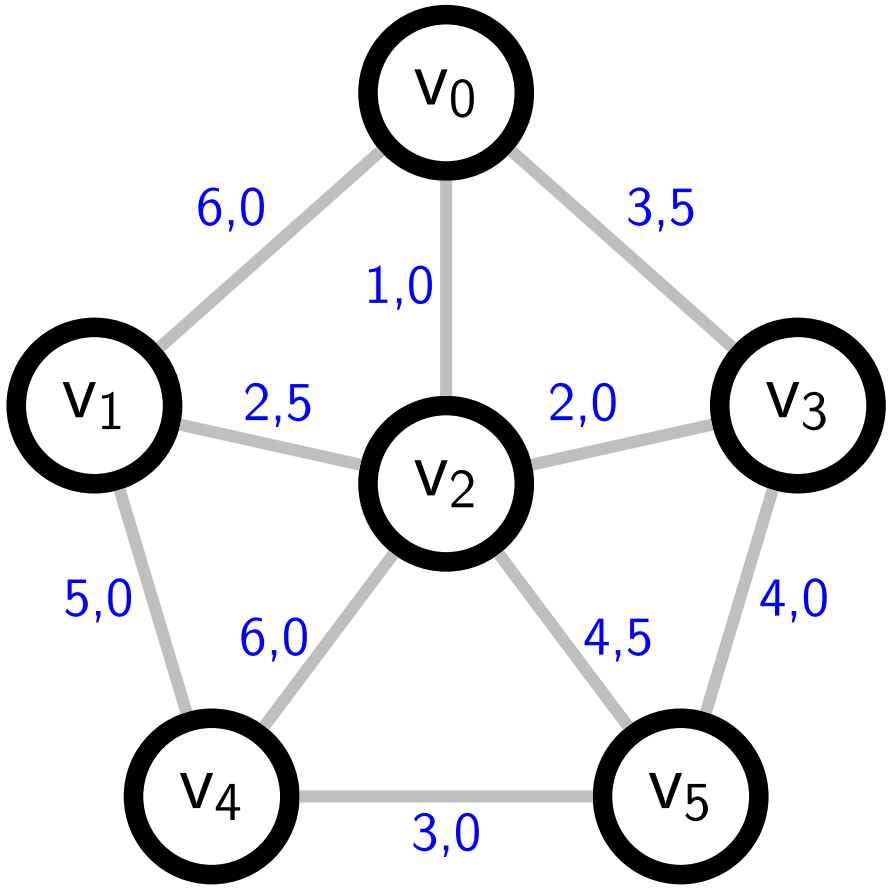
Existem diferentes abordagens para identificar essas árvores.

Algoritmo de Kruskal

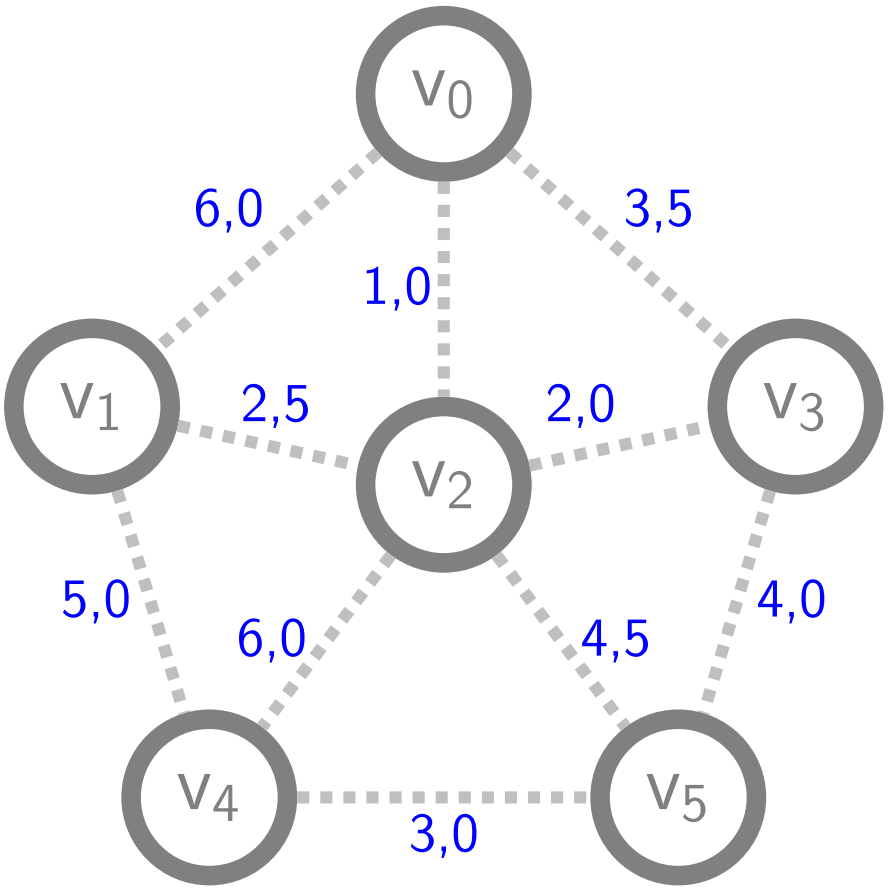
Ideia geral do algoritmo:

- Crie uma floresta apenas com os vértices do grafo G
- Ordene as arestas de G de forma crescente
- Percorra as arestas em ordem
  - Se a aresta unir vértices de diferentes árvores da floresta, adicione a aresta à árvore geradora de custo mínimo (una as duas árvores envolvidas)

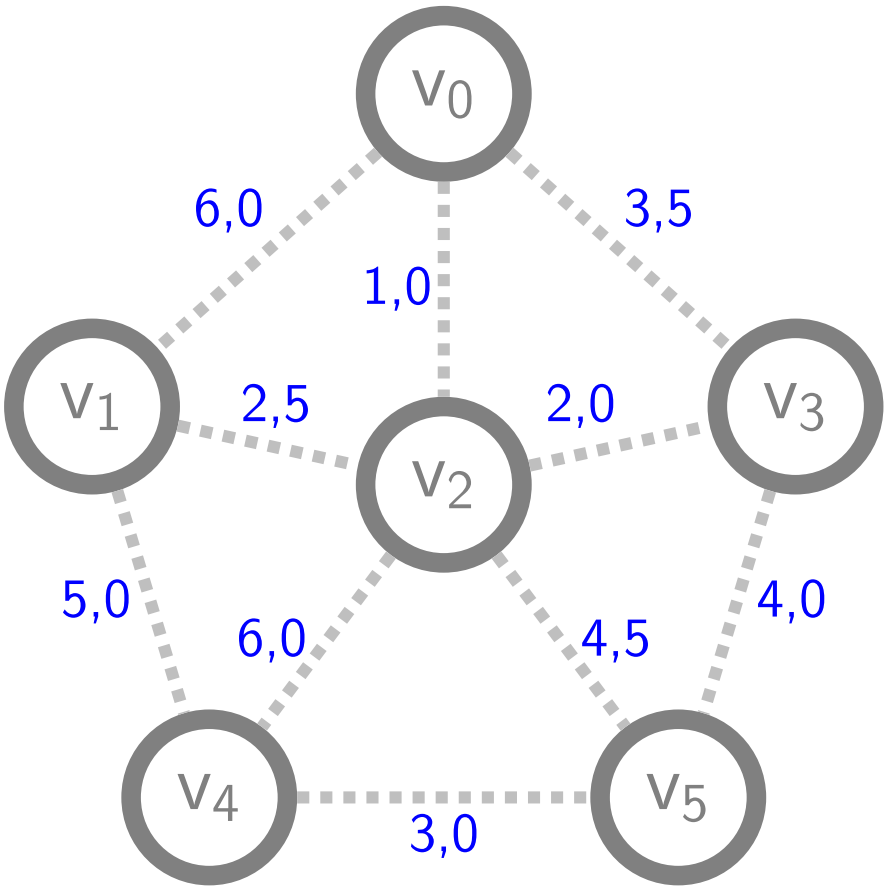
Árvore Geradora de Custo Mínimo - Prim



Árvore Geradora de Custo Mínimo - Kruskal



Árvore Geradora de Custo Mínimo - Kruskal

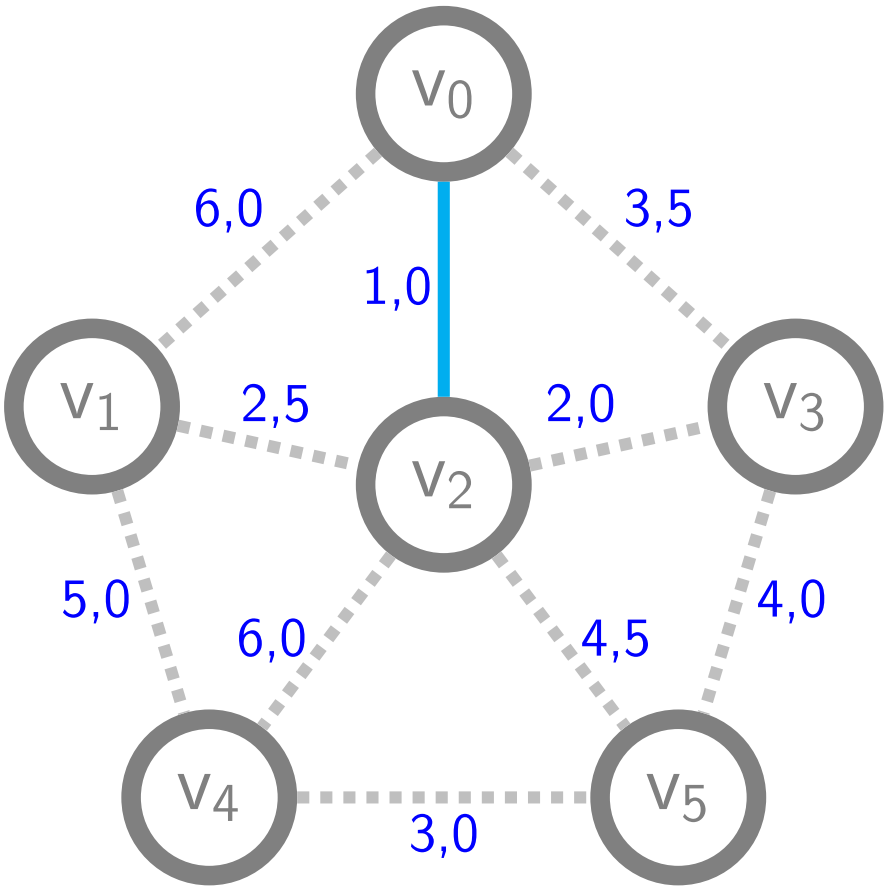


Custo:

Arestas Ordenadas:

0-2 (1,0)
2-3 (2,0)
2-1 (2,5)
4-5 (3,0)
0-3 (3,5)
3-5 (4,0)
2-5 (4,5)
1-4 (5,0)
0-1 (6,0)
2-4 (6,0)

