ICPC Template Manual

作者: 贺梦杰

August 24, 2019

Contents

Chapter 1

基础

1.1 测试

Chapter 2

搜索

Chapter 3

动态规划

3.1 树形动态规划

3.1.1 加分二叉树

3.1.1.1 问题描述

设一个 n 个节点的二叉树 tree 的中序遍历为（l,2,3,···,n），其中数字 1,2,3,···,n 为节点编号。每个节点都有一个分数（均为正整数），记第 i 个节点的分数为 di，tree 及它的每个子树都有一个加分，任一棵子树 subtree（也包含 tree 本身）的加分计算方法如下：
subtree 的左子树的加分 × subtree 的右子树的加分 + subtree 的根的分数
若某个子树为空，规定其加分为 1，叶子的加分就是叶节点本身的分数。不考虑它的空子树。
试求一棵符合中序遍历为（1,2,3,···,n）且加分最高的二叉树 tree。要求输出；
（1）tree 的最高加分
（2）tree 的前序遍历

3.1.1.2 输入格式

第 1 行：一个整数 n（n < 30），为节点个数。
第 2 行：n 个用空格隔开的整数，为每个节点的分数（分数< 100）。

3.1.1.3 输出格式

第 1 行：一个整数，为最高加分（结果不会超过 4,000,000,000）。
第 2 行：n 个用空格隔开的整数，为该树的前序遍历。

3.1.1.4 思路

这道题看上去是树形 DP，但仔细思考，我们发现很难依据中序遍历建出树。但中序遍历有其独特之处，即一旦根被确定，则左右子树也被确定。
所以我们应该用区间 DP 来解决这道题。
$f(i,j)$: 选 i 到 j 的节点作为一颗子树最大的得分

$$f(i,j) = \max_{i \le k \le j} \{f(i,k-1) * f(k+1,j) + f(k,k)\}$$

由题意，当 $i > j$，$f(i,j) = 1$ 为空节点。
前序遍历就是重新用 dfs 走一遍：每次找到使 $f(i,j)$ 最大的 k，然后以 k 为分界向左右两边递归。

3.1.2 洛谷 P2015 二叉苹果树

3.1.2.1 问题描述

有一棵苹果树，如果树枝有分叉，一定是分 2 叉（就是说没有只有 1 个儿子的结点）这棵树共有 N 个结点（叶子点或者树枝分叉点），编号为 1-N, 树根编号一定是 1。我们用一根树枝两端连接的结点的编号来描述一根树枝的位置。现在这颗树枝条太多了，需要剪枝。但是一些树枝上长有苹果。
给定需要保留的树枝数量，求出最多能留住多少苹果。

3.1.2.2 输入格式

第 1 行 2 个数，N 和 Q(1<=Q<= N,1<N<=100)。
N 表示树的结点数，Q 表示要保留的树枝数量。接下来 N-1 行描述树枝的信息。
每行 3 个整数，前两个是它连接的结点的编号。第 3 个数是这根树枝上苹果的数量。
每根树枝上的苹果不超过 30000 个。

3.1.2.3 输出格式

剩余苹果的最大数量。

3.1.2.4 思路

$f(i,j)$ 表示以 i 为节点的根保留 k 条边的最大值接下来对每一个节点分类讨论，共三种情况：
1. 全选左子树
2. 全选右子树
3. 左右子树都有一部分

为了方便，我们设左儿子为 ls，右儿子为 rs，连接左儿子的边为 le，连接右儿子的边为 re

$$f(i,j)=\max\left\{f(ls,j-1)+le,f(rs,j-1)+re,\max_{0\le k\le j-2}\{f(ls,k)+f(rs,j-2-k)\}+le+re\right\}$$

3.1.3 最大利润

3.1.3.1 问题描述

政府邀请了你在火车站开饭店，但不允许同时在两个相连接的火车站开。任意两个火车站有且只有一条路径，每个火车站最多有 50 个和它相连接的火车站。
告诉你每个火车站的利润，问你可以获得的最大利润为多少。
最佳投资方案是在 1，2，5，6 这 4 个火车站开饭店可以获得利润为 90

3.1.3.2 输入格式

第一行输入整数 N(<=100000)，表示有 N 个火车站，分别用 1，2。。。，N 来编号。接下来 N 行，每行一个整数表示每个站点的利润，接下来 N-1 行描述火车站网络，每行两个整数，表示相连接的两个站点。

3.1.3.3 输出格式

输出一个整数表示可以获得的最大利润。

3.1.3.4 思路

这道题虽然是多叉树，但状态仍然是比较简单的。
对于某个结点，如果选择该节点，则该节点的所有儿子都不能选，如果不选该节点，则它的儿子可选可不选。
所以，我们令 $f(i)$ 表示以 i 节点为根的子树中选 i 的最大利润，$h(i)$ 表示以 i 节点为根的子树中不选 i 的最大利润。a[i] 是 i 本身的利润，j 是 i 的儿子

$$f(i)=a[i]+\sum_j h(j)$$

$$h(i)=\sum_j \max\{f(j),h(j)\}$$

3.1.4 洛谷 P2014 选课

3.1.4.1 问题描述

学校实行学分制。每门的必修课都有固定的学分，同时还必须获得相应的选修课程学分。学校开设了 N（N<300）门的选修课程，每个学生可选课程的数量 M 是给定的。学生选修了这 M 门课并考核通过就能获得相应的学分。在选修课程中，有些课程可以直接选修，有些课程需要一定的基础知识，必须在选了其它的一些课程的基础上才能选修。例如《Frontpage》必须在选修了《Windows 操作基础》之后才能选修。我们称《Windows 操作基础》是《Frontpage》的先修课。每门课的直接先修课最多只有一门。两门课也可能存在相同的先修课。每门课都有一个课号，依次为 1，2，3，···。
你的任务是为自己确定一个选课方案，使得你能得到的学分最多，并且必须满足先修课优先的原则。假定课程之间不存在时间上的冲突。

3.1.4.2 输入格式

输入文件的第一行包括两个整数 N、M（中间用一个空格隔开），其中 $1\le N\le 300, 1\le M\le N$。以下 N 行每行代表一门课。课号依次为 1，2，···，N。每行有两个数（用一个空格隔开），第一个数为这门课先修课的课号（若不存在先修课则该项为 0），第二个数为这门课的学分。学分是不超过 10 的正整数。

3.1.4.3 输出格式

只有一个数：实际所选课程的学分总数。

3.1.4.4 思路一

分析一下，似乎也不是很难，对于某个节点，只有选择它，才能选择它的儿子们。
令 $f(x,i)$ 表示在以 x 结点为根的子树中选择 i 个点的最大学分。$f(x,1)=s[x]$，s[x] 是课程 x 本身的学分。
假设结点 x 有 k 个儿子，标号为 $y_1,y_2,..,y_k$，让他们分别选 $i_1,i_2,..,i_k$ 门课程，那么状态转移方程似乎是..

$$f(x,i)=s[x]+\max_{i_1+i_2+..+i_k=i-1}\sum_{1\le j\le k}f(y_j,i_j)$$

好像.. 写不出这样的循环啊，当然，如果用 dfs 强行写也不是不可以，但是时间复杂度必炸。

这时候我们要用一种类似**前缀和**的思维
即，我们每 dfs 完一棵子树，都进行一次完整的 dp 更新。假设当前根结点为 x，我们刚 dfs 完它的一棵子树 y，那么我们进行如下更新（对所有 i）：

$$f(x,i)=\max_{1\le j\le i}\{f(x,j)+f(y,i-j)\}$$

此时 $f(x,i)$ 表示在以 x 结点为根的子树中已经被 dfs 过的部分中选 i 个点的最大学分，而每次更新就像是把一棵新子树添加到答案中。
除此以外，还有一点要注意，就是我们要让 **i 递减更新**，因为大的 i 要用到之前小的 i，而小的 i 不要用到大的 i

3.1.4.5 思路二

先将**森林转二叉树 (左孩子右兄弟)**。如何转呢？设 D[x]、c[x] 和 b[x] 分别表示 x 结点的父亲、左孩子和右兄弟，对于每个节点，D[x] 是已知的，所以 **b[x]=c[D[x]],c[D[x]]=x**。
再分析，对于某个节点，如果要选其左孩子，则必须选它，而选右孩子则没有限制。
令 $f(x,i)$ 表示在以 x 结点为根的子树中选择 i 个点的最大学分。$f(x,1)=s[x]$，s[x] 是课程 x 本身的学分。
还是沿用上面**前缀和思想**，先由左孩子 (ls) 更新，再由右孩子 (rs) **倒着**更新，两次的方程如下：

$$f(x,i)=f(x,1)+f(ls,i-1)$$

$$f(x,i)=\max_{0\le j\le i}f(x,j)+f(rs,i-j)$$

3.1.5 HYSBZ 2427 软件安装

3.1.5.1 题目描述

现在我们的手头有 N 个软件，对于一个软件 i，它要占用 Wi 的磁盘空间，它的价值为 Vi。我们希望从中选择一些软件安装到一台磁盘容量为 M 的计算机上，使得这些软件的价值尽可能大（即 Vi 的和最大）。
但是现在有个问题：软件之间存在依赖关系，即软件 i 只有在安装了软件 j（包括软件 j 的直接或间接依赖）的情况下才能正确工作（软件 i 依赖软件 j)。幸运的是，一个软件最多依赖另外一个软件。如果一个软件不能正常工作，那么他能够发挥的作用为 0。
我们现在知道了软件之间的依赖关系：软件 i 依赖 Di。现在请你设计出一种方案，安装价值尽量大的软件。一个软件只能被安装一次，如果一个软件没有依赖则 Di=0，这是只要这个软件安装了，它就能正常工作。
subsubsection 输入格式 第 1 行：N,M （0<=N<=100,0<=M<=500）第 2 行：W1,W2, ···Wi, ···,Wn 第 3 行：V1,V2, ···Vi, ···,Vn 第 4 行：D1,D2, ···Di, ···,Dn

3.1.5.2 输出格式

一个整数，代表最大价值。

3.1.5.3 思路

注意，此图可能有环
仔细读题，题中说一个软件只依赖最多一个软件，所以在一个联通分之内最多只有一个环，并且该环可以指向环以外的节点，而环以外的节点不会指向环。
因此，我们可以**把环当做一个新节点**来处理，这个新节点所占空间和价值都为环内元素加和。
怎么确定环及环内元素呢？我推荐**着色法 (有向图为黑白灰，无向图为黑白)**。
有向图中白色为未探索的点，灰色为正在探索的点，黑色为已经探索过的并且确定不成环的点。若在探索过程中遇到灰点，则说明成环。
无向图中白色为未探索的点，黑色为已经探索过和正在探索的点。若在探索过程中遇到黑点，则说明成环。
处理完成后，DP 思路同 *"洛谷 P2014 选课"*

3.1.6 CF486D Valid Sets

3.1.6.1 题目描述

给定一棵树，每个点都有一个权，现在要你选择一个连通块，问有多少种选择方法，使得连通块中的最大点权和最小点权的差值小于等于 d。答案对 1e9+7 取模。

3.1.6.2 输入格式

The first line contains two space-separated integers d (0 d 2000) and n (1 n 2000).
The second line contains n space-separated positive integers a1, a2, ..., an(1 ai 2000).
Then the next n - 1 line each contain pair of integers u and v (1 u, v n) denoting that there is an edge between u and v. It is guaranteed that these edges form a tree.

