图论

吴清月

2022 年 2 月 9 日

定义

定义

欧拉路径：如果图中的一个路径包括每个边恰好一次，则该路径称为欧拉路径 (Euler path)。

欧拉回路：首尾相接的欧拉路径称为欧拉回路。

判定

判定

由于每一条边都要经过恰好一次，因此对于除了起点和终点之外的任意一个节点，只要进来，一定要出去。

判定

由于每一条边都要经过恰好一次，因此对于除了起点和终点之外的任意一个节点，只要进来，一定要出去。

一个无向图存在欧拉回路，当且仅当该图所有顶点度数都为偶数，且该图只有一个存在边的连通块。

一个无向图存在欧拉路径，当且仅当该图中奇点的数量为 0 或 2，且该图只有一个存在边的连通块。

判定

由于每一条边都要经过恰好一次，因此对于除了起点和终点之外的任意一个节点，只要进来，一定要出去。

一个无向图存在欧拉回路，当且仅当该图所有顶点度数都为偶数，且该图只有一个存在边的连通块。

一个无向图存在欧拉路径，当且仅当该图中奇点的数量为 0 或 2，且该图只有一个存在边的连通块。

一个有向图存在欧拉回路，当且仅当所有点的入度等于出度。

一个混合图存在欧拉回路，当且仅当存在一个对所有无向边定向的方案，使得所有点的入度等于出度。需要用网络流。

求法

求法

我们用 dfs 来求出一张图的欧拉回路。

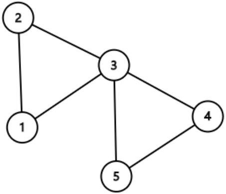
我们给每一条边一个 vis 数组代表是否访问过，接下来从一个点出发，遍历所有的边。

求法

我们用 dfs 来求出一张图的欧拉回路。

我们给每一条边一个 vis 数组代表是否访问过，接下来从一个点出发，遍历所有的边。

直接 dfs 并且记录的话会有一些问题，比如下面这张图：



从 1 号点出发进行 dfs，1-2-3-1，无路可走，剩下一个 3-4-5-3 的环没地方放。

求法

求法

为了解决这个问题，我们在记录答案的时候倒着记录，也就是当我们通过 (u, v) 这条边到达 v 的时候，先把 v dfs 完再加入 (v, u) 这条边。

求法

为了解决这个问题，我们在记录答案的时候倒着记录，也就是当我们通过 (u, v) 这条边到达 v 的时候，先把 v dfs 完再加入 (v, u) 这条边。

还是举上面这个例子，比如当前到达了 3 号点：

- 若先 dfs $(3, 1)$ 这条边：先加入 $(1, 3)$，再加入 $(3, 5), (5, 4), (4, 3)$，最后加入 $(3, 2)$。
- 若先 dfs $(3, 4)$ 这条边：dfs 出了 3-4-5-3-1 这条路径，直接加入 $(1, 3), (3, 5), (5, 4), (4, 3)$，最后加入 $(3, 2)$。

求法

为了解决这个问题，我们在记录答案的时候倒着记录，也就是当我们通过 (u, v) 这条边到达 v 的时候，先把 v dfs 完再加入 (v, u) 这条边。

还是举上面这个例子，比如当前到达了 3 号点：

- 若先 dfs $(3, 1)$ 这条边：先加入 $(1, 3)$，再加入 $(3, 5), (5, 4), (4, 3)$，最后加入 $(3, 2)$。
- 若先 dfs $(3, 4)$ 这条边：dfs 出了 3-4-5-3-1 这条路径，直接加入 $(1, 3), (3, 5), (5, 4), (4, 3)$，最后加入 $(3, 2)$。

还有一点需要注意。因为一个点可能被访问多次，一不小心可能会写成 $O(n^2)$ 的（因为每次遍历所有的出边）。解决方案就是设一个 cur 数组，每次直接从上一次访问到的出边继续遍历。

时间复杂度 $O(n + m)$。

代码

代码

```
void dfs(int x)
{
    for(int&hd=head[x];hd;hd=e[hd].nxt)
    {
        if(flag[hd>>1])continue;
        flag[hd>>1]=1;
        dfs(e[hd].to);
        a[++top]=x;
    }
}
```

洛谷 P1341 无序字母对

洛谷 P1341 无序字母对

给定 n 个互不相同的无序字母对，区分大小写，你需要构造一个长度为 $n+1$ 的字符串，使得这些字母对都出现过。

洛谷 P1341 无序字母对

洛谷 P1341 无序字母对

每一个字母作为图上的一个点。

对于一个无序字母对，在对应的两个字母之间连边。

洛谷 P1341 无序字母对

每一个字母作为图上的一个点。
对于一个无序字母对，在对应的两个字母之间连边。
然后跑一个欧拉路径。

CodeForces 547D

CodeForces 547D

给定平面上的 n 个点的坐标，你需要将每一个点进行红蓝染色，满足任意一行一列红蓝点的数量之差不超过 2。

$1 \leq n \leq 2 \times 10^5$

CodeForces 547D

CodeForces 547D

欧拉回路有一个很关键的性质：所有节点的入度等于出度。

CodeForces 547D

欧拉回路有一个很关键的性质：所有节点的入度等于出度。

首先将点的坐标进行离散化。

把行列抽象成节点，对于一个点 (x, y)，我们就从 x 到 y 连一条边。

连完之后可能会出现奇点，所有奇点向 0 连一条边。

CodeForces 547D

欧拉回路有一个很关键的性质：所有节点的入度等于出度。

首先将点的坐标进行离散化。

把行列抽象成节点，对于一个点 (x, y)，我们就从 x 到 y 连一条边。

连完之后可能会出现奇点，所有奇点向 0 连一条边。

然后对整张图跑欧拉回路，顺次交替染为红蓝两色即可。

时间复杂度 $O(n)$。

某道题

某道题

构造一个长度为 2^n 的 01 串，这个串收尾相接，你需要保证所有长度为 n 的 01 串都出现过。

$n \leq 18$

某道题

某道题

可以想到把每一个不同的 01 串看成一个节点，从一个串添加一个字符（0 或 1）到达另一个串看做一条边。

某道题

可以想到把每一个不同的 01 串看成一个节点，从一个串添加一个字符（0 或 1）到达另一个串看做一条边。

然后跑欧拉回路?

由于一共有 2^{n+1} 条边，我们构造出了一个长度为 2^{n+1} 的 01 串，其中每一个长度为 n 的串都出现了两次，一次后面跟着 0，一次后面跟着 1。

某道题

可以想到把每一个不同的 01 串看成一个节点，从一个串添加一个字符（0 或 1）到达另一个串看做一条边。

然后跑欧拉回路?

由于一共有 2^{n+1} 条边，我们构造出了一个长度为 2^{n+1} 的 01 串，其中每一个长度为 n 的串都出现了两次，一次后面跟着 0，一次后面跟着 1。

换句话说每一个长度为 $n+1$ 的串都出现了一次!

只需要把每一个长度为 $n-1$ 的字符串看做一个节点即可。

定义

定义

所谓拓扑排序，就是把有向图上的 n 个点重新标号为 1 到 n，满足对于任意一条边 (u, v)，都有 $u < v$。

并不是所有的图都能进行拓扑排序，只要图中有环，那么就可以导出矛盾。

定义

所谓拓扑排序，就是把有向图上的 n 个点重新标号为 1 到 n，满足对于任意一条边 (u, v)，都有 $u < v$。

并不是所有的图都能进行拓扑排序，只要图中有环，那么就可以导出矛盾。

可以进行拓扑排序的图称为有向无环图（DAG），有很多优美的性质，比如可以在拓扑序上进行 DP。

拓扑排序

拓扑排序

我们记录一下每一个点的入度和出度，用一个队列维护当前所有入度为 0 的点。

每次拿出来一个入度为 0 的点并且将它加到拓扑序中，然后枚举出边更新度数。

时间复杂度 $O(n+m)$。

拓扑排序

我们记录一下每一个点的入度和出度，用一个队列维护当前所有入度为 0 的点。

每次拿出来一个入度为 0 的点并且将它加到拓扑序中，然后枚举出边更新度数。

时间复杂度 $O(n+m)$。

在拓扑排序的过程中可以顺带进行 DP。

代码

代码

```
for(int i=1;i<=n;i++)
  if(d[i]==0)q.push(i);
while(!q.empty())
{
    int node=q.front();q.pop();res[++top]=node;
    for(int hd=head[node];hd;hd=e[hd].nxt)
    {
        d[e[hd].to]--;
        if(d[e[hd].to]==0)q.push(e[hd].to);
    }
}
```

