Chapter 3

第3章 拓扑排序

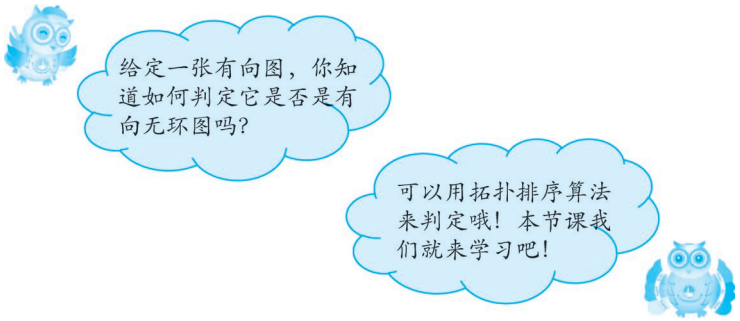
第 11 课　初识拓扑排序

导学牌

（1）理解有向无环图的含义。

（2）掌握拓扑排序与有向无环图的关系。

（3）掌握拓扑排序算法的基本思想及其算法实现。



学习坊

1. 有向无环图

在图论中，如果一个有向图无法从某个节点出发，经过若干条边又回到该节点，则这个图可称为有向无环图（directed acyclic graph，DAG）。

【例 11.1】　对于图 11.1(a)来说，图 G_1 是一个有向无环图；而对于图 11.1(b)来说，从节点 A 出发经过节点 D 和 B，又回到了节点 A，形成了一个环路，这就说明图 G_2 不是一个有向无环图。

(a) 图G_1　　(b) 图G_2

图　11.1

2. 拓扑排序

在有向图中，对所有节点进行排序，如果排序后能满足对于图中任意一条从节点 u 到节点 v 的边，都存在节点 u 排在节点 v 之前，那么称这个序列是图的一个拓扑序列，简称拓

扑序。如图 11.1(a)中，$BACD$ 和 $CBAD$ 都是图的拓扑序(拓扑序不唯一)。

如果一个图不是有向无环图，那么拓扑序是不存在的。因为如果有环，那么对于环路上的两个节点 u 和 v，既可以认为 u 在 v 之前，也可以认为 v 在 u 之前，它们之间的先后顺序是不确定的，这就与拓扑序的含义相矛盾了。如图 11.1 (b)中，既可以认为 A 在 D 之前，也可以认为 D 在 A 之前。

定理：当且仅当一张图是有向无环图时它才可以进行拓扑排序。

3. 拓扑排序算法

Kahn 算法是一种最常用的拓扑排序算法。它是基于贪心的算法思想。其基本思路是首先维护一个入度为 0 的节点集合，然后从该集合中选取一个节点进行处理，即将该节点放入结果序列中，并删除该节点及其引出的所有关联边。再更新相关联节点的入度，如果一个节点的入度更新后变为 0，则将该节点加入到入度为 0 的节点集合中。重复这个过程，直到入度为 0 的节点集合为空。最后，若图中仍有未处理的节点，说明该图存在环路，无法进行拓扑排序；否则可以进行拓扑排序，且结果序列就是该图的一个拓扑序。

