Graph

pengym

Yali High School

October 26, 2018

Preface

今天上午主要讲的是*noip*提高组中涉及到的图论相关**基础**知识.

Preface

今天上午主要讲的是*noip*提高组中涉及到的图论相关**基础**知识.

由于讲课的人实在是太菜了

Preface

今天上午主要讲的是*noip*提高组中涉及到的图论相关**基础**知识.

由于讲课的人实在是太菜了

因此,主要是讲知识点以及一些**简单**的题目为主.

Preface

今天上午主要讲的是$noip$提高组中涉及到的图论相关**基础**知识.

由于讲课的人实在是太菜了

因此,主要是讲知识点以及一些**简单**的题目为主.

保证绝对没有任何**超纲**的知识. (OVO)

Preface

今天上午主要讲的是$noip$提高组中涉及到的图论相关**基础**知识.

由于讲课的人实在是太菜了

因此,主要是讲知识点以及一些**简单**的题目为主.

保证绝对没有任何**超纲**的知识. (OVO)

下午就由比我不知道强多少倍的$ylsoi$来讲一些图论进阶神题!!!

Content

- Scc/Bcc(Pbc/Ebc)

Content

- Scc/Bcc(Pbc/Ebc)
- Toposort

Content

- Scc/Bcc(Pbc/Ebc)
- Toposort
- Shortest Path

Content

- Scc/Bcc(Pbc/Ebc)
- Toposort
- Shortest Path
- Planar Graph

Content

- Scc/Bcc(Pbc/Ebc)
- Toposort
- Shortest Path
- Planar Graph
- Euler Path/Hamilton Path

Content

- Scc/Bcc(Pbc/Ebc)
- Toposort
- Shortest Path
- Planar Graph
- Euler Path/Hamilton Path
- MST

各种定义

强连通——在**有向图**中,若存在两个点u和v,满足u可以到达v,且v也可以到达u,那么则称u和v**强连通**.

各种定义

强连通——在**有向图**中,若存在两个点u和v,满足u可以到达v,且v也可以到达u,那么则称u和v**强连通**.

强连通图——在**有向图**中,若图中的每两个点都强连通,那么则称这个有向图为**强连通图**. (PS:也可以理解为任意两个点可达)

各种定义

强连通——在**有向图**中,若存在两个点u和v,满足u可以到达v,且v也可以到达u,那么则称u和v**强连通**.

强连通图——在**有向图**中,若图中的每两个点都强连通,那么则称这个有向图为**强连通图**. (PS:也可以理解为任意两个点可达)

强连通分量——在**有向图**中的极大强连通子图称为**强连通分量**.

各种定义

强连通——在**有向图**中,若存在两个点u和v,满足u可以到达v,且v也可以到达u,那么则称u和v**强连通**.

强连通图——在**有向图**中,若图中的每两个点都强连通,那么则称这个有向图为**强连通图**. (PS:也可以理解为任意两个点可达)

强连通分量——在**有向图**中的极大强连通子图称为**强连通分量**.

缩点——把**有向图**中的所有的强连通分量缩成一个点,并按照原来的连接关系相连.则称这个过程为缩点.

各种定义

强连通——在**有向图**中,若存在两个点u和v,满足u可以到达v,且v也可以到达u,那么则称u和v**强连通**.

强连通图——在**有向图**中,若图中的每两个点都强连通,那么则称这个有向图为**强连通图**. (PS:也可以理解为任意两个点可达)

强连通分量——在**有向图**中的极大强连通子图称为**强连通分量**.

缩点——把**有向图**中的所有的强连通分量缩成一个点,并按照原来的连接关系相连.则称这个过程为缩点.

DAG——有向无环图的英文缩写. (PS:有向图缩点后一定为一个DAG图)

缩点

缩点是图论中比较重要的思想.

缩点

缩点是图论中比较重要的思想.

通常,我们对一个有向有环图进行缩点后,会有意想不到的收获!

缩点

缩点是图论中比较重要的思想.

通常,我们对一个有向有环图进行缩点后,会有意想不到的收获!

而缩点的主流算法有两个:

缩点

缩点是图论中比较重要的思想.

通常,我们对一个有向有环图进行缩点后,会有意想不到的收获!

而缩点的主流算法有两个:

$$Kosaraju$$

$$Tarjan$$

缩点

缩点是图论中比较重要的思想.

通常,我们对一个有向有环图进行缩点后,会有意想不到的收获!

而缩点的主流算法有两个:

$$Kosaraju$$

$$Tarjan$$

这些算法想必你们早就会了.

缩点

缩点是图论中比较重要的思想.

通常,我们对一个有向有环图进行缩点后,会有意想不到的收获!

而缩点的主流算法有两个:

$$Kosaraju$$

$$Tarjan$$

这些算法想必你们早就会了.

~~(本着拖一拖的心态)~~本着整理复习的心态,我大概的叙述一下流程!

Kosaraju

$Kosaraju$是非常好理解,也是非常好实现的.

Kosaraju

$Kosaraju$是非常好理解,也是非常好实现的.

1. 首先,对于一个图,我们建好它的**正图**以及**反图**;

Kosaraju

$Kosaraju$是非常好理解,也是非常好实现的.

1. 首先,对于一个图,我们建好它的**正图**以及**反图**;

2. 进行第一遍dfs,对每一个连通块在正图上进行遍历,并记录好他们的dfs的出栈顺序.

Kosaraju

$Kosaraju$是非常好理解,也是非常好实现的.

1. 首先,对于一个图,我们建好它的**正图**以及**反图**;

2. 进行第一遍dfs,对每一个连通块在正图上进行遍历,并记录好他们的dfs的出栈顺序.

3. 对于出栈顺序,从后往前进行第二次dfs,对于所有在反图上能够遍历到且未被访问的点染色,代表它们处于同一个强连通分量中.

Kosaraju

$Kosaraju$是非常好理解,也是非常好实现的.

1. 首先,对于一个图,我们建好它的**正图**以及**反图**;

2. 进行第一遍dfs,对每一个连通块在正图上进行遍历,并记录好他们的dfs的出栈顺序.

3. 对于出栈顺序,从后往前进行第二次dfs,对于所有在反图上能够遍历到且未被访问的点染色,代表它们处于同一个强连通分量中.

4. 最后重新建图,就实现了缩点.

Kosaraju

$Kosaraju$是非常好理解,也是非常好实现的.

1. 首先,对于一个图,我们建好它的**正图**以及**反图**;

2. 进行第一遍dfs,对每一个连通块在正图上进行遍历,并记录好他们的dfs的出栈顺序.

3. 对于出栈顺序,从后往前进行第二次dfs,对于所有在反图上能够遍历到且未被访问的点染色,代表它们处于同一个强连通分量中.

4. 最后重新建图,就实现了缩点.

时间复杂度:$O(n+m)$

Tarjan

$Tarjan$缩点是Tarjan发明的众多算法之一.

Tarjan

$Tarjan$缩点是Tarjan发明的众多算法之一.

主要核心在处理dfn以及$lowlink$这两个数组.

Tarjan

$Tarjan$缩点是Tarjan发明的众多算法之一.