3.1.6.3 输出格式

Print the number of valid sets modulo 1000000007.

3.1.6.4 思路

我起初的错误想法是，设 $f(x,i)$ 和 $g(x,i)$ 分别为以 x 为根的子树中最大值和最小值为 i 的方案数，企图通过对 g 和 f 的运算得出答案，但这是错误的，主要原因是 f 和 g 无法把范围限定住。
受到前面以子树为对象思考的影响，形成了**思维惯性**，这里我们并不是以子树为对象，在一遍 dfs 里算出所有答案。
而是**枚举**，枚举每一点 x，将 x 作为连通块的最大值，以 x 为根进行 dfs，求方案数。
另外要注意：因为点权可能相同，所以为了避免重复计算，我们需要定序，若权值相同，让编号大的点访问编号小的，而编号小的不能访问大的。

3.1.7 CF294E Shaass the Great

3.1.7.1 题目描述

给一颗带边权的树，让你选一条边删除，然后在得到的两个子树中各选一个点，用原来被删除的边连起来，重新拼成一棵树。使得这棵树的所有点对的距离总和最小。

3.1.7.2 输入格式

The first line of the input contains an integer n denoting the number of cities in the empire, $(2 \leq n \leq 5000)$. The next n - 1 lines each contains three integers ai, bi and wi showing that two cities ai and bi are connected using a road of length wi, $(1 \leq a_i, b_i \leq n, a_i \neq b_i, 1 \leq w_i \leq 106)$.

3.1.7.3 输出格式

所有点对距离总和最优解。

3.1.7.4 思路

设我们要删除的边为 e，以 e 为分界树被分割成了左右两个部分 (我们称为左树和右树)，最后我们连接的两个点为 vl 和 vr
则答案为:
左树内点对距离总和 + 右树内点对距离总和
+ 左树中所有点到 vl 的距离总和 * 右树中点的个数 + 右树中所有点到 vr 的距离总和 * 左树中点的个数
+ e 的权重 * 左树中点的个数 * 右树中点的个数

再观察一下，其实在删除 e 的情况下左右树内点对距离总和、左右树中的点的个数、e 的权重都是不变的，变化的只有左右树中所有点到 vl 和 vr 的距离总和
由此我们知道了，在删除 e 的情况下，答案最优的条件即为**找到 vl 和 vr，使得左树中所有点到 vl 的距离总和最小，右树中所有点到 vr 的距离总和最小**
于是我们知道了，要得到答案，就是要求一棵树的三个值：树内点的个数、树上所有点到某一点的距离总和的最小值、树内所有点对距离总和
树内点的个数最简单，不多说了，而树内所有点对的距离和也可以由树上所有点到某一点的距离和算得 (即树上所有点到**每**一点的和除以 2)

下面主要考虑如何在 $O(n)$ 的时间内求出树上所有点到每一点的距离：

考虑 dfs，结果发现一遍 dfs 无论如何都不可能得到，但是可以知道以某一点 x 为根的子树上所有点到根 x 的距离总和
在第一遍 dfs 的基础上，我们再进行一次 dfs，这次 dfs 我们不再是自底向上更新，而是自顶向下更新，更新父节点及其以上所有的点到 x 的距离总和
可以理解为，第一次 dfs 我们得到了 x 以下的所有点到 x 的距离总和，第二次 dfs 我们得到了 x 以上的所有点到 x 的距离总和

对于第二次 dfs，例如我们现在在 a 节点上，即将去往 x 结点，我们需要下传给 x 节点的距离有哪些呢？
1.a 结点上面传下来的距离
2.a 结点上除了 x 分支所有旁路上的距离 (由第一次 dfs 可以算出)
3.a 与 x 的边权

所以最后的算法是：**枚举每一条边**，运用以上算法，求出最小值

Chapter 4

字符串

TODO 字符串 Hash

4.1 字符串 Hash

```
#define ull unsigned long long
#define P 131
ull f[N], p[N];
void Init()
{
    p[0] = 1, f[0] = 0;
    for (int i = 1; i <= n; i++)
    {
        p[i] = p[i - 1] * P;
        f[i] = f[i - 1] * P + str[i];
    }
}
ull Hash(int left, int right)
{
    return f[right] - f[left - 1] * p[right - left + 1];
}
```

4.1.1 应用：最长回文子串

枚举回文子串的中心位置，求解从中心位置出发向左右两侧最长可扩展出多长的回文串，分奇、偶回文子串，二分长度。

```
#define ull unsigned long long
#define P 131
const int N = 1e6 + 50;
ull pre[N], suf[N], p[N];
string str;
int n;
void Init()
{
    pre[0] = suf[0] = 0;
    for (int i = 1; i <= n; i++)
    {
        pre[i] = pre[i - 1] * P + str[i - 1];
        suf[i] = suf[i - 1] * P + str[n - i];
    }
}
ull Hash(int left, int right)
{
    return pre[right] - pre[left - 1] * p[right - left + 1];
}
ull HashReverse(int left, int right)
{
    //WA 这里注意推导
    return suf[n - left + 1] - suf[n - right] * p[right - left + 1];
}
#define ODD 1
#define EVEN 2
bool Judge(int pos, int len, int flag)
{
    if (flag == ODD)
    {
        if (pos - len < 1 || pos > n || pos < 1 || pos + len > n)
            return false;
        return Hash(pos - len, pos) == HashReverse(pos, pos + len);
    }
    else
    {
        if (pos - len < 1 || pos - 1 > n || pos < 1 || pos + len - 1 > n)
            return false;
        return Hash(pos - len, pos - 1) == HashReverse(pos, pos + len - 1);
    }
}
```

```cpp
int Solve(int pos)
{
    int ans = 1;
    //奇数
    int left = 0;
    int right = N;
    int mid;
    while (left < right)
    {
        mid = (left + right + 1) >> 1;
        if (Judge(pos, mid, ODD))
            left = mid;
        else
            right = mid - 1;
    }
    ans = max(ans, 2 * left + 1);
    //偶数
    left = 0;
    right = N;
    while (left < right)
    {
        mid = (left + right + 1) >> 1;
        if (Judge(pos, mid, EVEN))
            left = mid;
        else
            right = mid - 1;
    }
    ans = max(ans, 2 * left);
    return ans;
}
```

4.1.2 应用：后缀数组

```cpp
#define ull unsigned long long
#define P 131
const int N = 3e5 + 50;
ull f[N], p[N];
char str[N];
int SA[N], n, Height[N];
void Init()
{
    p[0] = 1, f[0] = 0;
    for (int i = 1; i <= n; i++)
    {
        p[i] = p[i - 1] * P;
        f[i] = f[i - 1] * P + str[i];
    }
}
ull Hash(int left, int right)
{
    return f[right] - f[left - 1] * p[right - left + 1];
}
// k:[0,n) 表示后缀S(k,n-1)
// 最长公共前缀
int LCP(int a, int b)
{
    int left = 0;
    int right = N;
    int mid;
    while (left < right)
    {
        mid = (left + right + 1) >> 1;
        if (a + mid - 1 <= n && b + mid - 1 <= n && Hash(a, a + mid - 1) == Hash(b, b + mid - 1))
            left = mid;
        else
```

```
            right = mid - 1;
    }
    return left;
}
bool cmp(int a, int b)
{
    int len = LCP(a, b);
    return str[a + len] < str[b + len];
}
void calc_height()
{
    Height[1] = 0;
    for (int i = 2; i <= n; i++)
        Height[i] = LCP(SA[i], SA[i - 1]);
}
int main()
{
    scanf("%s", str + 1);
    n = strlen(str + 1);
    for (int i = 1; i <= n; i++)
        SA[i] = i;
    Init();
    sort(SA + 1, SA + n + 1, cmp);
    calc_height();
}
```

4.1.3 应用：二维 Hash

给定一个 M 行 N 列的 01 矩阵（只包含数字 0 或 1 的矩阵），再执行 Q 次询问，每次询问给出一个 A 行 B 列的 01 矩阵，求该矩阵是否在原矩阵中出现过。

做法：选取两个不同的 P 值分别对行列进行 Hash 处理，应用二维前缀和求取矩阵 Hash 值。

```
#define P 131
#define Q 13331
#define ull unsigned long long
void Init()
{
    char ch;
    for (int i = 1; i <= m; i++)
        for (int j = 1; j <= n; j++)
            cin >> ch, Hash[i][j] = Hash[i][j - 1] * P + ch;
    for (int i = 1; i <= m; i++)
        for (int j = 1; j <= n; j++)
            Hash[i][j] += Hash[i - 1][j] * Q;
}
ull temp = Hash[i][j] - Hash[i - a][j] * q[a] - Hash[i][j - b] * p[b] + Hash[i - a][j - b] * q[a] * p[b];
```

4.1.4 应用：一类同构判定的问题

参考：杨弋《Hash 在信息学竞赛中的一类应用》

4.2 后缀自动机

```cpp
const int MAXLEN = 1e6 + 50;
namespace SAM
{
    struct state
    {
        int len, link, next[26];
    } st[MAXLEN * 2];
    int sz, last;
    void SAM_Init()
    {
        st[0].len = 0, st[0].link = -1;
        sz++, last = 0;
    }
    void SAM_Extend(int c)
    {
        int cur = sz++;
        st[cur].len = st[last].len + 1;
        int p = last;
        while (p != -1 && !st[p].next[c])
        {
            st[p].next[c] = cur;
            p = st[p].link;
        }
        if (p == -1)
            st[cur].link = 0;
        else
        {
            int q = st[p].next[c];
            if (st[q].len == st[p].len + 1)
                st[cur].link = q;
            else
            {
                int clone = sz++;
                st[clone].len = st[p].len + 1;
                memcpy(st[clone].next, st[q].next, sizeof(st[clone].next));
                st[clone].link = st[q].link;
                while (p != -1 && st[p].next[c] == q)
                {
                    st[p].next[c] = clone;
                    p = st[p].link;
                }
                st[q].link = st[cur].link = clone;
            }
        }
        last = cur;
    }
    int id[MAXLEN * 2], c[MAXLEN * 2];
    void Topo()
    {
        //计数排序
        for (int i = 1; i < sz; i++)
            c[st[i].len]++;
        for (int i = 1; i < MAXLEN; i++)
            c[i] += c[i - 1];
        for (int i = 1; i < sz; i++)
            id[c[st[i].len]--] = i;
    }
} // namespace SAM
```

4.3 KMP

```
// KMP
next[1] = 0;
for (int i = 2, j = 0; i <= n; i++) {
    while (j > 0 && a[i] != a[j+1]) j = next[j];
    if (a[i] == a[j+1]) j++;
    next[i] = j;
}

for (int i = 1, j = 0; i <= m; i++) {
    while (j > 0 && (j == n || b[i] != a[j+1])) j = next[j];
    if (b[i] == a[j+1]) j++;
    f[i] = j;
    // if (f[i] == n)，此时就是A在B中的某一次出现
}
```

字符串循环元可利用 next 数组求解。

4.4 最小表示法

```
// 最小表示法
int n = strlen(s + 1);
for (int i = 1; i <= n; i++) s[n+i] = s[i];
int i = 1, j = 2, k;
while (i <= n && j <= n) {
    for (k = 0; k < n && s[i+k] == s[j+k]; k++);
    if (k == n) break; // s likes "aaaaa"
    if (s[i+k] > s[j+k]) {
        i = i + k + 1;
        if (i == j) i++;
    } else {
        j = j + k + 1;
        if (i == j) j++;
    }
}
ans = min(i, j);
```