出烂了的例题

出烂了的例题

给定一张 DAG，求最长链。边带权/不带权。

出烂了的例题

出烂了的例题

首先进行拓扑排序。

出烂了的例题

首先进行拓扑排序。

设 f_i 表示以 i 结尾的最长链，则有：

$$f_v = \max(f_u + w(u, v))$$

出烂了的例题

首先进行拓扑排序。

设 f_i 表示以 i 结尾的最长链，则有：

$$f_v = \max(f_u + w(u, v))$$

直接 DP 即可。

定义

定义

所谓最短路，就是把边权看做边的长度，从某个点 S 到另一个点 T 的最短路径。~~（这不是废话吗）~~

用更加数学化的语言描述就是，对于映射 $f\colon V \to \mathbb{R}$，满足 $f(S) = 0$ 且 $\forall (x, y, l) \in E, |f(x) - f(y)| \leq l$ 的情况下，$f(T)$ 的**最大值**。

单源最短路——Dijkstra

单源最短路——Dijkstra

在所有的边权均为正的情况下，我们可以使用 Dijkstra 算法求出一个点到所有其它点的最短路径。

单源最短路——Dijkstra

在所有的边权均为正的情况下，我们可以使用 Dijkstra 算法求出一个点到所有其它点的最短路径。

我们维护一个集合，表示这个集合内的点最短路径已经确定了。

单源最短路——Dijkstra

在所有的边权均为正的情况下，我们可以使用 Dijkstra 算法求出一个点到所有其它点的最短路径。

我们维护一个集合，表示这个集合内的点最短路径已经确定了。

每次我们从剩下的点中选择当前距离最小的点 u 加入这个集合，然后枚举另一个点 v 进行更新：

$$d_v = \min(d_v, d_u + w(u, v))$$

单源最短路——Dijkstra

在所有的边权均为正的情况下，我们可以使用 Dijkstra 算法求出一个点到所有其它点的最短路径。

我们维护一个集合，表示这个集合内的点最短路径已经确定了。

每次我们从剩下的点中选择当前距离最小的点 u 加入这个集合，然后枚举另一个点 v 进行更新：

$$d_v = \min(d_v, d_u + w(u, v))$$

直接这样做时间复杂度是 $O(n^2)$ 的。

堆优化

堆优化

我们注意到，复杂度主要来源于两个地方。

堆优化

我们注意到，复杂度主要来源于两个地方。

第一个是找出当前距离最小的点。这个可以用堆很容易地实现。

第二个是枚举 v，如果我们用邻接表存图，可以降到边数级别。

堆优化

我们注意到，复杂度主要来源于两个地方。

第一个是找出当前距离最小的点。这个可以用堆很容易地实现。

第二个是枚举 v，如果我们用邻接表存图，可以降到边数级别。

这样我们就把复杂度降到了 $O((n+m)\log n)$。

单源最短路——Bellman-Ford

单源最短路——Bellman-Ford

另一种求单源最短路的算法，复杂度不如 Dijkstra 优秀。

单源最短路——Bellman-Ford

另一种求单源最短路的算法，复杂度不如 Dijkstra 优秀。
考虑在上面出现过的松弛操作：

$$d_v = \min(d_v, d_u + w(u, v))$$

单源最短路——Bellman-Ford

另一种求单源最短路的算法，复杂度不如 Dijkstra 优秀。

考虑在上面出现过的松弛操作：

$$d_v = \min(d_v, d_u + w(u, v))$$

由于最短路径只会经过最多 n 个点，因此每一个点的最短路径只会被松弛至多 $n-1$ 次。

所以我们可以对整张图进行 $n-1$ 次松弛操作，每次枚举所有的边进行更新。

单源最短路——Bellman-Ford

另一种求单源最短路的算法，复杂度不如 Dijkstra 优秀。

考虑在上面出现过的松弛操作：

$$d_v = \min(d_v, d_u + w(u, v))$$

由于最短路径只会经过最多 n 个点，因此每一个点的最短路径只会被松弛至多 $n-1$ 次。

所以我们可以对整张图进行 $n-1$ 次松弛操作，每次枚举所有的边进行更新。

时间复杂度 $O(nm)$。

关于 SPFA

关于 SPFA

它死了。

关于 SPFA

它死了。

不过核心思路还是讲一讲吧，毕竟后面学费用流还要用。

关于 SPFA

它死了。

不过核心思路还是讲一讲吧，毕竟后面学费用流还要用。

Bellman-Ford 算法不够优秀，于是我们尝试改进这个算法。

注意到，在进行松弛操作的时候，如果点 u 的距离一直没有发生变化，那么就不需要再枚举这个点的出边进行松弛了。

关于 SPFA

它死了。

不过核心思路还是讲一讲吧，毕竟后面学费用流还要用。

Bellman-Ford 算法不够优秀，于是我们尝试改进这个算法。

注意到，在进行松弛操作的时候，如果点 u 的距离一直没有发生变化，那么就不需要再枚举这个点的出边进行松弛了。

也就是说我们可以用一个队列保存所有距离发生变化的点，每次取出一个点进行更新。

写出来代码一测，你别说还挺快的！

关于 SPFA

它死了。

不过核心思路还是讲一讲吧，毕竟后面学费用流还要用。

Bellman-Ford 算法不够优秀，于是我们尝试改进这个算法。

注意到，在进行松弛操作的时候，如果点 u 的距离一直没有发生变化，那么就不需要再枚举这个点的出边进行松弛了。

也就是说我们可以用一个队列保存所有距离发生变化的点，每次取出一个点进行更新。

写出来代码一测，你别说还挺快的！

于是 SPFA(Shortest Path Faster Algorithm) 就诞生了。

关于 SPFA

关于 SPFA

如果图是随机的，SPFA 的期望时间复杂度约为 $O(2m)$，比之前提到的任何一个算法都优秀，而且还可以有负权。

关于 SPFA

如果图是随机的，SPFA 的期望时间复杂度约为 $O(2m)$，比之前提到的任何一个算法都优秀，而且还可以有负权。

但是在最坏情况下它的复杂度和 Bellman-Ford 相同，都是 $O(nm)$，在正式比赛中，没有哪个出题人会放过它。（因为本来复杂度就是错的）

多源最短路——Floyd

多源最短路——Floyd

对于一张图，我们希望求出任意两个点之间的最短路径。

我们用 DP 的思想。设 $f_{i,j,k}$ 表示从 i 到 j，途中仅经过前 k 个点的最短路。

多源最短路——Floyd

对于一张图，我们希望求出任意两个点之间的最短路径。

我们用 DP 的思想。设 $f_{i,j,k}$ 表示从 i 到 j，途中仅经过前 k 个点的最短路。

由于每一个点在最短路中只会出现一次（不然就出现负环了，不存在最短路），所以可以很轻松地写出转移方程：

$$f_{i,j,k} = \min(f_{i,j,k-1}, f_{i,k,k-1} + f_{k,j,k-1})$$

时间复杂度是 $O(n^3)$。

多源最短路——Floyd

对于一张图，我们希望求出任意两个点之间的最短路径。

我们用 DP 的思想。设 $f_{i,j,k}$ 表示从 i 到 j，途中仅经过前 k 个点的最短路。

由于每一个点在最短路中只会出现一次（不然就出现负环了，不存在最短路），所以可以很轻松地写出转移方程：

$$f_{i,j,k} = \min(f_{i,j,k-1}, f_{i,k,k-1} + f_{k,j,k-1})$$

时间复杂度是 $O(n^3)$。

在实际求的过程中，最后一维可以用滚动数组优化掉，所以空间复杂度是 $O(n^2)$。

代码

代码

```
for(int k=1;k<=n;k++)
  for(int i=1;i<=n;i++)
    for(int j=1;j<=n;j++)
      dis[i][j]=min(dis[i][j],dis[i][k]+dis[k][j]);
```