主要核心在处理dfn以及$lowlink$这两个数组.

$dfn[x]$——代表有向图中x的dfs序

Tarjan

$Tarjan$缩点是Tarjan发明的众多算法之一.

主要核心在处理dfn以及$lowlink$这两个数组.

$dfn[x]$——代表有向图中x的dfs序

$lowlink[x]$——代表有向图中x的所有子结点(包括x)所能到的最小的dfn

Tarjan

$Tarjan$缩点是Tarjan发明的众多算法之一.

主要核心在处理dfn以及$lowlink$这两个数组.

$dfn[x]$——代表有向图中x的dfs序

$lowlink[x]$——代表有向图中x的所有子结点(包括x)所能到的最小的dfn

处理出这两个数组后,我们的缩点就变得十分简单.

Tarjan

如果一个点的dfn与自己的$lowlink$相等,那么只有两种情况:

Tarjan

如果一个点的dfn与自己的$lowlink$相等,那么只有两种情况:

1. 不存在强连通分量,$lowlink$没有由子节点更新,就只有自己的dfn最小.

Tarjan

如果一个点的dfn与自己的$lowlink$相等,那么只有两种情况:

1. 不存在强连通分量,$lowlink$没有由子节点更新,就只有自己的dfn最小.

2. 存在强连通分量,因为子节点可以通过遍历边再次遍历到父亲节点.

Tarjan

如果一个点的dfn与自己的$lowlink$相等,那么只有两种情况:

1. 不存在强连通分量,$lowlink$没有由子节点更新,就只有自己的dfn最小.

2. 存在强连通分量,因为子节点可以通过遍历边再次遍历到父亲节点.

但无论是哪一种情况,由于一个点也可以作为一个强连通分量,因此我们可以用栈维护dfs过程中遍历到的点.

Tarjan

如果一个点的dfn与自己的$lowlink$相等,那么只有两种情况:

1. 不存在强连通分量,$lowlink$没有由子节点更新,就只有自己的dfn最小.

2. 存在强连通分量,因为子节点可以通过遍历边再次遍历到父亲节点.

但无论是哪一种情况,由于一个点也可以作为一个强连通分量,因此我们可以用栈维护dfs过程中遍历到的点.

只要出现dfn与$lowlink$相同的情况,即代表着栈中的所有元素在同一个强连通分量中,我们只需将每一个元素记录好,并弹栈.

Tarjan

如果一个点的dfn与自己的$lowlink$相等,那么只有两种情况:

1. 不存在强连通分量,$lowlink$没有由子节点更新,就只有自己的dfn最小.

2. 存在强连通分量,因为子节点可以通过遍历边再次遍历到父亲节点.

但无论是哪一种情况,由于一个点也可以作为一个强连通分量,因此我们可以用栈维护dfs过程中遍历到的点.

只要出现dfn与$lowlink$相同的情况,即代表着栈中的所有元素在同一个强连通分量中,我们只需将每一个元素记录好,并弹栈.

时间复杂度同样也是优秀的$O(n+m)$

[HAOI2006]受欢迎的牛

每头奶牛都梦想成为牛棚里的明星.

[HAOI2006]受欢迎的牛

每头奶牛都梦想成为牛棚里的明星.

被所有奶牛喜欢的奶牛就是一头明星奶牛.

[HAOI2006]受欢迎的牛

每头奶牛都梦想成为牛棚里的明星.

被所有奶牛喜欢的奶牛就是一头明星奶牛.

所有奶牛都是自恋狂,每头奶牛总是喜欢自己.

[HAOI2006]受欢迎的牛

每头奶牛都梦想成为牛棚里的明星.

被所有奶牛喜欢的奶牛就是一头明星奶牛.

所有奶牛都是自恋狂,每头奶牛总是喜欢自己.

奶牛之间的“喜欢”是可以传递的.

[HAOI2006]受欢迎的牛

每头奶牛都梦想成为牛棚里的明星.

被所有奶牛喜欢的奶牛就是一头明星奶牛.

所有奶牛都是自恋狂,每头奶牛总是喜欢自己.

奶牛之间的“喜欢”是可以传递的.

如果A喜欢B,B喜欢C,那么A也喜欢C.

[HAOI2006]受欢迎的牛

每头奶牛都梦想成为牛棚里的明星.

被所有奶牛喜欢的奶牛就是一头明星奶牛.

所有奶牛都是自恋狂,每头奶牛总是喜欢自己.

奶牛之间的“喜欢”是可以传递的.

如果A喜欢B,B喜欢C,那么A也喜欢C.

牛栏里共有N头奶牛,给定一些奶牛之间的爱慕关系,请你算出有多少头奶牛可以当明星.

Solution

这题一看就是一个S*题.

Solution

这题一看就是一个S*题.

没什么好讲的吧,只是让那些从来没有接触过缩点的同学了解一下缩点是干什么的.

Solution

这题一看就是一个S*题.

没什么好讲的吧,只是让那些从来没有接触过缩点的同学了解一下缩点是干什么的.

题目实质就是问你这张图缩完点后,出度为0的强连通分量是否有且只有一个,如果满足条件就输出该强连通分量的大小.

Solution

这题一看就是一个S*题.

没什么好讲的吧,只是让那些从来没有接触过缩点的同学了解一下缩点是干什么的.

题目实质就是问你这张图缩完点后,出度为0的强连通分量是否有且只有一个,如果满足条件就输出该强连通分量的大小.

至于更深层次的题目,就让下午讲课的$ylsoi$来讲吧! ~~23333~~

各种定义

Bcc——Biconnected Component.即双连通分量.

各种定义

Bcc——Biconnected Component.即双连通分量.

Pbc——Point Biconnected Component.若一个无向图中的去掉任意一个**节点**都不会改变此图的连通性,即无割点,则称作点双连通图.一个无向图中的每一个极大点双连通子图称作此无向图的**点双连通分量**,简称点双.

各种定义

Bcc——Biconnected Component.即双连通分量.

Pbc——Point Biconnected Component.若一个无向图中的去掉任意一个**节点**都不会改变此图的连通性,即无割点,则称作点双连通图.一个无向图中的每一个极大点双连通子图称作此无向图的**点双连通分量**,简称点双.

Ebc——Edge Biconnected Component.若一个无向图中的去掉任意一条**边**都不会改变此图的连通性,则称作边双连通图.一个无向图中的每一个极大边双连通子图称作此无向图的**边双连通分量**,简称边双.

各种定义

割点——若在一个无向图中去掉一个点,使得图不连通,则称这个点为割点. (PS:一个割点可能属于多个点双连通分量)

各种定义

割点——若在一个无向图中去掉一个点,使得图不连通,则称这个点为割点. (PS:一个割点可能属于多个点双连通分量)

桥——连接两个边双连通分量的边即桥.

各种定义

割点——若在一个无向图中去掉一个点,使得图不连通,则称这个点为割点. (PS:一个割点可能属于多个点双连通分量)

桥——连接两个边双连通分量的边即桥.

以上几个概念都是无向图中的概念,可以与有向图的Scc中的概念进行对比,发现两者在某种程度上是类似的.