Árvore Geradora de Custo Mínimo - Kruskal

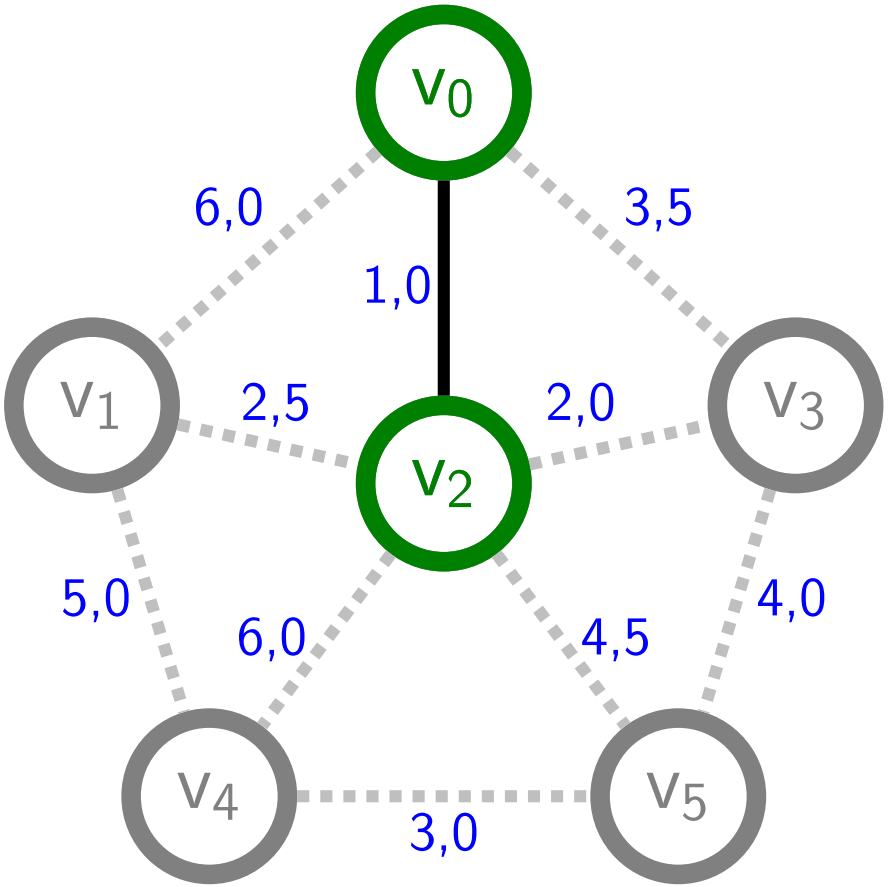


Custo:

Arestas Ordenadas:

0-2 (1,0)
2-3 (2,0)
2-1 (2,5)
4-5 (3,0)
0-3 (3,5)
3-5 (4,0)
2-5 (4,5)
1-4 (5,0)
0-1 (6,0)
2-4 (6,0)

Árvore Geradora de Custo Mínimo - Kruskal

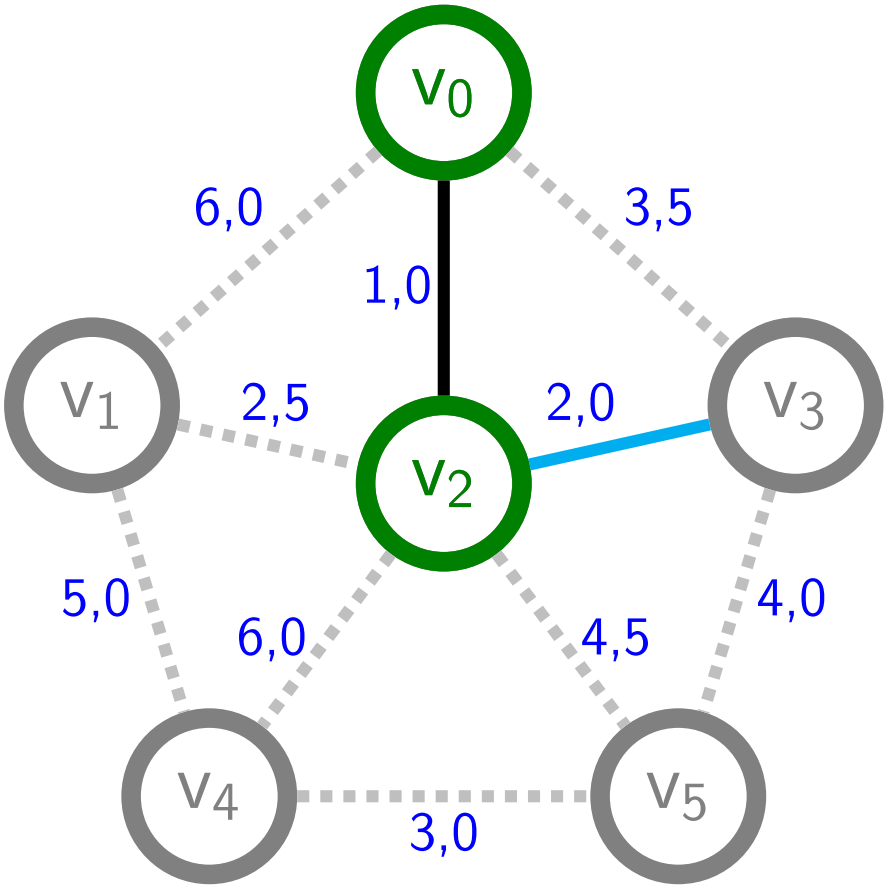


Custo: 1,0

Arestas Ordenadas:

0-2 (1,0)
2-3 (2,0)
2-1 (2,5)
4-5 (3,0)
0-3 (3,5)
3-5 (4,0)
2-5 (4,5)
1-4 (5,0)
0-1 (6,0)
2-4 (6,0)

Árvore Geradora de Custo Mínimo - Kruskal

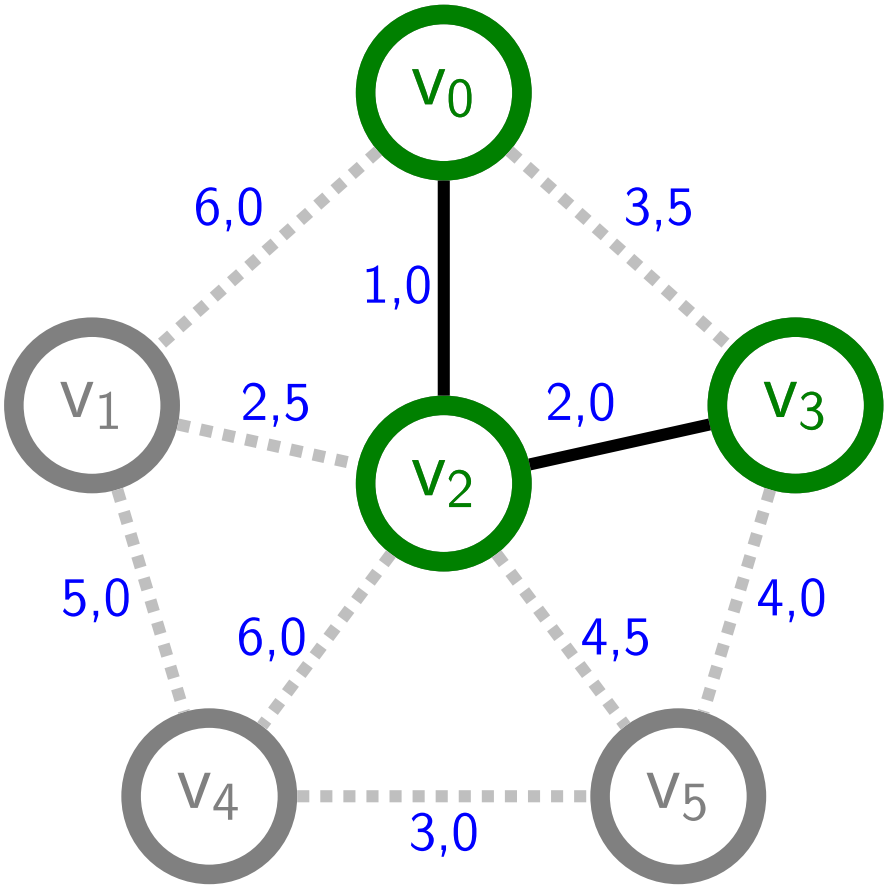


Custo: 1,0

Arestas Ordenadas:

0-2 (1,0)
2-3 (2,0)
2-1 (2,5)
4-5 (3,0)
0-3 (3,5)
3-5 (4,0)
2-5 (4,5)
1-4 (5,0)
0-1 (6,0)
2-4 (6,0)

Árvore Geradora de Custo Mínimo - Kruskal

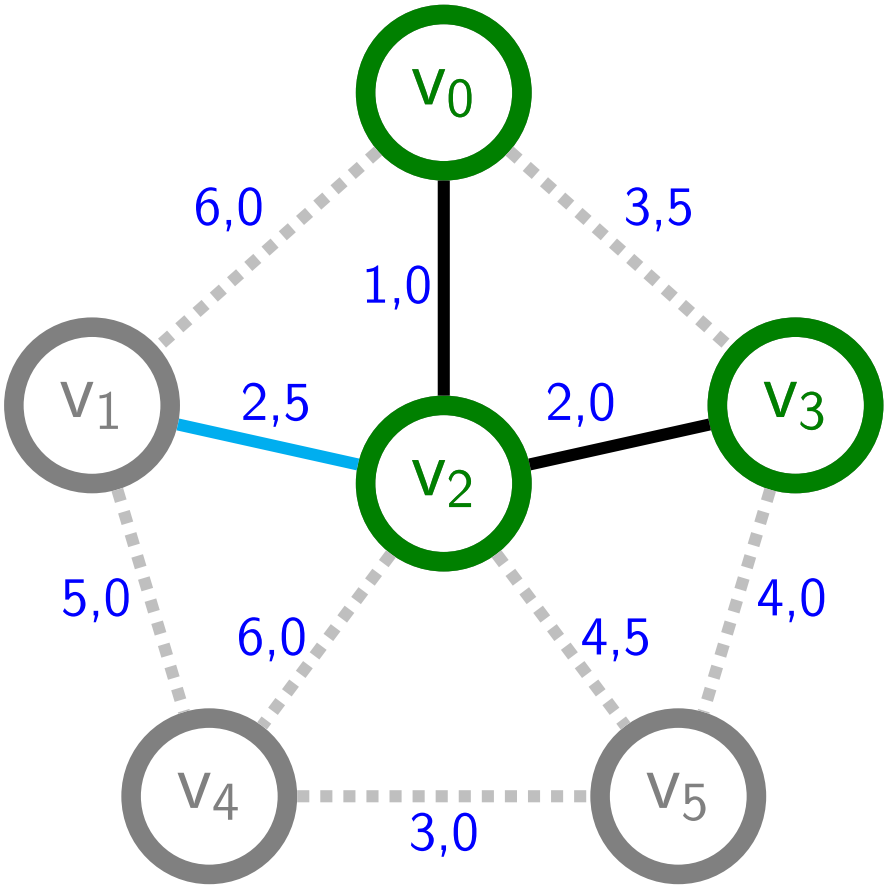


Custo: 3,0

Arestas Ordenadas:

