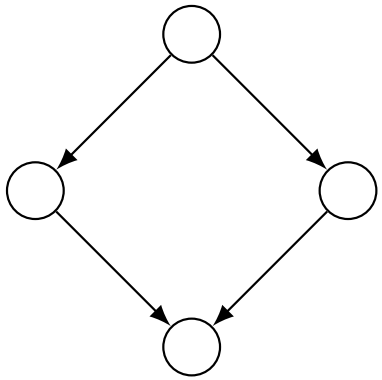
Grafos

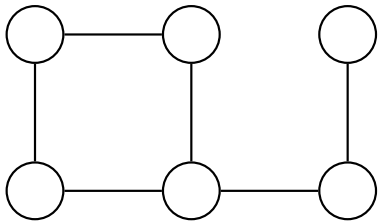
Fabio Gagliardi Cozman

PMR2300
Escola Politécnica da Universidade de São Paulo

Grafos

- Um grafo é uma estrutura que generaliza árvores, sendo formado por nós e arestas.
- Cada nó em um grafo pode ser conectado a vários outros nós por meio de arestas.
- Grafos são usados para representar redes rodoviárias, eletrônicas ou de fornecimento de energia; modelos biológicos, sociais, físicos...



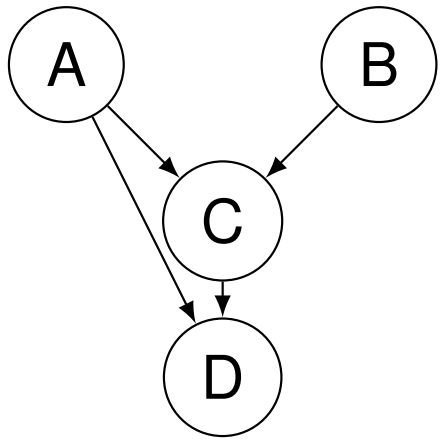


Definições

- Um *caminho* é uma sequência de arestas que vai de um nó a outro.
- Dois nós são *adjacentes* ou *vizinhos* se são conectados por arestas.
- Se todas as arestas têm direção (neste caso são chamadas *arcos*), o grafo é *direcionado*.
- Um grafo direcionado é *acíclico* quando não existe um caminho direcionado que parte de um nó e retorna ao mesmo nó.
- Quando um arco conecta dois nós A e B, partindo de A para B, o nó A é *pai* (ou origem do arco) e o nó B é *filho* (ou destino do arco).
- Se todas as arestas não têm direção, o grafo é *não-direcionado*.

Implementação: Lista de arestas

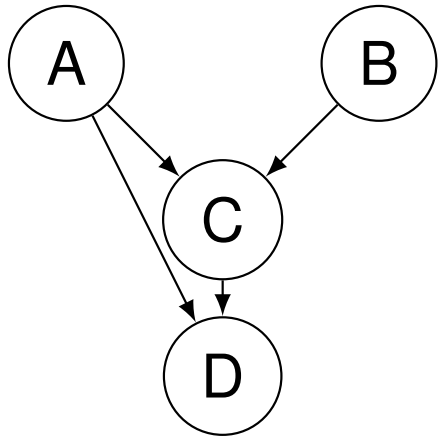
- Armazenamos cada nó como um objeto.
- Armazenamos cada aresta como um objeto contendo
  - nós origem e destino;
  - indicação sobre direcionalidade.
- Exemplo:



nós: A, B, C, D
arestas: (A,C,$\rightarrow$), (B,C,$\rightarrow$), (C,D,$\rightarrow$), (A,D,$\rightarrow$)

Implementação: Lista de Adjacências

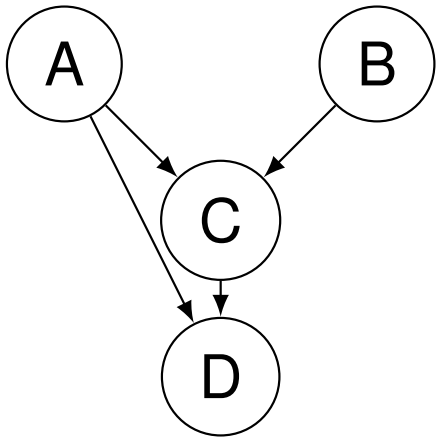
- Adicionamos, a cada nó, uma lista na qual estão contidos seus vizinhos, ou seus filhos, dependendo do que for mais adequado para a aplicação.
- Exemplo:



A $\rightarrow$ C, D
B $\rightarrow$ C
C $\rightarrow$ D
D $\rightarrow$

Implementação: Matriz de Adjacências

- Matriz armazena no elemento *ij* uma referência à aresta entre o nó *i* e o nó *j* (se a aresta existe), ou apenas indica presença/ausência de arestas.
- Exemplo:



$$\begin{bmatrix} \emptyset & \emptyset & e_{AC} & e_{AD} \\ \emptyset & \emptyset & e_{BC} & \emptyset \\ \emptyset & \emptyset & \emptyset & e_{CD} \\ \emptyset & \emptyset & \emptyset & \emptyset \end{bmatrix}, \text{ ou simplesmente } \begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}.$$

Busca

- É muito comum desejarmos encontrar o nó que contém um dado particular.

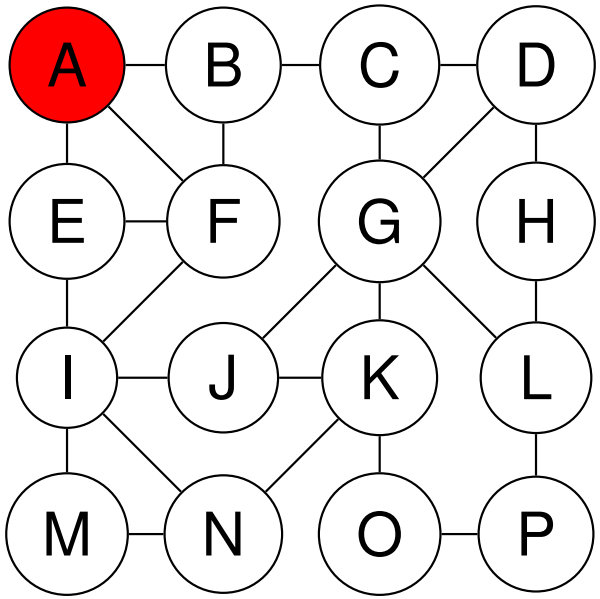
- Supondo que a busca pelo dado *d* começa em um nó *n*, existem duas maneiras básicas de realizar esse tipo de busca:
  - busca em profundidade;
  - busca em largura.

Busca em profundidade

- A partir de um nó, escolhemos um vizinho não-visitado e continuamos a busca nesse vizinho, marcando antes o nó como visitado.
- Se chegarmos a um ponto sem vizinhos não-visitados, retornamos a um nó que não tenha todos os vizinhos visitados.
- Algoritmo:

Depth-First-Search(grafo *G*, nó *n*, dado *d*):
    se dado *d* está em *n*, retorne *n* (sucesso!);
    marque *n* como visitado;
    para todos os vizinhos de *n*:
        se o vizinho *m* nao foi visitado,
            retorne DFS(*G*,*m*,*d*) se houve sucesso;
    retorne indicação de falha na busca.

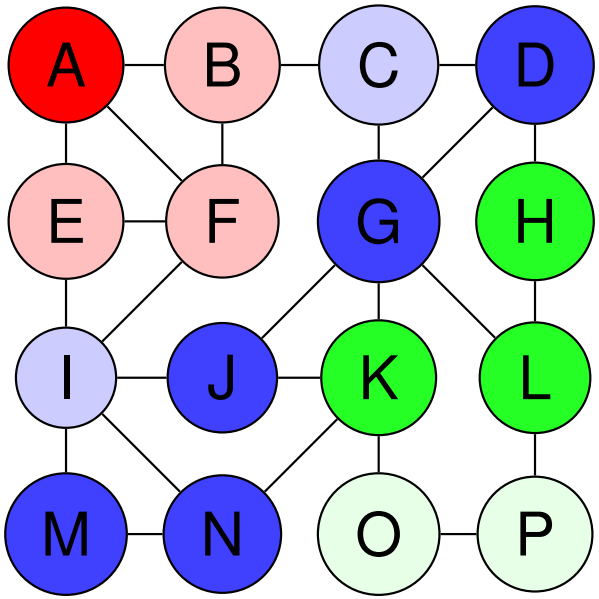
Busca em profundidade



Busca em Largura

- Nesse caso, o grafo é dividido em níveis a partir de um nó de onde a busca está sendo iniciada.
- O nó inicial está no nível 0; seus vizinhos estão no nível 1; os vizinhos destes nós que não foram visitados estão no nível 2.
- Em geral, o nível L_i é formado pelos vizinhos de nós em L_i que não estejam nos níveis $L_0, L_1, \cdots, L_{i-1}$.