Chapter 5

数据结构

5.1 并查集

```
#define MAX 1010
struct node
{
    int par;
    //int rank;
    //路径压缩后 rank=1或2 rank失去了意义
    int data;
};
node ns[MAX];
void Init()
{
    for (int i = 1; i < MAX; i++)
    {
        ns[i].par = i;
    }
}
int Find(int i)
{
    if (ns[i].par == i)
    {
        //返回根结点
        return i;
    }
    ns[i].par = Find(ns[i].par);
    //路径压缩
    return ns[i].par;
}
void Union(int i, int j)
{
    int pi = Find(i);
    int pj = Find(j);
    if (pi != pj)
    {
        ns[pi].par = pj;
    }
}
```

5.2 树状数组

推荐阅读：*https://www.cnblogs.com/RabbitHu/p/BIT.html*

5.2.1 单点修改，区间查询

```
#define N 1000100
long long c[N];
int n,q;
int lowbit(int x)
{
    return x&(-x);
}
void change(int x,int v)
{
    while(x<=n)
    {
        c[x]+=v;
        x+=lowbit(x);
    }
}
long long getsum(int x)
{
    long long ans=0;
    while(x>=1)
    {
        ans+=c[x];
        x-=lowbit(x);
    }
    return ans;
}
```

例题：*https://loj.ac/problem/130*

5.2.2 区间修改，单点查询

引入差分数组来解决树状数组的区间更新

```
//初始化
change(i,cur-pre);
//区间修改
change(l,x);
change(r+1,-x);
//单点查询
getsum(x)
```

例题：*https://loj.ac/problem/131*

5.2.3 区间修改，区间查询

```
//初始化
change(c1,i,cur-pre);
change(c2,i,i*(cur-pre));
//为什么这么写？ 你需要写一下前缀和的表达式
//区间修改
change(c1,l,x);
change(c2,l,l*x);
change(c1,r+1,-x);
change(c2,r+1,-(r+1)*x);
//区间查询
temp1=l*getsum(c1,l-1)-getsum(c2,l-1);
temp2=(r+1)*getsum(c1,r)-getsum(c2,r);
ans=temp2-temp1
```

例题：*https://loj.ac/problem/132*

5.3 二维树状数组

5.3.1 单点修改，区间查询

```
#define N 5050
long long tree[N][N];
long long n,m;
long long lowbit(long long x)
{
    return x&(-x);
}
void change(long long x,long long y,long long val)
{
    long long init_y=y;
    //这里注意n,m的限制
    while(x<=n)
    {
        y=init_y;
        while(y<=m)
        {
            tree[x][y]+=val;
            y+=lowbit(y);
        }
        x+=lowbit(x);
    }
}
long long getsum(long long x,long long y)
{
    long long ans=0;
    long long init_y=y;
    while(x>=1)
    {
        y=init_y;
        while(y>=1)
        {
            ans+=tree[x][y];
            y-=lowbit(y);
        }
        x-=lowbit(x);
    }
    //这里画图理解
    return ans;
}
//初始化
change(x,y,k);
//二维前缀和
ans = getsum(c,d)+getsum(a-1,b-1)-getsum(a-1,d)-getsum(c,b-1);
```

例题：*https://loj.ac/problem/133*

5.3.2 区间修改，区间查询

```
#define N 2050
long long t1[N][N];
long long t2[N][N];
long long t3[N][N];
long long t4[N][N];
long long n,m;
long long lowbit(long long x)
{
    return x&(-x);
}
long long getsum(long long x,long long y)
{
    long long ans=0;
    long long init_y=y;
```

```
    long long init_x=x;
    while(x>=1)
    {
        y=init_y;
        while(y>=1)
        {
            ans+=(init_x+1)*(init_y+1)*t1[x][y];
            ans-=(init_y+1)*t2[x][y];
            ans-=(init_x+1)*t3[x][y];
            ans+=t4[x][y];
            y-=lowbit(y);
        }
        x-=lowbit(x);
    }
    return ans;
}
void change(long long x,long long y,long long val)
{
    long long init_x=x;
    long long init_y=y;
    while(x<=n)
    {
        y=init_y;
        while(y<=m)
        {
            t1[x][y]+=val;
            t2[x][y]+=init_x*val;
            t3[x][y]+=init_y*val;
            t4[x][y]+=init_x*init_y*val;
            y+=lowbit(y);
        }
        x+=lowbit(x);
    }
}
//区间修改
change(c+1,d+1,x);
change(a,b,x);
change(a,d+1,-x);
change(c+1,b,-x);
//区间查询
ans=getsum(c,d)+getsum(a-1,b-1)-getsum(c,b-1)-getsum(a-1,d);
```

例题：*https://loj.ac/problem/135*

5.4 线段树

5.4.1 基础操作

```
const int N = 1e5 + 10;
#define ls(a) (a << 1)
#define rs(a) (a << 1 | 1)
struct node
{
    int val;
    int lazy;
};
node tree[N << 2];
int a[N];
void PushUp(int rt)
{
    tree[rt].val = tree[ls(rt)].val + tree[rs(rt)];
}
void PushDown(int ls, int rs, int rt)
{
    tree[ls(rt)].val += ls * tree[rt].lazy;
    tree[rs(rt)].val += rs * tree[rt].lazy;
    tree[ls(rt)].lazy += tree[rt].lazy;
    tree[rs(rt)].lazy += tree[rt].lazy;
    tree[rt].lazy = 0;
}
void Build(int left, int right, int rt)
{
    if (left == right)
    {
        tree[rt].val = a[left];
        return;
    }
    int mid = (left + right) >> 1;
    Build(left, mid, ls(rt));
    Build(mid + 1, right, rs(rt));
    PushUp(rt);
    //向上更新
}
```

5.4.2 单点更新

```
void Update(int left, int right, int rt, int pos, int val)
{
    if (left == right && left == pos)
    {
        tree[rt].val += val;
        return;
    }
    int mid = (left + right) >> 1;
    if (tree[rt].lazy)
    {
        PushDown(mid - left + 1, right - mid, rt);
    }
    if (mid >= pos)
        Update(left, mid, ls(rt), pos, val);
    else if (pos > mid)
        Update(mid + 1, right, rs(rt), pos, val);
    PushUp(rt);
}
```

例题：*https://www.luogu.org/problemnew/show/P3372*

5.4.3 区间更新

```
void Update(int left, int right, int rt, int s, int t, int val)
{
    if (left >= s && right <= t)
    {
        tree[rt].val += (right - left + 1) * val;
        tree[rt].lazy += val;
        return;
    }
    int mid = (left + right) >> 1;
    if (tree[rt].lazy)
    {
        PushDown(mid - left + 1, right - mid, rt);
    }
    if (mid < s)
        Update(mid + 1, right, rs(rt), s, t, val);
    else if (mid >= t)
        Update(left, mid, ls(rt), s, t, val);
    else
    {
        Update(left, mid, ls(rt), s, t, val);
        Update(mid + 1, right, rs(rt), s, t, val);
    }
    PushUp(rt);
}
```

5.4.4 区间查询

```
void Query(int left, int right, int s, int t, int rt)
{
    if (left >= s && right <= t)
    {
        return tree[rt].val;
    }
    int mid = (left + right) >> 1;
    if (tree[rt].lazy)
        PushDown(mid - left + 1, right - mid, rt);
    long long sum = 0;
    if (mid < s)
        sum += Query(mid + 1, right, rs(rt), s, t, val);
    else if (mid >= t)
        sum += Query(left, mid, ls(rt), s, t, val);
    else
    {
        sum += Query(left, mid, ls(rt), s, t, val);
        sum += Query(mid + 1, right, rs(rt), s, t, val);
    }
    return sum;
}
```

例题：*https://www.luogu.org/problemnew/show/P3373*

5.5 主席树

又称“可持久化 (权值) 线段树”，主要用于查询区间第 k 小 (大) 值。效率高于归并树低于划分树。

```cpp
#include <bits/stdc++.h>

using namespace std;
const int N = 1e5 + 5;
struct SegTreeNode {
    int l, r, m;
    int ls, rs;  // 左儿子、右儿子
    int s;       // 结点总数
} tr[N << 5];
// tcnt表示当前空余的节点编号, rt[i]为时间点为i的线段树的根结点
int tcnt, rt[N];

int n, m, a[N], i2x[N], len;

inline int x2i(int x) {
    return lower_bound(i2x + 1, i2x + 1 + len, x) - i2x;
}

// 区间为[l,r), 返回子树的根结点
int build(int l, int r) {
    int x = tcnt++;
    int mid = (l + r) / 2;
    tr[x].l = l, tr[x].r = r, tr[x].m = mid;
    tr[x].s = 0;
    if (r - l == 1)
        return x;
    tr[x].ls = build(l, mid);
    tr[x].rs = build(mid, r);
    return x;
}

// 在pos处插入数, pre为上一个版本的根结点
int insert(int k, int pre) {
    int cur = tcnt++;
    tr[cur] = tr[pre];
    tr[cur].s++;
    if (tr[cur].r - tr[cur].l == 1)
        return cur;
    if (k < tr[cur].m)
        tr[cur].ls = insert(k, tr[cur].ls);
    else
        tr[cur].rs = insert(k, tr[cur].rs);
    return cur;
}

// 查询区间(x,y]中的第k大值
// tr[x] 和 tr[y] 表示时间点不一样的*两棵*的线段树中的节点
int query(int x, int y, int k) {
    // 当前区间的左子树中数的个数 = y时间的左子树中数的个数 - x时间的左子树中数的个数
    int s = tr[tr[y].ls].s - tr[tr[x].ls].s;
    if (tr[x].r - tr[x].l == 1)
        return tr[x].l;  // 此处返回的l不是下标，而是一个权值，是离散化后的权值
    if (k <= s)
        return query(tr[x].ls, tr[y].ls, k);
    else
        return query(tr[x].rs, tr[y].rs, k - s);
}

inline void init() {
    int i;
    // 离散化
    for (i = 1; i <= n; i++)
        i2x[i] = a[i];
```

```
    sort(i2x + 1, i2x + 1 + n);
    len = unique(i2x + 1, i2x + 1 + n) - i2x - 1;
    // 将下标当成时间序列，依次“新建”线段树
    rt[0] = build(1, len + 1);
    for (i = 1; i <= n; i++)
        rt[i] = insert(x2i(a[i]), rt[i - 1]);
}

inline void work() {
    int i, l, r, k;
    for (i = 1; i <= m; i++) {
        scanf("%d%d%d", &l, &r, &k);
        // 此处的查询，应该查的是两个时间点的两棵线段树，返回的是离散化后的权值，需要i2x恢复
        printf("%d\n", i2x[query(rt[l - 1], rt[r], k)]);
    }
}

int main() {
    int i;
    scanf("%d%d", &n, &m);
    for (i = 1; i <= n; i++)
        scanf("%d", &a[i]);
    init();
    work();

    return 0;
}
```

