DAGs, Ordenação topológica

1. DAGs (*directed acyclic graphs*): grafos dirigidos acíclicos (sem circuitos *dirigidos*)

2. “Ordenação topológica”

Ordenação topológica; algoritmo natural

- Todo DAG tem necessariamente (pelo menos) uma fonte e (pelo menos) um sorvedouro.

- Algoritmo: coloque todos as fontes no início da ordenação topológica, remova-os do DAG, e repita.

Algoritmo natural; Programa 19.8

```
#include "QUEUE.h"
static int in[maxV];
void DAGts(Dag D, int ts[])
  { int i, v; link t;
    for (v = 0; v < D->V; v++)
      { in[v] = 0; ts[v] = -1; }
    for (v = 0; v < D->V; v++)
      for (t = D->adj[v]; t != NULL; t = t->next)
        in[t->v]++;
```

Algoritmo natural; Programa 19.8 (cont.)

```
  QUEUEinit(D->V);
  for (v = 0; v < D->V; v++)
    if (in[v] == 0) QUEUEput(v);
  for (i = 0; !QUEUEempty(); i++)
    {
      ts[i] = (v = QUEUEget());
      for (t = D->adj[v]; t != NULL; t = t->next)
        if (--in[t->v] == 0) QUEUEput(t->v);
    }
}
```

Ordenação topológica baseada em BeP

1. De fato, BeP dá naturalmente uma ordenação topológica reversa.

2. Usamos o vetor `post[]`, que nos diz em que ordem as chamadas recursivas da BeP terminaram.

3. Isto dá um ordenação topológica reversa (Propriedade 19.11).

▷ Busca em profundidade em grafos dirigidos; Programa 19.2 (aula de 10/4/2007)

BeP para Grafos Dirigidos, Programa 19.2

```
void dfsR(gGraph G, Edge e)
  { link t; int v, w = e.w; Edge x;
    show("tree", e);  st[e.w]=e.v;
    pre[w] = cnt++;
    for (t = G->adj[w]; t != NULL; t = t->next)
      if (pre[t->v] == -1) dfsR(G, EDGE(w, t->v));
      else
        { v = t->v; x = EDGE(w, v);
          if (post[v] == -1) show("back", x);
          else if (pre[v] > pre[w]) show("down", x);
          else show("cross", x);
        }
    post[w] = cntP++;
  }
```

BeP para Grafos Dirigidos, Exemplo

```
13 vertices, 17 edges
 0:  5  6  2  3  1
 1:
 2:  3
 3:  5  4
 4:  9
 5:
 6:  9  4
 7:  6
 8:  7
 9: 11 10 12
10:
11: 12
12:
```

BeP para Grafos Dirigidos, Exemplo

```
0-0 tree
  0-5 tree
  0-6 tree
    6-9 tree
      9-11 tree
        11-12 tree
      9-10 tree
      9-12 down
    6-4 tree
      4-9 cross
  0-2 tree
    2-3 tree
      3-5 cross
      3-4 cross
  0-3 down
  0-1 tree
7-7 tree
  7-6 cross
8-8 tree
  8-7 cross
```

BeP para Grafos Dirigidos, Exemplo

v:	0	1	2	3	4	5	6	7	8	9	10	11	12
pre[]:	0	10	8	9	7	1	2	11	12	3	6	4	5
post[]:	10	9	8	7	5	0	6	11	12	4	3	2	1
st[]:	0	0	0	2	6	0	0	7	8	6	9	9	11

Classificação dos arcos

```
e = (v, w)

pre post exemplo tipo
 <    >     9-12  down
 >    >     4-9   cross
 >    <     ???   back
```

Correção do uso de post[]

Propriedade 1 (Propriedade 19.11)**.** *O vetor* `post[]` *determina uma ordenação linear dos vértices que é uma ordenação topológica reversa.*

Prova. Suponha que `s` e `t` são tais que `post[s] < post[t]`. Suponha por contradição que existe o arco `(s,t)` em `G`. Note que então este arco seria um arco ascendente, o que é uma contradição, pois a existência de um arco ascendente implica na existência de um circuito dirigido em `G`. □

Programa 19.6

```
static int cnt0; static int pre[maxV];
void DAGts(Dag D, int ts[])
  { int v; cnt0 = 0;
    for (v = 0; v < D->V; v++)
      { ts[v] = -1; pre[v] = -1; }
    for (v = 0; v < D->V; v++)
      if (pre[v] == -1) TSdfsR(D, v, ts);
}
void TSdfsR(Dag D, int v, int ts[])
  { link t; pre[v] = 0;
    for (t = D->adj[v]; t != NULL; t = t->next)
      if (pre[t->v] == -1) TSdfsR(D, t->v, ts);
    ts[cnt0++] = v;
  }
```

Como obter uma ordenação topológia com post[]?

