SCP-S 2026 模拟讲评

洛谷网校
gza
2026年8月16日

洛谷
www.luogu.com.cn

选择题

1. 在 Linux 系统下本地测试函数式交互题，源程序名为 scp.cpp，源程序内引用了头文件 scp.h，题目下发交互库为 grader.cpp，则以下哪条编译命令可以输出正确的可执行文件 scp？（ ）

A. g++ grader.cpp scp.h -o scp -O2 -std=c++14 -static

B. g++ grader.cpp scp.cpp -o scp -O2 -std=c++14 -static

C. g++ scp.h scp.cpp -o scp -O2 -std=c++14 -static

D. g++ scp.cpp scp.h -o scp -O2 -std=c++14 -static

看到编译命令题，先把文件分成两类：.cpp 是需要编译、链接的源文件，.h 是由预处理器展开的头文件。然后再找谁提供 main、谁提供选手函数。scp.h 是头文件，由 scp.cpp 中的 #include 在预处理阶段展开，不应作为一个独立翻译单元参与链接。函数式交互题通常由 grader.cpp 提供 main 函数并调用选手在 scp.cpp 中实现的函数，因此应同时编译并链接这两个源文件：

g++ grader.cpp scp.cpp -o scp -O2 -std=c++14 -static

选 B。

选择题

2. 关于最短路算法，以下说法中完全正确的是。（）

A. 只要图中没有负环，Dijkstra 算法就一定能求出正确的最短路径。

B. SPFA 算法在任何图上的最坏时间复杂度都比 Dijkstra 优秀。

C. 标准的 Dijkstra 算法（基于贪心，结点出队后不再更新）在包含负权边的图中可能会得到错误的结果。

D. Floyd 算法只能求任意两点间的最短路，无法判断图中是否存在负环。

这类题不要凭“哪个算法更强”判断，而要逐一回忆每个算法正确性的必要条件：
Dijkstra 要求非负边权，SPFA 只是 Bellman-Ford 的队列优化，Floyd 的对角线还能反映负环。
Dijkstra 正确的常见前提是所有边权非负；没有负环并不足够，所以 A 错。
SPFA 的最坏复杂度可达 O(nm)，不能保证优于 Dijkstra，所以 B 错。
含负权边时，一个已经被标准 Dijkstra 确定的点可能还会被后来经过负边的路径改小，因此 C 对。
Floyd 后若存在某个 $d_{i,i} < 0$，即可判定存在负环，所以 D 错。
选 C。

选择题

3. 某算法的时间复杂度的递归式为 $T(n)=2T(\sqrt{n})+\Theta(\log n)$，则其渐进时间复杂度为？（ ）

A. $\Theta(\log n)$

B. $\Theta(\log\log n)$

C. $\Theta(\log^2 n)$

D. $\Theta(\log n\log\log n)$

令 $S(n)=2S\left(\frac{n}{2}\right)+\Theta(n)$，有 $S(n)=T(2^n)$。
又有 $S(n)=\Theta(n\log n)$，则有 $T(n)=\Theta(\log n\log\log n)$。
选 D。

选择题

4. 使用数 1,2,3,4,5 各一个以及 2 个加号、2 个减号可组成的后缀表达式的值最大为？（）

A. 9

B. 11

C. 12

D. 13

由于只有加减号，所以最终的答案可以看做在 12345 前分配正负号，要求有 2 个负号 3 个正号或者 3 个负号两个正号，贪心的想，答案肯定是 -1-2+3+4+5=13，但是这个上界取得到吗？答案是肯定的，5 1 2 3 4 + + - - 即可达到这个上界。
选 D。

选择题

5. 运行如下 C++ 代码片段后，关于数组 v 的状态，下述说法正确的是？（ ）

```
std::vector<int> v = {9, 2, 7, 4, 5, 8, 1, 3, 6};
std::nth_element(v.begin(), v.begin() + 3, v.end());
```

A. v[3] 的值必定是 4，且 v[0] 到 v[2] 的值必定依次是 1,2,3。

B. v[3] 的值必定是 4，且 v[0] 到 v[2] 包含 1,2,3 这三个数。

C. v[3] 的值必定是 4，且整个数组已经被完全排序。

D. v[0] 到 v[3] 包含 1,2,3,4，且右侧的 v[4] 到 v[8] 必定已按从小到大排序。

先区分“第 k 个位置正确”与“整个前缀有序”。nth_element 只保证分界点及左右两侧的大小关系，不负责两侧内部排序。
nth_element(first, nth, last) 保证 nth 位置放置完全排序后应在该位置的元素；其左侧元素都不大于它，右侧元素都不小于它，但两侧内部不保证有序。
数组元素互不相同，v.begin()+3 对应第 4 小的数 4。左侧三个位置只能放 1,2,3，但次序不确定。
选 B。

选择题

6. 使用 std::set 维护有序集合时，设集合 s 中现有 n 个元素，则使用 std::lower_bound(s.begin(), s.end(), x) 查找第一个 $\geq x$ 的元素的时间复杂度为？（ ）

A. $\Theta(\log n)$

B. $\Theta(\log^2 n)$

C. $\Theta(n)$

D. $\Theta(n \log n)$

std::set 的底层实现是平衡树，因此不支持常数时间的随机跳转，直接使用 std::lower_bound 会导致复杂度退化为 $O(n)$，但是使用 std::set 的成员函数 lower_bound（s.lower_bound(…)）可以保证复杂度为 $O(\log n)$，这是一个结论，大家记住就好了。具体的解释可以翻阅一下 set 的源码或者问一下 AI，这里不再赘述。

选 C。

选择题

7. 以下选项中，哪个序列不可能是对长度为 6 的字符串 S 运行 KMP 算法得到的 next 数组（定义 next[i] 为 $S[1...i]$ 的最长相等真前后缀长度）？（）

A. 0 0 0 1 2 3

B. 0 0 0 1 2 1

C. 0 1 0 1 2 1

D. 0 1 2 0 1 2

KMP 的 next 数组的含义是最长相等真前后缀长度，因此如果 next[i]=j，那么就有 S[1:j]=S[i-j+1:i]，因此我们可以对于 next 数组确定哪些字符是相等的，但是我们还是不确定有些字符之间是否相等，这就需要我们尝试构造原字符串，看是否符合选项中的 next 数组。
A 可由 abcabc 得到。
B 可由 abcaba 得到。
D 可由 aaabaa 得到。
稍微手玩一会儿就会发现 C 是构造不出来的。
选 C。

选择题

8. 对 10 个互不相同且均在 [0,31] 之间的整数建立深度为 5（定义为叶子到根的距离）

的 01-Trie，所可能得到的最多结点数和最少结点数之差为？（ ）

A. 0

B. 13

C. 22

D. 55

贪心地想，求最大值时让前缀尽量分散；求最小值时让前缀尽量共享。

最大值：深度 d 最多有 $\min(2^d, 10)$ 个点，加起来得到最大值为 35。

最小值：深度 d 最多有 $\lceil 10/2^{5-d} \rceil$ 个点，这个下界可以逐层实现：先把 10 个叶子尽量塞进同一个大子树，容量不足时才新开前缀，加起来得到最小值为 22。

$$35 - 22 = 13$$

选 B。

选择题

9. 使用数据结构维护序列区间信息 $al \oplus al+1 \oplus \cdots \oplus ar$，其中 $\oplus$ 为一般二元运算，下列说法正确的是？（ ）

A. ST 表能 $\Theta(1)$ 回答区间询问的原因是利用了区间的重叠，要求 $\oplus$ 有交换律。

B. 若使用线段树进行单点修改和区间查询，$\oplus$ 可以不满足结合律，但必须满足交换律。

C. 基于前缀差分方式实现的树状数组，要询问区间信息，则要求 $\oplus$ 必须可逆（存在逆元）。

D. 若使用平衡树维护序列区间信息，$\oplus$ 必须同时满足交换律和结合律，否则无法合并。

这一题乍一看很吓人，但是只要我们了解各个数据结构的使用条件就可以了。

ST 表是利用了两个重叠的区间合成出了查询区间，因此要求二元运算有可重复贡献的性质，即 $x \oplus x = x$。

线段树和平衡树只要求二元运算有结合律。

树状数组由于查询的是一个前缀，因此使用两个前缀相互抵消，就得到了一个区间，这要求二元运算可逆，即对于每个元素 x，存在 inv_x，使得 $x \oplus inv_x = e$，其中 e 是该二元运算的单位元，满足对于任意元素 x，有 $x \oplus e = e \oplus x = x$。

选择题

9. 使用数据结构维护序列区间信息 $al \oplus al+1 \oplus \cdots \oplus ar$，其中 $\oplus$ 为一般二元运算，下列说法正确的是？（ ）