0-2 (1,0)
2-3 (2,0)
2-1 (2,5)
4-5 (3,0)
0-3 (3,5)
3-5 (4,0)
2-5 (4,5)
1-4 (5,0)
0-1 (6,0)
2-4 (6,0)

Árvore Geradora de Custo Mínimo - Kruskal

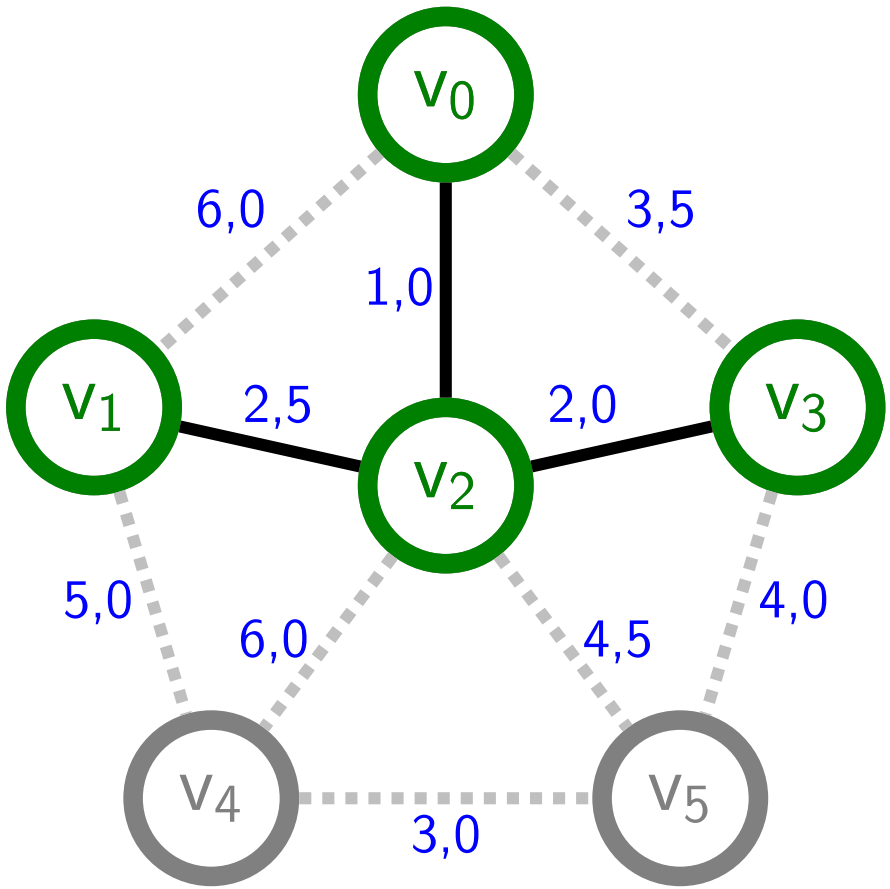


Custo: 3,0

Arestas Ordenadas:

0-2 (1,0)
2-3 (2,0)
2-1 (2,5)
4-5 (3,0)
0-3 (3,5)
3-5 (4,0)
2-5 (4,5)
1-4 (5,0)
0-1 (6,0)
2-4 (6,0)

Árvore Geradora de Custo Mínimo - Kruskal

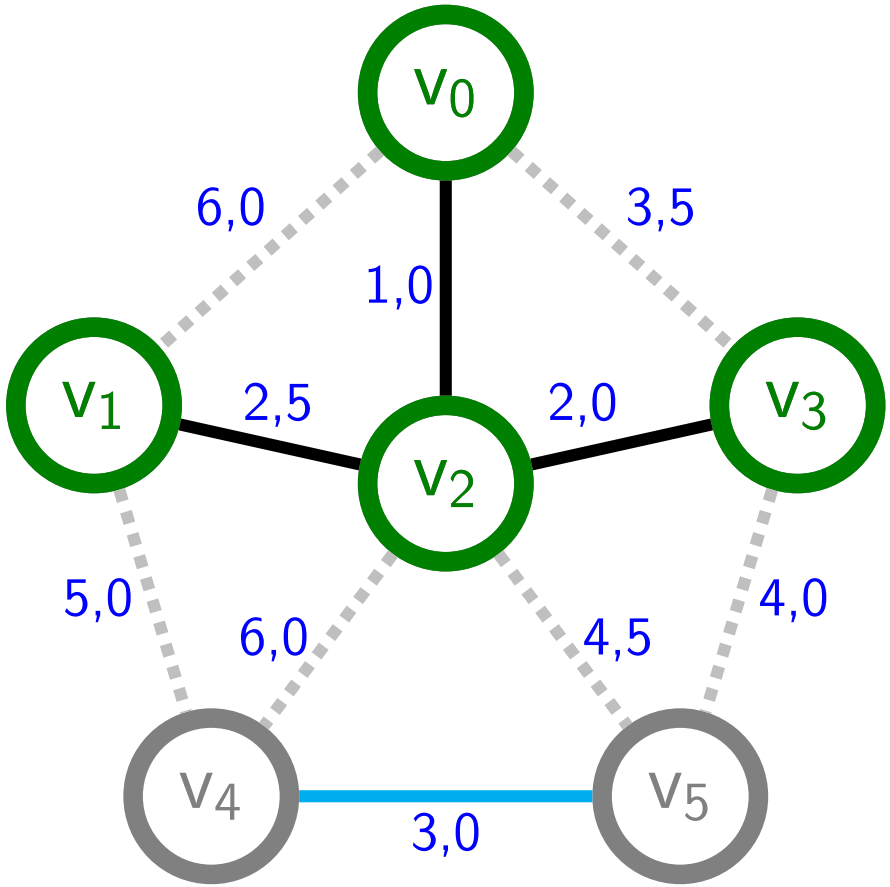


Custo: 5,5

Arestas Ordenadas:

0-2 (1,0)
2-3 (2,0)
2-1 (2,5)
4-5 (3,0)
0-3 (3,5)
3-5 (4,0)
2-5 (4,5)
1-4 (5,0)
0-1 (6,0)
2-4 (6,0)

Árvore Geradora de Custo Mínimo - Kruskal

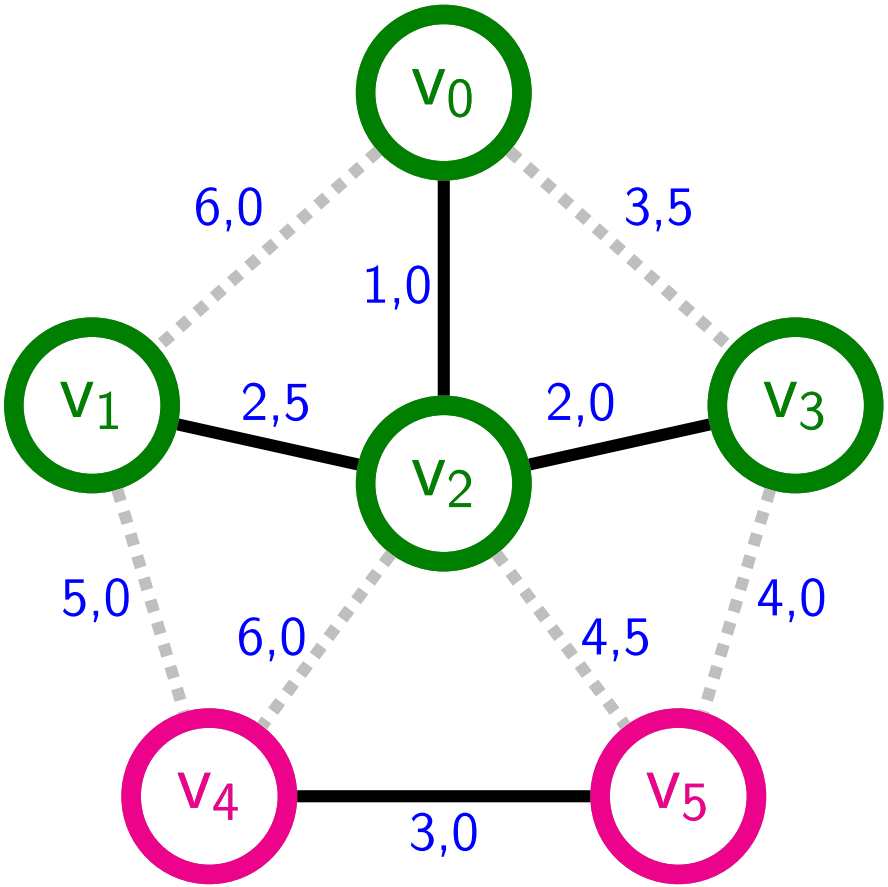


Custo: 5,5

Arestas Ordenadas:

0-2 (1,0)
2-3 (2,0)
2-1 (2,5)
4-5 (3,0)
0-3 (3,5)
3-5 (4,0)
2-5 (4,5)
1-4 (5,0)
0-1 (6,0)
2-4 (6,0)

Árvore Geradora de Custo Mínimo - Kruskal

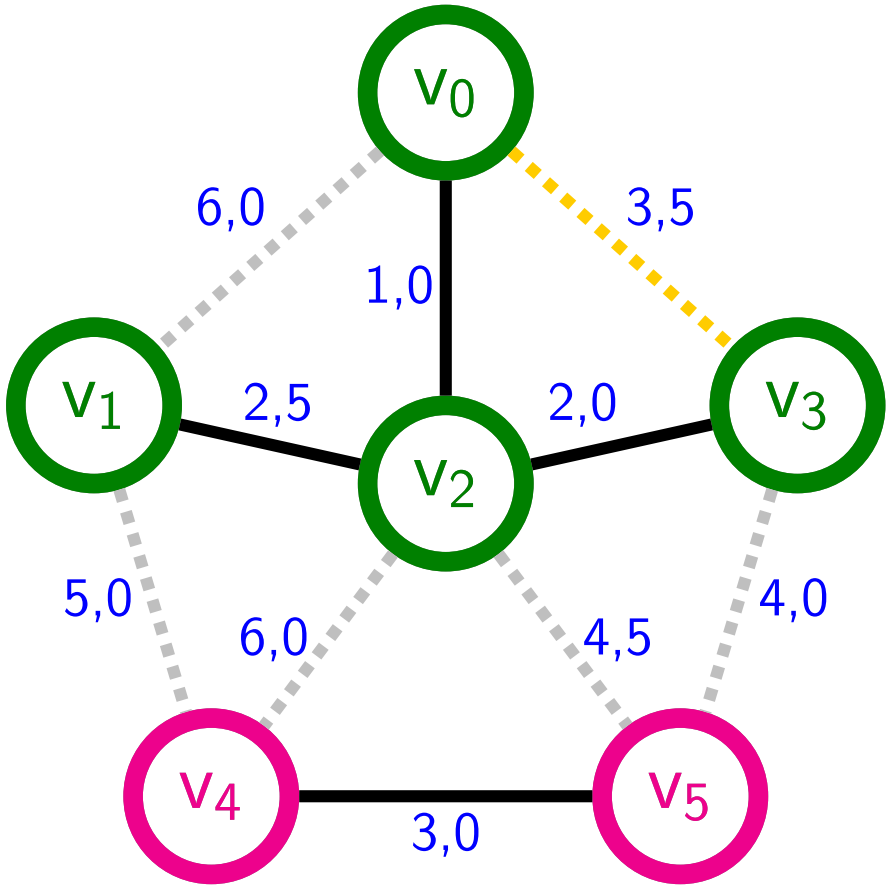


Custo: 8,5

Arestas Ordenadas:

0-2 (1,0)
2-3 (2,0)
2-1 (2,5)
4-5 (3,0)
0-3 (3,5)
3-5 (4,0)
2-5 (4,5)
1-4 (5,0)
0-1 (6,0)
2-4 (6,0)

Árvore Geradora de Custo Mínimo - Kruskal

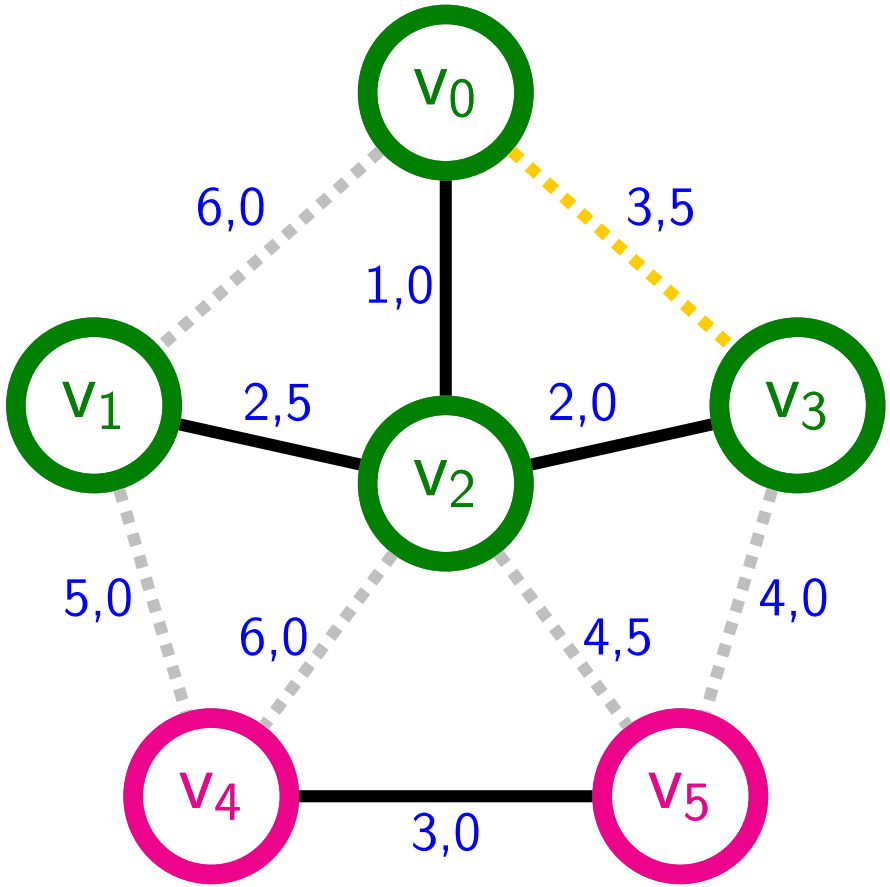


Custo: 8,5

Arestas Ordenadas:

0-2 (1,0)
2-3 (2,0)
2-1 (2,5)
4-5 (3,0)
0-3 (3,5)
3-5 (4,0)
2-5 (4,5)
1-4 (5,0)
0-1 (6,0)
2-4 (6,0)

Árvore Geradora de Custo Mínimo - Kruskal

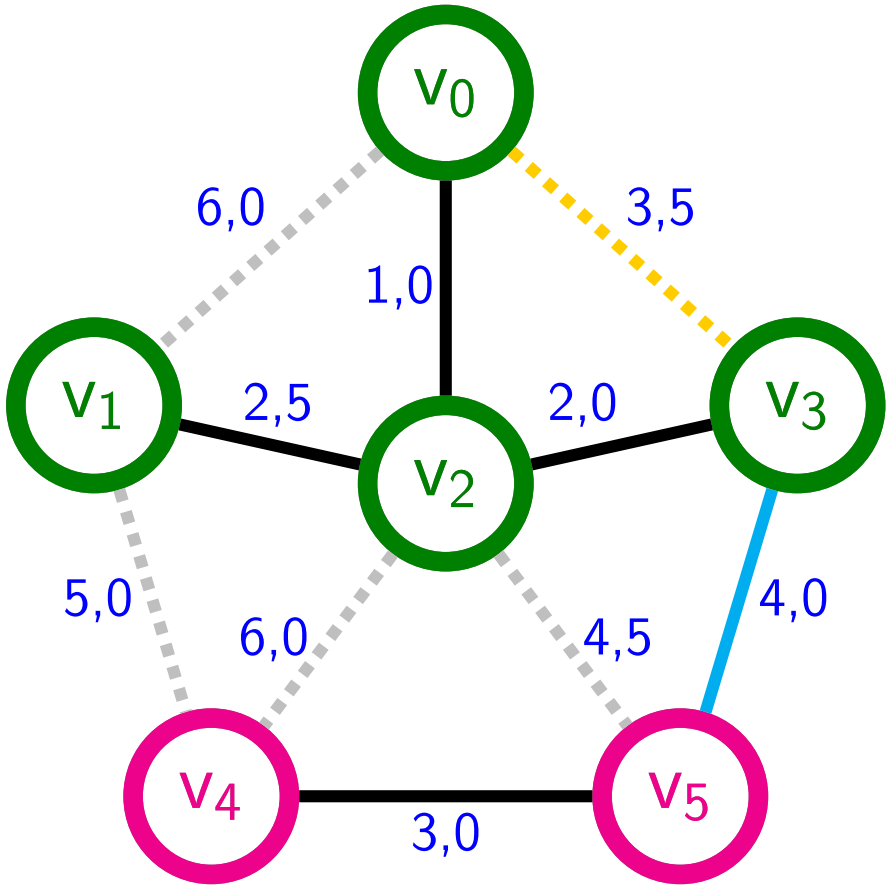


Custo: 8,5

Arestas Ordenadas:

0-2 (1,0)
2-3 (2,0)
2-1 (2,5)
4-5 (3,0)
~~0-3 (3,5)~~
3-5 (4,0)
2-5 (4,5)
1-4 (5,0)
0-1 (6,0)
2-4 (6,0)

Árvore Geradora de Custo Mínimo - Kruskal

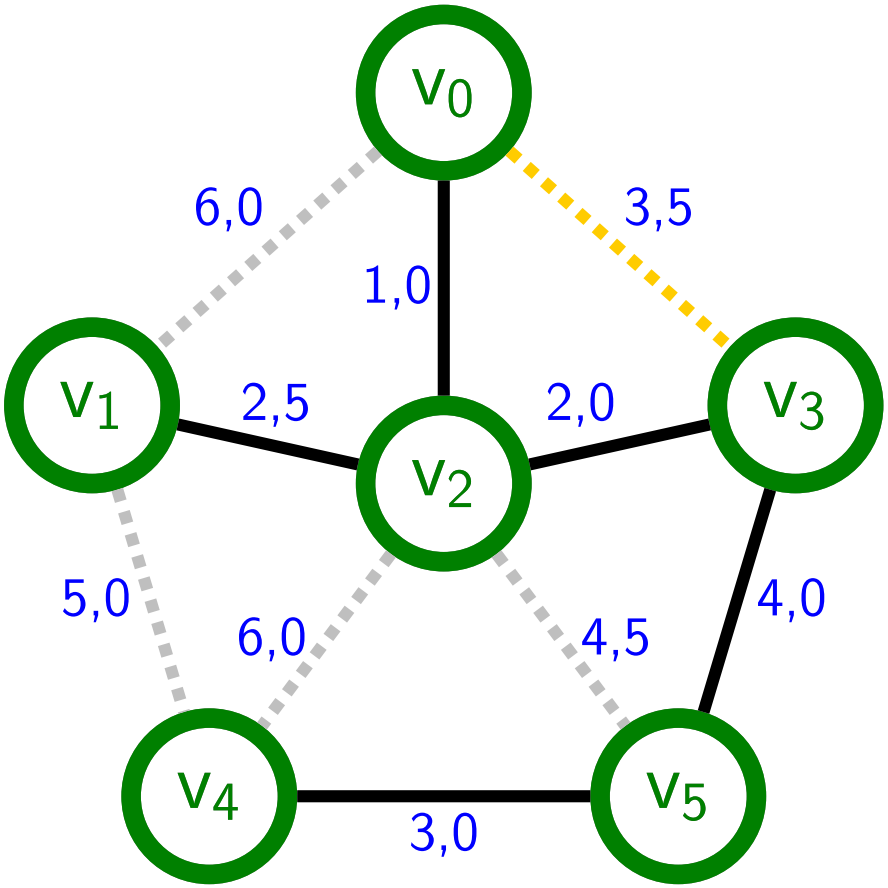


Custo: 8,5

Arestas Ordenadas:

0-2 (1,0)
2-3 (2,0)
2-1 (2,5)
4-5 (3,0)
~~0-3 (3,5)~~
3-5 (4,0)
2-5 (4,5)
1-4 (5,0)
0-1 (6,0)
2-4 (6,0)

Árvore Geradora de Custo Mínimo - Kruskal



Custo: 12,5

Arestas Ordenadas:

0-2 (1,0)
2-3 (2,0)
2-1 (2,5)
4-5 (3,0)
~~0-3 (3,5)~~
3-5 (4,0)
2-5 (4,5)
1-4 (5,0)
0-1 (6,0)
2-4 (6,0)

Algoritmo de Kruskal

Há **dois conceitos** muito importantes no algoritmo:

Algoritmo de Kruskal

Há **dois conceitos** muito importantes no algoritmo:

- **Ordenar** as arestas de forma crescente pelo custo

Algoritmo de Kruskal

Há **dois conceitos** muito importantes no algoritmo:

- **Ordenar** as arestas de forma crescente pelo custo
  - Conhecemos vários algoritmos de ordenação, como o **Heap Sort** com complexidade O(E log(E))

Algoritmo de Kruskal

Há **dois conceitos** muito importantes no algoritmo:

- **Ordenar** as arestas de forma crescente pelo custo
  - Conhecemos vários algoritmos de ordenação, como o **Heap Sort** com complexidade O(E log(E))
- Para cada aresta, temos que verificar se ela conecta **vértices de árvores diferentes**
  - Utilizaremos uma estrutura de dados conhecida como **conjuntos disjuntos** (*disjoint-set*) também conhecida como união-busca (*union–find*)

Conjuntos Disjuntos

Conjuntos Disjuntos

- Gerencia um **conjunto de elementos** e cada elemento pertence a um **único conjunto**

Conjuntos Disjuntos

- Gerencia um **conjunto de elementos** e cada elemento pertence a um **único conjunto**
- Principais operações:

Conjuntos Disjuntos

- Gerencia um **conjunto de elementos** e cada elemento pertence a um **único conjunto**
- Principais operações:
  - **Inicializar** conjuntos (tipicamente cada elemento inicia em um conjunto próprio)

Conjuntos Disjuntos

- Gerencia um **conjunto de elementos** e cada elemento pertence a um **único conjunto**
- Principais operações:
  - **Inicializar** conjuntos (tipicamente cada elemento inicia em um conjunto próprio)
  - Identificar/**Encontrar o conjunto** a que um elemento pertence
  - **União de conjuntos**: fusão de dois conjuntos em um só.