【例 11.2】 给定一张有向无环图 G，如图 11.2 所示，求该图的一个拓扑序列。

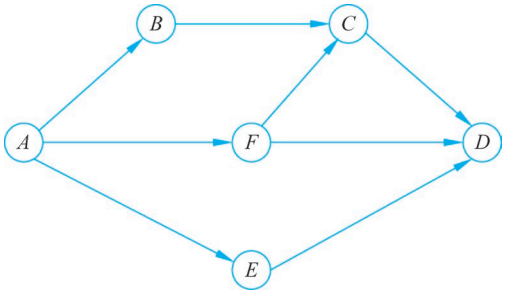


图 11.2

根据拓扑排序算法的算法思想，步骤及图示过程如下。

(1) 将节点 A 加入入度为 0 的节点集合。

(2) 选取节点 A 进行处理，即将节点 A 放入结果序列中，此时结果序列为$\{A\}$，再删除节点 A 及其引出的所有关联边，如图 11.3(a)所示。

(3) 更新入度为 0 的节点集合，即将节点 B、E、F 加入该集合。

(4) 依次选取节点 B、E、F 进行处理。首先选取节点 B，将其放入结果序列中，此时结果序列为$\{A,B\}$，再删除节点 B 及其引出的所有关联边，如图 11.3(b)所示。类似地，依次选取节点 E、F 进行处理，处理后如图 11.3(c)和图 11.3(d)所示。此时的结果序列为$\{A,B,E,F\}$。

(5) 更新入度为 0 的节点结合，即将节点 D 加入该集合。

(6) 选取节点 D 进行处理，如图 11.3(e)所示。处理完节点 D 之后，图中已无再需处理的节点，且入度为 0 的节点集合已为空，执行过程到此结束。此时的结果序列为$\{A,B,E,F,C,D\}$。

由上述分析可知，序列$\{A,B,E,F,C,D\}$就是图 11.2 的一个拓扑序列。

【例 11.3】 给定一张有向图 G，如图 11.4 所示，检测该图是否存在环路。

根据定理当且仅当一张图是有向无环图时，该图可以进行拓扑排序。因此，仅需要判断

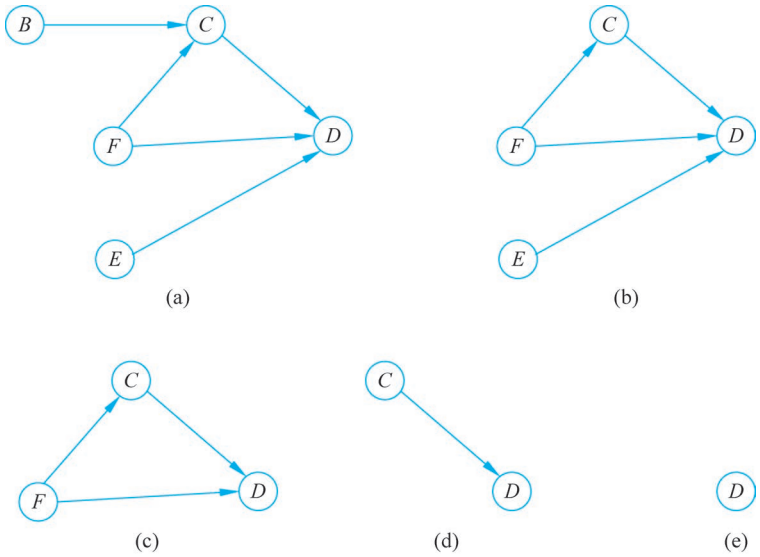


图 11.3

图 G 是否可以进行拓扑排序，若可以，说明图 G 不存在环路；否则，该图存在环路。

根据拓扑排序算法的算法思想，具体步骤如下。

(1) 将节点 A 加入入度为 0 的节点集合。

(2) 选取节点 A 进行处理，即将节点 A 放入结果序列中，再删除节点 A 及其引出的所有关联边，如图 11.5 所示。

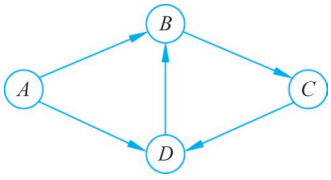


图 11.4

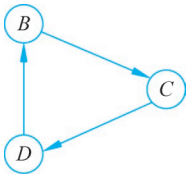


图 11.5

(3) 由于节点{B,C,D}的入度均不为零，所以已无新节点可以加入入度为零的集合即该集合为空，执行过程到此结束。但图中仍有未处理的节点{B,C,D}，这就说明图 G 是无法进行拓扑排序的，即该图存在环路。

4. 拓扑排序的算法实现

拓扑排序的算法通常使用队列(queue)数据结构来实现。假设在一个有向无环图 $G=(V,E)$，其中 V 表示点集，E 表示边集，节点 u 和 $v\in V$ 且有 u 连向 v 的边$(u\rightarrow v)\in E$，该图的一个拓扑序的具体实现步骤如下。