A. ST 表能 Θ(1) 回答区间询问的原因是利用了区间的重叠，要求 $\oplus$ 有交换律。

B. 若使用线段树进行单点修改和区间查询，$\oplus$ 可以不满足结合律，但必须满足交换律。

C. 基于前缀差分方式实现的树状数组，要询问区间信息，则要求 $\oplus$ 必须可逆（存在逆元）。

D. 若使用平衡树维护序列区间信息，$\oplus$ 必须同时满足交换律和结合律，否则无法合并。

然后我们来看选项。
ST 表要求的是可重复贡献，而不是交换律，因此 A 错。
线段树要求满足结合律，因此 B 错。
平衡树并未要求有交换律，因此 D 错。
选 C。

选择题

10. 对于一个包含 10 个顶点的无向简单图 G，设其补图为 $\overline{G}$，下列说法正确的是？（）

A. 若 G 不连通，那么 $\overline{G}$ 必然也不连通。

B. 存在某个 G，使得 G 和 $\overline{G}$ 都是一棵树。

C. 存在某个 G，使得 G 和 $\overline{G}$ 同时为二分图。

D. 不存在 G，使得 G 和 $\overline{G}$ 同时拥有欧拉回路。

取 G 为没有任何边的图，则 $\overline{G}$ 为完全图，显然连通，A 错误。

G 和 $\overline{G}$ 的边数相加等于 $\binom{10}{2} = 45$（并起来是完全图），而树的边数为 $10-1=9$，$9+9 \neq 45$，B 错误。

若 G 为二分图，设其两部大小分别为 x 和 $10-x$，则 $\overline{G}$ 种存在两个大小为 x 和 $10-x$ 的团，由于 $\max(x, 10-x) \geq 3$，所以 $\overline{G}$ 中一定存在一个大小为三的奇环，因此 $\overline{G}$ 不可能是二分图，C 错误。

存在欧拉回路的充要条件是所有点度数为偶数，但是一个点在两个图中的度数和为 9，不可能在每个图中度数都为偶数，因此 D 正确。

选 D。

选择题

11. 计算 $\begin{bmatrix}1 & 1\\ 1 & 0\end{bmatrix}^{2026}$ 每项对 3 取模的值为？（ ）

A. $\begin{bmatrix}1 & 2\\ 2 & 0\end{bmatrix}$　　B. $\begin{bmatrix}1 & 2\\ 2 & 1\end{bmatrix}$

C. $\begin{bmatrix}2 & 1\\ 1 & 0\end{bmatrix}$　　D. $\begin{bmatrix}2 & 1\\ 1 & 1\end{bmatrix}$

看到矩阵 $\begin{bmatrix}1 & 1\\ 1 & 0\end{bmatrix}$，应联想到斐波那契转移。指数很大但模数只有 3，继续想到斐波那契数模 3 的短周期，便可避免做十多次矩阵快速幂。

设 $A=\begin{bmatrix}1 & 1\\ 1 & 0\end{bmatrix}$。有

$$A^n=\begin{bmatrix}F_{n+1} & F_n\\ F_n & F_{n-1}\end{bmatrix}$$

斐波那契数列模 3 的周期为 8，而 $2026\equiv 2 \pmod 8$，故只需算

$$A^2=\begin{bmatrix}2 & 1\\ 1 & 1\end{bmatrix} \pmod 3$$

选 D。

选择题

12. 有一个正十二面体，共 20 个顶点、30 条棱，每个面都是正五边形，其棱长为 1，顶点之间只能通过棱互相到达，则所有 $\binom{20}{2}$ 个点对间最短路长度之和为？（）

A. 420

B. 500

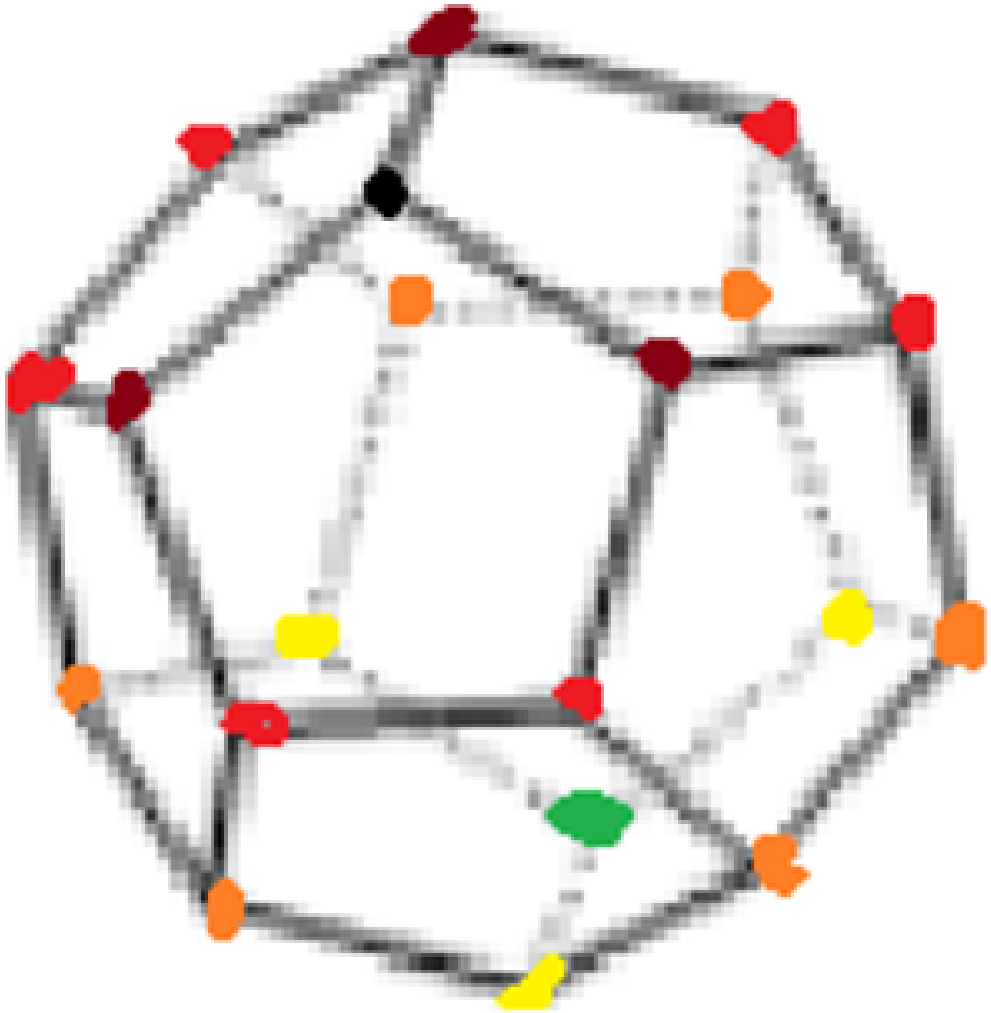
C. 600

D. 760

正十二面体的顶点是完全对称的，所以不必枚举 $\binom{20}{2}$ 个点对。只要固定一个顶点，按距离做 BFS 分层，再利用“每个点对被两个端点各统计一次”即可。

$$3 \times 1 + 6 \times 2 + 6 \times 3 + 3 \times 4 + 1 \times 5 = 50$$

对所有 20 个起点求和时每个无序点对被算两次，故答案为 $20 \times 50/2 = 500$，选 B。



选择题

13. 有一排 10 盏灯，初始时全部为关闭状态；每次操作可以选定一个连续区间，并将该区间内的灯亮灭状态反转，则恰好 3 次操作后，这 10 盏灯可能呈现出多少种不同的亮灭状态？（）

A. 120

B. 512

C. 848

D. 1024

直接研究十盏灯会被区间覆盖多少次比较乱。区间翻转最常见的转化是差分：一个区间只改变两个端点处的差分值。于是每次操作从“翻很多盏灯”变成“选择两个边界”。

在 10 盏灯的两端和相邻灯之间设置 11 个“边界”。翻转区间 [l,r]，等价于翻转边界 l-1 与 r。三次操作一共选择三对边界，最终状态由被选择奇数次的边界集合唯一确定。

灯的最终状态与这 11 个边界差分一一对应：从左到右累积差分即可恢复每盏灯。因此数边界集合就是数灯态，不会计重。

每次贡献两个端点，三次共出现六次端点，所以被选择奇数次的边界个数必为偶数，且不超过 6，只能是 0,2,4,6。反过来还要证明这些规模全部可达：

0 个：选三个边界间构成的三条两两连接；

2 个：同一区间操作三次，或用一条三边路径；

4 个：以一个边界为中心选择三条边；

6 个：选择三对互不相同的边界。

因此答案为：

$$\binom{11}{0}+\binom{11}{2}+\binom{11}{4}+\binom{11}{6}=1+55+330+462=848。$$

选 C。

选择题

14. 对于长度为 2026 的排列 p，初始时对于所有位置 i 均有 $p_i = i$；现在进行恰好 10^4 次操作，每次操作可以交换相邻两个位置上的数，则操作完成后得到的新排列 p' 的最长上升子序列长度最短是多少？（ ）