Conjuntos Disjuntos

Uma implementação típica utiliza **florestas**:

Conjuntos Disjuntos

Uma implementação típica utiliza **florestas**:

- Na **inicialização** cada elemento pertence a uma árvore diferente

Conjuntos Disjuntos

Uma implementação típica utiliza **florestas**:

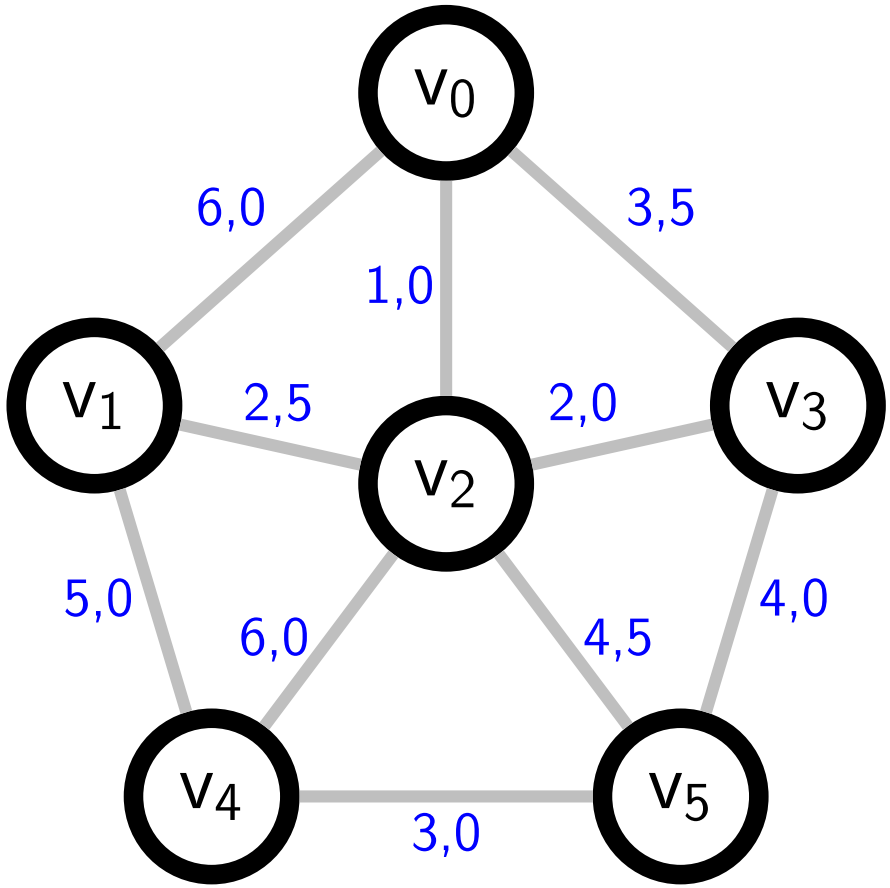
- Na **inicialização** cada elemento pertence a uma árvore diferente
- A **raiz de cada árvore representa o conjunto** correspondente a cada um de seus elementos (pode ou não haver um campo na raiz indicando o conjunto)

Conjuntos Disjuntos

Uma implementação típica utiliza **florestas**:

- Na **inicialização** cada elemento pertence a uma árvore diferente
- A **raiz de cada árvore representa o conjunto** correspondente a cada um de seus elementos (pode ou não haver um campo na raiz indicando o conjunto)
- Para **unir dois conjuntos**, colocamos a raiz de um conjunto como filha da raiz do outro conjunto.

Conjuntos Disjuntos e Kruskal



Conjuntos Disjuntos e Kruskal

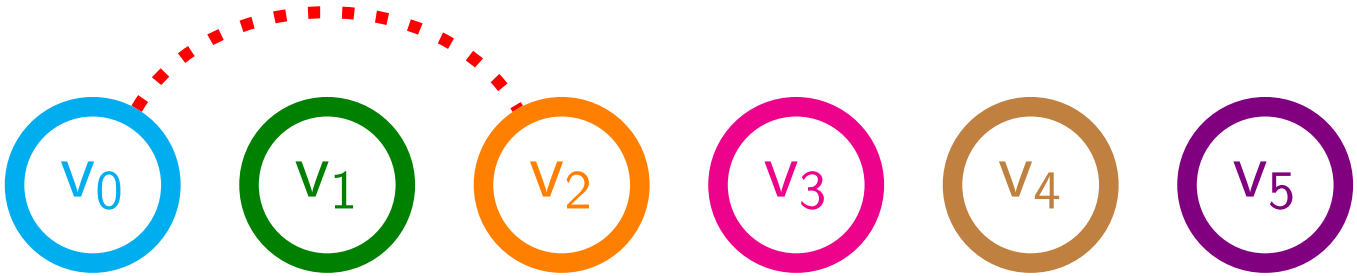
Conjuntos Disjuntos e Kruskal

Custo:

Arestas Ordenadas:

0-2 (1,0)
2-3 (2,0)
2-1 (2,5)
4-5 (3,0)
0-3 (3,5)
3-5 (4,0)
2-5 (4,5)
1-4 (5,0)
0-1 (6,0)
2-4 (6,0)

Conjuntos Disjuntos e Kruskal

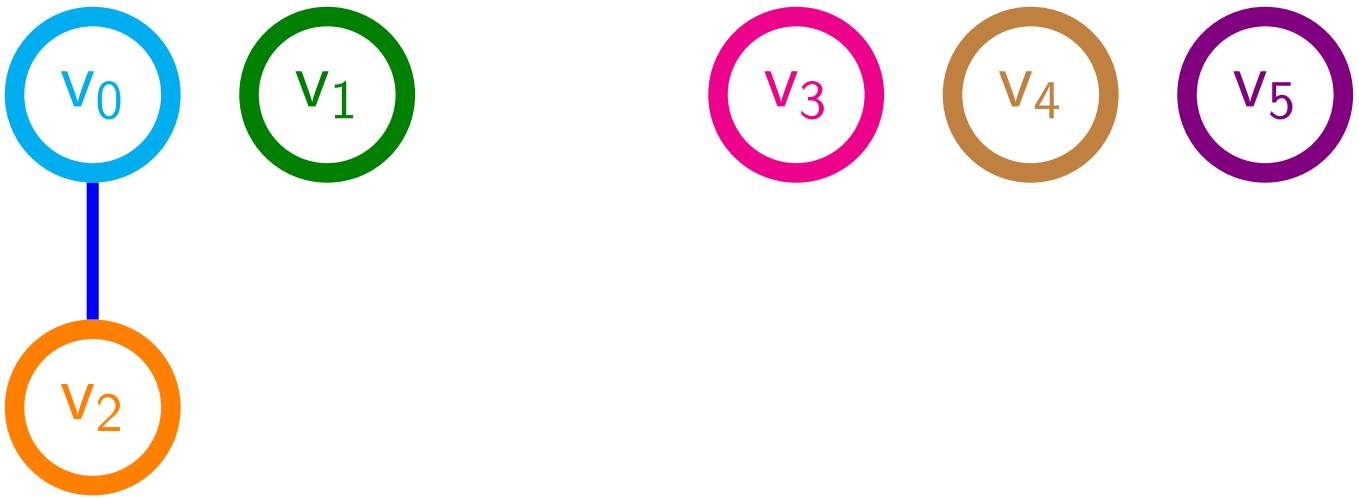


Custo:

Arestas Ordenadas:

0-2 (1,0)
2-3 (2,0)
2-1 (2,5)
4-5 (3,0)
0-3 (3,5)
3-5 (4,0)
2-5 (4,5)
1-4 (5,0)
0-1 (6,0)
2-4 (6,0)

Conjuntos Disjuntos e Kruskal

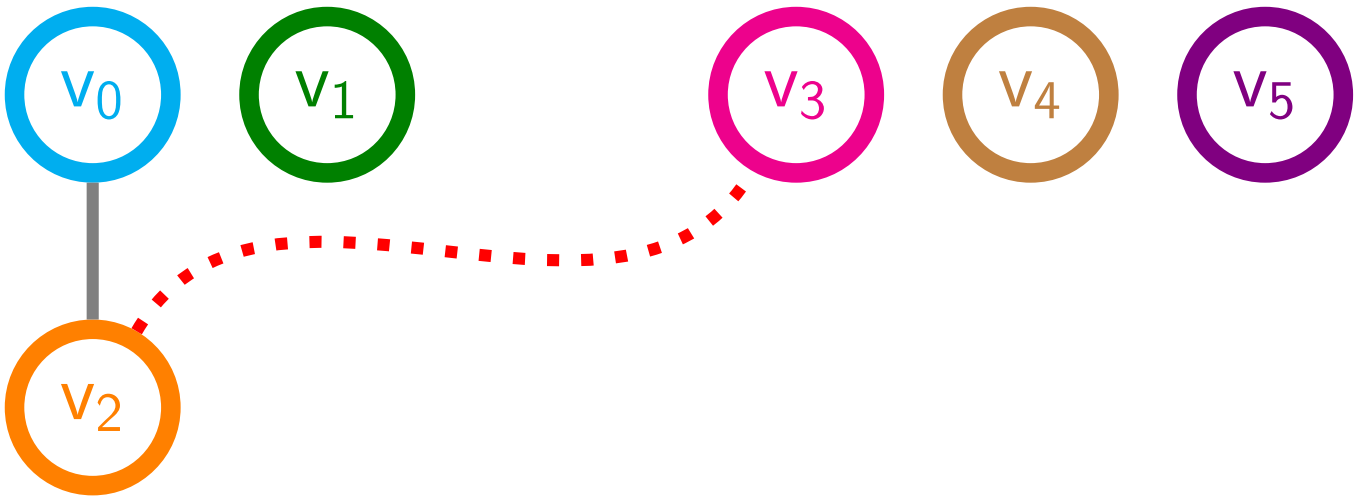


Custo: 1,0

Arestas Ordenadas:

0-2 (1,0)
2-3 (2,0)
2-1 (2,5)
4-5 (3,0)
0-3 (3,5)
3-5 (4,0)
2-5 (4,5)
1-4 (5,0)
0-1 (6,0)
2-4 (6,0)

Conjuntos Disjuntos e Kruskal



Custo: 1,0

Arestas Ordenadas:

0-2 (1,0)
2-3 (2,0)
2-1 (2,5)
4-5 (3,0)
0-3 (3,5)
3-5 (4,0)
2-5 (4,5)
1-4 (5,0)
0-1 (6,0)
2-4 (6,0)

