Grafos - parte 2

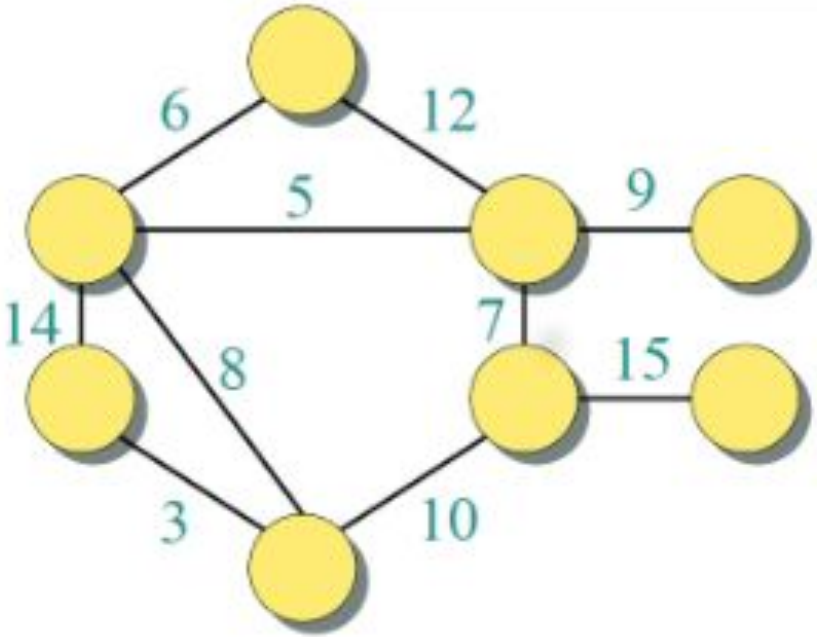
SCC5900 - Projeto de Algoritmos

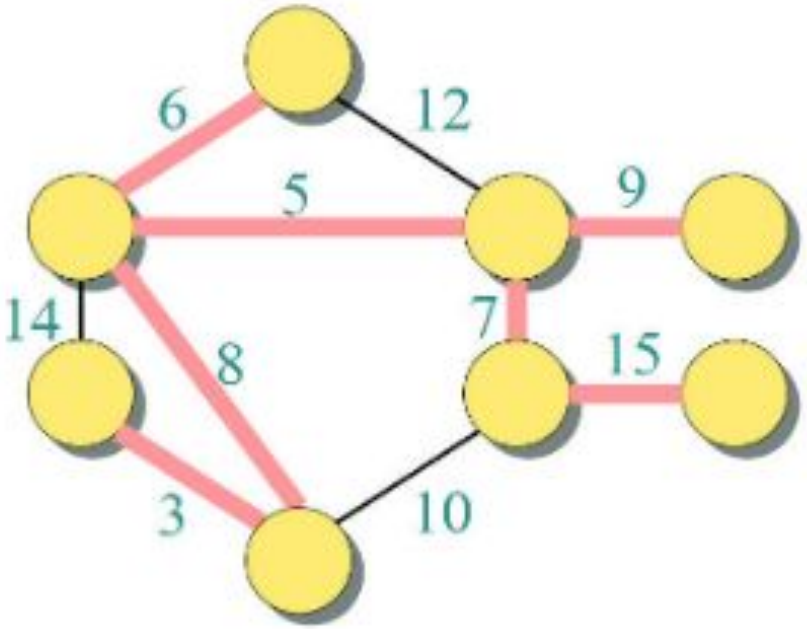
João Batista

Grafos

- Nesta aula veremos os seguintes conceitos
  - Minimum Spanning Tree (árvore geradora mínima)
    - Kruskal
    - Prim
  - Single-Source Shortest Path
    - Caminho mínimo a partir de um determinado vértice
      - Dijkstra
      - Bellman-Ford
  - All-Pair Shortest Path
    - Floyd Warshall
- Atentem também para o uso de estruturas de dados importantes

MST – Minimum Spanning Tree





MST – Minimum Spanning Tree

- Ha dois algoritmos bem conhecidos para estes problemas
  - Kruskal
  - Prim
- Kruskal
  - Ordene as arestas em ordem NAO decrescente e armazene-as em uma adjList
  - Gulosamente, adicione estas arestas na árvore geradora, MAS com o cuidado de não formar ciclos
  - CICLOS: Estrutura UNIONFIND !!!!
- Prim
  - Use uma lista de prioridade em que arestas são armazenadas em ordem crescente de Peso da aresta, seguido pelo nro do nó (em caso de empate)
  - Gulosamente, selecione o par (w,u). Para prevenir ciclos, ignore o vertice caso este já tenha sido visitado!

