# 8. 贪心

最优值变化趋势为单峰时才可用贪心.

## 8.1 区间问题

区间问题的贪心一般按区间的左端点排序或右端点排序或左右端点排序.实际应用中可尝试各种排序方法,举例子检验算法的合理性,并尝试证明该算法的正确性.

---

### 8.1.1 区间合并

#### 题意

给定$n$ $(1 \leq 1e5)$个区间$[l_i, r_i]$ $(-1e9 \leq l_i \leq r_i \leq 1e9)$,合并所有有交集的区间,两区间在端点处相交也视为有交集.输出合并完成后的区间个数.

#### 思路

贪心,先按区间左端点升序排序,然后扫一遍,把有交集的区间合并.

合并时,每次维护一个以$start$为起点、以$end$为终点的区间.假设更新到第$i$个区间,则第$(i+1)$个区间与该区间的关系有3种:

①内含:不用更新.

②相交:第$(i+1)$个区间只会在第$i$个区间的右边,则更新$end_i = \max\{end_i, end_{i+1}\}$.

③相离:将第$i$个区间放入答案,更新维护的区间为第$(i+1)$个区间.

#### 代码

```cpp
void merge(vector<PII>& segs) {
  vector<PII> res;  // 存答案

  sort(segs.begin(), segs.end());  // 默认按左端点排序

  int start = -2e9, end = -2e9;
  for (auto seg : segs) {
    if (end < seg.first) {
      if (start != -2e9)  // 不是最开始的区间
        res.push_back({ start,end });
      start = seg.first, end = seg.second;
    }
    else end = max(end, seg.second);
  }

  if (start != -2e9)  // 防止res数组为空
    res.push_back({ start,end });

  segs = res;
}

int main() {
```

```
  vector<PII> segs;
  while (n--) {
    int l, r; cin >> l >> r;
    segs.push_back({ l,r });
  }

  merge(segs);
  cout << segs.size();
}
```

## 8.1.2 区间选点

### 题意

给定$N$ $(1 \leq N \leq 1e5)$个闭区间$[a_i, b_i]$ $(-1e9 \leq a_i \leq b_i \leq 1e9)$,在数轴上选择尽量少的点使得每个区间至少包含一个选出的点,在区间端点上的点也视为在区间内.输出选择的点的最少数量.

### 思路

先按区间右端点升序排序,再从左往右枚举每个区间.若当前区间已包含点,则pass;否则,显然所选的点为当前区间的右端点最优.下面证明这样的做法能得到最优解$ans$:

①显然上述选法可让每个区间都包含一个点,即为一组合法的方案,设它选了$cnt$个点.因最优解是所有合法方案中的最小值,则$ans \leq cnt$.

②考察所有没被pass的区间.第一个区间必被选,所选的点为其右端点.因按右端点升序排序,则下一个与其无交集的区间在其右边.最后选了$cnt$个点,即$cnt$个做右端点依次递增且两两无交的区间.为保证每个区间内都有点,则至少需$cnt$个点,进而$ans \geq cnt$.

综上,$ans = cnt$.

### 代码

```
const int MAXN = 1e5 + 5;
struct Seg {
  int l, r;

  bool operator < (const Seg& a)const { return r < a.r; }  // 按右端点升序排序
}segs[MAXN];

int main() {
  for (int i = 0; i < n; i++) cin >> segs[i].l >> segs[i].r;

  sort(segs, segs + n);

  int ans = 0;
  int end = -2e9;  // 当前区间右端点,初始时没选,置为负无穷
  for (int i = 0; i < n; i++) {
    if (segs[i].l > end) {
      ans++;
      end = segs[i].r;
    }
  }
  cout << ans;
}
```

## 8.1.3 最大不相交区间数量

### 题意

给定$N$ $(1 \leq N \leq 1e5)$个闭区间$[a_i, b_i]$ $(-1e9 \leq a_i \leq b_i \leq 1e9)$,在数轴上选择尽量多的区间使得它们两两不交(含端点).输出可选取的区间的最大数量.

实际应用:会议室数量有限,在保证会议不冲突的前提下安排尽量多的会议.

### 思路

类似于**8.1.2**,先按区间右端点升序排序,再从左往右枚举每个区间.若当前区间已包含点,则pass;否则,选择该区间的右端点.下面证明最优解$ans$即选择的点的数量$cnt$:

①选出的区间两两不交,即是一种可行方案.而$ans$是所有可行方案中的最大值,则$ans \geq cnt$.

②选出的每个区间中都有一个点.反证,若$ans > cnt$,即可选出比$cnt$个更多的两两不交的区间,则至少需选择$ans$个点才能覆盖所有区间,而事实上只选择$cnt$个点即可覆盖所有区间,矛盾.故$ans \leq cnt$.

综上,$ans = cnt$.

### 代码

```cpp
const int MAXN = 1e5 + 5;
struct Seg {
  int l, r;

  bool operator < (const Seg& a)const { return r < a.r; }  // 按右端点升序排序
}segs[MAXN];

int main() {
  for (int i = 0; i < n; i++) cin >> segs[i].l >> segs[i].r;

  sort(segs, segs + n);

  int ans = 0;
  int end = -2e9;  // 当前区间右端点,初始时没选,置为负无穷
  for (int i = 0; i < n; i++) {
    if (segs[i].l > end) {
      ans++;
      end = segs[i].r;
    }
  }
  cout << ans;
}
```

## 8.1.4 区间分组

## 题意

给定$N$ $(1 \leq N \leq 1e5)$个闭区间$[a_i, b_i]$ $(-1e9 \leq a_i \leq b_i \leq 1e9)$.将这些区间分成尽量少的若干组,使得每组内的区间两两不交(含端点).输出组数的最小值.

## 思路

先将所有区间按左端点升序排序,再从左往右枚举每个区间,判断其是否能放到某个现有的组中,即判断某一组中最靠右的右端点是否在当前区间的左端点的右边:

①若不存在与当前区间不交的组,则开一个新组.

②若存在与当前区间不交的组,则将该区间放入该组,并更新该组右端点的最大值.

下证上述分法的组数$cnt$即最优解$ans$:

①上述分法得到的组中区间两两不交,即是一种可行方案,而$ans$是所有可行方案中的最小值,则$ans \leq cnt$.

②假设已分$(cnt-1)$个组,要分第$cnt$个组时,当前区间的左端点小于前$(cnt-1)$个组的右端点的最大值,即当前区间与前$(cnt-1)$个组中的区间都有交集.为保证每组内区间两两不交,至少需$cnt$个区间,即$ans \geq cnt$.

综上,$ans = cnt$.

为判断是否存在一个组的右端点的最大值小于当前区间的左端点,可用小根堆维护,即若所有组的$max_right$的最小值＜当前区间的左端点,即存在一个与当前区间不交的组.

## 代码

```cpp
const int MAXN = 1e5 + 5;
struct Seg {
  int l, r;

  bool operator < (const Seg& a)const { return l < a.l; }  // 按右端点升序排序
}segs[MAXN];

int main() {
  for (int i = 0; i < n; i++) cin >> segs[i].l >> segs[i].r;

  sort(segs, segs + n);

  priority_queue<int, vector<int>, greater<int>> heap;  // 小根堆
  for (int i = 0; i < n; i++) {
    auto cur = segs[i];

    if (heap.empty() || heap.top() >= cur.l)  // 不用开新组
      heap.push(cur.r);
    else {
      heap.pop();
      heap.push(cur.r);
    }
  }
  cout << heap.size();
}
```

# 8.1.5 区间覆盖

## 题意

给定$N$ $(1 \leq N \leq 1e5)$个闭区间$[a_i, b_i]$ $(-1e9 \leq a_i \leq b_i \leq 1e9)$和一个线段区间$[s, t]$ $(-1e9 \leq s \leq t \leq 1e9)$,选出尽量少的区间将该线段覆盖.若能覆盖,输出选择的最小区间数,否则输出$-1$.

## 思路

先将所有区间按左端点升序排序,再从左往右枚举每个区间,选择能覆盖线段左端点的区间中右端点最大的区间,更新$s$为右端点的最大值.下证该选法选出的区间数$cnt$即最优解$ans$,只考虑有解的情况:

将$ans$和$cnt$的选法分别按区间左端点排序,找到第一个不同的区间,则$ans$中该区间的右端点小于其在$cnt$中的右端点,则可将$ans$中的该区间延长为$cnt$中的该区间,这样替换后仍为最优解,且区间数量不变.同理替换后续不同的区间直至将$ans$的选法变为$cnt$的选法,故$ans = cnt$.

扫描每个区间可用双指针.

需要开一个bool变量来记录是否有解.

## 代码

```cpp
const int MAXN = 1e5 + 5;
struct Seg {
  int l, r;

  bool operator < (const Seg& a)const { return l < a.l; }  // 按右端点升序排序
}segs[MAXN];

int main() {
  for (int i = 0; i < n; i++) cin >> segs[i].l >> segs[i].r;

  sort(segs, segs + n);

  int ans = 0;
  bool ok = false;  // 记录是否有解
  for (int i = 0; i < n; i++) {
    int j = i, right = -2e9;  // 当前区间的右端点,初始时未选,置为负无穷
    while (j < n && segs[j].l <= s) {  // 找出能覆盖线段的左端点的区间中右端点最大的区间
      right = max(right, segs[j].r);
      j++;
    }

    if (right < s) {  // 找不到一个能覆盖线段左端点的区间,即无解
      ans = -1;
      break;
    }

    ans++;
    if (right >= t) {  // 找到一组解
      ok = true;
      break;
    }

    s = right;  // 将线段左端点更新为区间右端点的最大值
    i = j - 1;
  }
  if (!ok) ans = -1;
```

```
  cout << ans;
}
```

# 8.2 Huffman树

Huffman树是二叉树.

## 8.2.1 合并果子

### 题意

某人将果子按不同种类分成$n$ $(1 \le n \le 1e4)$堆,第$i$ $(1 \le i \le n)$堆有$a_i$ $(1 \le a_i \le 2e4)$个果子.现要将它们合成一堆,每次可合并两堆果子,消耗的体力等于两堆果子重量之和.显然经$(n-1)$次合并后只剩一堆,消耗的总体力等于每次合并消耗的体力之和.假设每个果子质量都为1,且已知果子的种类和每种果实的数目,设计一个合并方案使得消耗的体力最少,输出最小耗费体力(不超过$2^{31}$).

### 思路

显然总体力值是各节点的权值乘上它与根节点的距离之和.

每次合并最小的两堆,下证其正确性:

(1)先证最小的两堆对应的节点在树中深度最深,且可能互为兄弟:

①显然可能互为兄弟.

②若要合并的两堆之一对应的节点深度不是最深的,则可换一个深度更深的节点使得总体力值减小.

(2)再证可由局部最优解得到全局最优解.$n$堆果子合并其中最小的两堆$a$、$b$得到$(n-1)$堆果子,这是$n$堆果子的最优解$F(n)$;$(n-1)$堆果子再合并其中最小的两堆果子,这是$(n-1)$堆果子的最优解$F(n-1)$.下证$(n-1)$堆果子的最优解也是$n$堆果子的最优解.

考察将$(n-1)$堆果子合并成1堆的某个方案$f(n-1)$,它对应的$n$堆果子合并成1堆的某个方案$f(n) = f(n-1) + a + b$,则$F(n) = \min f(n)$.因每个$f(n)$都有$(a+b)$,则只需使$f(n-1)$最小,即合并$(n-1)$堆果子中最小的两堆.

综上,$(n-1)$堆果子的最优解也是$n$堆果子的最优解.

每次合并最小的两堆可用小根堆维护.

### 代码

```
int main() {
  priority_queue<int, vector<int>, greater<int>> heap;
  while (n--) {
    int x; cin >> x;
    heap.push(x);
  }

  int ans = 0;
  while (heap.size() > 1) {
    int a = heap.top(); heap.pop();
    int b = heap.top(); heap.pop();
    ans += a + b;
```

```
    heap.push(a + b);
  }
  cout << ans;
}
```

# 8.3 排序不等式

## 8.3.1 排队打水

### 题意

$n$ $(1 \leq n \leq 1e5)$个人排队在1个水龙头处打水,第$i$个人装满水所需时间为$t_i$ $(1 \leq t_i \leq 1e4)$.如何安排打水次序使得所有人等待时间之和最小,输出最小等待时间.

### 思路

总等待时间$= t_1 \times (n-1) + t_2 \times (n-2) + \cdots + t_{n-1} \times 1$.注意到每个$t_i$乘的数随$i$的增大而减小,为使总等待时间最小,应让$t_i$最大的乘最小的数.

下证按打水时间从小到大排队时总等待时间最短:

假设最优解不是按从小到大排序,则$\exists t_i > t_{i+1}$,此时$t_i$和$t_{i+1}$对总等待时间的贡献是$s_1 = t_i \times (n-i) + t_{i+1} \times (n-i-1)$.交换$t_i$和$t_{i+1}$,此时$t_i$和$t_{i+1}$对总等待时间的贡献是$s_2 = t_{i+1} \times (n-i) + t_i \times (n-i-1)$.

$s_1 - s_2 = (2n-2i-1)(t_i - t_{i+1}) > 0$ $(1 \leq i \leq n-1)$, 即存在总等待时间更小的方案,与该方案是最优解矛盾.

故按打水时间从小到大排队时总等待时间最短.

若$t_i = 1$,则总等待时间最多$1 + 2 + \cdots + 9999$,故要开long long.

### 代码

```
const int MAXN = 1e5 + 5;
int t[MAXN];

int main() {
  for (int i = 0; i < n; i++) cin >> t[i];

  sort(t, t + n);

  ll ans = 0;
  for (int i = 0; i < n; i++) ans += (ll)(t[i] * (n - i - 1));
  cout << ans;
}
```

## 8.3.2 魔术券

### 题意

商店的优惠券上印有整数$N$(可能是负数).若将优惠券用于产品,商店会给你$N$倍的产品价值的前.此外,商店免费提供一些赠品.若将优惠券用于赠品,你需向商店支付$N$倍的赠品价值的钱.现给定若干优惠券和若干商品,每个优惠券和商品至多只能选择一起,问最多能从商店拿回多少钱.

第一行输入一个整数$N_c$ $(1 \leq N_c \leq 1e5)$表示优惠券数量.第二行输入$N_c$个整数(绝对值不超过$100$),表示各优惠券上印的数值.第三行输入一个整数$N_p$ $(1 \leq N_p \leq 1e5)$表示产品数.第四行包含$N_p$个整数(绝对值不超过$100$),表示各产品的价值,其中商品的价值为正,赠品的价值为负.

### 思路

显然没必要选$N = 0$的券,应选正数乘正数或负数乘负数的组合.对负数乘负数等价于都取绝对值后相乘,故只需考虑正数乘正数,排序不等式秒.

实现时先将两序列升序排列.对正数的部分,用两个指针从序列的末尾往前移,直至一个指针指向的数非正数;对负数的部分,用两个指针从序列的开头往后移,直至一个指针指向的数非负数.

最坏每个数都是$100$,则最多$1e5 \times 100^2 = 1e9$,不会爆int.

### 代码

```cpp
const int MAXN = 1e5 + 5;
int n, m;  // 序列长度
int a[MAXN], b[MAXN];

int main() {
  cin >> n;
  for (int i = 0; i < n; i++) cin >> a[i];
  cin >> m;
  for (int i = 0; i < m; i++) cin >> b[i];

  sort(a, a + n), sort(b, b + m);

  int ans = 0;
  for (int i = 0, j = 0; i < n && j < m && a[i] < 0 && b[j] < 0; i++, j++)
    ans += a[i] * b[j];
  for (int i = n - 1, j = m - 1; i >= 0 && j >= 0 && a[i] > 0 && b[j] > 0; i--, j--)
    ans += a[i] * b[j];
  cout << ans;
}
```

# 8.4 绝对值不等式

# 8.4.1 货仓选址

## 题意

数轴上有$N$ $(1 \leq N \leq 1e5)$个商店,坐标分别为$x_1, \cdots, x_N$ $(0 \leq x_i \leq 4e4)$.现要在数轴上建一个仓库使得仓库到每个商店的距离之和最小.

