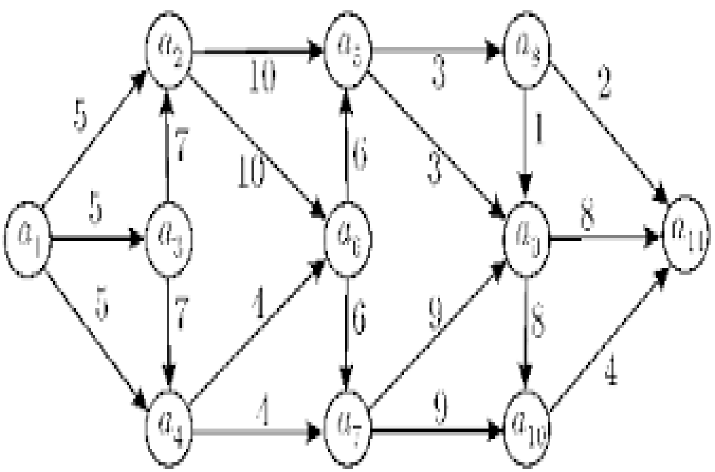
DISCIPLINA: ESTRUTURA DE DADOS II
2018

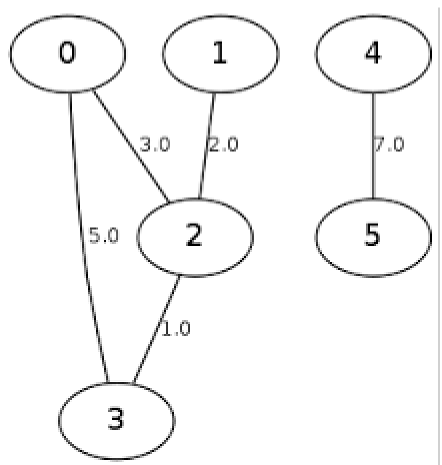
Caminhos mínimos

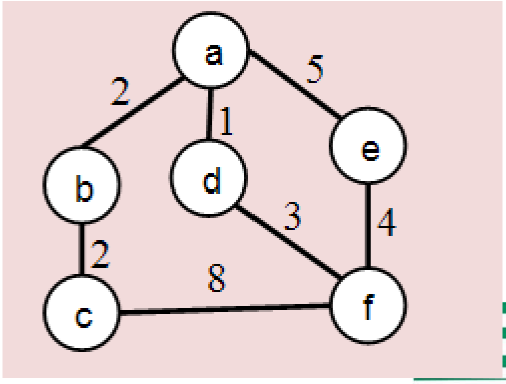
Prof. Luis Cuevas Rodríguez, PhD

Grafos ponderados

- Grafos ponderados: tem valores associados as arestas. Ex. custo, distancia, etc.







Problema de otimização

- Tem uma **função objetivo**
- Um conjunto de **restrições** (relacionados às variáveis de decisão)
- O problema pode ser de **minimização** ou de **maximização** da função objetivo.
- Os valores possíveis às variáveis de decisão são delimitados pelas restrições impostas sobre essas variáveis (formando um conjunto discreto (finito ou não) de soluções factíveis a um problema)
- A resposta para o problema de otimização (**ótimo global**), será o menor (ou maior) **valor possível para a função objetivo** para o qual o valor atribuído às variáveis não viole nenhuma restrição.
- Quando se tem valores cuja alteração discreta não conduz a resultados melhores, mas que não são também o óptimo global - a essas soluções chamamos de **ótimo local**.

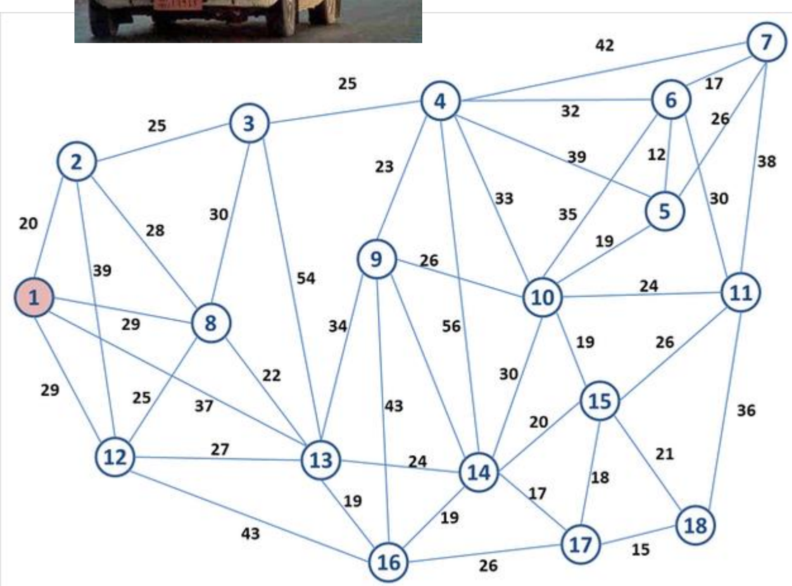
Problema de otimização - Exemplo

Caixeiro Viajante

- Visitar todas as cidades, passando por cada uma apenas uma vez e retornando ao seu ponto de origem, utilizando a **menor** distância possível.

Ir de uma cidade a outra pelo caminho mais curto.

- Determinar a ***menor*** rota desde uma cidade até outra.

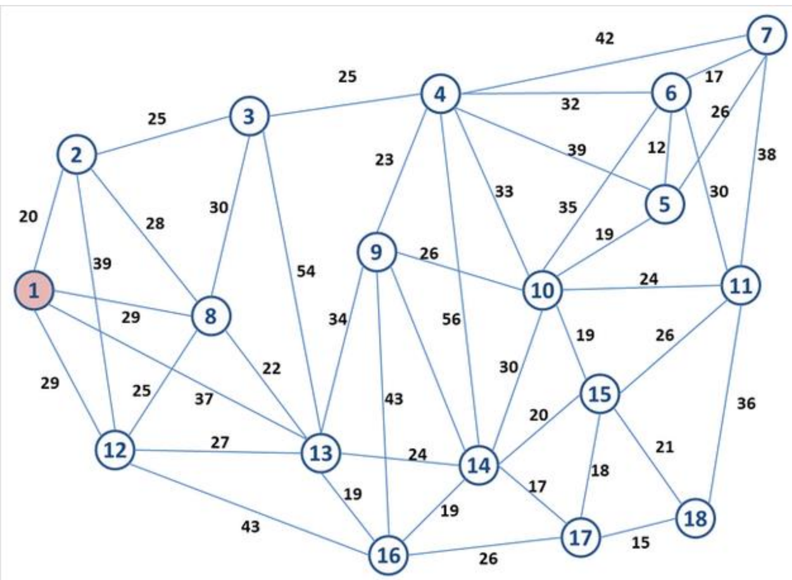


No domínio da teoria dos grafos

- cada cidade é identificada com um nó (ou vértice),
- as rotas que ligam cada par de nós são identicadas como arcos (ou arestas).
- A cada uma destas linhas estarão associadas as distâncias (ou custos) correspondentes.

https://otimizacaonapratica.wordpress.com/2015/11/09/o-problema-do-caixeiro-viajante/

Problema de otimização - Exemplo



- Se grafo é completo: é possível ir diretamente de uma cidade para qualquer outra
- Uma viagem que passe por todas as cidades corresponde a um **ciclo Hamiltoniano**, representado por um conjunto específico de linhas.
- A distância do ciclo é o somatório das distâncias das linhas presentes no mesmo.

ciclo Hamiltoniano: *é um caminho que permite passar por todos os vértices de um grafo G, não repetindo nenhum.* Um grafo que possua tal circuito é chamado de **grafo hamiltoniano**

Solução do Problema

- Procurar o(s) caminho(s) de custo mínimo,
- Função Objetivo:
  - $max_{Rota} = \sum_{i,j}(D_{i,j} * X_{i,j})$
    - ***i,j*** $\in$ ***E*** *(conjunto de arestas), i: vértice, j: vértice*
    - $\boldsymbol{D_{i,j}}$ é o custo ou distância do arco ***i,j***
    - $\boldsymbol{X_{i,j}}$ é a variável binária (0 ou 1) que decide se a rota ***i,j*** entrará ou não na solução do problema
- Restrições:
  - $\sum_{i,j \in E} X_{i,j} = 1$, $\forall$ *i (*Saída dos nós*)*
  - $\sum_{i,j \in E} X_{i,j} = 1$, $\forall$ *j (*Entrada nos nós*)*
  - Vértices de Inicio e fim da rota (para o segundo problema)

Problema do Caminho Mínimo.