代码

```
for(int k=1;k<=n;k++)
  for(int i=1;i<=n;i++)
    for(int j=1;j<=n;j++)
      dis[i][j]=min(dis[i][j],dis[i][k]+dis[k][j]);
```

注意三层循环的顺序不能颠倒。

Floyd 传递闭包

Floyd 传递闭包

有时候，我们需要维护一些有传递性的关系，比如相等，连通等等。(12 连通，23 连通，则 13 连通)

初始条件往往是已知若干个点对具有这些关系，然后让你弄出来所有的关系。

Floyd 传递闭包

有时候，我们需要维护一些有传递性的关系，比如相等，连通等等。(12 连通，23 连通，则 13 连通)

初始条件往往是已知若干个点对具有这些关系，然后让你弄出来所有的关系。

可以直接把 Floyd 算法做一下调整——

```
dis[i][j]=dis[i][j]|(dis[i][k]&dis[k][j]);
```

这个算法叫做传递闭包。

多源最短路——Johnson 重赋权

多源最短路——Johnson 重赋权

对于多源最短路，如果我们枚举一个点然后跑堆优化的 Dijkstra，那么复杂度是 $O(nm\log n)$ 的，在图比较稀疏的情况下，这个复杂度要优于 Floyd 算法的 $O(n^3)$。

多源最短路——Johnson 重赋权

对于多源最短路，如果我们枚举一个点然后跑堆优化的 Dijkstra，那么复杂度是 $O(nm\log n)$ 的，在图比较稀疏的情况下，这个复杂度要优于 Floyd 算法的 $O(n^3)$。

但是 Dijkstra 算法要求所有边权均非负。

于是就有了重赋权的技巧。

多源最短路——Johnson 重赋权

对于多源最短路，如果我们枚举一个点然后跑堆优化的 Dijkstra，那么复杂度是 $O(nm\log n)$ 的，在图比较稀疏的情况下，这个复杂度要优于 Floyd 算法的 $O(n^3)$。

但是 Dijkstra 算法要求所有边权均非负。

于是就有了重赋权的技巧。

我们新建一个 0 号点，并且从这个点出发向所有点连一条边权为 0 的边，然后跑单源最短路。(SPFA 或者 Bellman-Ford)

设距离数组为 h，接下来对于每条边 (u, v)，令 $w'(u, v) = w(u, v) + h(u) - h(v)$。

多源最短路——Johnson 重赋权

对于多源最短路，如果我们枚举一个点然后跑堆优化的 Dijkstra，那么复杂度是 $O(nm\log n)$ 的，在图比较稀疏的情况下，这个复杂度要优于 Floyd 算法的 $O(n^3)$。

但是 Dijkstra 算法要求所有边权均非负。

于是就有了重赋权的技巧。

我们新建一个 0 号点，并且从这个点出发向所有点连一条边权为 0 的边，然后跑单源最短路。(SPFA 或者 Bellman-Ford)

设距离数组为 h，接下来对于每条边 (u, v)，令 $w'(u, v) = w(u, v) + h(u) - h(v)$。

这样所有的边权就都变成非负了，我们就可以跑 Dijkstra 算法了。

证明

证明

首先由于 $h(v) \leq h(u) + w(u, v)$，新图的边权一定非负。

证明

首先由于 $h(v) \leq h(u) + w(u, v)$，新图的边权一定非负。
设新图上的最短路径为 d'，原图上的最短路径为 d。

证明

首先由于 $h(v) \le h(u) + w(u,v)$，新图的边权一定非负。
设新图上的最短路径为 d'，原图上的最短路径为 d。

$$
\begin{aligned}
d'(u,v) &= \min_{a_1,a_2,\dots,a_k} w'(u,a_1) + w'(a_1,a_2) + \cdots + w'(a_k,v) \\
&= \min_{a_1,a_2,\dots,a_k} w(u,a_1) + (h(u) - h(a_1)) + w(a_1,a_2) + \\
&\quad (h(a_2) - h(a_1)) + \cdots + w(a_k,v) + (h(v) - h(a_k)) \\
&= h(u) - h(v) + \min_{a_1,a_2,\dots,a_k} w(u,a_1) + \cdots + w(a_k,v) \\
&= h(u) - h(v) + d(u,v)
\end{aligned}
$$

最短路树（最短路图）

最短路树（最短路图）

所谓最短路树，就是在求完从 S 出发的单源最短路之后，只保留最短路上的边形成的数据结构。

最短路树（最短路图）

所谓最短路树，就是在求完从 S 出发的单源最短路之后，只保留最短路上的边形成的数据结构。

只需要在求的过程中维护一个 pre 数组表示这个点的前驱即可。

最短路树（最短路图）

所谓最短路树，就是在求完从 S 出发的单源最短路之后，只保留最短路上的边形成的数据结构。

只需要在求的过程中维护一个 pre 数组表示这个点的前驱即可。

很多最短路的变种都需要用这个算法。

【JLOI2011】飞行路线

【JLOI2011】飞行路线

给定一张 n 个点 m 条边的带权无向图，你可以把至多 k 条边边权变成 0，求从 s 到 t 的最短路。

$2 \le n \le 10^4, 1 \le m \le 5 \times 10^4, 0 \le k \le 10$

【JLOI2011】飞行路线

【JLOI2011】飞行路线

经典模型：分层图最短路。

设 (i,j) 表示从 s 到达 i，途中把 j 条边变成 0 的情况。把一个点拆成 k 个点。

【JLOI2011】飞行路线

经典模型：分层图最短路。

设 (i, j) 表示从 s 到达 i，途中把 j 条边变成 0 的情况。把一个点拆成 k 个点。

对于一条边 (u, v)：

- 从 (u, k) 到 (v, k) 连边权为 $w(u, v)$ 的边。
- 从 (u, k) 到 $(v, k+1)$ 连边权为 0 的边。

【JLOI2011】飞行路线

经典模型：分层图最短路。

设 (i,j) 表示从 s 到达 i，途中把 j 条边变成 0 的情况。把一个点拆成 k 个点。

对于一条边 (u,v)：

- 从 (u,k) 到 (v,k) 连边权为 $w(u,v)$ 的边。
- 从 (u,k) 到 $(v,k+1)$ 连边权为 0 的边。

最后求一遍从 $(s,0)$ 出发的单源最短路，答案即为 $\min_{i\le k} d(t,i)$。

时间复杂度 $O(k(n+m)\log m)$。

洛谷 P2761 软件补丁问题

洛谷 P2761 软件补丁问题

有一款软件有 n 个 BUG，还有 m 个补丁。

对于每一个补丁，都有两个 BUG 集合 B_1, B_2，表示只有当这个软件包含了 B_1 中的所有 BUG 而不包含 B_2 中的任意一个 BUG 的时候，这个补丁才能使用。