## 思路

设仓库坐标为$x$,则总距离$d = |x_1 - x| + |x_2 - x| + \cdots + |x_n - x|$.

将和数分组:$d = (|x_1 - x| + |x_n - x|) + (|x_2 - x| + |x_{n-1} - x|) + \cdots$.

只看其中两项,即求函数$f(x) = |a - x| + |b - x|$的最小值.不妨设$a \leq b$,则显然$x \in [a, b]$时$f$取得最小值,故$d \geq (x_n - x_1) + (x_{n-1} - x_2) + \cdots$,假设其中$x_n > x_1, x_{n-1} > x_2, \cdots$.

下证能取等,只需证每一项都能取等:第一项取等条件$x \in [x_1, x_n]$,第二项取等条件$x \in [x_2, x_{n-1}], \cdots$,则将数组$x_i$升序排序后可取等.

## 代码

```cpp
const int MAXN = 1e5 + 5;
int x[MAXN];

int main() {
  int n; cin >> n;
  for (int i = 0; i < n; i++) cin >> x[i];

  sort(x, x + n);

  int ans = 0;
  for (int i = 0; i < n; i++) ans += abs(x[i] - x[n / 2]);
  cout << ans;
}
```

---

# 8.4.2 七夕祭

## 题意

给定由$n \times m$个摊点组成的会场,摊点有不同种类.某人只对其中的一部分摊点感兴趣,他希望通过适当的调整使得各行中他感兴趣的摊点数一样多,各列中他感兴趣的摊点数也一样多.现可进行若干次操作,每次操作可交换相邻的两摊点.两摊点相邻当且仅当它们位于同一行或同一列的相邻位置上,规定每一行或每一列的第一个位置与最后一个位置相邻.问他的两个要求最多能满足几个,并求在此前提下,最少需进行多少次操作.

第一行输入三个整数$n, m, t$ $(1 \leq n, m \leq 1e5, 0 \leq t \leq \min\{n \times m, 1e5\})$,分别表示会场大小和某人感兴趣的摊点数.接下来$t$行每行输入两整数$x, y$ $(1 \leq x \leq n, 1 \leq y \leq m)$,表示某人对摊点$(x, y)$感兴趣.

若某人的两个要求都能满足,则输出"both";若只能满足行的要求,输出"row";若只能满足列的要求,输出"column";若都不能满足,输出"impossible".若答案非"impossible",输出最少交换次数.

## 思路

行内交换对行感兴趣的摊点数无影响,对列的影响是交换所在行中相邻两列的感兴趣的摊点.显然对行的操作和对列的操作独立,可先行内交换使得满足列的要求,再列内交换使得满足行的要求,两种交换次数都最小时总交换次数最小.

显然每行和每列可视为一个环,某人感兴趣的摊点可视为糖果,每次操作可选择环上的一个有糖果的点,将糖果传递给左边或右边的点.设环上的点按顺时针依次为$a_1,a_2,\cdots,a_n$,其中$a_n$与$a_1$相邻.逆时针,设$a_1$给$a_n$的糖果数为$x_1$,$a_2$给$a_1$的糖果数为$x_2,\cdots,a_3$给$a_2$的糖果数为$x_3,\cdots,a_n$给$a_{n-1}$的糖果数为$x_{n-1}$,其中$x_i\ \ (i=1,\cdots,n)$可正可负,其中负表示反着给.最小操作步数为$|x_1|+|x_2|+\cdots+|x_n|$.

因糖果守恒,则最优情况是每个点的糖果数都是总糖果数的平均数$avg$,则$\begin{cases}a_1-x_1+x_2=avg\\a_2-x_2+x_3=avg\\\cdots\\a_{n-1}-x_{n-1}+x_n=avg\\a_n-x_n+x_1=avg\end{cases}$,注意到左右两边分别相加得到恒等式,则其中最多只有$(n-1)$个独立的方程,该方程组无唯一解,不妨取$x_1$为自由变元.由$\begin{cases}x_1-x_2=a_1-avg\\x_2-x_3=a_2-avg\\\cdots\\x_{n-1}-x_n=a_{n-1}-avg\\x_n-x_1=a_n-avg\end{cases}$解得$\begin{cases}x_1=x_1-0\\x_2=x_1-(a_1-avg)\\x_3=x_1-(a_1+a_2-2avg)\\\cdots\\x_n=x_1-[a_1+a_2+\cdots+a_{n-1}-(n-1)avg]\end{cases}$.设$\begin{cases}c_1=0\\c_2=a_1-avg\\c_3=a_1+a_2-2avg\\\cdots\\c_n=a_1+a_2+\cdots+(n-1)avg\end{cases}$,则$|x_1|+|x_2|+\cdots+|x_n|=|x_1-c_1|+|x_2-c_2|+\cdots+|x_n-c_n|$,取中位数即可使其最小.

注意操作次数可能爆int.

## 代码

```cpp
const int MAXN = 1e5 + 5;
int n, m, t;  // 会场大小、感兴趣的摊点数
int row[MAXN], col[MAXN];  // 每行、每列中感兴趣的摊点数
int pre[MAXN];  // row[]或col[]的前缀和
int c[MAXN];

ll solve(int n, int a[]) {
  for (int i = 1; i <= n; i++) pre[i] = pre[i - 1] + a[i];

  if (pre[n] % n) return -1; // 无解
  int avg = pre[n] / n;

  c[1] = 0;
  for (int i = 2; i <= n; i++) c[i] = pre[i - 1] - (i - 1) * avg;

  sort(c + 1, c + n + 1);
  ll res = 0;
  for (int i = 1; i <= n; i++) res += (ll)abs(c[i] - c[(n + 1) / 2]);
  return res;
}

int main() {
  cin >> n >> m >> t;
  while (t--) {
```

```
    int x, y; cin >> x >> y;
    row[x]++, col[y]++;
  }

  ll r = solve(n, row), c = solve(m, col);
  if (~r && ~c) cout << "both " << r + c;
  else if (~r) cout << "row " << r;
  else if (~c) cout << "column " << c;
  else cout << "impossible";
}
```

# 8.5 排序

## 8.5.1 耍杂技的牛

### 题意

有编号分别为$1\sim N$的$N$ $(1\le N\le 5e4)$头牛,它们各自有自己的重量$W_i$ $(1\le W_i\le 1e4)$和强壮程度$S_i$ $(1\le S_i\le 1e9)$.现它们要叠罗汉,即站在彼此的身上形成垂直堆叠.一头牛撑不住的可能性取决于它之上所有牛的总重量(不含它自己)与其强壮程度之差,称该数为风险值.求一个牛的排序,使得所有牛的风险值之和最小,输出风险值的最小值.

### 思路

考察一个排序,从上到下编号依次为$1,\cdots,i,i+1,\cdots,n$.考察交换第$i$和第$(i+1)$头牛前后它们的风险值:

| 编号 | 交换前 | 交换后 |
| --- | --- | --- |
| $i$ | $W_1+\cdots+W_{i-1}-S_i$ | $W_1+\cdots+W_{i-1}+W_{i+1}-S_i$ |
| $i+1$ | $W_1+\cdots+W_i-S_{i+1}$ | $W_1+\cdots+W_{i-1}-S_{i+1}$ |

各风险值都有$W_1+\cdots+W_{i-1}$,可全部去掉,即比较:

| 编号 | 交换前 | 交换后 |
| --- | --- | --- |
| $i$ | $-S_i$ | $W_{i+1}-S_i$ |
| $i+1$ | $W_i-S_{i+1}$ | $-S_{i+1}$ |

考察何时有$W_i-S_{i+1}>W_{i+1}-S_i$,即$W_i+S_i>W_{i+1}+S_{i+1}$,此时$\max(-S_i,W_i-S_{i+1})>\max(W_{i+1}-S_i,-S_{i+1})$,这表明:若$W_i+S_i>W_{i+1}+S_{i+1}$,则交换第$i$和第$(i+1)$头牛可使风险值降低.

故按$W_i+S_i$的值升序排列时总风险值最低,否则存在逆序,则可交换使得风险值降低.

### 代码

```
const int MAXN = 5e4 + 5;
PII cow[MAXN];

int main() {
  for (int i = 0; i < n; i++) {
    int w, s; cin >> w >> s;
    cow[i] = { w + s,w };  // pair默认按first作为第一关键字排序
```

```
  }

  sort(cow, cow + n);

  int ans = -INF, sum = 0;  // 注意ans可能为负
  for (int i = 0; i < n; i++) {
    int w = cow[i].second, s = cow[i].first - cow[i].second;
    ans = max(ans, sum - s);
    sum += w;
  }
  cout << ans;
}
```

## 8.5.2 排成最小数

### 题意 ($0.2$ s)

给定一个数组,其中包含若干个非负整数,整数可能含前导零.现需将所有数字拼起来形成一个新的数,使得该数尽可能小.

先输入一个$n$ $(1 \le n \le 1e4)$,表示数组中元素的个数.接下来输入$n$个非负整数,每个数字不超过8位,可能包含前导零.

输出能排列出的最小数,不输出前导零.

### 思路

定义字符串的小于等于号:$str1 \le str2 \Leftrightarrow str1 + str2 \le str2 + str1$,其中后者比较字典序.

需验证这样定义的小于等于号可用于排序,即证明这种二元关系是全序关系.

[**证**] (完全性) 两字符串的字典序要么$\ge$,要么$\le$,显然成立.

(反对称性) $\begin{cases} s_1 \le s_2 \Leftrightarrow s_1 + s_2 \le s_2 + s_1 \\ s_2 \le s_1 \Leftrightarrow s_2 + s_1 \le s_1 + s_2 \end{cases} \Rightarrow s_1 + s_2 = s_2 + s_1$,这正是$s_1 = s_2$的定义.

(传递性) 设$s_1, s_2, s_3$的长度分别为$x, y, z$,且$s_1 \le s_2, s_2 \le s_3$.欲证$s_1 \le s_3$.

$s_1 \le s_2 \Leftrightarrow s_1 + s_2 \le s_2 + s_1$,因$\text{len}(s_1 + s_2) = \text{len}(s_2 + s_1)$,则字典序关系与数值关系等价,

即$\Leftrightarrow s_1 \times 10^y + s_2 \le s_2 \times 10^x + s_1 \Leftrightarrow s_1(10^y - 1) \le s_2(10^x - 1) \Leftrightarrow \dfrac{s_1}{s_2} \le \dfrac{10^x - 1}{10^y - 1} \cdots$①.

同理$s_2 \le s_3 \Leftrightarrow \dfrac{s_2}{s_2} \le \dfrac{10^y - 1}{10^z - 1} \cdots$②.

欲证式$s_1 \le s_3 \Leftrightarrow \dfrac{s_1}{s_2} \le \dfrac{10^x - 1}{10^z - 1}$.因①和②两边都为正,相乘即证.

用该定义的小于号对输入的字符串升序排列后拼在一起的数最小.

[**证**] 若不然,设最优解的排序为$s_1, s_2, \cdots, s_n$,且其中至少$\exists$一对$s_i > s_{i+1}$.

由小于等于号的定义:$s_i + s_{i+1} > s_{i+1} + s_i$.

考察$s_1 + \cdots + s_i + s_{i+1} + \cdots + s_n$与$s_1 + \cdots + s_{i+1} + s_i + \cdots + s_n$,

两者只有$s_i$和$s_{i+1}$的部分不同,而$s_i + s_{i+1} > s_{i+1} + s_i$,这表明前者大于后者,故前者非最优解,矛盾.

### 代码

```cpp
const int MAXN = 1e5 + 5;
int n;
string str[MAXN];

int main() {
  cin >> n;
  for (int i = 0; i < n; i++) cin >> str[i];

  sort(str, str + n, [](string s1, string s2) {
    return s1 + s2 < s2 + s1;
    });

  string ans = "";
  for (int i = 0; i < n; i++) ans += str[i];
  int idx = 0;  // 有效数字起始下标
  while (idx + 1 < ans.size() && ans[idx] == '0') idx++;  // 去除前导零
  cout << ans.substr(idx);
}
```

---

## 8.5.3 月饼

### 题意 $(0.15\ \mathrm{s})$

给定各种月饼的存量和价值,以及市场对月饼的需求量,求最大利润.

第一行输入整数$n, d$ $(1 \leq n \leq 1000, 1 \leq d \leq 500)$,分别表示月饼种类和市场对月饼的需求量.第二行输入$n$个整数,表示每种月饼的存量,每种月饼的存量不超过$200$.第三行输入$n$个整数,表示每种月饼的价值,总价值不超过$2000$.

输出最大利润,保留两位小数.

### 思路

求出每种月饼的性价比,优先选性价比高的月饼.

### 代码

```cpp
const int MAXN = 1005;
int n;  // 月饼种类数
double d;  // 市场需求量
struct Cake {
  double price, weight;

  bool operator<(const Cake& p)const { return price / weight > p.price / p.weight; }
}cakes[MAXN];

int main() {
  cin >> n >> d;
  for (int i = 0; i < n; i++) cin >> cakes[i].weight;
  for (int i = 0; i < n; i++) cin >> cakes[i].price;

  sort(cakes, cakes + n);

  double ans = 0;
```

```
  for (int i = 0; i < n && d > 0; i++) {
    double w = min(d, cakes[i].weight);
    d -= w;
    ans += cakes[i].price / cakes[i].weight * w;
  }
  cout << fixed << setprecision(2) << ans;
}
```

## 8.5.4 整数集合划分

### 题意

假设集合可包含重复元素.给定一个含$n$ $(2 \leq n \leq 1e5)$个正整数的集合,将其划分为两集合$A_1$、$A_2$,其中$A_1$包含$n_1$个元素,$A_2$包含$n_2$个元素,且$A_1$中的元素之和为$s_1$,$A_2$中的元素之和为$s_2$,要求在$|n_1 - n_2|$尽可能小的基础上$|s_1 - s_2|$尽可能大.数据保证集合中任意元素和所有元素之和小于$2^{31}$.

输出$|n_1 - n_2|$和$|s_1 - s_2|$.

### 思路

为使得$|n_1 - n_2|$尽可能小,当$n$为偶数时,两集合各分一半;当$n$为奇数时,一个集合在分一半的基础上多一个元素.

为使得$|s_1 - s_2|$尽可能大,亦即让$|s_2 - s_1|$尽可能大,可先将序列升序排列,$A_1$选前$\left\lfloor \frac{n}{2} \right\rfloor$个,$A_2$选后$\left\lfloor \frac{n}{2} \right\rfloor$个($n$为偶数时)或后$\left( \left\lfloor \frac{n}{2} \right\rfloor + 1 \right)$个.

### 代码

```
const int MAXN = 1e5 + 5;
int n;
int a[MAXN];

int main() {
  cin >> n;
  for (int i = 0; i < n; i++) cin >> a[i];

  sort(a, a + n);
  int s1 = 0, s2 = 0;
  for (int i = 0; i < n / 2; i++) s1 += a[i];
  for (int i = n / 2; i < n; i++) s2 += a[i];
  cout << n % 2 << ' ' << s2 - s1;
}
```

## 8.5.5 结绳

### 题意 ($0.2$ s)

给定若干段绳子,将它们串成一条绳.如下图,每次串连时将两段绳子对折后套接在一起,将这样得到的绳子视为另一段绳子,可再次对折并与另一段绳子串连.每次串连后,两段绳子长度减半.给定绳子的长度,求它们能串成的绳子的最大长度.

