

O Programa 7.15 utiliza a função `MaxTT` para obter o vértice de maior $t[u]$ dentre os vértices restantes u ainda não visitados por `VisitaDfs`. A função `MaxTT` e o `TipoTempoTermino` estão no Programa 7.16.

Programa 7.16 Função para obter o vértice de maior tempo de término dentre os vértices restantes ainda não visitados por `VisitaDfs`

```

type TipoTempoTermino = record
    t: array[TipoValorVertice] of TipoValorTempo;
    Restantes: array[TipoValorVertice] of boolean;
    NumRestantes: TipoValorVertice;
end;

function MaxTT (var TT: TipoTempoTermino): TipoValorVertice;
var i, Temp: integer;
begin
    i:=0;
    while not TT.Restantes[i] do i:=i+1;
    Temp:=TT.t[i];
    MaxTT:=i;
    for i:=0 to Grafo.NumVertices-1 do
        if TT.Restantes[i]
        then if Temp < TT.t[i]
            then begin
                Temp:=TT.t[i];
                MaxTT:=i;
            end;
    end; { MaxTT }

```

Análise O algoritmo para obter os componentes fortemente conectados de um grafo direcionado $G = (V, A)$ utiliza o algoritmo `BuscaEmProfundidade` para realizar duas buscas em profundidade, uma em G e outra em G^T . Logo, a complexidade total de `ComponentesFortementeConectados` é $O(|V| + |A|)$.

7.8 Árvore Geradora Mínima

Esta seção trata o problema de obter a árvore geradora mínima de um grafo não direcionado $G = (V, A)$. Uma aplicação típica para árvores geradoras mínimas ocorre no projeto de redes de comunicações conectando diversas localidades. Para conectar um conjunto de n localidades podemos usar um arranjo de $n-1$ conexões, cada uma conectando duas localidades. Assumindo que as conexões sejam realizadas por meio de cabos de transmissão, de todas as possibilidades de conexões, aquela que usa a menor quantidade de cabos é geralmente a mais desejável.

Esse problema pode ser modelado utilizando um grafo conectado, não direcionado $G = (V, A)$, em que V é o conjunto de cidades, A é o conjunto de possíveis conexões entre pares de localidades e, para cada aresta $(u, v) \in A$, existe um peso $p(u, v)$ especificando o custo (total de cabo necessário) para conectar u a v . Agora, o problema é encontrar um subconjunto $T \subseteq A$ que conecte todos os vértices de G e cujo peso total

$$p(T) = \sum_{(u,v) \in T} p(u, v)$$

seja minimizado. Uma vez que $G' = (V, T)$ é acíclico e conecta todos os vértices, T forma uma árvore chamada **árvore geradora** de G uma vez que T “gera” o grafo G . O problema de obter a árvore T é conhecido como **árvore geradora mínima**. A Figura 7.16(a) mostra o exemplo de um grafo não direcionado G com os pesos mostrados ao lado de cada aresta. A Figura 7.16(b) mostra a árvore geradora mínima T cujo peso total é 12. T não é única, pois a substituição da aresta $(3, 5)$ pela aresta $(2, 5)$ produz outra árvore geradora de custo 12.

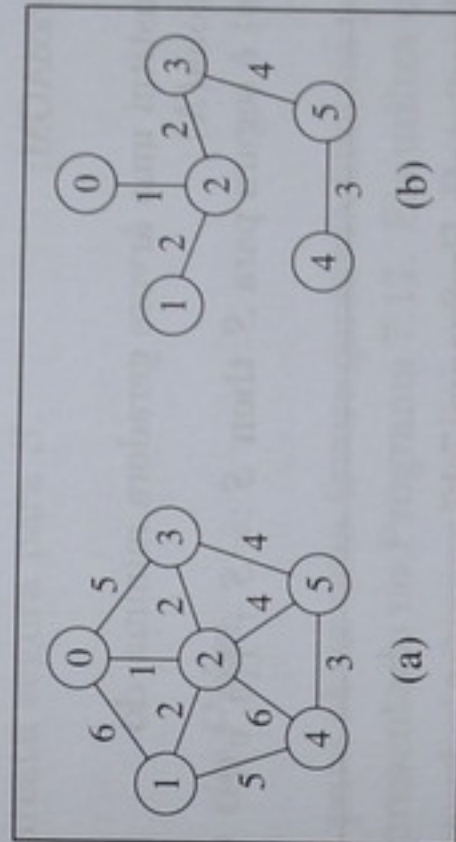


Figura 7.16 (a) Grafo não direcionado G ; (b) Árvore geradora mínima T de peso total 12.

Nesta seção, vamos estudar dois algoritmos para obter a árvore geradora mínima de um grafo: algoritmo de Prim (1957) e algoritmo de Kruskal (1956). Os dois algoritmos são **algoritmos gulosos**. Conforme mostrado na Seção 2.7, a cada passo do algoritmo guloso, uma escolha tem de ser feita dentre várias possíveis. A estratégia gulosa sempre faz a escolha melhor em cada momento. Por isso, tal estratégia nem sempre garante encontrar a solução ótima global para os problemas. Entretanto, para o problema da árvore geradora mínima, existem estratégias gulosas que obtêm a árvore geradora de peso total mínimo.

A Seção 7.8.1 introduz um algoritmo genérico para obter a árvore geradora mínima de um grafo por meio da adição de uma aresta de cada vez. Essa forma de obter a árvore geradora mínima de um grafo é mostrada em maiores detalhes em Cormen, Leiserson, Rivest e Stein (2001). As Seções 7.8.2 e 7.8.3 apresentam duas maneiras de implementar o algoritmo genérico: algoritmo de Prim e algoritmo de Kruskal, respectivamente.