Lec 2. 拓扑排序

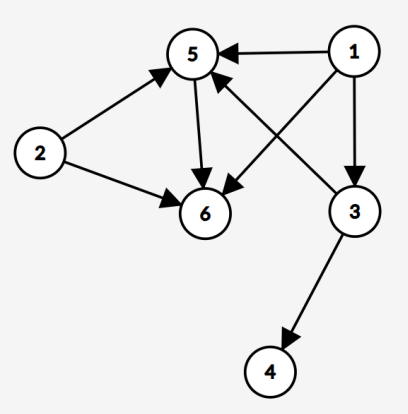
Alan233

2025 年 08 月 16 日

一　有向无环图 (Directed Acyclic Graph)

引入. 首先, 我们明确 **有向无环图** 的定义:

- **有向**: 表明这是一张**“有向图”**;
- **无环**: 表明图中不存在一个**“有向环”**.



如果这张图是有向无环图, 那么它就可以进行**拓扑排序** (这是我们这节课的主题).

1.1　判定是否为 DAG

如何判定一张图是否为有向无环图呢?

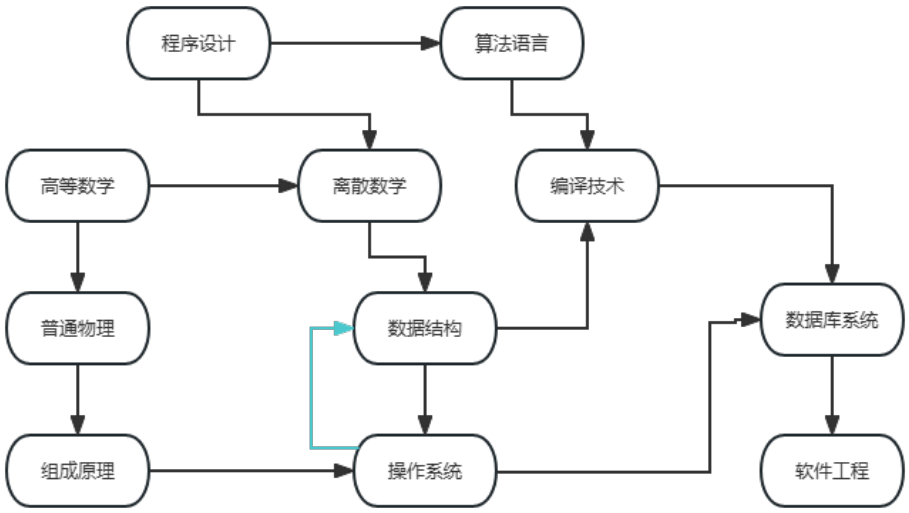
法一. 检验它是否可以进行拓扑排序即可.

法二. 对图进行一遍 dfs 得到一棵 dfs 生成树, 判断是否有连向祖先的非树边 (返祖边). 如果有的话, 那就有环了.

二　拓扑排序 (Topological Sort)

拓扑排序要解决的问题是「**给一个有向无环图的所有节点进行排序**」.

上过大学的同学都知道, 我们在大一大二会修读一些数理基础课程, 而后会逐渐转向专业课程的学习. 这些课程之间存在一定的依赖关系 (即预修要求). 如果学习课程 B 先得学习课程 A, 我们在图上就连一条 $A \to B$ 的边, 那么一种可能的依赖关系如下 (黑色边):



图源 OI-WIKI

可以发现, 这样的关系形成了一个 DAG, 是符合正常大学生的学习路线要求的.

但倘若这时候排课的老师一时糊涂, 多嘴说了一句「学 数据结构 之前得先修读 操作系统」, 此时图中就出现了一个有向环. 作为学生的你, 就会无比困惑: 那我这俩课该咋排课嘞?