Union-Find Disjoint Set

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

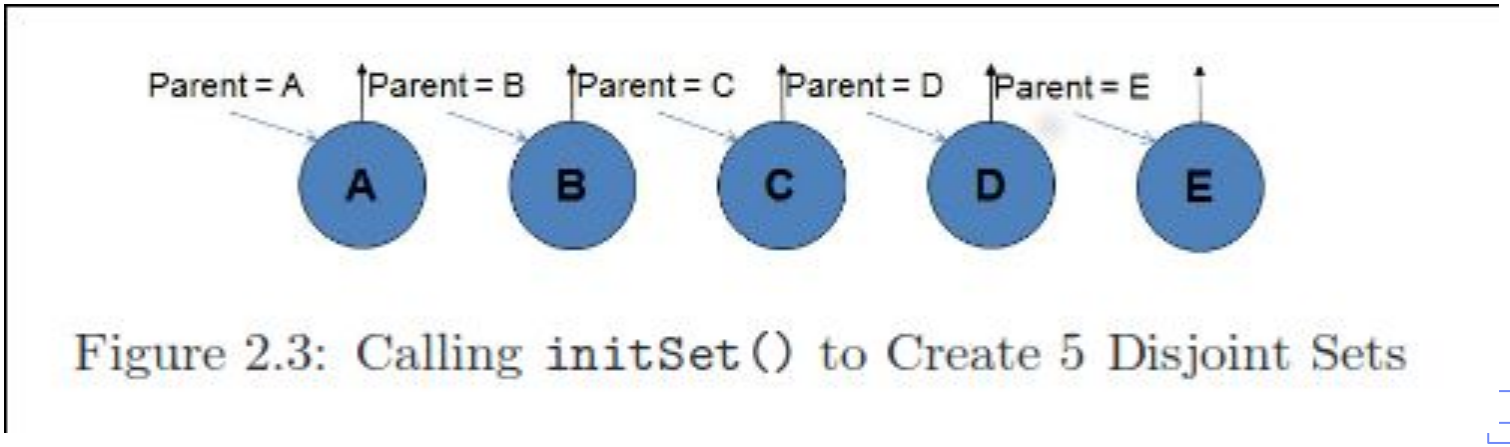


Figure 2.3: Calling `initSet()` to Create 5 Disjoint Sets

Union-Find Disjoint Set

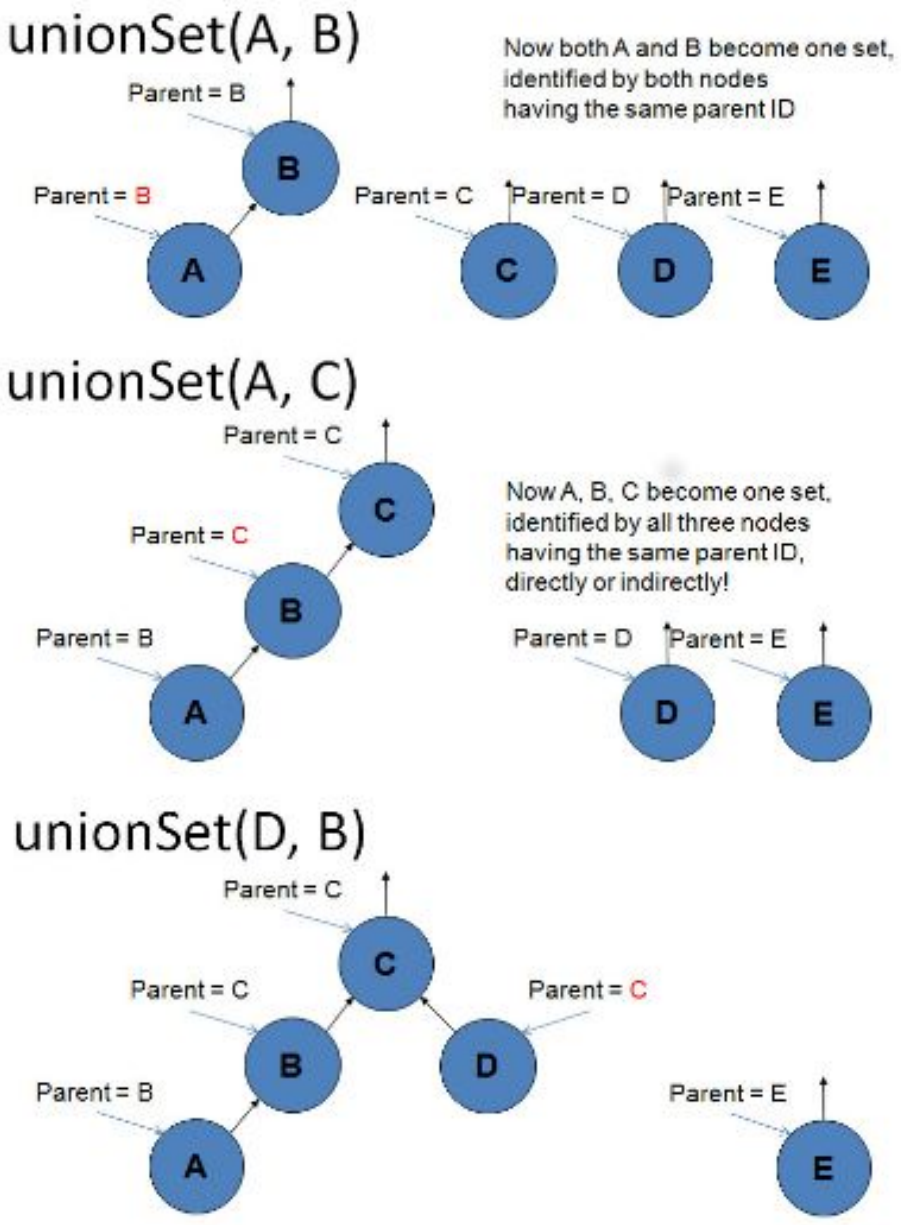
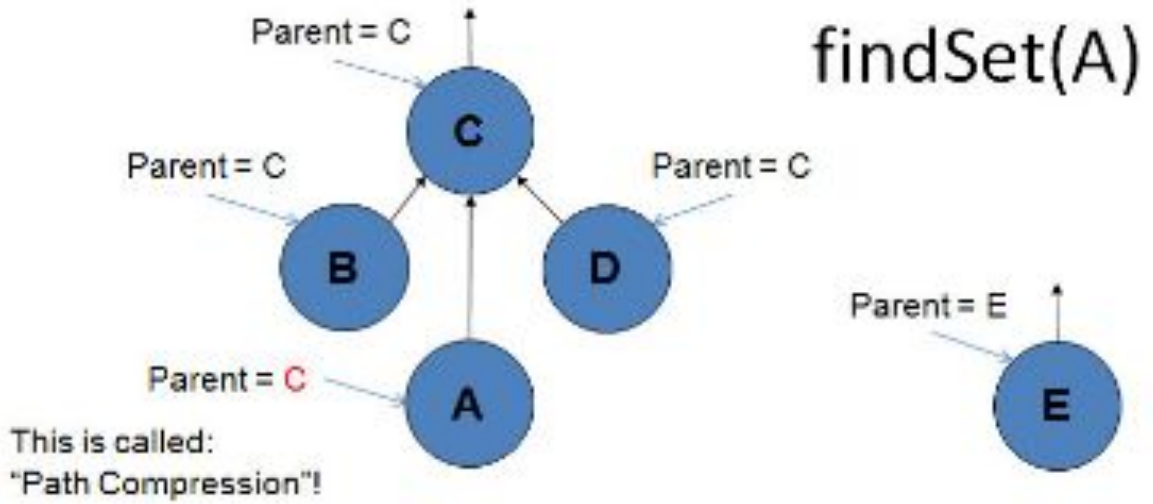


