Estruturas de Dados

Grafos VIII:

Árvores Geradoras Mínimas

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).

Organização

- Árvores Geradoras Mínimas
- Propriedades de Árvores Geradoras Mínimas
  - Propriedade de Ciclo
  - Propriedade de Partição
- Algoritmo Prim-Jarník
- Desempenho e Comparações
- Outras Árvores Geradoras

Árvores Geradoras Mínimas

Subgrafo Gerador:

- Subgrafo de um grafo *G* contendo todos os vértices de *G*.

Árvore Geradora:

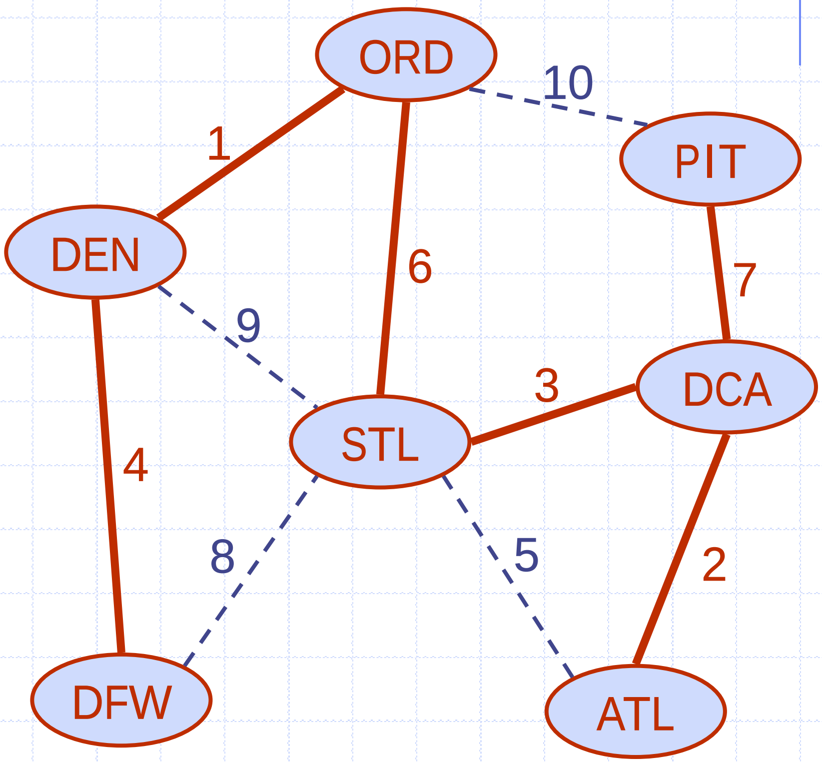
- Subgrafo gerador conexo e sem ciclos.

Árvore Geradora Mínima (**MST**):

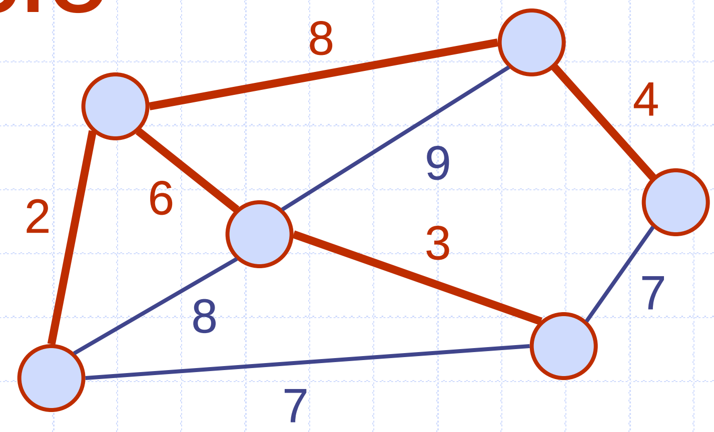
- Árvore geradora de um grafo ponderado com mínimo peso total de arestas.

Exemplos de Aplicações:

- Circuitos Integrados.
- Redes de Comunicações.
- ...



