

Algoritmos e Estruturas de Dados II

Grafos Ponderados & Caminhos Mínimos (Dijkstra)

Prof. Ricardo J. G. B. Campello

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

1

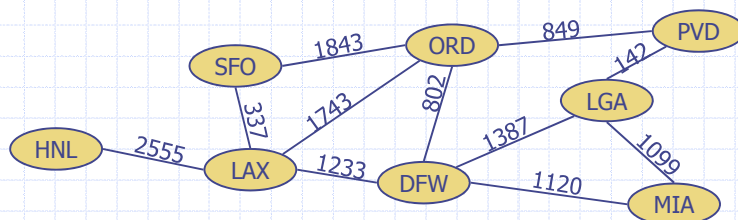
Organização

- ◆ Grafos Ponderados
- ◆ Caminhos Mínimos
 - Propriedades e Princípio da Otimalidade
- ◆ Algoritmo de Dijkstra
 - Hipóteses
 - Idéia
 - Exemplo
 - Algoritmo
 - Justificativas
- ◆ Análise de Complexidade Comparativa

2

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**, ...



3

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;
```

4

Caminhos Mais Curtos

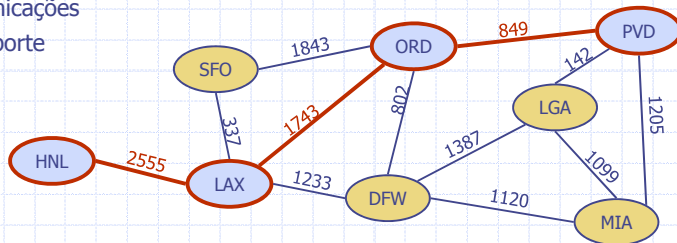
- ◆ Dado um grafo simples, direcionado e ponderado, e dois vértices u e v , tais que v é alcançável a partir de u , queremos encontrar um caminho de custo (distância) total mínimo entre u e v
 - Custo de um caminho é a soma total dos pesos das suas arestas

Exemplo:

- Caminho mais curto entre Providence (PVD) e Honolulu (HNL)

Muitas aplicações, especialmente em logística...

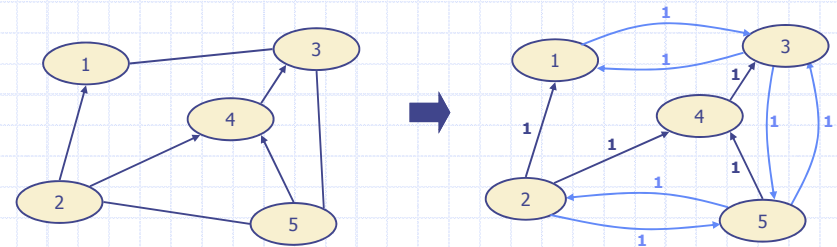
- Comunicações
- Transporte
- ...



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.

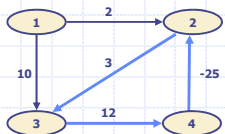


Propriedades

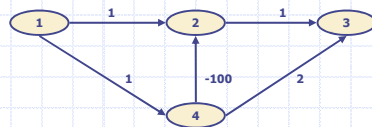
Propriedade 2 (Princípio de Otimalidade)

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).



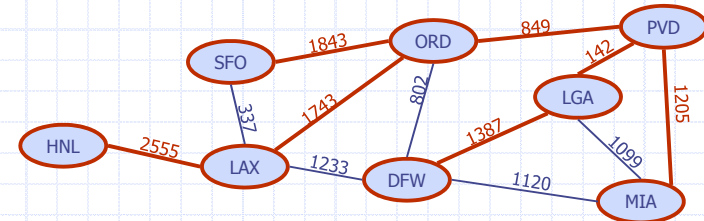
Ausência de Diciclo Negativo: 1-4-2-3 é caminho simples ótimo, assim como são 1-4, 1-4-2, 4-2, 4-2-3 e 2-3.

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):



Algoritmo de Dijkstra (one-to-all)

◆ Hipóteses sobre o grafo:

- Direcionado
- Simples*
- Ponderado apenas com **pesos não-negativos**
 - ◆ Hipótese mais restritiva que a exigida pelo princípio de otimalidade
 - ◆ Porém, não é restritiva para várias aplicações práticas

◆ Abordagem:

- **gulosa** (*greedy*)
- provada ser globalmente ótima nesse caso!

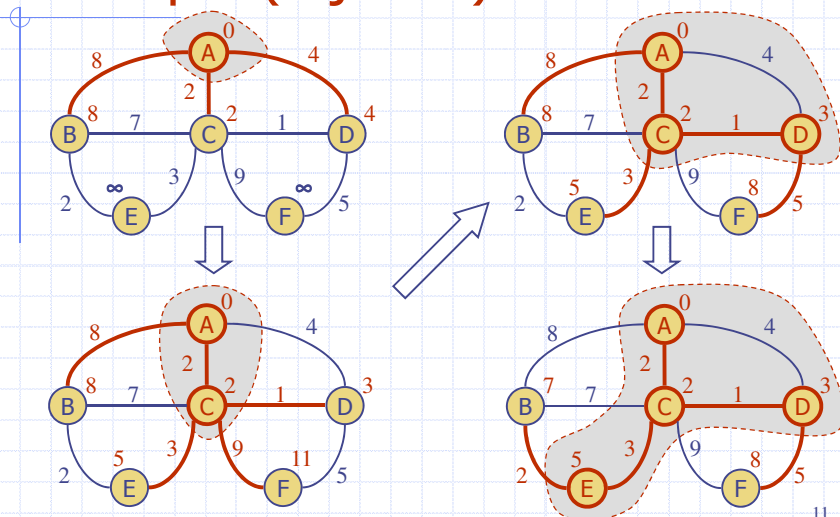
9

Algoritmo de Dijkstra (one-to-all)

◆ Idéia:

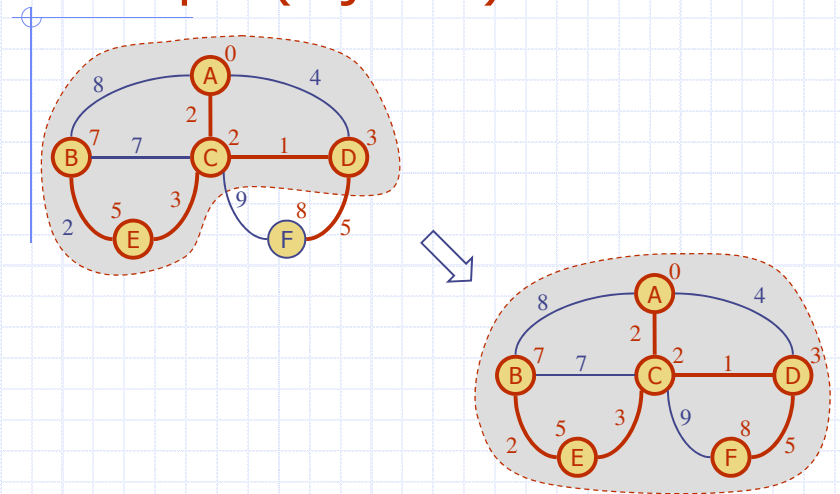
- Cada vértice v armazena a distância $d(v)$ do caminho mais curto conhecido partindo do vértice de origem s até v .
- Cresce-se a árvore de caminhos mais curtos definitivos (convergidos) como uma "nuvem", partindo de s .
- A cada iteração, um novo vértice u , adjacente a um daqueles que já convergiram (nuvem), terá também convergido.
 - ◆ Trata-se do vértice fora da nuvem com o menor valor $d(u)$
 - ◆ Esse vértice é então incluído na nuvem
- Apenas os vértices adjacentes a u que ainda não convergiram precisam ser atualizados, o que torna Dijkstra muito eficiente.
 - ◆ Algoritmo Bellman-Ford, por ex., atualiza todos a cada iteração!

Exemplo (Dijkstra)



11

Exemplo (Dijkstra)



12

Algoritmo Dijkstra

- Uma fila de prioridade Q armazena cada vértice v fora da nuvem de convergência.
- Vértice v armazena:
 - $d(v)$ (chave para Q)
 - variável $v.distance$
 - vértice predecessor no caminho mínimo
 - variável $v.parent$
- Fila de Prioridade Q :
 - insert**(Q, v, k): insere item v com chave k
 - remove**(Q): remove e retorna item com menor chave
 - replaceKey**(Q, v, k): substitui por k a chave do item v e reorganiza a fila

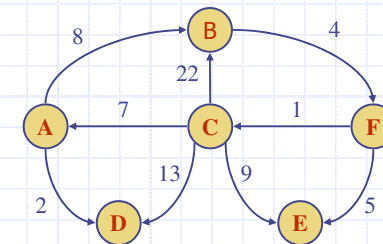
```