Tarjan

无向图中的缩点,一般就是指的缩点双或边双.

Tarjan

无向图中的缩点,一般就是指的缩点双或边双.

主流算法基本只有一种,就是神仙般的Tarjan发明的又一个神仙算法$Tarjan$.

Tarjan

无向图中的缩点,一般就是指的缩点双或边双.

主流算法基本只有一种,就是神仙般的Tarjan发明的又一个神仙算法$Tarjan$.

具体流程和有向图中的$Tarjan$差不多,思想也是类似的.实在不会的可以去网上搜一搜,在此就不再多提.

Tarjan

无向图中的缩点,一般就是指的缩点双或边双.

主流算法基本只有一种,就是神仙般的Tarjan发明的又一个神仙算法$Tarjan$.

具体流程和有向图中的$Tarjan$差不多,思想也是类似的.实在不会的可以去网上搜一搜,在此就不再多提.

(PS:其实对于一般的缩点双,我们可以写圆方树来代替,代码量似乎短一些,而且用处更大,但联赛涉及不到,所以还是不详细讲了).

[POJ3352]Road Construction

给一个无向连通图,至少添加几条边使得去掉图中任意一条边不改变图的连通性(即使它变为边双连通图).

Solutions

这题思路比较简单.

Solutions

这题思路比较简单.

先用$Tarjan$进行缩点,缩点之后一定是一个树.

Solutions

这题思路比较简单.

先用$Tarjan$进行缩点,缩点之后一定是一个树.

考虑度数为1的节点,对于这些节点显然是不满足题目要求的.

Solutions

这题思路比较简单.

先用$Tarjan$进行缩点,缩点之后一定是一个树.

考虑度数为1的节点,对于这些节点显然是不满足题目要求的.

设度数为1的节点的个数为x,那么我们至少要连$\frac{x+1}{2}$边才能使得这些节点度数大于等于2.

Solutions

这题思路比较简单.

先用$Tarjan$进行缩点,缩点之后一定是一个树.

考虑度数为1的节点,对于这些节点显然是不满足题目要求的.

设度数为1的节点的个数为x,那么我们至少要连$\frac{x+1}{2}$边才能使得这些节点度数大于等于2.

实际上,这样所构成的图,就已经是边双连通图.

拓扑排序

前面提到缩点之后有意想不到的收获,

拓扑排序

前面提到缩点之后有意想不到的收获,

其实主要原因是,把图变成有向无环图之后,可以不用对同一个点重复的遍历来计算贡献.

拓扑排序

前面提到缩点之后有意想不到的收获,

其实主要原因是,把图变成有向无环图之后,可以不用对同一个点重复的遍历来计算贡献.

也就是说,我们可以对缩点之后的图,进行**拓扑**DP.

拓扑排序

前面提到缩点之后有意想不到的收获,

其实主要原因是,把图变成有向无环图之后,可以不用对同一个点重复的遍历来计算贡献.

也就是说,我们可以对缩点之后的图,进行**拓扑**DP.

这也是一般缩点后的常见套路.

拓扑排序

前面提到缩点之后有意想不到的收获,

其实主要原因是,把图变成有向无环图之后,可以不用对同一个点重复的遍历来计算贡献.

也就是说,我们可以对缩点之后的图,进行**拓扑**DP.

这也是一般缩点后的常见套路.

拓扑排序应该大家都知道吧. 那就不细讲了,直接放一道题.

[POI2006]PRO-Professor Szu

n个别墅以及一个主建筑楼,从每个别墅都有很多种不同方式走到主建筑楼,其中不同的定义是(每条边可以走多次,如果走边的顺序有一条不同即称两方式不同). (PS:有向图)

[POI2006]PRO-Professor Szu

n个别墅以及一个主建筑楼,从每个别墅都有很多种不同方式走到主建筑楼,其中不同的定义是(每条边可以走多次,如果走边的顺序有一条不同即称两方式不同). (PS:有向图)

询问最多的不同方式是多少,以及有多少个别墅有这么多方式,按照顺序输出别墅编号.

[POI2006]PRO-Professor Szu

n个别墅以及一个主建筑楼,从每个别墅都有很多种不同方式走到主建筑楼,其中不同的定义是(每条边可以走多次,如果走边的顺序有一条不同即称两方式不同). (PS:有向图)

询问最多的不同方式是多少,以及有多少个别墅有这么多方式,按照顺序输出别墅编号.

如果最多不同方式超过了36500那么都视作zawsze.

Solution

这题是不是也是非常的水呀!

Solution

这题是不是也是非常的水呀!

首先,n个点到1个点的方案不好算,于是我们转换为1个点到所有点的方案数.

Solution

这题是不是也是非常的水呀!

首先,n个点到1个点的方案不好算,于是我们转换为1个点到所有点的方案数.

很显然,答案不会发生变化.

Solution

这题是不是也是非常的水呀!

首先,n个点到1个点的方案不好算,于是我们转换为1个点到所有点的方案数.

很显然,答案不会发生变化.

然后我们可以发现,只要出现了任何一个环(即Scc),那么所有经过这个环的点就可以在这个环内无限次的走,因此方案数也是无限大的.

Solution

这题是不是也是非常的水呀!

首先,n个点到1个点的方案不好算,于是我们转换为1个点到所有点的方案数.

很显然,答案不会发生变化.

然后我们可以发现,只要出现了任何一个环(即Scc),那么所有经过这个环的点就可以在这个环内无限次的走,因此方案数也是无限大的.

因此,我们先缩点,并将所有环内的点特判.之后就进行拓扑DP,统计方案数.

小总结

拓扑DP一般来说,非常好写.

小总结

拓扑DP一般来说,非常好写.

由于没有环,因此需要考虑的东西会少一点.

小总结

拓扑DP一般来说,非常好写.

由于没有环,因此需要考虑的东西会少一点.

我曾经一度以为联赛不会考缩点+拓扑DP.

小总结

拓扑DP一般来说,非常好写.

由于没有环,因此需要考虑的东西会少一点.

我曾经一度以为联赛不会考缩点+拓扑DP.

然而.....

小总结

拓扑DP一般来说,非常好写.

由于没有环,因此需要考虑的东西会少一点.

我曾经一度以为联赛不会考缩点+拓扑DP.

然而..... 详情见去年$Day1T3$逛公园.

小总结

拓扑DP一般来说,非常好写.

由于没有环,因此需要考虑的东西会少一点.

我曾经一度以为联赛不会考缩点+拓扑DP.

然而..... 详情见去年$Day1T3$逛公园.

~~(虽说好像一发记忆化搜索可以弄过去...)~~

最短路

最短路应该每一个人都会吧.

最短路

最短路应该每一个人都会吧.

常见算法一般来说有三个.

最短路

最短路应该每一个人都会吧.

常见算法一般来说有三个.

- *Floyd*

最短路

最短路应该每一个人都会吧.

常见算法一般来说有三个.

- *Floyd*
- *Dijkstra*

最短路

最短路应该每一个人都会吧.

常见算法一般来说有三个.

- *Floyd*
- *Dijkstra*
- **SPFA**

Floyd

Floyd:时间复杂度$O(n^3)$,空间复杂度$O(n^2)$.