5.6 带 Lazy 标记的线段树

以下是区间修改 + 区间最大值查询。
若是区间修改 + **区间和查询**，则在 pushdown 时需要将 lazy 标记**乘上区间长度**加到结点上。

```
#include <bits/stdc++.h>

using namespace std;
const int N = 1e5 + 10;

int n;  // 长度

struct Node {
    int l, r, m;
    int mx;   // [l,r)中的最大值
    int tag;  // lazy标记
} t[4 * N];   // 数组大小不要忘记 * 4

// 将lazy标记下推
inline void pushdown(int x) {
    Node& cur = t[x];
    if (cur.r - cur.l == 1)
        return;
    Node &lch = t[x * 2], &rch = t[x * 2 + 1];
    lch.mx += cur.tag, rch.mx += cur.tag;
    lch.tag += cur.tag, rch.tag += cur.tag;
    cur.tag = 0;
}

// 由x的儿子更新x结点，此时应确保x的儿子为最新
inline void pushup(int x) {
    Node& cur = t[x];
    if (cur.r - cur.l == 1)
        return;
    Node &lch = t[x * 2], &rch = t[x * 2 + 1];
    cur.mx = max(lch.mx, rch.mx);
}

// 建树。注意初始时叶子是否为0，若不是，需要pushup
void build(int l, int r, int x) {
    Node& cur = t[x];
    cur.l = l, cur.r = r, cur.m = (l + r) / 2;
    cur.mx = 0, cur.tag = 0;  // 初始化最大值、lazy标记
    if (r - l == 1)
        return;
    build(l, cur.m, x * 2);
    build(cur.m, r, x * 2 + 1);
    pushup(x);  // 若初始值非0，则这句一定要加
}

// [l,r)每个元素加v。注意打标记时应更新被打标记结点的mx，并且应立即pushdown,pushup，过程中也需要pushdown
    ,pushup
void update(int l, int r, int x, int v) {
    Node& cur = t[x];
    if (cur.l == l && cur.r == r) {
        cur.tag += v, cur.mx += v;  // 打标记的时候一定是同时更新max的
        pushdown(x), pushup(x);
        return;
    }
    pushdown(x);
    if (r <= cur.m)
        update(l, r, x * 2, v);
    else if (l >= cur.m)
        update(l, r, x * 2 + 1, v);
    else
        update(l, cur.m, x * 2, v), update(cur.m, r, x * 2 + 1, v);
    pushup(x);
```

```
}

// 查询[l,r)的最大值。过程中注意pushdown
int query(int l, int r, int x) {
    Node& cur = t[x];
    pushdown(x);
    if (cur.l == l && cur.r == r)
        return cur.mx;
    int mx = 0;
    if (r <= cur.m)
        mx = query(l, r, x * 2);
    else if (l >= cur.m)
        mx = query(l, r, x * 2 + 1);
    else
        mx = max(query(l, cur.m, x * 2), query(cur.m, r, x * 2 + 1));
    return mx;
}

int main() {
    ios::sync_with_stdio(0);
    cin.tie(0);

    int m, i, l, r, v;
    cin >> n >> m;
    build(1, n + 1, 1);  // 不要忘记build初始化，[l,r+1)
    for (i = 1; i <= m; i++) {
        cin >> v >> l >> r;
        if (!v)
            cout << query(l, r + 1, 1) << endl;
        else
            update(l, r + 1, 1, 1);
    }

    return 0;
}
```

5.7 归并树

5.7.1 简介

归并排序，自底向上越来越有序。而归并树则是将归并排序的中间结果保留了下来。
为什么要这么做呢？因为这样当询问区间时，总是可以将询问区间二分成某一次或几次归并排序的中间结果。

5.7.2 区间第 k 小值

5.7.2.1 问题简述

给定 n 个数，m 次查询，每次查询 [l,r] 内从小到大第 k 个数，输出这个数。

5.7.2.2 解决方案

对这 n 个数先排序，然后二分尝试，将二分得到的中间值代入归并树中，看是否排名为 k，**对所有排名小于等于 k 的取最大值**。
建树时间复杂度 O(NlogN)，查询时间复杂度 O(logNlogN)。

```
#include <bits/stdc++.h>

using namespace std;
typedef long long ll;
const int N = 1e5 + 10, INF = 1e9 + 8;

int a[N], n, m;

struct Node {
    int l, r, m;
    vector<int> a;
} t[N * 3];

// 由a[]数组初始化归并树
void build(int l, int r, int x) {
    Node& cur = t[x];
    cur.l = l, cur.r = r, cur.m = (l + r) / 2;
    cur.a.clear();
    if (r - l == 1) {
        cur.a.push_back(a[cur.l]);
        return;
    }
    build(l, cur.m, x * 2);
    build(cur.m, r, x * 2 + 1);
    Node &lch = t[x * 2], &rch = t[x * 2 + 1];
    vector<int>::iterator it1, it2;
    it1 = lch.a.begin(), it2 = rch.a.begin();
    // 合并左右子树
    while (it1 != lch.a.end() && it2 != rch.a.end()) {
        if (*it1 <= *it2)
            cur.a.push_back(*it1), it1++;
        else
            cur.a.push_back(*it2), it2++;
    }
    while (it1 != lch.a.end())
        cur.a.push_back(*it1), it1++;
    while (it2 != rch.a.end())
        cur.a.push_back(*it2), it2++;
}

// 查询在区间[l,r)内比v小的数的个数，稍加修改便可查询大于等于k的最小值
int query(int l, int r, int v, int x) {
    Node& cur = t[x];
    if (cur.l == l && cur.r == r)
        return lower_bound(cur.a.begin(), cur.a.end(), v) - cur.a.begin();
    if (r <= cur.m)
        return query(l, r, v, x * 2);
    else if (l >= cur.m)
```

```cpp
        return query(l, r, v, x * 2 + 1);
    else
        return query(l, cur.m, v, x * 2) + query(cur.m, r, v, x * 2 + 1);
}

// 查询区间[l,r)区间内的从小到大第k个值
int QR(int ql, int qr, int k) {
    int l = 1, r = n + 1, m, ans = -INF;
    while (l < r) {
        m = (l + r) / 2;
        int rank = query(ql, qr, a[m], 1) + 1;
        if (rank <= k)  // 可能有相等的值，所以需要<=
            ans = max(ans, a[m]);
        if (rank <= k)
            l = m + 1;
        else
            r = m;
    }
    return ans;
}

int main() {
    int i, l, r, k;
    while (scanf("%d%d", &n, &m) != EOF) {
        for (i = 1; i <= n; i++)
            scanf("%d", &a[i]);
        build(1, n + 1, 1);
        sort(a + 1, a + 1 + n);
        for (i = 1; i <= m; i++) {
            scanf("%d%d%d", &l, &r, &k);
            printf("%d\n", QR(l, r + 1, k));
        }
    }

    return 0;
}
```

5.7.2.3 思考

如何取区间内不重复的第 k 大？
结点中用两个 vector，其中一个是原来的不变，另一个用于存放不重复的数，查询的时候在后者中查。

5.7.3 区间内大于等于 v 的最小值

5.7.3.1 问题简述

给定 n 个数，m 次查询，每次查询 [l,r] 内比 v 大的最小值，输出这个最小值。

5.7.3.2 解决方案

归并树天生适合的问题。对于对应区间，直接返回 lower_bound 找到的第一个数注意，如果没找到返回一个无限大，对于其余祖先节点，返回左右子树中找到的最小值。

```cpp
int query(int l, int r, int v, int x) {
    Node& cur = t[x];
    if (cur.l == l && cur.r == r) {
        vector<int>::iterator it;
        it = lower_bound(cur.a.begin(), cur.a.end(), v);
        if (it == cur.a.end())
            return n + 1;
        return *it;
    }
    if (r <= cur.m)
        return query(l, r, v, x * 2);
    else if (l >= cur.m)
        return query(l, r, v, x * 2 + 1);
```

```
    else
        return min(query(l, cur.m, v, x * 2), query(cur.m, r, v, x * 2 + 1));
}
```

5.8 划分树

5.8.1 简介

划分树是模拟快速排序的。快速排序，自顶向下越来越有序。划分树是在进行快速排序的过程中记录当前子段中每个位置的数是否进入左子树，并用前缀和的思想统计它们。之后再查询时便可依据进入左子树的个数计算第 k 个数是在左子树还是右子树。

5.8.2 区间第 k 小值

5.8.2.1 问题简述

给定 n 个数，m 次查询，每次查询 [l,r] 内从小到大第 k 个数，输出这个数。

5.8.2.2 解决方案

```
#include <bits/stdc++.h>

using namespace std;
typedef long long ll;
const int N = 1e5 + 10;

int a[N], sorted[N], n;
int seg[21][N], toleft[21][N];

// 建树
void build(int l, int r, int dep) {
    if (l == r) {
        seg[dep + 1][l] = seg[dep][l];
        return;
    }
    int i, m = (l + r) / 2, lp = l, rp = m + 1, cnt = 0, t = 0;
    // cnt: 该节点内比sorted[m]小的元素个数
    for (i = l; i <= r; i++)
        cnt += seg[dep][i] < sorted[m];

    for (i = l; i <= r; i++) {
        if (i == l)
            toleft[dep][i] = 0;
        else
            toleft[dep][i] = toleft[dep][i - 1];

        if (seg[dep][i] < sorted[m])
            seg[dep + 1][lp++] = seg[dep][i], toleft[dep][i]++;
        else if (seg[dep][i] > sorted[m])
            seg[dep + 1][rp++] = seg[dep][i];
        else {                      // ==
            if (t < m - l + 1 - cnt)  // m-l+1-cnt: 左边最多可以放多少个sorted[m]
                seg[dep + 1][lp++] = seg[dep][i], toleft[dep][i]++, t++;
            else
                seg[dep + 1][rp++] = seg[dep][i];
        }
    }

    build(l, m, dep + 1);
    build(m + 1, r, dep + 1);
}

// 询问区间[l,r]内从小到大的第k个值，当前区间为[sl,sr](初始时为[1,n])，当前深度为dep(初始时为1)
int query(int l, int r, int k, int sl, int sr, int dep) {
    if (r == l)
        return seg[dep][l];
    int m = (sl + sr) / 2;
    int tlsl = (l - 1 >= sl ? toleft[dep][l - 1] : 0), tlsr = toleft[dep][sr];
    int tlr = toleft[dep][r];
    if (k <= tlr - tlsl)
```

```
        return query(sl + tlsl, m - (tlsr - tlr), k, sl, m, dep + 1);
    else
        return query(m + 1 + l - sl - tlsl, sr - (sr - r - (tlsr - tlr)), k - (tlr - tlsl), m + 1, sr, dep + 1);
}

int main() {
    int m, i, l, r, k;
    scanf("%d%d", &n, &m);

    // 以下4行是初始化
    for (i = 1; i <= n; i++)
        scanf("%d", &a[i]), sorted[i] = seg[1][i] = a[i];
    sort(sorted + 1, sorted + 1 + n);
    build(1, n, 1);

    for (i = 1; i <= m; i++) {
        scanf("%d%d%d", &l, &r, &k);
        int ans = query(l, r, k, 1, n, 1);
        printf("%d\n", ans);
    }

    return 0;
}
```