Busca em Largura

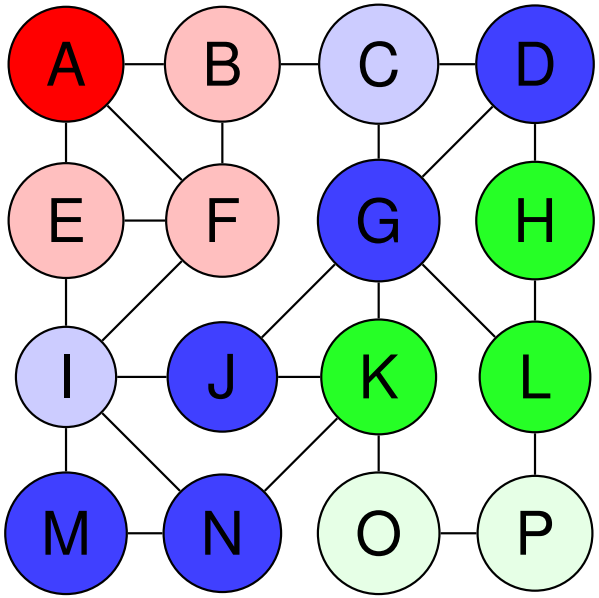


Busca em Largura

O código básico que realiza busca em largura usa lista auxiliar para armazenar os níveis do grafo:

Breadth-First-Search(grafo *G*, nó *n*, dado *d*):
    crie lista *L*, contendo nó *n*;
    enquanto *L* contiver algum nó, repita:
        retire o primeiro nó de *L*, chame esse nó de *m*;
        se dado *d* está em *m*, retorne *m* (sucesso!);
        caso contrário,
            marque *m* como visitado;
            coloque vizinhos de *m* no *final* de *L*
                (apenas vizinhos não-visitados e não em *L*!);
    retorne indicação de falha na busca.

Busca em Largura



Temos: $L = \{A\}$; $L = \{B, E, F\}$; $L = \{E, F, C\}$; $L = \{F, C, I\}$; $L = \{C, I\}$; $L = \{I, D, G\}$; $L = \{D, G, J, M, N\}$; etc

Caminho Mínimo

- Um grafo ponderado é um grafo que tem um peso numérico $w(e)$ associado a cada aresta e.
- O *comprimento* de um caminho c é a soma dos pesos nas arestas em c:

$$w(c) = \sum_{i=0}^{k-1} w((n_c, n_{c+1}))$$

com $c = ((n_0, n_1), \cdots, (n_{k-1}, n_k))$.

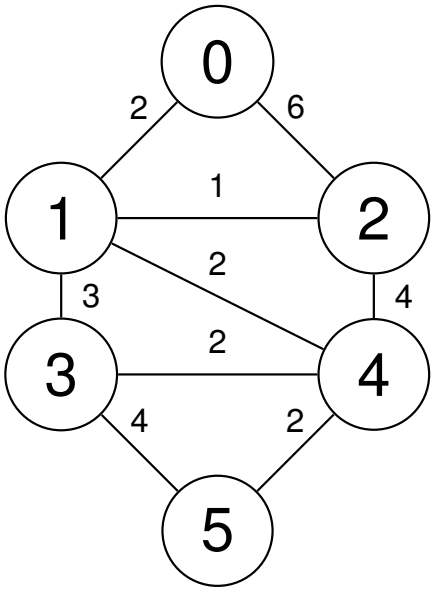
- A distância entre dois nós é o comprimento mínimo entre eles.
- Note: Vamos considerar apenas o caso em que todos os pesos são positivos ($w(\cdot) > 0$)!

