Algorithm Design VI

Decompositions of Graphs

Guoqiang Li
School of Software, Shanghai Jiao Tong University

SHANGHAI JIAO TONG UNIVERSITY

An Exercise

Let B be an $n \times n$ chessboard, where n is a power of 2. Use a divide-and-conquer argument to describe how to cover all squares of B except one with L-shaped tiles. For example, if $n = 2$, then there are four squares three of which can be covered by one L-shaped tile, and if $n = 4$, then there are 16 squares of which 15 can be covered by 5 L-shaped tiles.

分治法：先将n x n的棋盘分为四个(n / 2) * (n / 2)的子棋盘，在除去右下角的棋盘之外的三个棋盘中，递归的求解其需要的L形状的个数，之后将这三个产生的那一个未被覆盖的格子通过旋转，拼接成一个新的L形状，因而，总的需要的L形状的个数为：4 * N + 1，其中N为每一个子棋盘需要的L型个数。

Decompositions of Graphs

Exploring Graphs

EXPLORE(G, v)
input : $G = (V, E)$ is a graph; $v \in V$
output: $visited(u)$ to $true$ for all nodes u reachable from v

$visited(v) = true$;
PREVISIT(v); (此步骤相当于将节点v入栈)
for *each edge* $(v, u) \in E$ **do**
  if *not* $visited(u)$ **then** EXPLORE(G, u);
end
POSTVISIT(v); (此步骤相当于将节点v出栈，即进行节点v的回溯)

此步骤相当于递归的将与v节点直接相连的节点入栈，并递归的将间接连结的节点入栈。

一个EXPLORE算法是找到所有与参数节点v相通(有路径可以到达的)的节点。

Depth-First Search

Shanghai Jiao Tong University

```
DFS(G)

for all v ∈ V do
 |  visited(v) = false;
end
for all v ∈ V do
 |  if not visited(v) then Explore(G, v);
end
```

Connectivity in Undirected Graphs

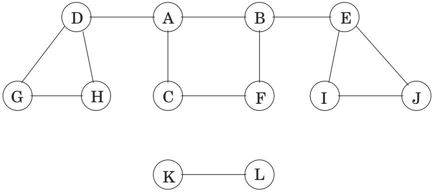
Types of Edges in Undirected Graphs

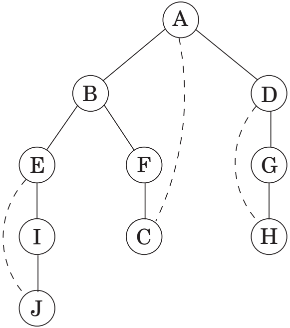
Those edges in G that are traversed by `EXPLORE` are tree edges.

Types of Edges in Undirected Graphs

Those edges in G that are traversed by `EXPLORE` are tree edges.

The rest are back edges.





Connectivity in Undirected Graphs

Definition

An undirected graph is connected, if there is a path between any pair of vertices.

Connectivity in Undirected Graphs

Definition

An undirected graph is connected, if there is a path between any pair of vertices.

Definition

A connected component is a subgraph that is internally connected but has no edges to the remaining vertices.

Connectivity in Undirected Graphs

Definition

An undirected graph is connected, if there is a path between any pair of vertices.

Definition

A connected component is a subgraph that is internally connected but has no edges to the remaining vertices.

When `EXPLORE` is started at a particular vertex, it identifies precisely the connected component containing that vertex.

Each time the DFS outer loop calls `EXPLORE`, a new connected component is picked out.

Connectivity in Undirected Graphs

DFS is trivially adapted to check if a graph is connected.

DFS is trivially adapted to check if a graph is connected.

More generally, to assign each node v an integer $ccnum[v]$ identifying the connected component to which it belongs.

```
PREVISIT(v)
```

$ccnum[v] = cc;$

where cc needs to be initialized to zero and to be incremented each time the DFS procedure calls EXPLORE.

DFS每次调用都会找出一个完整的连通分量，所以一次新的对于EXPLORE的调用就将对应于一个新的连通分量

Previsit and Postvisit Orderings

For each node, we will note down the times of two important events:

Previsit and Postvisit Orderings

For each node, we will note down the times of two important events:

- the moment of first discovery (corresponding to `PREVISIT`);
- and the moment of final departure (`POSTVISIT`).

