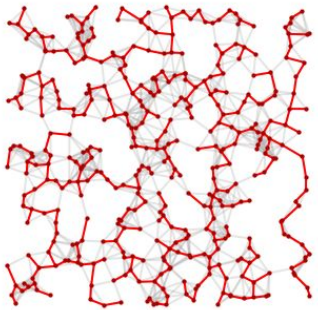
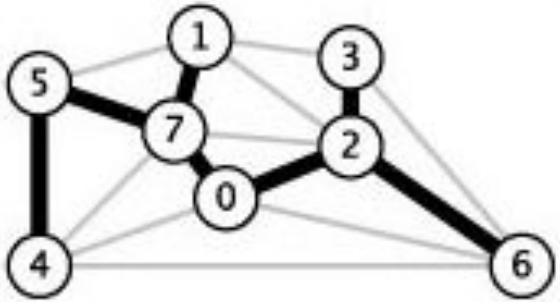
Árvore Geradora Mínima (Minimum Spanning Tree)

SCC 503 - Alg. Estrut. Dados II

MST - definição

- Uma subárvore de um grafo não-dirigido G é qualquer subgrafo de G que seja uma árvore (não radicada). É comum suprimir o sub e dizer que T é uma árvore de G.
- Uma árvore de um grafo não-dirigido G é geradora (= *spanning*) se contém todos os vértices de G. (Quem sabe árvore abrangente seria um





Como toda árvore é conexa, então uma árvore geradora é conexa !

Árvore Geradora - Aplicações

- Por que se haveria de eliminar arestas de um grafo de forma a criar uma árvore geradora:
  - redes elétricas, dados, etc: queremos atingir todas as casas e esta rede precisa estar conectada. Uma forma de evitar cabos duplicados seria calcular a árvore geradora
  - A cidade é enlameada e precisa de uma rede asfaltada mínima de forma que o único ônibus da cidade possa trafegar nela, atingindo os pontos vitais da cidade !
  - segmentação de imagens
  - extração de características em imagens
  -

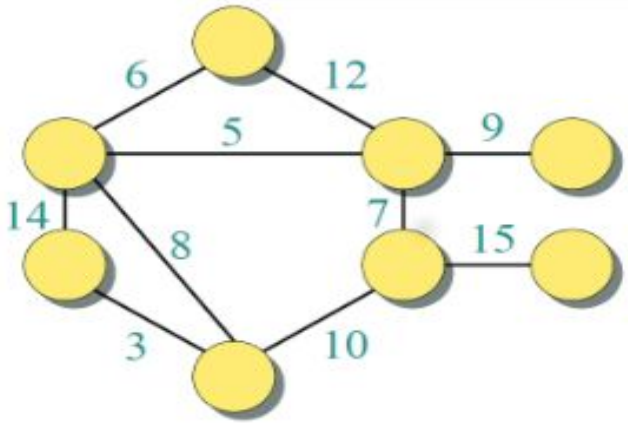
Árvore Geradora de Custo Mínimo (MST)

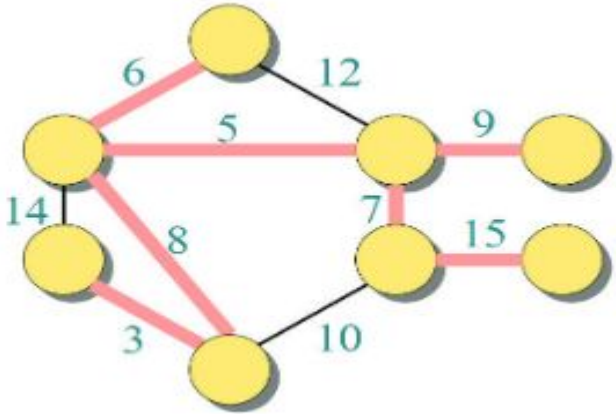
- Seja G um grafo não-dirigido com custos nas arestas. Os custos podem ser positivos ou negativos. O custo de um subgrafo não-dirigido T de G é a soma dos custos das arestas de T.