第一行输入整数$n$ $(2 \leq n \leq 1e4)$,表示绳子段数.第二行输入$n$个正整数,表示绳子的原始长度,所有绳子的初始长度不超过$1e4$.

输出能串连成的绳子的最大长度,答案向下取整.

## 思路

串连长度为$a$和$b$的两绳子得到的新绳长度为$\frac{a+b}{2}$.

Huffman树,将每段绳子视为两叶子节点,将它们串连得到父节点.显然最优解用到所有绳子,则最后的绳子是根节点对应的绳子.显然两棵二叉树结构相同时,最后得到的绳子长度相等,则长度为$l$的绳子对最终绳子长度的贡献取决于绳子对应的节点与根节点的距离$d$,贡献为$\frac{l}{2^d}$.

最优解是将长的绳子放在深度小的位置,短的绳子放在深度大的位置.

[**证**] 设绳长$z>y$.考虑交换两绳子对应的节点的位置,显然同深度节点交换不改变最终绳长.

若$z$在上,$y$在下,则它们对最终绳长的贡献为$\left(\frac{y}{\text{大}}+\frac{z}{\text{小}}\right)$.

若$y$在上,$z$在下,则它们对最终绳长的贡献为$\left(\frac{y}{\text{小}}+\frac{z}{\text{大}}\right)$.

$$\frac{y}{\text{大}}+\frac{z}{\text{小}}-\frac{y}{\text{小}}-\frac{z}{\text{大}}=\frac{1}{\text{大}}(y-z)-\frac{1}{\text{小}}(y-z)=\left(\frac{1}{\text{大}}-\frac{1}{\text{小}}\right)(y-z)>0.$$

最优策略是每次串连所有绳子中长度最短的两条.

设$x,y$是当前最短、次短的两段绳子,注意到$\frac{x+y}{2}\le y$,故每次串连当前最短和次短的两段绳子得到的绳子都是当前最短的绳子,故无需用堆.

## 代码

```cpp
const int MAXN = 1e5 + 5;
int n;
double a[MAXN];

int main() {
  cin >> n;
  for (int i = 0; i < n; i++) cin >> a[i];

  sort(a, a + n);

  for (int i = 1; i < n; i++) a[0] = (a[0] + a[i]) / 2;  // 后面的绳子合并到最短的绳子上
  cout << (int)a[0];
}
```

---

# 8.5.6 是否加油

## 题意 (0.2 s)

高速公路上有若干个加油站,不同加油站的油价可能不同.假设初始时油箱为空.

第一行输入四个整数$c_{max}, d, d_{avg}, n$ $(1 \le c_{max} \le 100, 1 \le d \le 3e4, 1 \le d_{avg} \le 20, 1 \le n \le 500)$,分别表示油箱的最大容量、起点到目的地的距离、每单位汽油可供行驶的距离、加油站总数.接下来$n$行每行输入一队非负数$p_i, d_i$ $(1 \le p_i \le 100, 0 \le d_i \le d)$,分别表示每升汽油的价格、该加油站与起点的距离.

若能到达目的地,输出到达目的地的最小花费,答案保留两位小数.若无法到达目的地,输出"The maximum travel distance = $x$",其中$x$为到达的最远距离,答案保留两位小数.

## 思路

因初始时油箱为空,若起点处无加油站,则无解.

每次加满油可行驶$c_{max} \times d_{avg}$的路程

从起点出发,在距离起点$c_{max} \times d_{avg}$的距离内找到第一个比起点处的加油站的油价低的加油站$A$(若存在),则在起点处的加油站加的油只需足以行驶到该$A$,再在$A$处加油,以$A$为起点继续行驶;若距离起点$c_{max} \times d_{avg}$的距离内无比起点处的加油站的油价更低的加油站,则选择在距离内油价最低的加油站$B$处加油,但未必要加满,而是以该加油站为起点继续考察航程内是否有比$B$的油价更低的加油站.

若中途的某个起点处的航程内无加油站,则无解.

将目的地视为一个油价为0的加油站.

## 代码

```cpp
const int MAXN = 505;
int c_max, d, d_avg, n;  // 油箱容量、起点到终点的距离、每升油能行驶的距离、加油站数
struct Stop {
  double price, dis;

  bool operator<(const Stop& p)const { return dis < p.dis; }
}stops[MAXN];

int main() {
  cin >> c_max >> d >> d_avg >> n;
  for (int i = 0; i < n; i++) cin >> stops[i].price >> stops[i].dis;
  stops[n] = { 0,(double)d };  // 终点视为油价为0的加油站

  sort(stops, stops + n + 1);

  if (stops[0].dis)  // 起点处无加油站,则无解
    return cout << "The maximum travel distance = 0.00", 0;

  double ans = 0, oil = 0;  // 花费、当前油量
  for (int i = 0; i < n;) {
    int k = -1;  // 记录下一个加油站的下标
    double max_dis = c_max * d_avg;
    for (int j = i + 1; j <= n && cmp(max_dis, stops[j].dis - stops[i].dis) >= 0; j++) {
      if (stops[j].price < stops[i].price) {  // 存在比起点加油站油价更低的加油站
        k = j;
        break;
      }
```

```
      else if (k == -1 || stops[j].price < stops[k].price)  // 不存在比起点加油站油价更低的加油站
        k = j;  // 找油价相对低的
    }

    if (k == -1) {
      printf("The maximum travel distance = %.2lf\n", stops[i].dis + max_dis);
      return 0;
    }

    if (stops[k].price <= stops[i].price) {  // 油只需足够行驶到油价更低的加油站
      ans += ((stops[k].dis - stops[i].dis) / d_avg - oil) * stops[i].price;
      oil = 0;  // 行驶到下个加油站刚好没油
    }
    else {
      ans += (c_max - oil) * stops[i].price;  // 加满油
      oil = c_max - (stops[k].dis - stops[i].dis) / d_avg;  // 行驶到下一个加油站
    }
    i = k;  // 以当前的加油站为起点
  }
  cout << fixed << setprecision(2) << ans;
}
```

## 8.5.7 Calling

### 题意

有$T$ $(1 \leq T \leq 1e5)$组测试数据.有六种正方形纸片,其中边长为$i$ $(1 \leq i \leq 6)$的正方形有$k_i$ $(0 \leq k_i \leq 1e4)$个.现需将它们放在至多$s$ $(0 \leq s \leq 1e9)$个边长为6的正方形框内,每个框未必放满,每个框内的正方形边长未必相同.若能放下,输出"Yes",否则输出"No".

### 思路

用变量$need$表示放下所有纸片需要的框数的最小值,将其与$s$比较,判断是否有解.

显然$6 \times 6, 5 \times 5, 4 \times 4$的纸片一个纸片需要一个框,$3 \times 3$的纸片一个框可放4个,$2 \times 2$和$1 \times 1$的纸片插空放.

贪心,先放$6 \times 6, 5 \times 5, 4 \times 4, 3 \times 3$的纸片,若还剩下$d \neq 0$个$3 \times 3$纸片,分三种情况:①若$d = 1$,则还能放下5个$2 \times 2$的纸片;②若$d = 2$,则还能放下3个$2 \times 2$的纸片;③若$d = 3$,则还能放下1个$2 \times 2$的纸片.用变量$put2$记录还能放的$2 \times 2$纸片数.

若还剩下$2 \times 2$的纸片,则开新的框放.边长为$2 \times 6$的纸片放完后用到的框的剩余面积即还能放的$1 \times 1$纸片数,将其与给定的纸片数比较,判断是否有解.

### 代码

```
const int MAXN = 7;
int s;  // 6x6框数
int cnt[MAXN];  // cnt[i]表示边长为i的正方形的个数

int main() {
  CaseT{
    cin >> s;
    for (int i = 1; i <= 6; i++) cin >> cnt[i];

```

```
    int need = cnt[6] + cnt[5] + cnt[4] + ceil(cnt[3] / 4.0);

    int put2 = cnt[4] * 5;  // 还能放的2x2的个数
    if (cnt[3] % 4 != 0) {  // 3x3还有剩
      if (cnt[3] % 4 == 1) put2 += 5;
      else if (cnt[3] % 4 == 2) put2 += 3;
      else if (cnt[3] % 4 == 3) put2 += 1;
    }

    if (cnt[2] > put2) need += ceil((cnt[2] - put2) / 9.0);  // 还有2x2没放,开新的框

    int put1 = 6 * 6 * need;  // 剩余面积
    for (int i = 2; i <= 6; i++) put1 -= i * i * cnt[i];
    if (cnt[1] > put1) need += ceil((cnt[1] - put1) / 36.0);  // 还有1x1没放,开新的框

    cout << (need <= s ? "Yes" : "No") << endl;
  }
}
```

## 8.5.7 Aroma's Search

原题指路:https://codeforces.com/contest/1292/problem/B

### 题意

平面上有无数个物品,编号$0,1,\cdots$,分别放置在点$P_0,P_1,\cdots$处,且一个点处只有一个物品,其中第0个物品的坐标为$(x_0,y_0)$,第$i\ \ (i>0)$个物品的坐标为$(a_x\cdot x_{i-1}+b_x,a_y\cdot y_{i-1}+b_y)$.某人从点$(x_s,y_s)$出发,每一秒可移动到上、下、左、右四个相邻的格子之一.求他在$t$秒内最多能获得多少物品.

第一行输入六个整数$x_0,y_0,a_x,a_y,b_x,b_y\ \ (1\le x_0,y_0\le 1e16,2\le a_x,a_y\le 100,0\le b_x,b_y\le 1e16)$.第二行输入三个整数$x_s,y_s,t\ \ (1\le x_s,y_s,t\le 1e16)$.

### 思路

显然所有点都在第一象限,且物品点向右上方越来越稀疏.

贪心策略:从起点出发,先收集靠近原点方向的物品,有多余时间先原路返回,再收集右上方的物品.

[**证**] (1)设点$P_{j-1}$与$P_j$间的Manhattan距离为$dis(P_{j-1},P_j)$,则$\sum_{j=1}^{i}dis(P_{j-1},P_j)=dis(P_0,P_i)$.

设点$P_{i+1}$与$P_i$的横坐标之差为$Xdis(P_{i+1},P_i)$.注意到点最密集时$a_x=2,b_x=0$,

此时$Xdis(P_{i+1},P_i)=(a_x\cdot x_i+b_x)-x_i=(a_x-1)\cdot x_i+b_x=x_i$.

则$Xdis(P_0,P_i)=x_i-x_0\le x_i-1<x_i=Xdis(P_{i+1},P_i)$.

同理考察纵坐标,不等式两边相加得:$dis(P_{i+1},P_i)>dis(P_0,P_i)$.

这表明:收集第$0\sim(i-1)$号物品所需时间小于收集第$(i+1)$号物品所需的时间.

(2)若收集完第$(i+1)$个物品时继续向右上方收集第$(i+2)$个物品,注意到$dis(P_{i+1},P_{i+2})>dis(P_i,P_{i+1})$,

则$dis(P_i,P_{i+1})+dis(P_{i+1},P_{i+2})>2dis(P_0,P_i)$,这表明:$i\ge 2$时先原路返回更优.

①$i=0$时显成立.

②$i=1$时,只考虑横坐标,代入得$x_1=2x_0, x_2=4x_0$.计算得$P_0\to P_1\to P_0\to P_2$和$P_0\to P_1\to P_2$耗时相等.

$a_x$最小为$2$,$b_x$最小为$0$,注意到$2^{64}>1e18$,故可能能到达的点数$n$不超过$64$.

先预处理出从起点出发最远能到达的物品点,注意起点右上方能直接到达的物品点也要预处理出.

枚举回头的分界点$P_i$,先从起点直接到$P_i$,然后$P_i\to P_{i-1}\to\cdots\to P_0$地返回$P_0$,有多余的时间原路返回,收集第$(i+1),(i+2),\cdots$个物品.

## 代码

```cpp
const int MAXN = 64;  // MAXN = log2(1e18)
ll x[MAXN], y[MAXN];

ll get_dis(ll x1, ll y1, ll x2, ll y2) { return abs(x1 - x2) + abs(y1 - y2); }

void solve() {
  ll x0, y0, ax, ay, bx, by; cin >> x0 >> y0 >> ax >> ay >> bx >> by;
  ll xs, ys, t; cin >> xs >> ys >> t;

  x[0] = x0, y[0] = y0;
  int n = 1;
  while (true) {  // 求从起点出发最远能到达的物品点
    x[n] = ax * x[n - 1] + bx, y[n] = ay * y[n - 1] + by;

    // 注意起点右上方的能直接到达的物品点也要求出
    if (x[n] > xs && y[n] > ys && get_dis(xs, ys, x[n], y[n]) > t) break;
    else n++;
  }

  ll ans = 0;
  for (int i = 0; i <= n; i++) {  // 枚举回头的分界点
    ll res = 0;  // 收集到的物品数
    ll rest = t;  // 剩余时间
    if (get_dis(xs, ys, x[i], y[i]) <= rest) {  // 起点直接到第i个物品
      rest -= get_dis(xs, ys, x[i], y[i]);
      res++;
    }
    else {
      ans = max(ans, res);
      continue;  // 注意不是break,因为有起点右上方的节点
    }

    for (int j = i; j; j--) {  // 第i个物品点经过第(i-1),(i-2),...个物品点回到起点
      if (get_dis(x[j], y[j], x[j - 1], y[j - 1]) <= rest) {
        rest -= get_dis(x[j], y[j], x[j - 1], y[j - 1]);
        res++;
      }
      else break;
    }

    for (int j = 1; j <= n; j++) {  // 从起点出发经过第1,2,...个物品点到达最远处
      if (get_dis(x[j - 1], y[j - 1], x[j], y[j]) <= rest) {
        rest -= get_dis(x[j - 1], y[j - 1], x[j], y[j]);
        res += j > i;  // 注意j>i时才是新物品点
      }
```

```
      else break;
    }
    ans = max(ans, res);
  }
  cout << ans;
}

int main() {
  solve();
}
```

## 8.5.8 Labi-Ribi

### 题意

初始时有编号$1 \sim n$的$n$个关卡,其中第$i$个关卡的BOSS等级为$h_i$,玩家只有等级不小于BOSS的等级时才能打败BOSS,打败BOSS后其他所有未打败的BOSS等级加$a_i$,玩家等级加$b_i$.求通过所有关卡的最小初始等级.现有$q$个新增关卡,每个关卡包含三个整数$h_i, a_i, b_i$,意义同上.每新增一个关卡后,求通过所有关卡的最小初始等级.

第一行输入一个整数$n$ $(1 \le n \le 1e5)$,表示初始关卡数.第二行输入$n$个整数$h_1, \cdots, h_n$ $(-1e9 \le h_i \le 1e9)$.接下来$n$行每行输入两个整数$a_i, b_i$ $(-1e9 \le a_i, b_i \le 1e9)$.接下来一行输入一个整数$q$ $(0 \le q \le 1000)$,表示新增关卡数.接下来$q$行每行输入三个整数$h_i, a_i, b_i$ $(-1e9 \le h_i, a_i, b_i \le 1e9)$.

先输出通过所有初始关卡的最小初始等级.对每个新增关卡,输出新增该关卡后通过所有关卡的最小初始等级.

### 思路

显然只需考虑$c_i = b_i - a_i$,问题转化为:有$n$个物品,能量分别为$h_1, \cdots, h_n$,只有自身能量$\ge h_i$时才能拿第$i$个物品,且拿完后自身能量$+= c_i$,求拿完所有物品所需的最小初始能量.

显然应先拿$c \ge 0$的物品,并按$h$升序拿.

对$c < 0$的物品,可能出现先拿物品$i$就不能拿物品$j$,但先拿物品$j$还可拿物品$i$的情况.设当前能量为$cur$,则$cur + c_i < h_j, cur + c_j \ge h_i$,整理得$h_i + c_i < h_j + c_j$,故将$c < 0$的物品按$h + c$值降序排列,对$h + c$值相等的物品,显然应先拿$h$大的.

