

Programa G.16 Função para obter o vértice de maior tempo de término dentre os vértices restantes ainda não visitados por *VisitaDFS*

```
typedef struct TipoTempoTermino {
    TipoValorTempo t[MAXNUMVERTICES + 1];
    short Restantes[MAXNUMVERTICES + 1];
    TipoValorVertice NumRestantes;
} TipoTempoTermino;

TipoValorVertice MaxTT(TipoTempoTermino *TT, TipoGrafo *Grafo)
{
    TipoValorVertice Result; short i = 0, Temp;
    while (!TT->Restantes[i]) i++;
    Temp = TT->t[i]; Result = i;
    for (i = 0; i <= Grafo->NumVertices - 1; i++)
    {
        if (TT->Restantes[i])
        {
            if (Temp < TT->t[i]) { Temp = TT->t[i]; Result = i; }
        }
    }
    return Result;
}
```

Programa G.17 Algoritmo genérico para obter a árvore geradora mínima

```
void GenericoAGM()
{
    1 S = ∅;
    2 while(S não constitui uma árvore geradora mínima)
    3     { (u,v) = seleciona(A);
    4       if(aresta (u,v) é segura para S) S = S + {(u,v)} }
    5 return S;
}
```

Programa G.18 Operadores para manter o heap indireto

```
/*—Entra aqui o operador Constroi da Seção 4.1.5 (Programa D.10) —*/
/*—Trocando a chamada Refaz (Esq, n, A) por RefazInd (Esq, n, A) —*/
void RefazInd(TipoIndice Esq, TipoIndice Dir, TipoItem *A,
              TipoPeso *P, TipoValorVertice *Pos)
{
    TipoIndice i = Esq; int j = i * 2; TipoItem x; x = A[i];
    while (j <= Dir)
    {
        if (j < Dir)
        {
            if (P[A[j].Chave] > P[A[j+1].Chave]) j++;
            if (P[x.Chave] <= P[A[j].Chave]) goto L999;
            A[i] = A[j]; Pos[A[j].Chave] = i; i = j;
            j = i * 2;
        }
        L999: A[i] = x;
        Pos[x.Chave] = i;
    }
}
```

Continuação do Programa G.18

```
void Constroi(TipoItem *A, TipoPeso *P, TipoValorVertice *Pos)
{
    TipoIndice Esq;
    Esq = n / 2 + 1;
    while (Esq > 1) { Esq--; RefazInd(Esq, n, A, P, Pos); }
}

TipoItem RetiraMinInd(TipoItem *A, TipoPeso *P, TipoValorVertice *Pos)
{
    TipoItem Result;
    if (n < 1) { printf("Erro: heap vazio\n"); return Result; }
    Result = A[1]; A[1] = A[n];
    Pos[A[n].Chave] = 1; n--;
    RefazInd(1, n, A, P, Pos);
    return Result;
}

void DiminuiChaveInd(TipoIndice i, TipoPeso ChaveNova, TipoItem *A,
                    TipoPeso *P, TipoValorVertice *Pos)
{
    TipoItem x;
    if (ChaveNova > P[A[i].Chave])
    {
        printf("Erro: ChaveNova maior que a chave atual\n");
        return;
    }
    P[A[i].Chave] = ChaveNova;
    while (i > 1 && P[A[i / 2].Chave] > P[A[i].Chave])
    {
        x = A[i / 2]; A[i / 2] = A[i];
        Pos[A[i].Chave] = i / 2; A[i] = x;
        Pos[x.Chave] = i; i /= 2;
    }
}
```

Programa G.19 Algoritmo de Prim para obter a árvore geradora mínima

```
/*—Entra aqui os operadores do tipo grafo do Programa G.3 —*/
/*—ou do Programa G.5 ou do Programa G.7, e os operadores —*/
/*—RefazInd, RetiraMinInd e DiminuiChaveInd do Programa G.17 —*/

void AgmPrim(TipoGrafo *Grafo, TipoValorVertice *Raiz)
{
    int Antecessor[MAXNUMVERTICES + 1];
    short Itensheap[MAXNUMVERTICES + 1];
    Vetor A;
    TipoPeso P[MAXNUMVERTICES + 1];
    TipoValorVertice Pos[MAXNUMVERTICES + 1];
    TipoValorVertice u, v;
    TipoItem TEMP;
    for (u = 0; u <= Grafo->NumVertices; u++)
    {
        /*Constroi o heap com todos os valores igual a INFINITO*/
        Antecessor[u] = -1; P[u] = INFINITO;
    }
}
```