- Uma árvore geradora mínima de G é qualquer árvore geradora de G que tenha custo mínimo. Em outras palavras, uma árvore geradora T de G é mínima se nenhuma outra árvore geradora tem custo estritamente menor que o de T. Árvores geradoras mínimas também são conhecidas pela abreviatura MST de *minimum spanning tree*.

- Então, dado um grafo não dirigido, com pesos, como podemos calcular a MST ?
  - Temos dois algoritmos bem conhecidos: PRIM e KRUSKAL.

MST - exemplo





Kruskal (Joseph Kruskal em 1956)

- Este algoritmo utiliza o conceito de Floresta F.
  - Um floresta geradora de um grafo G, é aquela que possui todos os vértices de G e ela não tem ciclos.
  - Uma aresta a é externa a F, se ela não pertence a F e se adicionarmos a à F (F+a), F permanece uma floresta, ou seja, CONTINUA SEM CICLOS !!!!
  - Podemos portanto generalizar o algoritmo da seguinte forma

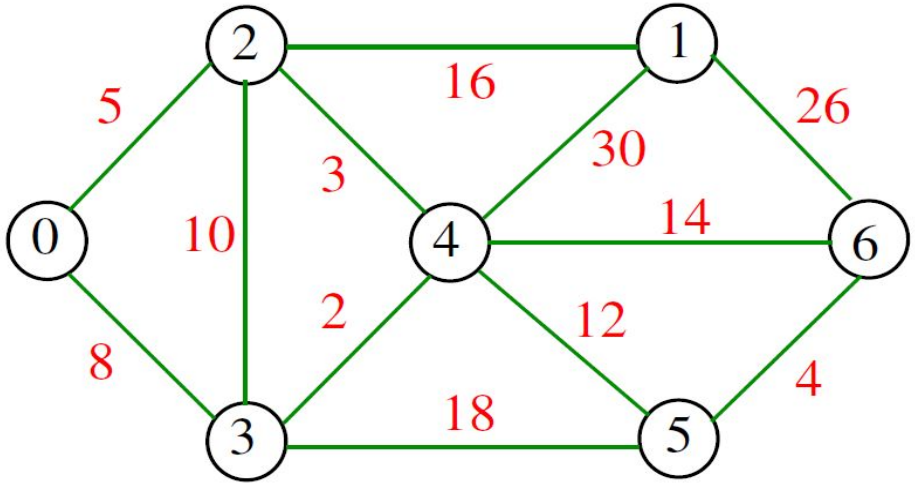
Kruskal

enquanto existe aresta externa a F **faça**

seja a uma aresta externa de custo mínimo

acrescente a a F

devolva T



floresta	custo
	0.0
3-4	0.2
3-4 2-4	0.5
3-4 2-4 5-6	0.9
3-4 2-4 5-6 0-2	1.4
3-4 2-4 5-6 0-2 4-5	2.6
3-4 2-4 5-6 0-2 4-5 2-1	4.2

Kruskal

- Veja que a questão central é não formar ciclos, à medida que arestas à são adicionadas ao grafo, certo ??
- Como podemos fazer isso??
- Pense na Floresta F composta de tantas componentes conexas quanto o nro de vértices (V) do grafo
    - inicialmente, temos V conjuntos (ou componentes conexas)
    - ao pegarmos gulosamente a aresta de menor custo, digamos (2-3) então UNIMOS os conjuntos 2 e 3.
    - Continue pegando arestas a (u e v) de forma que u e v sempre pertençam a componentes conexas diferentes !!!
- Tudo isso pode ser feito com a noção de conjuntos !!! captou ??