A. 110

B. 183

C. 186

D. 187

从单位排列出发，相邻交换的最少次数等于目标排列的逆序对数。

若一个排列的 LIS 长度不超过 r，由 Dilworth 定理，可把它划分成 r 个下降子序列。设这些子序列长度为 $s_1, \dots, s_r$，每个下降子序列内部任意一对都是逆序对，因此逆序对数至少为 $\sum \binom{s_i}{2}$。贪心地想，我们肯定要尽可能平均的分配 s_i。

先尝试 $r = 186$。因为 $2026 = 166 \times 11 + 20 \times 10$，最少逆序对数为

$$166\binom{11}{2} + 20\binom{10}{2}$$
$$= 166 \times 55 + 20 \times 45 = 10030$$
$$> 10000$$

所以最终 LIS 至少为 187。

选择题

14. 对于长度为 2026 的排列 p，初始时对于所有位置 i 均有 $p_i = i$；现在进行恰好 10^4 次操作，每次操作可以交换相邻两个位置上的数，则操作完成后得到的新排列 p' 的最长上升子序列长度最短是多少？（ ）

A. 110

B. 183

C. 186

D. 187

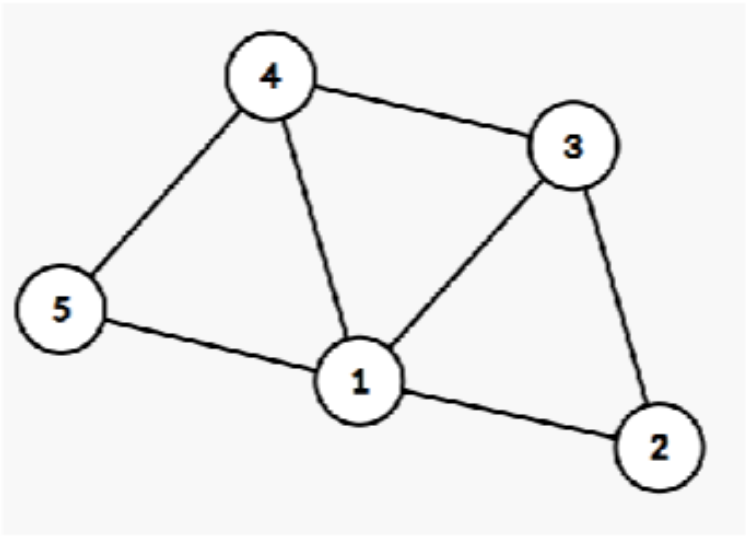
再看 $r = 187$。因为 $2026 = 156 \times 11 + 31 \times 10$，把连续数值分成 156 个长度为 11、31 个长度为 10 的块，每块内部倒序、块间数值递增。一个上升子序列在每个块内至多取一个数，但可以从每个块各取一个，因此 LIS 恰为 187。逆序对只出现在块内，数量 $156 \times 55 + 31 \times 45 = 9975$。

不妨假设我们最开始的时候塞了两个长度为 11 的块，那么排列的开头应该是 $11 \cdots 1\ 22 \cdots 12$，我们再进行 25 次的交换 1 和 22，此时排列开头变为了 $11 \cdots 2\ 22\ 1\ 21 \cdots 12$，显然 LIS 还是 187，且我们刚刚好进行了 10^4 次操作。

选 D。

选择题

15. 对于如下图所示的无向图 G，在所有将边赋予 [1,7] 中**互不相同**的整数边权的方案中，G 的最小生成树的权值和之和为？（ ）



A. 50400　　B. 53328　　C. 63408　　D. 80640

由于这个题实在是太重量级了，请同学们自行另开一个窗口来观看题面，接下来的几张 ppt 里面将不再包含题面。

选择题

我们想要计算的是最小生成树的边权和，根据经典的恒等式 $n = \sum_{i=1}^{\infty}[i \le n]$，我们设最小生成树的四条边权值为 r_1, r_2, r_3, r_4，则答案为 $\sum_{i=1}^{7}[i \le r_1] + [i \le r_2] + [i \le r_3] + [i \le r_4]$，里面的式子的含义是，在加入了权值为 $1 \sim i-1$ 的这些边以后，最小生成树里面还有几条边没加入，记这个为 e_i，再记 c_i 表示加入权值为 $1 \sim i$ 的这些边以后，当前的“最小生成树”里面有几个连通块。

有 $c_i = e_{i-1} + 1$，特殊的，$e_0 = 5$，因此 $\sum_{\mathrm{k}=1}^{7} e_{\mathrm{k}} = 4 + \sum_{\mathrm{k}=1}^{6}(c_{\mathrm{k}} - 1)$。

这把问题转化为了我们只需计算随机 k 边子图的连通块数期望。

选择题

我们先来看一些简单的情形：

$$\mathrm{k}=1$$

只加了一条边，肯定有四个连通块，$\mathrm{E}(c_1-1)=4$。

$$\mathrm{k}=2$$

由于这个图没有重边，所以 $k=2$ 时一定形成三个连通块，$E(c_2-1)=3$。

$$k=6$$

这个图是没有桥的，任意 6 边子图都连通，$E(c_6-1)=0$。

选择题

噩梦的开始。

$$\mathrm{k}=3$$

三条边若不成环，就是三边森林，此时有 2 个连通块，$c_3-1=1$。图中恰有三个三角形：1-4-5、1-3-4、1-2-3。若选中的三条边恰为一个三角形，则只有两个合并成功，连通块数为 3，所以 $c_3-1=2$。

三边集合共有 $\binom{7}{3}=35$ 个，因此

$$E(c_3-1)=\frac{2\times 3+1\times 32}{35}=\frac{38}{35}$$

选择题

$$k = 4$$

四条边若连通，因为有 5 个点，必定是一棵生成树。手玩可以发现不连通时恰有两个连通块，$c_4 - 1 = 1$。正难则反，我们只需数图中有多少棵生成树。如果不使用超纲的知识，这里只能慢慢枚举，可以数出来有 21 棵生成树，另外 14 个子图不连通，得到：

$$E(c_4 - 1) = \frac{14}{35} = \frac{2}{5}$$

简便的算法还是存在的，可以去看 P6178 【模板】Matrix-Tree 定理，只需要算一个四阶带状矩阵的行列式即可得到答案，枚举量较小。

选择题

$$k = 5$$

选五条边等价于从原图删两条边。如果想让图不连通（$c_5 - 1 \neq 0$），只能让 2 被孤立或 5 被孤立，因此只有两个五边子图满足 $c_5 - 1 = 1$，其他的全部都满足 $c_5 - 1 = 0$，不用计算，因此：

$$E(c_5 - 1) = \frac{2}{21}$$

选择题

终于，我们可以计算答案了！

$$\mathrm{E}(\mathrm{ans}) = 4 + 3 + 2 + \frac{38}{35} + \frac{2}{5} + \frac{2}{21}$$

$$= \frac{1111}{105}$$

因此所有方案的最小生成树权值和为 $\frac{1111}{105} \times 7! = 53328$。

选 B。

后话：这道题的难度极高，同学们可以做一下鉴赏，不会做不用感到难过。

阅读程序 1

```
#include<bits/stdc++.h>
using namespace std;
const int N=20,mod=998244353;
int n;
int a[N][N];
int f[1<<N];
int main(){
    cin>>n;
    for(int i=0;i<n;i++) for(int j=0;j<n;j++)
cin>>a[i][j];
    f[0]=1;
    for(int i=0;i<(1<<n);i++)
        for(int j=0;j<n;j++)
            if(i>>j&1)
                (f[i]+=1ll*f[i^(1<<j)]*a[__builtin_
popcount(i)-1][j]%mod)%=mod;
    cout<<f[(1<<n)-1];
}
```

16. （1分）当 n=20 时，程序不会出现数组越界访问。（T）
f 最多访问到 $2^{20}-1$，a 最多访问到 a[19][19]，显然不会出现数组越界。
17. 只要输入符合限定要求，无论输入的是什么，输出都一定是正整数。（F）
注意到答案是取模的，所以如果答案是 998244353 的倍数，输出的数字就是 0，不是正整数。这里给出一个输出 0 的例子。

$$\begin{bmatrix} 1 & 1 \\ 1 & 998244352 \end{bmatrix}$$

18. 删除第 16 行的 1ll*，不会对程序的输出产生任何影响。（F）
f[...] 与 a[...] 都可能接近 10^9。删除 1ll* 后会先进行 `int` 乘法，可能发生有符号整数溢出，输出当然可能改变。