(1) 用数组 $d[u]$记录节点 u 的入度数。

(2) 用一个序列 ans 记录拓扑序，初始时为空。

(3) 用一个队列 Q 记录所有加入拓扑序的点，初始时为空。

(4) 将所有入度 $d[u]=0$ 的节点加入队列 Q。

(5) 若队列 Q 非空,取出 Q 头部的元素,加入序列 ans,然后删除节点 u,同时对于所有点 u 连向的点 v,令 $d[u]=d[v]-1$(删除以节点 u 为起点的所有关联边),若造成新的点的入度为 0,则将点加入 Q。

(6) 重复第(5)点直到队列 Q 为空。

(7) 输出序列 ans。

【例 11.4】 给定一张有向图 G,要求找到 G 的一个拓扑序。如果 G 不是一个有向无环图,输出 -1 即可。

输入: 第一行,两个整数 n 和 m($1\leqslant n,m\leqslant 10^5$),表示图 G 有 n 个点和 m 条边。接下来 m 行,分别表示 m 条边的两个端点。

输出: 如果图 G 可以拓扑排序,输出一个拓扑序列;否则,输出 -1。

样例输入 1:

```
4 4
1 2
1 3
2 4
3 4
```

样例输出 1:

```
1 2 3 4
```

样例输入 2:

```
4 4
1 2
2 3
3 4
4 2
```

样例输出 2:

```
-1
```

算法解析:

根据题意,可以使用拓扑排序算法来判断有向图 G 是否有环。如果有环,输出 -1;否则,输出图 G 的一个拓扑序。具体实现过程如上文“拓扑排序的算法实现”中所示,此处略。

以样例 1 和样例 2 为例,可以画出如图 11.6(a)和图 11.6(b)所示的图。

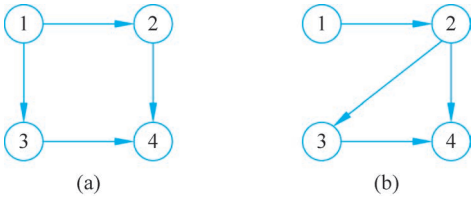


图 11.6

显然,根据上述拓扑排序算法的思想,可以得出序列{1,2,3,4}和{1,3,2,4}均是图 11.6(a)的拓扑序;而图 11.6(b)不是一个有向无环图,因此无法进行拓扑排序,所以输出 -1 即可。

参考程序:

根据以上算法解析,可以编写程序如图 11.7 所示。

```
#include<bits/stdc++.h>
using namespace std;
const int maxn=1e6+5;
vector<int> G[maxn],ans;
queue<int> q;
int n,m,d[maxn];
void toposort(){
    for(int i=1;i<=n;i++) if(d[i]==0) q.push(i);
    while(!q.empty()){
        int u=q.front(); q.pop();  //取出队首元素，删除队首元素
        ans.push_back(u);          //将队首元素加入结果序列ans
        for(int i=0;i<G[u].size();i++){
            int v=G[u][i];
            d[v]--;                //删除u连向的v，即v的入度减1
            if(d[v]==0) q.push(v); //将新增入度为0的点加入队列q
        }
    }
    if(ans.size()==n){
        for(int i=0;i<n;i++) cout<<ans[i]<<" ";
        cout<<endl;
    }else cout<<"-1"<<endl;
}
int main(){
    cin>>n>>m;
    for(int i=0;i<m;i++){
        int u,v; cin>>u>>v;
        G[u].push_back(v);
        d[v]++;
    }
    toposort();
    return 0;
}
```