Figure 2.4: Calling `unionSet(i, j)` to Union Disjoint Sets

Union-Find Disjoint Set



```
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]));
        }
```

```
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;
        }
    }

}
```

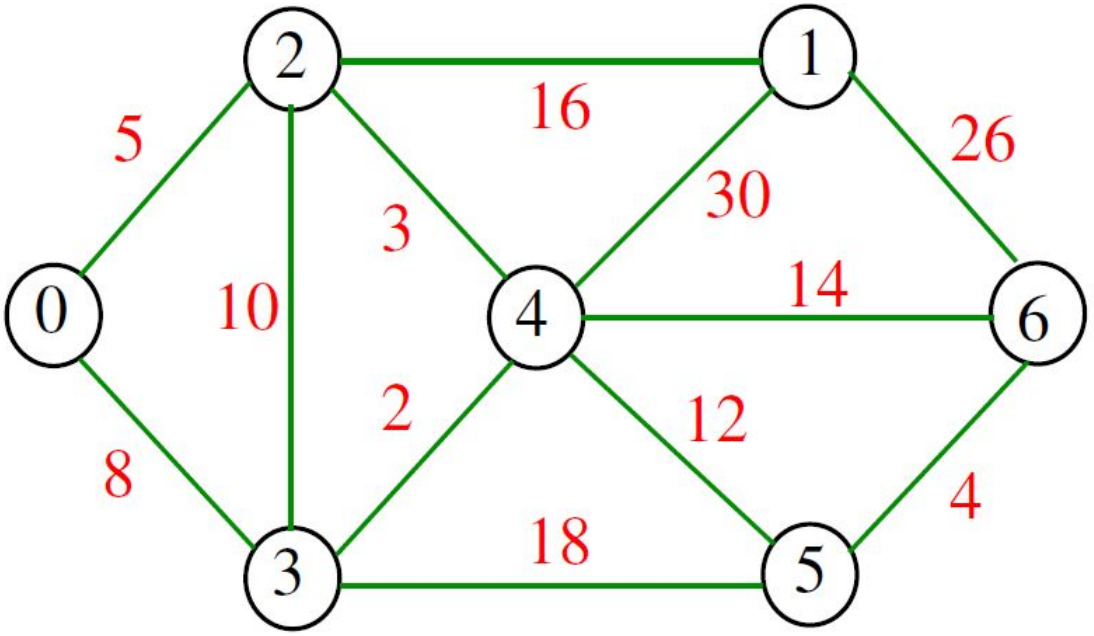
Kruskal

enquanto existe aresta externa a F **faça**

seja a uma aresta externa de custo mínimo

acrescente a a F

devolva T

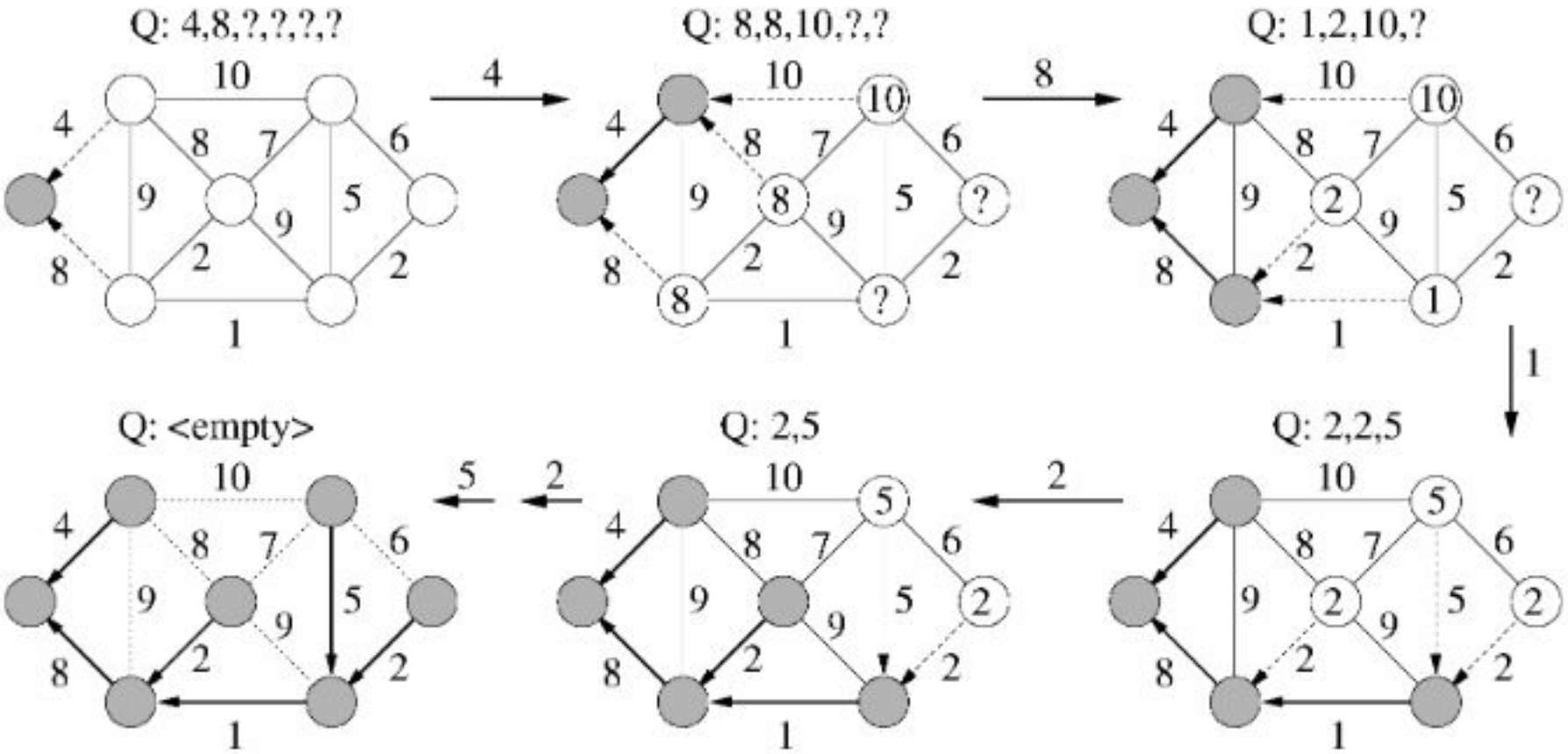


floresta	custo
	0.0
3-4	0.2
3-4 2-4	0.5
3-4 2-4 5-6	0.9
3-4 2-4 5-6 0-2	1.4
3-4 2-4 5-6 0-2 4-5	2.6
3-4 2-4 5-6 0-2 4-5 2-1	4.2

Kruskal

```
for (int i=0; i<A; i++) {
    scanf("%d %d %d", &u, &v, &w);
    EdgeList.push_back(make_pair(w, ii(u,v)));
}
sort(EdgeList.begin(), EdgeList.end());

int mst_cost = 0;

UnionFind uf(V); // inicialmente todas as arestas sao conjuntos disjuntos...

// percorra todas as arestas...
for(int i=0; i<A; i++){
    pair<int, ii> front = EdgeList[i]; // pega a aresta de menor peso!
    // se os vertices que compoem a aresta estao em conjuntos separados
    // nao ha ciclo, junta os dois !
    if (!uf.isSameSet(front.second.first, front.second.second)) {
        mst_cost += front.first;  // atualiza o custo
        // coloca ambos no mesmo set...
        uf.unionSet(front.second.first, front.second.second);
    }
}
```

Prim



Prim

```
// lembre-se, o grafo nao eh direcionado!
for (int i=0; i<A; i++) {
    scanf("%d %d %d", &u, &v, &w);
    adjList[u].push_back(make_pair(v,w));