阅读程序 1

```
#include<bits/stdc++.h>
using namespace std;
const int N=20,mod=998244353;
int n;
int a[N][N];
int f[1<<N];
int main(){
    cin>>n;
    for(int i=0;i<n;i++) for(int j=0;j<n;j++)
cin>>a[i][j];
    f[0]=1;
    for(int i=0;i<(1<<n);i++)
        for(int j=0;j<n;j++)
            if(i>>j&1)
                (f[i]+=1ll*f[i^(1<<j)]*a[__builtin_
popcount(i)-1][j]%mod)%=mod;
    cout<<f[(1<<n)-1];
}
```

19. 当输入为 3 1 2 3 2 3 1 3 1 2 时，输出为（ ）？

A. 52

B. 53

C. 54

D. 55

我们可以对程序进行些许的观察。

看到下标 i ^ (1 << j)，可知程序在从集合 i 中删去一个列 j。又因为所取矩阵行号是 popcount(i)-1，集合中有几个列，就正在处理第几行。因此若 $|S| = k$，$f[S]$表示给前 k 行分别选择集合 S 中互不相同列时，所有方案的乘积和。转移枚举最后一行选择的列 j，从 $f[S \setminus \{j\}]$转移而来。最终输出矩阵的积和式。

得知了这一点后，稍加计算就可以得出答案是 C，当然直接模拟程序也是可以的。

阅读程序 1

```
#include<bits/stdc++.h>
using namespace std;
const int N=20,mod=998244353;
int n;
int a[N][N];
int f[1<<N];
int main(){
    cin>>n;
    for(int i=0;i<n;i++) for(int j=0;j<n;j++)
cin>>a[i][j];
    f[0]=1;
    for(int i=0;i<(1<<n);i++)
        for(int j=0;j<n;j++)
            if(i>>j&1)
                (f[i]+=1ll*f[i^(1<<j)]*a[__builtin_
popcount(i)-1][j]%mod)%=mod;
    cout<<f[(1<<n)-1];
}
```

20. 该代码的时间复杂度为（ ）？

A. $\Theta(n^2)$

B. $\Theta(2^n)$

C. $\Theta(n \times 2^n)$

D. $\Theta(n^2 \times 2^n)$

外层循环枚举量 2^n，内层循环枚举量 n，总体时间复杂度 $\Theta(n \times 2^n)$，选 C。

21. 当 n=14 时，代码第 15 行的 if 语句中，表达式值为真的次数是（ ）？

A. 114688

B. 229176

C. 114588

D. 229376

观察到答案为 $\sum_{i=0}^{2^n-1}\sum_{j=0}^{n}[i$二进制下第j位为1$]$，我们反过来考虑每一位被统计了几次，显然全部 2^n个二进制数中刚好有一半这一位上是 1，所以答案是 $2^{n-1}n = 114688$。

选 A。

阅读程序 2

首先看 init 函数：

```
void init() {
    len[0] = 1, len[1] = 2, dp[1][1] = 1;
    for (int i = 2; i <= 86; i++) {
        len[i] = len[i - 1] + len[i - 2];
        int op = len[i - 1] & 1;
        dp[i][0] = dp[i - 1][0] + dp[i - 2][0 ^ op];
        dp[i][1] = dp[i - 1][1] + dp[i - 2][1 ^ op];
    }
}
```

容易看出 len 中存的是斐波那契数列，并且代表一个串的长度。

再结合 dp 的初值和转移，那么我们求的其实是这样一列串中，偶数 / 奇数位置上的 1 个数，其中每一项由前面两项拼接得到：

`0, 01, 010, 01001, 01001010, 010010100l001, ...`

阅读程序 2

再来看 solve1，求的就是上面说的串的前缀中，偶数 / 奇数位置上的 1 个数：

```
int solve1(int n, int p) {
    string s = "0", nxt = "01", tmp;
    while (nxt.length() < n)
        tmp = nxt + s, s = nxt, nxt = tmp;
    int ans = 0;
    for (int i = p; i < n; i += 2) ans += nxt[i] - '0';
    return ans;
}
```

那么可以猜测 solve2 和 solve3 都在干同一个事情，这里不再赘述。

下面直接来看题。

阅读程序 2

22. solve1 函数中使用朴素的字符串拼接，时间复杂度为 $\Theta(n^2)$。（F）

```
int solve1(int n, int p) {
    string s = "0", nxt = "01", tmp;
    while (nxt.length() < n)
        tmp = nxt + s, s = nxt, nxt = tmp;
    int ans = 0;
    for (int i = p; i < n; i += 2) ans += nxt[i] - '0';
    return ans;
}
```

拼接的总复杂度是斐波那契数列前缀和，而长度的增长是指数级别的，等比数列求和，总和依旧是 $\Theta(n)$ 的。

阅读程序 2

23. 将第 11 行的 `len[i - 1] & 1` 改为 `i % 3 < 2`，程序仍然能正常运行，且输出结果不变。（T）

`len` 中存的是从第二项开始的斐波那契数列：

i	0	1	2	3	4	5	6	7	8
len	1	2	3	5	8	13	21	34	55
% 2	1	0	1	1	0	1	1	0	1

有长为 3 的循环节，对比可得和 `i % 3 < 2` 是相等的，注意是 `len[i - 1]`。

阅读程序 2

24. 对于输入范围内的所有 n，`solve3(n, 0)` 与 `solve3(n, 1)` 的返回值之差的绝对值均不超过 1。（F）

感觉就很不能对，手玩一下不难找到反例：

`n = 15`，`010010100100101`，奇数位置 4 个，偶数位置 2 个。

但如果保证输入的 n 是斐波那契数的话就是对的，可以通过计算 dp 数组的值，找循环节来验证，这在后面 26 题会用到，在此略去。

阅读程序 2

25. solve2 函数在最坏情况下的时间复杂度是？（D）

A. $\Theta(n \log \log n)$　　B. $\Theta(n \log n)$

C. $\Theta(n \log n \log \log n)$　　D. $\Theta(n \log^2 n)$

```
int value(int pos) {
    if (pos <= 1) return pos;
    int k = 0;
    while (len[k + 1] <= pos) k++;
    return value(pos - len[k]);
}

int solve2(int n, int p) {
    int ans = 0;
    for (int i = p; i < n; i += 2) ans += value(i);
    return ans;
}
```

递归 O(log n) 层，每层从小到大找 k，复杂度 $O(\log^2 n)$，卡满：fib[n] - 1

调用 O(n) 次 solve2，总复杂度 $O(n\log^2 n)$

阅读程序 2

26. 对于输入数据 `233 0`，输出结果的第一个数为？（ ）

A. 44　　B. 45　　C. 55　　D. 72

233 是斐波那契数，等于 len[11]，它前两项是 89，144，根据前面的分析，答案一定是 44、45 之一。

手玩一下可以得到 dp[i][0] - dp[i][1] 有长为 6 的循环节 -1，-1，0，1，1，0，由此可知 dp[11][1] 比 dp[11][0] 大 1，故选 B。

阅读程序 2

27. 若某次调用 `solve3(n, p)` 时，语句 `ans += dp[k][p]` 恰好执行了 8 次，则输入的 n 最小可能是多少？（ ）

A. 986　　B. 1596　　C. 2583　　D. 4180

根据齐肯多夫定理，任何正整数均可表示成若干个不连续的斐波那契数之和。

换句话说，求至少需要表示成 8 个不同且不相邻的斐波那契数的，最小的 n。

那肯定从 `fib[0]` 开始从小加到大。直接一个个加很魔怔，不难通过数学归纳法证明（f 是斐波那契数列）：

$$\sum_{k=0}^{n} f_{2k} = f_{2n+1} - 1$$

那么答案为 `fib[15] - 1 = 1596`，选 B。

阅读程序 3

```
int tot, hd[S];
struct node { int nxt; ull key; } mp[M];
int get(ull key) {
    int u = key % S;
    for (int i = hd[u]; i; i = mp[i].nxt)
        if (mp[i].key == key) return i;
    return mp[++tot] = {hd[u], key}, hd[u] = tot;
}
void dfs1(int u, int p) {
    vis[u] = 1;
    for (auto [v, i] : G[u]) if (i ^ p) {
        if (!vis[v])
            dep[v] = dep[u] + 1, T[u].push_back({v, i}),
            dfs1(v, i), h[u] ^= h[v], he[i] = h[v];
        else if (dep[v] < dep[u])
            he[i] = rnd(), h[u] ^= he[i], h[v] ^= he[i];
    }
}
```