可将$n$个初始关卡的最后一个也作为一个询问,若每次询问都对序列排序,总时间复杂度$O\left((n-1)\log(n-1) + \sum_{i=1}^{q+1}(n-1+i)\log(n-1+i)\right) = O(qn\log n)$,最坏为$1.6e9$,会TLE.考虑优化,不将$n$个初始关卡的最后一个作为询问,注意到每次插入一个新关卡再排序后,大部分关卡的顺序不变,故可直接扫一遍已有序的关卡,将新关卡插入到排序后的位置.总时间复杂度$O\left(n\log n + \sum_{i=1}^{q}(n+i-1)\right) = O\left(n\log n + \frac{q(2n+q-1)}{2}\right) \le O(nq)$,最坏$1e8$,可过.

注意cal()函数中的初始等级和当前等级初始化为$-INF$,因为可能所有BOSS的等级都是负的.注意答案可能爆int.

### 代码

```
const int MAXN = 1e5 + 5;
int n;
struct Stage {
  int h, a, b;
  ll c;
```

```
  bool operator<(const Stage& B)const {
    if (c >= 0 && B.c >= 0) return h < B.h;  // c≥0的物品按h升序排列
    else if (c < 0 && B.c < 0) return h + c > B.h + B.c;  // c<0的物品按h+c降序排列
    else return c > B.c;  // c≥0的物品排在前
  }
};
vector<Stage> stages;

ll cal() {
  ll res = -INFF;  // 初始等级
  ll cur = -INFF;  // 当前等级

  for (auto& st : stages) {
    res += cur >= st.h ? 0 : st.h - cur;
    cur += st.c + (cur >= st.h ? 0 : st.h - cur);
  }

  return res;
}

void solve() {
  cin >> n;
  stages.resize(n);

  for (int i = 0; i < n; i++) cin >> stages[i].h;
  for (int i = 0; i < n; i++) {
    int a, b; cin >> a >> b;
    stages[i].a = a, stages[i].b = b, stages[i].c = b - a;
  }

  sort(all(stages));

  cout << cal() << endl;

  CaseT{
    int h,a,b; cin >> h >> a >> b;
    Stage st({ h, a, b, b - a });

    // 将新的关卡插到排序后的位置
    bool ok = false;
    for (int i = 0; i < stages.size(); i++) {
      if (st < stages[i]) {
        stages.insert(stages.begin() + i, st);
        ok = true;
        break;
      }
    }
    if (!ok) stages.push_back(st);

    cout << cal() << endl;
  }
}

int main() {
  solve();
}
```

## 8.5.9 Max Pair Matching

### 题意

给定$2n$个数对$(a_i, b_i)$ $(a_i, b_i \geq 1, 1 \leq i \leq 2n)$,考察一张由$2n$个节点构成的完全图,其中$edge < i, j >$ $(1 \leq i, j \leq 2n, i \neq j)$的边权$w_{ij} = \max\{|a_i - a_j|, |a_i - b_j|, |b_i - a_j|, |b_i - b_j|\}$.从中选出$n$条边,使得任意两条选中的边无公共顶点,求选出的边的边权之和的最大值.

第一行输入一个整数$n$ $(1 \leq n \leq 1e5)$.接下来$2n$行每行输入两个整数$a, b$ $(1 \leq a, b \leq 1e9)$.

### 思路

因选择的$n$条边中任意两条选中的边无公共顶点,则每个数只能被选一次,进而每对$(a_i, b_i)$中都会选出一个数带正号,另一个数带负号,即边权之和的式子中有$n$个正项和$n$个负项.不妨设$a_i \leq b_i$,否则交换$a_i$和$b_i$,则最优解中在$2n$对$(a_i, b_i)$中选择$n$个$b_i$和$n$个$-a_i$.

对$S = \sum_{i=1}^{2n} b_i$,考察将其中的$n$个$b_i$换为$-a_i$.注意到每换一个会使得$S$减少$a_i + b_i$,为使得边权之和最大,应使得$a_i + b_i$尽可能小.故将所有数对$(a_i, b_i)$按$a_i + b_i$的值升序排列,在前$n$个中取$-a_i$,在后$n$个数中取$b_i$.总时间复杂度$O(n \log n)$.

### 代码

```cpp
void solve() {
  int n; cin >> n;
  n <<= 1;

  vii a(n);
  for (int i = 0; i < n; i++) {
    int x, y; cin >> x >> y;
    if (x > y) swap(x, y);
    a[i] = { x,y };
  }

  sort(all(a), [&](const pii& a, const pii& b) {
    return a.first + a.second < b.first + b.second;
    });

  ll ans = 0;
  for (int i = 0; i < n; i++)
    ans += -a[i].first * (i < n / 2) + a[i].second * (i >= n / 2);
  cout << ans;
}

int main() {
  solve();
}
```

# 8.5.10 Alice and Bob-1

原题指路:https://codeforces.com/gym/103186/problem/J

## 题意

有$n$ $(1 \leq n \leq 5000)$个元素$a_1, \cdots, a_n$ $(-1e9 \leq a_i \leq 1e9)$,Alice和Bob轮流取走一个元素,Alice先手.取完所有元素后,两人拥有的价值定义为各自取的元素之和的绝对值.设Alice和Bob拥有的价值分别为$A$和$B$,Alice希望$A-B$尽量大,Bob希望$A-B$尽量小,两人都采取最优策略,求$A-B$.

## 思路

贪心策略:两人轮流取当前的最大元素.

**[证]** Alice希望$A-B$尽量大,Bob希望$A-B$尽量小,都等价于两人希望自己拥有的价值尽量大.

设Alice和Bob拥有的价值分别为$|A|$和$|B|$.

因价值有绝对值,故所有数取反不影响答案,不妨设$S=\sum_{i=1}^{n} a_i \geq 0$..

$ans=|A|-|B|=|A|-|S-A|=\begin{cases} S, A \geq S \\ 2A-S, 0<A<S \\ -S, A \leq 0 \end{cases}$.作图知:该分段函数单调增,故证.

先将$a[]$升序排列.因集合元素可能全正、全负、有正有负,但答案都能归结为两种情况:①最大值被Alice取走,使得$ans$去掉绝对值后正得更多;②最大值被Bob取走,使得$ans$去掉绝对值后负得更多,两种情况取$\max$即可.

## 代码

```cpp
void solve() {
  int n; cin >> n;
  vi a(n + 1);
  ll sum = 0;
  for (int i = 1; i <= n; i++) {
    cin >> a[i];
    sum += a[i];
  }

  sort(a.begin() + 1, a.end());

  ll sum1 = 0, sum2 = 0;
  for (int i = 1; i <= n; i += 2) sum1 += a[i];
  for (int i = n; i >= 1; i -= 2) sum2 += a[i];

  cout << max(abs(sum1) - abs(sum - sum1), abs(sum2) - abs(sum - sum2));
}

int main() {
  solve();
}
```

# 8.5.11 Minimum Value Rectangle

原题指路: https://codeforces.com/problemset/problem/1027/C

## 题意 (2 s)

有$t$ $(t \geq 1)$组测试数据. 每组测试数据给定$n$根长度分别为$a_1, \cdots, a_n$ $(1 \leq a_i \leq 1e4)$的木棒. 对长为$x$、宽为$y$的矩形, 定义参数$P = x + y$. 现需从中选4根木棒构成一个参数为$P$、面积为$S$的矩形, 使得$\frac{P^2}{S}$最小, 输出任一方案. 数据保证有解, 且所有测试数据的$n$之和不超过$1e6$.

## 思路

设取的木棒长度分别为$x, x, y, y$, 则$\frac{P^2}{S} = \frac{4(x+y)^2}{xy} = 4\left(\frac{x}{y} + \frac{y}{x}\right) + 8$.

令$t = \frac{x}{y}$, 则$\frac{P^2}{S} = 4\left(t + \frac{1}{t}\right) + 8$, 其中对勾函数$t + \frac{1}{t}$当$t = \frac{1}{t}$, 即$t = 1$时取得最小值.

为使得$\frac{P^2}{S}$最小, 应尽量使得$t = \frac{x}{y}$接近1. 将$a[]$排序后, 每次检查相邻的木棒即可.

为防止精度问题, $\frac{P_1^2}{S_1} < \frac{P_2^2}{S_2}$可用$P_1^2 \cdot S_2 < P_2^2 \cdot S_1$判断.

## 代码

```cpp
void solve() {
    int n; cin >> n;
    vector<int> a(n);
    for (auto& ai : a) cin >> ai;
    sort(all(a));

    vector<int> sticks;
    for (int i = 0; i + 1 < n; i++) {
        if (a[i] == a[i + 1])
            sticks.push_back(a[i++]);
    }

    int x, y;
    int P2 = INF, S = 1;

    int siz = sticks.size();
    for (int i = 0; i + 1 < siz; i++) {
        int tmpx = sticks[i], tmpy = sticks[i + 1];
        int tmpP2 = (ll)(tmpx + tmpy) * (tmpx + tmpy), tmpS = tmpx * tmpy;
        if ((ll)tmpP2 * S < (ll)P2 * tmpS) {
            x = tmpx, y = tmpy;
            P2 = tmpP2, S = tmpS;
        }
    }
    cout << x << ' ' << x << ' ' << y << ' ' << y << endl;
}

int main() {
    CaseT
    solve();
}
```

# 8.6 转化为图论

# 8.7 杂题

## 8.7.1 ABC Legacy

### 题意

给定一个长度为$2n$ $(1 \leq n \leq 1e5)$的且只包含字符'A'、'B'、'C'的字符串$s$.判断是否能将$s$分割为$n$个不相交的子串,每个子串是"AB"、"AC"、"BC"之一.若能,输出"YES"并输出所有子串包含的两个字符的下标;否则输出"NO".

### 思路I

设$s$能分割为$n$个不相交的子串,其中"AB"、"AC"、"BC"各$x, y, z$个.设$s$中'A'、'B'、'C'分别出现$cnt_A, cnt_B, cnt_C$次,则$\begin{cases} cnt_A = x + y \\ cnt_B = x + z \\ cnt_C = y + z \end{cases}$,解得$\begin{cases} x = \dfrac{cnt_A + cnt_B - cnt_C}{2} \\ y = \dfrac{cnt_A - cnt_B + cnt_C}{2} \\ z = \dfrac{-cnt_A + cnt_B + cnt_C}{2} \end{cases}$,则有解的必要条件是$x, y, z \in \mathbb{N}$.

考察'B'出现的位置,它将作为子串"AB"的第二个字母出现$x$次,作为子串"BC"的第一个字母出现$z$次.显然最优解中'B'出现的前$z$个位置用于产生子串"BC",出现的后$x$个位置用于产生子串"AB",同时使得子串"AB"中'A'尽量靠右,子串"BC"中的'C'尽量靠左,在中间产生子串"AC".

$vis[i]$表示字符$s[i]$是否已匹配.先正着扫一遍$s$,将出现位置靠前的$z$个'B'与其右边最靠前的未匹配的'C'匹配(若存在,下同).再倒着扫一遍$s$,将出现位置靠后的$x$个'B'与其左边靠后的未匹配的'A'匹配.对剩下的字符,检查其是否是"ACACAC···"即可.时间复杂度$O(n)$.

### 代码I -> set实现(思路清晰,但TLE)

```
void solve() {
  int n; cin >> n;
  n <<= 1;
  string s; cin >> s;
  s = " " + s;

  set<int> A, B, C;  // 分别记录字符'A'、'B'、'C'出现的下标
  for (int i = 1; i <= n; i++) {
    if (s[i] == 'A') A.insert(i);
    else if (s[i] == 'B') B.insert(i);
    else C.insert(i);
  }
  ;
  int tmpa = (int)A.size() + (int)B.size() - (int)C.size(),
    tmpb = (int)A.size() - (int)B.size() + (int)C.size(),
    tmpc = -(int)A.size() + (int)B.size() + (int)C.size();
  if (tmpa < 0 || tmpb < 0 || tmpc < 0 || (tmpa & 1) || (tmpb & 1) || (tmpc & 1)) {
    cout << "NO";
    return;
  }
```

```cpp
  int x = tmpa >> 1, y = tmpb >> 1, z = tmpc >> 1;

  vii ans;
  int cnt = 0;  // 当前匹配的B的个数
  while (true) {
    if (B.empty() || cnt == z) break;

    int i = *B.begin();  // B中最小值即当前字符'B'最早出现的下标
    auto tmp = upper_bound(all(C), i);  // 找到i右边第一个字符'C'出现的下标
    if (tmp != C.end()) {
      ans.push_back({ i,*tmp });
      B.erase(i), C.erase(*tmp);
      cnt++;
    }
    else break;
  }

  cnt = 0;
  while (true) {
    if (B.empty() || cnt == x) break;

    int i = *B.rbegin();  // B中最大值即当前字符'B'最晚出现的下标
    auto tmp = upper_bound(rall(A), i, greater<int>());  // 找到i左边第一个字符'A'出现的下标
    if (tmp != A.rend()) {
      ans.push_back({ *tmp,i });
      B.erase(i), A.erase(*tmp);
      cnt++;
    }
    else break;
  }

  if (B.size() || A.size() != C.size()) {
    cout << "NO";
    return;
  }

  while (A.size()) {
    int tmpa = *A.begin(), tmpc = *C.begin();
    if (tmpa > tmpc) {
      cout << "NO";
      return;
    }
    else {
      ans.push_back({ tmpa,tmpc });
      A.erase(tmpa), C.erase(tmpc);
    }
  }

  cout << "YES" << endl;
  for (auto [l, r] : ans) cout << l << ' ' << r << endl;
}

int main() {
  solve();
}
```

## 代码II

```cpp
void solve() {
  int n; cin >> n;
  n <<= 1;
  string s; cin >> s;
  s = " " + s;

  int cnta = 0, cntb = 0, cntc = 0;
  for (int i = 1; i <= n; i++) {
    if (s[i] == 'A') cnta++;
    else if (s[i] == 'B') cntb++;
    else cntc++;
  }
  ;
  int tmpa = cnta + cntb - cntc, tmpb = cnta - cntb + cntc, tmpc = -cnta + cntb + cntc;
  if (tmpa < 0 || tmpb < 0 || tmpc < 0 || (tmpa & 1) || (tmpb & 1) || (tmpc & 1)) {
    cout << "NO";
    return;
  }
  int x = tmpa >> 1, y = tmpb >> 1, z = tmpc >> 1;

  vii ans;
  vb vis(n + 1);
  deque<int> posb;  // 字符'B'出现的下标
  int cnt = 0;  // 当前匹配的'B'的个数
  for (int i = 1; i <= n && cnt < z; i++) {  // 正着扫一遍s,匹配"BC"
    if (s[i] == 'B') posb.push_back(i);
    else if (s[i] == 'C') {
      if (posb.size()) {
        ans.push_back({ posb.front(),i });
        vis[posb.front()] = vis[i] = true;
        posb.pop_front();
        cnt++, cntc--;
      }
    }
  }

  cntb-=cnt, cnt = 0;  // 更新未匹配的'B'的个数,并清空cnt
  posb.clear();
  for (int i = n; i >= 1 && cnt < x; i--) {  // 倒着扫一遍s,匹配"AB"
    if (vis[i]) continue;  // 防止一个字符重复被使用

    if (s[i] == 'B') posb.push_back(i);
    else if (s[i] == 'A') {
      if (posb.size()) {
        ans.push_back({ i,posb.front() });
        vis[i] = vis[posb.front()] = true;
        posb.pop_front();
        cnt++, cnta--;
      }
    }
  }
  cntb -= cnt;  // 更新未匹配的'B'的个数

  if (cntb || cnta != cntc) {  // 还有未匹配的'B'则无解
    cout << "NO";
```

```
    return;
  }

  deque<int> posa, posc;
  for (int i = 1; i <= n; i++) {
    if (!vis[i]) {
      if (s[i] == 'A') posa.push_back(i);
      else posc.push_back(i);
    }
  }

  while (posa.size()) {
    int tmpa = posa.front(), tmpc = posc.front();
    if (tmpa > tmpc) {
      cout << "NO";
      return;
    }
    else {
      ans.push_back({ tmpa,tmpc });
      posa.pop_front(), posc.pop_front();
    }
  }

  cout << "YES" << endl;
  for (auto [l, r] : ans) cout << l << ' ' << r << endl;
}

int main() {
  solve();
}
```

## 思路II

将'A'、'C'分别视为左括号、右括号,而'B'可视为左括号与'C'匹配,也可视为右括号与'A'匹配,转化为括号匹配问题.