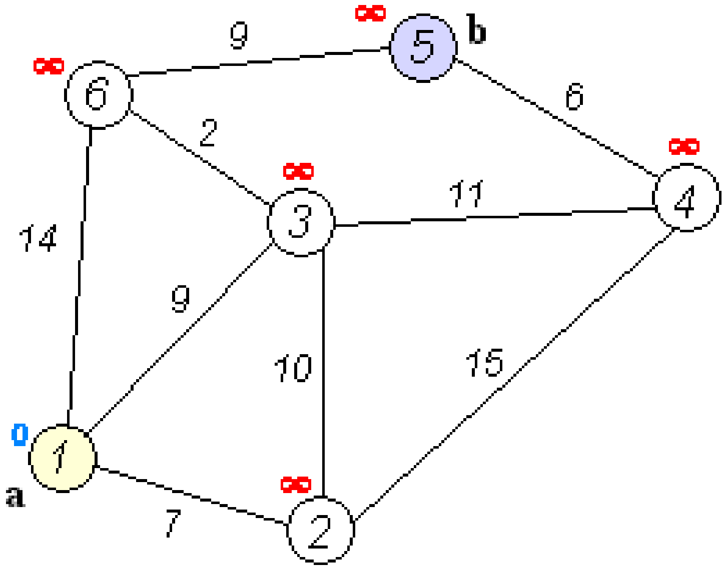
- Dada um grafo ***G=(V,E),*** uma função-custo ***c*** e um nó ***r*** $\in$ ***V***. Encontrar, para cada nó ***v*** $\in$ ***V***, um caminho de ***r*** a ***v*** que tenha custo **mínimo**.
- Solução do problema pode estar baseada na BFS (busca em largura)
- Algoritmos
  - Dijkstra
  - Bellman-Ford

Algoritmo de Dijkstra

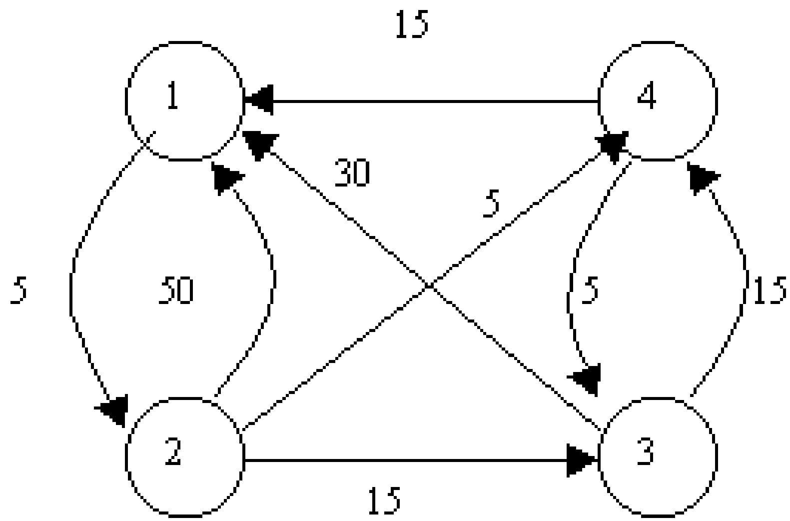
- Proposto por Edsger Dijkstra 1956 e publicado em 1959.
- Cientista da computação holandês conhecido por suas contribuições nas áreas de desenvolvimento de algoritmos e programas, de linguagens de programação, sistemas operacionais e processamento distribuído.
- Seu algoritmo
  - soluciona o problema do caminho mais curto num grafo dirigido ou não dirigido com arestas de peso **não negativo**
  - Grafo deve ser fortemenete conexo ou conexo. tempo computacional O([m+n]log n)
    - m: número de arestas e n: número de vértices

Algoritmo de Dijkstra

- Identifica um conjunto *S* de menores caminhos, iniciado com um vértice inicial *I*.
- Busca-se nas adjacências dos vértices pertencentes a *S* aquele vértice com menor distância relativa a I e adiciona-o a *S.*
- Repetindo os passos até que todos os vértices alcançáveis por *I* estejam em *S*.
- Arestas que ligam vértices já pertencentes a *S* são desconsideradas.



Algoritmo de Dijkstra – Exemplo 1

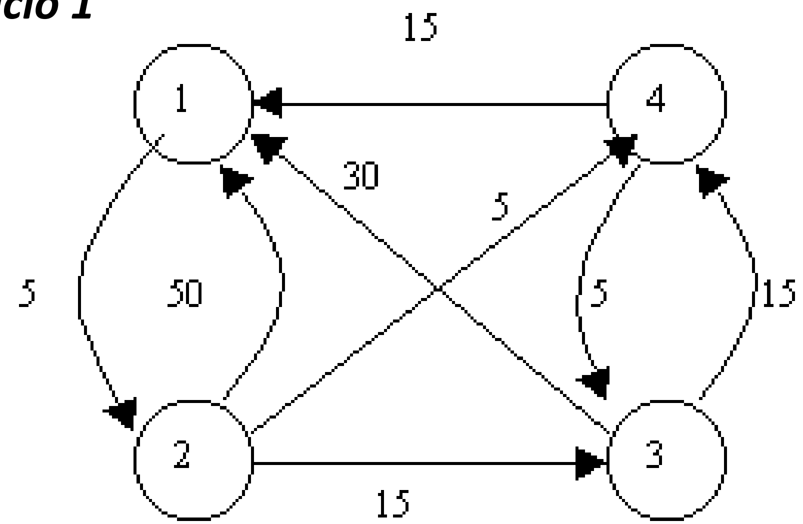


1 ➔	2,5
2 ➔	1,50 ➔ 4,5 ➔ 3,15
3 ➔	1,30 ➔ 4,15
4 ➔	1,15 ➔ 3,5

- A árvore de caminhos mínimos pode ser representada por dois vetores indexados pelos vértices v
  - **dist[v]**: armazena o comprimento (custo) do caminho mínimo entre *s* e *v*.
  - **prev[v]**: armazena a última aresta do caminho mínimo entre *s* e *v*.

