Algoritmos e Estruturas de Dados II

Grafos VI:

Grafos Ponderados & Caminhos Mínimos (Bellman-Ford)

Ricardo J. G. B. Campello

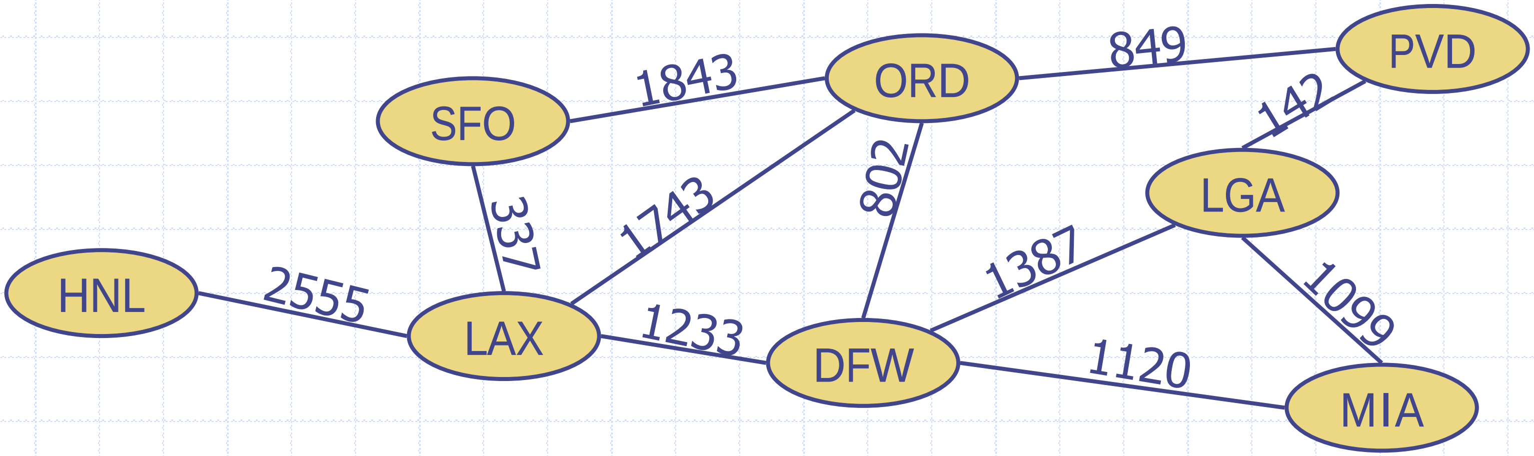
Parte deste material é baseado em adaptações e extensões de slides disponíveis em http://ww3.datastructures.net (Goodrich & Tamassia).

Organização

- Grafos Ponderados
  - Adaptação da Estrutura de Dados Alternativa
  - Propriedades
- Caminhos Mínimos
  - Equações Funcionais
  - Algoritmo Bellman-Ford

Grafos Ponderados

- Um **grafo ponderado** é um grafo que possui rótulos numéricos (**pesos**) associados a cada aresta. Possui ampla aplicabilidade.
- Existem algoritmos especializados para calcular o **caminho mais curto** (*shortest path*) entre quaisquer dois vértices do grafo. Por exemplo:
  - **Bellman-Ford**, **Floyd-Warshall**, **Dijkstra**, entre outros.
- O mesmo vale para a **árvore geradora mínima**:
  - **Prim**, **Prim-Jarník**, **Kruskal**, ...



