

Continuação do Programa G.19

```

A[u+1].Chave = u; /*Heap a ser construido*/
Itensheap[u] = TRUE; Pos[u] = u + 1;
}
n = Grafo->NumVertices; P[*Raiz] = 0;
Constroi(A, P, Pos);
while (n >= 1) /*enquanto heap nao vazio*/
{ TEMP = RetiraMinInd(A, P, Pos);
  u = TEMP.Chave; Itensheap[u] = FALSE;
  if (u != *Raiz)
    printf("Aresta de arvore: v[%d] v[%d]" ,u,Antecessor[u]); getchar();
  if (!ListaAdjVazia(&u, Grafo))
  { Aux = PrimeiroListaAdj(&u, Grafo);
    FimListaAdj = FALSE;
    while (!FimListaAdj)
    { ProxAdj(&u, Grafo, &v, &Peso, &Aux, &FimListaAdj);
      if (Itensheap[v] && Peso < P[v])
      { Antecessor[v] = u;
        DiminuiChaveInd(Pos[v], Peso, A, P, Pos);
      }
    }
  }
}
}
}
}

```

Programa G.20 Relaxamento de uma aresta

```

if (p[v] > p[u] + peso da aresta (u,v))
{ p[v] = p[u] + peso da aresta (u,v); Antecessor[v] = u; }

```

Programa G.21 Primeiro refinamento do algoritmo de Dijkstra

```

void Dijkstra(Grafo, Raiz)
{
  1. for (v=0; v < Grafo.NumVertices; v++)
  2.   p[v] = Infinito;
  3.   Antecessor[v] = -1;
  4. p[Raiz] = 0;
  5. Constroi heap no vetor A;
  6. S = Ø;
  7. while (heap > 1)
  8.   u = RetiraMin(A);
  9.   S = S + u;
  10. for (v ∈ ListaAdjacentes[u])
  11.   if (p[v] > p[u] + peso da aresta(u,v))
  12.     p[v] = p[u] + peso da aresta(u,v);
  13.     Antecessor[v] = u;
}

```

Programa G.22 Implementação do algoritmo de Dijkstra

```

/*--- Entram 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 Dijkstra(TipoGrafo *Grafo, TipoValorVertice *Raiz)
{ TipoPeso P[MAXNUMVERTICES + 1];
  TipoValorVertice Pos[MAXNUMVERTICES + 1];
  long Antecessor[MAXNUMVERTICES + 1];
  short Itensheap[MAXNUMVERTICES + 1];
  TipoVetor A;
  TipoValorVertice u, v;
  Tipotem temp;
  for (u = 0; u <= Grafo->NumVertices; u++)
  { /*Constroi o heap com todos os valores igual a INFINITO*/
    Antecessor[u] = -1; P[u] = INFINITO;
    A[u+1].Chave = u; /*Heap a ser construido*/
    Itensheap[u] = TRUE; Pos[u] = u + 1;
  }
  n = Grafo->NumVertices;
  P[*Raiz] = 0;
  Constroi(A, P, Pos);
  while (n >= 1)
  { /*enquanto heap nao vazio*/
    temp = RetiraMinInd(A, P, Pos);
    u = temp.Chave; Itensheap[u] = FALSE;
    if (!ListaAdjVazia(&u, Grafo))
    { Aux = PrimeiroListaAdj(&u, Grafo); FimListaAdj = FALSE;
      while (!FimListaAdj)
      { ProxAdj(&u, Grafo, &v, &Peso, &Aux, &FimListaAdj);
        if (P[v] > (P[u] + Peso))
        { P[v] = P[u] + Peso; Antecessor[v] = u;
          DiminuiChaveInd(Pos[v], P[v], A, P, Pos);
          printf("Caminho: v[%d] v[%ld] d[%d]" ,
                v, Antecessor[v], P[v]);
          scanf("%s", n);
          getchar();
        }
      }
    }
  }
}

```

Programa G.23 Estrutura do tipo hipergrafo implementado como matriz de incidência

```

#define MAXNUMVERTICES 100
#define MAXNUMARESTAS 4500
#define MAXR 5
typedef int TipoValorVertice;

```