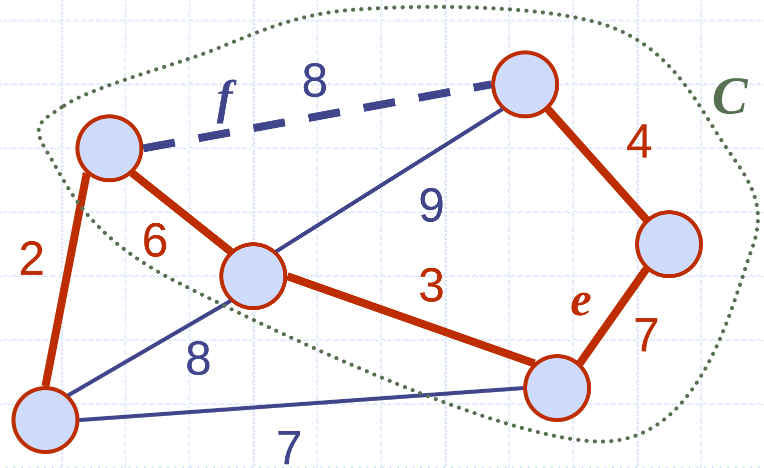
Propriedade de Ciclo



Propriedade:

- Seja T uma MST de um grafo ponderado G.
- Sejam e uma aresta de G tal que $e \notin T$ e C um ciclo formado por e com T.
- Para cada aresta f de C, $peso(f) \leq peso(e)$.

Substituir f por e produz uma árvore geradora melhor



Prova:

- Por contradição.
- Se $peso(f) > peso(e)$ então pode-se obter uma árvore geradora de menor peso total substituindo f por e.

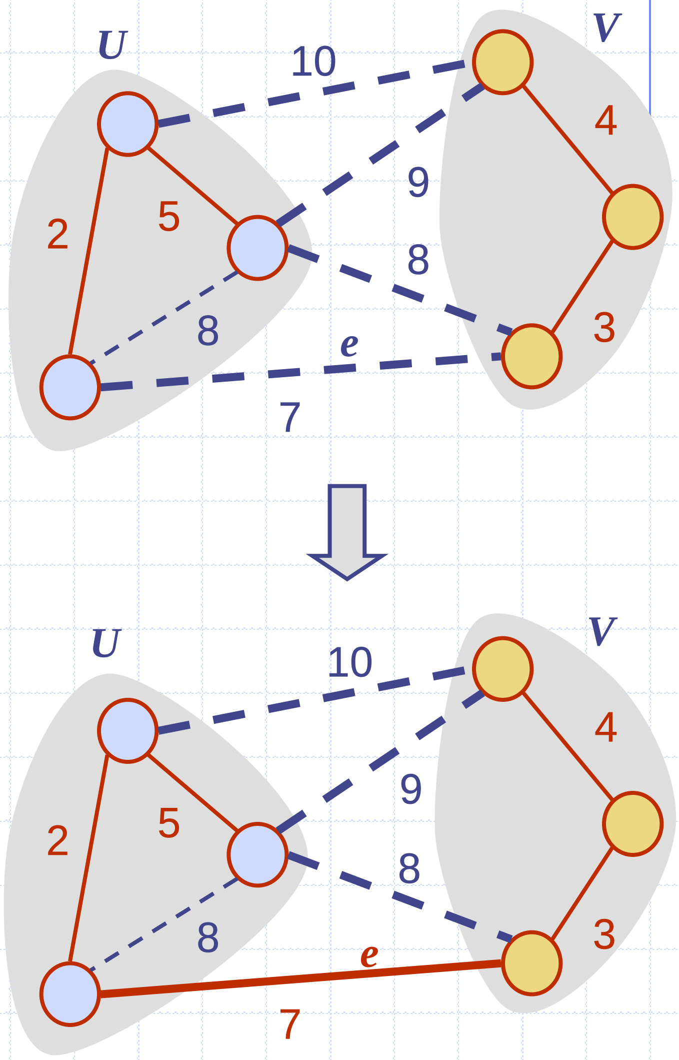
Propriedade de Partição

Propriedade:

- Considere uma partição dos vértices de G em sub-conjuntos disjuntos U e V.
- Seja e a aresta de peso mínimo ligando U e V.
- Existe uma MST de G que contém a aresta e.