5.9 左偏树

5.9.1 模板

```cpp
const int N = 1e3 + 10;
struct Node {
    int k, d, fa, ch[2];  // 键，距离，父亲，左儿子，右儿子
} t[N];

// 取右子树的标号
int& rs(int x) {
    return t[x].ch[t[t[x].ch[1]].d < t[t[x].ch[0]].d];
}

// 用于删除非根节点后向上更新、
// 建议单独用，因为需要修改父节点
void pushup(int x) {
    if (!x)
        return;
    if (t[x].d != t[rs(x)].d + 1) {
        t[x].d = t[rs(x)].d + 1;
        pushup(t[x].fa);
    }
}

// 整个堆加上、减去一个数或乘上一个整数(不改变相对大小)，类似于lazy标记
void pushdown(int x) {
}

// 合并x和y
int merge(int x, int y) {
    // 若一个堆为空，则返回另一个堆
    if (!x || !y)
        return x | y;
    // 取较小的作为根
    if (t[x].k > t[y].k)
        swap(x, y);
    // 下传标记，这么写的条件是必须保证堆顶元素时刻都是最新的
    pushdown(x);
    // 递归合并右儿子和另一个堆 // 若不满足左偏树性质则交换两儿子 // 更新右子树的父亲，只有右子树有父亲
    t[rs(x) = merge(rs(x), y)].fa = x;
    // 更新dist
    t[x].d = t[rs(x)].d + 1;
    return x;
}
```

5.9.2 模板题 P3377 【模板】左偏树（可并堆）

5.9.2.1 题目描述

如题，一开始有 N 个小根堆，每个堆包含且仅包含一个数。接下来需要支持两种操作：
操作 1：1 x y 将第 x 个数和第 y 个数所在的小根堆合并（若第 x 或第 y 个数已经被删除或第 x 和第 y 个数在用一个堆内，则无视此操作）
操作 2：2 x 输出第 x 个数所在的堆最小数，并将其删除（若第 x 个数已经被删除，则输出-1 并无视删除操作）。当堆里有多个最小值时，优先删除原序列的靠前的。

5.9.2.2 涉及知识点

1. 左偏树的基本操作（合并、删除）
2. 并查集查询结点所在的堆的根

需要注意的是：
合并前要检查是否已经在同一堆中。
左偏树的深度可能达到 $O(n)$，因此找一个点所在的堆顶要用并查集维护，不能直接暴力跳父亲。（虽然很多题数据水，暴力跳父亲可以过⋯⋯）（用并查集维护根时要保证原根指向新根，新根指向自己。）

5.9.2.3 代码

```
#include <bits/stdc++.h>

using namespace std;

const int N = 1e5 + 10;

bitset<N> f;  // 用于标记某个元素是否被删除

// 关键字
struct Key {
    int a, b;
    bool operator<(const Key& rhs) const {
        if (a != rhs.a)
            return a < rhs.a;
        return b < rhs.b;
    }
};

// 左偏树节点
struct Node {
    Key k;
    int d, fa, ch[2];
} t[N];

int& rs(int x) {
    return t[x].ch[t[t[x].ch[1]].d < t[t[x].ch[0]].d];
}

int merge(int x, int y) {
    if (!x || !y)
        return x | y;
    if (t[y].k < t[x].k)
        swap(x, y);
    t[rs(x) = merge(rs(x), y)].fa = x;
    t[x].d = t[rs(x)].d + 1;
    return x;
}

struct UF {
    int fa, t;
} uf[N];  // 并查集

int find(int x) {
    if (x == uf[x].fa)
        return x;
    return uf[x].fa = find(uf[x].fa);
}

void ufUnion(int x, int y) {
    x = find(x), y = find(y);
    uf[y].fa = x;
}

int main() {
    ios::sync_with_stdio(0);
    cin.tie(0);

    int n, m, i, x, y;
    cin >> n >> m;
    for (i = 1; i <= n; i++) {
        cin >> x;
        uf[i].fa = i, uf[i].t = i;
        t[i].d = 1, t[i].k = Key({x, i});
    }
```

```
    for (i = 1; i <= m; i++) {
        cin >> x;
        if (x == 1) {
            cin >> x >> y;
            if (f.test(x) || f.test(y) || uf[find(x)].t == uf[find(y)].t)
                continue;
            int root = merge(uf[find(x)].t, uf[find(y)].t);
            ufUnion(x, y);
            uf[find(x)].t = root;
        } else {
            cin >> x;
            if (f.test(x)) {
                cout << -1 << endl;
                continue;
            }
            cout << t[uf[find(x)].t].k.a << endl;
            f[t[uf[find(x)].t].k.b] = 1;
            uf[find(x)].t = merge(t[uf[find(x)].t].ch[0], t[uf[find(x)].t].ch[1]);
        }
    }

    return 0;
}
```

5.9.3 洛谷 P1552 [APIO2012] 派遣

5.9.3.1 题目描述

题目太长，简述。你有预算 m 元，给定 n 个人，具有上下级关系，构成一棵树，每个人有两个参数-花费、领导力。让你选择先选一个点作为领导者，然后在以领导者为根的子树中任意选择一些点 (要求花费不超过 m)，得到的价值为领导者的领导力 * 选定的人数。问最大价值为多少?

5.9.3.2 涉及知识点

树上问题
可并堆的合并
用堆维护背包

5.9.3.3 思路

大根堆中存储每个点的花费。递归地，对于一个点 x，我们合并 x 的所有儿子节点的堆，并计算其总和，如果总和大于 m，不断弹出堆顶元素并更新总和，直到总和小于等于 m。有点像带反悔的贪心。

5.9.4 洛谷 P3261 [JLOI2015] 城池攻占

5.9.4.1 题目描述

你要用 m 个骑士攻占 n 个城池。n 个城池 (1 到 n) 构成了一棵有根树，1 号城池为根，其余城池父节点为 fi。m 个骑士 (1 到 m)，其中第 i 个骑士的初始战斗力为 si，第一个攻击的城池为 ci。
每个城池有一个防御值 hi，如果一个骑士的战斗力大于等于城池的生命值，那么骑士就可以占领这座城池；否则占领失败，骑士将在这座城池牺牲。占领一个城池以后，骑士的战斗力将发生变化，然后继续攻击管辖这座城池的城池，直到占领 1 号城池，或牺牲为止。
除 1 号城池外，每个城池 i 会给出一个战斗力变化参数 ai;vi。若 ai =0，攻占城池 i 以后骑士战斗力会增加 vi；若 ai =1，攻占城池 i 以后，战斗力会乘以 vi。注意每个骑士是单独计算的。也就是说一个骑士攻击一座城池，不管结果如何，均不会影响其他骑士攻击这座城池的结果。
现在的问题是，对于每个城池，输出有多少个骑士在这里牺牲；对于每个骑士，输出他攻占的城池数量。

5.9.4.2 涉及知识点

树上问题
可并堆的合并
堆的整体操作 (打标记，pushdown)

5.9.4.3 注意

打标记之后，应该立即更新被打标记的点，然后 pushdown，pushdown 总是由父节点发起帮助儿子提前更新。这样，如果某个点有标记，则表示该点本身是最新的，但他应该为儿子更新。也即上一层的标记是为下一层准备的。

5.9.4.4 打标记、下传代码

```
// x被打标记，立即更新x的值，并且将标记存放在此(准备之后让pushdown给下一层更新)
inline void mark(ll x, ll a, ll b) {
    if (!x)
        return;
    t[x].k.s = t[x].k.s * b + a;            // 更新自身
    t[x].a *= b, t[x].b *= b, t[x].a += a;  // 寄存标记
}

// 下传标记，本质上就是再给儿子们mark()一下，然后清空自身标记
inline void pushdown(ll x) {
    if (!x)
        return;
    mark(t[x].ch[0], t[x].a, t[x].b);
    mark(t[x].ch[1], t[x].a, t[x].b);
    t[x].a = 0, t[x].b = 1;
}
```

5.9.5 洛谷 P3273 [SCOI2011] 棘手的操作

5.9.5.1 题目描述

有 N 个节点，标号从 1 到 N，这 N 个节点一开始相互不连通。第 i 个节点的初始权值为 a[i]，接下来有如下一些操作：
U x y: 加一条边，连接第 x 个节点和第 y 个节点
A1 x v: 将第 x 个节点的权值增加 v
A2 x v: 将第 x 个节点所在的连通块的所有节点的权值都增加 v
A3 v: 将所有节点的权值都增加 v
F1 x: 输出第 x 个节点当前的权值
F2 x: 输出第 x 个节点所在的连通块中，权值最大的节点的权值
F3: 输出所有节点中，权值最大的节点的权值

5.9.5.2 涉及知识点

左偏树
并查集
multiset
整体标记
启发式合并

5.9.5.3 思路

这题题如其名，非常棘手。
首先，找一个节点所在堆的堆顶要用并查集，而不能暴力向上跳。
再考虑单点查询，若用普通的方法打标记，就得查询点到根路径上的标记之和，最坏情况下可以达到的复杂度。
如果只有堆顶有标记，就可以快速地查询了，但如何做到呢？
可以用类似启发式合并的方式，每次合并的时候把较小的那个堆标记暴力下传到每个节点，然后把较大的堆的标记作为合并后的堆的标记。由于合并后有另一个堆的标记，所以较小的堆下传标记时要下传其标记减去另一个堆的标记。由于每个节点每被合并一次所在堆的大小至少乘二，所以每个节点最多被下放次标记，暴力下放标记的总复杂度就是 $O(n)$。
再考虑单点加，先删除，再更新，最后插入即可。
然后是全局最大值，可以用一个平衡树/支持删除任意节点的堆（如左偏树）/multiset 来维护每个堆的堆顶。

所以，每个操作分别如下：
1. 暴力下传点数较小的堆的标记，合并两个堆，更新 size、tag，在 multiset 中删去合并后不在堆顶的那个原堆

顶。
2. 删除节点，更新值，插入回来，更新 multiset。需要分删除节点是否为根来讨论一下。
3. 堆顶打标记，更新 multiset。
4. 打全局标记。
5. 查询值 + 堆顶标记 + 全局标记。
6. 查询根的值 + 堆顶标记 + 全局标记。
7. 查询 multiset 最大值 + 全局标记。

5.9.6 洛谷 P4331 Sequence 数字序列

5.9.6.1 题目描述

这是一道论文题
给定一个整数序列 $a_1, a_2, ..., a_n$，求出一个递增序列 $b_1 < b_2 < ... < b_n$，使得序列 a_i 和 b_i 的各项之差的绝对值之和 $|a_1 - b_1| + |a_2 - b_2| + ... + |a_n - b_n|$ 最小。

5.9.6.2 涉及知识点

堆的合并 (因为没有整体标记，所以这里可以用 __gnu_pbds::priority_queue)
堆维护区间中位数
递增序列转非递减序列（减下标法）

5.9.6.3 思路

递增序列转非递减序列：把 a[i] 减去 i，易知 b[i] 也减去 i 后答案不变，本来 b 要求是递增序列，这样就转化成了不下降序列，方便操作。
堆维护区间中位数：大根堆，每次合并之后，如果堆内元素个数大于区间的一半，则一直 pop 直到等于一半，堆顶元素即为中位数。(这么做的前提是中位数大的区间内的最小值小于等于另一个区间内仅比那个区间中位数大的数)