Grafos Ponderados

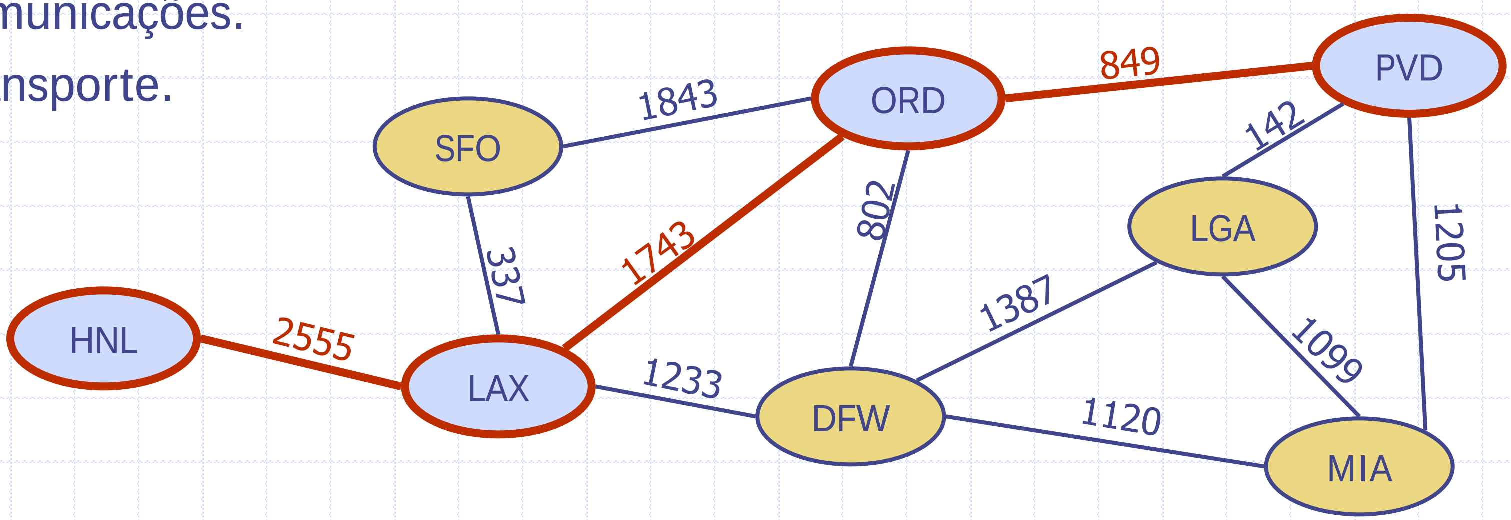
◆ Estrutura de Dados Simples:

```
typedef struct {
    int v;                    /* opposite vertex */
    int weight;               /* edge weight */
} edge;

typedef struct {
    edge edges[MAXV+1][MAXDEGREE];  /* adjacency info */
    int degree[MAXV+1];             /* outdegree of each vertex */
    int nvertices;                  /* number of vertices in the graph */
    int nedges;                     /* number of edges in the graph */
} graph;
```

Caminhos Mais Curtos

- Dado um grafo simples, direcionado e ponderado, e dois vértices ***u*** e ***v***, tais que ***v*** é alcançável a partir de ***u***, queremos encontrar um caminho de custo (distância) total mínimo entre ***u*** e ***v*.**
  - Custo de um caminho é a soma total dos pesos das suas arestas.
- Exemplo:
  - Caminho mais curto entre Providence (PVD) e Honolulu (HNL).
- Principais Aplicações
  - Comunicações.
  - Transporte.
  - ...