Algoritmo de Dijkstra

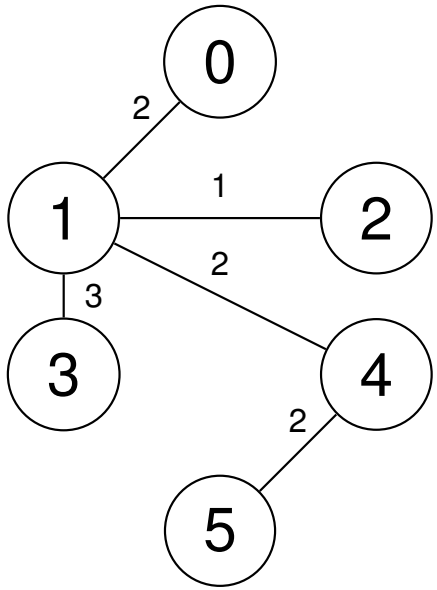
- O algoritmo é basicamente uma busca em largura "ponderada", começando a partir da origem *n* do caminho procurado.

- A busca se expande em torno de *n* examinando nós em ordem de distância.

Algoritmo de Dijkstra



... solução:



Exemplo — Algoritmo de Dijkstra

- Vamos supor que todas as arestas são não-direcionadas e que não há um nó com aresta para ele mesmo nem pares de nós com arestas "paralelas" entre eles.
- Definimos uma função $D(v)$ que retorna a distância entre n e v encontrada até o momento.
  - Inicialmente, $D(n) = 0$ e $D(v) = \infty$ para $v \neq n$.
- Também definimos um conjunto C contendo os nós já explorados. No início esse conjunto é vazio.

Algoritmo de Dijkstra

1. A cada iteração, colocamos em C um vértice v (ainda não em C) que tenha menor $D(v)$.
2. Neste ponto atualizamos $D(z)$ para todo nó z que é vizinho de v e não está em C:
   - se $D(v) + w((v,z)) < D(z)$, então

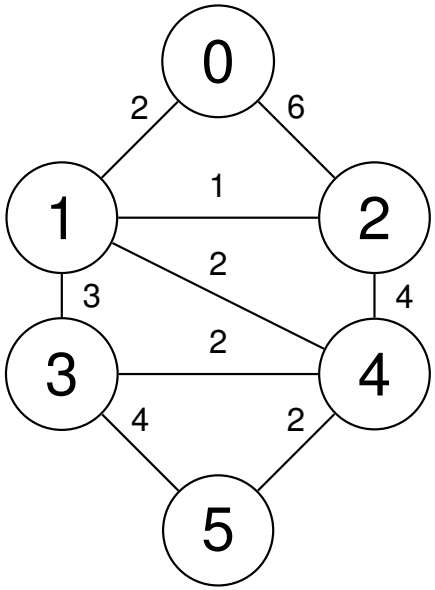
$$D(z) \leftarrow D(v) + w((v,z)).$$

Implementação (grafo não-direcionado, conectado, pesos não-negativos, sem arestas paralelas)

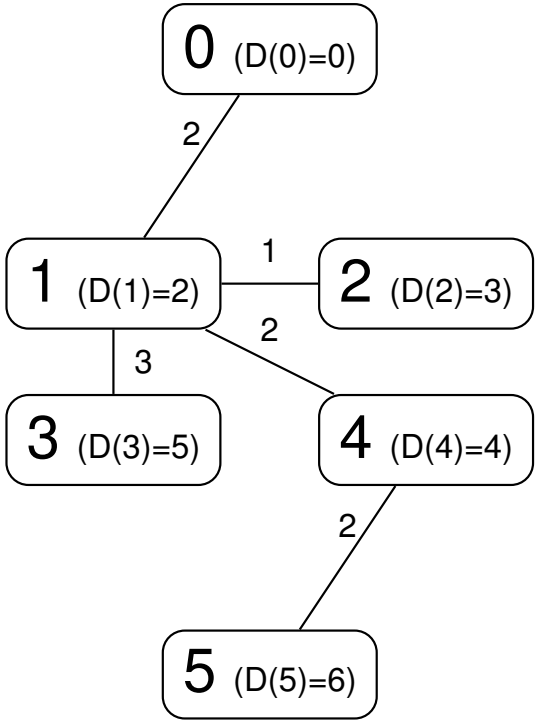
Dijkstra(grafo G, nó n, nó m)
    $D(n) = 0$;
    $D(v) = \infty$ para $v \neq n$;
    crie lista C vazia;
    enquanto houver algum nó fora de C, repita:
        insira em C o nó v com menor valor de $D(\cdot)$;
        se $v = m$, retorne;
        para cada vizinho z de v e que não está em C:
            se D(v) + w((v,z)) < D(z),
                então D(z) = D(v) + w((v,z)).

Ao executar esse algoritmo, devemos marcar as arestas que conectam nós à medida que nós são inseridos em C.