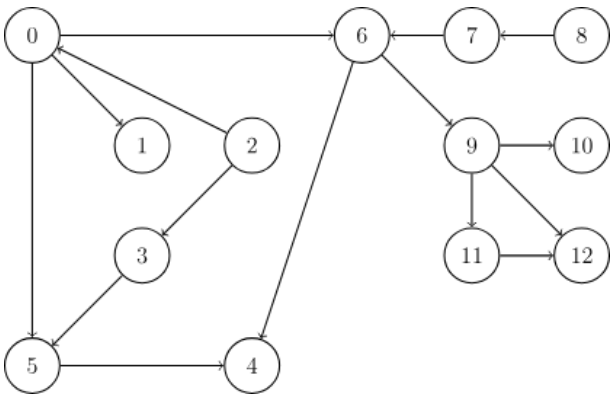
由此, 我们引入拓扑排序. 它的目的是将图中的点排成一列, 使得对于任何一条有向边 $u \to v$, 都满足 u 排在 v 的前列.

2.1 Kahn 算法

这是一个很经典且常用的做法, 在课程中我们只介绍该算法 (另一种 DFS 算法用不上).

初始状态下, 队列 `queue` 加入所有**入度为 0** 的点.

- 每次取出 `queue` 中的队首 u, 然后将 u 的所有边 $u \to v_1, u \to v_2, ..., u \to v_t$ 删除.
- 对于边 $u \to v_i$, 删除后会导致 v_i 的入度减 1. 如果 v_i 的入度减至 0, 则将 v_i 加入 `queue` 中.
- 不断重复以上过程, 直至 `queue` 为空.



对其排序的一种可能结果为 $2 \to 8 \to 0 \to 3 \to 7 \to 1 \to 5 \to 6 \to 9 \to 4 \to 11 \to 10 \to 12$.

时间复杂度为 $O(V + E)$. 实现如下:

```cpp
/* 传入邻接表以及节点数 n
   返回顶点序列 idx
*/
vector<int> topo_sort(vector<int> *adj, int n) {
  vector<int> deg_in(n, 0), idx;
  queue<int> q;

  for (int i = 0; i < n; i++) {
    for (auto j : adj[i])
      deg_in[j]++;
  }

  for (int i = 0; i < n; i++) {
    if (deg_in[i] == 0)
      q.push(i);
  }
  while (!q.empty()) {
    int u = q.front(); q.pop();
    idx.push_back(u);
    for (auto v : adj[u]) {
      if (--deg_in[v] == 0)
        q.push(v);
    }
  }
  return idx;
}
```

2.2 求字典序最大 / 最小的拓扑排序

以求字典序最大为例, 我们只需要将上述的 `queue` 改为大根堆的 `priority_queue` 即可, 时间复杂度 $O(E + V \log V)$.

```cpp
vector<int> topo_sort(vector<int> *adj, int n) {
  vector<int> deg_in(n, 0), idx;
  priority_queue<int> pq;

  for (int i = 0; i < n; i++) {
    for (auto j : adj[i])
      deg_in[j]++;
  }

  for (int i = 0; i < n; i++) {
    if (deg_in[i] == 0)
      pq.push(i);
  }
  while (!pq.empty()) {
    int u = pq.top(); pq.pop();
    idx.push_back(u);
```

```
    for (auto v : adj[u]) {
      if (--deg_in[v] == 0)
        pq.push(v);
    }
  }
  return idx;
}
```

2.3 杂物[1]

题目 1. John 要给他的 n 头奶牛挤奶，有无数名挤奶员. 对于第 i 头奶牛，它需要在第 $j_1, j_2, ..., j_t$ 头奶牛都被挤完奶后才可以被挤奶，挤奶的时间为 l_i. 求至少需要花费多少时间，才能保证所有奶牛都被挤完奶.

$3 \le n \le 10000$，题目保证 $j_1, j_2, ..., j_t < i$.

由于题目保证 $j_1, j_2, ..., j_t < i$，因此 $1, 2, ..., n$ 就是一个可行的拓扑序，我们可以直接边顺序读入边进行拓扑排序的 DP.

时间复杂度 $O(n)$.

```
#include <bits/stdc++.h>
using namespace std;

int main() {
  int n, ans = 0;
  cin >> n;
  vector<int> t(n);
  for (int i = 0; i < n; i++) {
    int id, len;
    cin >> id >> len;
    int j;
    while ((cin >> j) && j) {
      --j;
      t[i] = max(t[i], t[j]);
    }
    t[i] += len;
    ans = max(ans, t[i]);
  }
  cout << ans << endl;
  return 0;
}
```

2.4 最长路[2]

[1] https://www.luogu.com.cn/problem/P1113
[2] https://www.luogu.com.cn/problem/P1807

题目 2. 设 G 为有 n 个顶点的带权有向无环图, G 中各顶点的编号为 $1 \sim n$.