Union-Find Disjoint Set

- Uma estrutura simples para manipular conjuntos
  - Unir conjuntos
  - Encontrar elementos e verificar se estão em conjuntos separados

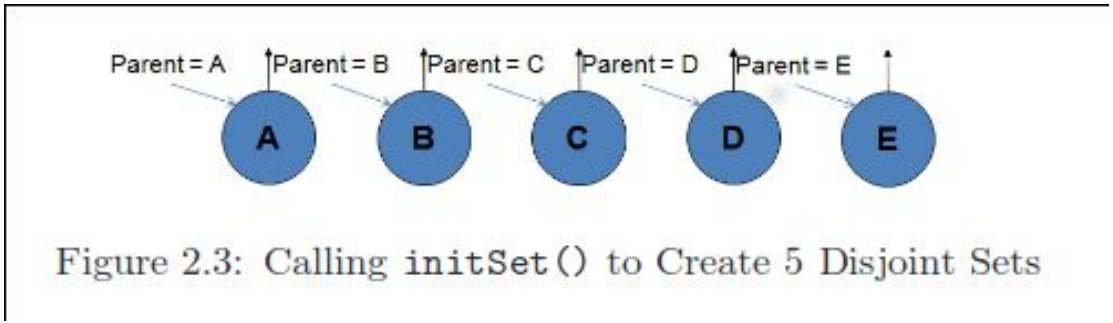
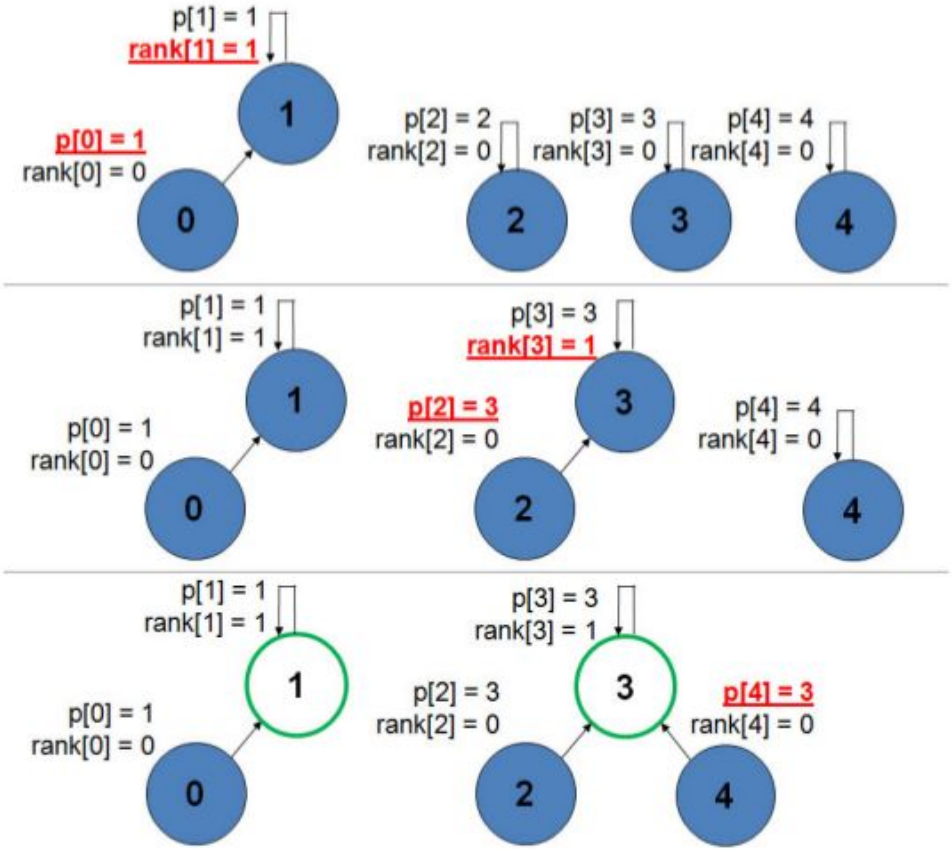