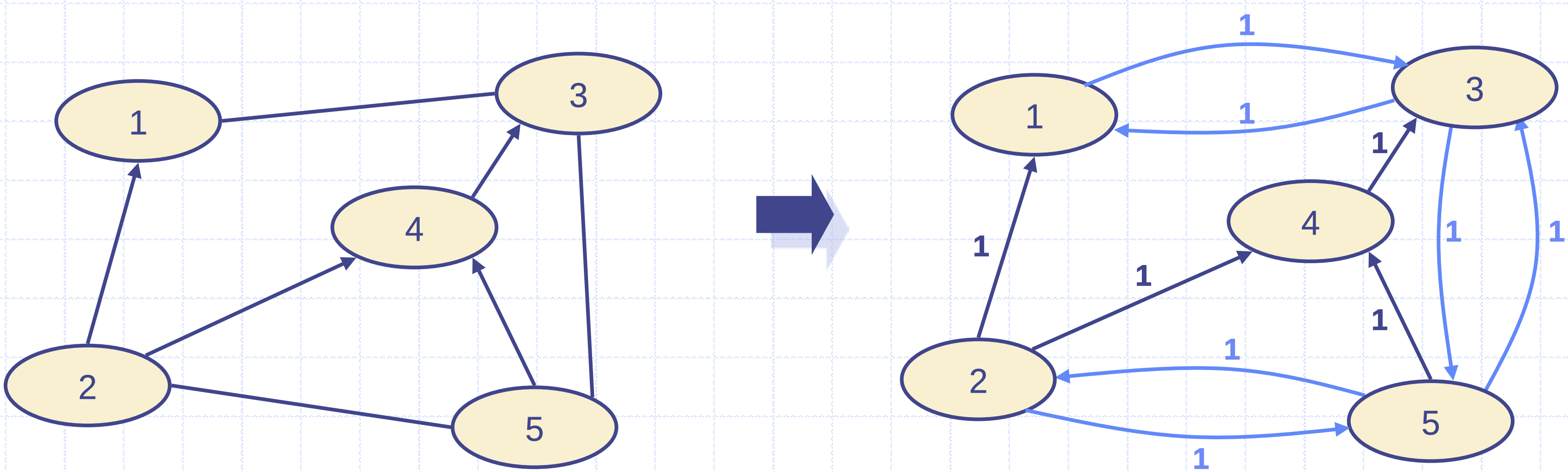


Propriedades

Propriedade 1

(a) Qualquer grafo não direcionado ou misto pode ser transformado em direcionado (digrafo puro) substituindo cada aresta não direcionada por duas em direções opostas; e

(b) Grafos não ponderados podem ser considerados ponderados com pesos unitários.

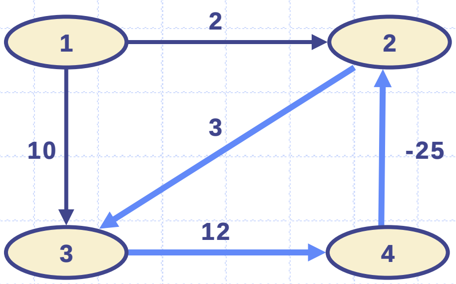


Propriedades

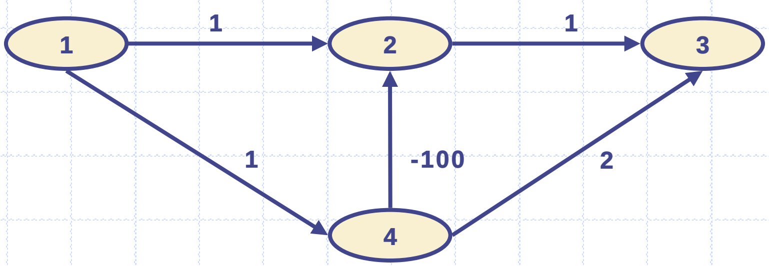
Propriedade 2 (Princípio de Otimalidade)

Se não existirem ciclos direcionados com custo total negativo (**diciclos negativos**), então:

- Um sub-caminho simples de um caminho mais curto simples é também um caminho mais curto simples por si só.
- Em outras palavras, se um vértice ***x*** faz parte do caminho mais curto de um outro vértice ***u*** até um terceiro vértice ***v***, então o sub-caminho entre ***u*** e ***x*** é o caminho mais curto entre esses dois vértices.



Presença de Diciclo Negativo: 1-2-3 é o caminho simples ótimo, mas 1-2 não é (ver 1-3-4-2).



