DAGs, Ordenação topológica

DAGs, Ordenação topológica

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

DAGs, Ordenação topológica

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

2. “Ordenação topológica”

Ordenação topológica; algoritmo natural

Ordenação topológica; algoritmo natural

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

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

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

Ordenação topológica baseada em BeP

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

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.

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)

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)

4. Programa 19.2 (aula de 23/04)

BeP para Grafos Dirigidos, Programa 19.2

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

BeP para Grafos Dirigidos, Exemplo

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

BeP para Grafos Dirigidos, Exemplo

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

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

Classificação dos arcos

e = (v, w)

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

Correção do uso de post[]

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.*

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.*

Demonstração. Suponha que `s` e `t` são tais que `post[s] < post[t]`. Suponha por contradição que existe o arco `(s,t)` em `G`.

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.*

Demonstração. 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

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[]?

Como obter uma ordenação topológia com post[]?

1. Invertemos os arcos de G (custa espaço)

Como obter uma ordenação topológia com post[]?

1. Invertemos os arcos de G (custa espaço)

2. Usamos uma pilha

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

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;
  }
```