Chapter 6

图论

6.1 最短路

6.1.1 单源最短路径

6.1.1.1 Dijkstra

```
void Dijkstra()
{
    memset(dist, 0x3f, sizeof(dist));
    memset(vis, 0, sizeof(vis));
    priority_queue<pii, vector<pii>, greater<pii>> q;
    dist[1] = 0;
    q.push({dist[1], 1});
    while (!q.empty())
    {
        int x = q.top().second;
        q.pop();
        if (!vis[x])
        {
            vis[x] = 1;
            for (auto it : v[x])
            {
                int y = it.first;
                if (dist[y] > dist[x] + it.second)
                {
                    dist[y] = dist[x] + it.second;
                    q.push({dist[y], y});
                }
            }
        }
    }
}
```

6.1.1.2 Bellman-Ford 和 SPFA

```
void SPFA()
{
    memset(dis, 0x3f, sizeof(dis));
    memset(vis, 0, sizeof(vis));
    queue<int> q;
    dis[1] = 0;
    vis[1] = 1;
    q.push(1);
    while (!q.empty())
    {
        int x = q.front();
        q.pop();
        vis[x] = 0;
        for (int i = 0; i < v[x].size(); i++)
        {
            int y = v[x][i].first;
            int z = v[x][i].second;
            if (dis[y] > dis[x] + z)
            {
                dis[y] = dis[x] + z;
                if (!vis[y])
                    q.push(y), vis[y] = 1;
            }
        }
    }
}
```

例题分析

POJ3662 Telephone Lines（分层图最短路/二分答案，双端队列 BFS）

P1073 最优贸易（原图与反图，枚举节点）

P3008 [USACO11JAN] 道路和飞机 Roads and Planes（DAG，拓扑序，连通块）

6.1.2 任意两点间最短路径

6.1.2.1 Floyd

```cpp
void get_path(int i, int j)
{
    if (!path[i][j])
        return;
    get_path(i, path[i][j]);
    p.push_back(path[i][j]);
    get_path(path[i][j], j);
}
void Floyd()
{
    memcpy(d, a, sizeof(d));
    for (int k = 1; k <= n; k++)
    {
        for (int i = 1; i < k; i++)
        {
            for (int j = i + 1; j < k; j++)
            {
                //注意溢出
                ll temp = d[i][j] + a[i][k] + a[k][j];
                if (ans > temp)
                {
                    ans = temp;
                    p.clear();
                    p.push_back(i);
                    get_path(i, j);
                    p.push_back(j);
                    p.push_back(k);
                }
            }
        }
        for (int i = 1; i <= n; i++)
        {
            for (int j = 1; j <= n; j++)
            {
                ll temp = d[i][k] + d[k][j];
                if (d[i][j] > temp)
                {
                    d[i][j] = temp;
                    path[i][j] = k;
                }
            }
        }
    }
}
```

例题分析

POJ1094 Sorting It All Out（传递闭包）

POJ1734 Sightseeing trip（无向图最小环）

POJ3613 Cow Relays（离散化，广义矩阵乘法，快速幂）

6.2 最小生成树

6.2.1 Kruskal

基于并查集

```
void Init()
{
    for (int i = 1; i <= n; i++)
        fa[i] = i;
}
int Find(int x)
{
    if (x == fa[x])
        return x;
    return fa[x] = Find(fa[x]);
}
void Kruskal()
{
    Init();
    sort(e.begin(), e.end());
    int ans=0;
    for (int i = 0; i < e.size(); i++)
    {
        int u = e[i].u, v = e[i].v;
        int fu = Find(u), fv = Find(v);
        if (fu != fv)
        {
            fa[fu] = fv;
            ans += e[i].w;
        }
    }
}
```

6.2.2 Prim

```
void Prim()
{
    memset(vis, 0, sizeof(vis));
    memset(d, 0x3f, sizeof(d));
    d[1] = 0;
    int temp = n;
    int ret = 0;
    while (temp--)
    {
        int min_pos = 0;
        for (int i = 1; i <= n; i++)
            if (!vis[i] && (!min_pos || d[i] < d[min_pos]))
                min_pos = i;
        if (min_pos)
        {
            vis[min_pos] = 1;
            ret += d[min_pos];
            for (int i = 1; i <= n; i++)
                if (!vis[i]) d[i] = min(d[i], weight[min_pos][i]);
        }
    }
}
```

例题分析

走廊泼水节（Kruskal，最小生成树扩充为完全图）

POJ1639 Picnic Planning（度限制最小生成树，连通块，树形 DP）

```
#include <algorithm>
#include <cstring>
#include <iostream>
```

```
#include <map>
#include <string>
#include <vector>
using namespace std;
#define inf 0x3f3f3f3f
#define N 25
#define M 500
map<string, int> name;
struct edge
{
    int u, v, w;
    bool operator<(const edge &e) const
    {
        return w < e.w;
    }
};
int n, s, ptot = 0, a[N][N], ans, fa[N], d[N], ver[N];
vector<edge> e;
bool vis[N][N];
edge dp[N]; //dp[i]  1...i路径上的最大边
void Init()
{
    for (int i = 1; i <= ptot; i++)
        fa[i] = i;
}
int Find(int x)
{
    if (x == fa[x])
        return x;
    return fa[x] = Find(fa[x]);
}
void Kruskal()
{
    Init();
    sort(e.begin(), e.end());
    for (int i = 0; i < e.size(); i++)
    {
        int u = e[i].u, v = e[i].v;
        if (u != 1 && v != 1)
        {
            int fu = Find(u), fv = Find(v);
            if (fu != fv)
            {
                fa[fu] = fv;
                vis[u][v] = vis[v][u] = 1;
                ans += e[i].w;
            }
        }
    }
}
void DFS(int cur, int pre)
{
    for (int i = 2; i <= ptot; i++)
    {
        if (i != pre && vis[cur][i])
        {
            if (dp[i].w == -1)
            {
                if (dp[cur].w < a[cur][i])
                {
                    dp[i].u = cur;
                    dp[i].v = i;
                    dp[i].w = a[cur][i];
                }
                else
                    dp[i] = dp[cur];
```

```
                }
                DFS(i, cur);
            }
        }
}
int main()
{
    ios::sync_with_stdio(false);
    cin.tie(0);
    cin >> n;
    string s1, s2;
    int len;
    name["Park"] = ++ptot;
    memset(a, 0x3f, sizeof(a));
    memset(d, 0x3f, sizeof(d));
    //Park: 1
    for (int i = 0; i < n; i++)
    {
        cin >> s1 >> s2 >> len;
        if (!name[s1])
            name[s1] = ++ptot;
        if (!name[s2])
            name[s2] = ++ptot;
        int u = name[s1], v = name[s2];
        a[u][v] = a[v][u] = min(a[u][v], len); //无向图邻接矩阵
        e.push_back({u, v, len});
    }
    cin >> s; //度数限制
    ans = 0;
    Kruskal();
    for (int i = 2; i <= ptot; i++)
    {
        if (a[1][i] != inf)
        {
            int rt = Find(i);
            if (d[rt] > a[1][i])
                d[rt] = a[1][i], ver[rt] = i;
        }
    }
    for (int i = 2; i <= ptot; i++)
    {
        if (d[i] != inf)
        {
            s--;
            ans += d[i];
            vis[1][ver[i]] = vis[ver[i]][1] = 1;
        }
    }
    while (s-- > 0)
    {
        memset(dp, -1, sizeof(dp));
        dp[1].w = -inf;
        for (int i = 2; i <= ptot; i++)
        {
            if (vis[1][i])
                dp[i].w = -inf;
        }
        DFS(1, -1);
        int w = -inf;
        int v;
        for (int i = 2; i <= ptot; i++)
        {
            if (w < dp[i].w - a[1][i])
            {
                w = dp[i].w - a[1][i];
                v = i;
```

```
            }
        }
        if (w <= 0)
            break;
        ans -= w;
        vis[1][v] = vis[v][1] = 1;
        vis[dp[v].u][dp[v].v] = vis[dp[v].v][dp[v].u] = 0;
    }
    cout << "Total miles driven: " << ans << endl;
    system("pause");
    return 0;
}
```

POJ2728 Desert King （最优比率生成树，0/1 分数规划，二分）
黑暗城堡（最短路径生成树计数，最短路，排序）

6.3 树的直径

6.3.1 树形 DP 求树的直径

仅能求出直径长度，无法得知路径信息，可处理负权边。

```
int dp[N];
//dp[rt] 以rt为根的子树 从rt出发最远可达距离
/*
    对于每个结点x f[x]:经过节点x的最长链长度
*/
void DP(int rt)
{
    dp[rt]=0;//单点
    vis[rt]=1;
    for(int i=head[rt];i;i=nxt[i])
    {
        int s=ver[i];
        if(!vis[s])
        {
            DP(s);
            diameter=max(diameter,dp[rt]+dp[s]+edge[i]);
            dp[rt]=max(dp[rt],dp[s]+edge[i]);
        }
    }
}
```

6.3.2 两次 BFS/DFS 求树的直径

无法处理负权边，容易记录路径

```
void DFS(int start,bool record_path)
{
    vis[start]=1;
    for(int i=head[start];i;i=nxt[i])
    {
        int s=ver[i];
        if(!vis[s])
        {
            dis[s]=dis[start]+edge[i];
            if(record_path) path[s]=i;
            DFS(s,record_path);
        }
    }
    vis[start]=0;//清理
}
```

例题分析

P3629 [APIO2010] 巡逻（两种求树直径方法的综合应用）

P1099 树网的核（枚举）

6.4 最近公共祖先（LCA）

6.4.1 树上倍增

```
void BFS()
{
    queue<int> q;
    q.push(1);
    d[1] = 1;
    while (!q.empty())
    {
        int x = q.front();
        q.pop();
        for (int i = head[x]; i; i = nxt[i])
```

```
        {
            int y = ver[i];
            if (!d[y])
            {
                d[y] = d[x] + 1;
                fa[y][0] = x;
                for (int j = 1; j <= k; j++)
                {
                    fa[y][j] = fa[fa[y][j - 1]][j - 1];
                }
                q.push(y);
            }
        }
    }
}
int LCA(int x, int y)
{
    if (d[x] < d[y])
        swap(x, y);
    for (int i = k; i >= 0; i--)
        if (d[fa[x][i]] >= d[y])
            x = fa[x][i];
    if (x == y)
        return y;
    for (int i = k; i >= 0; i--)
        if (fa[x][i] != fa[y][i])
            x = fa[x][i], y = fa[y][i];
    return fa[x][0];
}
```

6.4.2 Tarjan

```
int Find(int x)
{
    if (x == fa[x])
        return x;
    return fa[x] = Find(fa[x]);
}
void Tarjan(int x)
{
    vis[x] = 1;
    for (int i = head[x]; i; i = nxt[i])
    {
        int y = ver[i];
        if (!vis[y])
        {
            Tarjan(y);
            fa[y] = x;
        }
    }
    for (int i = 0; i < q[x].size(); i++)
    {
        int y = q[x][i].first, id = q[x][i].second;
        if (vis[y] == 2)
            lca[id] = Find(y);
    }
    vis[x] = 2;
}
```