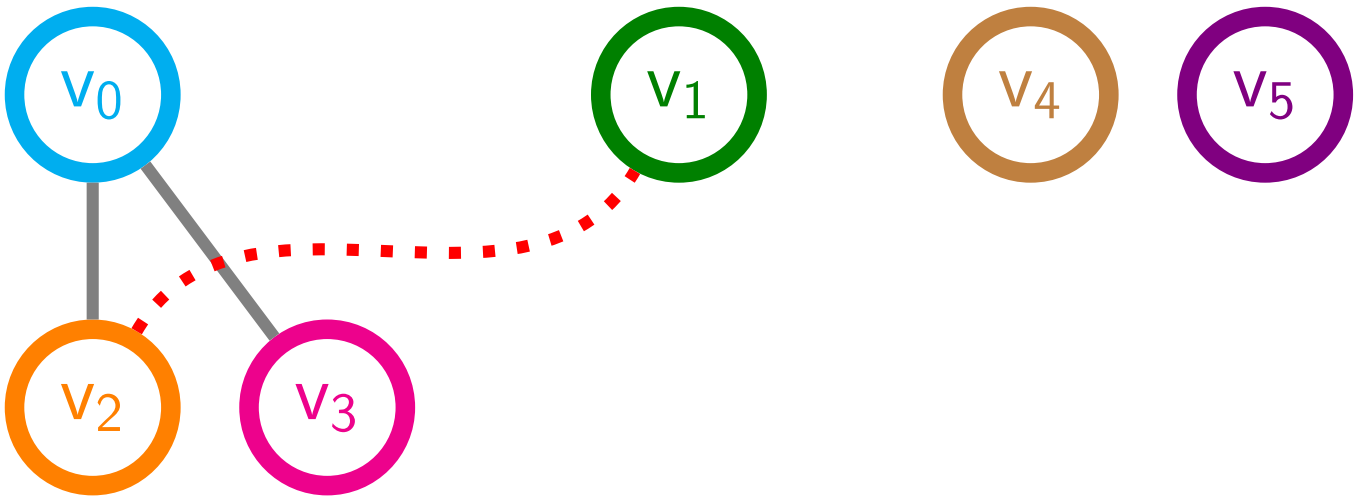
Conjuntos Disjuntos e Kruskal

Custo: 3,0

Arestas Ordenadas:

0-2 (1,0)
2-3 (2,0)
2-1 (2,5)
4-5 (3,0)
0-3 (3,5)
3-5 (4,0)
2-5 (4,5)
1-4 (5,0)
0-1 (6,0)
2-4 (6,0)

Conjuntos Disjuntos e Kruskal

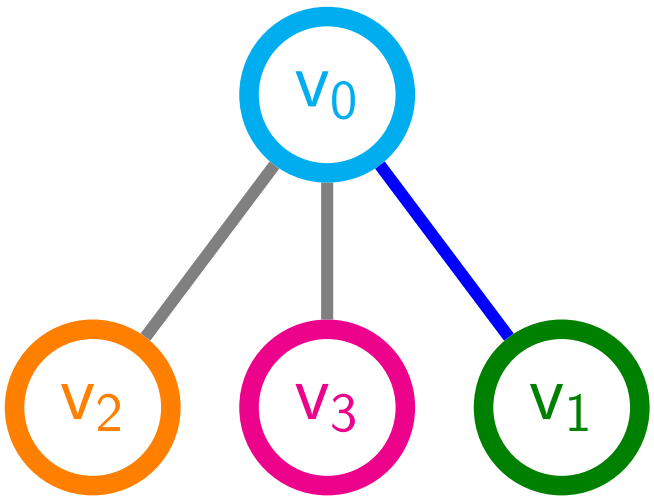


Custo: 3,0

Arestas Ordenadas:

0-2 (1,0)
2-3 (2,0)
2-1 (2,5)
4-5 (3,0)
0-3 (3,5)
3-5 (4,0)
2-5 (4,5)
1-4 (5,0)
0-1 (6,0)
2-4 (6,0)

Conjuntos Disjuntos e Kruskal

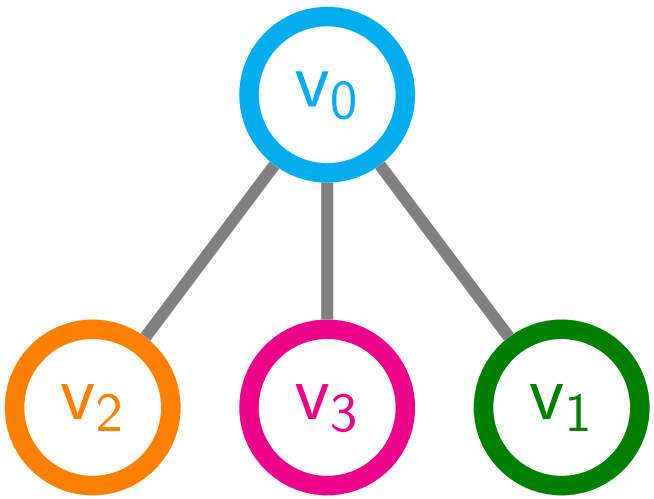


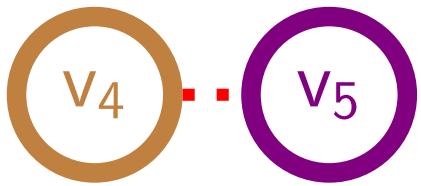
Custo: 5,5

Arestas Ordenadas:

0-2 (1,0)
2-3 (2,0)
2-1 (2,5)
4-5 (3,0)
0-3 (3,5)
3-5 (4,0)
2-5 (4,5)
1-4 (5,0)
0-1 (6,0)
2-4 (6,0)

Conjuntos Disjuntos e Kruskal



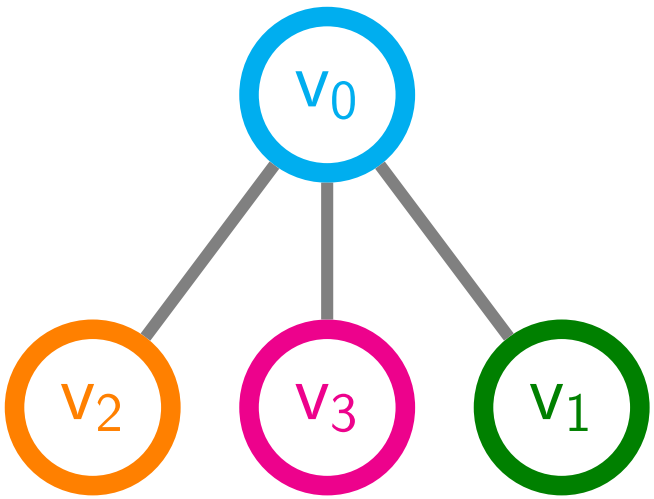


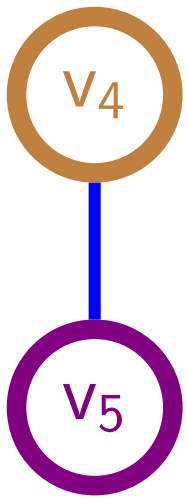
Custo: 5,5

Arestas Ordenadas:

0-2 (1,0)
2-3 (2,0)
2-1 (2,5)
4-5 (3,0)
0-3 (3,5)
3-5 (4,0)
2-5 (4,5)
1-4 (5,0)
0-1 (6,0)
2-4 (6,0)

Conjuntos Disjuntos e Kruskal



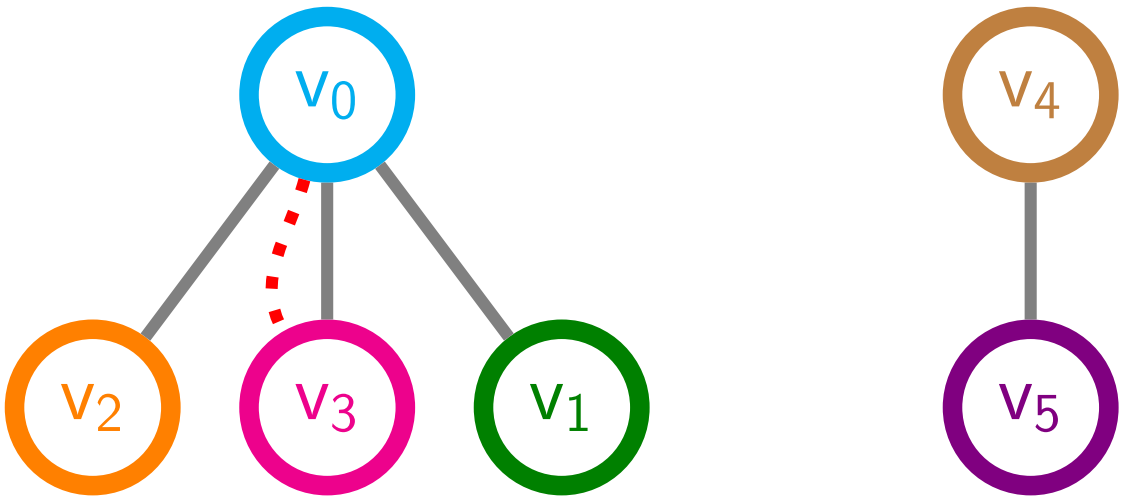


Custo: 8,5

Arestas Ordenadas:

0-2 (1,0)
2-3 (2,0)
2-1 (2,5)
4-5 (3,0)
0-3 (3,5)
3-5 (4,0)
2-5 (4,5)
1-4 (5,0)
0-1 (6,0)
2-4 (6,0)

Conjuntos Disjuntos e Kruskal

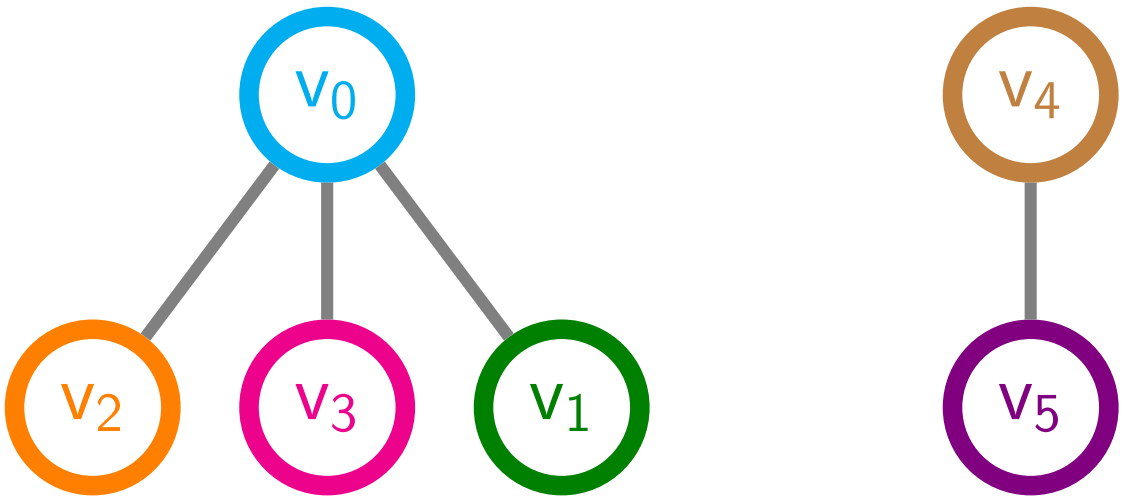


Custo: 8,5

Arestas Ordenadas:

0-2 (1,0)
2-3 (2,0)
2-1 (2,5)
4-5 (3,0)
0-3 (3,5)
3-5 (4,0)
2-5 (4,5)
1-4 (5,0)
0-1 (6,0)
2-4 (6,0)