哈希表，返回 key 对应的编号，没有就新建一个。

DFS 树上非树边一定是祖先后代的关系用树上前缀和（异或两次恰好在较高的顶点处抵消），对每条树边求出跨越它的非树边的权值异或和。

给非树边赋权，贡献异或到顶点处。

阅读程序 3

先不急着看 dfs2，主函数长这样：

```
int main() {
    ios::sync_with_stdio(0), cin.tie(0);
    cin >> n >> m;
    for (int i = 1, u, v; i <= m; i++)
        cin >> u >> v >> w[i], sum = (sum + w[i]) % P,
        G[u].push_back({v, i}), G[v].push_back({u, i});
    dfs1(1, 0);
    for (int i = 1, j; i <= m; i++) if (he[i])
        j = get(he[i]), ct[j]++, sw[j] = (sw[j] + w[i]) % P;
    dfs2(1, 0, 0, 0);
    for (int i = 1; i <= n; i++) cout << ans[i] << ' ';
}
```

对每个不同的哈希值计算其对应的边数和权值和，自然要问，哈希值相同意味着什么？

阅读程序 3

做过 P6658 边三连通分量的话应该能反应过来，这里再说一下。

若 he[i] == he[j]，由于不考虑异或哈希冲突，有两种情况：

1. i 为树边，j 为非树边。意味着 j 是跨越 i 的唯一非树边，那么删去 i，j 后，原本 DFS 树上，i 的子树和 1 变得不连通。

2. i，j 均为树边。意味着覆盖 i，j 的非树边集合相同，删去 i，j 后，它们之间的连通块和 1 变得不连通。

也就是说，哈希值相同，等价于 i，j 能构成一个割集。

那么不用看 dfs2 也能猜到代码求的是什么了：割掉两条边，使得 1 和 u 不连通，所有方案的 w[i] * w[j] 之和。

其实现在已经能把题给选出来了，但还是说一下 dfs2 的式子。

阅读程序 3

先放个完整的 dfs2 在这，还请另开一个窗口来观赏完整代码。

```
void dfs2(int u, ll c, ll b, ll b2) {
    ans[u] = ((c + b * sum - (b * b + b2) % P * iv2) % P + P) % P;
    for (auto [v, i] : T[u]) {
        ll w = ::w[i];
        if (!he[i]) {
            dfs2(v, c, (b + w) % P, (b2 + w * w) % P);
            continue;
        }
        int j = get(he[i]);
        if (ct[j] == 1) dfs2(v, c, b, b2);
        else {
            int d = (w * (sw[j] - cur[j] * 2 - w) % P + P) % P;
            cur[j] = (cur[j] + w) % P;
            dfs2(v, (c + d) % P, b, b2);
            cur[j] = (cur[j] - w + P) % P;
        }
    }
}
```

阅读程序 3

答案有两类：选中了桥，或者选中的两条都不是桥。

容易看出 b，b2 分别是 1 到 u 经过的桥的权值和 / 平方和。

通过简单的容斥处理算重，不难得到第一类答案为：

$$(\sum_{\text{bridge}} w) \times \text{sum} - \frac{(\sum_{\text{bridge}} w)^2 + \sum_{\text{bridge}} w^2}{2}$$

再考虑第二类答案，也就是 c。考虑走到 u 的时候答案的变化量，割断当前边 e 和一条所在环上剩下未走的边可以使得 u 不连通；但对于环上走过的边，虽然在 u 的祖先链上算入了答案，但在 u 处不再能产生贡献，加加减减一下，这部分答案为：

$$w_e \times (\sum_{\text{circuit}} w - 2\sum_{\text{current}} w - w_e)$$

加起来就是总答案了。

阅读程序 3

下面看题。

28. 代码中使用了哈希表，攻击者可以在不获知随机数种子的前提下，通过构造数据，使得程序在该数据下的期望时间复杂度退化至平方级别。（ ）

哈希值是随机生成的，不能预测随机数的话是 hack 不了的。选 F。

29. 若对于 $i \neq j$ 有 `he[i]` 和 `he[j]` 相等且均不为 0，则图中存在一个环同时包含边 i,j。（ ）

根据上面的分析，这应当是显然的。选 T。

30. 若 $m = n - 1$，且对于 $i < n$ 有 $u_i = i, v_i = i + 1, w_i = 1$，则输出的所有数之和（在对 P 取模意义下）等于 $\frac{1}{3}n(n-1)(n-2)$。（ ）

写出来式子后是二重求和的形式，依次化简即可，在此略去。也可以直接代特殊值检验。选 T。

阅读程序 3

31. 这份程序对于每个点 u，求得了（ ）的答案对 P 取模的结果。

A. 选择两条边 $i<j$，满足存在一条 1 到 u 的路径 p，使得 i,j 中至少有一条边在 p 上，所有方案的 $w_i \times w_j$ 之和。

B. 选择两条边 $i<j$，满足存在一条 1 到 u 的路径 p，使得 i,j 都在 p 上，所有方案的 $w_i \times w_j$ 之和。

C. 选择两条边 $i<j$，满足对于所有 1 到 u 的路径 p，都有 i,j 中至少有一条边在 p 上，所有方案的 $w_i \times w_j$ 之和。

D. 选择两条边 $i<j$，满足对于所有 1 到 u 的路径 p，都有 i,j 都在 p 上，所有方案的 $w_i \times w_j$ 之和。

前面已经说过，选 C。

32. 在 `dfs2` 函数中，若执行了分支 `if (ct[j] == 1) dfs2(v, c, b, b2)`，设当前正沿树边 e 向下遍历，则 e 在原图中满足？（ ）

A. 在原图中将边 e 删去后，图的连通块数量必定会增加。

B. 在原图中将边 e 删去后，图中必定会产生至少一条新的割边。

C. 边 e 在原图中必然不属于任何一个简单环。

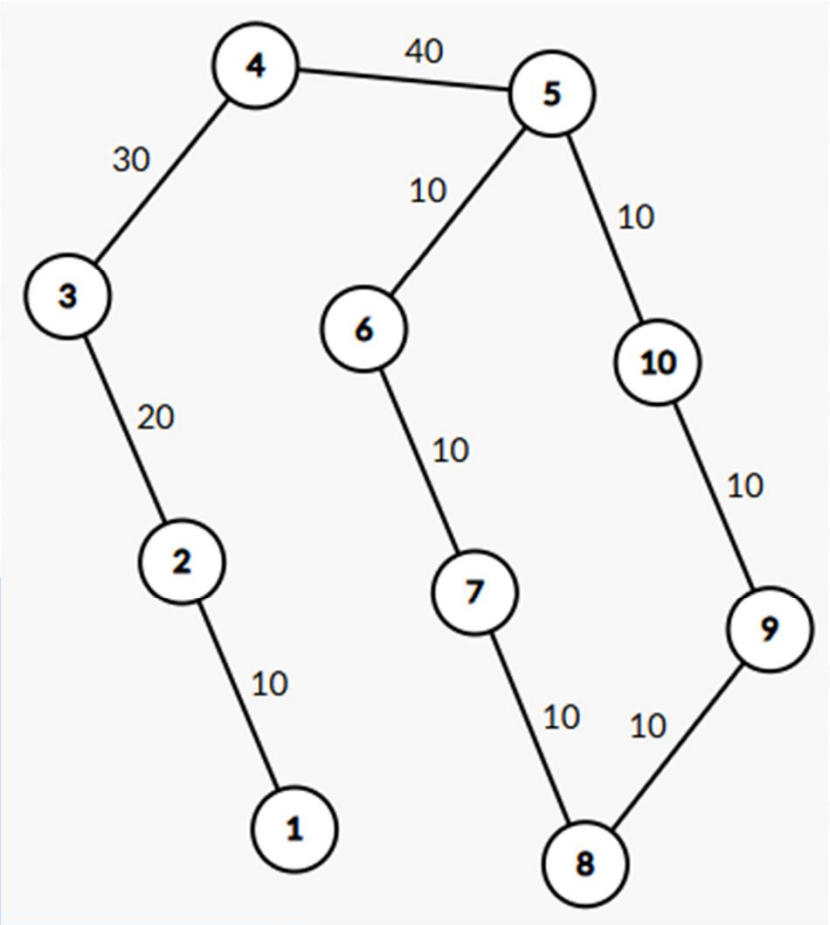
D. 在原图中将边 e 删去后，图中不会产生任何新的割边。

说明当前边 `e` 不存在能与之配对的边 `e'`，使得 `(e, e')` 是割集。
那么很显然 D 是对的。

阅读程序 3

33. 对于如下的图，ans[8] 为？

A. 6900　　B. 9500　　C. 10400　　D. 16900



直接按照给出的公式计算即可。

包含桥的贡献为：

```
100 * 160 - (10000 + 3000) / 2 = 9500
```

不包含桥的贡献为：

```
10 * 50 + 10 * 30 + 10 * 10 = 900
```

因此：ans[8] = 9500 + 900 = 10400

选 C。

完善程序 1

```
int qpow(int a, int b) {
    int res = 1;
    while (b) {
        if (b & 1) res = 1ull * res * a % mod;
        ①
        b >>= 1;
    }
    return res;
}
```