Previsit and Postvisit Orderings

For each node, we will note down the times of two important events:

- the moment of first discovery (corresponding to PREVISIT);
- and the moment of final departure (POSTVISIT).

PREVISIT(v)

$pre[v] = clock;$
$clock++;$

POSTVISIT(v)

$post[v] = clock;$
$clock++;$

Previsit and Postvisit Orderings

For each node, we will note down the times of two important events:

- the moment of first discovery (corresponding to PREVISIT);
- and the moment of final departure (POSTVISIT).

PREVISIT(v)

$pre[v] = clock;$
$clock++;$

POSTVISIT(v)

$post[v] = clock;$
$clock++;$

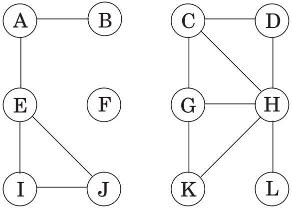
Lemma

For any nodes u and v, the two intervals $[pre(u), post(u)]$ and $[pre(v), post(v)]$ are either disjoint or one is contained within the other.

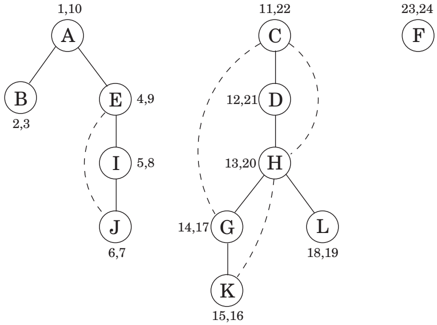
包含的情况：从u出发可以到达v或者从v出发到达了u。对于第一种情况，由于递归，v进入对应的prevVisit肯定会晚于u节点，而v节点的PostVisit的时间一定会早于u节点。
不相交的情况：u节点与v节点之间没有路径联通。

Previsit and Postvisit Orderings

(a)



(b)



Connectivity in Directed Graphs

Types of Edges in Directed Graphs

DFS yields a search tree/forests.

Types of Edges in Directed Graphs

DFS yields a search tree/forests.

- root.

Types of Edges in Directed Graphs

SHANGHAI JIAO TONG UNIVERSITY

DFS yields a search tree/forests.

- root.
- descendant and ancestor.

Types of Edges in Directed Graphs

DFS yields a search tree/forests.

- root.
- descendant and ancestor.
- parent and child.

Types of Edges in Directed Graphs

DFS yields a search tree/forests.

- root.
- descendant and ancestor.
- parent and child.
- Tree edges are actually part of the DFS forest.

Types of Edges in Directed Graphs

DFS yields a search tree/forests.

- root.
- descendant and ancestor.
- parent and child.
- Tree edges are actually part of the DFS forest.
- Forward edges lead from a node to a nonchild descendant in the DFS tree.

Types of Edges in Directed Graphs

DFS yields a search tree/forests.

- root.
- descendant and ancestor.
- parent and child.
- Tree edges are actually part of the DFS forest.
- Forward edges lead from a node to a nonchild descendant in the DFS tree.
- Back edges lead to an ancestor in the DFS tree.

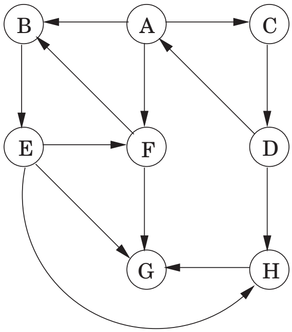
Types of Edges in Directed Graphs

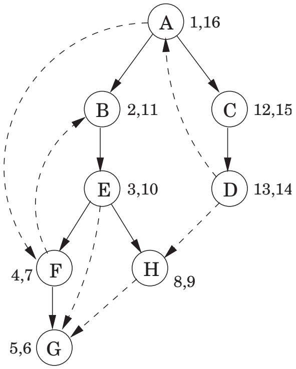
DFS yields a search tree/forests.

- root.
- descendant and ancestor.
- parent and child.
- Tree edges are actually part of the DFS forest. (tree edge:DFS遍历过程中会经过的边)
- Forward edges lead from a node to a nonchild descendant in the DFS tree. (要注意是非孩子节点的后继节点)
- Back edges lead to an ancestor in the DFS tree.
- Cross edges lead to neither descendant nor ancestor.