Floyd

Floyd:时间复杂度$O(n^3)$,空间复杂度$O(n^2)$.

*Floyd*是著名的多源最短路.

Floyd

Floyd:时间复杂度$O(n^3)$,空间复杂度$O(n^2)$.

*Floyd*是著名的多源最短路.

看上去时间复杂度与空间复杂度非常的不好,但是在某些特定的题目中有奇效.

Floyd

$Floyd$:时间复杂度$O(n^3)$,空间复杂度$O(n^2)$.

$Floyd$是著名的多源最短路.

看上去时间复杂度与空间复杂度非常的不好,但是在某些特定的题目中有奇效.

比如说:$NOIP2016Day1T3$换教室. 一个十分秀的期望DP与$Floyd$最短路的好题.

Dijkstra

$Dijkstra$:时间复杂度$O(nlogn)$,空间复杂度$O(m+n)$.

Dijkstra

Dijkstra:时间复杂度$O(nlogn)$,空间复杂度$O(m+n)$.

*Dijkstra*是求单源最短路.

Dijkstra

$Dijkstra$:时间复杂度$O(nlogn)$,空间复杂度$O(m+n)$.

$Dijkstra$是求单源最短路.

朴素的$Dijkstra$是$O(n^2)$,很显然,我们一般都会打堆优化后的$Dijkstra$.

Dijkstra

$Dijkstra$:时间复杂度$O(nlogn)$,空间复杂度$O(m+n)$.

$Dijkstra$是求单源最短路.

朴素的$Dijkstra$是$O(n^2)$,很显然,我们一般都会打堆优化后的$Dijkstra$.

它的优点在于复杂度十分的稳定,基于贪心的思想.

Dijkstra

$Dijkstra$:时间复杂度$O(nlogn)$,空间复杂度$O(m+n)$.

$Dijkstra$是求单源最短路.

朴素的$Dijkstra$是$O(n^2)$,很显然,我们一般都会打堆优化后的$Dijkstra$.

它的优点在于复杂度十分的稳定,基于贪心的思想.

应用显然也非常广泛.

SPFA

不知道你们有没有注意到.

SPFA

不知道你们有没有注意到.

所有有关**SPFA**的地方,我都加粗了.....

SPFA

不知道你们有没有注意到.

所有有关**SPFA**的地方,我都加粗了.....

因为.....

SPFA

不知道你们有没有注意到.

所有有关**SPFA**的地方,我都加粗了.....

因为.....

关于**SPFA**,它死了...

SPFA

不知道你们有没有注意到.

所有有关**SPFA**的地方,我都加粗了.....

因为.....

关于**SPFA**,它死了...

也不好说联赛如果有最短路,出题人会不会卡**SPFA**,不过最好还是打$Dijkstra$.

SPFA

不知道你们有没有注意到.

所有有关**SPFA**的地方,我都加粗了.....

因为.....

关于**SPFA**,它死了...

也不好说联赛如果有最短路,出题人会不会卡**SPFA**,不过最好还是打$Dijkstra$.

当然,**SPFA**也是有它的独特作用的.

SPFA

不知道你们有没有注意到.

所有有关**SPFA**的地方,我都加粗了.....

因为.....

关于**SPFA**,它死了...

也不好说联赛如果有最短路,出题人会不会卡**SPFA**,不过最好还是打$Dijkstra$.

当然,**SPFA**也是有它的独特作用的.

无论是哪一种费用流,都必须要用**SPFA**去跑最小费用.

顺带一提

关于最短路,

顺带一提

关于最短路,

除了需要知道最短路的基本算法,也需要了解最短路的计数.

顺带一提

关于最短路,

除了需要知道最短路的基本算法,也需要了解最短路的计数.

而最短路最难的地方在于它的建模,

差分约束系统

最短路有一种经典的应用就是——差分约束系统.

差分约束系统

最短路有一种经典的应用就是——差分约束系统.

比如有一组不等式:

差分约束系统

最短路有一种经典的应用就是——差分约束系统.

比如有一组不等式:

$x_1 - x_2 \leq s_1$

$x_3 - x_1 \geq s_2$

$x_2 - x_3 = s_3$

差分约束系统

最短路有一种经典的应用就是——差分约束系统.

比如有一组不等式:

$x_1 - x_2 \leq s_1$

$x_3 - x_1 \geq s_2$

$x_2 - x_3 = s_3$

其中,s_1, s_2, s_3为常数.

差分约束系统

最短路有一种经典的应用就是——差分约束系统.

比如有一组不等式:

$x_1 - x_2 \leq s_1$

$x_3 - x_1 \geq s_2$

$x_2 - x_3 = s_3$

其中,s_1, s_2, s_3为常数.

这些不等式,全都是两个未知数的差与某个常数的不等$(\leq, \geq, =)$关系.这样的不等式组就称作差分约束系统.

差分约束与最短路

看到这样的两个未知数作差,有没有产生一种神奇的想法...

差分约束与最短路

看到这样的两个未知数作差,有没有产生一种神奇的想法...

我们考虑最短路中的松弛操作, $dis[u] \leq dis[v] + w[i]$

差分约束与最短路

看到这样的两个未知数作差,有没有产生一种神奇的想法...

我们考虑最短路中的松弛操作, $dis[u] \leq dis[v] + w[i]$

我们将右边移项即:$dis[u] - dis[v] \leq w[i]$

差分约束与最短路

看到这样的两个未知数作差,有没有产生一种神奇的想法...

我们考虑最短路中的松弛操作, $dis[u] \leq dis[v] + w[i]$

我们将右边移项即:$dis[u] - dis[v] \leq w[i]$

因此我们就可以根据这个性质,建图跑最短路求解.

差分约束与最短路

看到这样的两个未知数作差,有没有产生一种神奇的想法...

我们考虑最短路中的松弛操作, $dis[u] \leq dis[v] + w[i]$

我们将右边移项即:$dis[u] - dis[v] \leq w[i]$

因此我们就可以根据这个性质,建图跑最短路求解.

当然具体操作还有一些细节,比如不等式的符号要化成相同的,$=$要拆成$\leq, \geq$之类的.

差分约束与最短路

看到这样的两个未知数作差,有没有产生一种神奇的想法...

我们考虑最短路中的松弛操作, $dis[u] \leq dis[v] + w[i]$

我们将右边移项即:$dis[u] - dis[v] \leq w[i]$

因此我们就可以根据这个性质,建图跑最短路求解.

当然具体操作还有一些细节,比如不等式的符号要化成相同的,=要拆成$\leq, \geq$之类的.

可以去看一下这个人的博客;
https://blog.csdn.net/liuzhushiqiang/article/details/9345881 .

平面图

平面图在图论中是一种十分常见的图.

平面图

平面图在图论中是一种十分常见的图.

所有可以画在平面上并且使不同边互不重叠的图都是平面图.

平面图

平面图在图论中是一种十分常见的图.

所有可以画在平面上并且使不同边互不重叠的图都是平面图.

常见的平面图是网格图.

平面图

平面图在图论中是一种十分常见的图.

所有可以画在平面上并且使不同边互不重叠的图都是平面图.

