

mostra a Figura 7.16(b) na Seção 7.8. Uma floresta geradora de um grafo  $G = (V, A)$  é um subgrafo que contém todos os vértices de  $G$  e forma uma floresta.

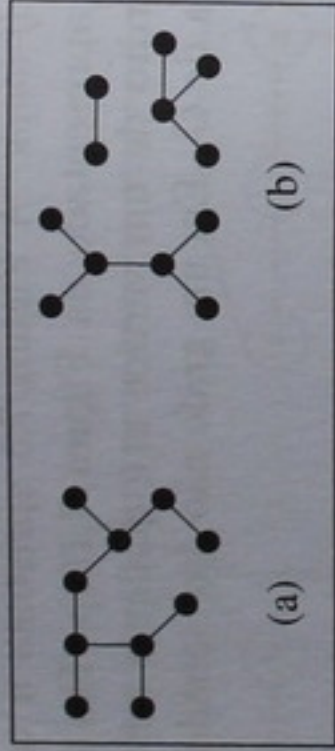


Figura 7.6 (a) Uma árvore livre; (b) Uma floresta.

## 7.2 O Tipo Abstrato de Dados Grafo

É importante considerar os algoritmos em grafos como tipos abstratos de dados, com um conjunto de operações associado a uma estrutura de dados, de tal forma que haja uma independência de implementação para as operações. Algumas das operações mais comuns incluem:

1. **FGVazio(Grafo):** Cria um grafo vazio. O procedimento retorna em Grafo um grafo contendo  $|V|$  vértices e nenhuma aresta.
2. **InserAresta( $V_1, V_2, \text{Peso}, \text{Grafo}$ ):** Insere uma aresta no grafo. O procedimento recebe a aresta  $(V_1, V_2)$  e seu Peso para serem inseridos em Grafo.
3. **ExisteAresta( $V_1, V_2, \text{Grafo}$ ):** Verifica se existe uma determinada aresta. A função retorna *true* se a aresta  $(V_1, V_2)$  está presente em Grafo, senão retorna *false*.
4. **Obtém a lista de vértices adjacentes a um determinado vértice.** Esta operação aparece na maioria dos algoritmos em grafos e, pela sua importância, será tratada separadamente logo a seguir.
5. **RetiraAresta( $V_1, V_2, \text{Peso}, \text{Grafo}$ ):** Retira uma aresta do grafo. O procedimento retira a aresta  $(V_1, V_2)$  de Grafo, retornando o peso da aresta na variável *Peso*.
6. **LiberaGrafo(Grafo):** Libera o espaço ocupado por um grafo. O procedimento libera toda a memória alocada para o grafo quando houve alocação dinâmica de memória, como no caso do uso de listas encadeadas.
7. **ImprimeGrafo(Grafo):** Imprime um grafo.
8. **GrafoTransposto(Grafo, GrafoT):** Obtém o transposto de um grafo direcionado. O procedimento é apresentado na Seção 7.7.
9. **RetiraMin(A):** Obtém a aresta de menor peso de um grafo. A função retira a aresta de menor peso dentre as arestas armazenadas no vetor *A*.

Uma operação que aparece com frequência é a de obter a lista de vértices adjacentes a um determinado vértice. Para implementar este operador de forma independente da representação escolhida para a aplicação em pauta, precisamos de três operações sobre grafos, a saber:

1. **ListaAdjVazia( $v, \text{Grafo}$ )** é uma função que retorna *true* se a lista de adjacentes de  $v$  está vazia, senão retorna *false*.
2. **PrimeiroListaAdj( $v, \text{Grafo}$ )** é uma função que retorna o endereço do primeiro vértice na lista de adjacentes de  $v$ .
3. **ProxAdj( $v, \text{Grafo}, u, \text{Peso}, \text{Aux}, \text{FimListaAdj}$ )** é um procedimento que retorna o vértice  $u$  (apontado por *Aux*) da lista de adjacentes de  $v$ , bem como o peso relacionado à aresta  $(v, u)$ . Ao retornar, *Aux* aponta para o próximo vértice da lista de adjacentes de  $v$ , e a variável booleana *FimListaAdj* retorna *true* se o final da lista de adjacentes for encontrado, senão retorna *false*.

Assim, em algoritmos sobre grafos é comum encontrar um pseudocomando do tipo:

**for**  $u \in \text{ListaAdjacentes}(v)$  **do** { faz algo com  $u$  }

O Programa 7.1 apresenta um possível refinamento do pseudocomando.

Programa 7.1 Trecho de programa para obter lista de adjacentes de um vértice de um grafo

```
if not ListaAdjVazia (v, Grafo)
then begin
    Aux := PrimeiroListaAdj (v, Grafo);
    FimListaAdj := false;
    while not FimListaAdj do
        ProxAdj (v, Grafo, u, Peso, Aux, FimListaAdj);
end;
```

Existem duas representações usuais para grafos: as matrizes de adjacência e as listas de adjacência. A Seção 7.2.1 apresenta a implementação de matrizes de adjacência usando arranjos. A Seção 7.2.2 apresenta a implementação de listas de adjacência usando apontadores, e a Seção 7.2.3 apresenta a implementação de listas de adjacência usando arranjos. Qualquer uma dessas representações pode ser usada tanto para grafos direcionados quanto para grafos não direcionados.

### 7.2.1 Implementação por meio de Matrizes de Adjacência

A matriz de adjacência de um grafo  $G = (V, A)$  contendo  $n$  vértices é uma matriz  $n \times n$  de *bits*, em que  $A[i, j]$  é 1 (ou verdadeiro, no caso de booleanos) se e somente se existir um arco do vértice  $i$  para o vértice  $j$ . Para grafos ponderados,