对于每一个补丁，它会修复一个 BUG 集合 F_1，同时引入另一个 BUG 集合 F_2，运行需要 t 的时间。

求将所有 BUG 都修复需要的最短时间，无法修复输出 -1。

$1 \le n \le 20, 1 \le m \le 100$

洛谷 P2701 软件补丁问题

洛谷 P2701 软件补丁问题

每一个 BUG 集合对应一个点，一个补丁对应一条边。

对于每一条补丁，枚举所有可以使用的集合，然后看会转移到哪里。

洛谷 P2701 软件补丁问题

每一个 BUG 集合对应一个点，一个补丁对应一条边。

对于每一条补丁，枚举所有可以使用的集合，然后看会转移到哪里。

最后求一遍最短路。

时间复杂度有点紧，需要亿点点信仰。

【NOIP2017】逛公园

【NOIP2017】逛公园

给你一张 n 个点 m 条边的图，问你从 1 到 n，与最短路的差不超过 k 的路径有多少条。

可能有 0 边，如果数量无限输出 -1。

$n \leq 100000, m \leq 200000, k \leq 50$

【NOIP2017】逛公园

【NOIP2017】逛公园

首先跑一边最短路。

【NOIP2017】逛公园

首先跑一边最短路。

一看 k 这么小，我们就设 $f_{i,j}$ 表示从 1 到 i，与最短路的差等于 j 的有多少条。第二维就到 50。

然后先枚举 j，内层用最短路的步骤进行转移。

【NOIP2017】逛公园

首先跑一边最短路。

一看 k 这么小，我们就设 $f_{i,j}$ 表示从 1 到 i，与最短路的差等于 j 的有多少条。第二维就到 50。

然后先枚举 j，内层用最短路的步骤进行转移。

注意特判 -1 的情况，出现 -1 需要满足存在 0 环并且存在经过这个 0 环的满足条件的路径。洛谷数据很水，建议交到 UOJ 上测试。

CodeForces 1163F

CodeForces 1163F

给你一张 n 个点 m 条边的图，有 q 次询问，每次问你如果更改一条边的边权，从 1 到 n 的最短路是多少。

询问之间相互独立。

$n, m, q \leq 2 \times 10^5$

CodeForces 1163F

CodeForces 1163F

首先建出来两棵最短路树，分别以 1 和 n 为根。

CodeForces 1163F

首先建出来两棵最短路树，分别以 1 和 n 为根。

接下来对修改分情况：

CodeForces 1163F

首先建出来两棵最短路树，分别以 1 和 n 为根。

接下来对修改分情况：

1. 在最短路上，改小了：答案即为最短路。

CodeForces 1163F

首先建出来两棵最短路树，分别以 1 和 n 为根。
接下来对修改分情况：

1. 在最短路上，改小了：答案即为最短路。
2. 不在最短路上，改大了：无影响。

CodeForces 1163F

首先建出来两棵最短路树，分别以 1 和 n 为根。

接下来对修改分情况：

1. 在最短路上，改小了：答案即为最短路。
2. 不在最短路上，改大了：无影响。
3. 不在最短路上，改小了：新的最短路要么不变要么经过被修改的边，设修改了 (u, v)，则直接用 $d_1(u) + w(u, v) + d_2(v)$ 和最短路取 min 即可。

CodeForces 1163F

首先建出来两棵最短路树，分别以 1 和 n 为根。

接下来对修改分情况：

1. 在最短路上，改小了：答案即为最短路。
2. 不在最短路上，改大了：无影响。
3. 不在最短路上，改小了：新的最短路要么不变要么经过被修改的边，设修改了 (u, v)，则直接用 $d_1(u) + w(u, v) + d_2(v)$ 和最短路取 min 即可。
4. 在最短路上，改大了：新的最短路有可能绕过被修改的边。

CodeForces 1163F

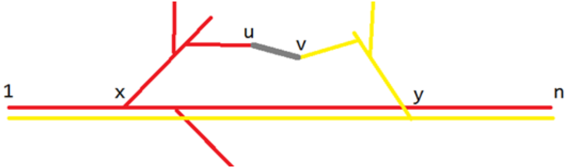
CodeForces 1163F

下面着重讨论第四种情况。

CodeForces 1163F

下面着重讨论第四种情况。

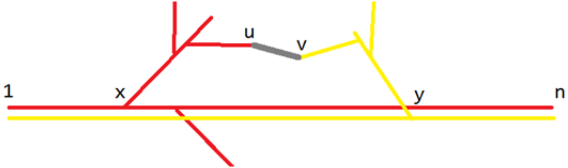
考虑修改之后的最短路，答案一定是在第一棵最短路树上从 1 到达一个点 u，然后走一条 (u, v) 到达另一个点 v，再在第二棵最短路树上从 v 到达 n，途中绕过被修改的边。



CodeForces 1163F

下面着重讨论第四种情况。

考虑修改之后的最短路，答案一定是在第一棵最短路树上从 1 到达一个点 u，然后走一条 (u, v) 到达另一个点 v，再在第二棵最短路树上从 v 到达 n，途中绕过被修改的边。



也就是说，我们可以考虑枚举合法的 (u, v)，用 $d_1(u) + w(u, v) + d_2(v)$ 更新答案。

这个算法是 $O(mq)$ 的。

CodeForces 1163F

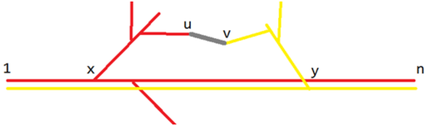
CodeForces 1163F

考虑优化这个算法。

CodeForces 1163F

考虑优化这个算法。

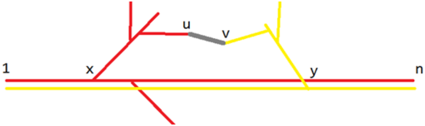
我们可以预先枚举一个 (u, v)，然后看哪些边被更改的时候我们可以用 (u, v) 去更新。



CodeForces 1163F

考虑优化这个算法。

我们可以预先枚举一个 (u, v)，然后看哪些边被更改的时候我们可以用 (u, v) 去更新。

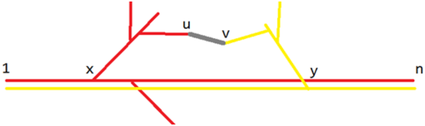


如图所示。容易发现如果修改的边位于 x 和 y 之间，我们就可以通过 (u, v) 来绕过被修改的边。

CodeForces 1163F

考虑优化这个算法。

我们可以预先枚举一个 (u, v)，然后看哪些边被更改的时候我们可以用 (u, v) 去更新。



如图所示。容易发现如果修改的边位于 x 和 y 之间，我们就可以通过 (u, v) 来绕过被修改的边。

在 1 到 n 的最短路上建立一棵线段树，维护删掉每一条边的答案。只需要支持区间取 min 和单点查询。

时间复杂度 $O((m + q) \log n)$。

Prim 算法

Prim 算法

类比 Dijkstra 算法，我们维护一个集合 S，表示这个集合中的生成树已经确定了。

Prim 算法

类比 Dijkstra 算法，我们维护一个集合 S，表示这个集合中的生成树已经确定了。

算法流程和 Dijkstra 一样，唯一的区别是用 $w(u,v)$ 去更新 d_v 而不是用 $d_u + w(u,v)$。

时间复杂度 $O(n^2)$，同样可以用堆优化。

Kruskal 算法

Kruskal 算法

因为是求的最小生成树，所以我们用贪心的思路，把所有的边权从小到大排序，然后一条一条尝试加入，用并查集维护连通性。

Kruskal 算法

因为是求的最小生成树，所以我们用贪心的思路，把所有的边权从小到大排序，然后一条一条尝试加入，用并查集维护连通性。

可以发现这样一定能得到原图的最小生成树，证明如下：

Proof.

如果某一条边 (u, v) 不属于最小生成树，那么考虑最小生成树上连接 u, v 的路径，这上面一定有一条边权不小于 $w(u, v)$ 的边（因为我们是从小到大枚举的所有边），这样替换后答案一定不会变劣。 $\square$

Kruskal 算法

因为是求的最小生成树，所以我们用贪心的思路，把所有的边权从小到大排序，然后一条一条尝试加入，用并查集维护连通性。

可以发现这样一定能得到原图的最小生成树，证明如下：

Proof.

如果某一条边 (u, v) 不属于最小生成树，那么考虑最小生成树上连接 u, v 的路径，这上面一定有一条边权不小于 $w(u, v)$ 的边（因为我们是从小到大枚举的所有边），这样替换后答案一定不会变劣。 □

时间复杂度 $O(m \log m)$。

Kruskal 重构树

Kruskal 重构树

Kruskal 重构树是基于 Kruskal 最小生成树算法的一种算法，它主要通过将边权转化为点权来实现。

Kruskal 重构树

Kruskal 重构树是基于 Kruskal 最小生成树算法的一种算法，它主要通过将边权转化为点权来实现。

这个算法的流程如下：

1. 将所有边按照边权排序，设 $r(x)$ 表示 x 所在连通块的根节点。（注意这里要用并查集）

Kruskal 重构树

Kruskal 重构树是基于 Kruskal 最小生成树算法的一种算法，它主要通过将边权转化为点权来实现。

这个算法的流程如下：

1. 将所有边按照边权排序，设 $r(x)$ 表示 x 所在连通块的根节点。（注意这里要用并查集）
2. 枚举所有的边 (u, v)，若 u, v 不连通，则
   - 新建一个点 x，令 x 的权值为 $w(u, v)$。
   - 连接 $(x, r(u))$ 和 $(x, r(v))$。
   - 令 $r(u) = r(v) = x$。

Kruskal 重构树

Kruskal 重构树是基于 Kruskal 最小生成树算法的一种算法，它主要通过将边权转化为点权来实现。

这个算法的流程如下：

1. 将所有边按照边权排序，设 $r(x)$ 表示 x 所在连通块的根节点。（注意这里要用并查集）
2. 枚举所有的边 (u, v)，若 u, v 不连通，则
   - 新建一个点 x，令 x 的权值为 $w(u, v)$。
   - 连接 $(x, r(u))$ 和 $(x, r(v))$。
   - 令 $r(u) = r(v) = x$。
3. 不断重复以上过程，直到所有点均连通。

时间复杂度 $O(m \log m)$。

性质

性质

这样，我们就得到了一棵有 $2n-1$ 个节点的二叉树，其中叶节点为原图中的点，其余的点代表原图中的边，并且满足父节点权值大于等于子节点。

性质

这样，我们就得到了一棵有 $2n-1$ 个节点的二叉树，其中叶节点为原图中的点，其余的点代表原图中的边，并且满足父节点权值大于等于子节点。

它有什么用呢?