常见的平面图是网格图.

常见的非平面图是五个顶点的完全图,以及六个顶点的完全二分图.

平面图

平面图在图论中是一种十分常见的图.

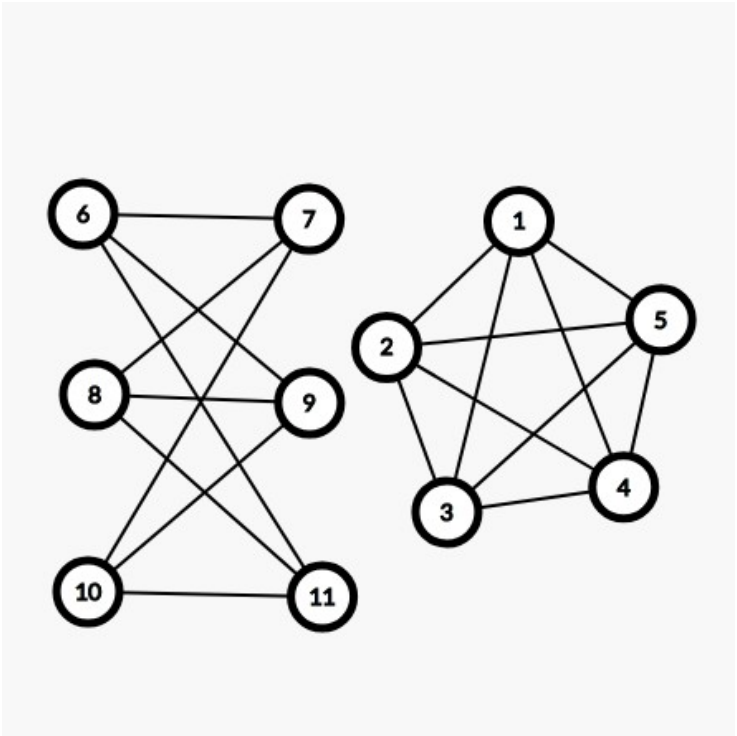
所有可以画在平面上并且使不同边互不重叠的图都是平面图.

常见的平面图是网格图.

常见的非平面图是五个顶点的完全图,以及六个顶点的完全二分图.

有一条定理:一个图是平面图当且仅当它不包含一个是以上两种非平面图的的分割的子图. (一个图A是另一个图B的分割是指:A是在B的基础上,在某些边的中间加上顶点)

平面图



平面图与对偶图

对偶图是与平面图相伴的图.

平面图与对偶图

对偶图是与平面图相伴的图.

把平面图中被包围的区域与最外层区域看成点,原图每一条边所属的两个相邻的区域对应在对偶图中的点相连就是对偶图.

平面图与对偶图

对偶图是与平面图相伴的图.

把平面图中被包围的区域与最外层区域看成点,原图每一条边所属的两个相邻的区域对应在对偶图中的点相连就是对偶图.

当某些问题在平面图中不好解决的时候,将平面图转换为对偶图是一种十分好的想法.

平面图与对偶图

对偶图是与平面图相伴的图.

把平面图中被包围的区域与最外层区域看成点,原图每一条边所属的两个相邻的区域对应在对偶图中的点相连就是对偶图.

当某些问题在平面图中不好解决的时候,将平面图转换为对偶图是一种十分好的想法.

然而一般的平面图转对偶图是有些麻烦的,需要用到最左转线法.而网格图中是较为容易转换的.因此这种思想最常见的应用就是在网格图上.

平面图与对偶图

对偶图是与平面图相伴的图.

把平面图中被包围的区域与最外层区域看成点,原图每一条边所属的两个相邻的区域对应在对偶图中的点相连就是对偶图.

当某些问题在平面图中不好解决的时候,将平面图转换为对偶图是一种十分好的想法.

然而一般的平面图转对偶图是有些麻烦的,需要用到最左转线法.而网格图中是较为容易转换的.因此这种思想最常见的应用就是在网格图上.

当然还有一个著名的结论,平面图上的最大流等于其对偶图上的最短路.也就是说只要一个图是平面图,那么我们最大流的复杂度就可以降低为$O(nlogn)$,当然,在某些情况下是不一定会跑得过优化后的$Dinic$

欧拉路径

欧拉路径其实是整个图论的起始,关于图论问题的探究最早就是欧拉对七桥问题的研究.

欧拉路径

欧拉路径其实是整个图论的起始,关于图论问题的探究最早就是欧拉对七桥问题的研究.

欧拉路径——如果图G中的一条路径包括每个边恰好一次,则该路径称为欧拉路径.即可以不重复的遍历完整张图.

欧拉路径

欧拉路径其实是整个图论的起始,关于图论问题的探究最早就是欧拉对七桥问题的研究.

欧拉路径——如果图G中的一条路径包括每个边恰好一次,则该路径称为欧拉路径.即可以不重复的遍历完整张图.

欧拉回路——在欧拉路径的前提下,如果从某一个点出发,由同一个点结束,那么则称这条欧拉路径为欧拉回路.

欧拉路径

欧拉路径其实是整个图论的起始,关于图论问题的探究最早就是欧拉对七桥问题的研究.

欧拉路径——如果图G中的一条路径包括每个边恰好一次,则该路径称为欧拉路径.即可以不重复的遍历完整张图.

欧拉回路——在欧拉路径的前提下,如果从某一个点出发,由同一个点结束,那么则称这条欧拉路径为欧拉回路.

简单来说,欧拉路径就是指的我们平日所说的一笔画问题,欧拉回路是其中的一种特殊情况.

欧拉路径

欧拉路径其实是整个图论的起始,关于图论问题的探究最早就是欧拉对七桥问题的研究.

欧拉路径——如果图G中的一条路径包括每个边恰好一次,则该路径称为欧拉路径.即可以不重复的遍历完整张图.

欧拉回路——在欧拉路径的前提下,如果从某一个点出发,由同一个点结束,那么则称这条欧拉路径为欧拉回路.

简单来说,欧拉路径就是指的我们平日所说的一笔画问题,欧拉回路是其中的一种特殊情况.

当然,如果图不连通,那就什么都不存在了.

无向图的欧拉路径

无向图中判断欧拉回路十分简单,只需要每个点的度数均为偶数.

无向图的欧拉路径

无向图中判断欧拉回路十分简单,只需要每个点的度数均为偶数.

考虑每一个点的度数都是偶数,即可以看成整个图由一个又一个环组成,因此,一定存在欧拉回路.

无向图的欧拉路径

无向图中判断欧拉回路十分简单,只需要每个点的度数均为偶数.

考虑每一个点的度数都是偶数,即可以看成整个图由一个又一个环组成,因此,一定存在欧拉回路.

欧拉路径的判定稍微宽松一定,允许存在两个点的度数为奇度数.可以知道,这条欧拉路径的起点与终点一定是这两个点.

无向图的欧拉路径

无向图中判断欧拉回路十分简单,只需要每个点的度数均为偶数.

考虑每一个点的度数都是偶数,即可以看成整个图由一个又一个环组成,因此,一定存在欧拉回路.