图　11.7

运行结果：

```
4 4
1 2
1 3
2 4
3 4
1 2 3 4
```

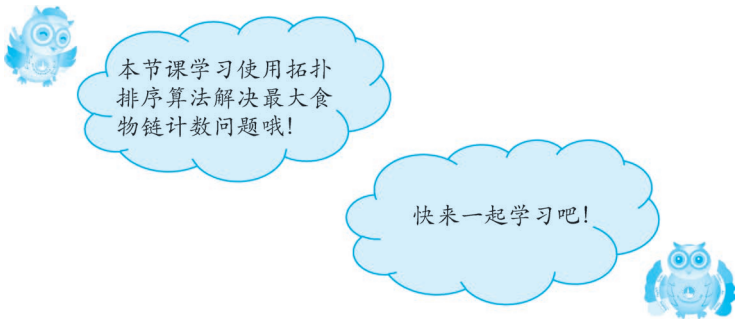
思考：对于样例 1 来说，为什么运行结果是序列{1,2,3,4}，而非序列{1,3,2,4}呢？

运行结果与输入时的顺序有关，如果将样例 1 的输入顺序改为先输入(1,3)，再输入(1,2)，那么运行结果将会随之改变为序列{1,3,2,4}。当然，通常情况下，找到图的一个拓扑序即可。

第 12 课　最大食物链计数

导学牌

学会使用拓扑排序算法解决最大食物链计数问题。



学习坊

【例 12.1】 最大食物链计数。给出一个食物网，要求求出这个食物网中最大食物链的数量。最大食物链是指生物学意义上的食物链，即最左端是不会捕食其他生物的生产者，最右端是不会被其他生物捕食的消费者。

由于这个结果可能过大，只需要输出总数模上 80112002 的结果。

输入： 第一行，两个正整数 n、m，表示生物种类 n 和吃与被吃的关系数 m。接下来 m 行，每行两个正整数 A 和 B，表示被吃的生物 A 和吃 A 的生物 B。

输出： 一行一个整数，为最大食物链数量模 80112002 的结果。

说明： 数据中不会出现环，满足生物学的要求。对于 20％的数据，$n\leqslant 40$，$m\leqslant 400$；对于 40％的数据，$n\leqslant 100$，$m\leqslant 2000$；对于 60％的数据，$n\leqslant 1000$，$m\leqslant 60000$；对于 80％的数据，$n\leqslant 2000$，$m\leqslant 200000$；对于 100％的数据，$n\leqslant 5000$，$m\leqslant 500000$。

注： 题目出自 https://www.luogu.com.cn/problem/P4017。

样例输入：

```
5 7
1 2
1 3
2 3
3 5
2 5
4 5
3 4
```

样例输出：

```
5
```

算法解析：

根据题意，首先可以将这张食物网抽象成一张有向图。在食物网上寻找一条最大食物链，也就是在这张有向图上，寻找一条最大的路径，所谓最大路径是指起点没有入度（入度为0），终点没有出度（出度为0）的路径。然后统计这种最大路径的数量，答案可能很大，要求对 80112002 取模。这是一道图上 DP 计数问题。

首先，定义状态 dp[u]：用来表示从合法起点（入度为 0 的点）出发走到 u 的路径数量。

然后，计算状态 dp[u]：首先对图进行拓扑排序，然后按拓扑序计算每个 dp[u]值。假设所有连向 u 的节点集合为 H_u，则有 $dp[u]=\sum_{v\in H_u} dp[v]$（$v$ 表示所有连向到 u 的点）。