    adjList[v].push_back(make_pair(u,w));
}

taken.assign(V,0);  // nenhum vertice foi visitado ainda..
process(0); // comeca pelo vertice V=0. ISso é arbitrario....

int mst_cost = 0;

while(!pq.empty()){  // enquanto houver elementos na fila...
    //print_queue(pq);
    ii front = pq.top(); pq.pop(); // pega primeiro valor da fila: tem o menor peso dentre os adjs...
    // lembre-se de que o vertice u foi armazenado como segundo, pra priorizar a busca pelo peso..
    //std::cout << "(" << front.first << "," << front.second <<")" << " ** ";
    u = -front.second;
    w = -front.first;
    //std::cout << "(" << u << "," << front.second <<")" << " ** ";
    if (!taken[u]) { // este vertice nao foi conectado ainda na MST ...
        mst_cost += w;
        process(u);   // popula a fila com os adjacentes de u....
    }
}
```

Caminho mínimo

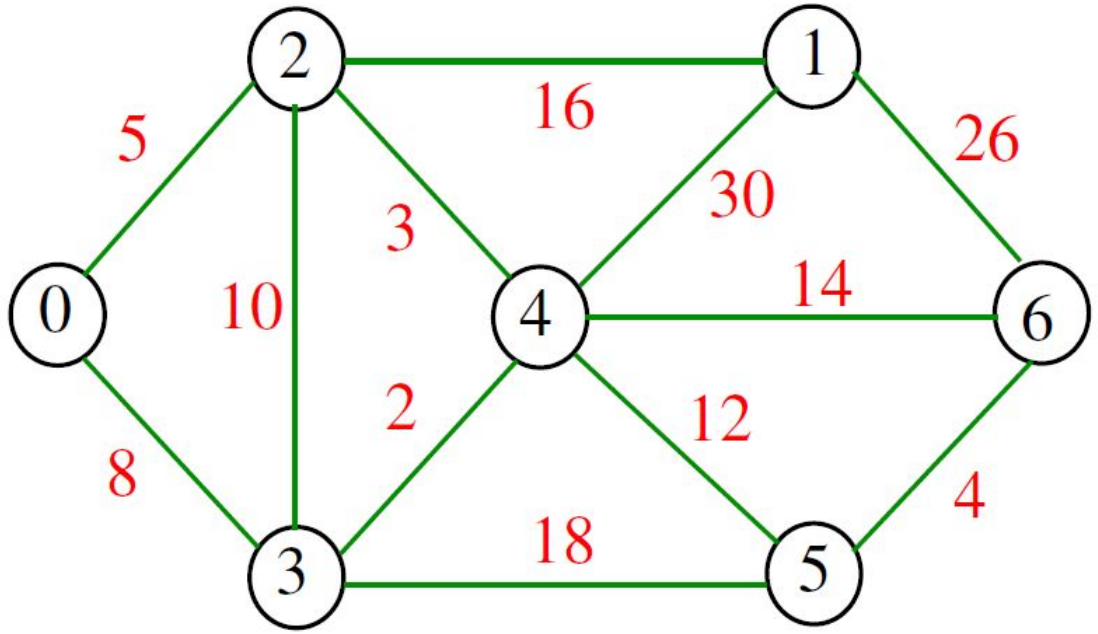
- Se grafo não for ponderado??

```
// breadth search... utilizar fila....
// inicia com s na fila...
queue<int> q; q.push(s);


while(!q.empty()){  // enquanto houver elementos na fila...
    int u = q.front(); q.pop();

    // para todos os adjacentes de u
    for(int j=0; j<adjList[u].size(); j++){
        ii v = adjList[u][j];
        // o vertice nao foi visitado ainda
        if (d[v.first] == INF){
            d[v.first] = d[u]+1; //vai acumulando a distancia..
            parent[v.first] = u;
            q.push(v.first);
        }
    }
}
```