Exemplo

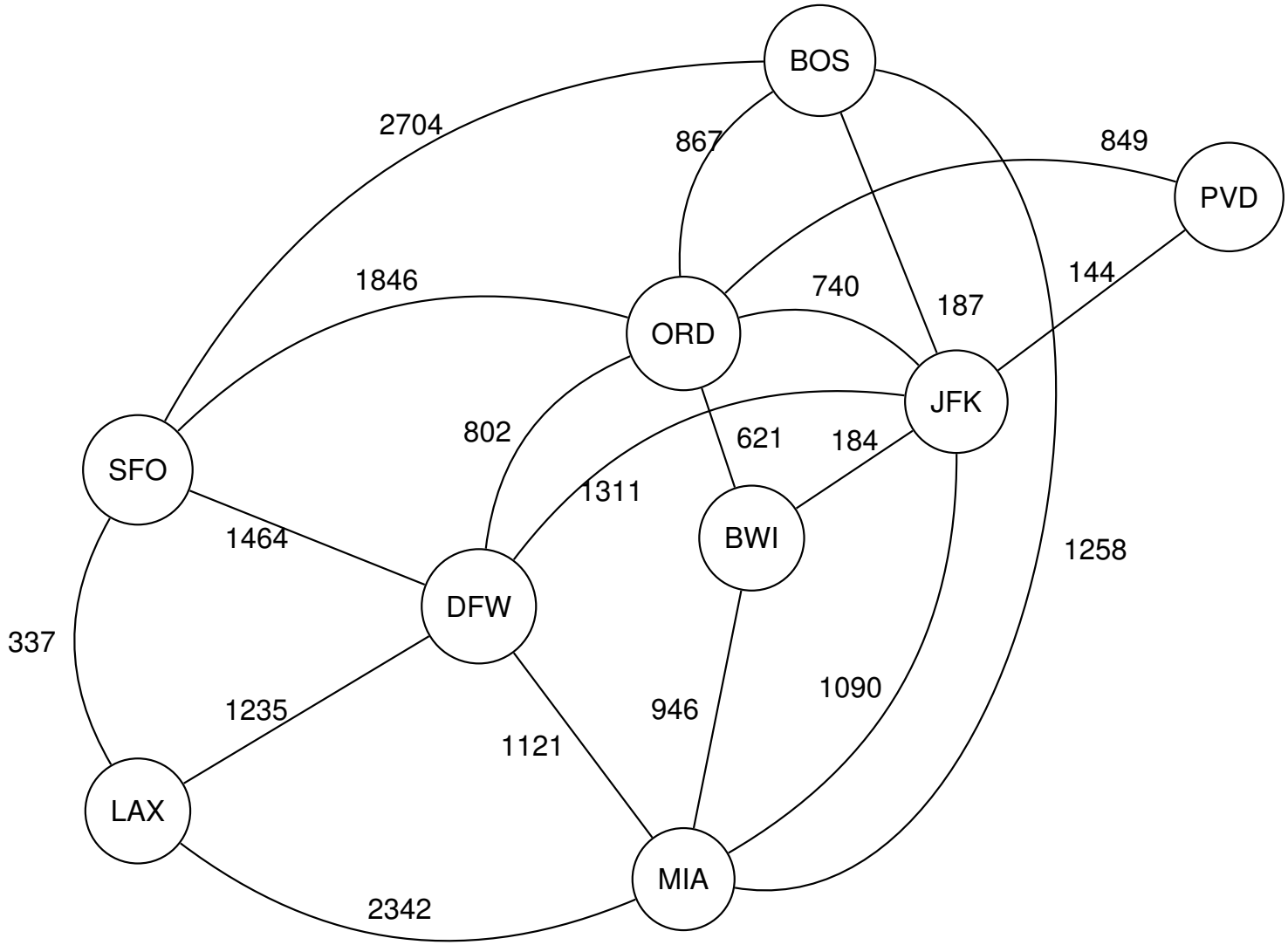


... solução:



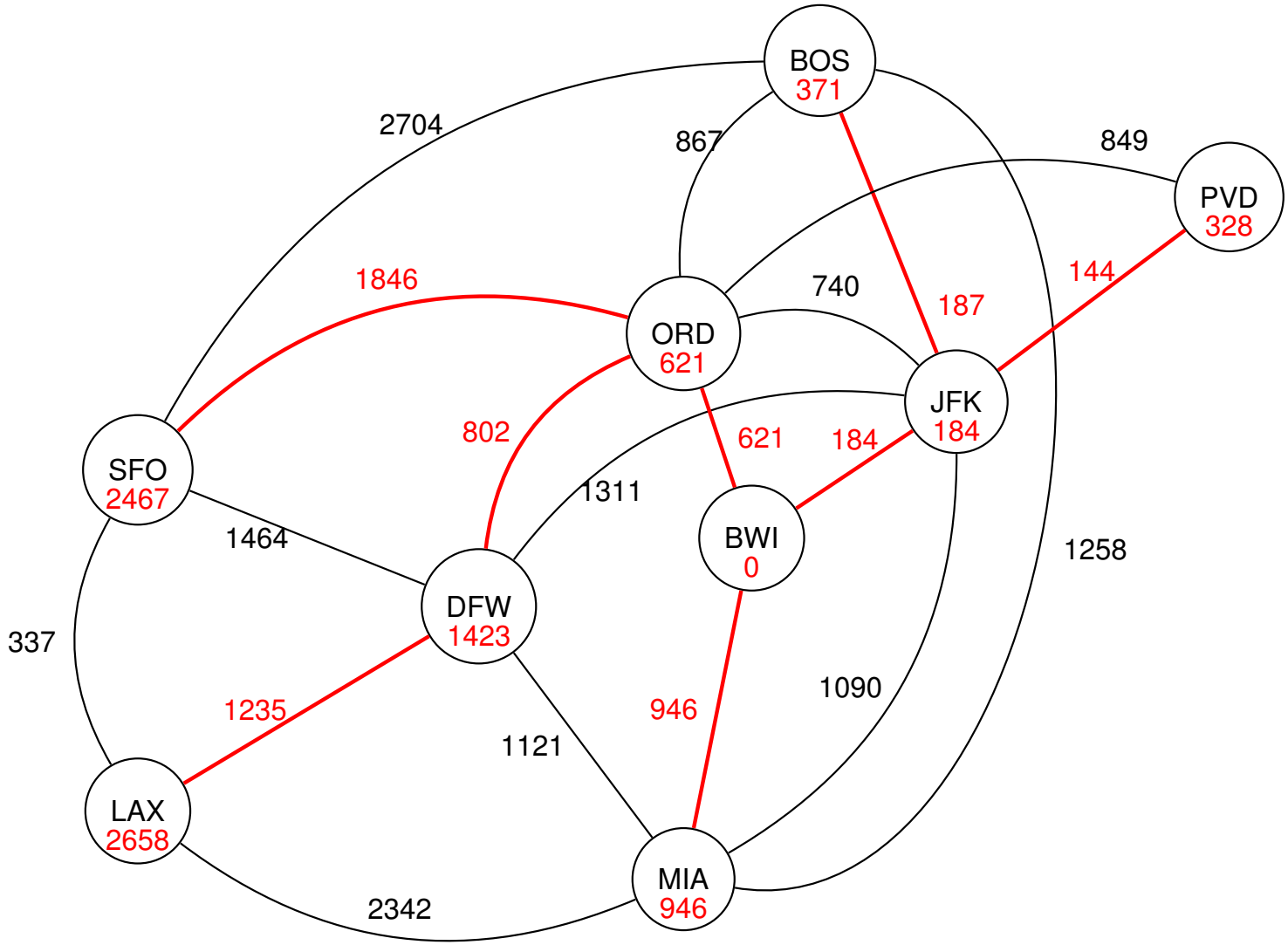
Exemplo: melhor caminho entre aeroportos nos EUA

Partindo do nó BWI.



Exemplo: melhor caminho entre aeroportos nos EUA

Partindo do nó BWI.