- 求 u, v 之间路径上的最大边权 $\rightarrow$ 求重构树上 u, v 两个点的 LCA。
- 只保留边权小于等于 x 的边形成的树 $\rightarrow$ 重构树上点权小于等于 x 的点的子树。
- $\cdots\cdots$

Borůvka 算法

Borůvka 算法

第三种求最小生成树的算法，虽然比较冷门但是很多题需要用到这个算法。

Borůvka 算法

第三种求最小生成树的算法，虽然比较冷门但是很多题需要用到这个算法。

我们维护当前形成的所有连通块，接下来对于每一个连通块，找到边权最小的出边，然后合并两个连通块。

不断重复这个操作，直到整张图变成一个连通块。

Borůvka 算法

第三种求最小生成树的算法，虽然比较冷门但是很多题需要用到这个算法。

我们维护当前形成的所有连通块，接下来对于每一个连通块，找到边权最小的出边，然后合并两个连通块。

不断重复这个操作，直到整张图变成一个连通块。

由于每次操作连通块数量至少减半，所以时间复杂度最坏为 $O((n+m)\log n)$，随机图的话复杂度可以降到 $O(n+m)$。

【NOIP2013】货车运输

【NOIP2013】货车运输

一张 n 个点 m 条边的图，每一条边有一个限重。

现在有 q 组询问，每次问你从 u 到 v，在不超过限重的情况下，重量最大可以是多少。

$n \leq 10^4, m \leq 5 \times 10^4, q \leq 3 \times 10^4$

【NOIP2013】货车运输

【NOIP2013】货车运输

按照限重求一个最大生成树，接下来每次询问等价于求一条链上的最小值。

用树上倍增即可，时间复杂度 $O(n\log n)$。

【NOIP2013】货车运输

按照限重求一个最大生成树，接下来每次询问等价于求一条链上的最小值。

用树上倍增即可，时间复杂度 $O(n\log n)$。

也可以用 Kruskal 重构树，建出来重构树之后每次询问等价于求 LCA。时间复杂度还是 $O(n\log n)$。

【NOI2018】归程

【NOI2018】归程

n 个点 m 条边的图，每一条边有一个长度 l 和海拔 a。

接下来有 q 天，每一天有一个起始位置 s 和水位线 h。海拔高度不超过 h 的所有边都有积水。你的目标是回到 1 号点。

s 的位置有一辆车，车只能走没有积水的边，你可以在任意节点下车然后步行回到 1 号点。

求每一天你要步行的最短距离。强制在线。

$n \leq 2 \times 10^5, m \leq 4 \times 10^5, q \leq 4 \times 10^5$

多测，组数不超过 3 组。

【NOI2018】归程

【NOI2018】归程

预处理出每一个点到 1 的最短路，把它当做点权，接下来就是求能够到达的所有点中的最小点权。

首先考虑离线算法：将所有的海拔排序，然后从高到低加入，用并查集维护连通性和集合内最小的点权。

【NOI2018】归程

预处理出每一个点到 1 的最短路，把它当做点权，接下来就是求能够到达的所有点中的最小点权。

首先考虑离线算法：将所有的海拔排序，然后从高到低加入，用并查集维护连通性和集合内最小的点权。

但是在线怎么做呢?

1. 可持久化并查集，时间复杂度 $O(n\log^2 n)$。
2. Kruskal 重构树。重构树上每一个点记录一下海拔高度和子树内的最小点权。由于父节点的最小点权一定小于子节点，答案即为 s 的深度最浅的满足海拔高度大于 h 的祖先的点权。时间复杂度 $O(n\log n)$。

某道正睿题

某道正睿题

给你平面上的 n 个点，求最大曼哈顿距离生成树。

$n \leq 100000$

某道正睿题

某道正睿题

首先上一个套路：曼哈顿转切比雪夫。

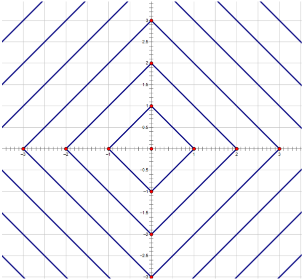
两个点的切比雪夫距离定义为 $\max(|x_1 - x_2|, |y_1 - y_2|)$。

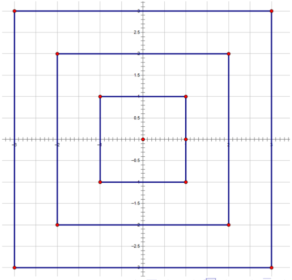
某道正睿题

首先上一个套路：曼哈顿转切比雪夫。

两个点的切比雪夫距离定义为 $\max(|x_1 - x_2|, |y_1 - y_2|)$。

如果我们以一个点 (x, y) 为中心，将到它的距离相等的点连成一条“等距线”，那么曼哈顿距离和切比雪夫距离分别长成下面两个样子：





某道正睿题

某道正睿题

直观地讲，我们将坐标系旋转 45°，就可以实现两个距离之间的互相转化。

对于一个点 (x, y)，我们把它变成 $(x + y, x - y)$ 即可（如果切比雪夫转曼哈顿的话最后还得除以 2）。

某道正睿题

直观地讲，我们将坐标系旋转 $45°$，就可以实现两个距离之间的互相转化。

对于一个点 (x, y)，我们把它变成 $(x+y, x-y)$ 即可（如果切比雪夫转曼哈顿的话最后还得除以 2）。

这样做有什么好处呢?

我们考虑之前提到的 Borůvka 算法。对于每个连通块，我们希望求出离它最远的点。

容易发现，最远的点只有可能是横/纵坐标最小/最大的点，直接记录一下四个方向的最大值和次大值即可。

某道正睿题

直观地讲，我们将坐标系旋转 $45°$，就可以实现两个距离之间的互相转化。

对于一个点 (x, y)，我们把它变成 $(x+y, x-y)$ 即可（如果切比雪夫转曼哈顿的话最后还得除以 2）。

这样做有什么好处呢?