合法的括号匹配共$n$个左括号和$n$个右括号,则有解的必要条件是$\max\{cnta, cntc\} \leq n$.

显然'B'中有$cntbc = n - cnta$个需作为左括号,将它们加入队列中,它们将产生"BC".其余的'B'作为右括号与'A'匹配.

## 代码II

```
deque<int> que[3];  // posa,posb,posc

void solve() {
  int n; cin >> n;
  string s; cin >> s;

  int cnta = 0, cntb = 0, cntc = 0;
  for (auto ch : s) {
    if (ch == 'A') cnta++;
    else if (ch == 'B') cntb++;
    else cntc++;
  }
  ;
  int cntbc = n - cnta;  // 还需补cntbc个左括号
  if (cntbc < 0) {  // 括号不足
    cout << "NO" << endl;
```

```
    return;
  }

  n <<= 1, s = " " + s;
  vii ans;
  for (int i = 1; i <= n; i++) {
    if (s[i] == 'A') que[0].push_back(i);
    else if (s[i] == 'B') {
      if (cntbc) {  // 'B'作为左括号
        que[1].push_back(i);
        cntbc--;
      }
      else {  // 匹配"AB"
        if (que[0].empty()) {  // 无'A'
          cout << "NO" << endl;
          return;
        }

        ans.push_back({ que[0].back(),i});
        que[0].pop_back();
      }
    }
    else {  // 'B'作为左括号,匹配"AC"、"BC"
      if (que[0].empty() && que[1].empty()) {
        cout << "NO" << endl;
        return;
      }

      if (que[1].size()) {  // 优先匹配'B'
        ans.push_back({ que[1].back(),i });
        que[1].pop_back();
      }
      else {
        ans.push_back({ que[0].back(),i });
        que[0].pop_back();
      }
    }
  }

  cout << "YES" << endl;
  for (auto [l, r] : ans) cout << l << ' ' << r << endl;
}

int main() {
  solve();
}
```

## 8.7.2 to Pay Respects

## 题意

打BOSS,它不会攻击你,但它会念再生咒语.

游戏共$N$轮,每轮按顺序发生如下事件:①BOSS可以选择念一次再生咒语;②若你还有能量,你可以选择念一次中毒咒语;③你攻击BOSS,造成$X$点伤害;④本轮的负面效果生效.

有两种状态:再生和中毒.当前BOSS的状态可用三个整数描述:当前血量$hp$、当前中毒等级$p$、当前再生等级$r$.初始时BOSS无中毒和再生等级,即$p=r=0$.每一级中毒会造成$P$点伤害,每一级再生会恢复$R$点血量.再生咒语会令$r++$.中毒咒语会令$p++$,若此时$r>0$,则同时$r--$.每轮结束后$hp-=x+P\cdot p-R\cdot r$(该值可能为负,如BOSS回的血比收到的攻击更多时).

每轮开始时你都知道BOSS是否选择念再生咒语.你有$K$次机会念中毒咒语.若BOSS的初始血量足够高,即$N$轮内你无法击败BOSS,求你最大能对BOSS造成多少伤害,即求$hp_{start}-hp_{end}$的最大值.注意BOSS的血量可以大于其初始血量.

第一行输入五个整数$N,X,R,P,K$ $(1\le N,X,R,P,K\le 1e6,0\le K\le N)$.第二行输入一个长度为$N$的$01$串描述BOSS是否念再生咒语,其中'1'表示BOSS在对应的轮念重生咒语.

## 思路I

中毒咒语对再生咒语的影响可视为增加了攻击力.

因每轮的伤害分开计算,可先忽略念中毒咒语的次数限制,分别统计每轮念中毒咒语时本轮能造成的伤害$damage$,最后降序排列,取前$k$大即可.$ans$从$nx$开始,从左往右扫一遍时间线.若第$i$ $(1\le i\le N)$轮BOSS不念再生咒语,则本轮造成的伤害为$(N-i+1)\cdot P$;否则本轮造成的伤害为$(N-i+1)\cdot(P+R)$,并$ans-=L(N-i+1)\cdot R$.总时间复杂度$O(n\log n)$.

## 代码I

```cpp
void solve() {
  int n, x, r, p, k; cin >> n >> x >> r >> p >> k;
  string s; cin >> s; s = " " + s;

  ll ans = (ll)n * x;
  vl damage;
  for (int i = 1; i <= n; i++) {
    if (s[i] == '1') {
      // 中毒咒语对再生咒语的影响视为增加攻击力
      damage.push_back((ll)(n - i + 1) * (p + r));
      ans -= (ll)(n - i + 1) * r;
    }
    else damage.push_back((ll)(n - i + 1) * p);
  }

  sort(rall(damage));

  for (int i = 0; i < k; i++) ans += damage[i];
  cout << ans;
}

int main() {
  solve();
}
```

## 思路II

因再生和中毒的等级叠加后产生的效果更强,显然应尽量早地念中毒咒语.注意到BOSS念再生咒语时,可念一次中毒咒语消除再生咒语的影响,但当BOSS在靠后的轮次念再生咒语,此时再念中毒咒语可能不优.故最优解中尽量在时间线的某个前缀念中毒咒语.

念中毒咒语有两种效果:①令$p++$;②令$p++,r--$.先在第$1\sim k$轮念中毒咒语.每次考察最靠后一个①类型的中毒咒语位置换为第一个②类型的中毒咒语位置是否会让答案更优.过程用双指针维护(下面的代码未明显写出双指针,但本质相同),总时间复杂度$O(n)$.

## 代码II

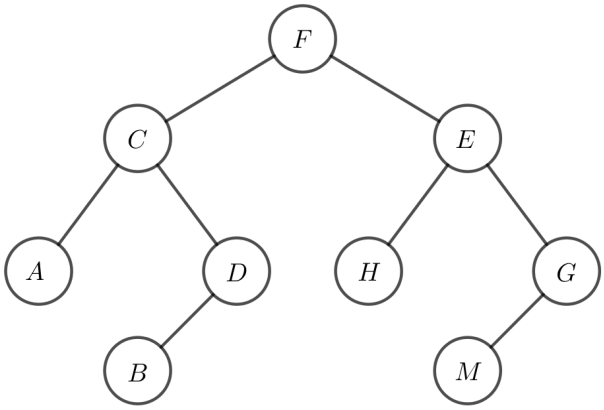
```cpp
void solve() {
  int n, x, r, p, k; cin >> n >> x >> r >> p >> k;
  string s; cin >> s; s = " " + s;

  deque<int> que1, que2;  // 念①②类型咒语的位置
  ll res = (ll)n * x;
  int curp = 0, curr = 0;  // 当前的中毒等级、再生等级
  for (int i = 1; i <= n; i++) {
    if (i > k) {
      if (s[i] == '1') {
        curr++;
        que2.push_back(i);
      }

      res += (ll)p * curp - (ll)r * curr;
      continue;
    }

    // 第1~k轮念中毒咒语,curr保持为0
    curp++;
    res += (ll)p * curp;
    if (s[i] != '1') que1.push_back(i);
  }

  ll ans = res;
  while (que1.size() && que2.size()) {
    int tmp1 = que1.back(), tmp2 = que2.front();  // 最靠后的一个①类型咒语、最靠前的一个②类型咒语
    res -= (ll)(tmp2 - tmp1) * p - (ll)(n - tmp2 + 1) * r;  // 将tmp1处的念中毒咒语换到tmp2处
    if (res > ans) {
      ans = res;
      que1.pop_back();
    }
    else res = ans;
    que2.pop_front();
  }
  cout << ans << endl;
}

int main() {
  solve();
}
```

# 8.7.3 Amazing Tree

## 题意

有$t$ $(1 \leq t \leq 1e5)$组测试数据.每组测试数据第一行输入一个整数$n$ $(2 \leq n \leq 2e5)$,表示树的节点数.接下来$(n-1)$行每行输入两个整数$u, v$ $(1 \leq u, v \leq n, u \neq v)$,表示节点$u$与$v$间存在边.每棵树遍历时可任一选择起点,也可任意选定兄弟节点的遍历顺序.对每棵树,求字典序最小的后序遍历.数据保证所有测试数据的$n$之和不超过$2e5$.

## 思路




固定树的后序遍历:$A\ B\ D\ C\ H\ M\ G\ E\ F$.

后序遍历的起点是叶子节点,以每个叶子节点为第一个节点可产生一个后序遍历.为使得字典序最小,显然应取编号最小的叶子节点为后序遍历的起点,设为节点$v$.

设节点$u$是节点$v$是唯一邻居.设以$u$为根节点的子树所含节点的最小编号为$minidx[u]$.为得到以$v$为起点的后序遍历,应先从根节点出发往下搜到$v$,且$v$的前驱为$u$,遍历完$v$返回$u$,此时有两种情况:

(1)$u$是根节点,此时应将子树按$minidx[]$升序排列,依次遍历子树,最后将$u$加入后序遍历.

(2)$u$非根节点,此时先将子树按$minidx[]$升序排列,设有$k$棵子树,则先依次遍历前$(k-1)$棵.

①若$u$的子树中$minidx[]$的最大值(即排序后的第$k$棵子树的$minidx[]$)小于$u$,则先遍历第$k$棵子树,再将$u$加入后序遍历.

②若$u$的子树中$minidx[]$的最大值$> u$,则先将$u$加入后序遍历,再遍历第$k$棵子树.

实现时以最小编号的节点为根节点,则遍历时都是往下搜.

## 代码

```cpp
const int MAXN = 2e5 + 5;
int n;
vi edges[MAXN];
int d[MAXN];  // 每个节点的度数
vi ans;  // 后序遍历
int minidx[MAXN];  // minidx[u]表示以u为根节点的子树所包含的节点的编号的最小值

void dfs1(int u, int fa) {  // 预处理minidx[]:当前节点、前驱节点
  minidx[u] = n;  // 初始化为节点的最大编号

  bool is_leaf = true;  // 记录当前节点是否是叶子节点
  for (auto v : edges[u]) {
    if (v == fa) continue;

```

```
    is_leaf = false;
    dfs1(v, u);  // 递归求子树的信息
    minidx[u] = min(minidx[u], minidx[v]);
  }

  if (is_leaf) minidx[u] = u;  // 叶子节点的minidx是自己的编号
}

void dfs2(int u, int fa, bool is_root) {  // 树形DP:当前节点、前驱节点、u是否是子树根节点
  vii subtree;  // 对子树中的所有节点v,存{minidx[v],v}
  for (auto v : edges[u]) {
    if (v == fa) continue;

    subtree.push_back({ minidx[v], v });
  }

  if (!subtree.size()) {  // 叶子节点直接加入后序遍历
    ans.push_back(u);
    return;
  }

  sort(all(subtree));  // 将子树中的所有节点按minidx[]升序排列

  if (is_root) {  // u是根节点
    for (auto [idx, v] : subtree) dfs2(v, u, true);  // 节点v是子树的根节点
    ans.push_back(u);
  }
  else {  // u非根节点
    for (int i = 0; i < subtree.size() - 1; i++)  // 遍历前(k-1)棵子树
      dfs2(subtree[i].second, u, true);  // 节点subtree[i].second是子树的根节点

    auto [idx, v] = subtree.back();
    if (idx < u) {
      dfs2(v, u, true);  // 节点v是子树的根节点
      ans.push_back(u);
    }
    else {
      ans.push_back(u);
      dfs2(v, u, false);  // 节点v不是子树的根节点
    }
  }
}

void solve() {
  cin >> n;
  ans.clear();
  for (int i = 1; i <= n; i++) d[i] = 0, edges[i].clear();

  for (int i = 0; i < n - 1; i++) {
    int u, v; cin >> u >> v;
    edges[u].push_back(v), edges[v].push_back(u);
    d[u]++, d[v]++;
  }

  int leaf = 0;  // 编号最小的叶子节点作为搜索起点
  for (int i = 1; i <= n; i++) {
    if (d[i] == 1) {  // 叶子节点
      leaf = i;
```

```
      break;
    }
  }

  dfs1(leaf, -1);  // 从编号最小的叶子节点开始搜,其无前驱节点
  dfs2(leaf, -1, false);  // 从编号最小的叶子节点开始搜,其无前驱节点,它不是原树中的根节点

  for (int i = 0; i < ans.size(); i++) cout << ans[i] << " \n"[i == ans.size() - 1];
}

int main() {
  CaseT  // 单测时注释掉该行
  solve();
}
```

## 8.7.4 Windblume Festival

### 题意 (3 s)

有$T$组测试数据.每组测试数据有$n$ $(1 \le n \le 1e6)$个数$a_i$ $(-1e9 \le a_i \le 1e9, 1 \le i \le n)$围成一圈,其中第$x$ $(1 \le x \le n)$个数与第$(x \bmod n) + 1$个数相邻.现进行若干次操作直至只剩下一个数:设当前还剩下$k$ $(2 \le k \le n)$个数,每次操作取定一个$x$ $(1 \le x \le k)$,将令$a_i \leftarrow a_i - a_{(i \bmod k)+1}$后删去$a_{(i \bmod k)+1}$,删去后$a_i$与$a_{(i \bmod k)+1}$的下一个相邻.求最后剩下的数的最大值.数据保证所有测试数据的$n$之和不超过1e6.

### 思路

猜测最后剩下的数的最大值非负,若不全为零,则最大值为正.

下面以$n = 4$为例推导贪心策略:

(i) 原数列中有正有负时,先考虑只有一个负数的情况.设当前环为$p_1 \to n \to p_2 \to p_3 \to p_1$ $(n < 0, p_i > 0)$.不妨设$p_1 = \max\{p_1, p_2, p_3\}$.要使得最后剩下的数最大,最后一次操作应为$p_1$减去一个尽量大的负数,显然该负数是$n - p_2 - p_3$,则最后剩下的数为$p_1 - (n - p_2 - p_3) = p_1 + p_2 + p_3 - n = p_1 + p_2 + p_3 + |n| > 0$.显然这可推广到多个负数的情况.

(ii) 原数列全为正时,设当前环为$p_1 \to p_4 \to p_2 \to p_3 \to p_1$,且$p_1 > p_2 > p_3 > p_4 > 0$.为使得最后剩下的数最大,应让原本最大的$p_1$减去一个尽量小的数,最好能减去负数.策略:①$p_4 \leftarrow p_4 - p_2$,此时$p_4 < 0$;②$p_4 \leftarrow p_4 - p_3$,此时$p_4 < 0$;③$p_1 \leftarrow p_1 - p_4$,此时$p_1 > 0$且最大,最大值为
$p_1 - p_4 = p_1 - (p_4 - p_3) = p_1 - (p_4 - p_2 - p_3) = p_1 + p_2 + p_3 - p_4 > 0$.