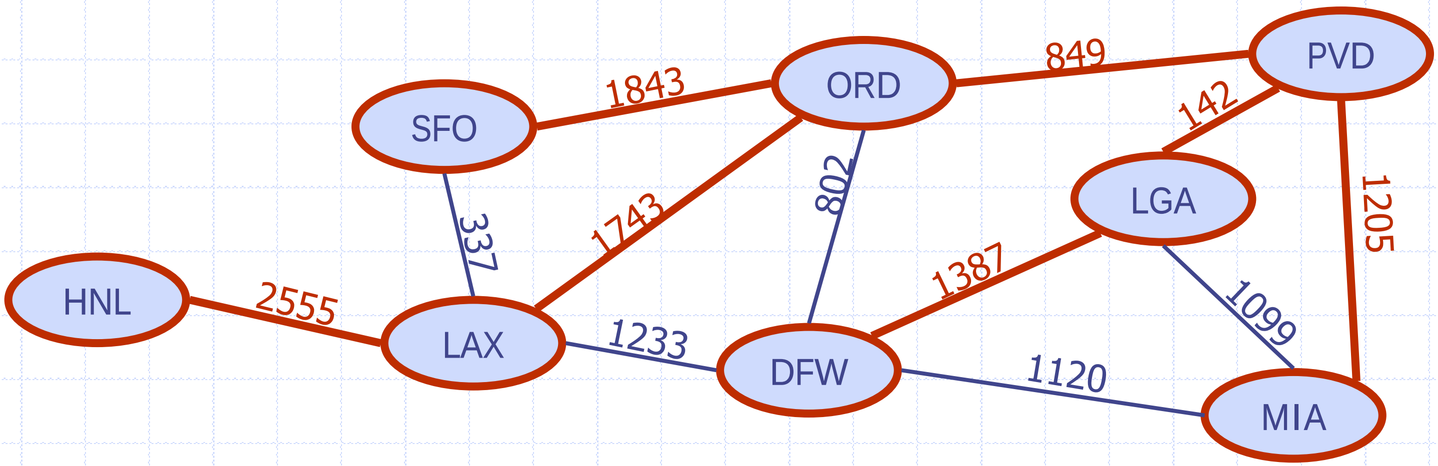
Ausência de Diciclo Negativo: 1-4-2-3 é caminho simples ótimo, assim como são 1-4, 1-4-2, 4-2, 4-2-3 e 2-3.

Propriedades

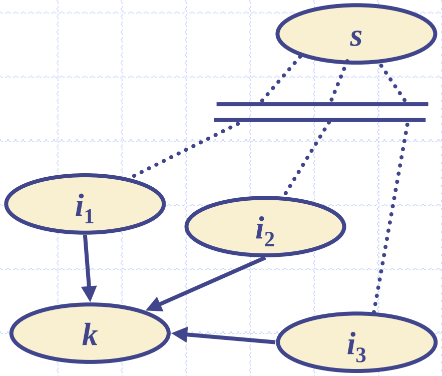
Propriedade 3

A propriedade 2 implica que existe uma árvore de caminhos mais curtos simples de um vértice inicial até todos os outros vértices.

Exemplo (a partir de PVD):



Equações Funcionais



Caso um-para-todos (one-to-all):

A propriedade 2 implica um conjunto de equações necessárias e suficientes para a otimalidade de caminhos mais curtos.

Seja $c_{i,k}$ o peso da aresta direcionada (i, k) ligando algum vértice i ao vértice k e $d[k]$ a distância do caminho mais curto entre um vértice de origem s e um vértice qualquer k do grafo.

Então:

$$\begin{cases} d[s] = 0 \\ d[k] = \min_i \{ d[i] + c_{i,k} : (i, k) \text{ existe} \} \quad \forall\, k \neq s \end{cases}$$

PS. d[k] é definida como d[k] = ∞ se não existe caminho entre s e k .

Algoritmo Bellman-Ford (one-to-all)

As equações funcionais claramente apresentam uma relação de inter-dependência recursiva entre os valores de $d[k]$ para os diferentes vértices, que impedem uma solução direta.

A idéia do algoritmo Bellman-Ford é avaliar repetidamente as equações utilizando os valores da iteração precedente.