我们考虑之前提到的 Borůvka 算法。对于每个连通块，我们希望求出离它最远的点。

容易发现，最远的点只有可能是横/纵坐标最小/最大的点，直接记录一下四个方向的最大值和次大值即可。

时间复杂度其实是 $O(n)$ 的，因为第一次合并完之后就只剩下最多两个集合了。

扯淡

扯淡

Tarjan 算法不是某个特定的算法，而是一群算法。

扯淡

Tarjan 算法不是某个特定的算法，而是一群算法。

目前已经知道的有：

- 强连通分量
- 割点/割边/桥
- 点双连通分量
- 边双连通分量
- 离线 $O(n)$ 求 LCA

扯淡

Tarjan 算法不是某个特定的算法，而是一群算法。

目前已经知道的有：

- 强连通分量
- 割点/割边/桥
- 点双连通分量
- 边双连通分量
- 离线 $O(n)$ 求 LCA

此外还有很多 Tarjan 独立/合作创造的算法：Splay，LCT，斐波那契堆，斜堆，配对堆，可持久化数据结构，⋯⋯

有向图——强连通分量

有向图——强连通分量

如果对于两个点 u, v，同时存在从 u 到 v 的一条路径和从 v 到 u 的一条路径，那么就称这两个点强连通。

如果一张图的任意两个点均强连通，那么就称这张图为强连通图。

有向图——强连通分量

如果对于两个点 u, v，同时存在从 u 到 v 的一条路径和从 v 到 u 的一条路径，那么就称这两个点强连通。

如果一张图的任意两个点均强连通，那么就称这张图为强连通图。

强连通分量指的是一张有向图的极大强连通子图。

Tarjan 算法可以用来找出一张有向图的所有强连通分量。

有向图——强连通分量

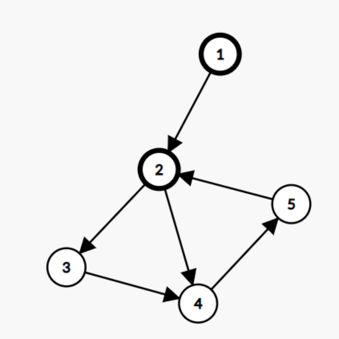
有向图——强连通分量

我们用 dfs 的方式来找出一张图的强连通分量。

有向图——强连通分量

我们用 dfs 的方式来找出一张图的强连通分量。

建出 dfs 树，记录一下每一个节点的时间戳 dfn，然后我们考虑强连通分量应该满足什么条件。



有向图——强连通分量

有向图——强连通分量

2345 四个节点形成了一个强连通分量，在 dfs 树上，从 2 号节点出发无论如何都不能回到 1。

有向图——强连通分量

2345 四个节点形成了一个强连通分量，在 dfs 树上，从 2 号节点出发无论如何都不能回到 1。

我们可以再记录一个 low 数组，表示每一个点能够到达的最小的时间戳，如果一个点的 dfn=low，那么这个点下方就形成了一个强连通分量。

有向图——强连通分量

2345 四个节点形成了一个强连通分量，在 dfs 树上，从 2 号节点出发无论如何都不能回到 1。

我们可以再记录一个 low 数组，表示每一个点能够到达的最小的时间戳，如果一个点的 dfn=low，那么这个点下方就形成了一个强连通分量。

在 dfs 的过程中，对于 (u, v) 这条边：

- 若 v 未被访问，则递归进去 dfs 并且用 low[v] 更新 low[u]。
- 若 v 已经被访问并且在栈中，则直接用 dfn[v] 更新 low[u]。

有向图——强连通分量

2345 四个节点形成了一个强连通分量，在 dfs 树上，从 2 号节点出发无论如何都不能回到 1。

我们可以再记录一个 low 数组，表示每一个点能够到达的最小的时间戳，如果一个点的 dfn=low，那么这个点下方就形成了一个强连通分量。

在 dfs 的过程中，对于 (u, v) 这条边：

- 若 v 未被访问，则递归进去 dfs 并且用 low[v] 更新 low[u]。
- 若 v 已经被访问并且在栈中，则直接用 dfn[v] 更新 low[u]。

最后如果 dfn[u]=low[u]，则直接把栈中一直到 u 的所有点拿出来作为一个强连通分量。

时间复杂度 $O(n)$。

有向图——缩点

有向图——缩点

跑出来强连通分量之后，我们可以把一个强连通分量看成一个点。

接下来枚举所有的边，如果是一个强连通分量里的就忽略，否则连接两个对应的强连通分量。这个操作称为缩点。

有向图——缩点

跑出来强连通分量之后，我们可以把一个强连通分量看成一个点。

接下来枚举所有的边，如果是一个强连通分量里的就忽略，否则连接两个对应的强连通分量。这个操作称为缩点。

缩点后就变成了一张有向无环图，处理连通性问题的时候会方便很多。

无向图——割点

无向图——割点

对于一张无向图，我们希望求出它的割点。

无向图的割点定义为删掉这个点之后，连通块数量会发生改变的点。

无向图——割点

无向图——割点

类比上面，我们还是记录一下 dfn 和 low。

对于 u 的一个子节点 v，若 dfn[u]≤low[v]，则 u 是割点（因为 v 无法绕过 u 往上走）。

无向图——割点

类比上面，我们还是记录一下 dfn 和 low。

对于 u 的一个子节点 v，若 dfn[u]≤low[v]，则 u 是割点（因为 v 无法绕过 u 往上走）。

不过需要注意两点：

- 根节点不能用这种方法，而是应该看它的子节点数量是否大于等于 2，如果是那么根节点就是割点。
- 枚举出边的时候要特判掉父子边的情况。

无向图——桥

无向图——桥

无向图的桥定义为删掉这条边后，连通块数量会发生改变的边。

无向图——桥

无向图的桥定义为删掉这条边后，连通块数量会发生改变的边。

和上面的方法几乎一模一样，唯一的区别是判断 dfn[u]<low[v] 而不是 dfn[u]≤low[v]。（如果从 v 出发连 u 都无法到达，那么 (u, v) 就是一个桥边）

甚至连根节点都不需要特判了。

无向图——点/边双连通分量

无向图——点/边双连通分量

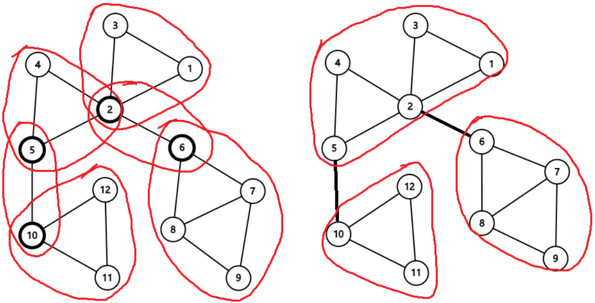
如果两个点之间存在两条**点**互不相交的路径，那么就称这两个点是**点双连通**的。