Prova:

- Também por contradição.
- Utiliza a propriedade de ciclo...



Propriedade de Partição

Importância:

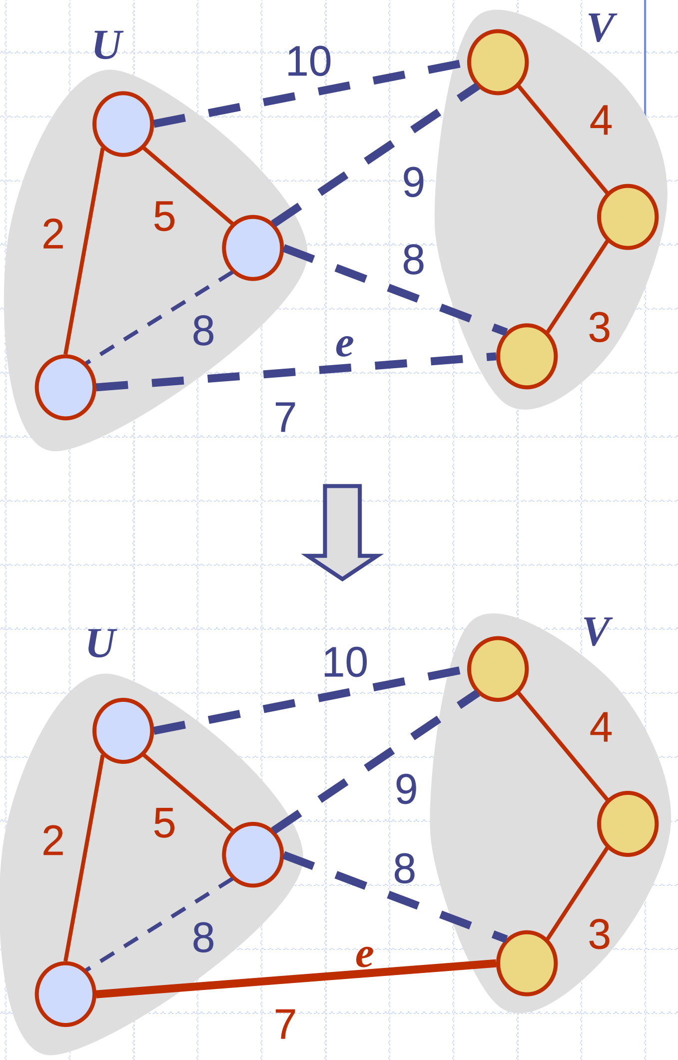
- Podemos crescer MSTs unindo sub-árvores geradoras mínimas através da aresta de peso mínimo.

Principais Algoritmos:

- **Prim-Jarník**: sub-árvore geradora única.
- **Kruskal**: múltiplas sub-árvores geradoras.

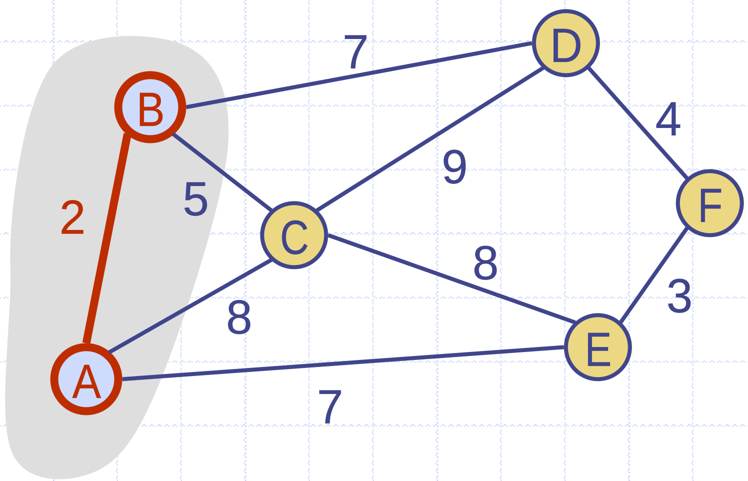
Observações:

- Por simplicidade os algoritmos serão discutidos para grafos não direcionados e simples (usual em aplicações de MSTs).
- Pesos das arestas podem ser negativos.



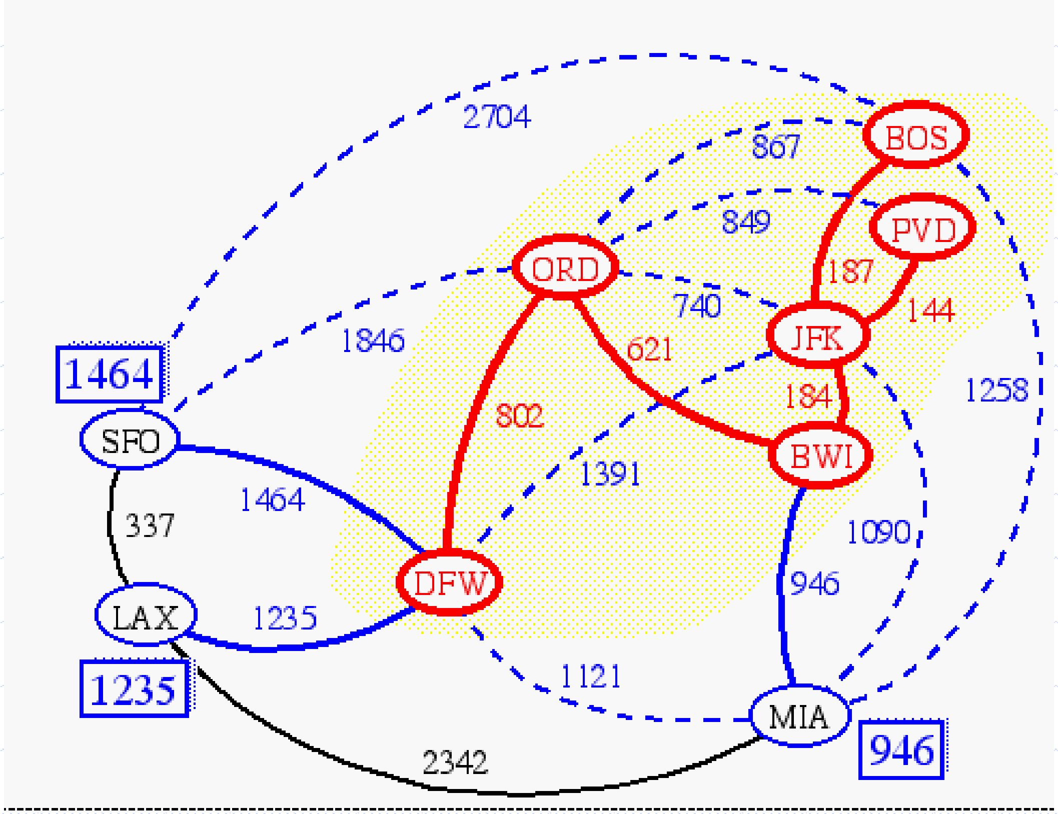
Algoritmo Prim-Jarník

- Muito similar ao algoritmo de Dijkstra.
- Toma um vértice arbitrário s e cresce a MST como uma “nuvem” de vértices, partindo de s.
- Utiliza a propriedade de partição vendo o grafo como particionado em dois conjuntos de vértices, aqueles dentro e fora da MST.
- Em cada passo, adiciona-se à MST:
  - a menor aresta ligando a MST ao seu exterior
  - o vértice u oposto através dela