最后，答案 ans 为所有合法终点（出度为 0 的点）的 dp 值之和。

处理边界情况：让所有合法起点（入度为 0 的点）的 dp 值为 1，表示从入度为 0 的点走到自己这个点有一种方案数。其他 dp 值为 0。

以样例为例，可以画出如图 12.1 所示的图。

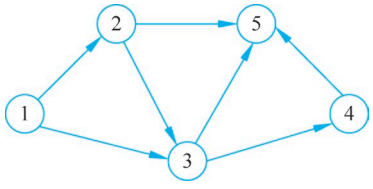


图 12.1

首先，对图 12.1 进行拓扑排序。

然后，根据拓扑序不断更新 dp[u]的值，有 $dp[u]=\sum_{v\in H_u} dp[v]$（$v$ 表示所有连向 u 的点），计算具体如下。

① dp[1]=1　//表示能够走到 1 的路径之和，即(1→1)，共 1 条。

② dp[2]=dp[1]=1　//表示(1→2)的路径之和，共 1 条。

③ dp[3]=dp[1]+dp[2]=2　//表示(1→3)和(2→3)的路径之和，共 2 条。

④ dp[4]=dp[3] =2　//表示(3→4)的路径之和，共 2 条。

⑤ dp[5]=dp[2]+dp[3]+dp[4] =5　//表示(2→5)、(3→5)、(4→5)的路径之和，共 5 条。

最后，ans 为所有合法终点（出度为 0 的点）的 dp[u]之和，而该样例中，出度为 0 点只有 5 号点（见图 12.1），因此 ans=dp[5]=5。

编写程序：

根据以上算法解析，可以编写程序如图 12.2 所示。

```
00  #include<bits/stdc++.h>
01  using namespace std;
02  const int maxn=1e6+10,mod=80112002;
03  vector<int> G[maxn],H[maxn],ans;
04  int n,m,d[maxn],dp[maxn];
05  queue<int> q;
06  void toposort(){
07      for(int i=1;i<=n;i++) if(!d[i]) q.push(i);
08      while(!q.empty()){
09          int u=q.front(); q.pop();
10          ans.push_back(u);
11          for(int i=0;i<G[u].size();i++){
12              int v=G[u][i]; d[v]--;
13              if(d[v]==0) q.push(v);
14          }
15      }
16  }
```

图 12.2

```
int main(){
    cin>>n>>m;
    for(int i=1;i<=m;i++){
        int u,v; cin>>u>>v;
        G[u].push_back(v); d[v]++;
        H[v].push_back(u); //反向边
    }
    toposort();
    for(int i=0;i<ans.size();i++){
        int u=ans[i];
        if(H[u].size()==0) dp[u]=1;
        for(int j=0;j<H[u].size();j++){
            int v=H[u][j];
            dp[u]=(dp[u]+dp[v])%mod;
        }
    }
    int Ans=0;
    for(int i=0;i<=ans.size();i++)
      if(G[i].size()==0) Ans=(Ans+dp[i])%mod;
    cout<<Ans<<endl;
    return 0;
}
```

图 12.2(续)

运行结果：

```
5 7
1 2
1 3
2 3
3 5
2 5
4 5
3 4
5
```

第 13 课　最　长　路

导学牌

学会使用拓扑排序算法解决最长路问题。

学习坊

【例 13.1】 最长路。设 G 为有 n 个节点的带权有向无环图，G 中各节点的编号为 1 到 n，请设计算法，计算图 G 中 1，n 间的最长路径。

输入： 第一行有两个整数，分别代表图的点数 n 和边数 m。第二行到第 $m+1$ 行，每行 3 个整数 u，v，$w(u<v)$，代表存在一条从 u 到 v 边权为 w 的边。

输出： 一行一个整数，代表 1 到 n 的最长路。若 1 无法到达 n，请输出 -1。

说明： 对于 20% 的数据，$n\leqslant 100$，$m\leqslant 10^3$；对于 40% 的数据，$n\leqslant 10^3$，$m\leqslant 10^4$；对于 100% 的数据，$1\leqslant n\leqslant 1500$，$0\leqslant m\leqslant 5\times 10^4$，$1\leqslant u,v\leqslant n$，$-10^5\leqslant w\leqslant 10^5$。

注： 题目出自 https://www.luogu.com.cn/problem/P1807。

样例输入：

```
4 4
1 2 3
1 3 2
2 4 3
3 4 5
```

样例输出：

```
7
```

算法解析：

根据题意，由于图 G 是一个带权值的 DAG，因此很容易想到这是一道拓扑排序＋DP 问题。即先对图 G 进行拓扑排序，再按拓扑序进行 DP。

定义状态 dp[u]：用来表示从 1 到 u 的最长路径长度。