Algoritmo de Dijkstra – Exempo1

Inicio 1

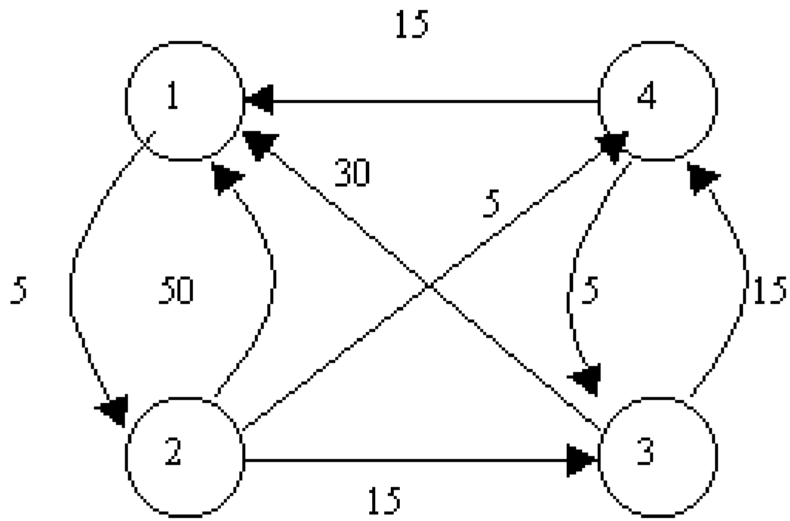


1 →	2,5
2 →	1,50 → 4,5 → 3,15
3 →	1,30 → 4,15
4 →	1,15 → 3,5

Desde 1

V	1	2	3	4
dist[v]	0	∞	∞	∞
prev[v]	NULL	NULL	NULL	NULL

Algoritmo de Dijkstra – Exemplo1

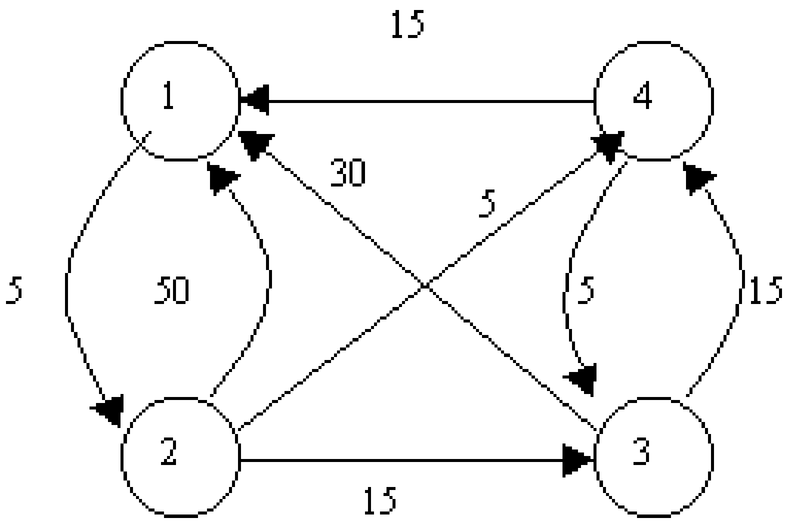


1 →	2,5
2 →	1,50 → 4,5 → 3,15
3 →	1,30 → 4,15
4 →	1,15 → 3,5

Desde 1

V	1	2	3	4
dist[v]	0	5	∞	∞
prev[v]	NULL	1	NULL	NULL

Algoritmo de Dijkstra-Exemplo1



1 ➔	2,5
2 ➔	1,50 ➔ 4,5 ➔ 3,15
3 ➔	1,30 ➔ 4,15
4 ➔	1,15 ➔ 3,5

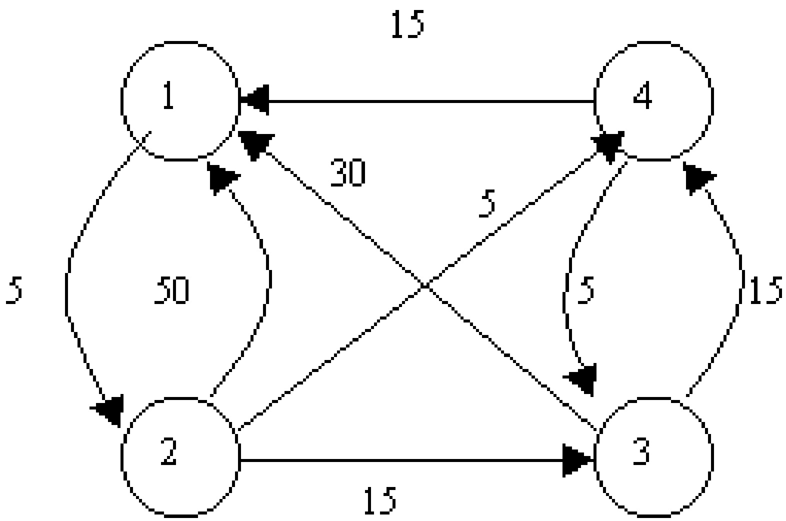
Desde 2 – Selecionar o no aberto como menor custo No 4

V	1	2	3	4
dist[v]	0	5	20	10
prev[v]	NULL	1	2	2

Prof. Luis Cuevas Rodríguez, PhD

UEA
UNIVERSIDADE DO ESTADO DO AMAZONAS

Algoritmo de Dijkstra-Exemplo1



1 →	2,5
2 →	1,50 → 4,5 → 3,15
3 →	1,30 → 4,15
4 →	1,15 → 3,5

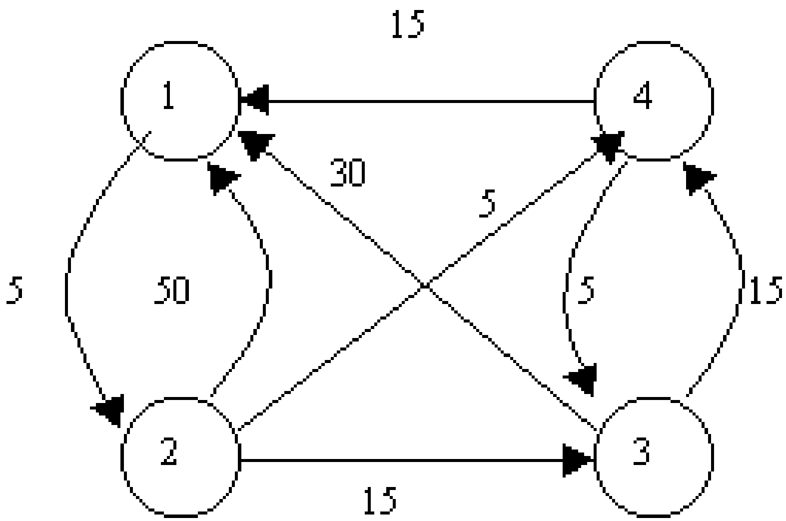
Desde 4

V	1	2	3	4
dist[v]	0	5	15	10
prev[v]	NULL	1	4	2

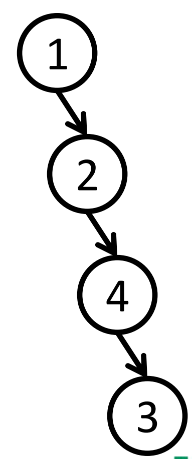
Algoritmo de Dijkstra

- Quando todos os vértices tiverem sido avaliados, os valores obtidos serão os custos mínimos dos caminhos que partem do vértice tomado como raiz da busca até os demais vértices do grafo.
- O caminho propriamente dito é obtido a partir dos vértices chamados acima de precedentes. Árvore de caminhos mínimos