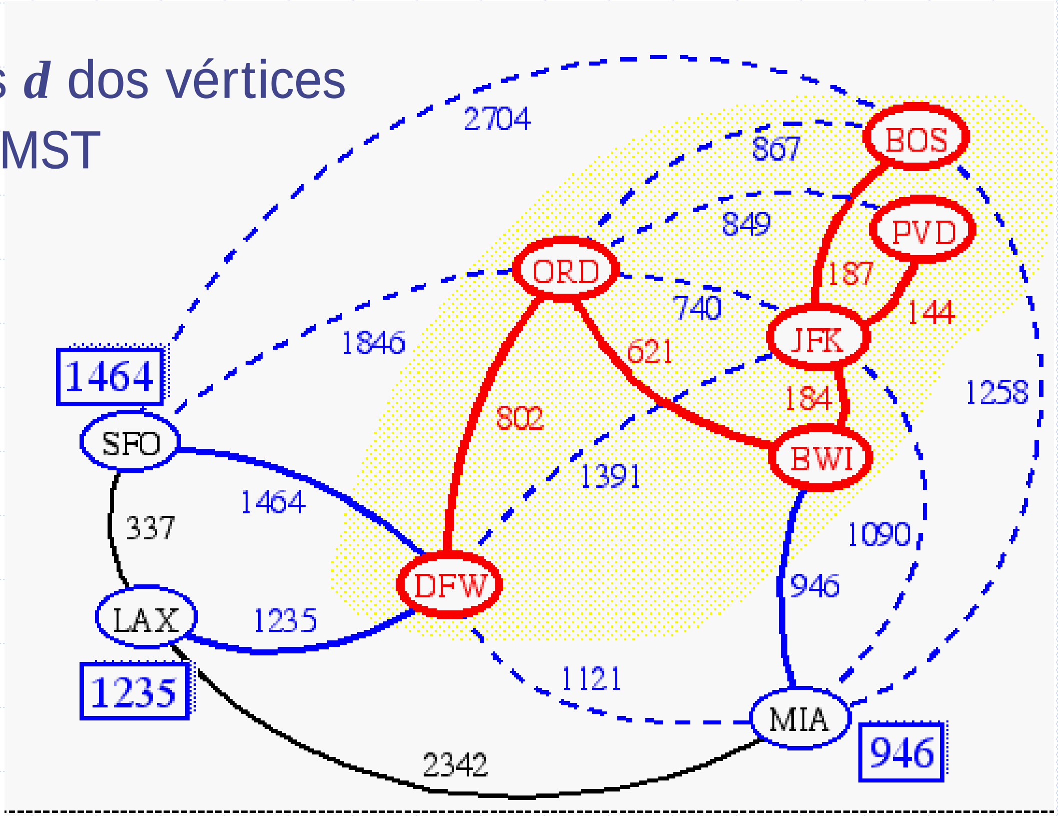
Algoritmo Prim-Jarník

- **Implementação Eficiente**:
  - armazena em cada vértice $\boldsymbol{v}$ um valor $\boldsymbol{d}(\boldsymbol{v})$
  - $\boldsymbol{d}(\boldsymbol{v})$ = menor peso de uma aresta conectando $\boldsymbol{v}$ a um vértice na MST



Algoritmo Prim-Jarník

- Em Cada Passo:
  - Adiciona-se à MST o vértice ***u*** com o menor valor ***d*** dentre os vértices de fora da MST
  - Atualiza-se os valores ***d*** dos vértices adjacentes a ***u*** fora da MST