(iii) 原数列全为负时,设当前环为$n_1 \to n_2 \to n_3 \to n_4 \to n_1$,且$0 > n_1 > n_2 > n_3 > n_4$.为使得最后剩下的数最大,应让原本最大的$n_1$减去一个尽量大的负数,或尝试让$n_1$变为正数,再减去一个尽量大的负数.策略:①$n_1 \leftarrow n_1 - n_2$,此时$n_1 > 0$;②$n_1 \leftarrow n_1 - n_3$,此时$n_1 > 0$;③$n_1 \leftarrow n_1 - n_4$,此时$n_1 > 0$且最大,最大值为
$n_1 - n_4 = (n_1 - n_3) - n_4 = (n_1 - n_2 - n_3) - n_4 = n_1 + |n_2| + |n_3| + |n_4| > 0$.

(iv) 原数列中有零时,将零都视为正数或负数,划归到上述三种情况.

显然上述贪心策略可推广到有$n$个数的情况,也证明了最后剩下的数的最大值非负,若不全为零,则最大值为正.

合并结论:$ans = \begin{cases} \sum_{i=1}^{n} |a_i|, \text{原数列有正有负时} \\ \sum_{i=1}^{n} |a_i| - 2|a_x|, \text{其余情况, 其中} |a_x| \text{是绝对值最小者} \end{cases}$ $(n \ge 2)$.$n = 1$时不符合该结论,要特判.

### 代码

```cpp
int n;  // 数的个数

int main() {
  CaseT{
    cin >> n;

    bool flag1 = 0, flag2 = 0;  // 有正数、有负数
    ll res = 0;  // 绝对值之和
    int minabs = INF;  // 绝对值最小值

    if (n == 1) {  // 特判n=1的情况
      int tmp; cin >> tmp;
      cout << tmp << endl;
      continue;
    }

    while (n--) {
      int tmp; cin >> tmp;
      if (tmp > 0) flag1 = 1;
      else if (tmp < 0) flag2 = 1;

      res += abs(tmp);
      minabs = min(minabs, abs(tmp));
    }
    cout << (res - (ll)2 * (flag1 + flag2 != 2) * minabs) << endl;
  }
}
```

---

## 8.7.5 Collecting Diamonds

### 题意

给定一长度为$n$ $(1 \leq s \leq 2e5)$的字符串$s$,下标从1开始.现有操作:选择一个下标$i \in [1, n-2]$ $s.t.$ $s_i = A, s_{i+1} = B, s_{i+2} = C$,若$i$是奇数,则删除$s_i$和$s_{i+2}$;否则删除$s_{i+1}$.将剩下的字符串拼起来,并更新$n$为新的字符串长度.求最大操作次数.

### 思路

注意到若删除'B',则其附近的'A'和'C'无法再删除,且该操作会改变在其后面的字符下标的奇偶性;若删除'A'和'C',若两端还有'A'和'C',则会与中间剩下的'B'拼起来,可作为下一次的操作对象.显然应在删除'B'前删除尽量多的'A'和'C'.因一个'B'与一对'A'和'C'对应,记录当前删除的'B'的个数$Bcnt$即可求出操作次数.

为防止时间复杂度变为$O(n^2)$,实现时不进行操作后的字符串拼接和重新赋下标的操作,而通过下标的奇偶性和之前是否删除过'B'(即改变后面元素的奇偶性)来判断本次操作删除的是'A'和'C'还是'B',显然这样并不影响操作次数.

### 代码

```cpp
const int MAXN = 2e5 + 5;
char s[MAXN];

void solve() {
  cin >> s + 1;
```

```
6
7   int n = strlen(s + 1);
8   int Bcnt = 0;  // 记录当前删了多少个'B'
9   int ans = 0;
10  for (int i = 1; i <= n; i++) {
11    if (s[i] == 'B') {
12      int ACcnt = 1;  // 记录最多可删多少个'A'和'C',作为半径故从1开始
13      while (i - ACcnt >= 1 && i + ACcnt <= n && s[i - ACcnt] == 'A' && s[i + ACcnt] ==
    'C') ACcnt++;
14      if (--ACcnt == 0) continue;  // 附近不是"ABC",注意减掉一开始多的1
15
16      if (i - 1 & 1) {  // 注意是看下标(i-1)的奇偶性
17        ACcnt--;  // 若之前未删过'B',则本次操作删除'A'和'C'
18        ans++;  // 此次删除可能是删除'A'和'C'或下面的删除'B',取决于之前是否删过'B'
19
20        if (!ACcnt) {  // 'A'和'C'删完了
21          if(Bcnt) Bcnt++;  // 若之前删过'B',则本次操作删除'B'
22
23          continue;
24        }
25      }
26
27      // (i-1)是偶数
28      Bcnt++;  // 还有剩下的"ABC",还能再删一个'B'
29      ans += min(ACcnt, Bcnt);  // 一个'B'与一对'A'和'C'对应
30    }
31  }
32  cout << ans;
33 }
34
35 int main() {
36   solve();
37 }
```

## 8.7.6 Equal Frequencies

原题指路:https://codeforces.com/contest/1782/problem/C

### 题意 (2 s)

称一个字符串是好的,如果每种字符出现的次数都相等.

有$t$ $(1 \le t \le 1e4)$组测试数据.每组测试数据第一行输入一个整数$n$ $(1 \le n \le 1e5)$,表示字符串$s$的长度.第二行输入一个长度为$n$的只包含小写英文字母的字符串$s$.数据保证所有测试数据的$n$之和不超过$1e5$.

对每组测试数据,求将$s$变换成好的字符串$t$所需改变的最小字符数,输出任一变换后好的字符串$t$.

### 思路I

显然字符种类数$c$是$n$不超过$26$的约数,每种字符的出现次数为$\frac{n}{c}$.

设字符$ch$在$s$中出现的次数为$cnt[ch]$.设最优解中保留$c$ $(c \mid n)$种字符,则将字符$j$替换为字符$i$更优的充要条件是:$\left|cnt_i - \frac{n}{c}\right| + cnt_j < \left|cnt_j - \frac{n}{c}\right| + cnt_i$,即$\left|cnt_i - \frac{n}{c}\right| - cnt_i < \left|cnt_j - \frac{n}{c}\right| - cnt_j$.注意计算需改变的最小字符数时每种字符会贡献两次答案.

枚举保留的字符种数$c$ $(1 \le c \le 26, c \mid n)$,贪心出应保留的字符后计算每种字符应出现的次数,将出现次数过多的字符替换为出现次数不足的字符即可.

## 代码I

```cpp
const int MAXS = 26;

void solve() {
  int n; string s; cin >> n >> s;

  vector<int> frequency(MAXS);
  for (auto ch : s) frequency[ch - 'a']++;

  int ans = n + 1;  // 改变的最小字符数
  string t = s;
  for (int c = 1; c <= MAXS; c++) {  // 枚举保留的字符种类
    if (n % c) continue;

    auto cnt = frequency;
    vector<int> ord(MAXS);
    iota(all(ord), 0);
    sort(all(ord), [&](int i, int j) {
      return abs(cnt[i] - n / c) - cnt[i] < abs(cnt[j] - n / c) - cnt[j];
    });

    int res = 0;
    vector<bool> use(MAXS);
    for (int i = 0; i < MAXS; i++) {
      if (i < c) {
        use[ord[i]] = true;
        res += abs(cnt[ord[i]] - n / c);
      }
      else res += cnt[ord[i]];
    }

    if ((res /= 2) >= ans) continue;  // 每种字符贡献两次答案
    else ans = res;

    t = s;
    for (auto& ch : t) {
      int x = ch - 'a', need = use[x] ? n / c : 0;
      if (need < cnt[x]) {  // 该字符出现次数过多
        for (int i = 0; i < c; i++) {  // 将其替换为要保留的个数不足的字符
          if (cnt[ord[i]] < n / c) {
            ch = 'a' + ord[i];
            cnt[x]--, cnt[ord[i]]++;
            break;
          }
        }
      }
    }
  }
  cout << ans << endl << t << endl;
}

int main() {
  CaseT
```

```
    solve();
}
```

## 思路II

显然将出现次数较少的字符替换为需保留的出现次数不足的字符最优,将所有字符按出现次数排列,模拟该过程即可.

不同于**思路I**,该方法计算需改变的最小字符数时不会重复贡献.

## 代码II

```cpp
const int MAXS = 26;

void solve() {
  int n; string s; cin >> n >> s;

  map<int, vector<int>> pos;  // 每种字符在s中出现的下标
  for (int i = 0; i < n; i++) pos[s[i] - 'a'].push_back(i);

  vector<pair<int, int>> chs;  // first为字符出现的次数,second为字符种类
  for (int c = 0; c < MAXS; c++) {
    if (pos.count(c)) chs.push_back({ pos[c].size(),c });
    else chs.push_back({ 0,c });
  }
  sort(rall(chs));

  int ans = n + 1;  // 改变的最小字符数
  int kind;  // 保留的字符种数
  for (int c = 1; c <= MAXS; c++) {  // 枚举保留的字符种数
    if (n % c) continue;

    int cnt = n / c;  // 每种字符应出现的次数
    int res = 0;  // 改变的最小字符数
    for (int i = 0; i < MAXS; i++) {
      if (i >= c) res += chs[i].first;  // 需删去的字符
      else if (chs[i].first > cnt) res += chs[i].first - cnt;  // 需保留但出现次数过多的字符
    }

    if (res < ans) ans = res, kind = c;
  }

  int cnt = n / kind;  // 每种字符应出现的次数
  vector<int> trash;  // 可供替换的字符的下标
  vector<char> need;  // 缺少的字符的种类
  for (int i = 0; i < MAXS; i++) {
    if (i < kind) {  // 需保留的字符
      if (chs[i].first < cnt) {
        for (int j = 0; j < cnt - chs[i].first; j++)
          need.push_back('a' + chs[i].second);
      }

      if (chs[i].first > cnt) {
        for (int j = 0; j < chs[i].first - cnt; j++)
          trash.push_back(pos[chs[i].second][j]);
      }
    }
```

```
    else {  // 需删去的字符
      if (pos.count(chs[i].second))
        for (auto idx : pos[chs[i].second]) trash.push_back(idx);
    }
  }

  cout << ans << endl;
  for (int i = 0; i < ans; i++) s[trash[i]] = need[i];
  cout << s << endl;
}

int main() {
  CaseT
    solve();
}
```

## 8.7.7 Going to the Cinema

原题指路:https://codeforces.com/contest/1782/problem/B

### 题意 (2 s)

编号$1 \sim n$的$n$个人计划去电影院,每个人可选择去或不去.$i$号人有一个参数$a_i$,表示自己去当且仅当除自己外至少还有$a_i$个人去,这表明$i$号人会伤心,如果满足如下两条件之一:①自己去,但除自己外去的人数$< a_i$;②自己不去,但除自己外去的人数$\geq a_i$.问有多少种选择去的人的集合的方案,使得没有人伤心.

有$t$ $(1 \leq t \leq 1e4)$组测试数据.每组测试数据第一行输入一个整数$n$ $(2 \leq n \leq 2e5)$.第二行输入$n$个整数$a_1, \cdots, a_n$ $(0 \leq a_i \leq n-1)$.数据保证所有测试数据的$n$之和不超过$2e5$.

### 思路

设去的人的集合$A = \{p_1, \cdots, p_a\}$,不去的人的集合$B = \{q_1, \cdots, q_b\}$,则$A \bigcup B = \{1, \cdots, n\}, a + b = n$.

①对$A$中的每个人$a_i$,他去的充要条件为$a - 1 \geq a_i$,则集合$A$合法当且仅当$a - 1 \geq \max\limits_{1 \leq i \leq a} a_{p_i}$.

②对$B$中的每个人$a_j$,他不去的充要条件为$a < a_j$,则集合$B$合法当且仅当$a < \min\limits_{1 \leq j \leq b} a_{q_j}$.

枚举去的人数$i \in [0, n]$.将$a[]$非降序排列后,对每个下标$i$,去的人的集合$A = \{a_1, \cdots, a_{i-1}\}$,不去的人的集合$B = \{a_i, \cdots, a_n\}$,则$(i-1) - 1 \geq a_{i-1}, (i-1) < a_i$.

### 代码

```
void solve() {
  int n; cin >> n;
  vector<int> a(n + 2);
  a[0] = -INF, a[n + 1] = INF;  // 哨兵
  for (int i = 1; i <= n; i++) cin >> a[i];
  sort(a.begin() + 1, a.end());

  int ans = 0;
  for (int i = 1; i <= n + 1; i++)  // 枚举去的人数+1
    ans += (i - 2 >= a[i - 1]) && (i - 1 < a[i]);
  cout << ans << endl;
}

```

```
int main() {
  CaseT
    solve();
}
```

## 8.7.8 Find The Array

原题指路:https://codeforces.com/problemset/problem/1463/B

### 题意 (2 s)

给定一个长度为$n$的序列$a_1,\cdots,a_n\ \ (1\le a_i\le 1e9)$,设$a[]$的元素之和为$s$.构造一个长度为$n$的序列$b_1,\cdots,b_n\ \ (1\le b_i\le b_i)$,满足如下两个条件:①任意两个相邻元素间都存在整除关系;②$2\sum_{i=1}^{n}|a_i-b_i|\le s$.可以证明一定有解.

有$t\ \ (1\le t\le 1000)$组测试数据.每组测试数据第一行输入一个整数$n\ \ (2\le n\le 50)$.第二行输入$n$个整数$a_1,\cdots,a_n\ \ (1\le a_i\le 1e9)$.

### 思路I

[**引理**] 对$\forall a\in\mathbb{Z}^+$,区间$\left[\frac{a}{2},\frac{3a}{2}\right]$中至少存在一个2的幂次.

[**证**] 若不然,设$2^k<\frac{a}{2}\cdots①,\ 2^{k+1}>\frac{3a}{2}$,其中后式等价于$\frac{1}{2^{k+1}}<\frac{2}{3a}\cdots②$.

①乘②得:$\frac{1}{2}<\frac{1}{3}$,矛盾.

显然$2\cdot|a_i-b_i|\le a_i\ \ (1\le i\le n)$时满足要求,此时$b_i\in\left[\frac{a_i}{2},\frac{3a_i}{2}\right]$,此处只考察$\frac{3}{2}a_i\le 1e9$的情况,$\frac{3}{2}a_i>1e9$时类似.

由**引理**:区间$\left[\frac{a_i}{2},\frac{3a_i}{2}\right]$中至少存在一个2的幂次,进而存在$b_i\ \ (1\le i\le n)$都取2的幂次的合法方案.

对每个$a_i$,暴力枚举2的幂次即可.

### 代码I

```
void solve() {
  CaseT{
    int a; cin >> a;

    int ans = 1;
    while (2 * ans < a) ans *= 2;
    cout << ans << ' ';
  }
  cout << endl;
}

int main() {
  CaseT
    solve();
```

```
15 }
```

### 思路II

注意到每对相邻元素中有一个为1时一定有整除关系,为使得$\sum_{i=1}^{n}|a_i-b_i|$不过大,考虑尽量保留$a[]$中的元素.

考察序列$[a_1,1,a_3,1,\cdots]$和$[1,a_2,1,a_4,\cdots]$,下证这两个序列中至少有一个序列满足条件$2\sum_{i=1}^{n}|a_i-b_i|\le s$.

[**证**] 设$a[]$的奇数、偶数下标的元素和分别为$s_{odd}$、$s_{even}$.

因$s=s_{odd}+s_{even}$,由反证法:$s_{odd}\le\frac{s}{2}$或$s_{even}\le\frac{s}{2}$.WLOG,不妨设$s_{odd}\le\frac{s}{2}$.

对序列$[1,a_2,1,a_4,\cdots]$,有$\sum_{i=1}^{n}|a_i-b_i|=\sum_{i\in odds}|a_i-1|\le s_{odd}\le\frac{s}{2}$,则$2\sum_{i=1}^{n}|a_i-b_i|\le s$.