Algoritmo de Dijkstra-Exemplo1



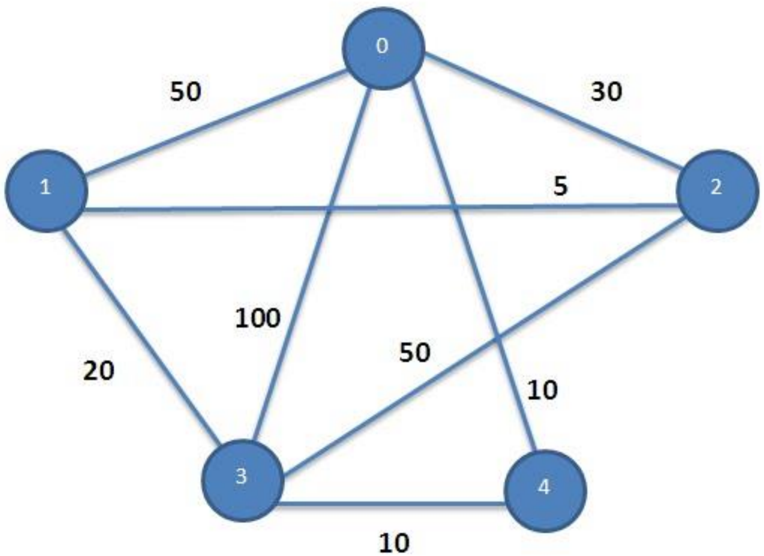
1 ➔	2,5
2 ➔	1,50 ➔ 4,5 ➔ 3,15
3 ➔	1,30 ➔ 4,15
4 ➔	1,15 ➔ 3,5



V	1	2	3	4
dist[v]	0	5	15	10
prev[v]	NULL	1	4	2

Algoritmo de Dijkstra

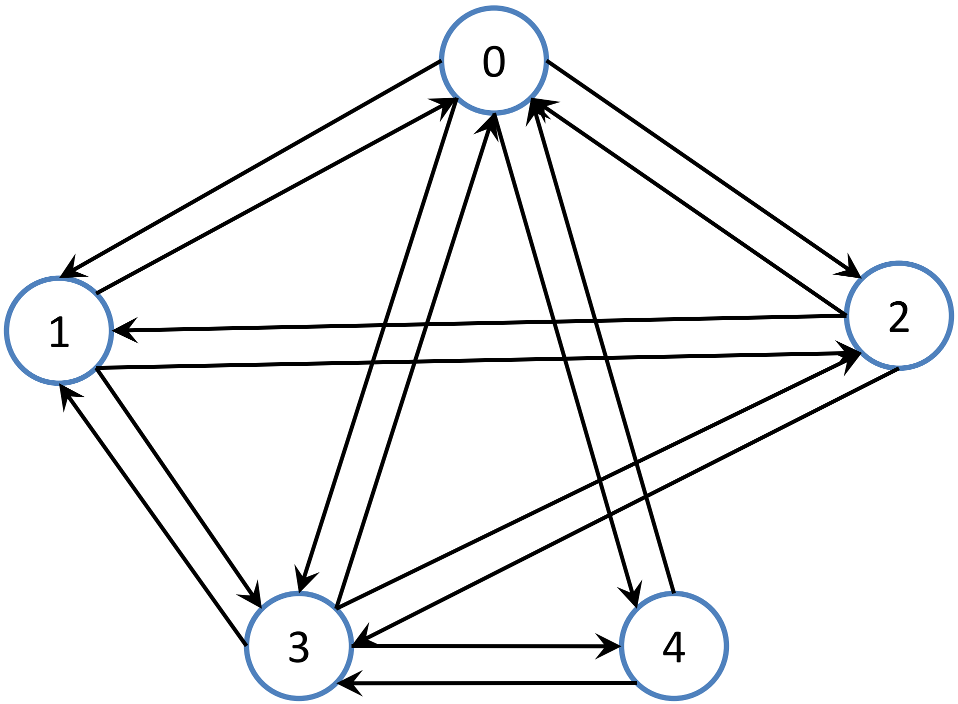
- Grafos não direcionados



0 ➔	1,50 ➔ 2,30 ➔ 3, 100 ➔ 4,10
1 ➔	0,50 ➔2,5 ➔3,20
2 ➔	0,30 ➔ 1,5 ➔ 3,50
3 ➔	1,20 ➔ 0,100 ➔ 2,50 ➔4,10
4 ➔	0,10 ➔ 3,10

- Transformar o Grafo não direcionado em direcionado

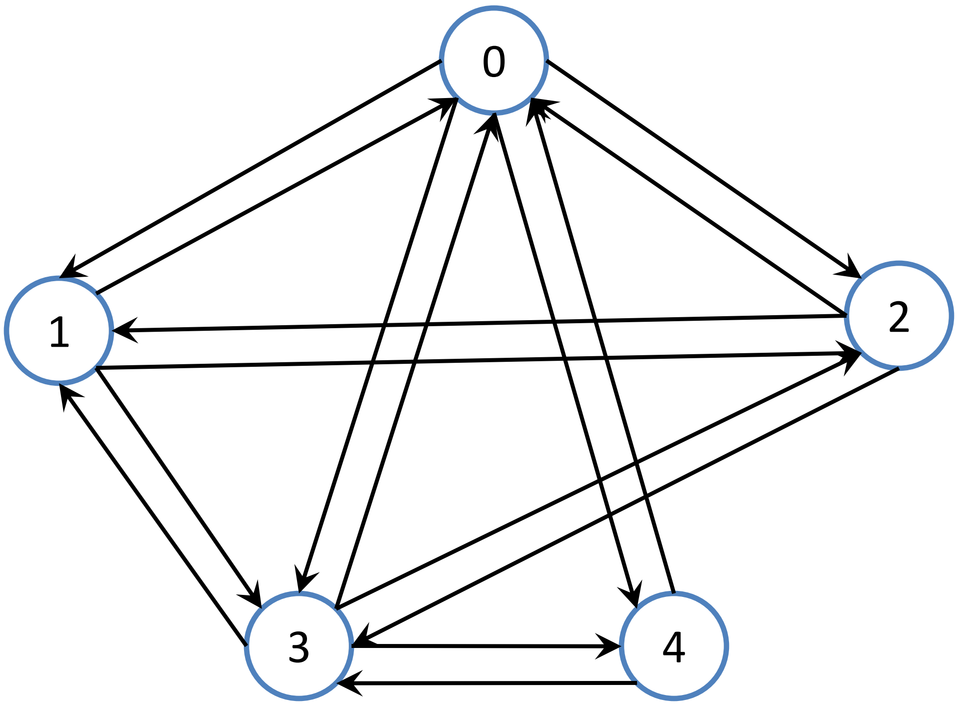
Algoritmo de Dijkstra – Não direcionado



0 ➔	1,50 ➔ 2,30 ➔ 3, 100 ➔ 4,10
1 ➔	0,50 ➔2,5 ➔3,20
2 ➔	0,30 ➔ 1,5 ➔ 3,50
3 ➔	1,20 ➔ 0,100 ➔ 2,50 ➔4,10
4 ➔	0,10 ➔ 3,10