对于横跨边u->v,v一定是先被DFS访问的，不然的话这个就不是cross edge而是一个tree edge

Directed Graphs





Types of Edges

SHANGHAI JIAO TONG UNIVERSITY

pre/post ordering for (u, v)				Edge type
$[_u$	$[_v$	$]_v$	$]_u$	Tree/forward
$[_v$	$[_u$	$]_u$	$]_v$	Back
$[_v$	$]_v$	$[_u$	$]_u$	Cross

Types of Edges

SHANGHAI JIAO TONG UNIVERSITY

pre/post ordering for (u, v)				Edge type
$[_u$	$[_v$	$]_v$	$]_u$	Tree/forward
$[_v$	$[_u$	$]_u$	$]_v$	Back
$[_v$	$]_v$	$[_u$	$]_u$	Cross

Q: Is that all?

是的。因为对于边u-->v不可能存在uuvv的时间情况。
证明:对于边u-->v,有如下几种情况：
1.v是u的后继，则这条边是一个tree/forward edge，因而应该是uvvu
2.v是u的前驱，则这条边应该是一个back edge，因而应该是vuuv
3.v与u之间无path，则路径u-->v是一个cross edge，因而应该是vvuu，而不能是uuvv，否则的话，若u先访问而u到v有一条边，也应该是uvvu(情况1)
综上，是不存在uuvv这种情况的

Directed Acyclic Graphs (DAG)

Definition

A cycle in a directed graph is a circular path

$$v_0 \to v_1 \to v_2 \to \ldots v_k \to v_0$$

Directed Acyclic Graphs (DAG)

Definition

A cycle in a directed graph is a circular path

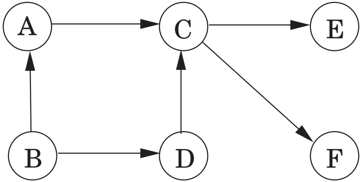
$$v_0 \to v_1 \to v_2 \to \dots v_k \to v_0$$

Lemma

A directed graph has a cycle if and only if its depth-first search reveals a back edge.

证明：1."=>":因为存在环路，所以有v0->v1->....vk->v0.设通过DFS当前访问到了节点vi(i <= k).则有之后的DFS路径为vi->vi+1->...v0->v1->...v(i-1)。所以可以发现，v(i-1)是vi的前驱，但是vi有一条到v(i-1)的边，则这条边就是一个back edge
2."<=":设这个back edge为u-->v。则根据back edge定义可以知道u是v的后继(不一定是直接后继)。所以存在路径v->v1->...->u是节点v到节点u的tree edge path。所以这个路径与边u->v就构成了一个从u出发又回到u的环路，
所以，综上，"<==>"(if and only if)成立

Directed Acyclic Graphs (DAG)



Directed Acyclic Graphs (DAG)

Linearization/Topologically Sort: Order the vertices such that every edge goes from a earlier vertex to a later one.

Directed Acyclic Graphs (DAG)

Linearization/Topologically Sort: Order the vertices such that every edge goes from a earlier vertex to a later one.

Q: What types of dags can be linearized?

Directed Acyclic Graphs (DAG)

Linearization/Topologically Sort: Order the vertices such that every edge goes from a earlier vertex to a later one.

Q: What types of dags can be linearized?

A: All of them.

Directed Acyclic Graphs (DAG)

Linearization/Topologically Sort: Order the vertices such that every edge goes from a earlier vertex to a later one.

Q: What types of dags can be linearized?

A: All of them.

DFS tells us exactly how to do it: perform tasks in decreasing order of their post numbers.

Directed Acyclic Graphs (DAG)

SHANGHAI JIAO TONG UNIVERSITY

DAG的线性序列/拓扑排序

Linearization/Topologically Sort: Order the vertices such that every edge goes from a earlier vertex to a later one.

Q: What types of dags can be linearized?

A: All of them.

DFS tells us exactly how to do it: perform tasks in decreasing order of their post numbers.

The only edges (u, v) in a graph for which $post(u) < post(v)$ are back edges, and we have seen that a DAG cannot have back edges.