### 代码II

```cpp
void solve() {
  int n; cin >> n;
  ll odd = 0, even = 0;  // a[]的奇数、偶数下标的元素之和
  vector<int> a(n + 1);
  for (int i = 1; i <= n; i++) {
    cin >> a[i];

    if (i & 1) odd += a[i];
    else even += a[i];
  }

  if (odd < even) {
    for (int i = 1; i <= n; i++)
      cout << (i & 1 ? 1 : a[i]) << " \n"[i == n];
  }
  else {
    for (int i = 1; i <= n; i++)
      cout << (i & 1 ? a[i] : 1) << " \n"[i == n];
  }
}

int main() {
  CaseT
    solve();
}
```

## 8.7.9 LIS or Reverse LIS?

原题指路:https://codeforces.com/problemset/problem/1682/C

### 题意

对一个长度为$n$的序列$a=[a_1,\cdots,a_n]$,定义序列$a'=[a_n,\cdots,a_1]$,定义$LIS(a)$为$a[]$的最长严格上升子列的长度.定义序列$a=[a_1,\cdots,a_n]$的权值为$\min\{LIS(a),LIS(a')\}$.给定一个序列$a_1,\cdots,a_n$,现可将其任意排序,求其权值的最大值.

有$t$ $(1\le t\le 1e4)$组测试数据.每组测试数据第一行输入一个整数$n$ $(1\le n\le 2e5)$.第二行输入$n$个整数$a_1,\cdots,a_n$ $(1\le a_i\le 1e9)$.数据保证所有测试数据的$n$之和不超过$2e5$.

### 思路

对序列$a[]$,定义$LDS(a)$为$a[]$的最长严格下降子列的长度,则$LDS(a)=LIS(a')$.

注意到$a[]$的LIS和LDS至多有一个公共下标,且可出现公共下标时,保留公共下标更优.显然应尽量选只出现一次的元素作为LIS和LDS的公共元素,因为出现次数$\ge 2$的元素可作为非公共下标的元素分别在LIS和LDS中出现.

设出现次数$=1$的元素个数为$one$,出现次数$>1$的元素个数为$others$,则$ans=\left\lceil\frac{one}{2}\right\rceil+others$,即将出现次数为1的元素尽量平分到LIS和LDS中.

### 代码

```cpp
void solve() {
  map<int, int> cnt;
  CaseT{
    int a; cin >> a;
    cnt[a]++;
  }

  int one = 0, others = 0;
  for (auto [u, v] : cnt) {
    one += v == 1;
    others += v > 1;
  }
  cout << (one + 1) / 2 + others << endl;
}

int main() {
  CaseT
    solve();
}
```

---

## 8.7.10 The Forbidden Permutation

原题指路:https://codeforces.com/contest/1778/problem/B

### 题意 (2 s)

给定一个$1\sim n$的排列$p_1,\cdots,p_n$、一个长度为$m$且元素相异的序列$a_1,\cdots,a_m$ $(1\le a_i\le n)$和一个整数$d$,设$pos(x)$表示数$x$在$p[]$中的下标.称$a[]$是不好的,如果对$\forall i\in[1,m-1]$,都有$pos(a_i)<pos(a_{i+1})\le pos(a_i)+d$,否则称$a[]$是好的.现可进行若干次操作,每次操作可交换$a[]$中的相邻两元素,求使得$a[]$是好的所需的最小操作次数.

有$t$ $(1\le t\le 1e4)$组测试数据.每组测试数据第一行输入三个整数$n,m,d$ $(2\le m\le n\le 2e5,1\le d\le n)$.第二行输入一个$1\sim n$的排列$p_1,\cdots,p_n$.第三行输入$m$个相异的整数$a_1,\cdots,a_m$ $(1\le a_i\le n)$.数据保证所有测试数据的$n$之和不超过$5e5$.

## 思路

对初始序列$a[]$,

(1)若$\exists i \in [1, m-1]\ s.t.\ pos(a_i) \geq pos(a_{i+1})$或$pos(a_{i+1}) > pos(a_i) + d$,则$a[]$已是好的.

(2)若不然,则只需通过操作产生至少一个满足$pos(a_i) \geq pos(a_{i+1})$或$pos(a_{i+1}) > pos(a_i) + d$的下标$i$即可.

对每个$i \in [1, m-1]$,需使得$pos(a_i) \geq pos(a_{i+1})$或$pos(a_{i+1}) > pos(a_i) + d$,

则$ans$是所有下标所需的操作次数的最小值.

①使得$pos(a_i) \geq pos(a_{i+1})$所需的最小操作次数为$pos(a_{i+1}) - pos(a_i)$.

②使得$pos(a_{i+1}) > pos(a_i) + d$所需的最小操作次数为$d + 1 - [pos(a_i) - pos(a_{i+1})]$,

该情况合法当且仅当$d + 2 \leq n$,因为两数在$p[]$中至多相距$(n-1)$.

## 代码

```cpp
void solve() {
  int n, m, d; cin >> n >> m >> d;
  vector<int> pos(n + 1), a(m + 1);
  for (int i = 1; i <= n; i++) {
    int x; cin >> x;
    pos[x] = i;
  }
  for (int i = 1; i <= m; i++) cin >> a[i];

  int ans = INF;
  for (int i = 2; i <= m; i++) {
    if (pos[a[i]] > pos[a[i - 1]] && pos[a[i]] <= pos[a[i - 1]] + d) {
      ans = min(ans, pos[a[i]] - pos[a[i - 1]]);
      if (d + 2 <= n) ans = min(ans, d + 1 - (pos[a[i]] - pos[a[i - 1]]));
    }
    else ans = 0;
  }
  cout << ans << endl;
}

int main() {
  CaseT
    solve();
}
```

---

# 8.7.11 Sum of Substrings

原题指路: https://codeforces.com/problemset/problem/1691/C

## 题意

给定一个长度为$n$的0/1串$s = s_1 \cdots s_n$.定义序列$d = [d_1, \cdots, d_{n-1}]$,其中$d_i\ \ (1 \leq i \leq n-1)$表示十进制数$\overline{s_i s_{i+1}}$.定义函数$f(s) = \sum_{i=1}^{n-1} d_i$.现有操作:交换$s$中相邻的数位.问至多$k$次操作后$f(s)$能达到的最小值.

有$t\ \ (1 \leq t \leq 1e5)$组测试数据.每组测试数据第一行输入两个整数$n, k\ \ (2 \leq n \leq 1e5, 0 \leq k \leq 1e9)$.第二行输入一个长度为$n$的0/1串$s$.数据保证所有测试数据的$n$之和不超过$1e5$.

## 思路

注意到$f(s)=\sum_{i=1}^{n-1}(10\cdot d_i+d_{i+1})=10\cdot d_1+11\sum_{i=2}^{n-1}d_i+d_n$,为使得$f(s)$减小,可将中间的1与首尾的0交换,且应优先考虑将中间的1与末尾的0交换.

## 代码

```cpp
void solve() {
  int n, k; string s; cin >> n >> k >> s;
  s = " " + s;

  if (s[n] == '0') {  // 优先考虑将中间的1与末尾的0交换
    for (int i = n - 1; i; i--) {
      if (s[i] == '1') {
        int dis = n - i;
        if (k >= dis) {
          k -= dis;
          swap(s[n], s[i]);
        }
        break;
      }
    }
  }

  if (s[1] == '0') {
    for (int i = 2; i <= n - 1; i++) {
      if (s[i] == '1') {
        int dis = i - 1;
        if (k >= dis) {
          k -= dis;
          swap(s[1], s[i]);
        }
        break;
      }
    }
  }

  int ans = 10 * (s[1] - '0') + (s[n] - '0');
  for (int i = 2; i <= n - 1; i++) ans += 11 * (s[i] - '0');
  cout << ans << endl;
}

int main() {
  CaseT
    solve();
}
```

# 8.7.12 Fishingprince Plays With Array

原题指路:https://codeforces.com/problemset/problem/1696/C

## 题意 (2 s)

给定一个长度为$n$的序列$a=[a_1,\cdots,a_n]$和一个整数$m$.

现有如下两种操作,其中$n$为此时序列$a[]$的长度:

①选择一个$m$的倍数$a_i$ $(1\le i\le n)$,将其替换为$m$个$\frac{a_i}{m}$.

②选择一个下标$i$ $(1\le i\le n-m+1)$ $s.t.$ $a_i=\cdots=a_{i+m-1}$,将它们替换为1个$m\cdot a_i$.

给定一个长度为$k$的序列$b=[b_1,\cdots,b_k]$,问经过若干次操作能否将$a[]$变为$b[]$.

有$t$ $(1\le t\le 1e4)$组测试数据.每组测试数据第一行输入两个整数$n,m$ $(1\le n\le 5e4,2\le m\le 1e9)$.第二行输入$n$个整数$a_1,\cdots,a_n$ $(1\le a_i\le 1e9)$.第三行输入一个整数$k$ $(1\le k\le 5e4)$.第四行输入$k$个整数$b_1,\cdots,b_k$ $(1\le b_i\le 1e9)$.数据保证所有测试数据的$(n+k)$之和不超过$2e5$.

## 思路

注意到操作①和操作②互逆,则只需考察$a[]$和$b[]$分别只进行一种操作直至不能再进行该操作时得到的序列$a'[]$和$b'[]$是否相等.因操作②涉及到下标的选取,而操作①不涉及,故考察$a[]$混合$b[]$只进行操作①直至不能再进行时得到的序列是否相等.

每次都暴力检查两个序列是否相等会TLE.考虑优化,将每个序列元素分解为$p_i\cdot m^{s_i}$的形式后,只需比较两序列的$p_i$和$s_i$是否对应相等即可.时间复杂度$O((n+k)\log m)$.

## 代码

```cpp
void solve() {
  int n, m; cin >> n >> m;
  vector<int> a(n);
  for (auto& ai : a) cin >> ai;
  int k; cin >> k;
  vector<int> b(k);
  for (auto& bi : b) cin >> bi;

  auto get = [&](auto a) {
    vector<pair<int, ll>> res;
    for (int i = 0; i < a.size(); i++) {
      int p = a[i], s = 1;
      while (p % m == 0) {
        p /= m;
        s *= m;
      }

      if (res.size() && res.back().first == p) res.back().second += s;
      else res.push_back({ p, s });
    }
    return res;
  };

  cout << (get(a) == get(b) ? "Yes" : "No") << endl;
}

int main() {
```

```
  CaseT
    solve();
}
```

## 8.7.13 Suitable Replacement

原题指路:https://codeforces.com/problemset/problem/825/D

### 题意

给定一个只包含小写英文字母的字符串$t$ $(1 \leq |t| \leq 1e6)$、只包含小写英文字母和字符'?'的字符串$s$ $(1 \leq |s| \leq 1e6)$. 现将$s$中的'?'替换为任意字符后任意重排, 使得$s$包含的不相交的子串$t$的个数最多, 输出一个方案.

### 思路

注意到$s$包含的不相交的子串$t$的个数只与$s$和$t$中每种字符出现的次数有关, 且$s$和$t$中每种字符的个数的比例越接近, $s$包含的不相交的子串$t$的个数越多. 具体地, 对字符$ch$, 设它在$s$和$t$中分别出现$cnt_1[ch]$和$cnt_2[ch]$次, 则$\frac{cnt_1[ch]}{cnt_2[ch]}$越大, $s$包含的不相交的子串$t$的个数越多. 故贪心策略: 对$s$中的每个'?', 选一个$ratio = \frac{cnt_1[ch]}{cnt_2[ch]}$最小的字符$ch$, 将'?'替换为该字符.

### 代码

```
const int MAXZ = 26;

void solve() {
  string s, t; cin >> s >> t;

  vector<int> cnt1(MAXZ), cnt2(MAXZ);
  for (auto ch : s)
    if (ch != '?') cnt1[ch - 'a']++;
  for (auto ch : t) cnt2[ch - 'a']++;

  for (auto& ch : s) {
    if (ch != '?') continue;

    double ratio = INF;  // cnt1[ch] / cnt2[ch]最小的ch
    int c;
    for (int i = 0; i < MAXZ; i++) {
      if (ratio * cnt2[i] > cnt1[i]) {
        c = i;
        ratio = (double)cnt1[i] / cnt2[i];
      }
    }

    ch = c + 'a';
    cnt1[c]++;
  }
  cout << s << endl;
}

int main() {
  solve();
}
```

## 8.7.14 Meximum Array

原题指路: https://codeforces.com/problemset/problem/1628/A

### 题意 (2 s)

给定一个长度为$n$的非负整数序列$a=[a_1,\cdots,a_n]$. 现用如下步骤构造序列$b[]$: ①取整数$k\in[1,|a|]$, 其中$|a|$为$a[]$此时的长度; ②将$a[]$的前$k$个元素的mex值加入$b[]$, 并删除$a[]$的前$k$个元素. 求可得到的字典序最大的$b[]$.

有$t$ $(1\le t\le 100)$组测试数据. 每组测试数据第一行输入一个整数$n$ $(1\le n\le 2e5)$. 第二行输入$n$个非负整数$a_1,\cdots,a_n$ $(0\le a_i\le n)$. 数据保证所有测试数据的$n$之和不超过$2e5$.

### 思路I

显然应使得每次加入$b[]$的值尽量大. 贪心策略: 每次取最短的、可取得最大mex值的$a[]$的前缀. 时间复杂度$O(n)$.

### 代码I

```cpp
void solve() {
    int n; cin >> n;
    vector<int> a(n), cnt(n + 1);
    for (auto& ai : a) {
        cin >> ai;
        cnt[ai]++;
    }

    vector<int> ans;
    for (int i = 0; i < n;) {
        int mex = 0;
        while (cnt[mex]) mex++;

        vector<bool> have(mex);
        int tot = 0;
        while (i < n) {
            if (a[i] < mex && !have[a[i]]) {
                have[a[i]] = true;
                tot++;
            }

            cnt[a[i]]--;
            i++;
            if (tot == mex) break;
        }
        ans.push_back(mex);
    }

    cout << ans.size() << endl;
    for (int i = 0; i < ans.size(); i++)
        cout << ans[i] << " \n"[i == (int)ans.size() - 1];
}

int main() {
    CaseT
    solve();
}
```

## 思路II

用BIT维护序列中每个元素是否出现, 因元素值域包含$0$, 故整体加一个$1$的偏移量. 在BIT上二分可求得剩余元素的mex值, 注意二分边界也要加偏移量和特判$mex = 0$的情况. 时间复杂度$O(n \log^2 n)$.

## 代码II

```cpp
struct FenwickTree {
    int n;
    vector<int> BIT;

    FenwickTree(int _n) :n(_n) {
        BIT.resize(n + 1);
    }

    int lowbit(int x) {
        return x & -x;
    }

    void modify(int pos, int val) {
        for (int i = pos; i <= n; i += lowbit(i))
            BIT[i] += val;
    }

    int query(int pos) {
        int res = 0;
        for (int i = pos; i; i -= lowbit(i))
            res += BIT[i];
        return res;
    }
};

void solve() {
    int n; cin >> n;
    vector<int> a(n);
    FenwickTree bit(n + 1);  // 注意值域开到(n + 1)
    vector<int> cnt(n + 1);
    for (int i = 0; i < n; i++) {
        cin >> a[i];

        if (!cnt[a[i]])
            bit.modify(a[i] + 1, 1);  // BIT中元素整体加1的偏移量
        cnt[a[i]]++;
    }

    auto get = [&]() {  // 求剩余元素的mex
        int l = 1, r = n + 1;  // 注意值域整体加1的偏移量
        while (l < r) {
            int mid = l + r + 1 >> 1;
            if (bit.query(mid) == mid) l = mid;
            else r = mid - 1;
        }

        if (l == 1 && bit.query(l) < l) l = 0;  // 特判mex = 0的情况
        return l;
```

```
    };

    vector<int> ans;
    for (int i = 0; i < n;) {
        int mex = get();
        vector<bool> have(mex);
        int tot = 0;
        while (i < n) {
            if (a[i] < mex && !have[a[i]]) {
                have[a[i]] = true;
                tot++;
            }

            if (--cnt[a[i]] == 0)
                bit.modify(a[i] + 1, -1);
            i++;
            if (tot == mex) break;
        }
        ans.push_back(mex);
    }

    cout << ans.size() << endl;
    for (int i = 0; i < ans.size(); i++)
        cout << ans[i] << " \n"[i == (int)ans.size() - 1];
}

int main() {
    CaseT
    solve();
}
```

## 8.7.15 Road to Post Office

原题指路: https://codeforces.com/contest/702/problem/D

### 题意

某人距离目的地$d$ $(1 \le d \le 1e12)$米, 他的车每开$k$ $(1 \le k \le 1e6)$米需花时间$t$ $(1 \le t \le 1e6)$修复, 修复后可再开$k$米, 但$k$米后需再修复, 初始时他的车已修复(时间不计). 他开车的速度为$a$秒/米, 走路速度为$b$ $(1 \le a < b \le 1e6)$秒/米, 他在路上随时可以弃车走路. 求他到达目的地所需的最少时间.