Passo 0 (Inicialização): Dado o vértice de origem s, inicialize as distâncias como:

$$d^0[k] = \begin{cases} 0 \text{ se } k = s. \\ \infty \text{ caso contrário.} \end{cases}$$

e o contador de iterações como $t = 1$.

Passo 1 (Atualização): Para cada vértice $k \neq s$ faça:

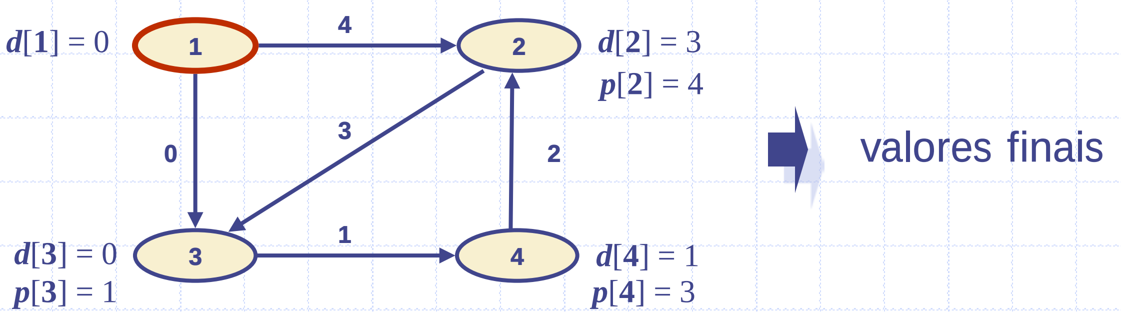
$$d^t[k] = \min_i \{ d^{t-1}[i] + c_{i,k} : (i, k) \text{ existe} \}$$

Se $d^t[k] < d^{t-1}[k]$ faça também $p[k] = \arg\min_i \{ d^{t-1}[i] + c_{i,k} : (i, k) \text{ existe} \}$

Passo 3 (Critérios de Parada): Termine se $d^t[k] = d^{t-1}[k]$ para todos os vértices k ou se $t = n$ (iterações = no. de vértices do grafo). Caso contrário faça $t = t + 1$ e volte para o Passo 1.