6.5 树上差分

例题分析

POJ3417 Network（LCA，树上差分，边覆盖）

6302 雨天的尾巴（LCA，树上差分，点覆盖，权值线段树，线段树合并）
P1600 天天爱跑步（LCA，树上差分）

6.6 LCA 的综合应用

例题分析
CH56C 异象石（dfn 时间戳，LCA）
P4180 【模板】严格次小生成树 [BJWC2010]（树上倍增）

```
#include <algorithm>
#include <iostream>
#include <math.h>
#include <queue>
#include <stdio.h>
using namespace std;
const int N = 1e5 + 50;
const int M = 6e5 + 50;
#define ll long long
#define pii pair<int, int>
#define inf 0x3f3f3f3f
int n, m, k, F[N][20], d[N], fa[N];
int head[N], ver[M], nxt[M], edge[M], tot;
ll G[N][20][2];
void add(int x, int y, int z)
{
    ver[++tot] = y, nxt[tot] = head[x], head[x] = tot, edge[tot] = z;
}
struct edge
{
    int x, y, z;
    bool used;
    bool operator<(const edge &e) const
    {
        return z < e.z;
    }
} e[M];
int Find(int x)
{
    if (fa[x] == x)
        return x;
    return fa[x] = Find(fa[x]);
}
ll Kruskal()
{
    for (int i = 1; i <= n; i++)
        fa[i] = i;
    sort(e + 1, e + 1 + m);
    ll ans = 0;
    int cnt = 0;
    for (int i = 1; i <= m; i++)
    {
        int fx = Find(e[i].x), fy = Find(e[i].y);
        if (fx != fy)
        {
            fa[fx] = fy;
            ans += e[i].z;
            e[i].used = true;
            cnt++;
            if (cnt >= n - 1)
                break;
        }
    }
    return ans;
}
void BFS()
```

```
{
    k = log2(n) + 1;
    queue<int> q;
    q.push(1), d[1] = 1;
    for (int i = 0; i <= k; i++)
        G[1][i][0] = G[1][i][1] = -inf;
    while (q.size())
    {
        int x = q.front();
        q.pop();
        for (int i = head[x]; i; i = nxt[i])
        {
            int y = ver[i];
            if (!d[y])
            {
                d[y] = d[x] + 1;
                F[y][0] = x;
                G[y][0][0] = edge[i];
                G[y][0][1] = -inf;
                for (int j = 1; j <= k; j++)
                {
                    F[y][j] = F[F[y][j - 1]][j - 1];
                    G[y][j][0] = max(G[y][j - 1][0], G[F[y][j - 1]][j - 1][0]);
                    if (G[y][j - 1][0] == G[F[y][j - 1]][j - 1][0])
                        G[y][j][1] = max(G[y][j - 1][1], G[F[y][j - 1]][j - 1][1]);
                    else if (G[y][j - 1][0] > G[F[y][j - 1]][j - 1][0])
                        G[y][j][1] = max(G[F[y][j - 1]][j - 1][0], G[y][j - 1][1]);
                    else
                        G[y][j][1] = max(G[y][j - 1][0], G[F[y][j - 1]][j - 1][1]);
                }
                q.push(y);
            }
        }
    }
}
pii LCA(int x, int y)
{
    ll val1 = -inf, val2 = -inf;
    if (d[y] > d[x])
        swap(x, y);
    for (int i = k; i >= 0; i--)
    {
        if (d[F[x][i]] >= d[y])
        {
            if (G[x][i][0] > val1)
                val1 = G[x][i][0], val2 = max(val2, G[x][i][1]);
            else if (G[x][i][0] < val1)
                val2 = max(val2, G[x][i][0]);
            x = F[x][i];
        }
    }
    if (x == y)
        return make_pair(val1, val2);
    for (int i = k; i >= 0; i--)
    {
        if (F[x][i] != F[y][i])
        {
            val1 = max(val1, max(G[x][i][0], G[y][i][0]));
            val2 = max(val2, (val1 == G[x][i][0]) ? G[x][i][1] : G[x][i][0]);
            val2 = max(val2, (val1 == G[y][i][0]) ? G[y][i][1] : G[y][i][0]);
            x = F[x][i], y = F[y][i];
        }
    }
    val1 = max(val1, max(G[x][0][0], G[y][0][0]));
    val2 = max(val2, (val1 == G[x][0][0]) ? G[x][0][1] : G[x][0][0]);
    val2 = max(val2, (val1 == G[y][0][0]) ? G[y][0][1] : G[y][0][0]);
```

```
    return make_pair(val1, val2);
}
int main()
{
    scanf("%d%d", &n, &m);
    for (int i = 1; i <= m; i++)
        scanf("%d%d%d", &e[i].x, &e[i].y, &e[i].z), e[i].used = false;
    ll sum = Kruskal(), ans = 0x3f3f3f3f3f3f3f3f;
    for (int i = 1; i <= m; i++)
        if (e[i].used)
            add(e[i].x, e[i].y, e[i].z), add(e[i].y, e[i].x, e[i].z);
    BFS();
    for (int i = 1; i <= m; i++)
    {
        if (!e[i].used)
        {
            pii temp = LCA(e[i].x, e[i].y);
            if (e[i].z > temp.first)
                ans = min(ans, sum - temp.first + e[i].z);
            else if (e[i].z == temp.first)
                ans = min(ans, sum - temp.second + e[i].z);
        }
    }
    cout << ans << endl;
    //system("pause");
    return 0;
}
```

P1084 疫情控制（二分，树上倍增，贪心）

6.7 基环树

6.8 负环与差分约束

6.8.1 负环

例题分析

POJ3621 Sightseeing Cows（0/1 分数规划，SPFA 判定负环）

6.8.2 差分约束系统

例题分析

POJ1201 Intervals（单源最长路）

6.9 Tarjan 算法与无向图连通性

6.9.1 无向图的割点与桥

6.9.1.1 割边判定法则

```cpp
void Tarjan(int x, int in_edge)
{
    dfn[x] = low[x] = ++num;
    for (int i = head[x]; i; i = nxt[i])
    {
        int y = ver[i];
        if (!dfn[y])
        {
            Tarjan(y, i);
            low[x] = min(low[x], low[y]);
            if (low[y] > dfn[x])
            {
                bridge[i] = bridge[i ^ 1] = true;
            }
        }
        else if (i != (in_edge ^ 1))
            low[x] = min(low[x], dfn[y]);
    }
}
```

6.9.1.2 割点判定法则

```cpp
void Tarjan(int x)
{
    dfn[x] = low[x] = ++num;
    int flag = 0;
    for (int i = head[x]; i; i = nxt[i])
    {
        int y = ver[i];
        if (!dfn[y])
        {
            Tarjan(y);
            low[x] = min(low[x], low[y]);
            if (low[y] >= dfn[x])
            {
                flag++;
                if (x != root || flag >= 2)
                    cut[x] = true;
            }
        }
        else
            low[x] = min(low[x], dfn[y]);
    }
}
```

例题分析
P3469 [POI2008]BLO-Blockade（割点，连通块计数）

6.9.2 无向图的双连通分量

6.9.2.1 边双连通分量 e-DCC 与其缩点

```cpp
void DFS(int x)
{
    color[x] = dcc;
    for (int i = head[x]; i; i = nxt[i])
    {
        int y = ver[i];
        if (!color[y] && !bridge[i])
```

```
            DFS(y);
        }
}
void e_DCC()
{
    dcc = 0;
    for (int i = 1; i <= n; i++)
        if (!color[i])
            ++dcc, DFS(i);
    totc = 1;
    for (int i = 2; i <= tot; i++)
    {
        int u = ver[i ^ 1], v = ver[i];
        if (color[u] != color[v])
            add_c(color[u], color[v]);
    }
}
```

6.9.2.2 点双连通分量 v-DCC 与其缩点

```
void Tarjan(int x)
{
    dfn[x] = low[x] = ++num;
    int flag = 0;
    stack[++top] = x;
    if (x == root && !head[x])
    {
        dcc[++cnt].push_back(x);
        return;
    }
    for (int i = head[x]; i; i = nxt[i])
    {
        int y = ver[i];
        if (!dfn[y])
        {
            Tarjan(y);
            low[x] = min(low[x], low[y]);
            if (low[y] >= dfn[x])
            {
                flag++;
                if (x != root || flag >= 2)
                    cut[x] = true;
                cnt++;
                int z;
                do
                {
                    z = stack[top--];
                    dcc[cnt].push_back(z);
                } while (z != y);
                dcc[cnt].push_back(x);
            }
        }
        else
            low[x] = min(low[x], dfn[y]);
    }
}
void v_DCC()
{
    cnt = 0;
    top = 0;
    for (int i = 1; i <= n; i++)
    {
        if (!dfn[i])
            root = i, Tarjan(i);
    }
```

```
    // 给每个割点一个新的编号(编号从cnt+1开始)
    num = cnt;
    for (int i = 1; i <= n; i++)
        if (cut[i]) new_id[i] = ++num;
    // 建新图，从每个v-DCC到它包含的所有割点连边
    tc = 1;
    for (int i = 1; i <= cnt; i++)
        for (int j = 0; j < dcc[i].size(); j++)
        {
            int x = dcc[i][j];
            if (cut[x]) {
                add_c(i, new_id[x]);
                add_c(new_id[x], i);
            }
            else c[x] = i; // 除割点外，其它点仅属于1个v-DCC
        }
}
```

例题分析

POJ3694 Network（e-DCC 缩点，LCA，并查集）

POJ2942 Knights of the Round Table（补图，v-DCC，染色法奇环判定）

6.9.3 欧拉路问题

欧拉图的判定

无向图连通，所有点度数为偶数。

欧拉路的存在性判定

无向图连通，恰有两个节点度数为奇数，其他节点度数均为偶数

```
// 模拟系统栈，答案栈
void Euler() {
    stack[++top] = 1;
    while (top > 0) {
        int x = stack[top], i = head[x];
        // 找到一条尚未访问的边
        while (i && vis[i]) i = Next[i];
        // 沿着这条边模拟递归过程，标记该边，并更新表头
        if (i) {
            stack[++top] = ver[i];
            head[x] = Next[i];
            vis[i] = vis[i ^ 1] = true;
        }
        // 与x相连的所有边均已访问，模拟回溯过程，并记录于答案栈中
        else {
            top--;
            ans[++t] = x;
        }
    }
}
```

例题分析

POJ2230 Watchcow（欧拉回路）

6.10 Tarjan 算法与有向图连通性

6.10.1 强连通分量（SCC）判定法则

```
void Tarjan(int x)
{
    dfn[x]=low[x]=++num;
    stack[++top]=x,in_stack[x]=true;
    for(int i=head[x];i;i=nxt[i])
    {
        int y=ver[i];
        if(!dfn[y])
        {
            Tarjan(y);
            low[x]=min(low[x],low[y]);
        }
        else if(in_stack[y])
            low[x]=min(low[x],dfn[y]);
    }
    if(dfn[x]==low[x])
    {
        cnt++;
        int y;
        do
        {
            y=stack[top--],in_stack[y]=false;
            color[y]=cnt, scc[cnt].push_back(y);
        } while (x!=y);
    }
}
```

6.10.2 SCC -> DAG

```
void SCC()
{
    for (int i = 0; i <= n; i++)
        if (!dfn[i])
            Tarjan(i);
    //缩点
    for (int x = 1; x <= n; x++)
    {
        for (int i = head[x]; i; i = nxt[i])
        {
            int y = ver1[i];
            if (color[x] != color[y])
                add_c(color[x], color[y]);
        }
    }
}
```