Algoritmo Dijkstra( $G, s$ )
 $Q \leftarrow$  nova fila de prioridade
para todo  $v \in vertices(G)$ 
    se  $v = s$      $v.distance \leftarrow 0$ 
    senão        $v.distance \leftarrow \infty$ 
    insert( $Q, v, v.distance$ )
enquanto  $\neg empty(Q)$ 
     $u \leftarrow remove(Q)$ 
    para todo  $e \in outgoingEdges(G, u)$ 
         $z \leftarrow opposite(G, u, e)$ 
        se  $z.distance > u.distance + e.weight$ 
             $z.distance \leftarrow u.distance + e.weight$ 
             $z.parent \leftarrow u$ 
            replaceKey( $Q, z, z.distance$ )
    
```

* Nota: variável $e.weight$ é o peso armazenado na aresta e

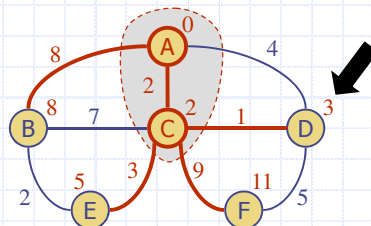
Exercício

- Execute Dijkstra no grafo direcionado abaixo a partir da origem no vértice F:
 - Ilustre o conteúdo da fila de prioridade como uma lista ordenada a cada iteração.



Justificativa (Dijkstra)

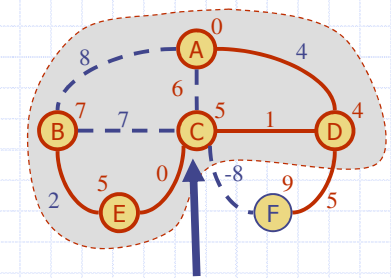
- Dijkstra é baseado na abordagem gulosa de adicionar vértices por distância crescente:
 - A menor distância $d(u)$ fora da nuvem é certamente definitiva pois qualquer outro caminho só com pesos não negativos não pode ser menor!



Porque Dijkstra Não Funciona com Pesos Negativos ?



- Se um vértice com aresta incidente negativa pudesse ser adicionado à nuvem de convergência, ele poderia modificar as distâncias de vértices já dentro da nuvem.



Menor distância de C a partir da origem A é 1, mas C já está na nuvem com $d(C)=5$!

Desempenho

◆ Dijkstra:

- É possível implementar de forma eficiente como:
 - ◆ $O((n+m) \log n)$
- Alternativamente, pode-se substituir a fila de prioridade por uma busca seqüencial em uma lista:
 - ◆ $O(n^2 + m)$
 - ◆ Pode ser mais rápida para grafos densos (m grande).
 - ◆ Implementação em C: ver (Skiena & Revilla, 2003).
- Obtenção de todos os caminhos (**all-to-all**):
 - ◆ n execuções de Dijkstra, uma para cada vértice como origem
 - ◆ $O((n^2 + nm) \log n)$ ou $O(n^3 + nm)$

17

Comparações

◆ Bellman-Ford (one-to-all):

- Complexidade (tempo): $O(nm)$
- Tipicamente mais lento que Dijkstra, mas menos restritivo, no sentido que **restringe apenas diciclos negativos**
- É capaz de detectar naturalmente a presença de tais diciclos
- Facilmente adaptável para caminhos máximos

◆ Floyd-Warshall (all-to-all):

- Complexidade (tempo): $O(n^3)$
- Tipicamente mais rápido que múltiplas execuções de Dijkstra
- Também **restringe apenas diciclos negativos**
- Matriz de distâncias indica todas alcançabilidades entre vértices

Exercícios

1. Compare a complexidade computacional dos algoritmos de Dijkstra e Bellman-Ford quando o grafo for:
 - esparso (m proporcional a n)
 - denso (m proporcional a n^2)
2. 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 Dijkstra 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
 - Apresente também a árvore de caminhos mínimos resultante de cada execução
3. Retire algumas arestas do grafo do exercício anterior de forma tal que o digrafo permaneça fortemente conexo e cada par de vértices adjacentes só possua uma aresta direcionada incidente em ambos. Então repita o Exercício 2.

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, 2nd Edition, 2001.
- ◆ S. Skiena e M. Revilla, *Programming Challenges: The Programming Contest Training Manual*, Springer-Verlag, 2003.

20