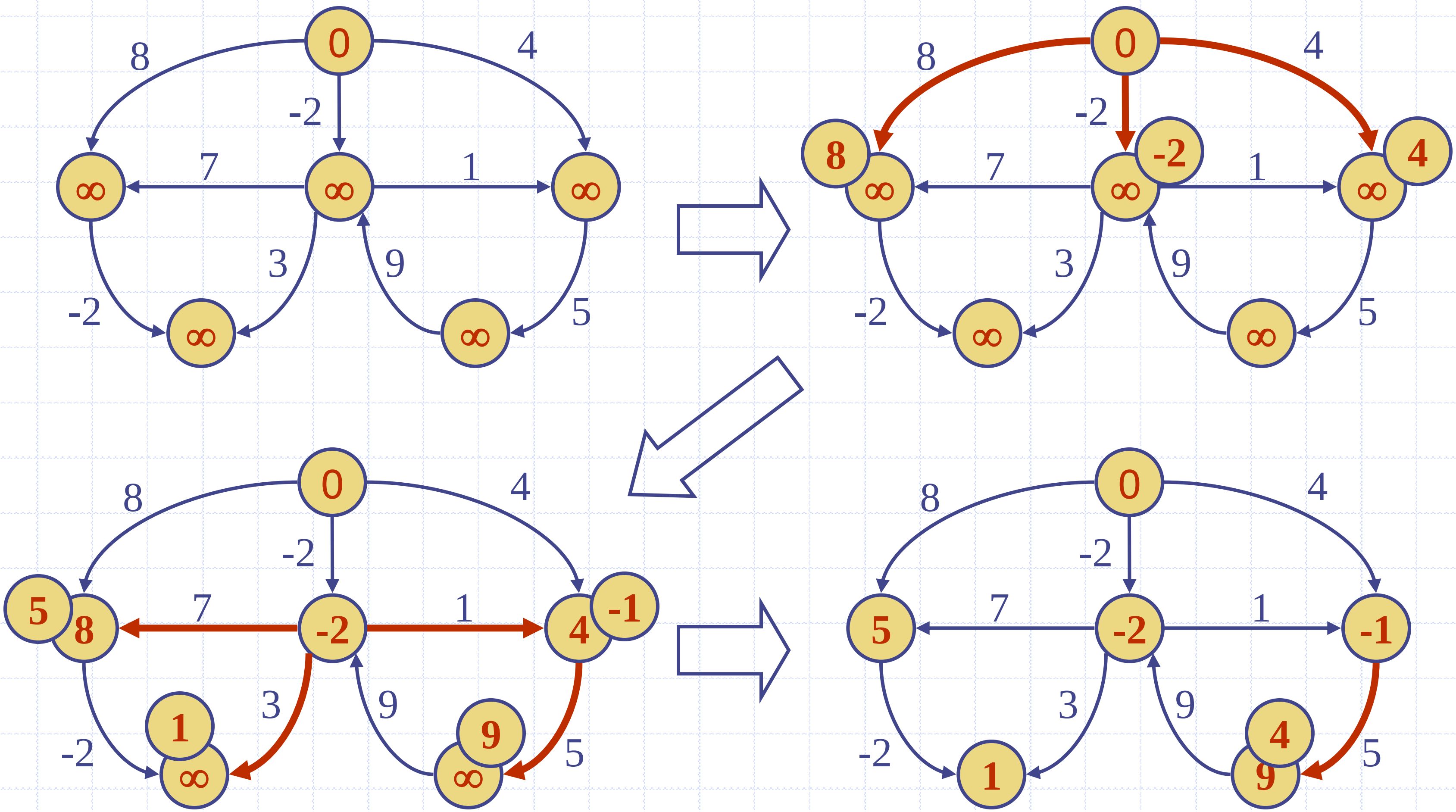
Algoritmo Bellman-Ford

- **Encontrando os Caminhos Ótimos**:
  - Os valores $d[k]$ representam os custos ou distâncias dos caminhos mais curtos a partir da origem até qualquer vértice k.
  - Os caminhos em si podem ser recuperados através dos valores auxiliares $p[k]$ também computados no passo 1 do algoritmo.
  - Para um dado vértice k, $p[k]$ é simplesmente o índice ou rótulo do vértice que precede k no caminho ótimo da origem s até k.
  - Para obter o caminho ótimo de s até k basta portanto seguir os índices p a partir de k retroativamente até s.
  - Exemplo:



Algoritmo Bellman-Ford

Exemplo: por simplicidade apenas os valores ***d*** são mostrados (nos vértices).



Algoritmo Bellman-Ford

- **Justificativa**: O que garante que os valores $d^t[k]$ são ótimos?
  - Alguns vértices estão 1 aresta distantes da origem <u>ao longo de um caminho ótimo</u>, outros 2 arestas distantes <u>ao longo de um caminho ótimo</u>, outros 3 arestas, e assim por diante.
  - Nenhum vértice está mais que $n - 1$ arestas distante ao longo de um caminho ótimo (sempre simples na ausência de diciclos negativos).
  - O valor inicial para a origem, $d^t[s] = 0$ em $t = 0$, é correto e não se altera nas iterações posteriores.
  - Após a iteração $t = 1$, todos os valores que dependem diretamente deste, ou seja, aqueles referentes aos vértices apenas 1 aresta distante de s ao longo de um caminho ótimo, estarão corretos em definitivo.
  - No geral, após $t = i$, todos os valores referentes aos vértices i arestas distantes de s ao longo de um caminho ótimo terão convergido.
  - Logo, o problema estará resolvido em no máximo $n - 1$ iterações.