例题分析

POJ1236 Network of Schools（SCC->DAG，入度出度）

P3275 [SCOI2011] 糖果（SPFA TLE，SCC->DAG，Topo，DP）

6.10.3 有向图的必经点与必经边

对于有向无环图（DAG）：

在原图中按照拓扑序进行动态规划，求出起点 S 到图中每个点 x 的路径条数 fs[x]。

在反图上再次按照拓扑序进行动态规划，求出每个点 x 到终点 T 的路径条数 ft[x]。

显然，fs[T] 表示从 S 到 T 的路径总条数。根据乘法原理：

1：对于一条有向边 (x,y)，若 fs[x]*ft[y]=fs[T]，则 (x,y) 是有向无环图从 S 到 T 的必经边。

2：对于一个点 x，若 fs[x]*ft[x]=fs[T]，则 x 是有向无环图从 S 到 T 的必经点。

路径条数规模较大，可对大质数取模后保存，但有概率误判。

例题分析

6703 PKU ACM Team's Excursion （DAG 必经边，枚举，DP）

6.10.4 2-SAT 问题

原命题与逆否命题互为等价命题，在建立有向边关系时要注意对称性。

例题分析

POJ3678 Katu Puzzle（2-SAT，Tarjan,，SCC）

POJ3683 Priest John's Busiest Day（2-SAT 方案构造）

```
for (int i = 1; i <= 2 * n; i++)
{
    val[i] = color[i] > color[opp[i]];
    //若c[i] 大于 c[opp[i]]   opp[i]赋0
}
```

6.11 二分图的匹配

6.11.1 二分图判定

一张无向图是二分图，当且仅当图中不存在奇环（长度为奇数的环）。

```
//染色法判定奇环
bool DFS(int x,int color)
{
    vis[x]=color;
    for(int i=head[x];i;i=nxt[i])
    {
        int y=ver[i];
        if(!vis[y])
        {
            if(!DFS(y,3-color)) return false;
        }
        else if(vis[y]==color) return false;
    }
    return true;
}
```

例题分析
P1525 关押罪犯（判定二分图，二分）

6.11.2 二分图最大匹配

二分图匹配的模型要素
1：节点能分成独立的两个集合，每个集合内部有 0 条边。“0 要素”
2：每个节点只能与 1 条匹配边相连。“1 要素”

```
\\匈牙利算法
\\在主函数中对每个左部节点调用寻找增广路时，需要对 vis 重置。
bool DFS(int x)
{
    for (int i = head[x]; i; i = nxt[i])
    {
        int y = ver[i];
        if (!vis[y])
        {
            vis[y] = 1;
            if (!match[y] || DFS(match[y]))
            {
                match[y] = x;
                return true;
            }
        }
    }
    return false;
}
```

例题分析
6801 棋盘覆盖（奇偶染色）
6802 車的放置（行列）
6803 导弹防御塔（二分，拆点多重匹配）

6.11.3 二分图带权匹配

二分图带权最大匹配的前提是匹配数最大，然后再最大化匹配边的权值总和。

```
/*KM 稠密图上效率高于费用流，但是有较大局限性，只能在满足“带权最大匹配一定是完备匹配”的图中正确求解。
w[][]:边权
la[], lb[]: 左，右部点顶标
visa[], visb[]: 访问标记，是否在交错树中
ans: Σw[match[i]][i]
*/
bool DFS(int x)
```

```
{
    visa[x] = true;
    for (int y = 1; y <= n; y++)
    {
        if (!visb[y])
        {
            double temp = fabs(la[x] + lb[y] - w[x][y]);//对于浮点数，相等子图的判定
            if (temp < eps)
            {
                visb[y] = true;
                if (!match[y] || DFS(match[y]))
                {
                    match[y] = x;
                    return true;
                }
            }
            else
                upd[y] = min(upd[y], la[x] + lb[y] - w[x][y]);
        }
    }
    return false;
}
void KM()
{
    for (int i = 1; i <= n; i++)
    {
        la[i] = -inf;
        lb[i] = 0;
        for (int j = 1; j <= n; j++)
            la[i] = max(la[i], w[i][j]);
    }
    for (int i = 1; i <= n; i++)
    {
        while (true)
        {
            memset(visa, 0, sizeof(visa));
            memset(visb, 0, sizeof(visb));
            for (int j = 1; j <= n; j++)
                upd[j] = inf;
            if (DFS(i))
                break;
            else
            {
                delta = inf;
                for (int j = 1; j <= n; j++)
                    if (!visb[j])
                        delta = min(delta, upd[j]);
                for (int j = 1; j <= n; j++)
                {
                    if (visa[j])
                        la[j] -= delta;
                    if (visb[j])
                        lb[j] += delta;
                }
            }
        }
    }
}
```

例题分析

POJ3565 Ants（三角形不等式，二分图带权最小匹配）

6.12 二分图的覆盖与独立集

6.12.1 二分图最小点覆盖

二分图最小覆盖模型特点:
每条边有 2 个端点，二者至少选择一个。“2 要素”

6.12.1.1 König’s theorem

二分图最小点覆盖包含的点数等于二分图最大匹配包含的边数。
例题分析
POJ1325 Machine Schedule（二分图最小覆盖）
POJ2226 Muddy Fields（行列连续块，二分图最小覆盖）

6.12.2 二分图最大独立集

无向图 G 的最大团等于其补图 G’ 的最大独立集。(补图转化)
设 G 是有 n 个节点的二分图，G 的最大独立集的大小等于 n 减去最大匹配数。
例题分析
6901 骑士放置（奇偶染色）

6.12.3 有向无环图的最小路径点覆盖

给定一张有向无环图，要求用尽量少的不相交的简单路径，覆盖有向无环图的所有顶点（也就是每个顶点恰好被覆盖一次）。这个问题被称为有向无环图的最小路径点覆盖，简称“最小路径覆盖”。

有向无环图 G 的最小路径点覆盖包含的路径条数，等于 n（有向无环图的点数）减去拆点二分图 G2 的最大匹配数。

若简单路径可相交，即一个节点可被覆盖多次，这个问题称为有向无环图的最小路径可重复点覆盖。

对于这个问题，可先对 G 求传递闭包，得到有向无环图 G’，再在 G’ 上求一般的（路径不可相交的）最小路径点覆盖。

例题分析
6902 Vani 和 Cl2 捉迷藏（最小路径可重复点覆盖，构造方案）

```cpp
// 构造方案，先把所有路径终点（左部非匹配点）作为藏身点
for (int i = 1; i <= n; i++) succ[match[i]] = true;
for (int i = 1, k = 0; i <= n; i++)
    if (!succ[i]) hide[++k] = i;
memset(vis, 0, sizeof(vis));
bool modify = true;
while (modify) {
    modify = false;
    // 求出 next(hide)
    for (int i = 1; i <= ans; i++)
        for (int j = 1; j <= n; j++)
            if (cl[hide[i]][j]) vis[j] = true;
    for (int i = 1; i <= ans; i++)
        if (vis[hide[i]]) {
            modify = true;
            // 不断向上移动
            while (vis[hide[i]]) hide[i] = match[hide[i]];
        }
}
for (int i = 1; i <= ans; i++) printf("%d ", hide[i]);
cout << endl;
```

6.13 网络流初步

6.13.1 最大流

6.13.1.1 Edmonds Karp 增广路

```
bool BFS() {
    memset(vis, 0, sizeof(vis));
    queue<int> q;
    q.push(S); vis[S] = 1;
    incf[S] = inf; // 增广路上各边的最小剩余容量
    while (q.size()) {
        int x = q.front(); q.pop();
        for (int i = head[x]; i; i = Next[i])
            if (edge[i]) {
                int y = ver[i];
                if (vis[y]) continue;
                incf[y] = min(incf[x], edge[i]);
                pre[y] = i; // 记录前驱，便于找到最长路的实际方案
                q.push(y), vis[y] = 1;
                if (y == t) return 1;
            }
    }
    return 0;
}
void Update() { // 更新增广路及其反向边的剩余容量
    int x = t;
    while (x != s) {
        int i = pre[x];
        edge[i] -= incf[t];
        edge[i ^ 1] += incf[t]; // 利用“成对存储”的xor 1技巧
        x = ver[i ^ 1];
    }
    maxflow += incf[t];
}
```

6.13.1.2 Dinic

```
//可加入当前弧优化（&）：在增广时复制head[]到cur[]，在增广时同步修改cur[]，目的是递归时跳过已增广的边。
bool BFS()
{
    memset(d, 0, sizeof(d));
    queue<int> q;
    q.push(S);
    d[S] = 1; //不为1  陷入死循环
    while (q.size())
    {
        int x = q.front();
        q.pop();
        for (int i = head[x]; i; i = nxt[i])
        {
            int y = ver[i];
            if (edge[i] && !d[y])
            {
                d[y] = d[x] + 1;
                q.push(y);
                if (y == T)
                    return true;
            }
        }
    }
    return false;
}
int Dinic(int x, int flow)
{
```

```
    if (x == T)
        return flow;
    int rest = flow, k;
    for (int i = head[x]; i && rest; i = nxt[i])
    {
        int y = ver[i];
        if (edge[i] && d[y] == d[x] + 1)
        {
            k = Dinic(y, min(edge[i], rest));
            if (!k)
                d[y] = 0;
            edge[i] -= k;
            edge[i ^ 1] += k;
            rest -= k;
        }
    }
    return flow - rest;
}
```

6.13.1.3 二分图最大匹配的必须边与可行边

在一般的二分图中，可以用最大流计算任一组最大匹配。

此时：必须边的判定条件为：(x,y) 流量为 1 ，并且在残量网络上属于不同的 SCC。

可行边的判定条件为：(x,y) 流量为 1 ，或者在残量网络上属于同一个 SCC。

例题分析

CH17C 舞动的夜晚（Dinic，Tarjan，二分图可行边）

6.13.2 最小割

6.13.2.1 最大流最小割定理

任何一个网络的最大流量等于最小割中边的容量之和。

例题分析

POJ1966 Cable TV Network （枚举，点边转化）

6.13.3 费用流

6.13.3.1 Edmonds Karp 增广路

BFS 寻找增广路 -> SPFA 寻找单位费用之和最小的增广路（将费用作为边权，在残量网络上求最短路）。

注意：反向边的费用为相反数。

```
bool SPFA()
{
    memset(dis, 0xcf, sizeof(dis));//-inf
    memset(vis, 0, sizeof(vis));
    queue<int> q;
    dis[S] = 0, vis[S] = 1, incf[S] = 1 << 30;
    q.push(S);
    while (q.size())
    {
        int x = q.front();
        q.pop();
        vis[x] = 0;
        for (int i = head[x]; i; i = nxt[i])
        {
            if (edge[i])
            {
                int y = ver[i];
                if (dis[y] < dis[x] + cost[i])
                {
                    dis[y] = dis[x] + cost[i];
                    incf[y] = min(incf[x], edge[i]);
                    pre[y] = i;
                    if (!vis[y])
                        q.push(y), vis[y] = 1;
```

```
                }
            }
        }
    }
    if (dis[T] == 0xcfcfcfcf)
        return false;
    return true;
}
int max_flow, ans;
void Update()
{
    int x = T;
    while (x != S)
    {
        int i = pre[x];
        edge[i] -= incf[T];
        edge[i ^ 1] += incf[T];
        x = ver[i ^ 1];
    }
    max_flow += incf[T];
    ans += incf[T] * dis[T];
}
```

例题分析

POJ3422 Kaka's Matrix Travels（点边转化，费用流）