Figure 2.3: Calling `initSet()` to Create 5 Disjoint Sets

Union-Find Disjoint Set



Union-Find Disjoint Set

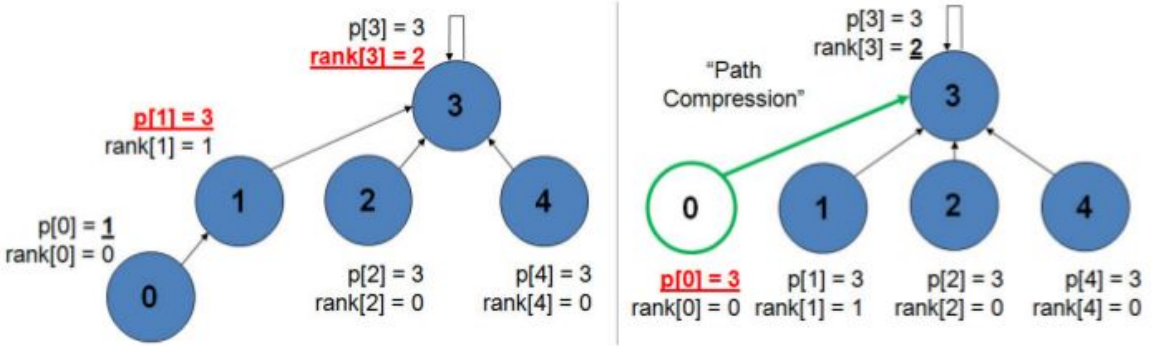
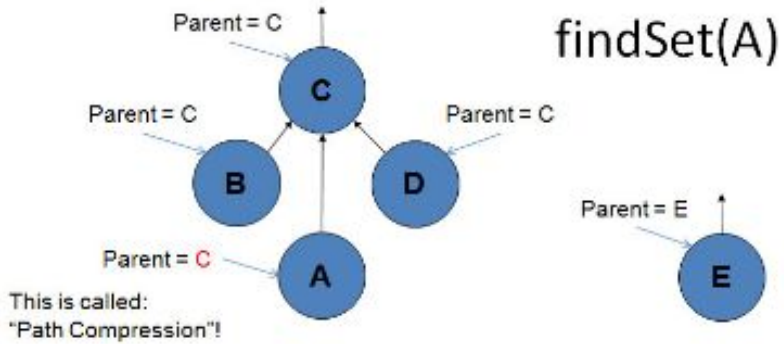


Figure 2.7: `unionSet(0, 3)` → `findSet(0)`



```
class UnionFind {
    private:
        typedef vector<int> vi;
        vi p;
        vi rank;

    public:

        UnionFind(int N){
            rank.assign (N, 0);  p.assign(N,0);
            for(int i =0; i<N; i++)
                p[i] = i;
        }

        int findSet(int i){
            if (p[i] == i)
                return i;
            return (findSet(p[i]));
        }
```

vi p; // p[i] amazena o pai do item i
vi rank; // rank[i] armazena a altura da árvora na qual i é a raiz

inicialmente rank[i] é igual a zero e p[i] = i, ou seja, todos vértices representam conjuntos disjuntos!

```
bool isSameSet(int i, int j){
    return (findSet(i) == findSet(j));
}

void unionSet(int i, int j){
    if (!isSameSet(i,j)){
        int x = findSet(i);
        int y = findSet(j);

        if (rank[x] > rank[y])
            p[y] = x;
        else {
            p[x] = y;
            if (rank[x] == rank[y])
                rank[y] = rank[y]+1;
        }
    }

}
```

unionSet une dois conjuntos disjuntos

- Para manter a árvore o menos alta possível, o vértice de maior ranking será o pai daquele de menor ranking !

Kruskal

- Pronto..
- Agora é so escrever o algoritmo. Seja um grafo de V vertices.