Algoritmo Bellman-Ford

- **Encontrando Diciclos Negativos**:
  - A justificativa anterior garante que se o grafo não contiver diciclos negativos todos os valores terão convergido no máximo após $t = n - 1$.
  - De fato, se houver um diciclo negativo alcançável a partir do vértice de origem s, então a cada iteração o algoritmo indefinidamente encontrará um caminho de ainda menor custo através do diciclo e os valores nunca irão convergir.
  - Portanto, se algum valor ainda se alterar na iteração $t = n$, então o grafo necessariamente possui um diciclo negativo.
  - Logo, o algoritmo Bellman-Ford pode também ser utilizado para detectar a presença de diciclos negativos em grafos.

Algoritmo Bellman-Ford

- **Complexidade**:
  - Atribuição de valores iniciais aos n vértices:
    - $O(n)$
  - A cada iteração, cada uma das m arestas direcionadas é verificada uma vez (no vértice de entrada) para atualização de valores:
    - $O(m)$
  - Como são n iterações:
    - $O(nm + n) \Rightarrow O(nm)$

Algoritmo *BellmanFord*(*G, s*)

Entrada: Grafo ***G*** e um vértice ***s*** de ***G***

Saída: Grafo ***G*** modificado com cada vértice ***v*** incluindo a distância e o vértice predecessor do caminho mais curto a partir de ***s*** ; ou ***null*** se ***G*** contém diciclos negativos.

```
para todo v ∈ vertices(G)
    se v = s   v.distance ← 0
    senão
        v.distance ← ∞
        v.parent ← null
para t ← 1 até n faça
    flag_stop ← 1
    para todo v ∈ vertices(G)
        para todo e ∈ incomingEdges(G, v)
            z ← opposite(G, v, e)
            se z.distance + e.weight < v.distance
                v.distance ← z.distance + e.weight
                v.parent ← z
                flag_stop ← 0
    se flag_stop = 1 retorne G
retorne null
```

Exercícios

1. Modifique o princípio de otimalidade, as equações funcionais e o pseudo-código de Bellman-Ford para encontrar os caminhos máximos (*longest paths*) ao invés dos caminhos mínimos.
2. Seja um digrafo simples sem diciclos negativos. Se um dado vértice k desse digrafo não for alcançável a partir de um outro vértice s, qual será o valor final de $d[k]$ após a execução de Bellman-Ford com origem em s? Justifique.
3. Qual tipo de grafo leva o tempo de execução de Bellman-Ford ao pior caso possível? Justifique e apresente qual é esse tempo de pior caso em termos de complexidade assintótica.
4. Descreva um método que utilize Bellman-Ford para obter os caminhos mínimos a partir de qualquer vértice para qualquer outro vértice alcançável de um grafo (algoritmo tipo all-to-all). Qual a ordem de complexidade desse método?

Exercícios

5. Desenhe um grafo não-direcionado simples, conexo e ponderado com 8 vértices e 16 arestas. Transforme esse grafo em um digrafo e exercite o algoritmo Bellman-Ford executando-o manualmente a partir de diferentes origens:
   - Apresente cada execução através de duas matrizes, uma com as distâncias (d[k]) e outra com os vértices predecessores nos caminhos mínimos (p[k]). Nessas matrizes, cada coluna k corresponde a um vértice do grafo e cada linha t corresponde a uma iteração do algoritmo.

6. Implemente o algoritmo Bellman-Ford em C, com base na estrutura de dados para grafos alternativa discutida em aula. Valide sua implementação comparando os resultados de execução com aqueles obtidos manualmente no Exercício 5.

Referências

- M. T. Goodrich and R. Tamassia, *Data Structures and Algorithms in C++/Java*, John Wiley & Sons, 2002/2005.
- N. Ziviani, Projeto de Algoritmos, Thomson, 2a. Edição, 2004.
- T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*, MIT Press, 2^{nd} Edition, 2001.
- S. Skiena e M. Revilla, *Programming Challenges: The Programming Contest Training Manual*, Springer-Verlag, 2003.