证明：根据前述内容可以知道，拓扑排序的定义时使得每一条边u-->v,都有u的序列编号小于v的。而对于一个图中的edge，其post关系只有uvvu，vuuv，vvuu几种。而对于DAG，其中不存在back edge，所以只有uvvu和vvuu两种情况。所以在DAG中的任意一条边u-->v,都有u的post > v的post。所以通过对于post进行降序排列得到拓扑序列的方式是合理的

Directed Acyclic Graphs (DAG)

Lemma

In a DAG, every edge leads to a vertex with a lower post number.

Directed Acyclic Graphs (DAG)

There is a linear-time algorithm for ordering the nodes of a DAG.

Directed Acyclic Graphs (DAG)

There is a linear-time algorithm for ordering the nodes of a DAG.

Acyclicity, linearizability, and the absence of back edges during a depth-first search - are the same thing.

Directed Acyclic Graphs (DAG)

There is a linear-time algorithm for ordering the nodes of a DAG.

Acyclicity, linearizability, and the absence of back edges during a depth-first search - are the same thing.

The vertex with the smallest post number comes last in this linearization, and it must be a sink - no outgoing edges.

Directed Acyclic Graphs (DAG)

SHANGHAI JIAO TONG UNIVERSITY

There is a linear-time algorithm for ordering the nodes of a DAG. (就相当于进行了一次DFS并同时进行time的编号)

Acyclicity, linearizability, and the absence of back edges during a depth-first search - are the same thing.

The vertex with the smallest post number comes last in this linearization, and it must be a sink - no outgoing edges.

Symmetrically, the one with the highest post is a source, a node with no incoming edges.

sink：溯，只进不出
source：源，只出不进

Directed Acyclic Graphs (DAG)

Lemma

Every DAG has at least one source and at least one sink.

Directed Acyclic Graphs (DAG)

Lemma

Every DAG has at least one source and at least one sink.

The guaranteed existence of a source suggests an alternative approach to linearization:

Directed Acyclic Graphs (DAG)

Lemma

Every DAG has at least one source and at least one sink.

The guaranteed existence of a source suggests an alternative approach to linearization:

1. Find a source, output it, and delete it from the graph.

SHANGHAI JIAO TONG UNIVERSITY

Lemma

Every DAG has at least one source and at least one sink.

The guaranteed existence of a source suggests an alternative approach to linearization:

1. Find a source, output it, and delete it from the graph.
2. Repeat until the graph is empty.

使用邻接表较好，同时用一个队列来记录结果序列，这里的"删除"可以通过遍历要移除节点对应的领接表行来实现。即，将里面的每个节点的入度-1

Strongly Connected Components

(强联通分量)

Defining Connectivity for Directed Graphs

Definition

Two nodes u and v of a directed graph are connected if there is a path from u to v and a path from v to u.

Defining Connectivity for Directed Graphs

Definition

Two nodes u and v of a directed graph are connected if there is a path from u to v and a path from v to u.

This relation partitions V into disjoint sets that we call strongly connected components (SCC).

Defining Connectivity for Directed Graphs

SHANGHAI JIAO TONG UNIVERSITY

Definition

Two nodes u and v of a directed graph are connected if there is a path from u to v and a path from v to u.

一个SCC可以理解为一个有向图的最大联通子图

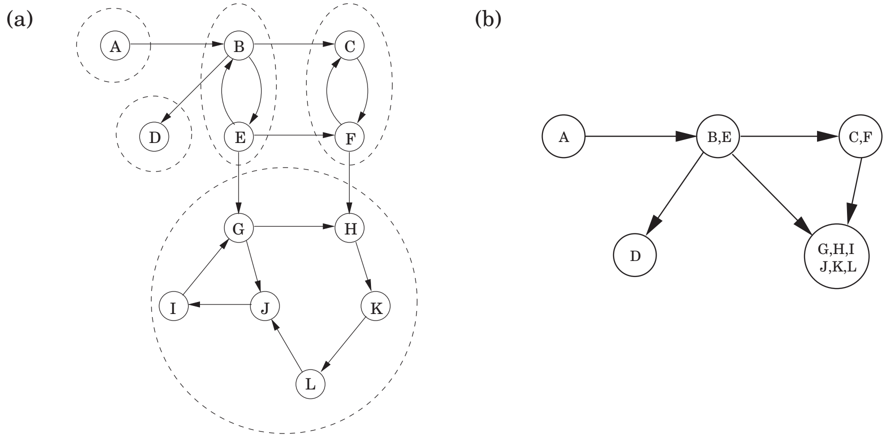
This relation partitions V into disjoint sets that we call strongly connected components (SCC).