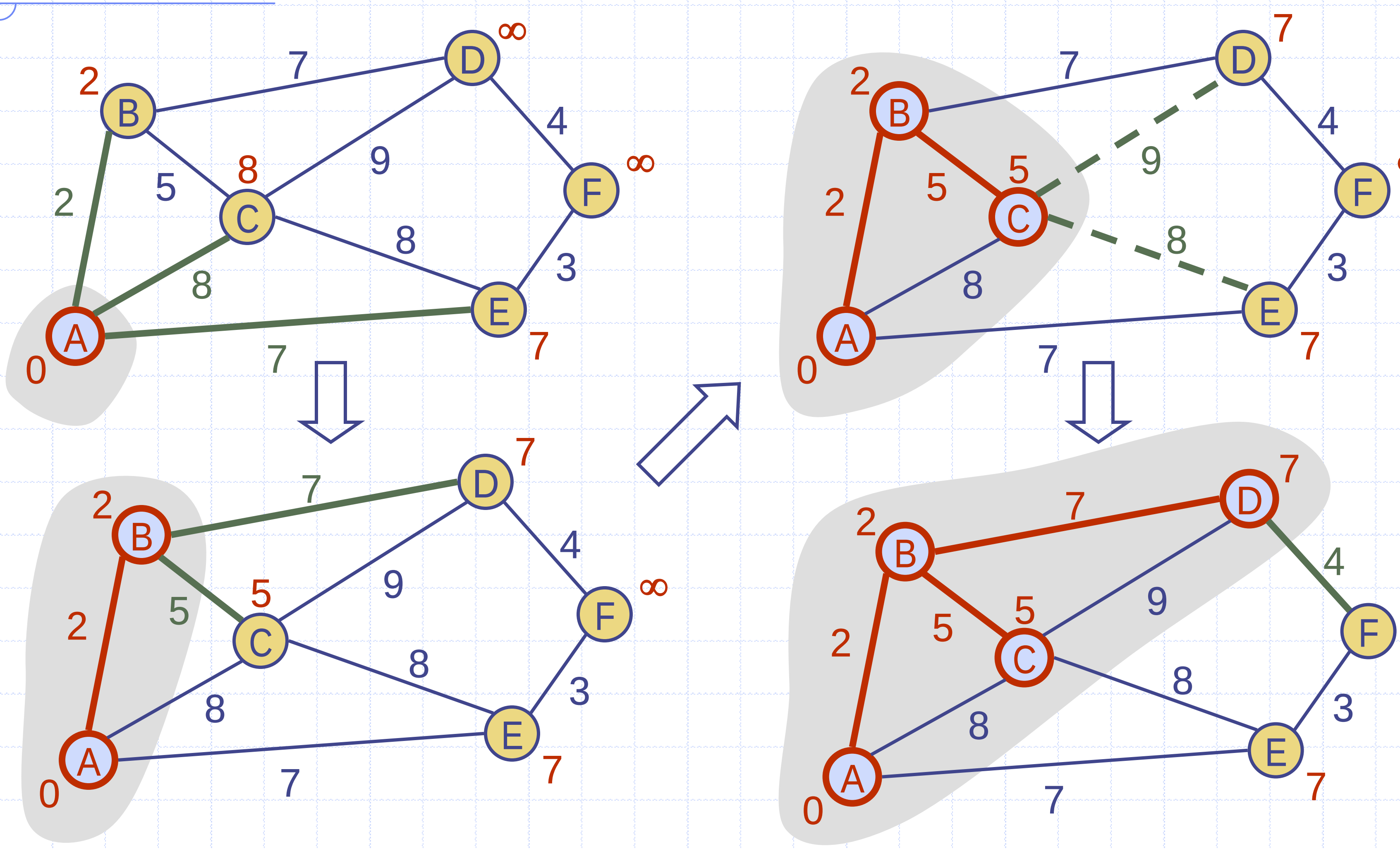


Algoritmo Prim-Jarník

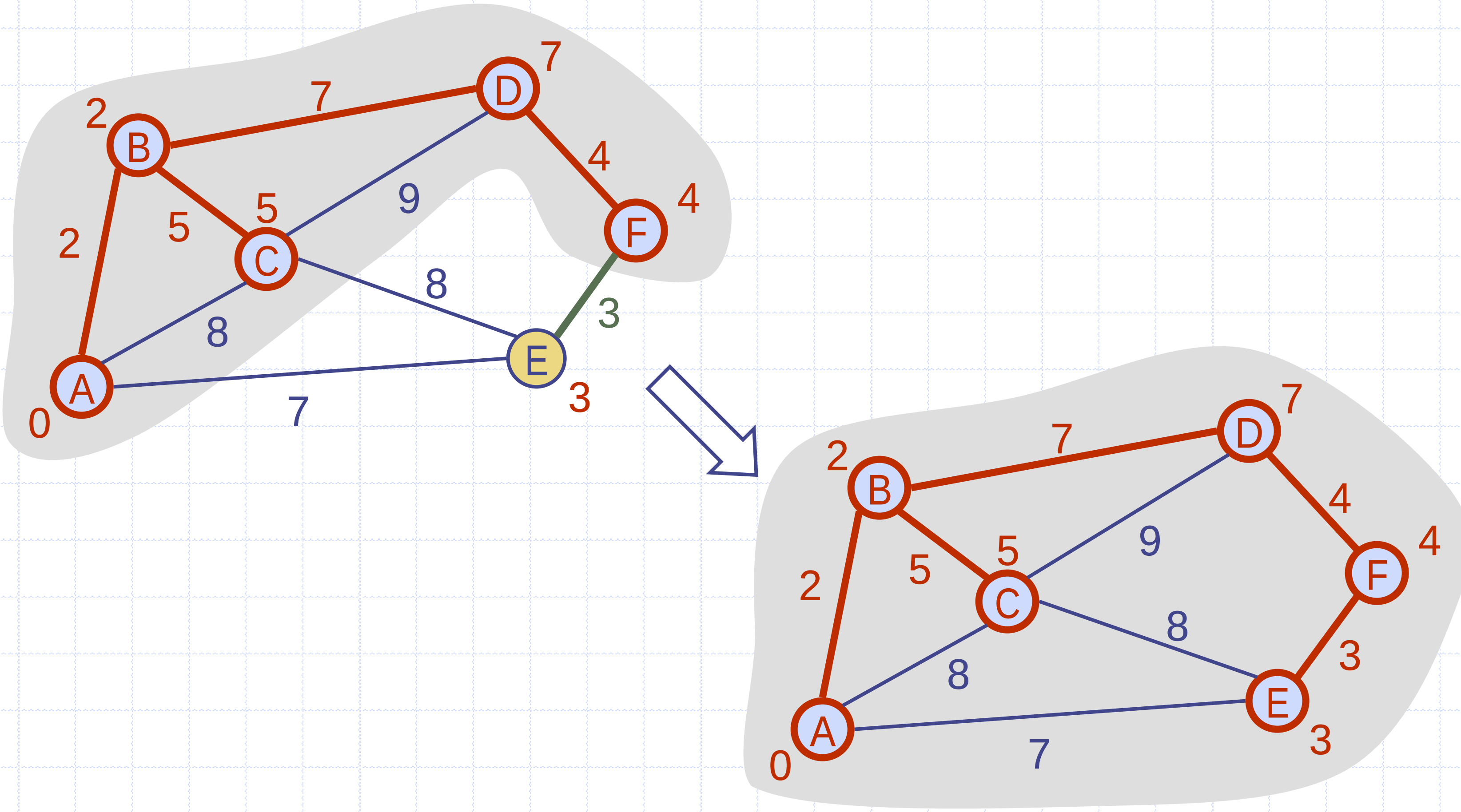
- Uma fila de prioridade Q armazena cada vértice v fora da nuvem de convergência.
- Vértice v armazena:
  - $d(v)$ (chave para Q)
  - vértice predecessor na MST
- Fila de Prioridade Q:
  - **insert**(Q, v, k): insere item v com chave k.
  - **remove**(Q): remove e retorna item com menor chave
  - **replaceKey**(Q, v, k): substitui por k a chave do item v e reorganiza a fila.

```
Algoritmo PrimJarník(G, s)
  Q ← nova fila de prioridade
  para todo v ∈ vertices(G)
     se v = s      v.distance ← 0
     senão         v.distance ← ∞
     v.label ← Fora      // da nuvem
     insert(Q, v, v.distance)
  enquanto ¬ empty(Q)
    u ← remove(Q)
    v.label ← Dentro   // da nuvem
    para todo e ∈ incidentEdges(G, u)
      z ← opposite(G, u, e)
      se z.label = Fora  E  z.distance > e.weight
         z.distance ← e.weight
         z.parent ← u
         replaceKey(Q, z, z.distance)
```