如果两个点之间存在两条**边**互不相交的路径，那么就称这两个点是**边双连通**的。

其余的定义参考强连通分量。

无向图——点/边双连通分量

无向图——点/边双连通分量



加粗的点/边代表割点和桥，红圈表示点/边双连通分量。

无向图——点/边双连通分量

无向图——点/边双连通分量

可以发现，割点将整张图分成了若干个点双连通分量，并且一个割点可以在多个点双连通分量中。

而桥则把整张图拆成了若干个边双连通分量，并且桥不在任意一个边双连通分量中。

无向图——点/边双连通分量

可以发现，割点将整张图分成了若干个点双连通分量，并且一个割点可以在多个点双连通分量中。

而桥则把整张图拆成了若干个边双连通分量，并且桥不在任意一个边双连通分量中。

魔改一下强连通分量算法即可。

当然，无向图也可以缩点，不过主要还是可以用来建圆方树。

【NOIP2009】最优贸易

【NOIP2009】最优贸易

给定一张混合图，每个点有一个价格。

你需要从一个点 u 买入，然后走到另一个点 v 卖出。求能赚的最大差价。

$1 \leq n \leq 100000, 1 \leq m \leq 500000$

【NOIP2009】最优贸易

【NOIP2009】最优贸易

首先求强连通分量，然后缩点，每一个点记录一个最小价格和最大价格。

接下来我们只需要对每一个点，求出能够到达这个点的最小价格是多少。

【NOIP2009】最优贸易

首先求强连通分量，然后缩点，每一个点记录一个最小价格和最大价格。

接下来我们只需要对每一个点，求出能够到达这个点的最小价格是多少。

直接 DAG 上 DP 即可。时间复杂度 $O(n)$。

【HAOI2006】受欢迎的牛

【HAOI2006】受欢迎的牛

一张 n 个点 m 条边的图，求有哪些点可以被其余所有点到达。

$n \leq 10^4, m \leq 5 \times 10^4$（注意是 2006 年）

【HAOI2006】受欢迎的牛

【HAOI2006】受欢迎的牛

板子题。

首先跑强连通分量，然后缩点。

接下来就变成了一张有向无环图。

【HAOI2006】受欢迎的牛

板子题。

首先跑强连通分量，然后缩点。

接下来就变成了一张有向无环图。

如果出度为 0 的点只有一个，那么这个强连通分量里所有的点都是合法的。

否则，不存在合法的点。

【HNOI2012】矿场搭建

【HNOI2012】矿场搭建

一个矿场可以描述为 n 个点 m 条边的图。

你需要在若干个位置设置逃生出口，使得无论哪一个点坍塌，其余所有的点都有通向逃生出口的路径。

求最少设置多少个出口，以及设置最少出口的方案数。

【HNOI2012】矿场搭建

【HNOI2012】矿场搭建

假设图是连通的。

【HNOI2012】矿场搭建

假设图是连通的。

首先求一遍点双连通分量，并且找出所有的割点。

【HNOI2012】矿场搭建

假设图是连通的。

首先求一遍点双连通分量，并且找出所有的割点。

如果坍塌的不是割点，由于整张图还是连通的，所以只需要在这个点之外有一个出口即可。

如果坍塌的是割点，则要求去掉这个割点之后每一个连通块内至少有一个出口。

【HNOI2012】矿场搭建

假设图是连通的。

首先求一遍点双连通分量，并且找出所有的割点。

如果坍塌的不是割点，由于整张图还是连通的，所以只需要在这个点之外有一个出口即可。

如果坍塌的是割点，则要求去掉这个割点之后每一个连通块内至少有一个出口。

可以发现，如果一个点双连通分量里只有一个割点，那么这个双连通分量中必须要设置一个不同于割点的出口。

特判一下整张图双连通的情况，这种情况下随便找两个点弄两个出口即可。

【HNOI2012】矿场搭建

假设图是连通的。

首先求一遍点双连通分量，并且找出所有的割点。

如果坍塌的不是割点，由于整张图还是连通的，所以只需要在这个点之外有一个出口即可。

如果坍塌的是割点，则要求去掉这个割点之后每一个连通块内至少有一个出口。

可以发现，如果一个点双连通分量里只有一个割点，那么这个双连通分量中必须要设置一个不同于割点的出口。

特判一下整张图双连通的情况，这种情况下随便找两个点弄两个出口即可。

方案数乘一下就完事了。

POJ3352 Road Construction

POJ3352 Road Construction

给你一张图，求至少添加多少条边可以使整张图边双连通。

POJ3352 Road Construction

POJ3352 Road Construction

首先求一遍边双连通分量，然后缩点。
可以发现缩完点之后一定是一棵树。

POJ3352 Road Construction

首先求一遍边双连通分量，然后缩点。

可以发现缩完点之后一定是一棵树。

将叶子两两相连即可。如果有 k 个叶子，答案即为 $\lceil \frac{k}{2} \rceil$。注意特判一下 $k = 1$ 的情况。

【AHOI2005】航线规划

【AHOI2005】航线规划

给你一张 n 个点 m 条边的图，你需要支持两个操作：

- 删除一条边
- 询问两个点之间的关键边条数。关键边定义为删掉后会使得两个点不连通的边。

$n \leq 30000, m \leq 100000$

【AHOI2005】航线规划

【AHOI2005】航线规划

假设已经求出了边双连通分量并且完成了缩点。

那么关键边条数就是两个点之间桥边的数量，也就是缩点后两个点之间的距离。

【AHOI2005】航线规划

假设已经求出了边双连通分量并且完成了缩点。

那么关键边条数就是两个点之间桥边的数量，也就是缩点后两个点之间的距离。

删除一条边不太好处理，我们考虑加入一条边。加边等价于将树上的一条链合并为一个点。

问题转化为给你一棵树，支持缩一条链和询问两点之间的距离。

【AHOI2005】航线规划

【AHOI2005】航线规划

时间反演一下，变成插入边和询问。

如果我们在 u 和 v 之间加入了一条边，那么就把树上 u 到 v 之间的所有边都标记为不是关键边。

【AHOI2005】航线规划

时间反演一下，变成插入边和询问。

如果我们在 u 和 v 之间加入了一条边，那么就把树上 u 到 v 之间的所有边都标记为不是关键边。

至于询问，就是看树上 u 到 v 的路径上有多少条边还没有被标记。

树剖/LCT 维护。（有点超纲）时间复杂度 $O(m\log^2 n)$ 或 $O(m\log n)$。

相关定义

相关定义

匹配：在图论中，一个匹配（matching）是一个边的集合，其中任意两条边都没有公共顶点。

相关定义

匹配：在图论中，一个匹配（matching）是一个边的集合，其中任意两条边都没有公共顶点。

最大匹配：一个图所有匹配中，所含匹配边数最多的匹配，称为这个图的最大匹配。

完美匹配：如果一个图的某个匹配中，所有的顶点都是匹配点，那么它就是一个完美匹配。

相关定义

匹配：在图论中，一个匹配（matching）是一个边的集合，其中任意两条边都没有公共顶点。

最大匹配：一个图所有匹配中，所含匹配边数最多的匹配，称为这个图的最大匹配。

完美匹配：如果一个图的某个匹配中，所有的顶点都是匹配点，那么它就是一个完美匹配。

如果要求一般图的最大匹配，需要用 $O(n^3)$ 的带花树，至少是 NOI+ 的算法。在联赛阶段，我们一般只关注二分图的匹配问题。

二分图：如果一个图的顶点能够被分为两个集合 X, Y，满足每一个集合内部都没有边相连，那么这张图被称作是一张二分图。

最大匹配——匈牙利算法

最大匹配——匈牙利算法

在进行匈牙利算法之前，我们先做两个比较重要的定义：

交替路：从一个未匹配点出发，依次经过非匹配边——匹配边——非匹配边——……形成的路径叫交替路。

增广路：从一个未匹配点出发，依次经过非匹配边——匹配边——非匹配边——……——非匹配边，最后到达一个未匹配点形成的路径叫增广路。

最大匹配——匈牙利算法

在进行匈牙利算法之前，我们先做两个比较重要的定义：

交替路：从一个未匹配点出发，依次经过非匹配边——匹配边——非匹配边——⋯⋯形成的路径叫交替路。

增广路：从一个未匹配点出发，依次经过非匹配边——匹配边——非匹配边——⋯⋯——非匹配边，最后到达一个未匹配点形成的路径叫增广路。

注意到，一旦我们找出了一条增广路，将这条路径上所有匹配边和非匹配边取反，就可以让匹配数量 $+1$。

匈牙利算法就是基于这个原理。

最大匹配——匈牙利算法

最大匹配——匈牙利算法

假设我们已经得到了一个匹配，希望找到一个更大的匹配。

最大匹配——匈牙利算法

假设我们已经得到了一个匹配，希望找到一个更大的匹配。

我们从一个未匹配点出发进行 dfs，如果找出了一个增广路，就代表增广成功，我们找到了一个更大的匹配。

如果增广失败，可以证明此时就是最大匹配。

最大匹配——匈牙利算法

假设我们已经得到了一个匹配，希望找到一个更大的匹配。

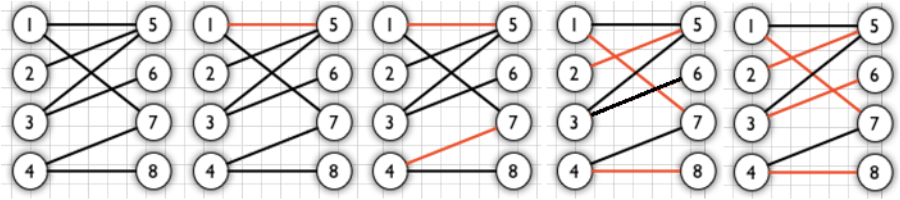
我们从一个未匹配点出发进行 dfs，如果找出了一个增广路，就代表增广成功，我们找到了一个更大的匹配。

如果增广失败，可以证明此时就是最大匹配。

由于每个点只会被增广一次，所以时间复杂度是 $O(n(n+m))$。

最大匹配——匈牙利算法

最大匹配——匈牙利算法



在第三次增广中，我们找到了 2-5-1-7-4-8 这样一条增广路，匹配边和非匹配边取反后匹配数量多了 1。

二分图最大权匹配——KM 算法

二分图最大权匹配——KM 算法

现在我们把所有的边都带上权值，希望求出所有最大匹配中权值之和最大的匹配。

二分图最大权匹配——KM 算法

现在我们把所有的边都带上权值，希望求出所有最大匹配中权值之和最大的匹配。

我们的思路是给每一个点赋一个“期望值”，也叫作顶标函数 c，对于 (u, v) 这条边来说，只有 $c(u) + c(v) = w(u, v)$ 的时候，才能被使用。