Lemma

Every directed graph is a DAG of its SCC.

这个lemma说的是将一个有向图中的每一个SCC都收缩为一个节点，则收缩之后的图为一个DAG。证明：直接反证。如果收缩之后的图不是DAG，则其中存在环路，则对于两个SCC，就说明其中里至少各存在一个点是相互联通的，所以这两个SCC本可以合并，而这与SCC的定义是矛盾的(因为此时SCC不再是最大的连通子图了)。所以一定是一个DAG

Strongly Connected Components



An Efficient Algorithm

Lemma

If the `EXPLORE` subroutine at node u, then it will terminate precisely when all nodes reachable from u have been visited.

An Efficient Algorithm

Lemma

If the `EXPLORE` *subroutine at node* u*, then it will terminate precisely when all nodes reachable from* u *have been visited.*

If we call explore on a node that lies somewhere in a sink SCC, then we will retrieve exactly that component.

An Efficient Algorithm

Lemma

If the `EXPLORE` *subroutine at node* u, *then it will terminate precisely when all nodes reachable from* u *have been visited.*

If we call explore on a node that lies somewhere in a sink SCC, then we will retrieve exactly that component.

We have two problems:

An Efficient Algorithm

Lemma

If the `EXPLORE` *subroutine at node* u*, then it will terminate precisely when all nodes reachable from* u *have been visited.*

If we call explore on a node that lies somewhere in a sink SCC, then we will retrieve exactly that component.

We have two problems:

1. How do we find a node that we know for sure lies in a sink SCC?

An Efficient Algorithm

SHANGHAI JIAO TONG UNIVERSITY

Lemma

If the EXPLORE *subroutine at node u, then it will terminate precisely when all nodes reachable from u have been visited.*

这个与无向图是类似的，对于一个节点u的一次EXPLORE结果是返回所有u能到达的节点

If we call explore on a node that lies somewhere in a sink SCC, then we will retrieve exactly that component. (SCC)

We have two problems:

1. How do we find a node that we know for sure lies in a sink SCC?
2. How do we continue once this first component has been discovered?

An Efficient Algorithm

Lemma

The node that receives the highest post number in a depth-first search must lie in a source SCC.

An Efficient Algorithm

Lemma

The node that receives the highest post number in a depth-first search must lie in a source SCC.

Lemma

If C and C' are SCC, and there is an edge from a node in C to a node in C', then the highest post number in C is bigger than the highest post number in C'.

An Efficient Algorithm

Shanghai Jiao Tong University

Lemma

The node that receives the highest post number in a depth-first search must lie in a source SCC.

Lemma

If C and C' are SCC, and there is an edge from a node in C to a node in C', then the highest post number in C is bigger than the highest post number in C'.

Hence the SCCs can be linearized by arranging them in decreasing order of their highest post numbers.

对于上述第二个lemma的证明：对于lemma中存在的C-->C'的边，设其为u-->v.则只有u先被访问与v先被访问两种情况。对于u先被访问的情况，在u通过边u-->v到达C'之后，v会将C'中所有点都访问完之后再返回点u(DFS算法),所以u的post num大于C'中任意一个post num。而u是先被访问的，所以C的最大post一定会大于等于u的post，所以C的post_max > C'的post_max
对于v先被访问的情况，则u只能被后访问(因为存在u到v的一条边，且为一个DAG)，所以C的post_max一定大于C'的post_max。
同时，根据上述的lemma2，还可以知道最大的post_num对应的点一定在source SCC中，因为在对于DFS进行拓扑一定是从一个source SCC开始的。

Solving Problem A

Consider the reverse graph G^R, the same as G but with all edges reversed.

Solving Problem A

Consider the reverse graph G^R, the same as G but with all edges reversed.

G^R has exactly the same SCCs as G.

Solving Problem A

Consider the reverse graph G^R, the same as G but with all edges reversed.

G^R has exactly the same SCCs as G.

If we do a depth-first search of G^R, the node with the highest post number will come from a source SCC in G^R.