V	0	1	2	3	4
dist[v]	0	∞	∞	∞	∞
prev[v]	NULL	NULL	NULL	NULL	NULL

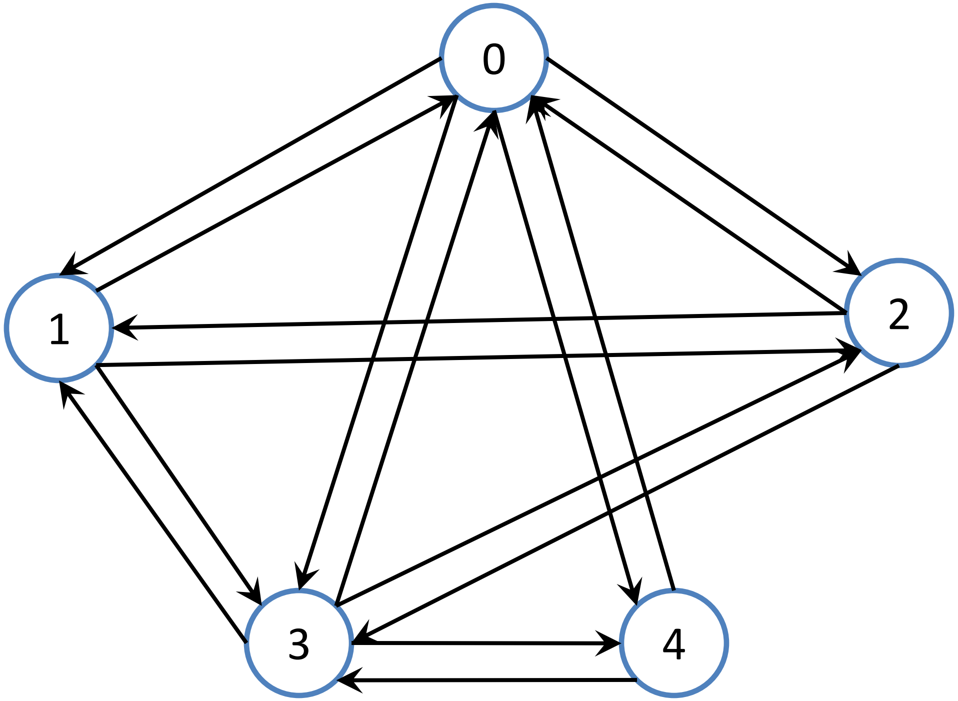
Algoritmo de Dijkstra – Não direcionado



0 → 1,50 → 2,30 → 3, 100 → 4,10
1 → 0,50 →2,5 →3,20
2 → 0,30 → 1,5 → 3,50
3 → 1,20 → 0,100 → 2,50 →4,10
4 → 0,10 → 3,10

V	**0**	1	2	3	4
dist[v]	0	50	30	100	10
prev[v]	NULL	0	0	0	0

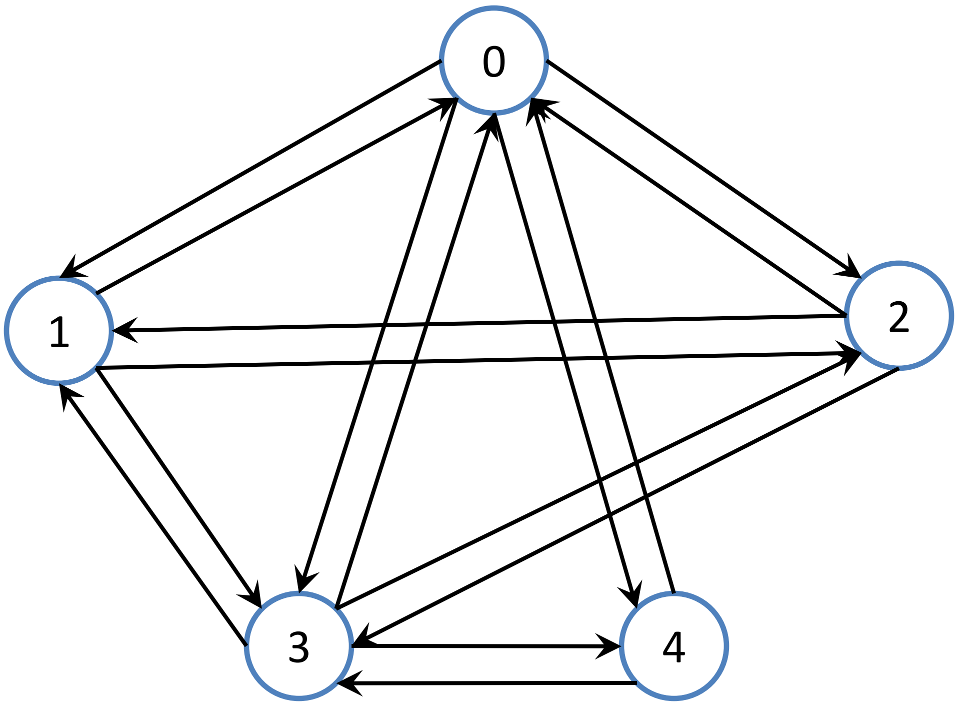
Algoritmo de Dijkstra – Não direcionado



0 →	1,50 → 2,30 → 3, 100 → 4,10
1 →	0,50 →2,5 →3,20
2 →	0,30 → 1,5 → 3,50
3 →	1,20 → 0,100 → 2,50 →4,10
4 →	0,10 → 3,10

V	0	1	2	3	**4**
dist[v]	0	50	30	20	10
prev[v]	NULL	0	0	4	0

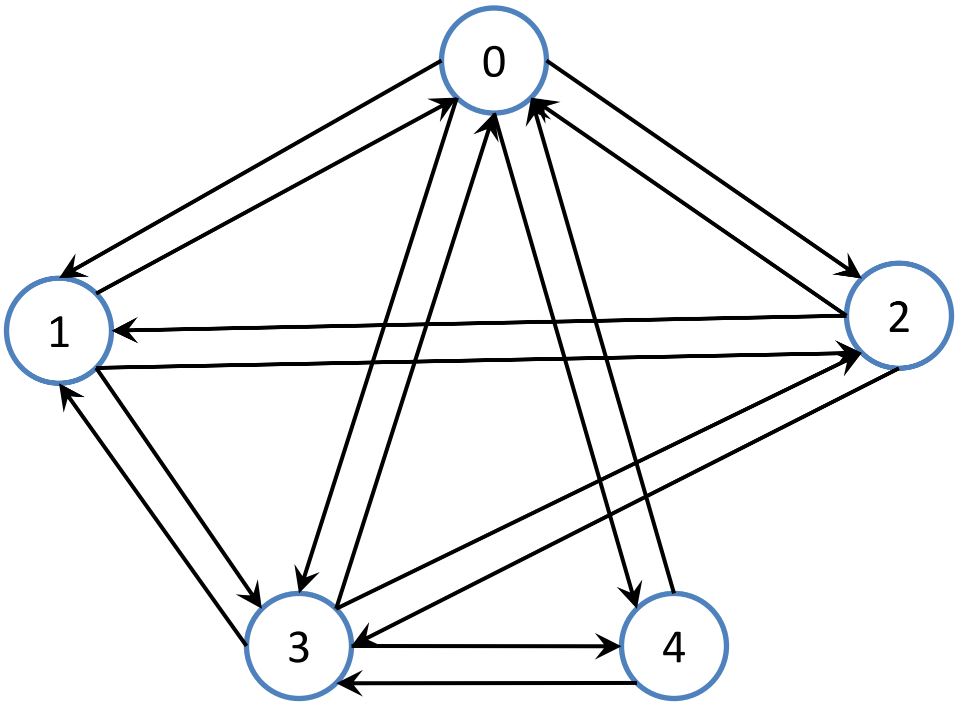
Algoritmo de Dijkstra – Não direcionado



0 →	1,50 → 2,30 → 3, 100 → 4,10
1 →	0,50 →2,5 →3,20
2 →	0,30 → 1,5 → 3,50
3 →	1,20 → 0,100 → 2,50 →4,10
4 →	0,10 → 3,10