Exemplo



Exemplo



Desempenho e Comparações

- Prim-Jarník é uma pequena modificação de **Dijkstra**.
  - A análise de complexidade de ambos é praticamente idêntica
  - Pode ser implementado com tempo de execução $O(\,(n + m)\log n\,)$:
- Esse tempo de pior caso é o mesmo do algoritmo de **Kruskal**.
- Em termos de fatores constantes (**Prim-Jarník vs. Kruskal**):
  - ambos os algoritmos são bastante similares, com fatores baixos
  - ambos apresentam desempenho similar na prática
- No entanto, Prim-Jarník apresenta maior simplicidade em EDs:
  - utiliza apenas uma fila de prioridade.

Desempenho e Comparações

- Versão mais simples (**Prim**):
  - Armazena os vértices v fora da nuvem de convergência em uma lista
  - Faz busca seqüencial na lista para encontrar aquele com menor $d(v)$
    - $O(n^2 + m)$
  - Implementação em C: ver (Skiena & Revilla, 2003).
- **Nota**:
  - Algoritmos Prim-Jarník, Prim e Kruskal são dos poucos **algoritmos gulosos** que levam a soluções globalmente ótimas:
    - Estratégia de gula conduz a MSTs!

Outras Árvores Geradoras

- Árvores Geradoras de Gargalo Mínimo:
  - Árvore geradora cuja aresta de maior peso é a menor possível.
  - Qualquer MST possui essa propriedade!
- Árvores Geradoras Máximas:
  - Árvore geradora com maior peso total de arestas
  - Corresponde à MST do grafo com pesos com sinais invertidos
  - Dado que os algoritmos para MSTs podem ser aplicados a grafos com pesos negativos, basta aplicá-los ao grafo modificado

Exercícios

1. Existem 8 ilhas em um lago e deseja-se construir sete pontes para conectá-las de forma que cada ilha possa ser alcançada a partir de cada outra. O custo de construir uma ponte é proporcional ao seu comprimento. As distâncias entre os pares de ilhas são dados na seguinte tabela:

-	240	210	340	280	200	345	120
-	-	265	175	215	180	185	155
-	-	-	260	115	350	435	195
-	-	-	-	160	330	295	230
-	-	-	-	-	360	400	170
-	-	-	-	-	-	175	205
-	-	-	-	-	-	-	305
-	-	-	-	-	-	-	-

Mostre como usar Prim-Jarník para responder quais as pontes que minimizam o custo total de construção? Qual é esse custo?

Exercícios

2. Desenhe um grafo não-direcionado simples, conexo e ponderado com 8 vértices e 16 arestas. Exercite o algoritmo Prim-Jarník executando-o manualmente a partir de diferentes origens:
   - Apresente cada execução através de duas matrizes, uma com os valores d[k] e outra com os vértices predecessores na MST (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 MST resultante de cada execução
3. O algoritmo de Prim-Jarník em princípio assume que o grafo é simples, o que implica que não existem arestas paralelas. Como modificar um grafo com arestas paralelas de tal forma que este se torne um grafo simples e uma MST no grafo modificado seja também uma MST no grafo original ?
4. O que é preciso modificar em Prim-Jarník para que este opere em digrafos? Qual propriedade deve possuir o digrafo para que exista uma MST a partir de um vértice de origem s ?

Bibliografia

- 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.