Note

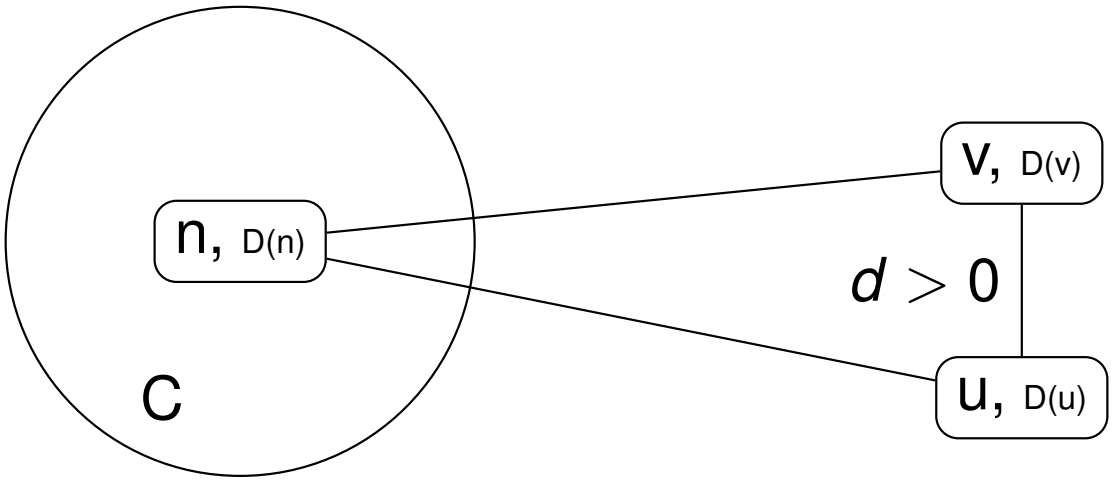
- Quando um nó v é colocado em C, seu rótulo $D(v)$ contém a distância correta entre n e v.

- Se algoritmo passar por todos os nós, então ao final todos os caminhos mínimos partindo de n estarão determinados.

Demonstração

- Demonstraremos que quando nó entra em C, $D(v)$ está correto.
- Usaremos indução finita.
- A sentença está correta para o primeiro nó, que é n com $D(n) = 0$.
- Assuma que os primeiros N nós colocados em C tenham $D(.)$ correto. Considere agora o nó v que tem menor $D(.)$ dentre os nós fora de C.
- Suponha (para obter uma contradição) que haja um caminho entre n e v de comprimento menor que $D(v)$.

Demonstração



Suponha (para obter uma contradição) que haja um caminho entre n e v de comprimento menor que $D(v)$.

- Este caminho deve passar por um nó u não-visitado (fora de C), tal que o caminho até u tem todos os nós com $D(.)$ correto (por hipótese).
- Como u está no caminho mais curto até v, $D(u) < D(v)$.
- Contradição: teríamos escolhido u em vez de v para colocar em C.

Árvores geradoras

- Dado um grafo não-direcionado e conectado, podemos escolher um conjunto de arestas que liga todos os nós, e que forma uma árvore.
- Qualquer árvore gerada desta forma é uma *árvore geradora* do grafo.
- Busca em profundidade produz uma árvore geradora (basta manter apenas as arestas que são percorridas durante a busca).
- Busca em largura também produz uma árvore geradora.
- O algoritmo de Dijkstra também produz uma árvore geradora, se for rodado até passar por todos os nós.

Árvores geradoras mínimas

- Uma árvore geradora obtida a partir de um grafo ponderado é *mínima* quando a soma de seus pesos é mínima entre todas as árvores geradoras do grafo.
- Existem dois algoritmos populares que produzem árvores geradoras mínimas a partir de um grafo ponderado conectado: algoritmo de Prim, e algoritmo de Kruskal.
- O algoritmo de Prim é praticamente idêntico ao algoritmo de Dijkstra, com uma pequena mudança na atualização dos nós.

Algoritmo de Prim (grafo não-direcionado, conectado, pesos não-negativos, sem arestas paralelas)

Prim(grafo G)

Escolha um nó n qualquer;
$D(n) = 0$;
$D(v) = \infty$ para $v \neq n$;
crie lista C vazia;
enquanto houver algum nó fora de C, repita:
insira em C o nó v com menor valor de $D(\cdot)$;
para cada vizinho z de v e que não está em C:
se w((v,z)) < D(z),
então D(z) = w((v,z)).

Ao executar esse algoritmo, devemos marcar as arestas que conectam nós à medida que nós são inseridos em C.

Algoritmo de Kruskal

1. Crie um grafo G' com todos os nós e nenhuma aresta.
2. Coloque todas as arestas em uma lista L.
3. Repita enquanto houver um ou mais nós em L:
   - Retire de L a aresta e de menor peso.
   - Se for possível adicionar essa aresta e a G' sem criar nenhum ciclo, adicione a aresta e a G'.
   - Caso contrário, descarte a aresta e.