V	**0**	1	2	**3**	4
dist[v]	0	40	30	20	10
prev[v]	NULL	3	0	4	0

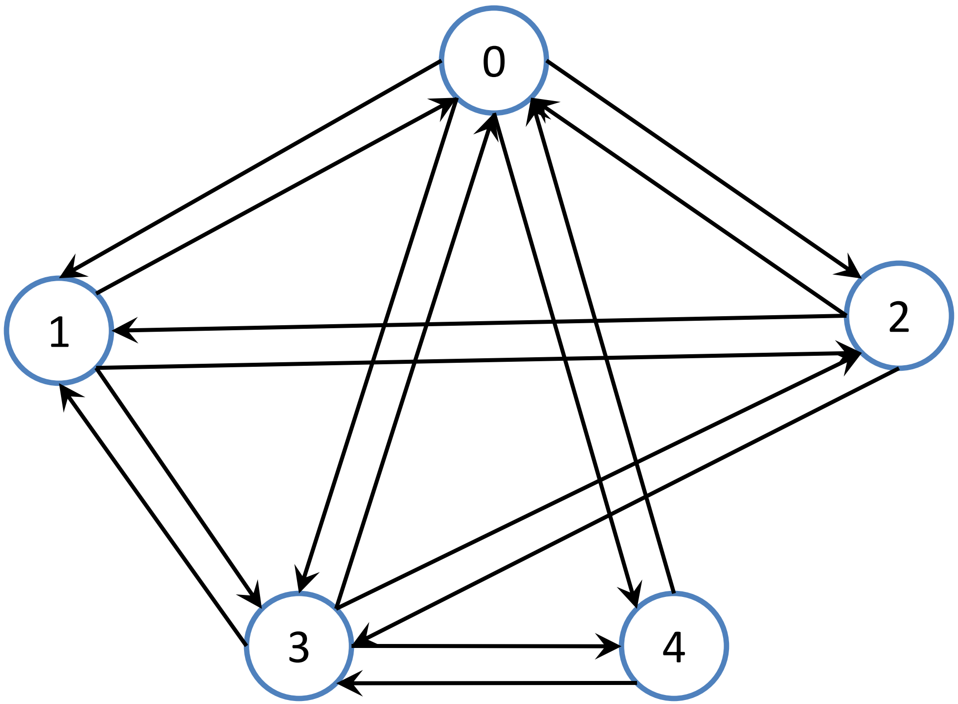
Algoritmo de Dijkstra – Não direcionado



0 →	1,50 → 2,30 → 3, 100 → 4,10
1 →	0,50 →2,5 →3,20
2 →	0,30 → 1,5 → 3,50
3 →	1,20 → 0,100 → 2,50 →4,10
4 →	0,10 → 3,10

V	**0**	1	**2**	**3**	4
dist[v]	0	35	30	20	10
prev[v]	NULL	2	0	4	0

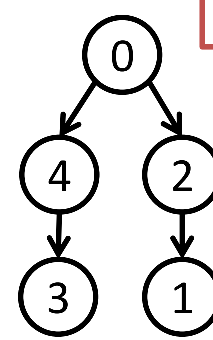
Algoritmo de Dijkstra – Não direcionado



0 ➔	1,50 ➔ 2,30 ➔ 3, 100 ➔ 4,10
1 ➔	0,50 ➔2,5 ➔3,20
2 ➔	0,30 ➔ 1,5 ➔ 3,50
3 ➔	1,20 ➔ 0,100 ➔ 2,50 ➔4,10
4 ➔	0,10 ➔ 3,10

Caminhos:
0, 4, 3
0, 2, 1

V	**0**	**1**	**2**	**3**	4
dist[v]	0	35	30	20	10
prev[v]	NULL	2	0	4	0

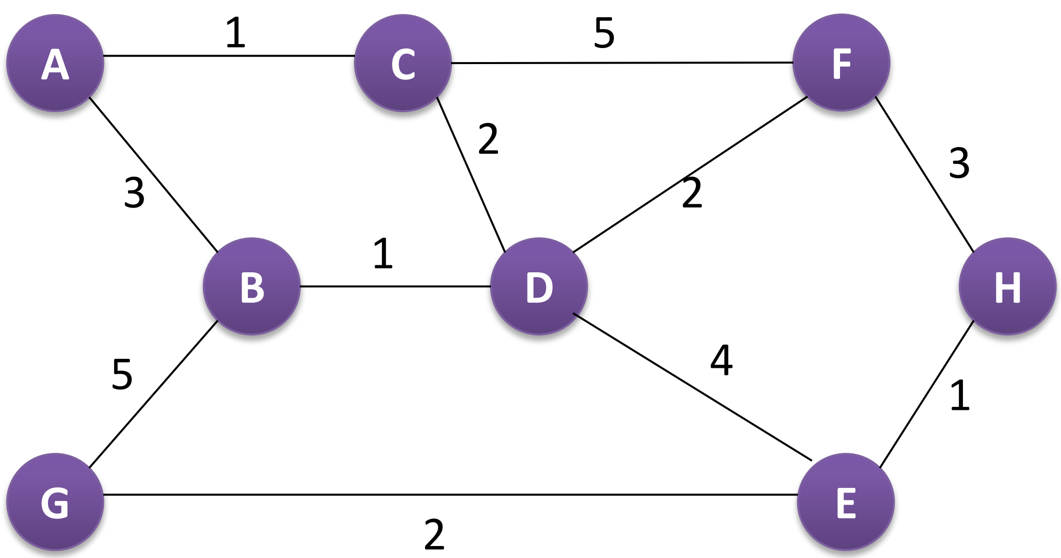


Prof. Luis Cuevas Rodríguez, PhD

UNIVERSIDADE DO ESTADO DO AMAZONAS

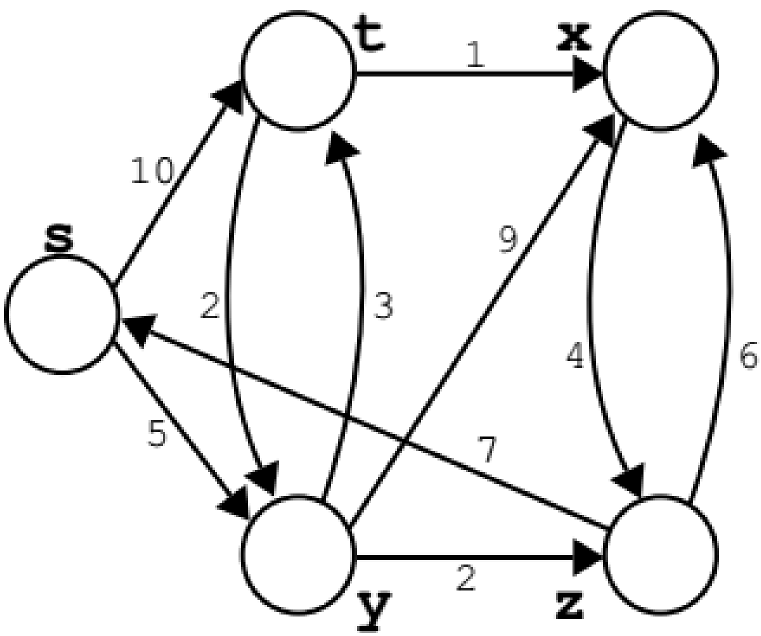
Exercícios

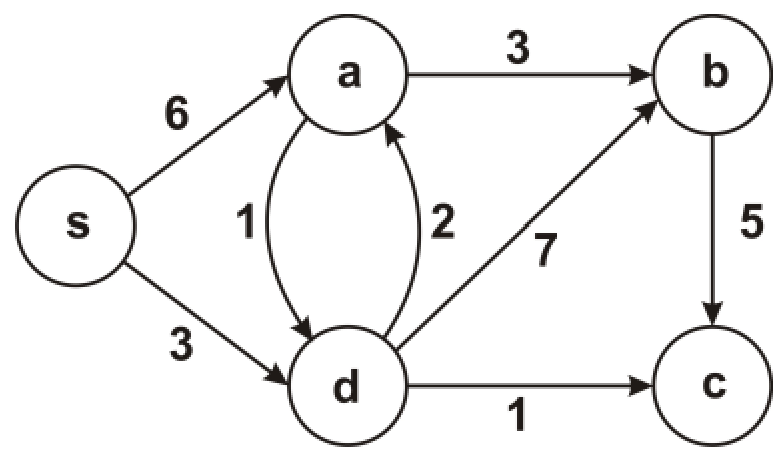
- Obter os caminhos mínimos



Exercícios

- Obter o caminho mínimo usando o algoritmo de *Dijkstra*, tendo como vértice inicial o vértice *s*





UEA
UNIVERSIDADE DO ESTADO DO AMAZONAS

Exercício

- Mostre que o Algoritmo de Dijkstra se comporta como uma busca em largura se os pesos de todas as arestas são iguais a um.