Grafo ponderado arestas não negativas - Dijkstra

```
// a fila comecao com o elemento source s e a distancia claro eh zero...
// armazenaremos (distancia d, vertice u) nesta ordem...
priority_queue<ii, vector<ii>, greater<ii> > pq;
pq.push(ii(0,s));


while(!pq.empty()){  // enquanto houver elementos na fila...

    ii front = pq.top(); pq.pop();
    int d = front.first;  u = front.second;

    // atencao.. faca um teste no papel...este algoritmo permite que valore distintos
    // de distancias para um mesmo mesmo vertice u seja armazenada nela.. essa
    // verificacao abaixo faz com que uma distancia maior seja ignorada...
    if (d > dist[u]) continue;

    // para todos os adjacentes de u
    for(int j=0; j<adjList[u].size(); j++){
        ii v = adjList[u][j];
        if (dist[u] + v.second < dist[v.first]){
            dist[v.first] = dist[u]+ v.second; //vai acumulando a distancia..
            parent[v.first] = u;
            pq.push(ii(dist[v.first], v.first));
        }
    }
}
```

Alg. Dijkstra apresentado



Construa a fila de prioridade e teste o algoritmo....

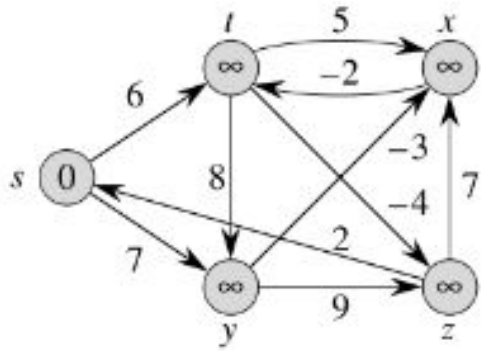
SSSP em grafos com arestas negativas

- O alg. de Dijkstra apresentado funciona para arestas negativas??
- O que acontece se houver ciclos negativos?
- Há duas soluções:
  - Uma que computaca o caminho a partir de um vértice origem s
    - Bellman-Ford
  - Outra que computa a All-pair shortest Path
    - Floyd Warshall

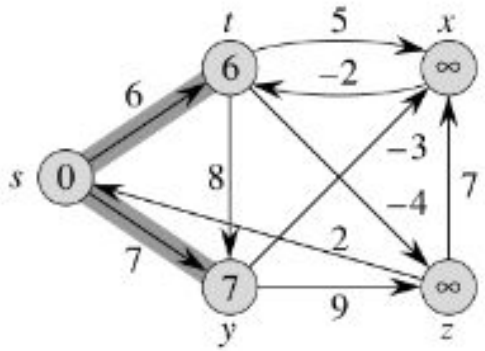
Bellman-Ford

- Percorra o conjunto de arestas V-1 vezes
- Para cada vez, tente relaxas todas as arestas
- Qual a complexidade??
- É mais lento que Dijkstra..
- E se houver ciclo negativo?? existe solução de caminho mínimo???

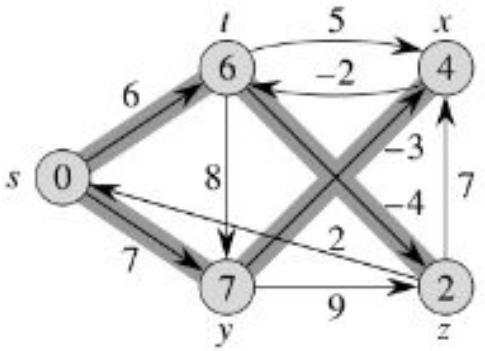
Bellman-Ford



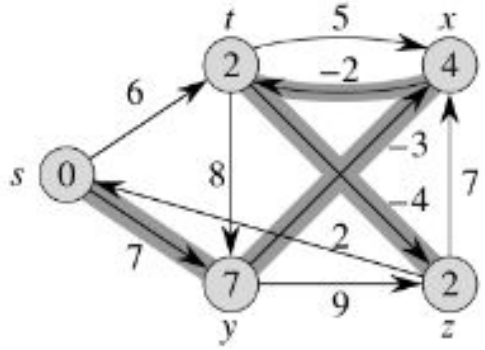
(a)



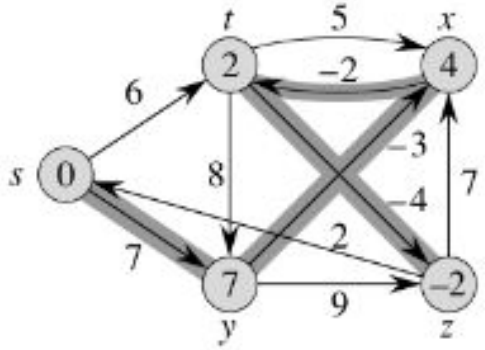
(b)



(c)



(d)



(e)