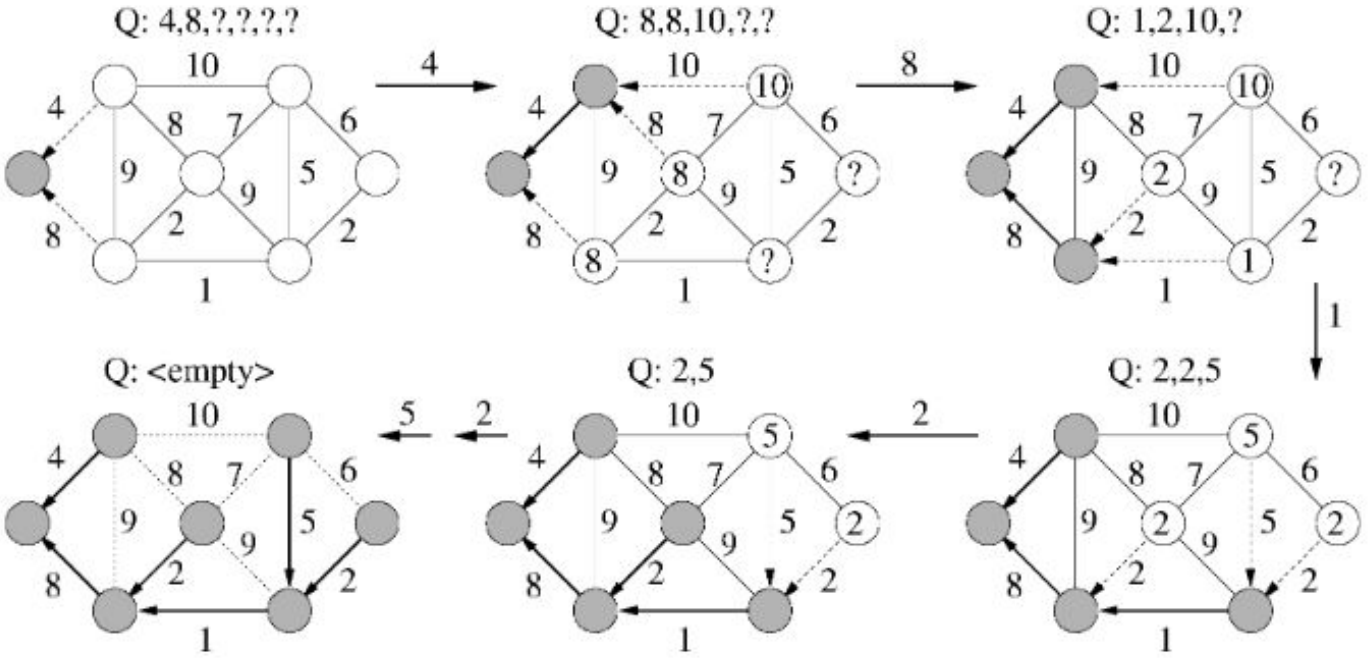
1. coloque as arestas numa lista em ordem não decrescente.
   a. esta lista é do tipo vector<pair<peso, ii> > list, onde ii é a aresta v-w
2. inicie o Union-Find >>> UnioFind uf(V);

para todas as arestas i em A
    el = list[i]; // retira o primeiro elemento da lista….
    se (v e w em el NAO estiverem no mesmo conjunto)
        MST_cost += custo armazenado em el;
        uf.UnionSet (v, w em el);

PRIM (Robert C. Prim em 1957)

- Use uma lista de prioridade composta de pares (peso, w) armazenados em ordem crescente. Esta informação vem da aresta (v, w) analisada
- Gulosamente, selecione o par (peso, w). Se o vértice w já estiver visitado, não continue.

Prim



Prim

1. Arbitrariamente escolha um vértice source s.
2. Visitado[s] = true;
3. crie uma funcao addQueue (s) que adiciona os adjacentes w e pesos na fila de prioridade (peso, w)

```
Enquanto a fila nao estiver vazia
        dado = fila.pop;
        se ( nao visitado dado.second)
                custo += dado.first;
                addQueue(dado.second);
```