Algoritmo de Dijkstra

```
Dijkstra(G , s ):
    Para todos os nós v de G fazer:
        v.dist = ∞
        v.visitado = falso
s.dist = 0
s.pred = s
Enquanto existirem nós não visitados fazer:
    Seleccionar nó u não visitado com menor valor de dist
    u.visitado = verdadeiro
    Para cada aresta (u, v ) de G fazer:
        Se v.visitado = falso e u.dist + peso (u, v ) < v.dist então
            v.dist = u.dist + peso (u, v )
            v.pred = u
```

Dijkstra - Complexidade

Dijkstra(G , s):
Para todos os nós v de G **fazer**:
    v.dist = ∞
    v.visitado = falso
s.dist = 0
s.pred = s
Enquanto existirem nós não visitados **fazer**:
    Seleccionar nó *u* não visitado com menor valor de dist
    u.visitado = verdadeiro
    Para cada aresta (u, v) de G **fazer**:
        Se v.visitado = falso **e** u.dist + peso (u, v) < v.dist então
            v.dist = u.dist + peso (u, v)
            v.pred = u

Executa |V| vezes

Executa |V| vezes

Depende do algoritmo usado

Executa |E| vesses, máximo |V| vezes

SUMA(O(|V|), O(SUMA(O(|V|x busca menor),O(|V|x|E|)))

Dijkstra - Complexidade

- Complexidade
  - Busca iterativa
    - SUMA(O(|V|), O(SUMA(O(|V|x|V|),O(|V|x|E|)))= SOMA (O(|V|),O(|V|²))=O(|V|²)
  - usarmos uma fila de prioridade (min-heap)
    - SUMA(O(|V|), O(SUMA(O(|V|x log |V|),O(|V|x|E|)))= SOMA (O(|V|),O(|V|x log |V|))=O(|V| log |V|)
    - Mas, quando percorre as arestas |E| tem que atualizar a fila de prioridade e |E| ≥ |V|, pelo que a complexidade é O(|E| log |V|)

Implementação

- Buscar o vértice com a menor distância.

```
int procura_no(float d[], bool v[], int verts){
    float min_dist = HUGE_VALF;
    int min_no = -1;
    for (int i= 0; i<verts; i++){
        if (v[i]== false && min_dist > d[i]){
            min_dist = d[i];
            min_no = i;
        }
    }
    return min_no;
}
```

d[] → vetor com as distâncias para cada nó; v[] → vetor nó visitado; verts→ numero de vértices do grafo

Implementação

```
void dijkstra(GRAFO* gr, int no){
    float dist[gr->vertices];
    bool visit[gr->vertices];
    int pred [gr->vertices];
    int no_trab;

    for (int i= 0; i<gr->vertices; i++){
        dist[i]=HUGE_VALF;
        visit[i]= false;
    }
    dist[no]=0;
    pred[no]=no;
    no_trab = procura_no(dist,visit,gr->vertices);
```

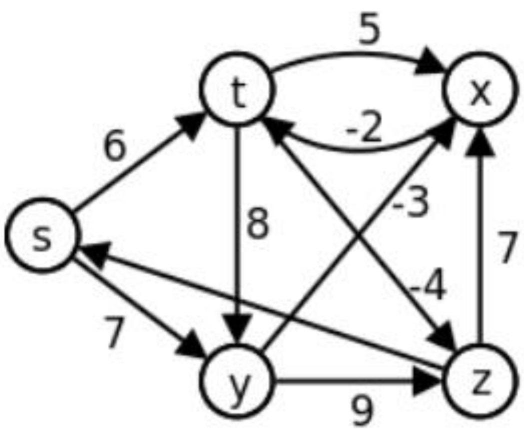
Implementação

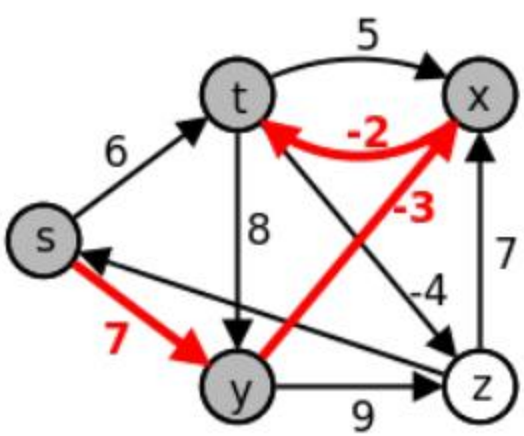
```
while(no_trab != -1){
    visit[no_trab]=true;

    ADJACENCIA* ad=gr->vert[no_trab].cad;
    cout << no_trab;
    while (ad){
        if (visit[ad->vertice]==false &&
            ((dist[no_trab]+ad->peso)< dist[ad->vertice]))
        {
            dist[ad->vertice]= dist[no_trab]+ad->peso;
            pred[ad->vertice]= no_trab;
        }
        ad = ad->prox;
    }
    no_trab = procura_no(dist,visit,gr->vertices);
}
imprime_vetor_int(pred,gr->vertices);
imprime_vetor_float(dist, gr->vertices);
}
```

Arestas de pesos negativos

Qual é a distância de *s* até *t*?





	s	t	y	x	Z
dist	0	∞	∞	∞	∞
pred	N	N	N	N	N

	s	t	y	x	Z
dist	0	6	7	∞	∞
pred	N	S	S	N	N

	s	**t**	y	x	z
dist	0	**6**	7	11	2
pred	N	**s**	s	t	t

	s	**t**	y	x	**z**
dist	0	6	7	9	2
pred	N	s	s	z	t

- A distância é 6
  - caminho={s, t}
- Mas tem uma distância menor 2
  - caminho {s, y, x, t}

Não é solução: tornar todas as arestas positivas acrescentando uma constante (penaliza os caminhos de acordo com o número de arestas que têm)