容易发现，此时的答案就是 $\sum c(i)$。

二分图最大权匹配——KM 算法

现在我们把所有的边都带上权值，希望求出所有最大匹配中权值之和最大的匹配。

我们的思路是给每一个点赋一个“期望值”，也叫作顶标函数 c，对于 (u,v) 这条边来说，只有 $c(u)+c(v)=w(u,v)$ 的时候，才能被使用。

容易发现，此时的答案就是 $\sum c(i)$。

初始，我们令左边所有点的 $c(u)=\max_v w(u,v)$，也就是说最理想的情况下，每一个点都被权值最大的出边匹配。

二分图最大权匹配——KM 算法

二分图最大权匹配——KM 算法

接下来开始增广，每次只找符合要求的边。我们定义只走这些边访问到的子图为相等子图。

如果能够找到增广路就直接增广，否则，就把这次增广访问到的左边的所有点的 $c-1$，右边所有点的 $c+1$。

二分图最大权匹配——KM 算法

接下来开始增广，每次只找符合要求的边。我们定义只走这些边访问到的子图为相等子图。

如果能够找到增广路就直接增广，否则，就把这次增广访问到的左边的所有点的 $c-1$，右边所有点的 $c+1$。

经过这样一通操作，我们发现原来的匹配每一条边仍然满足条件。同时由于访问到的点左边比右边多一个（其余的都匹配上了），所以这样会导致总的权值 -1。

二分图最大权匹配——KM 算法

接下来开始增广，每次只找符合要求的边。我们定义只走这些边访问到的子图为相等子图。

如果能够找到增广路就直接增广，否则，就把这次增广访问到的左边的所有点的 $c-1$，右边所有点的 $c+1$。

经过这样一通操作，我们发现原来的匹配每一条边仍然满足条件。同时由于访问到的点左边比右边多一个（其余的都匹配上了），所以这样会导致总的权值 -1。

接下来再尝试进行增广，重复上述过程。直接这样做时间复杂度是 $O(n^3c)$ 的。（进行 n 次增广，每次修改 c 次顶标，访问所有 n^2 条边）

二分图最大权匹配——KM 算法

二分图最大权匹配——KM 算法

一些优化：

- 由于修改顶标的目标是让相等子图变大，因此可以每次加减一个最小差值 delta。这样每次增广只会被修改最多 n 次顶标，时间复杂度降到 $O(n^4)$。
- 注意到每次重新进行 dfs 太不优秀了，可以直接进行 bfs，每次修改完顶标之后接着上一次做。时间复杂度降到 $O(n^3)$。

二分图最大权匹配——KM 算法

一些优化:

- 由于修改顶标的目标是让相等子图变大，因此可以每次加减一个最小差值 delta。这样每次增广只会被修改最多 n 次顶标，时间复杂度降到 $O(n^4)$。
- 注意到每次重新进行 dfs 太不优秀了，可以直接进行 bfs，每次修改完顶标之后接着上一次做。时间复杂度降到 $O(n^3)$。

(哪来那么多麻烦事，直接写费用流不就完事了吗!)

一般图的情况？

一般图的情况？

一般图最大匹配？

$O(n^3)$ 带花树。集训队集训的时候考过一道（然而我不会）。

一般图的情况?

一般图最大匹配?

$O(n^3)$ 带花树。集训队集训的时候考过一道（然而我不会）。

一般图最大权匹配?

带权带花树?

一些技巧

一些技巧

最小点覆盖：选取最少的点，使得每一条边的两端至少有一个点被选中。

一些技巧

最小点覆盖：选取最少的点，使得每一条边的两端至少有一个点被选中。

二分图的最小点覆盖 = 最大匹配。

Proof.

1. 由于最大匹配中的边必须被覆盖，因此匹配中的每一个点对中都至少有一个被选中。
2. 选中这些点后，如果还有边没有被覆盖，则找到一条增广路，矛盾。

□

一些技巧

一些技巧

最大独立集：选取最多的点，使得任意两个点不相邻。

一些技巧

最大独立集：选取最多的点，使得任意两个点不相邻。

最大独立集 = 点数-最小点覆盖。

Proof.

1. 由于最小点覆盖覆盖了所有边，因此选取剩余的点一定是一个合法的独立集。
2. 若存在更大的独立集，则取补集后得到了一个更小的点覆盖，矛盾。

□

一些技巧

一些技巧

最小边覆盖：选取最少的边，使得每一个点都被覆盖。

一些技巧

最小边覆盖：选取最少的边，使得每一个点都被覆盖。

最小边覆盖 = 点数-最大匹配。

Proof.

1. 先选取所有的匹配边，然后对剩下的每一个点都选择一条和它相连的边，可以得到一个边覆盖。
2. 若存在更小的边覆盖，则因为连通块数量 = 点数-边数，这个边覆盖在原图上形成了更多的连通块，每一个连通块内选一条边，我们就得到了一个更大的匹配。

□

一些技巧

一些技巧

最小不相交路径覆盖：一张有向图，用最少的链覆盖所有的点，链之间不能有公共点。

将点和边分别作为二分图的两边，然后跑匹配，最小链覆盖 = 原图点数-最大匹配。

一些技巧

最小不相交路径覆盖：一张有向图，用最少的链覆盖所有的点，链之间不能有公共点。

将点和边分别作为二分图的两边，然后跑匹配，最小链覆盖 = 原图点数-最大匹配。

最小可相交路径覆盖：一张有向图，用最少的链覆盖所有的点，链之间可以有公共点。

先跑一遍传递闭包，然后变成最小不相交路径覆盖。

【ZJOI2007】矩阵游戏

【ZJOI2007】矩阵游戏

给一个 $n \times n$ 的黑白方阵，你可以任意交换两行或者两列。
求是否能够让主对角线上都是黑色。
$n \leq 200$，多测最多 20 组。

【ZJOI2007】矩阵游戏

【ZJOI2007】矩阵游戏

建一个二分图，左边是行，右边是列。如果 (i, j) 是黑色就从 i 到 j 连边。

最后看是否存在完美匹配。

POJ2446 Chessboard

POJ2446 Chessboard

一个 $n \times m$ 的网格，其中有若干个位置被删掉了。
你需要用 1×2 的骨牌覆盖其余所有的位置，判断是否可行。
$1 \le n, m \le 32$

POJ2446 Chessboard

POJ2446 Chessboard

将整个棋盘进行黑白染色，则一个骨牌一定覆盖了相邻两个颜色不同的位置。

相邻位置连边，然后跑二分图匹配，如果存在完美匹配那就可以，否则不行。

ZOJ3988 Prime Set

ZOJ3988 Prime Set

给定 n 个整数 $a_1, \dots, a_n$，你需要从中选出 k 个数对 $\{a_{x_1}, a_{y_1}\}, \{a_{x_2}, a_{y_2}\}, \dots, \{a_{x_k}, a_{y_k}\}$（数对之间可以有重复元素），使得每一对数对的和都是质数。

你需要最大化 $\bigcup_{i=1}^{k}\{x_i, y_i\}$。

$1 \le n \le 3 \times 10^3, 1 \le k \le \frac{n(n-1)}{2}, 1 \le a_i \le 10^6$

ZOJ3988 Prime Set

ZOJ3988 Prime Set

首先跑欧拉筛，得出所有可以形成质数的数对。

先忽略 $1+1=2$ 的情况，那么所有合法的数对一定奇偶不同。

ZOJ3988 Prime Set

首先跑欧拉筛，得出所有可以形成质数的数对。

先忽略 $1+1=2$ 的情况，那么所有合法的数对一定奇偶不同。

建图，跑一遍最大匹配，然后再把 $1+1=2$ 的情况考虑进去，剩下的贪心选取。

注意 1 的位置需要特殊处理。

POJ2226 Muddy Fields

POJ2226 Muddy Fields

一个 $n \times m$ 的矩阵，其中有一些格子上有泥。

你需要用木板覆盖所有有泥的格子。木板的宽度是 1，长度任意。

木板不能盖到没有泥的格子，木板之间可以重叠。

求最少用多少块木板。

$1 \leq n, m \leq 50$

POJ2226 Muddy Fields

POJ2226 Muddy Fields

木板肯定尽量长，所以只需要起始位置和方向，这块木板就确定了。

对于一个有泥的格子 (i, j)，它被覆盖等价于经过它的横竖方向的木板至少存在其中一个。

POJ2226 Muddy Fields

木板肯定尽量长，所以只需要起始位置和方向，这块木板就确定了。

对于一个有泥的格子 (i, j)，它被覆盖等价于经过它的横竖方向的木板至少存在其中一个。

建图，则问题变成了求二分图的最小点覆盖。