计算图 G 中从点 1 到点 n 的最长路.

$1 \le n \le 1500$, $0 \le m \le 5 \times 10^4$, $-10^5 \le w \le 10^5$.

跟上一题同样的道理, 只不过这次需要先跑拓扑排序得到一个顶点序列, 在此序列上顺序进行 DP 即可.

时间复杂度 $O(n+m)$.

2.5 排序[3]

题目 3. 一个不同的值的升序排序数列指的是**一个从左到右元素依次增大**的序列. 例如, 一个有序的数列 A, B, C, D 表示 $A < B, B < C, C < D$.

现在, 给定 m 个形如 $A < B$ 的关系, 判断是否能够根据这些关系确定这个数列的顺序.

输出形式如下:

- 若根据前 x 个关系即可确定这 n 个元素的顺序, 输出

`Sorted sequence determined after x relations: ...`

- 若根据前 x 个关系即发现存在矛盾, 输出

`Inconsistency found after x relations.`

- 若根据这 m 个关系无法确定这 n 个元素的顺序, 输出

`Sorted sequence cannot be determined.`

$1 \le m \le 600$.

首先, 对于一个给定的关系 `relations`, 我们考虑如何判断「有唯一解, 有若干解还是无解」:

- 初始设定一个 `unique_tag` 用于记录答案是否唯一.
- 进行拓扑排序. 若在中途某个时刻, 队列 `queue` 中存在 ≥ 2 个元素, 这就意味着这俩元素之间没有制约关系, 可以随便取, 那么就让 `unique_tag` 置为 `false`.
- 拓扑排序完毕, 若发现拓扑到的点数 $< n$, 则意味着存在有向环, 导致环上的所有点无法被拓扑, 那么就返回**无解**;
- 否则, 根据 `unique_tag` 的值, 我们可以快速锁定答案是否唯一.

接着, 我们按照题意模拟即可, 时间复杂度 $O(m^2)$. 完整代码如下:

```cpp
#include <bits/stdc++.h>
using namespace std;

const int N = 600;
```

[3] https://www.luogu.com.cn/problem/P1347

```
vector<int> idx;

int topo_sort(vector<int> *adj, int n) {
  idx.clear();

  vector<int> deg_in(n, 0);
  queue<int> q;

  for (int i = 0; i < n; i++) {
    for (auto j : adj[i])
      deg_in[j]++;
  }

  for (int i = 0; i < n; i++) {
    if (deg_in[i] == 0)
      q.push(i);
  }

  bool unique_tag = true;
  while (!q.empty()) {
    int u = q.front(); q.pop();
    idx.push_back(u);
    if (!q.empty()) unique_tag = false;
    for (auto v : adj[u]) {
      if (--deg_in[v] == 0)
        q.push(v);
    }
  }

  if (idx.size() != n) {
    return -1;
  } else {
    return unique_tag;
  }
}

int check(int n, auto relations, int t) {
  vector<int> adj[N];
  for (int i = 0; i < t; i++) {
    adj[relations[i][0]].push_back(relations[i][1]);
  }
  return topo_sort(adj, n);
}

int main() {
  int n, m;
  cin >> n >> m;
  vector<array<int, 2>> relations(m);
  for (int i = 0; i < m; i++) {
```

```
    char opt[5];
    cin >> opt;
    relations[i][0] = opt[0] - 'A';
    relations[i][1] = opt[2] - 'A';
    int status = check(n, relations, i + 1);
    if (status == -1) {
      cout << "Inconsistency found after " << (i + 1) << " relations." << endl;
      return 0;
    } else if (status == 1) {
      cout << "Sorted sequence determined after " << (i + 1) << " relations: ";
      for (auto it : idx)
        cout << char(it + 'A');
      cout << "." << endl;
      return 0;
    }
  }
  cout << "Sorted sequence cannot be determined." << endl;
  return 0;
}
```