34. ① 处应填（ ）。

A. a = 1ull * a * a % mod;
B. a = 1ull * a * b % mod;
C. res = 1ull * a * a % mod;
D. a = 1ull * res * res % mod;

这是典型的快速幂，选 A

完善程序 1

Fun fact:这个题本来是没有提示的，奈何 cz 告诉我没有提示太变态了，于是放了个提示把这个题变成了送分题。

除了第一个空是不用读题的以外，其他的题都需要你对这个题的做法有一个大致的了解。

我们使用线性筛求解，设 $f_i = i^i$，我们在质数处使用快速幂，复杂度 $O\left(\frac{n}{\log n}\log n\right)$。

对于非质数的情况，有 $f_{pq} = f_p^q f_q^p$，不妨设 $p < q$，则有 $p < \sqrt{n}$，因此对 f_p^q，我们只需要使用光速幂即可。具体方法如下，令 $B = \lfloor\sqrt{n}\rfloor$，将 q 表示为 $aB + b$，对于 $1 \le i \le B$ 预处理 f_p^{iB} 与 f_p^i 即可。复杂度 $O(\sqrt{n} \times \sqrt{n})$。

接下来考虑怎么计算 f_q^p。考虑线性筛合并时会在外层循环枚举 q 而在内层循环枚举 p，因此我们要计算的是 $f_q^{pr_1, pr_2, \cdots}$，pr_i 每次的增量就是质数间隔，因此对于 $1 \le i \le \ln\frac{n}{q}$ 处理出 f_q^i 即可，对于单个 q 来说复杂度为 $O\left(\frac{\frac{n}{q}}{\ln\left(\frac{n}{q}\right)} + \ln\frac{n}{q}\right)$，对于所有 q 累加为 $O(n)$。

完善程序 1

```
if (i <= B) {
    bsgs1[i][0] = 1;
    for (int j = 1; j <= B; j++)
        bsgs1[i][j] = 1ull * bsgs1[i][j - 1] * f[i] % mod;
    bsgs2[i][0] = 1;
    for (int j = 1; j <= B; j++)
        ②
}
```

35. ② 处应填（ ）。

A. bsgs2[i][j] = 1ull * bsgs2[i][j - 1] * qpow(i, B) % mod;
B. bsgs2[i][j] = 1ull * bsgs1[i][j - 1] * bsgs2[i][B - 1] % mod;
C. bsgs2[i][j] = 1ull * bsgs1[i][j - 1] * qpow(f[i], B) % mod;
D. bsgs2[i][j] = 1ull * bsgs2[i][j - 1] * bsgs1[i][B] % mod;

这里是在预处理 f_p^{iB}，选 D。

完善程序 1

```
for (int j = 1; j <= len && i * pr[j] <= n; j++) {
    vis[pr[j] * i] = 1;
    int now = pr[j] - pr[j - 1];
    ③
        powers[ex] = 1ull * powers[ex - 1] * f[i] % mod;
    gap = max(gap, now);
    cur = 1ull * cur * powers[now] % mod;
    ④
    if (⑤)
        break;
}
```

这里我们是在计算 f_q^i，从老的最大质数间隔计算到新的质数间隔即可。注意到这里为了保证复杂度正确，我们不能选 A 而要选 B。

36. ③ 处应填（ ）。

A. for(int ex = 1; ex <= now; ex++)

B. for(int ex = gap + 1; ex <= now; ex++)

C. for(int ex = 1; ex <= now; ex += B)

D. for(int ex = gap + 1; ex <= now; ex += B)

完善程序 1

```
for (int j = 1; j <= len && i * pr[j] <= n; j++) {
    vis[pr[j] * i] = 1;
    int now = pr[j] - pr[j - 1];
    ③
        powers[ex] = 1ull * powers[ex - 1] * f[i] % mod;
    gap = max(gap, now);
    cur = 1ull * cur * powers[now] % mod;
    ④
    if (⑤)
        break;
}
```

37. ④ 处应填（ ）。

A. f[pr[j] * i] = 1ull * bsgs2[pr[j] * i][i % B] * bsgs1[pr[j] * i][i / B] % mod * cur % mod;

B. f[pr[j] * i] = 1ull * bsgs1[pr[j]][i % B] * bsgs2[pr[j]][i / B] % mod * cur % mod;

C. f[pr[j] * i] = 1ull * bsgs2[pr[j]][i % B] * bsgs1[pr[j]][i / B] % mod * cur % mod;

D. f[pr[j] * i] = 1ull * bsgs1[pr[j] * i][i % B] * bsgs2[pr[j] * i][i / B] % mod * cur % mod;

bsgs1[pr[j]][i % B] * bsgs2[pr[j]][i / B] % mod 就是 f_p^q，cur 就是 f_q^p。

选 B。

完善程序 1

```
for (int j = 1; j <= len && i * pr[j] <= n; j++) {
    vis[pr[j] * i] = 1;
    int now = pr[j] - pr[j - 1];
    ③
        powers[ex] = 1ull * powers[ex - 1] * f[i] % mod;
    gap = max(gap, now);
    cur = 1ull * cur * powers[now] % mod;
    ④
    if (⑤)
        break;
}
```

38. ⑤ 处应填（ ）。

A. pr[j] > i

B. !(pr[j] % i)

C. !(i % pr[j])

D. i > pr[j]

考察线性筛相关知识，这没啥好说的，选 C。

不会的同学可以去复习一下线性筛。

完善程序 2 の一些闲聊……

去年 SCP 完以后，我就萌发了一个想要在初赛题里面塞 toptree 的想法，今年出题群一建我就和 cz 说想出 toptree，cz 也是立刻否决了这个想法，但是经过我和这个题的出题人蒸了几下，加上 2021 年 CSP-S 出过四毛子的前车之鉴，最终把这个超级削弱版的 toptree 给大家端了上来。

实际上这个题的难度并不大，只要大家不被题面吓晕了认真做，都是能做出来的。

当然叠个甲，我没有参与这个题的出题工作，所以不要骂我谢谢。

希望这个题能够调动大家对 toptree 的热情，有兴趣的同学可以找饭神（_Ch1F4N_ - 个人中心 - 洛谷）（不是出题人）学习一下 toptree!

完善 2 的大部分课件内容抄录自饭神的课件，感谢饭神。

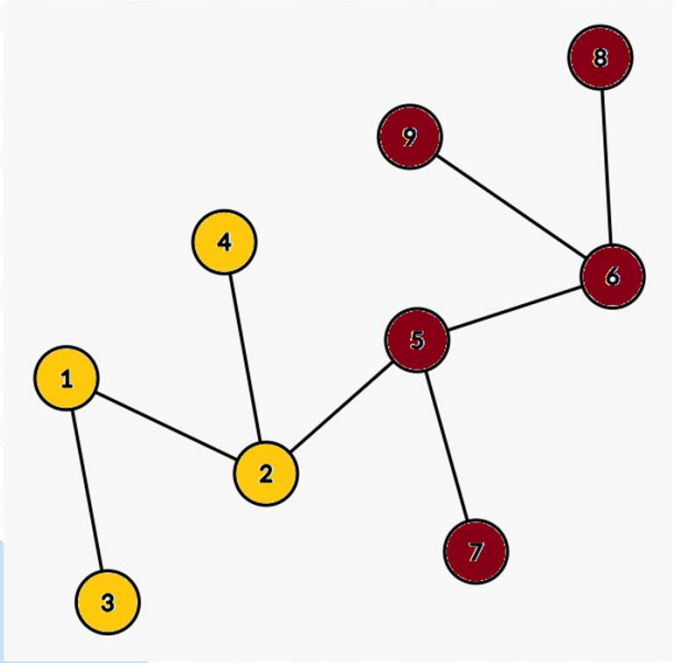
我们使用静态 Top tree 的目的是?