欧拉路径的判定稍微宽松一定,允许存在两个点的度数为奇度数.可以知道,这条欧拉路径的起点与终点一定是这两个点.

至于,找欧拉路径的代码也比较简单.

无向图的欧拉路径

无向图中判断欧拉回路十分简单,只需要每个点的度数均为偶数.

考虑每一个点的度数都是偶数,即可以看成整个图由一个又一个环组成,因此,一定存在欧拉回路.

欧拉路径的判定稍微宽松一定,允许存在两个点的度数为奇度数.可以知道,这条欧拉路径的起点与终点一定是这两个点.

至于,找欧拉路径的代码也比较简单.

网上似乎把名字说的特别神奇,其实就是在dfs的过程中进行删边,保证不重复走同一条边,同时再用栈记录一下dfs序,反过来就是一条欧拉路径.

有向图的欧拉路径

有向图中的欧拉回路与无向图中略有不同. 它要求每一个点的入度等于出度.

有向图的欧拉路径

有向图中的欧拉回路与无向图中略有不同. 它要求每一个点的入度等于出度.

考虑如果入度和出度相等,那么每进入一个点就一定能从这个点出来.因此,一定存在欧拉回路.

有向图的欧拉路径

有向图中的欧拉回路与无向图中略有不同. 它要求每一个点的入度等于出度.

考虑如果入度和出度相等,那么每进入一个点就一定能从这个点出来.因此,一定存在欧拉回路.

欧拉路径:最多有一点入度等于出度+1,最多有一点入度等于出度−1,就会有一条从出度大于入度(没有则等于)的点出发,到达出度小于入度(没有则等于)的点的一条欧拉路径.

有向图的欧拉路径

有向图中的欧拉回路与无向图中略有不同. 它要求每一个点的入度等于出度.

考虑如果入度和出度相等,那么每进入一个点就一定能从这个点出来.因此,一定存在欧拉回路.

欧拉路径:最多有一点入度等于出度+1,最多有一点入度等于出度−1,就会有一条从出度大于入度(没有则等于)的点出发,到达出度小于入度(没有则等于)的点的一条欧拉路径.

这个证明的方法与无向图的类似.

有向图的欧拉路径

有向图中的欧拉回路与无向图中略有不同. 它要求每一个点的入度等于出度.

考虑如果入度和出度相等,那么每进入一个点就一定能从这个点出来.因此,一定存在欧拉回路.

欧拉路径:最多有一点入度等于出度+1,最多有一点入度等于出度−1,就会有一条从出度大于入度(没有则等于)的点出发,到达出度小于入度(没有则等于)的点的一条欧拉路径.

这个证明的方法与无向图的类似.

当然有向图中找欧拉路径的代码也是十分简单,就是把无向图中的删边操作改为删单向边就行了.

混合图的欧拉路径

混合图指的是有向边与无向边同时存在的图.

混合图的欧拉路径

混合图指的是有向边与无向边同时存在的图.

混合图上的欧拉路径比较难找,而且联赛也不常考. ~~根本没考过...~~

混合图的欧拉路径

混合图指的是有向边与无向边同时存在的图.

混合图上的欧拉路径比较难找,而且联赛也不常考. ~~根本没考过...~~

但是,我还是大概说一下做法.

混合图的欧拉路径

混合图指的是有向边与无向边同时存在的图.

混合图上的欧拉路径比较难找,而且联赛也不常考. ~~根本没考过...~~

但是,我还是大概说一下做法.

先给所有的无向边随机定向. 定向后,这就变成一个有向图.而有向图中的欧拉路径的判断方法我们是知道的.因此,我们根据每个点的入度与出度进行判断并求出路径. 然而这时是不一定可以形成欧拉路径的,因此我们需要调整一开始的随机定向.至于这个调整的过程,可以用网络流实现.

混合图的欧拉路径

混合图指的是有向边与无向边同时存在的图.

混合图上的欧拉路径比较难找,而且联赛也不常考. ~~根本没考过...~~

但是,我还是大概说一下做法.

先给所有的无向边随机定向. 定向后,这就变成一个有向图.而有向图中的欧拉路径的判断方法我们是知道的.因此,我们根据每个点的入度与出度进行判断并求出路径. 然而这时是不一定可以形成欧拉路径的,因此我们需要调整一开始的随机定向.至于这个调整的过程,可以用网络流实现.

这个算法现在没什么大用,一时半会也讲不清楚.还是看别人的$blog$吧.
https://blog.csdn.net/pi9nc/article/details/12223693 .

[BZOJ 3724] Final Krolestwo

你有一个无向连通图,边的总数为偶数.

[BZOJ 3724] Final Krolestwo

你有一个无向连通图,边的总数为偶数.

设图中有k个奇点(度数为奇数的点),你需要把它们配成$\frac{k}{2}$个点对.对于每个点对(u,v),你需要用一条长度为偶数(假设每条边长度为1)的路径将u和v连接.

[BZOJ 3724] Final Krolestwo

你有一个无向连通图,边的总数为偶数.

设图中有k个奇点(度数为奇数的点),你需要把它们配成$\frac{k}{2}$个点对.对于每个点对(u,v),你需要用一条长度为偶数(假设每条边长度为1)的路径将u和v连接.

每条路径允许经过重复的点,但不允许经过重复的边.这$\frac{k}{2}$条路径之间也不能有重复的边.

[BZOJ 3724] Final Krolestwo

你有一个无向连通图,边的总数为偶数.

设图中有k个奇点(度数为奇数的点),你需要把它们配成$\frac{k}{2}$个点对.对于每个点对(u,v),你需要用一条长度为偶数(假设每条边长度为1)的路径将u和v连接.

每条路径允许经过重复的点,但不允许经过重复的边.这$\frac{k}{2}$条路径之间也不能有重复的边.

$2 \leq n, m \leq 250000$

Solution

由于这题太神仙了,讲课人太菜了,应该不能很好的讲清楚这一道题(OVO).

Solution

由于这题太神仙了,讲课人太菜了,应该不能很好的讲清楚这一道题(OVO).

因此就让机房的图论大佬$oyiya$来讲一下这道题.

Solution

由于这题太神仙了,讲课人太菜了,应该不能很好的讲清楚这一道题(OVO).

因此就让机房的图论大佬$oyiya$来讲一下这道题.

顺便帮他推一下blog.
https://www.cnblogs.com/hongyj/p/9484241.html

哈密顿回路

哈密顿回路指的是图中存在一条回路只经过每一个结点一次.

哈密顿回路

哈密顿回路指的是图中存在一条回路只经过每一个结点一次.

与欧拉路径不同的是,寻找哈密顿回路是一个$NP-complete$问题.

哈密顿回路

哈密顿回路指的是图中存在一条回路只经过每一个结点一次.

与欧拉路径不同的是,寻找哈密顿回路是一个$NP-complete$问题.

因此我们不能在多项式时间内找到解,

哈密顿回路

哈密顿回路指的是图中存在一条回路只经过每一个结点一次.

与欧拉路径不同的是,寻找哈密顿回路是一个$NP-complete$问题.

因此我们不能在多项式时间内找到解,

甚至判定一个图是否存在哈密顿回路,也没有普遍的判定方法,都是判定是否满足哈密顿回路的各种充分条件.

