

Figura 5.18 O passo de ranking constrói a estrutura de dados usada para computar a função $rank:[0,5] \rightarrow [0,2]$ em tempo O(1).

A função rank pode ser implementada utilizando um algoritmo proposto por Pagh (2001). O algoritmo usa ϵM bits adicionais, onde $0 < \epsilon < 1$ para armazenar explicitamente o rank de cada k-ésimo índice de g em uma tabela TabRank, onde $k = \lfloor \log(M)/\epsilon \rfloor$. Para garantir uma avaliação da função rank(u) em tempo O(1), é necessário usar uma tabela T_r auxiliar. A Figura 5.19 mostra as tabelas TabRank e T_r para o exemplo da Figura 5.18.

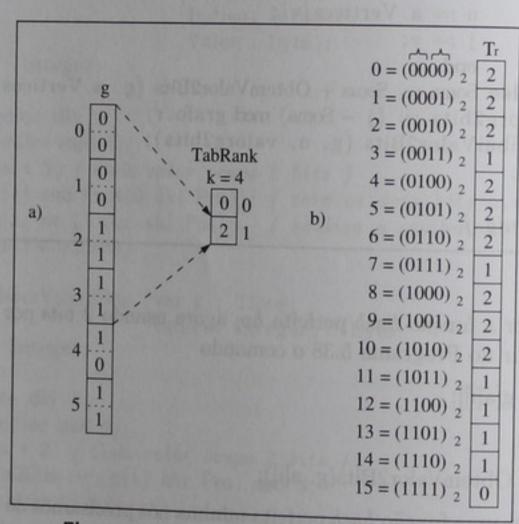


Figura 5.19 O uso das tabelas TabRank e T_r .

A tabela TabRank mostrada na Figura 5.19(a) armazena em cada entrada o número total de valores de 2 bits diferentes de r=3 até cada k-ésima posição do arranjo g (exclusive). No exemplo consideramos k=4. Assim, existem 0 valores até a posição 0 e 2 valores até a posição 4 de g. O Programa 5.40 mostra o procedimento para gerar a tabela TabRank.

Programa 5.40 Gera a tabela TabRank

Para calcular o rank(u) usando as tabelas TabRank e T_r são necessários dois passos: (i) obter o rank do maior índice precomputado $v \leq u$ em TabRank; e (ii) usar T_r para contar o número de vértices atribuídos da posição v até u-1. No exemplo da Figura 5.19(b) a tabela T_r possui 16 entradas necessárias para armazenas todas as combinações possíveis de 4 bits. Por exemplo, a posição 0, cujo valor binário é $(0000)_2$, contém dois valores diferentes de r=3; na posição 3, cujo valor binário é $(0011)_2$, contém apenas um valor diferente de r=3, e assim por diante. Cabe notar que cada valor de $r\geq 2$ requer uma tabela T_r diferente que deve ser gerada a priori pelo Programa 5.41. O procedimento para gerar a tabela T_r considera que T_r é indexada por um número de 8 bits e, portanto, MaxTrValue = 255. Além disso, no máximo 4 vértices podem ser empacotados em um byte, razão pela qual o anel interno vai de 1 a 4.

Programa 5.41 Gera a tabela T_r

```
Procedure GeraTr (var Tr: TipoTr);
var i, j, v, Soma: Integer;
begin
   Soma := 0;
   for i := 0 to MAXTRVALUE do
        begin
        Soma := 0; v := i;
        for j := 1 to 4 do
        begin
        if ((v and 3) <> NAOATRIBUIDO) then Soma := Soma + 1;
        v := v shr 2;
        end;
        Tr[i] := Soma;
        end;
end; { GeraTr }
```

A função hash perfeita mínima resultante tem a seguinte forma:

$$hpm(x) = rank(hp(x)) \tag{5.2}$$