1. Invertemos os arcos de G (custa espaço)

2. Usamos uma pilha

3. Rotulamos em ordem reversa!

Programa 19.7

```
void TSdfsR(Dag D, int v, int ts[])
  { int w;
    pre[v] = 0;
    for (w = 0; w < D->V; w++)
      if (D->adj[w][v] != 0)
        if (pre[w] == -1) TSdfsR(D, w, ts);
    ts[cnt0++] = v;
  }
```

Componentes fortemente conexos

1. Subgrafos induzidos pela relação de equivalência $x \leftrightarrow y$ (acessibilidade mútua)
2. Algoritmos para identificação de componented fortemente conexos:
   - ▷ Calculamos o fecho transitivo de G e calculamos $\leftrightarrow$ [custo do fecho transitivo $\times$ número de arcos no fecho]
   - ▷ Algoritmo de Kosaraju (década de 80) [$O(n+m)$]
   - ▷ Algoritmo de Tarjan (1972) [$O(n+m)$]
   - ▷ Algoritmo de Gabow (1999) [$O(n+m)$]

Algoritmo de Kosaraju

1. Invertemos G

2. Executamos uma BeP e determinamos `post[]`

3. Executamos uma BeP na ordem dada por `post[]` *invertida*

Programa 19.10, Algoritmo de Kosaraju

```
static int post[maxV], postR[maxV];
static int cnt0, cnt1;
void SCdfsR(Graph G, int w)
  { link t;
    G->sc[w] = cnt1;
    for (t = G->adj[w]; t != NULL; t = t->next)
      if (G->sc[t->v] == -1) SCdfsR(G, t->v);
    post[cnt0++] = w;
  }
```

Programa 19.10, Algoritmo de Kosaraju (cont.)

```
int GRAPHsc(Graph G)
  { int v; Graph R = GRAPHreverse(G);
    cnt0 = 0; cnt1 = 0;
    for (v = 0; v < G->V; v++) R->sc[v] = -1;
    for (v = 0; v < G->V; v++) if (R->sc[v] == -1) SCdfsR(R, v);
    cnt0 = 0; cnt1 = 0;
    for (v = 0; v < G->V; v++) G->sc[v] = -1;
    for (v = 0; v < G->V; v++) postR[v] = post[v];
    for (v = G->V-1; v >= 0; v--)
      if (G->sc[postR[v]] == -1)
        { SCdfsR(G, postR[v]); cnt1++; }
    GRAPHdestroy(R);
    return cnt1;
  }
```

Programa 19.10, Algoritmo de Kosaraju (cont.)

```
int GRAPHstrongreach(Graph G, int s, int t)
  { return G->sc[s] == G->sc[t]; }
```

Exemplo; grafo da Figura 19.28

```
13 vertices, 22 arcs
 0:  6  1  5  0
 1:  1
 2:  3  0  2
 3:  2  5  3
 4:  2 11  3  4
 5:  4  5
 6:  4  9  6
 7:  8  6  7
 8:  9  7  8
 9: 10 11  9
10: 12 10
11: 12 11
12:  9 12
```

Exemplo, Figura 19.28; Reverso de G

```
0-0 tree
  0-2 tree
    2-4 tree
      4-6 tree
        6-7 tree
          7-8 tree
            8-7 back
        6-0 cross
      4-5 tree
        5-3 tree
          3-4 back
          3-2 cross
        5-0 cross
    2-3 down
```

```
1-1 tree
  1-0 cross
9-9 tree
  9-12 tree
    12-11 tree
      11-9 back
      11-4 cross
    12-10 tree
      10-9 cross
  9-8 cross
  9-6 cross
```

Exemplo, Figura 19.28

v:	0	1	2	3	4	5	6	7	8	9	10	11	12
pre[]:	0	8	1	7	2	6	3	4	5	9	12	11	10
post[]:	8	7	6	3	5	4	2	0	1	11	10	12	9
st[]:	0	1	0	5	2	4	4	6	7	9	12	12	9

Exemplo, Figura 19.28

```
9-9 tree
  9-10 tree
    10-12 tree
      12-9 cross
  9-11 tree
    11-12 cross
1-1 tree
0-0 tree
  0-6 tree
    6-4 tree
      4-2 tree
        2-3 tree
          3-2 cross
          3-5 tree
                5-4 cross
            2-0 cross
          4-11 cross
          4-3 down
        6-9 cross
      0-1 cross
      0-5 down
  7-7 tree
    7-8 tree
      8-9 cross
      8-7 cross
    7-6 cross
```

Algoritmo de Kosaraju

Propriedade 2 (Propriedade 19.14)**.** *O método de Kosaraju encontra os componentes fortemente conexos de G em tempo $O(n + m)$.*

Prova. (. . .) □