维护树上信息，下文将从设计树上数据结构的角度出发一步步讲解 Top tree

为了简单，下文所有 Top tree 都指静态 Top tree

我们需要一个划分树的方式

我们考虑将树划分成一个个联通子图，具体而言



图中所有红色节点与黄色节点
分别构成一张联通子图

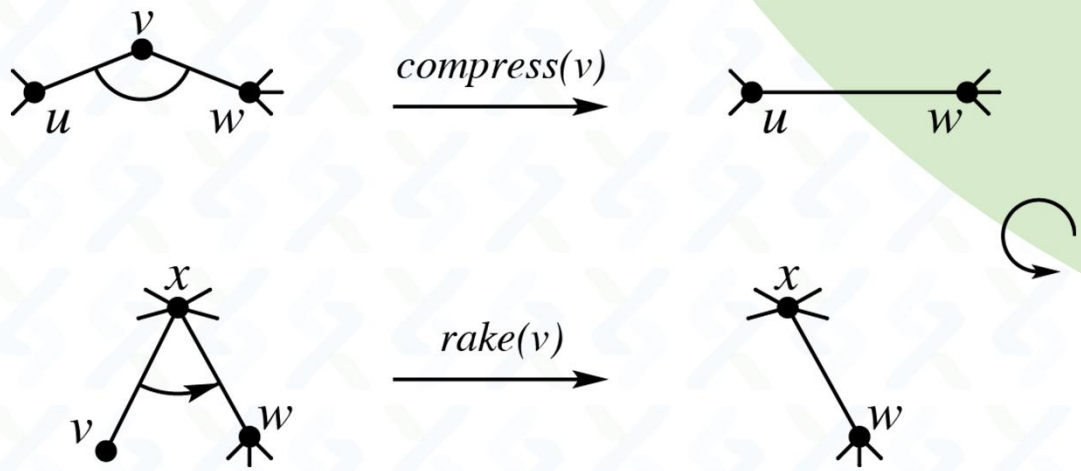
我们需要一个表示方法

怎么表示划分出来的联通子图呢?

我们不妨取联通子图的两点的 u,v 满足以 u 作为根时 v 是一个叶子，用一条 u,v 之间的边表示整个联通子图，下文会称这两个点是簇的两个界点

我们需要一个合并联通子图的方法

直接找到合适划分是困难的，合并却是简单的，考虑合并两个联通子图（两条边）的方法



这个图貌似已经包浆了，如图，描述了两种合并两条边的过程
合并后新的边代表了合并前两个联通子图的并

给出 Top tree 一些定义

上文所说的联通子图下文称为簇

Top tree 描述的正是簇的合并结构

更具体地，Top tree 是一棵完全二叉树，每个点代表一个簇，根代表整棵树，叶子代表原始树上所有边以及其两个端点构成的联通子图

问题1

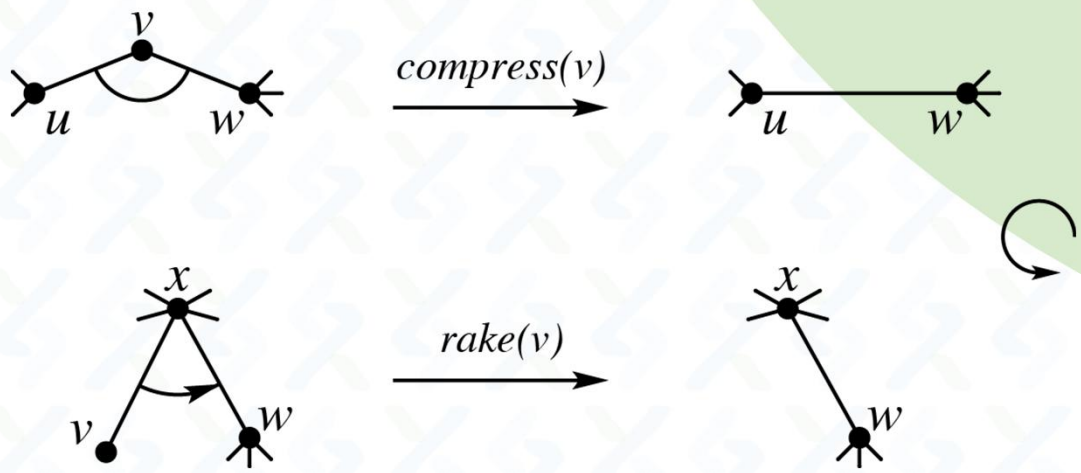
在合并方法中我们发现，每次合并的两个簇都有一个公共点，那么这两个簇的点集并不构成对合并后新簇点集的一个划分（因为有重复），这是不好的，怎么办呢？

我们是在划分边集

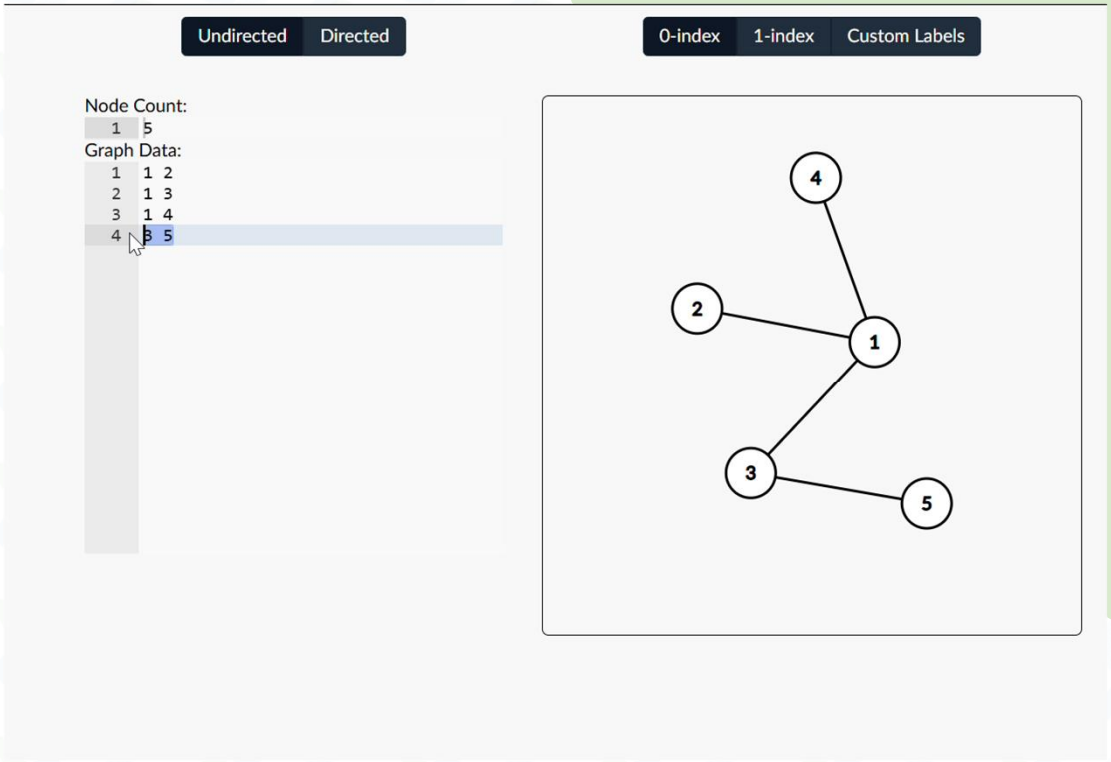
事实上，簇这个结构中只存储其中所有边的信息，而不是点，那么就做到了不重复的划分，所以说，Top tree 也描述的是原始树上所有边一步步合并的过程

形象的描述边合并的过程

还是这张图，我们考虑每次将原树上一组形如下图的两条边用合并后的边**代替其出现在原树上**



合并过程演示

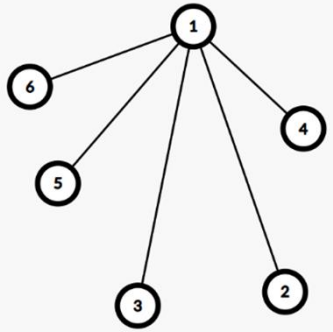


我们先对边 (1,3) 与 (3,5) 执行了 compress 操作，合并为 (1,5) 然后依次对边 (1,5) (1,4) 与边 (1,5) (1,2) 执行 rake 操作

我们需要找到一个良好的合并顺序

我们希望 Top tree 的树高比较矮，该如何办呢？下面直接给出一种使得树高不超过 O(log n) 的合并顺序

先对原树做重链剖分，我们自下往上，对于一条重链，我们假设重链上每个点其轻儿子树内所有边都被合并为轻儿子向父亲的边



右图中 2 是 1 的重儿子，也就是说 3,4,5,6 的子树内所有边都被合并到了边 (1,3) (1,4) (1,5) (1,6) 上，此时这些边都代表一个簇

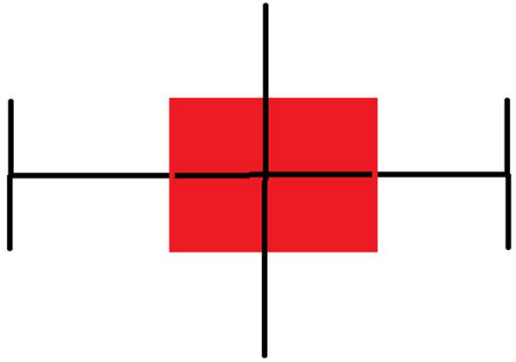
我们需要找到一个良好的合并顺序

考虑将簇 (1,2) (1,3) (1,4) (1,5) (1,6) 合并，合并过程是我们先将所有簇写在一个序列上，再按照簇的大小（簇内边数量）为权值，每次选取带权中点（最大的权值之和不超过总权值一半的一个前缀位置），将序列断成两半，分治处理，显然分治处理后两半都会收缩成一个簇，我们再将这两个簇用 rake 的方法合并，合并完这些簇后再将这些摆在原来边 (1,2) 的位置上，由于 rake 两个簇时两个簇的顺序无所谓，所以可以决定产生的新簇在原来哪个簇的位置上