Conjuntos Disjuntos e Kruskal

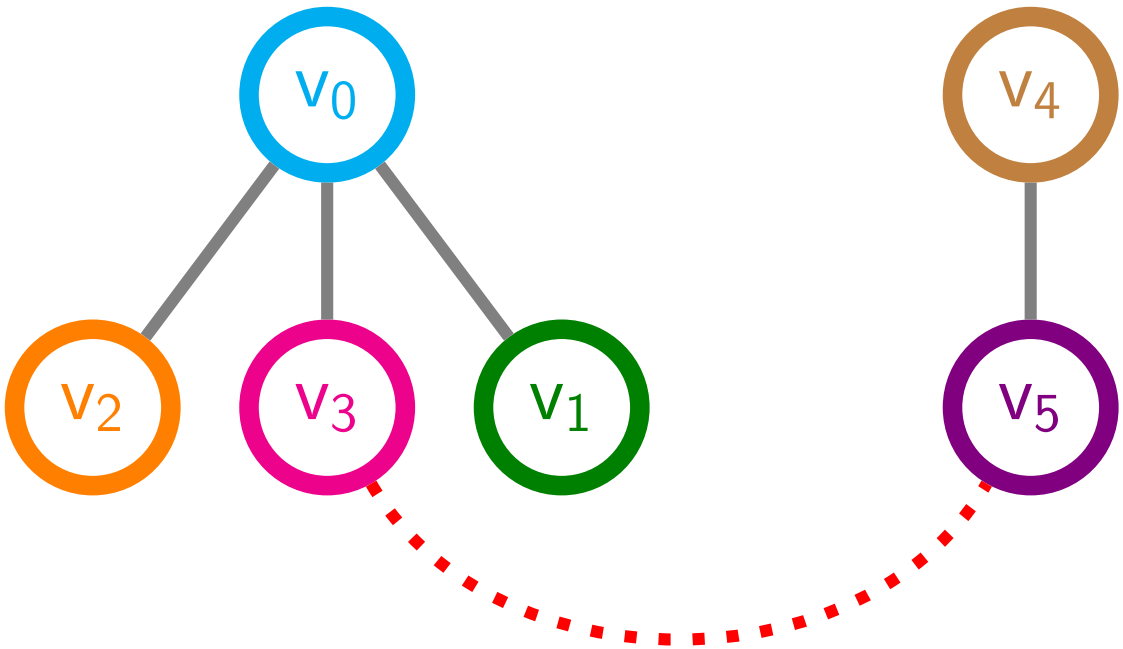


Custo: 8,5

Arestas Ordenadas:

0-2 (1,0)
2-3 (2,0)
2-1 (2,5)
4-5 (3,0)
~~0-3 (3,5)~~
3-5 (4,0)
2-5 (4,5)
1-4 (5,0)
0-1 (6,0)
2-4 (6,0)

Conjuntos Disjuntos e Kruskal

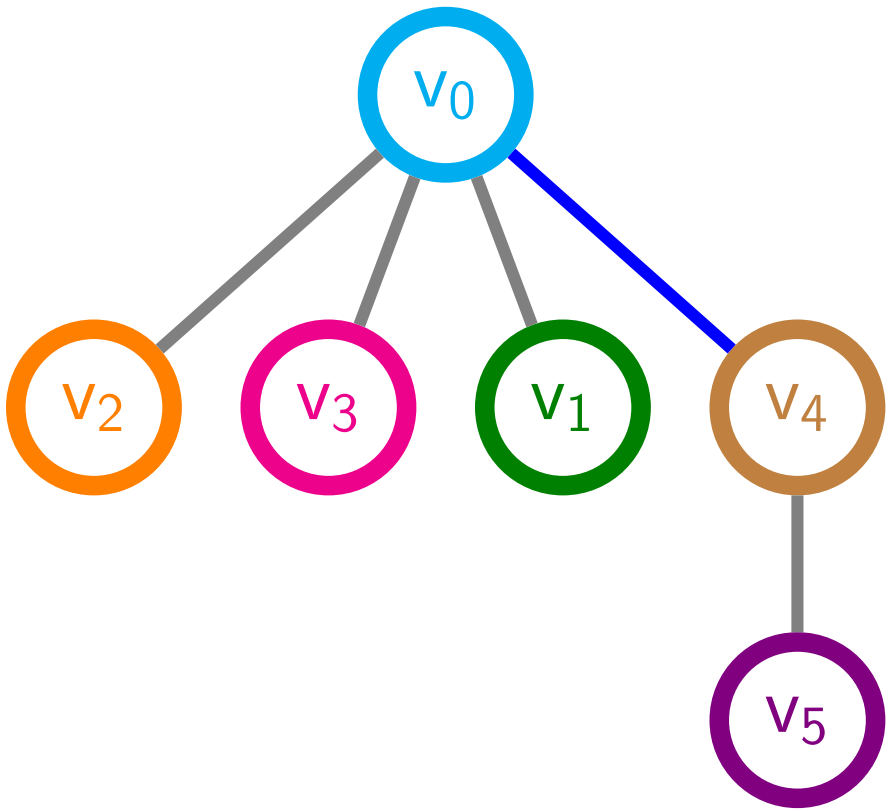


Custo: 8,5

Arestas Ordenadas:

0-2 (1,0)
2-3 (2,0)
2-1 (2,5)
4-5 (3,0)
~~0-3 (3,5)~~
3-5 (4,0)
2-5 (4,5)
1-4 (5,0)
0-1 (6,0)
2-4 (6,0)

Conjuntos Disjuntos e Kruskal

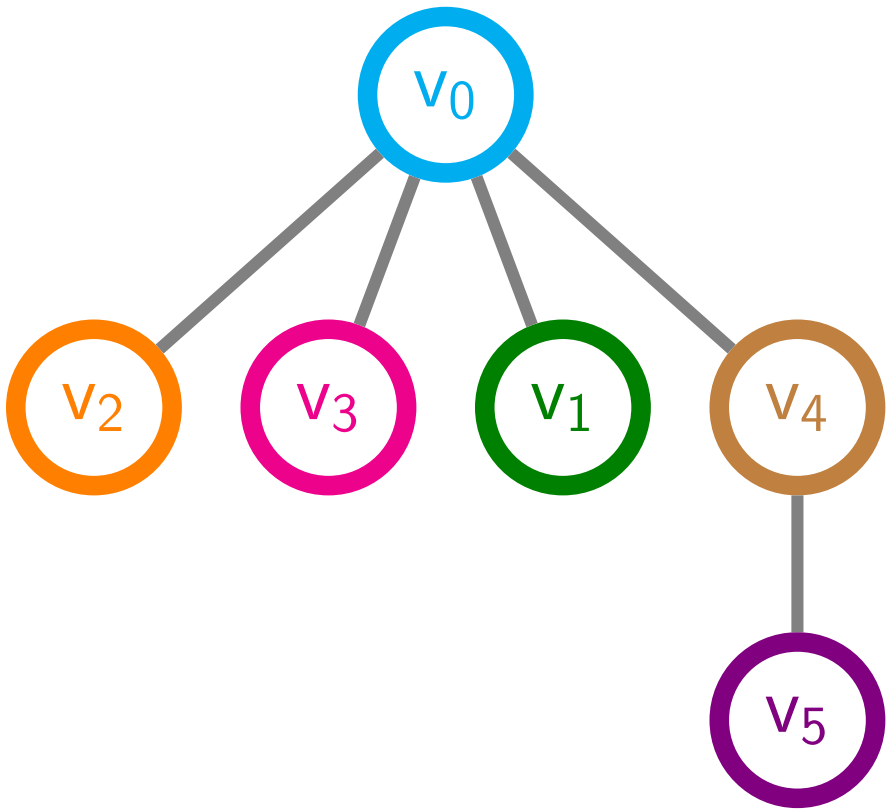


Custo: 12,5

Arestas Ordenadas:

0-2 (1,0)
2-3 (2,0)
2-1 (2,5)
4-5 (3,0)
~~0-3 (3,5)~~
3-5 (4,0)
2-5 (4,5)
1-4 (5,0)
0-1 (6,0)
2-4 (6,0)

Conjuntos Disjuntos e Kruskal

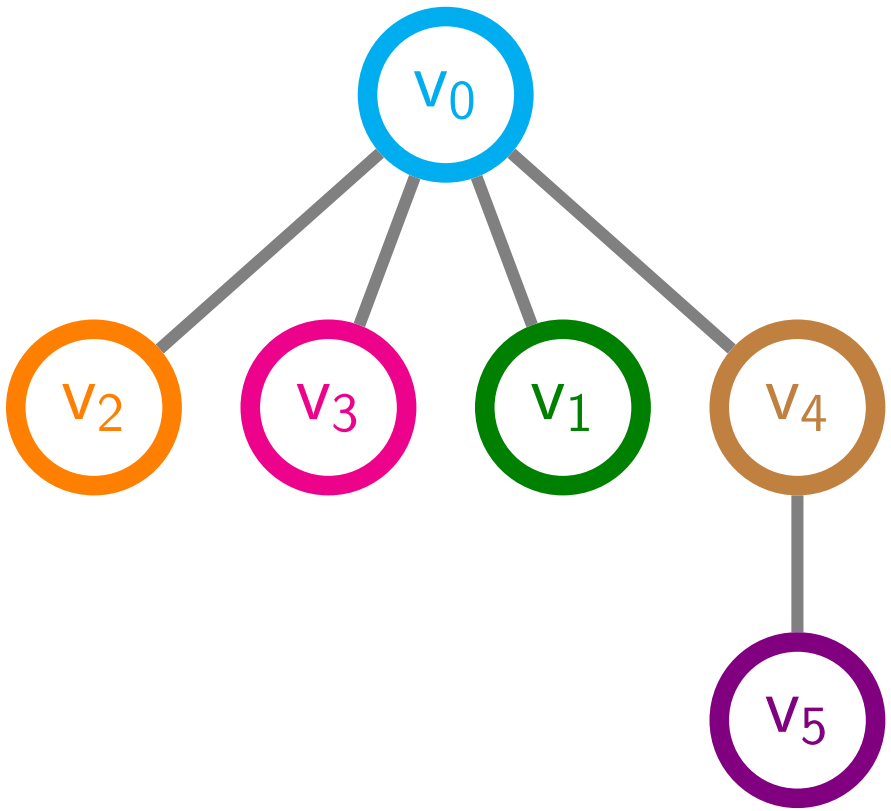


Custo: 12,5

Arestas Ordenadas:

0-2 (1,0)
2-3 (2,0)
2-1 (2,5)
4-5 (3,0)
~~0-3 (3,5)~~
3-5 (4,0)
2-5 (4,5)
1-4 (5,0)
0-1 (6,0)
2-4 (6,0)

Conjuntos Disjuntos e Kruskal



Custo: 12,5

Arestas Ordenadas:

0-2 (1,0)
2-3 (2,0)
2-1 (2,5)
4-5 (3,0)
~~0-3 (3,5)~~
3-5 (4,0)
2-5 (4,5)
1-4 (5,0)
0-1 (6,0)
2-4 (6,0)

Conjuntos Disjuntos e Kruskal

Complexidade	**Conjuntos Disjuntos**
Inicialização	$O(V)$
Identificação do conjunto a que um elemento pertence	$O(log(V))$[a]
União de conjuntos	$O(log(V))$[b]

[a] Para garantir essa complexidade é necessário, durante a união, colocar a raiz da árvore *menor* como filha da raiz da árvore *maior*.