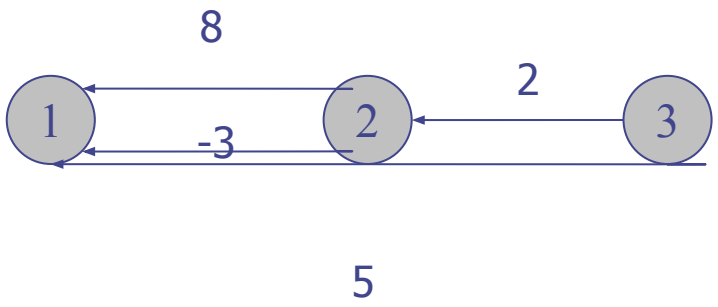
Bellman-Ford

```
for (int i=0; i<A; i++) {
    scanf("%d %d %d", &u, &v, &w);
    adjList[u].push_back(make_pair(v,w));
}

// o vetor de distancia a partir de um vertice quqlquer tem inicialmente distancia INFINITA
vi dist(V, INF);
vi parent(V);   // grava a trilha, guardando o pai  de cada vertice..

dist[s] = 0;  // a distancia de s para s eh zero !!!


// percorra a lista de adjacencia V-1 vezes...
for (int i=0; i< V-1; i++)
    // agora visite todas as arestas...
    for (int u=0; u< V; u++)
        // para todos os adjacentes de u
        for(int j=0; j<adjList[u].size(); j++){
            ii v = adjList[u][j];
            if (dist[u] + v.second < dist[v.first]){
                dist[v.first] = dist[u]+ v.second; //vai acumulando a distancia..
                parent[v.first] = u;
            }
        }
```

Floyd Warshall

- Análogo ao Bellman-Ford, mas calcula o caminho mínimo para todos os pares de vértices..
- Utiliza Matriz de adjacência, por eficiência
- Complexidade alta: V^3

Floyd-Warshall

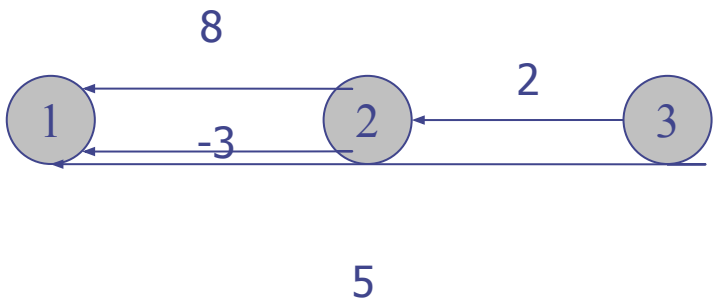


	1	2	3
1			
2			
3			

A

- Inicialmente os custos entre vértices adjacentes são inseridos na tabela *A*
- Pesos de *self-loops* não são considerados

Floyd-Warshall

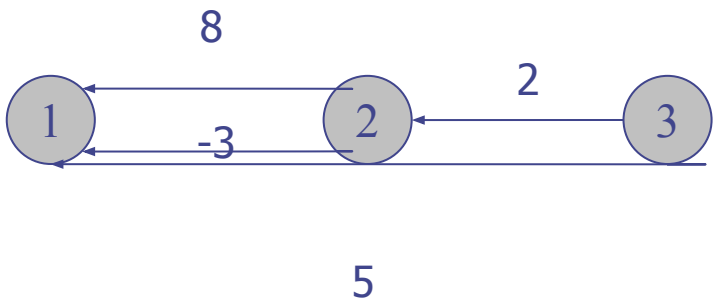


	1	2	3
1	0	8	5
2	-3	0	∞
3	∞	2	0

A

- Inicialmente os custos entre vértices adjacentes são inseridos na tabela *A*
- Pesos de *self-loops* não são considerados

Floyd-Warshall

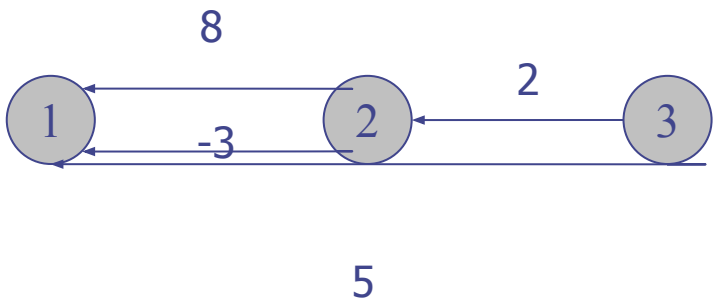


	1	2	3
1	0	8	5
2	-3	0	∞
3	∞	2	0

A

- A matriz *A* é percorrida *|V|* vezes
- A cada iteração *k*, verifica-se se um caminho entre dos vértices *(v, w)* que passa também pelo vértice *k* é mais curto do que o caminho mais curto conhecido

Floyd-Warshall



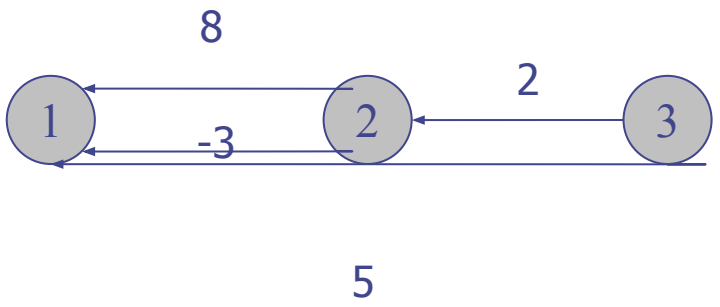
Ou seja:

$$A[v, w] = \min(A[v, w], A[v, k] + A[k, w])$$

	1	2	3
1	0	8	5
2	-3	0	∞
3	∞	2	0

A

Floyd-Warshall



Ou seja:

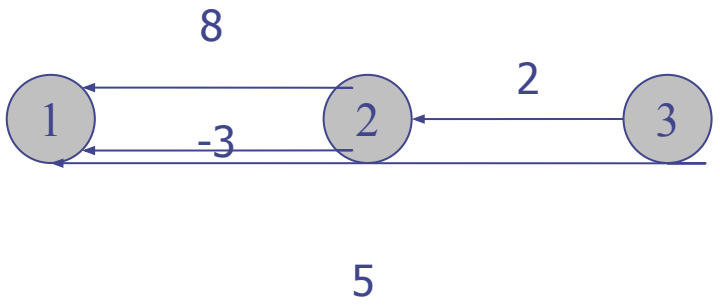
$A[1, 1] = \min(A[1, 1], A[1, 1] + A[1, 1])$

	1	2	3
1	0	8	5
2	-3	0	∞
3	∞	2	0

A

$k = 1$

Floyd-Warshall



Ou seja:

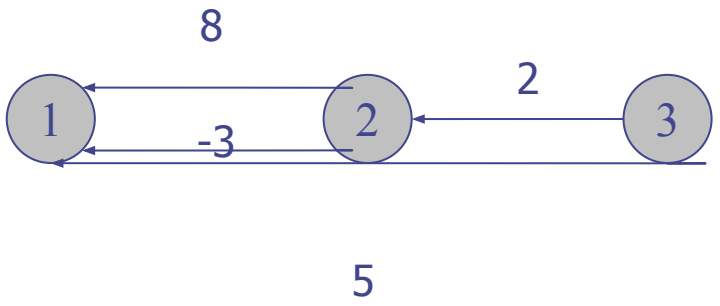
$A[1, 2] = \min(A[1, 2], A[1, 1] + A[1, 2])$

	1	2	3
1	0	8	5
2	-3	0	∞
3	∞	2	0

A

$k = 1$

Floyd-Warshall



Ou seja:

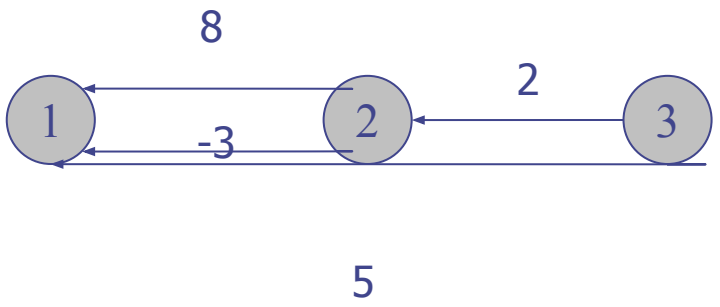
$A[1, 3] = \min(A[1, 3], A[1, 1] + A[1, 3])$

	1	2	3
1	0	8	5
2	-3	0	∞
3	∞	2	0

A

$k = 1$

Floyd-Warshall



Ou seja:

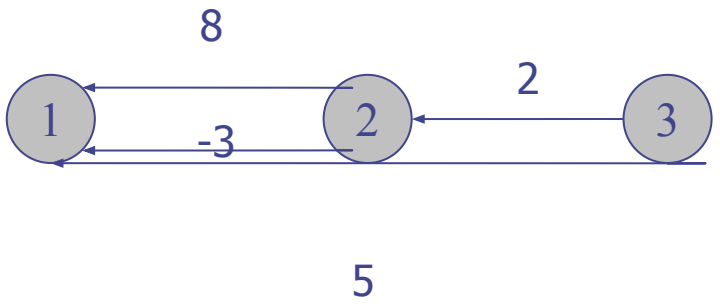
$A[2, 1] = \min(A[2, 1], A[2, 1] + A[1, 1])$

	1	2	3
1	0	8	5
2	-3	0	∞
3	∞	2	0

A

$k = 1$

Floyd-Warshall



Ou seja:

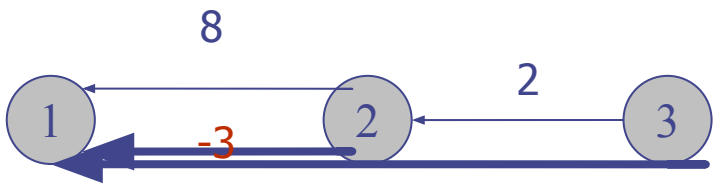
$A[2, 2] = \min(A[2, 2], A[2, 1] + A[1, 2])$

	1	2	3
1	0	8	5
2	-3	0	∞
3	∞	2	0

A

$k = 1$

Floyd-Warshall



5

Ou seja:

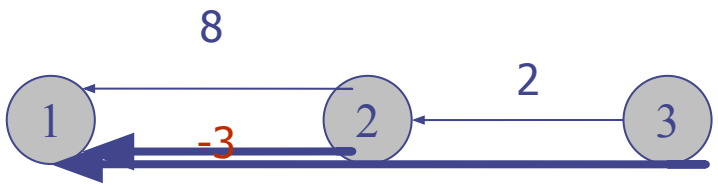
$A[2, 3] = \min(A[2, 3], A[2, 1] + A[1, 3])$

	1	2	3
1	0	8	5
2	-3	0	∞
3	∞	2	0

A

$k = 1$

Floyd-Warshall



5

Ou seja:

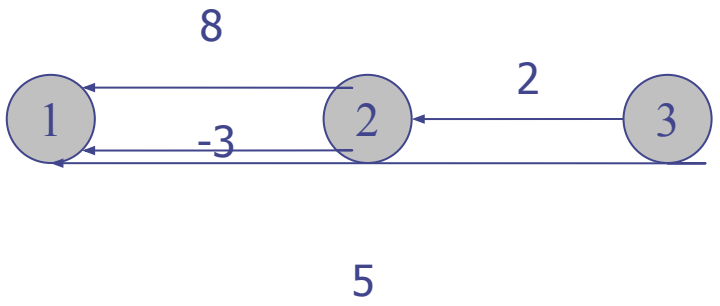
$A[2, 3] = \min(A[2, 3], A[2, 1] + A[1, 3])$

	1	2	3
1	0	8	5
2	-3	0	2
3	∞	2	0

A

$k = 1$

Floyd-Warshall



Ou seja:

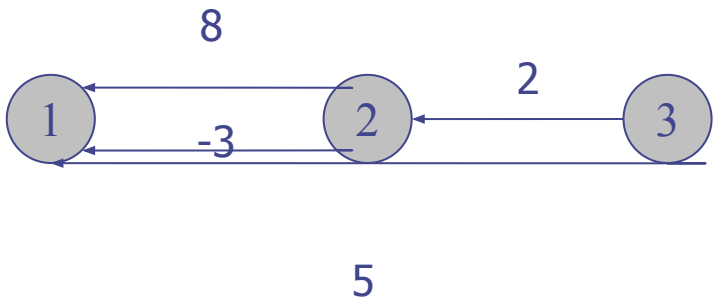
$A[3, 1] = \min(A[3, 1], A[3, 1] + A[1, 3])$