ALGORITMO DE BELLMAN-FORD

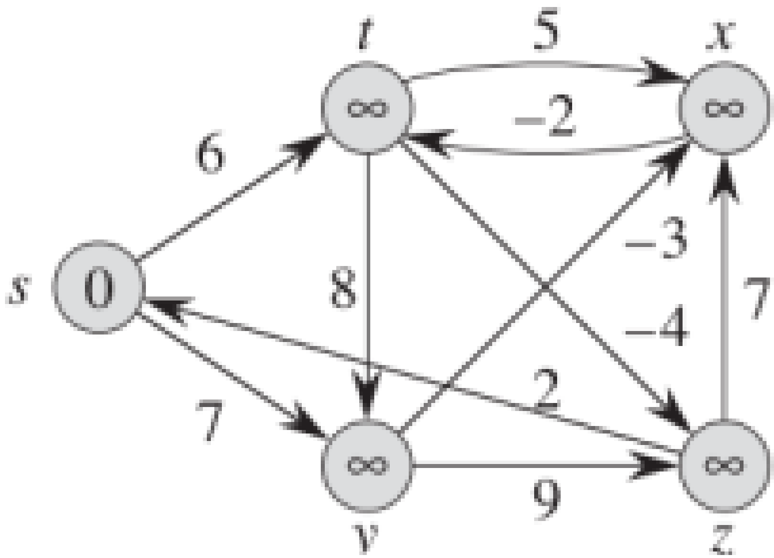
Prof. Luis Cuevas Rodríguez, PhD

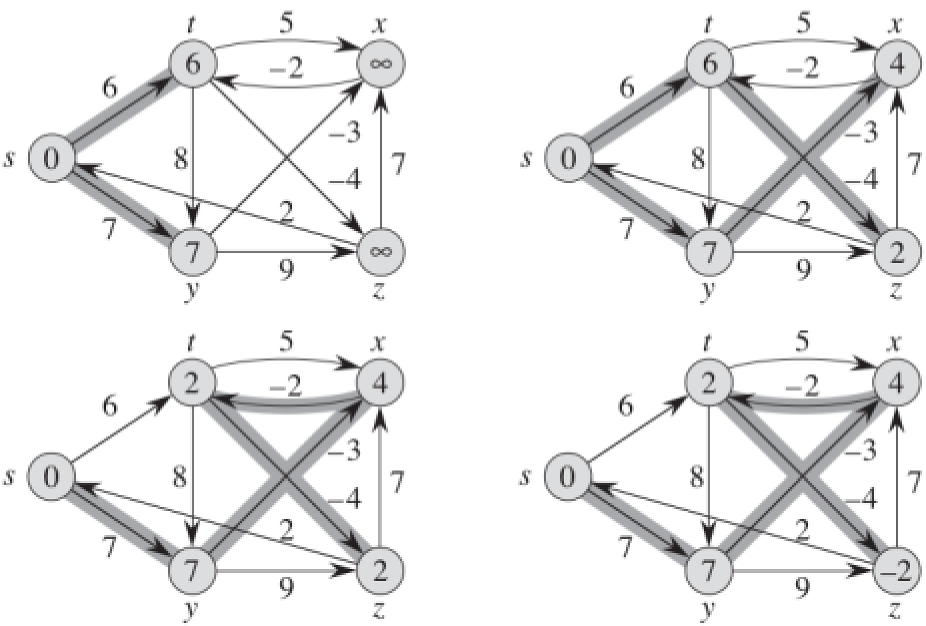
UNIVERSIDADE DO ESTADO DO AMAZONAS

Algoritmo de Bellman-Ford

- Resolver o mesmo problema mas com arestas de **pesos negativos.**
- Passos
  - Inicializar: Inicializar os vetores dist[v], pred[v]
  - Relaxamento das arestas: diminuir a estimativa da distância de um vértice *dist[v]*
  - Verificar Ciclos Negativos: ciclo com peso menor que 0.

Algoritmo de Bellman-Ford





	s	t	x	y	z
dist	∞	∞	∞	∞	∞
prec	N	N	N	N	N

	s	t	x	y	Z
dist	0	∞	∞	∞	∞
prec	s	N	N	N	N

	s	t	x	y	z
dist	0	6	4	7	2
prec	N	s	y	s	t

primeiro relaxamento

	s	t	x	y	z
dist	0	2	4	7	2
prec	N	x	y	s	t

segundo relaxamento

	s	t	x	y	z
dist	0	2	4	7	-2
prec	N	x	y	s	t

terceiro relaxamento

Quarto relaxamento:não provoca mudanças

UEA UNIVERSIDADE DO ESTADO DO AMAZONAS

Algoritmo de Bellman-Ford

- Verificar Ciclos Negativos
- Ciclos Negativos: Se têm um custo < 0
- Se tem ciclo negativo o problema não tem solução (ciclo infinito sempre melhorando o custo)
- Solução: depois de relaxar |V|-1 vezes , tentar relaxar novamente |V|-1 vezes e se têm pelo menos uma mudança esta num ciclo negativo

Algoritmo de Bellman-Ford

Bellman-Ford(G, s):
Para todos os nós v de G **fazer**:
  v.dist = ∞
s.dist = 0
s.pred = s
Para i=1 até |V| - 1 **fazer**:
  Para todas as arestas (u, v) de G **fazer**:
    Se u.dist + peso(u, v) < v.dist **então**
      v.dist = u.dist + peso(u, v)
      v.pred = u
Para i=1 até |V| - 1 **fazer**:
  Para todas as arestas (u; v) de G **fazer**:
    Se u.dist + peso(u; v) < v.dist **então**
      erro("Existe ciclo negativo!")

Executa |V| vezes

Executa |V|-1 vezes

Executa |E| vezes

Executa |E| vezes

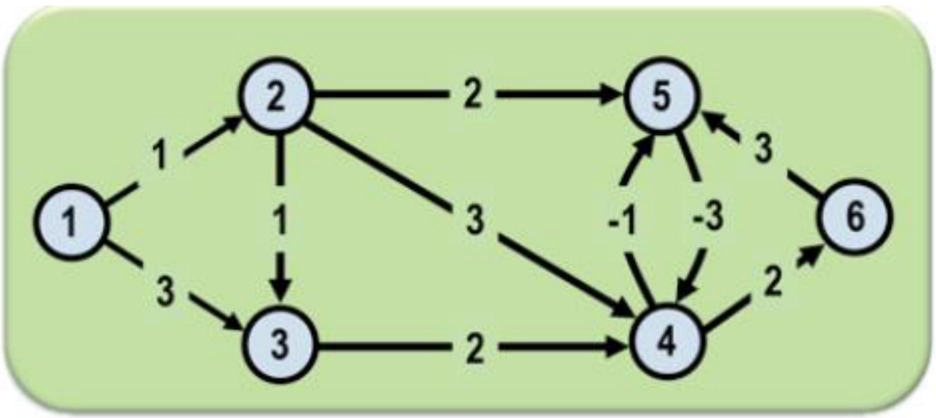
Bellman-Ford / Complexidade

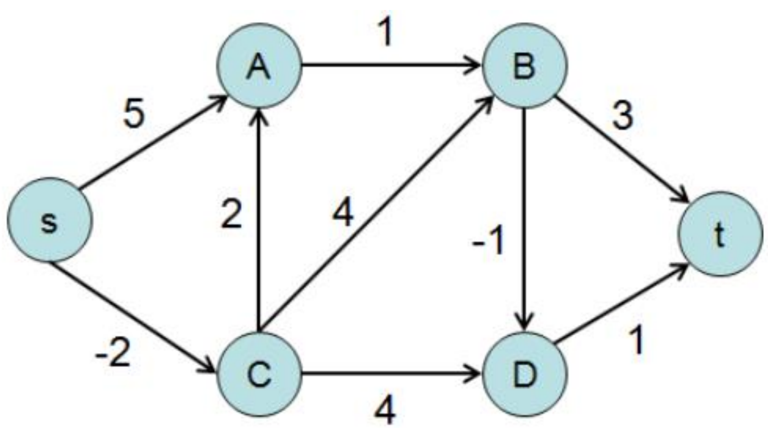
- Complexidade
  - SUMA(O(|V|), O(|V|-1x|E|),O(|V|-1x|E|))) =O(|V|-1 x |E|)

  - =O(|V| x |E|)

Exercícios

Execute o algoritmo de *Bellman-Ford* para





Prof. Luis Cuevas Rodríguez, PhD

UEA
UNIVERSIDADE DO ESTADO DO AMAZONAS

Exercícios

- Execute o algoritmo de *Bellman-Ford* para