[b] O custo da união está relacionado à identificação dos conjuntos dos dois elementos envolvidos.

Conjuntos Disjuntos

```
typedef struct auxConj{
   int id;
   int altura;
   struct auxConj* pai;
} ElementoC, * PontC;
```

Conjuntos Disjuntos

```
typedef struct auxConj{
   int id;
   int altura;
   struct auxConj* pai;
} ElementoC, * PontC;

typedef struct {
   PontC* elementos;
   int numElementos;
} Conjunto;
```

Conjuntos Disjuntos - Inicialização

```
bool inicializaConjunto(Conjunto* c, int numElementos) {
  if (c==NULL || numElementos<1) return false;
  c->numElementos = numElementos;
  c->elementos = (PontC*)malloc(sizeof(PontC)*numElementos);
  int x;
  PontC novo;
  for (x=0;x<numElementos;x++){
    c->elementos[x] = (PontC)malloc(sizeof(ElementoC));
    c->elementos[x]->id = x;
    c->elementos[x]->altura = 0;
    c->elementos[x]->pai = NULL;
  }
  return true;
}
```

Conjuntos Disjuntos - Inicialização

```
bool inicializaConjunto(Conjunto* c, int numElementos) {
  if (c==NULL || numElementos<1) return false;
  c->numElementos = numElementos;
  c->elementos = (PontC*)malloc(sizeof(PontC)*numElementos);
  int x;
  PontC novo;
  for (x=0;x<numElementos;x++){
    c->elementos[x] = (PontC)malloc(sizeof(ElementoC));
    c->elementos[x]->id = x;
    c->elementos[x]->altura = 0;
    c->elementos[x]->pai = NULL;
  }
  return true;
}
```

$O(V)$

Conjuntos Disjuntos

```
int max(int x, int y){
   if (x>y) return x;
   return y;
}
```

Conjuntos Disjuntos - União

```
bool uniaoDeConjuntos(Conjunto* c, int u, int v){
  if (!c || u<0 || v<0 ||
      u>=c->numElementos || v>=c->numElementos) return false;
  PontC raizU = c->elementos[u];
  PontC raizV = c->elementos[v];
  while(raizU->pai) raizU = raizU->pai;
  while(raizV->pai) raizV = raizV->pai;
  if (raizU == raizV) return false;









}
```

Conjuntos Disjuntos - União

```
bool uniaoDeConjuntos(Conjunto* c, int u, int v){
  if (!c || u<0 || v<0 ||
      u>=c->numElementos || v>=c->numElementos) return false;
  PontC raizU = c->elementos[u];
  PontC raizV = c->elementos[v];
  while(raizU->pai) raizU = raizU->pai;
  while(raizV->pai) raizV = raizV->pai;
  if (raizU == raizV) return false;
   if (raizU->altura>=raizV->altura){
     raizV->pai = raizU;
     raizU->altura = max(raizU->altura, raizV->altura + 1);
   }else{
     raizU->pai = raizV;
   }
  return true;
}
```

Conjuntos Disjuntos - União

```
bool uniaoDeConjuntos(Conjunto* c, int u, int v){
  if (!c || u<0 || v<0 ||
      u>=c->numElementos || v>=c->numElementos) return false;
  PontC raizU = c->elementos[u];
  PontC raizV = c->elementos[v];
  while(raizU->pai) raizU = raizU->pai;
  while(raizV->pai) raizV = raizV->pai;
  if (raizU == raizV) return false;
   if (raizU->altura>=raizV->altura){
     raizV->pai = raizU;
     raizU->altura = max(raizU->altura, raizV->altura + 1);
   }else{
     raizU->pai = raizV;
   }
  return true;
}
```

$O(log(V))$
(altura da árvore)

Conjuntos Disjuntos - *Find*

```
PontC identificaConjunto(Conjunto* c, int u){
  if (!c || u<0 || u>=c->numElementos) return NULL;
  PontC atual = c->elementos[u];
  while(atual->pai) {
    atual = atual->pai;
    c->elementos[u]->pai = atual;
  }
  return atual;
}
```

Conjuntos Disjuntos - *Find*

```
PontC identificaConjunto(Conjunto* c, int u){
  if (!c || u<0 || u>=c->numElementos) return NULL;
  PontC atual = c->elementos[u];
  while(atual->pai) {
    atual = atual->pai;
    c->elementos[u]->pai = atual;
  }
  return atual;
}
```

```
PontC identificaConjuntoRec(Conjunto* c, int u){
  if (!c || u<0 || u>=c->numElementos) return NULL;
  if (!c->elementos[u]->pai) return c->elementos[u];
  if (c->elementos[u]->pai) c->elementos[u]->pai =
            identificaConjuntoRec(c, c->elementos[u]->pai->id);
  return c->elementos[u]->pai;
}
```

Conjuntos Disjuntos - *Find*

```
PontC identificaConjunto(Conjunto* c, int u){
  if (!c || u<0 || u>=c->numElementos) return NULL;
  PontC atual = c->elementos[u];
  while(atual->pai) {
    atual = atual->pai;
    c->elementos[u]->pai = atual;
  }
  return atual;
}
```

$O(log(V))$
(altura da árvore)

```
PontC identificaConjuntoRec(Conjunto* c, int u){
  if (!c || u<0 || u>=c->numElementos) return NULL;
  if (!c->elementos[u]->pai) return c->elementos[u];
  if (c->elementos[u]->pai) c->elementos[u]->pai =
            identificaConjuntoRec(c, c->elementos[u]->pai->id);
  return c->elementos[u]->pai;
}
```

Árvore Geradora Mínima - Kruskal

```
Grafo* mstKruskal(Grafo* g){
```

Árvore Geradora Mínima - Kruskal

```
Grafo* mstKruskal(Grafo* g){
  if (!g || g->numVertices < 1) return NULL;
```

Árvore Geradora Mínima - Kruskal

```
Grafo* mstKruskal(Grafo* g){
  if (!g || g->numVertices < 1) return NULL;
  int n, e, x, ok = 0;
  n = g->numVertices;
  e = g->numArestas;
  Aresta atual;
```

Árvore Geradora Mínima - Kruskal

```
Grafo* mstKruskal(Grafo* g){
  if (!g || g->numVertices < 1) return NULL;
  int n, e, x, ok = 0;
  n = g->numVertices;
  e = g->numArestas;
  Aresta atual;
  Grafo* mst = (Grafo*)malloc(sizeof(Grafo));
  inicializaGrafo(mst, n);
```

Árvore Geradora Mínima - Kruskal

```
Grafo* mstKruskal(Grafo* g){
  if (!g || g->numVertices < 1) return NULL;
  int n, e, x, ok = 0;
  n = g->numVertices;
  e = g->numArestas;
  Aresta atual;
  Grafo* mst = (Grafo*)malloc(sizeof(Grafo));
  inicializaGrafo(mst, n);
  Aresta* arestas = arranjoDeArestas(g);
```

Árvore Geradora Mínima - Kruskal

```
Grafo* mstKruskal(Grafo* g){
  if (!g || g->numVertices < 1) return NULL;
  int n, e, x, ok = 0;
  n = g->numVertices;
  e = g->numArestas;
  Aresta atual;
  Grafo* mst = (Grafo*)malloc(sizeof(Grafo));
  inicializaGrafo(mst, n);
  Aresta* arestas = arranjoDeArestas(g);
  ordenaArestas(arestas, e);
  ...
```

Árvore Geradora Mínima - Kruskal

...

}

Árvore Geradora Mínima - Kruskal

```
...
Conjunto c;
inicializaConjunto(&c, n);
```

```
}
```

Árvore Geradora Mínima - Kruskal

```
...
Conjunto c;
inicializaConjunto(&c, n);
for(x=0;x<e;x++){
  atual = arestas[x];









  }


}
```

Árvore Geradora Mínima - Kruskal

```
...
Conjunto c;
inicializaConjunto(&c, n);
for(x=0;x<e;x++){
  atual = arestas[x];
  if ( identificaConjunto(&c, atual.origem) !=
       identificaConjunto(&c, atual.destino) ){




  }
}


}
```

Árvore Geradora Mínima - Kruskal

```
...
Conjunto c;
inicializaConjunto(&c, n);
for(x=0;x<e;x++){
  atual = arestas[x];
  if (identificaConjunto(&c, atual.origem) !=
      identificaConjunto(&c, atual.destino)){
    uniaoDeConjuntos(&c, atual.origem, atual.destino);



  }
}


}
```

Árvore Geradora Mínima - Kruskal

```
...
Conjunto c;
inicializaConjunto(&c, n);
for(x=0;x<e;x++){
  atual = arestas[x];
  if ( identificaConjunto(&c, atual.origem) !=
       identificaConjunto(&c, atual.destino) ){
    uniaoDeConjuntos(&c, atual.origem, atual.destino);
    insereAresta(mst, atual.origem, atual.destino, atual.peso);


  }
}


}
```

Árvore Geradora Mínima - Kruskal

```
...
Conjunto c;
inicializaConjunto(&c, n);
for(x=0;x<e;x++){
  atual = arestas[x];
  if (identificaConjunto(&c, atual.origem) !=
      identificaConjunto(&c, atual.destino)){
    uniaoDeConjuntos(&c, atual.origem, atual.destino);
    insereAresta(mst, atual.origem, atual.destino, atual.peso);
    ok++;
    if (ok==g->numVertices-1) break;
  }
}

}
```

Árvore Geradora Mínima - Kruskal

```
...
Conjunto c;
inicializaConjunto(&c, n);
for(x=0;x<e;x++){
  atual = arestas[x];
  if (identificaConjunto(&c, atual.origem) !=
      identificaConjunto(&c, atual.destino)){
    uniaoDeConjuntos(&c, atual.origem, atual.destino);
    insereAresta(mst, atual.origem, atual.destino, atual.peso);
    ok++;
    if (ok==g->numVertices-1) break;
  }
}
liberaConjunto(&c);
return mst;
}
```

Árvore Geradora Mínima - Kruskal

```
...
Conjunto c;
inicializaConjunto(&c, n);
for(x=0;x<e;x++){
  atual = arestas[x];
  if (identificaConjunto(&c, atual.origem) !=
      identificaConjunto(&c, atual.destino)){
    uniaoDeConjuntos(&c, atual.origem, atual.destino);
    insereAresta(mst, atual.origem, atual.destino, atual.peso);
    ok++;
    if (ok==g->numVertices-1) break;
  }
}
liberaConjunto(&c);
return mst;
}
```

$O(Elog(E))$

ou $O(Elog(V))$

(arestas já ordenadas)

Algoritmos e Estruturas de Dados II
Aula 11 – Árvores Geradoras de Custo Mínimo - Algoritmo de Kruskal

Prof. Luciano A. Digiampietri
digiampietri@usp.br
@digiampietri