Solving Problem A

转置图：构成的SCC不变，但是post_num会反过来，可以用转置图的方式来找到原图的sink SCC(即转置图中的source SCC)

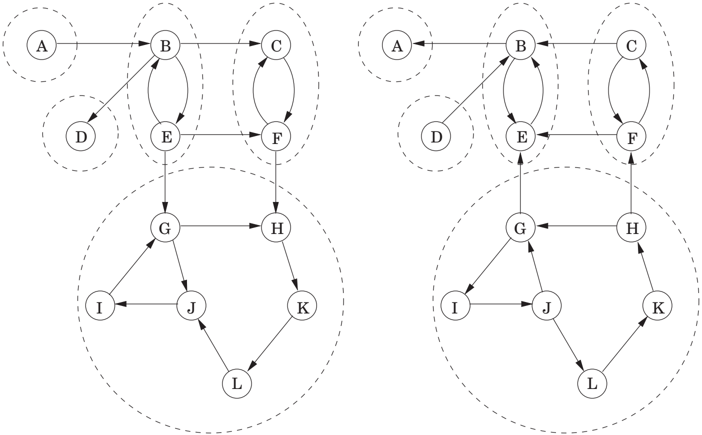
Consider the reverse graph G^R, the same as G but with all edges reversed.

G^R has exactly the same SCCs as G.

If we do a depth-first search of G^R, the node with the highest post number will come from a source SCC in G^R.

It is a sink SCC in G.

Strongly Connected Components



Solving Problem B

Once we have found the first SCC and deleted it from the graph, the node with the highest post number among those remaining will belong to a sink SCC of whatever remains of G.

Solving Problem B

Once we have found the first SCC and deleted it from the graph, the node with the highest post number among those remaining will belong to a sink SCC of whatever remains of G.

Therefore we can keep using the post numbering from our initial depth-first search on G^R to successively output the second strongly connected component, the third SCC, and so on.

The Linear-Time Algorithm

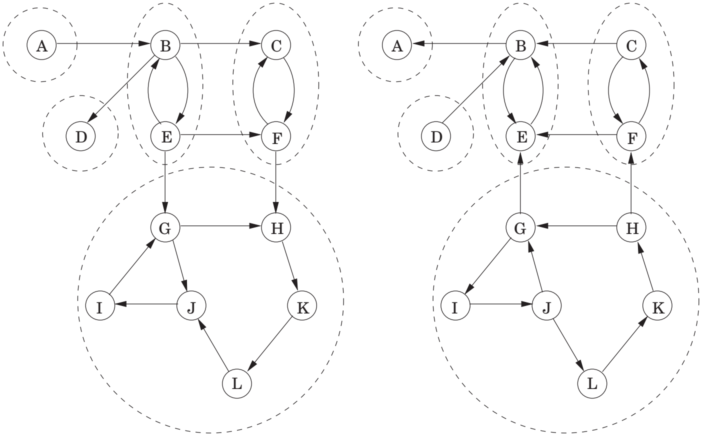
Shanghai Jiao Tong University

1. Run depth-first search on G^R.

The Linear-Time Algorithm

1. Run depth-first search on G^R.
2. Run the EXPLORE algorithm on G, and during the depth-first search, process the vertices in decreasing order of their post numbers from step 1.

Strongly Connected Components



Think About

How the SCC algorithm works when the graph is very, very huge?

Think About

How about edges instead of paths?

Exercises

Exercises 1

Suppose a CS curriculum consists of n courses, all of them mandatory. The prerequisite graph G has a node for each course, and an edge from course v to course w if and only if v is a prerequisite for w. Find an algorithm that works directly with this graph representation, and computes the minimum number of semesters necessary to complete the curriculum (assume that a student can take any number of courses in one semester). The running time of your algorithm should be linear.

考的是拓扑排序过程。就是在每次将最新的一批课程(即当前的sources)放入队列时将结果计数器加1，因为这些课程没有先修课程，可以一学期一起上完(QAQ),所以当最终拓扑排序结束的时候就得到了最少需要的学期数量。

Exercises 2

Give an efficient algorithm which takes as input a directed graph $G = (V, E)$, and determines whether or not there is a vertex $s \in V$ from which all other vertices are reachable.

先找出SCC(O(N)),在对于SCC收缩之后的DAG进行DFS(O(N)),如果能找到DAG中的某一个点能到达其他所有的节点，则就存在这样的v，并且这个v可以是对应的DAG节点对应的SCC中的任意一个节点。