簇的位置是什么？前面说过每个簇代表一条边，合并到这一部分时边在原树上的位置就是此时簇的位置

我们需要找到一个良好的合并顺序

这样子合并树高会很低吗？为了方便地解释，下文称按照带权中点划分形成的结构叫做分治树



注意到至多一个簇跨过了带权中点，假若这个簇不是第一个簇，就会将这个簇划给右边，那么左边合并得到的簇在这一次合并后大小翻倍

我们需要找到一个良好的合并顺序

对于右边，再次找到带权中点划分后，之前那个横跨带权中点的簇因为成为右边第一个簇，故在再次划分后一定在左边，那么右边的部分合并之后得到的簇大小一定不超过上一页 ppt 中描述的那次合并后得到的簇的大小的一半

对于右边再次划分后的左边部分，要么合并后形成的簇其大小也不超过上一页 ppt 描述的合并后得到的簇的大小的一半，要么左边部分就是分治树的叶子，因为假若不是后者的情况，左边部分大小就不会超过右边部分，那么两个部分加起来就大于等于上一页 ppt 后形成的簇的大小，这会导致上一页 ppt 中的左边部分大小为 0，这是不合法的

我们需要找到一个良好的合并顺序

假若上上页 ppt 中描述的横跨分治中心的簇是第一个簇，就会把其单独划分到右边作为分治树的一个叶子

通过上面的文字，我们可以得出，分治树上任意一个非叶子节点代表的簇，跳两次父亲后所代表簇大小至少翻倍，因此分治树的树高是 log (分治树根簇大小/分治树叶子簇大小) + O(1) 的，下文为了与重链上的分治树区别，把现在的分治树称为 rake 树

我们需要找到一个良好的合并顺序

很好，对于我们当前考虑的重链上每个点只剩下了去往其重儿子的边，如果你学过全局平衡二叉树的话，你应该知道我们接下来要干什么了，考虑对重链上所有边按照其大小为权值做带权分治，每次分治后左边右边合并得到的簇用 compress 操作合并为一个簇，我们称此处的分治树为 compress 树，相仿的分析，compress 树的树高也是 log (分治树根簇大小/分治树叶子簇大小) + O(1)

我们需要找到一个良好的合并顺序

显然每条边只会在跳轻边时经过一次 rake 树，只会在跳完整个重链时经过一次 compress 树，两者均至多 O(log n) 次，故 Top tree 的总树高是 O(log n) 的

回到本题

本题的树保证了是一颗毛毛虫，也就是说，我们只需要先将所有主链上的点上挂着的边 rake 起来，再对整条重链 compress 起来就行了，不必像标准的静态 toptree 一样做重链剖分。

稍微来解释一下程序里面的变量都存的是什么。

每一个 Node 都是一个簇，w 存的是上下界点的距离，l 存的是上界点到簇内点的最大距离，r 存的是下界点到簇内点的距离，d 存的是这个簇的直径，c 存的是与外界相连的界点数量。

R 把一个只有单边界的分支簇 b 挂到已有簇 a 的左端或右端

C 把左右相邻的两个双边界簇压成一个更长的双边界簇。

稍微喘口气，我们来看题目。以及等会儿依旧不放代码，请同学自行观看题面。

完善程序 2

39. ① 处应填（ ）。

A. (a.c == 1 || s == 1 ? a.l : a.r) + b.l

B. (a.c == 1 ? max(a.d, b.d) : (s == 0 ? a.l : a.r)) + b.l

C. (a.c == 1 || s == 0 ? a.l : a.w) + b.l

D. (a.c == 1 || s == 0 ? a.l : a.r) + b.l

第一空就放了个送命题，我们来仔细看一下。

右上图是等待合并的两个簇。

相信大家做这个题的时候都对 s 的含义有很大的疑惑，这里来解释一下。

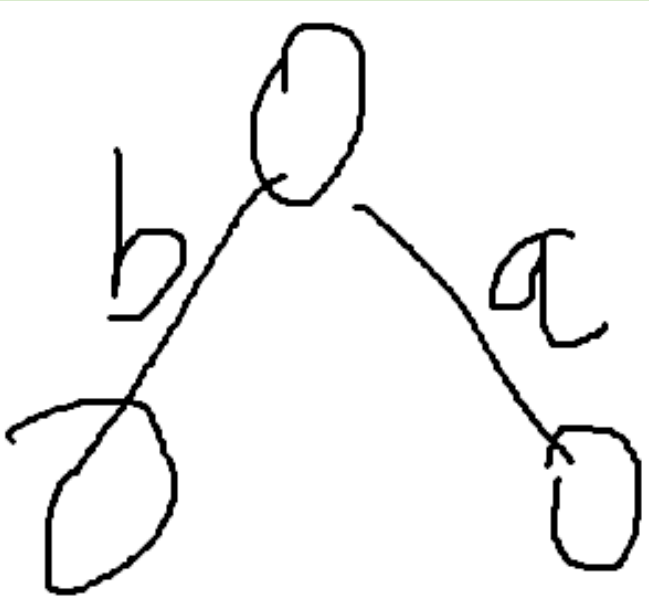
s=0：顶上那个点同时是簇 a 和簇 b 的上界点，左边的点是 b 的下界点，右边的点是 a 的下界点。

s=1：顶上那个点同时是簇 a 的下界点和簇 b 的上界点，左边的点是 b 的下界点，右边的点是 a 的上界点。

这样就可以完美解释右下图的两行代码了。

我们回到 39 题。当 a 只有一个边界时左右含义相同，使用 a.l；当 a 有两个边界且 s==0 时，b 挂在左端，使用 a.l；s==1 时挂在右端，使用 a.r。故跨簇直径为 (a.c == 1 || s == 0 ? a.l : a.r) + b.l

选 D。




```
} else if (s == 0) {
    z.l = max(a.l, b.l); z.r = max(a.r, a.w + b.l);
} else {
    z.l = max(a.l, a.w + b.l); z.r = max(a.r, b.l);
```

完善程序 2

40. ② 处应填（ ）。

A. y = merge(y, star(x), 'R', 0);

B. y = merge(y, star(x), 'R', 1);

C. y = merge(star(x), y, 'R', 0);

D. y = merge(y, star(x), 'C');

make(x) 构造的是从主链点 x 到 x+1 的边簇。path(a[x]) 表示主链边 (x,x+1)，star(x) 是挂在顶点 x 的叶簇，题面明确说“把分支簇挂到对应主链边的左端”，因此填 y = merge(y, star(x), 'R', 0);

选 A。

完善程序 2

41.③ 处应填（ ）。

A. merge(q, p, 'C')

B. merge(p, q, 'R', 0)

C. merge(p, q, 'R', 1)

D. merge(p, q, 'C')

线段树左儿子的区间在前、右儿子的区间在后，二者各有两个边界，我们按顺序 compress

选 D。

完善程序 2

42.④ 处应填（ ）。

A. bel[x] < n

B. x < n

C. bel[x] <= n

D. st[bel[x]] != 0

顶点 v<n 的叶簇被挂在主链边 (v,v+1) 的线段树叶子中，修改后要调用 change 重算这个位置到根的信息。顶点 n 没有右侧主链边，其 star(n) 在查询时由第 91 行直接追加；第 70 行更新 f 后查询自然能读到新值，无需改线段树。因此条件是 bel[x]<n

选 A。

完善程序 2

43.⑤ 处应填（ ）。

A. x == y ? leaf(0) : query(1, 1, n - 1, x, y - 1)

B. x == y ? star(x) : query(1, 1, n - 1, x, y)

C. x == y ? star(x) : query(1, 1, n - 1, x, y - 1)

D. x == y ? star(x) : query(1, 1, n - 1, x + 1, y - 1)

当 x<y 时，应查询主链边 $x, x+1, \ldots, y-1$。make(i) 已把 star(i) 挂在主链边 i 的左端，因此查询结果已经包含顶点 $x, \ldots, y-1$ 的全部叶簇；还缺右端点 y 的叶簇，随后第 91 行用 s=1 把 star(y) 挂到右端。

当 x=y 时，区间没有主链边，只包含一个主链顶点及其叶子，直接取 star(x)。故填 x == y ? star(x) : query(1, 1, n - 1, x, y - 1)

选 C。

吐槽环节

有什么想吐槽的喵？

洛谷

谢谢大家

Thank you all for listening!