哈密顿回路

哈密顿回路指的是图中存在一条回路只经过每一个结点一次.

与欧拉路径不同的是,寻找哈密顿回路是一个$NP-complete$问题.

因此我们不能在多项式时间内找到解,

甚至判定一个图是否存在哈密顿回路,也没有普遍的判定方法,都是判定是否满足哈密顿回路的各种充分条件.

联赛显然也不会考判定,因此只是稍微提一下...

哈密顿回路

哈密顿回路指的是图中存在一条回路只经过每一个结点一次.

与欧拉路径不同的是,寻找哈密顿回路是一个$NP-complete$问题.

因此我们不能在多项式时间内找到解,

甚至判定一个图是否存在哈密顿回路,也没有普遍的判定方法,都是判定是否满足哈密顿回路的各种充分条件.

联赛显然也不会考判定,因此只是稍微提一下...

但是,与哈密顿回路有关的题目还是很多的.不过由于哈密顿路的特殊性,这些题目多半都和哈密顿路径没有**任何**关系

[HDU 3435] A new Graph Game

有n个点和m条边,你可以删去任意条边,使得所有点在一个哈密顿路径上,并且路径的权值最小.

[HDU 3435] A new Graph Game

有n个点和m条边,你可以删去任意条边,使得所有点在一个哈密顿路径上,并且路径的权值最小.

$1 \leq n \leq 1000, 0 \leq m \leq 10000$

[HDU 3435] A new Graph Game

有n个点和m条边,你可以删去任意条边,使得所有点在一个哈密顿路径上,并且路径的权值最小.

$1 \leq n \leq 1000, 0 \leq m \leq 10000$

有多组数据$T \leq 20$

Solution

正如前面所说,这道题和哈密顿路径没有**任何**的关系.

Solution

正如前面所说,这道题和哈密顿路径没有**任何**的关系.

考虑到哈密顿路径的定义,问题的实质就是求有向图的最小环覆盖.

Solution

正如前面所说,这道题和哈密顿路径没有**任何**的关系.

考虑到哈密顿路径的定义,问题的实质就是求有向图的最小环覆盖.

那么这就是一个十分经典的二分图建模套路了.

Solution

正如前面所说,这道题和哈密顿路径没有**任何**的关系.

考虑到哈密顿路径的定义,问题的实质就是求有向图的最小环覆盖.

那么这就是一个十分经典的二分图建模套路了.

将一个点拆成入点与出点.并按照原图的连接关系在二分图上连边.即若原图中$u->v$,则二分图中将u的出点与v的入点相连.

Solution

正如前面所说,这道题和哈密顿路径没有**任何**的关系.

考虑到哈密顿路径的定义,问题的实质就是求有向图的最小环覆盖.

那么这就是一个十分经典的二分图建模套路了.

将一个点拆成入点与出点.并按照原图的连接关系在二分图上连边.即若原图中$u->v$,则二分图中将u的出点与v的入点相连.

很显然,二分图中的完美匹配就是有向图中的一个环覆盖.

Solution

正如前面所说,这道题和哈密顿路径没有**任何**的关系.

考虑到哈密顿路径的定义,问题的实质就是求有向图的最小环覆盖.

那么这就是一个十分经典的二分图建模套路了.

将一个点拆成入点与出点.并按照原图的连接关系在二分图上连边.即若原图中$u->v$,则二分图中将u的出点与v的入点相连.

很显然,二分图中的完美匹配就是有向图中的一个环覆盖.

也就是说我们只需要做一个带权二分图匹配.显然KM跑不过,因此我们跑费用流就行了.

生成树

生成树是一个应用较广的算法.似乎联赛中较难的图论题都与生成树有关.

生成树

生成树是一个应用较广的算法.似乎联赛中较难的图论题都与生成树有关.

对于生成树,主要算法是最小(大)生成树,生成树计数以及最小(大)生成树上的性质.

生成树

生成树是一个应用较广的算法.似乎联赛中较难的图论题都与生成树有关.

对于生成树,主要算法是最小(大)生成树,生成树计数以及最小(大)生成树上的性质.

首先还是来讲最小(大)生成树.

最小生成树

最小生成树的算法主要有两个:$Prim$和$Kruskal$

最小生成树

最小生成树的算法主要有两个:$Prim$和$Kruskal$

$Prim$与$Dijkstra$的打法一样,复杂度也是一样的.似乎它们只是理解方式不一样.

最小生成树

最小生成树的算法主要有两个:$Prim$和$Kruskal$

$Prim$与$Dijkstra$的打法一样,复杂度也是一样的.似乎它们只是理解方式不一样.

$Kruskal$一般来说更常见.~~毕竟好打233.~~

最小生成树

最小生成树的算法主要有两个:$Prim$和$Kruskal$

$Prim$与$Dijkstra$的打法一样,复杂度也是一样的.似乎它们只是理解方式不一样.

$Kruskal$一般来说更常见.~~毕竟好打233.~~

然而在稠密图中,还是多使用$Prim$.否则,$Kruskal$中的快排可能就会花费大量时间.

性质

最小生成树相关的性质有许多.

性质

最小生成树相关的性质有许多.

- 原图中相同权值的边对生成树的贡献相同

性质

最小生成树相关的性质有许多.

- 原图中相同权值的边对生成树的贡献相同
- 图上两个点之间路径上的最小(大)权值的最大(小)的边一定在最大(小)生成树上.

性质

最小生成树相关的性质有许多.

- 原图中相同权值的边对生成树的贡献相同
- 图上两个点之间路径上的最小(大)权值的最大(小)的边一定在最大(小)生成树上.
- 不同的生成树中,某一种权值的边连接完成后,形成的联通块状态是一样的.

性质

最小生成树相关的性质有许多.

- 原图中相同权值的边对生成树的贡献相同
- 图上两个点之间路径上的最小(大)权值的最大(小)的边一定在最大(小)生成树上.
- 不同的生成树中,某一种权值的边连接完成后,形成的联通块状态是一样的.
- 任何一棵非最小生成树的生成树一定可以通过换一次边的方法得到权值更小的生成树.

性质

最小生成树相关的性质有许多.

- 原图中相同权值的边对生成树的贡献相同
- 图上两个点之间路径上的最小(大)权值的最大(小)的边一定在最大(小)生成树上.
- 不同的生成树中,某一种权值的边连接完成后,形成的联通块状态是一样的.
- 任何一棵非最小生成树的生成树一定可以通过换一次边的方法得到权值更小的生成树.

而其中运用最多的性质是第二点,而最重要的是第一点.第三点可以用来计数,第四点是用LCT维护动态最小(大)生成树的依据.

性质

第一个性质的意思即一个图的最小(大)生成树可能会有很多,但是无论是哪一种方案,最小生成树上存在的每一种权值的数量不变.

性质

第一个性质的意思即一个图的最小(大)生成树可能会有很多,但是无论是哪一种方案,最小生成树上存在的每一种权值的数量不变.

第二个性质最好的体现就是$NOIP2013$货车运输.