	1	2	3
1	0	8	5
2	-3	0	2
3	∞	2	0

A

$k = 1$

Floyd-Warshall



Ou seja:

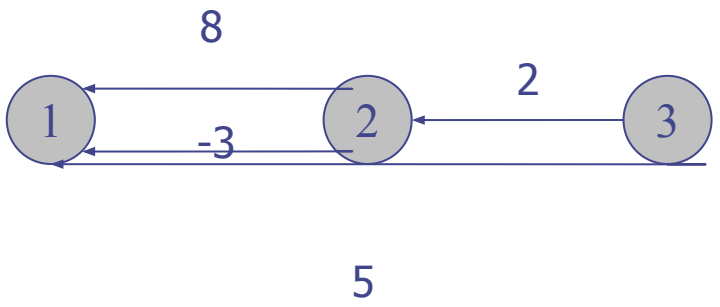
$A[3, 2] = \min(A[3, 2], A[3, 1] + A[1, 2])$

	1	2	3
1	0	8	5
2	-3	0	2
3	∞	2	0

A

$k = 1$

Floyd-Warshall



Ou seja:

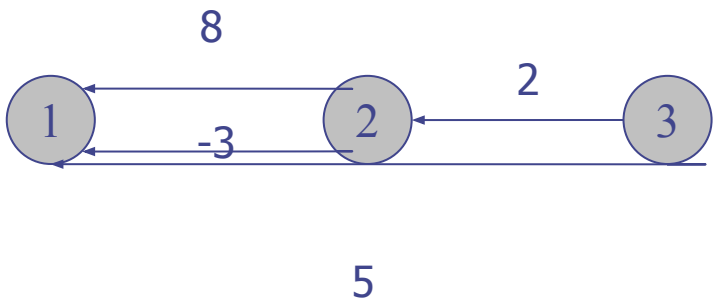
$A[3, 3] = \min(A[3, 3], A[3, 1] + A[1, 3])$

	1	2	3
1	0	8	5
2	-3	0	2
3	∞	2	0

A

$k = 1$

Floyd-Warshall

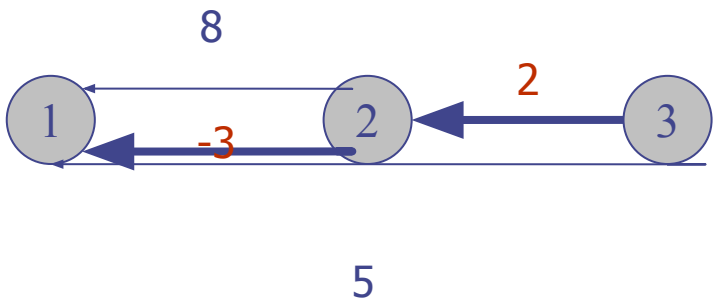


	1	2	3
1	0	8	5
2	-3	0	2
3	∞	2	0

A

- Ao final da iteração *k=1* tem-se todos os caminhos mais curtos entre *v* e *w* que podem passar pelo vértice 1.
- O processo se repete para *k=2* e *k=3*.

Floyd-Warshall



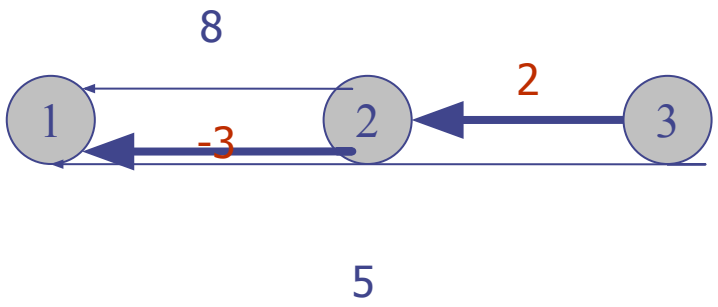
$A[3, 1] = \min(A[3, 1], A[3, 2] + A[2, 1])$

	1	2	3
1	0	8	5
2	-3	0	2
3	∞	2	0

A

$k =$ 2

Floyd-Warshall



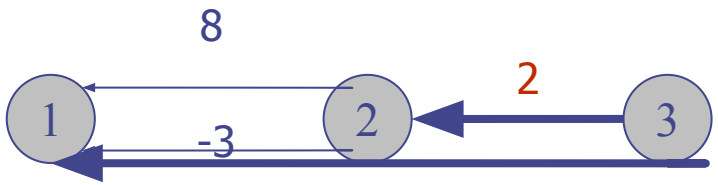
$A[3, 1] = \min(A[3, 1], A[3, 2] + A[2, 1])$

	1	2	3
1	0	8	5
2	-3	0	2
3	-1	2	0

A

$k =$ 2

Floyd-Warshall



5

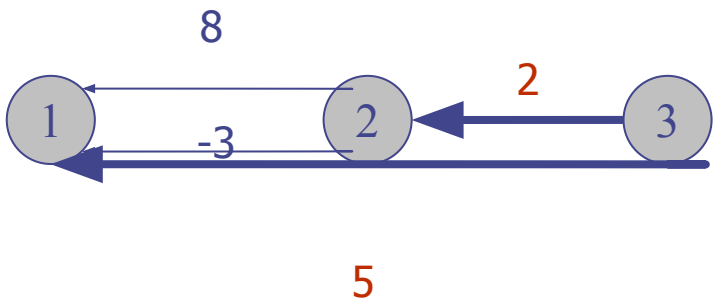
$A[1, 2] = \min(A[1, 2], A[1, 3] + A[3, 2])$

	1	2	3
1	0	8	5
2	-3	0	2
3	5	2	0

A

$k =$ 3

Floyd-Warshall



$A[1, 2] = \min(A[1, 2], A[1, 3] + A[3, 2])$

	1	2	3
1	0	7	5
2	-3	0	2
3	5	2	0

A

$k = 3$