ao final do processamento, Antecessor conterá, de fato, uma árvore de caminhos mais curtos. Essa árvore de caminhos mais curtos é como a árvore de busca em largura da Seção 7.5, só que ela conterá caminhos mais curtos que são definidos em termos dos pesos de cada aresta de G em vez do número de arestas. Assim como as árvores de busca em largura, caminhos mais curtos não são necessariamente únicos, podendo haver mais de um caminho de peso mínimo.

A **árvore de caminhos mais curtos** com raiz em $u \in V$ é um subgrafo direcionado $G' = (V', A')$, em que $V' \subseteq V$ e $A' \subseteq A$, tal que:

1. V' é o conjunto de vértices alcançáveis a partir de $s \in G$;
2. G' forma uma árvore de raiz s;
3. para todos os vértices $v \in V'$, o caminho simples de s até v é um caminho mais curto de s até v em G.

O algoritmo que vamos apresentar nesta seção é conhecido como algoritmo de Dijkstra (1959). O algoritmo mantém um conjunto S de vértices cujos caminhos mais curtos até um vértice origem já são conhecidos. Ao final de sua execução, o algoritmo produz uma árvore de caminhos mais curtos de um vértice origem s para todos os vértices alcançáveis a partir de s.

Relaxamento

O algoritmo de Dijkstra utiliza a técnica de relaxamento. Para cada vértice $v \in V$, o atributo $p[v]$ é um limite superior do peso de um caminho mais curto do vértice origem s até v. O vetor $p[v]$ contém uma estimativa de um caminho mais curto. O primeiro passo do algoritmo é inicializar os antecessores e as estimativas de caminhos mais curtos. Após o passo de inicialização, $Antecessor[v] = nil$ para todo vértice $v \in V$, $p[u] = 0$ para o vértice origem s, e $p[v] = \infty$ para $v \in V - \{s\}$.

O processo de **relaxamento** de uma aresta (u, v) consiste em verificar se é possível melhorar o melhor caminho obtido até o momento até v se passarmos por u. Se isso acontecer, então $p[v]$ e $Antecessor[v]$ devem ser atualizados. Em outras palavras, o passo de relaxamento pode decrementar o valor da estimativa de caminho mais curto $p[v]$ e atualizar o antecessor de v em $Antecessor[v]$. O pseudocódigo do Programa 7.20 mostra como a operação de relaxamento deve ser implementada.

Programa 7.20 *Relaxamento de uma aresta*

```
if p[v] > p[u] + peso da aresta (u,v)
then p[v] = p[u] + peso da aresta (u,v)
     Antecessor[v] := u
```


O primeiro refinamento do algoritmo de Dijkstra pode ser visto no Programa 7.21. As linhas 1-3 realizam a inicialização dos antecessores e das estimativas de caminhos mais curtos. A linha 4 inicializa a distância do vértice raiz a ele mesmo como sendo zero. A linha 5 constrói o *heap* sobre todos os vértices do grafo, e a linha 6 inicializa o conjunto solução S como vazio. A linha 7 é um anel que executa enquanto o *heap* for diferente de vazio. O algoritmo mantém o invariante seguinte: o número de elementos do *heap* é igual a $V - S$ no início do anel **while** na linha 7. Desde que $S = \emptyset$ no início do anel, então o invariante é verdadeiro. A cada iteração do anel nas linhas 8-13 um vértice u é extraído do *heap* e adicionado ao conjunto S, mantendo assim o invariante. Na linha 8 a operação RetiraMin obtém o vértice u que contém o caminho mais curto estimado até aquele momento e o adiciona ao conjunto solução S. A seguir, no anel da linha 10, a operação de relaxamento é realizada sobre cada aresta (u, v) adjacente ao vértice u, atualizando o caminho estimado $p[v]$ e o $Antecessor[v]$ se o caminho mais curto para v puder ser melhorado usando o caminho por meio de u.

Programa 7.21 *Primeiro refinamento do algoritmo de Dijkstra*

```
   procedure Dijkstra (Grafo, Raiz);
1    for v := 0 to Grafo.NumVertices-1 do
2      p[v] := INFINITO;
3      Antecessor[v] := -1;
4    p[Raiz] := 0;
5    Constroi heap no vetor A;
6    S := ∅;
7    While heap > 1 do
8      u := RetiraMin(A);
9      S := S + u
10     for v ∈ ListaAdjacentes[u] do
11       if p[v] > p[u] + peso da aresta (u,v)
12       then p[v] = p[u] + peso da aresta (u,v)
13            Antecessor[v] := u
```

A Figura 7.20 mostra o funcionamento do algorimo de Dijkstra. Como ilustrado na figura, a árvore começa pelo vértice 0. A cada passo, um vértice é adicionado à árvore S de caminhos mais curtos. Arestas em negrito pertencem à árvore de caminhos mais curtos sendo construída. Essa estratégia é **gulosa**, uma vez que a árvore é aumentada a cada passo com uma aresta que contribui com o mínimo possível para o custo (peso) total de cada caminho.

A Tabela 7.1 mostra os valores de S e $p[v]$, $v = 0, 1, 2, 3, 4$, a cada iteração do algoritmo de Dijkstra.