性质

第一个性质的意思即一个图的最小(大)生成树可能会有很多,但是无论是哪一种方案,最小生成树上存在的每一种权值的数量不变.

第二个性质最好的体现就是$NOIP2013$货车运输.

这些性质都很显然.

性质

第一个性质的意思即一个图的最小(大)生成树可能会有很多,但是无论是哪一种方案,最小生成树上存在的每一种权值的数量不变.

第二个性质最好的体现就是$NOIP2013$货车运输.

这些性质都很显然.

关键就是在做题的时候是否想得到.

Matrix Tree

解决生成树计数最主要的算法就是矩阵树定理($Matrix\ Tree$).

Matrix Tree

解决生成树计数最主要的算法就是矩阵树定理($Matrix\ Tree$).

这个定理是基于基尔霍夫矩阵实现的.

Matrix Tree

解决生成树计数最主要的算法就是矩阵树定理($Matrix\ Tree$).

这个定理是基于基尔霍夫矩阵实现的.

也就是说要记录好一个图的度数矩阵以及邻接矩阵,并将两者相减.

Matrix Tree

解决生成树计数最主要的算法就是矩阵树定理($Matrix\ Tree$).

这个定理是基于基尔霍夫矩阵实现的.

也就是说要记录好一个图的度数矩阵以及邻接矩阵,并将两者相减.

接着在算出基尔霍夫矩阵的行列式的值,这个值就是该图中生成树的个数.

Matrix Tree

解决生成树计数最主要的算法就是矩阵树定理($Matrix\ Tree$).

这个定理是基于基尔霍夫矩阵实现的.

也就是说要记录好一个图的度数矩阵以及邻接矩阵,并将两者相减.

接着在算出基尔霍夫矩阵的行列式的值,这个值就是该图中生成树的个数.

(凭借矩阵树定理与性质三可以算出最小生成树的个数...)

[BZOJ 2753] 滑雪与时间胶囊

这里分布着M条供滑行的轨道和N个轨道之间的景点,而且每个景点都有一编号$i(1 \leq i \leq N)$和一高度H_i.小A能从景点i滑到景点j当且仅当存在一条i和j之间的边,且i的高度不小于j.

[BZOJ 2753] 滑雪与时间胶囊

这里分布着M条供滑行的轨道和N个轨道之间的景点,而且每个景点都有一编号$i(1 \leq i \leq N)$和一高度H_i.小A能从景点i滑到景点j当且仅当存在一条i和j之间的边,且i的高度不小于j.

小A喜欢用最短的滑行路径去访问尽量多的景点.如果仅仅访问一条路径上的景点,他会觉得数量太少.于是小A拿出了他随身携带的时间胶囊.吃下之后可以立即回到上个经过的景点.这种神奇的药物是可以连续食用的,即能够回到较长时间之前到过的景点.小A站在1号景点望着山,想知道在不考虑时间胶囊消耗的情况下,以最短滑行距离滑到尽量多的景点的方案.

[BZOJ 2753] 滑雪与时间胶囊

这里分布着M条供滑行的轨道和N个轨道之间的景点,而且每个景点都有一编号$i(1 \leq i \leq N)$和一高度H_i.小A能从景点i滑到景点j当且仅当存在一条i和j之间的边,且i的高度不小于j.

小A喜欢用最短的滑行路径去访问尽量多的景点.如果仅仅访问一条路径上的景点,他会觉得数量太少.于是小A拿出了他随身携带的时间胶囊.吃下之后可以立即回到上个经过的景点.这种神奇的药物是可以连续食用的,即能够回到较长时间之前到过的景点.小A站在1号景点望着山,想知道在不考虑时间胶囊消耗的情况下,以最短滑行距离滑到尽量多的景点的方案.

$1 \leq M \leq 1000000, 1 \leq H_i \leq 1000000000, 1 \leq K_i \leq 1000000000$

Solutions

这题的题目看起来很长,仔细分析一下,发现题目要求的就是最小树形图.

Solutions

这题的题目看起来很长,仔细分析一下,发现题目要求的就是最小树形图.

第一问就是求1能够达到的所有点数,直接dfs搜一下就行了.

Solutions

这题的题目看起来很长,仔细分析一下,发现题目要求的就是最小树形图.

第一问就是求1能够达到的所有点数,直接dfs搜一下就行了.

这题的关键是第二问.如何求有向图的最小生成树.

Solutions

这题的题目看起来很长,仔细分析一下,发现题目要求的就是最小树形图.

第一问就是求1能够达到的所有点数,直接dfs搜一下就行了.

这题的关键是第二问.如何求有向图的最小生成树.

若是直接上朱刘算法,复杂度是错误的.(当然若是你会Tarjan的$O(m+nlogn)$最小树形图算法就没什么好说的了.)

Solutions

这题的题目看起来很长,仔细分析一下,发现题目要求的就是最小树形图.

第一问就是求1能够达到的所有点数,直接dfs搜一下就行了.

这题的关键是第二问.如何求有向图的最小生成树.

若是直接上朱刘算法,复杂度是错误的.(当然若是你会Tarjan的$O(m+nlogn)$最小树形图算法就没什么好说的了.)

因此我们考虑正常的$kruskal$为什么不支持有向图.

Solutions

因为$kruskal$贪心的每次选择边权最小的边,由于存在反向边,最后不一定保证根节点可以到达每一个子节点.

Solutions

因为$kruskal$贪心的每次选择边权最小的边,由于存在反向边,最后不一定保证根节点可以到达每一个子节点.

本题中反向边的存在当且仅当两点间的高度相同,而当存在高度差的时候,我们优先选择较高的点,那么较高的点一定先访问,这样就一定不会存在反向边.

Solutions

因为$kruskal$贪心的每次选择边权最小的边,由于存在反向边,最后不一定保证根节点可以到达每一个子节点.

本题中反向边的存在当且仅当两点间的高度相同,而当存在高度差的时候,我们优先选择较高的点,那么较高的点一定先访问,这样就一定不会存在反向边.

对于高度相同的边,我们贪心的选择边权较小的,这样就一定不会存在上面的情况.

Solutions

因为$kruskal$贪心的每次选择边权最小的边,由于存在反向边,最后不一定保证根节点可以到达每一个子节点.

本题中反向边的存在当且仅当两点间的高度相同,而当存在高度差的时候,我们优先选择较高的点,那么较高的点一定先访问,这样就一定不会存在反向边.

对于高度相同的边,我们贪心的选择边权较小的,这样就一定不会存在上面的情况.

复杂度$O(mlogm)$.

总结

其实图论的知识非常的多并且很杂，

总结

其实图论的知识非常的多并且很杂,

我和$ylsoi$如果都是题目与算法一起讲是很难整理全的.

总结

其实图论的知识非常的多并且很杂,

我和$ylsoi$如果都是题目与算法一起讲是很难整理全的.

因此,我上午就讲所有联赛中涉及到的图论算法讲完.

总结

其实图论的知识非常的多并且很杂,

我和$ylsoi$如果都是题目与算法一起讲是很难整理全的.

因此,我上午就讲所有联赛中涉及到的图论算法讲完.

下午就由$ylsoi$讲各种我看不懂,不会做的神仙图论题.

Thanks!