### 思路

按是否修车分类.

(1)不修车

①$d < k$, 即不修车也能走完全程, 则$ans = d \cdot a$. 注意$d = k$时, 题目认为到达目的地时也需修车.

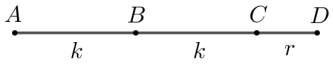
②先开$k$米后弃车走路, 则$t + k \cdot a > k \cdot b$, 即$t > k(b - a)$, 此时$ans = d \cdot a + (d - k) \cdot b$.

(2)修车

①修车使得全程开车, 此时$ans = d \cdot a + \left\lfloor \frac{d}{k} \right\rfloor \cdot t$.

②前面都开车, 到最后一段不修车, 改为走路, 此时$ans = (d - d\%k) \cdot a + d\%k \cdot b + \left( \left\lfloor \frac{d}{k} - 1 \right\rfloor \right) \cdot t$.

③前面都开车, 到某段不修车, 改为走路. 不失一般性, 考察下图所示的情况:




不妨设$A$时刻花$t$的时间修车后开车到$B$, 不修车, 改为走路, 此时$ans' = k \cdot a + t + (k + r) \cdot b$.

若用(2)②的策略, 所需时间$ans = 2 \cdot k \cdot a + 2 \cdot t + r \cdot b$.

因$ans - ans' = k(a - b) + t$, 而$t \le k(b - a)$, 则$ans \le ans'$, 故(2)②的策略更优.

综上, 特判(1)①的情况后, 对其余情况的答案取$\min$即可.

### 代码

```cpp
void solve() {
    ll d; int k, a, b, t; cin >> d >> k >> a >> b >> t;

    if (d < k) {
        cout << d * a << endl;
        return;
    }

    cout << min({
        (ll)k * a + (d - k) * b,  // 不修车
        (ll)d * a + d / k * t,  // 全程开车
        (ll)(d - d % k) * a + d % k * b + (d / k - 1) * t
        }) << endl;
}

int main() {
    solve();
}
```

---

## 8.7.16 Luba And The Ticket

原题指路: https://codeforces.com/problemset/problem/845/B

### 题意 (2 s)

称一个6位整数(可能包含前导零)是好的, 如果前三个数码之和与后三个数码之和相等.

给定一个6位整数(可能包含前导零). 现有操作: 修改该整数的一个数位. 求将其变为好的所需的最小操作次数.

### 思路I

枚举所有好的6位整数, 更新答案.

总时间复杂度$O(cnt) = O(1)$, 其中$cnt$为好的整数的个数.

## 代码I

```cpp
void solve() {
    string n; cin >> n;

    auto get = [&](int x) {  // 将整数转为6位字符串
        string res = to_string(x);
        return string(max((int)6 - (int)res.length(), 0), '0') + res;
    };

    auto check = [&](string s) {  // 检查字符串是否是好的
        return s[0] + s[1] + s[2] == s[3] + s[4] + s[5];
    };

    int ans = 6;
    for (int i = 0; i <= 999999; i++) {
        auto m = get(i);
        if (check(m)) {
            int res = 0;
            for (int j = 0; j < 6; j++)
                res += n[j] != m[j];
            ans = min(ans, res);
        }
    }
    cout << ans << endl;
}

int main() {
    solve();
}
```

## 思路II

将每个6位整数作为一个节点, 从输入串开始BFS, 求输入串到任一好的节点的最短路.

总时间复杂度$O(cnt) = O(1)$, 其中$cnt$为好的整数的个数.

## 代码II

```cpp
void solve() {
    string n; cin >> n;

    queue<string> que;
    que.push(n);

    map<string, int> dis;
    dis[n] = 0;

    auto check = [&](string s) {  // 检查字符串是否是好的
        return s[0] + s[1] + s[2] == s[3] + s[4] + s[5];
    };

    while (que.size()) {
        auto tmp = que.front(); que.pop();
        if (check(tmp)) {
            cout << dis[tmp] << endl;
```

```
            return;
        }

        for (int i = 0; i < 6; i++) {
            for (char ch = '0'; ch <= '9'; ch++) {
                if (ch != tmp[i]) {
                    string cur = tmp;
                    cur[i] = ch;

                    if (!dis.count(cur)) {
                        dis[cur] = dis[tmp] + 1;
                        que.push(cur);
                    }
                }
            }
        }
    }
}

int main() {
    solve();
}
```

## 思路III

设输入的整数的前三个数码之和为$fir$, 后三个数码之和为$sec$. 若$fir = sec$, 则无需操作; 若$fir \neq sec$, 由对称性, 不妨设$fir < sec$, 否则交换整数的前三个数码和后三个数码即可, 显然不影响答案.

为使得$fir = sec$, 可选择增大整数的前三个数码或减小整数的后三个数码. 对前三个数码, 每个数码至多增大到$9$, 则每个数码$d$对前三个数码之和的最大增加量为$(9 - d)$. 同理对后三个数码, 每个数码$d$对后三个数码之和的最大减小量为$d$. 预处理出每个数码对所在的半边的和的最大改变量, 将贡献非升序排列后, 优先用较大贡献的数码来减小$diff = |fir - sec|$, 直至$diff \leq 0$.

总时间复杂度$O(6 \log 6) = O(1)$.

## 代码III

```
void solve() {
    string n; cin >> n;

    int fir = 0, sec = 0;
    for (int i = 0; i < 6; i++) {
        if (i < 3) fir += n[i] - '0';
        else sec += n[i] - '0';
    }

    if (fir > sec) {
        swap(fir, sec);
        n = n.substr(3) + n.substr(0, 3);
    }

    vector<int> delta(6);  // delta[i]表示n[i]对diff = |fir - sec|的最大改变量
    for (int i = 0; i < 6; i++) {
        if (i < 3) delta[i] = '9' - n[i];
        else delta[i] = n[i] - '0';
    }
```

```
    sort(rall(delta));

    int ans = 0;
    int diff = sec - fir;
    for (auto d : delta) {
        if (diff <= 0) break;

        ans++;
        diff -= d;
    }
    cout << ans << endl;
}

int main() {
    solve();
}
```

---

## 8.7.17 Light It Up

原题指路: https://codeforces.com/problemset/problem/1000/B

### 题意

给定两个整数$n, m$ $(1 \le n \le 1e5, 2 \le m \le 1e9)$. 某灯在0时刻亮, 在$a_1, \cdots, a_n$ $(0 < a_1 < \cdots < a_n < m)$时刻反转状态. 现有操作: 在区间$(0, m)$中的不在$a[]$中的整点处添加一个反转状态的时间点. 求至多进行一次操作后, 灯亮的总时间的最大值.

### 思路

设$a_0 = 0, a_{n+1} = m$. 若在$a_{i-1}$ $(1 \le i \le n + 1)$与$a_i$时刻间添加一个反转状态的时间点$x$, 显然最优策略$x = a_{i-1} + 1$或$x = a_i - 1$. 枚举所有添加的位置, 更新答案即可.

### 代码

```
void solve() {
    int n, m; cin >> n >> m;
    vector<int> a(n + 2);
    for (int i = 1; i <= n; i++) cin >> a[i];
    a[n + 1] = m;  // 边界, 省略a[0] = 0

    vector<int> pre(n + 2);  // 灯亮的时间的前缀和
    for (int i = 1; i <= n + 1; i++)
        pre[i] = pre[i - 1] + (i & 1 ? a[i] - a[i - 1] : 0);

    int ans = pre[n + 1];  // 不操作
    for (int i = 1; i <= n + 1; i++) {
        if (a[i - 1] + 1 <= m && a[i - 1] + 1 != a[i])  // 在x = a[i - 1] + 1处添加
            ans = max(ans, pre[i - 1] + (m - a[i - 1] - 1 - (pre[n + 1] - pre[i - 1])));
        if (a[i] - 1 >= 0 && a[i] - 1 != a[i - 1])  // 在x = a[i] - 1处添加
            ans = max(ans, pre[i - 1] + (m - a[i] + 1 - (pre[n + 1] - pre[i - 1])));
    }
    cout << ans << endl;
}

int main() {
```

```
    solve();
}
```

## 8.7.18 Annoying Present

原题指路: https://codeforces.com/problemset/problem/1009/C

### 题意 (2 s)

有一个长度为$n$ $(1 \le n \le 1e5)$的序列$a = [a_1, \cdots, a_n]$, 初始时$a[]$的元素都为0. 现有$m$ $(1 \le m \le 1e5)$个操作, 每次操作给定两个整数$x, d$ $(-1000 \le x, d \le 1000)$, 表示先取定一个下标$i \in [1, n]$, 再对所有的$j \in [1, n]$, 令$a_i += x + d \cdot |i - j|$. 求所有操作结束后$a[]$的算术平均值的最大值, 误差不超过$1e-6$.

### 思路

因$a[]$的元素个数固定, 为使得$a[]$的算术平均值最大, 应使得$\sum_{j=1}^{n} a_j$最大.

因每个操作中$x$为定值, 为使得$\sum_{j=1}^{n} a_j$最大, 应使得$d \cdot \sum_{j=1}^{n} |i-j|$最大.

显然$\sum_{j=1}^{n} |i-j|$是关于$i$的二次函数. 按$d$的正负分为如下三种情况:

(1)$d < 0$时, 应使得$\sum_{j=1}^{n} |i-j|$最小, 则$i$取抛物线的对称轴附近的整点时$\sum_{j=1}^{n} |i-j|$取得最小值.

①$n = 2k+1$ $(k \in \mathbb{N}^*)$时, 取$i = k+1$, 此时$d \cdot \sum_{j=1}^{n} |i-j|$取得最大值:

$$d \cdot \sum_{j=1}^{n} |i-j| = d \cdot \sum_{j=1}^{2k+1} |k+1-j| = d \cdot \left[ \sum_{j=1}^{k} (k+1-j) + \sum_{j=k+2}^{2k+1} (j-k-1) \right]$$
$$= d \cdot \left( \frac{(k+1) \cdot k}{2} + \frac{(1+k) \cdot k}{2} \right) = d \cdot \frac{n-1}{2} \cdot \frac{n+1}{2}.$$

②$n = 2k$ $(k \in \mathbb{N}^+)$时, 取$i = k$, 此时$d \cdot \sum_{j=1}^{n} |i-j|$取得最大值:

$$d \cdot \sum_{j=1}^{n} |i-j| = d \cdot \sum_{j=1}^{2k} |k-j| = d \cdot \left[ \sum_{j=1}^{k-1} (k-j) + \sum_{j=k+1}^{2k} (j-k) \right]$$
$$= d \cdot \left( \frac{(k-1+1) \cdot (k-1)}{2} + \frac{(1+k) \cdot k}{2} \right) = d \cdot \left( \frac{n}{2} \right)^2.$$

(2)$d = 0$时, $d \cdot \sum_{j=1}^{n} |i-j|$对答案的贡献为0.

(3)$d > 0$时, 应使得$\sum_{j=1}^{n} |i-j|$最大, 则$i = 1$或$i = n$时$\sum_{j=1}^{n} |i-j|$取得最大值.

①$i = 1$时, $d \cdot \sum_{j=1}^{n} |i-j| = d \cdot \sum_{j=1}^{n} |1-j| = d \cdot \sum_{j=2}^{n} (j-1) = d \cdot \frac{n(n-1)}{2}$.

②$i=n$时, $d\cdot\sum_{j=1}^{n}|i-j|=d\cdot\sum_{j=1}^{n}|n-j|=d\cdot\sum_{j=1}^{n-1}(n-j)=d\cdot\frac{n(n-1)}{2}$.

### 代码

```cpp
void solve() {
    int n, m; cin >> n >> m;

    ll pos = (ll)n * (n - 1) / 2;
    ll neg = n & 1 ? (ll)(n - 1) * (n + 1) / 4 : (ll)n * n / 4;

    ll ans = 0;
    while (m--) {
        int x, d; cin >> x >> d;

        ans += n * x;
        if (d <= 0) ans += neg * d;
        else ans += pos * d;
    }
    cout << fixed << setprecision(12) << (double)ans / n << endl;
}

int main() {
    solve();
}
```

---

## 8.7.19 Math Show

原题指路: https://codeforces.com/problemset/problem/846/B

### 题意

有$n$ $(1\le n\le 45)$个任务, 每个任务有编号$1\sim k$的$k$ $(1\le k\le 45)$个子任务构成, 完成子任务$i$ $(1\le i\le k)$耗时$t_i$ $(1\le t_i\le 1e6)$. 完成每个子任务得1分, 完整地完成一个任务, 即完成其$k$个子任务可多得1分, 即完整地完成一个任务得分为$(k+1)$. 现可以任意顺序完成任务, 求$m$ $(0\le m\le 2e9)$个单位时间的最大得分.

### 思路

易想到背包, 但$m\le 2e9$, 会TLE.

若完整地完成一个任务无额外得分, 则应贪心地完成耗时少的任务.

有额外得分时, 这样的贪心是错误的, 反例如下:

```
2 2 5
2 3
```

①若采用上述的贪心策略, 则应完成2个子任务1, 得分为2.

②若完整地完成一个任务, 则得分为$3>2$.

因$n\le 45$, 考虑枚举完整地完成的任务数$i\in[0,n]$, 再对剩余的时间贪心, 并更新答案.

总时间复杂度$O(n^2\cdot k)$.

### 代码

```cpp
void solve() {
    int n, sub, m; cin >> n >> sub >> m;  // 任务数、子任务数、总时间
    vector<int> t(sub);
    int sum = 0;
    for (auto& ti : t) {
        cin >> ti;
        sum += ti;
    }
    sort(all(t));

    int ans = 0;
    for (int i = 0; i <= n; i++) {  // 枚举完整完成的任务数
        int res = (sub + 1) * i, cost = sum * i;  // 得分、耗时
        if (cost > m) break;  // cost = m时也需更新答案

        for (int j = 0; j < sub; j++) {
            for (int k = 0; k < n - i; k++) {
                if ((cost += t[j]) <= m) res++;  // cost = m时也需更新答案
                else break;
            }
            if (cost >= m) break;  // cost = m时的答案已更新
        }
        ans = max(ans, res);
    }
    cout << ans << endl;
}

int main() {
    solve();
}
```

---

## 8.7.20 Flipper

原题指路: https://codeforces.com/contest/1833/problem/D

### 题意

有 $t$ $(1 \le t \le 1000)$ 组测试数据. 每组测试数据给定一个 $1 \sim n$ $(1 \le n \le 2000)$ 的排列 $p = [p_1, \cdots, p_n]$ . 现做如下操作恰一次: ① 选择区间 $[l, r]$ $(1 \le l \le r \le n)$ , 将区间 $p[l \cdots r]$ 反转; ② 交换前缀 $p[1 \cdots (l-1)]$ 和后缀 $p[(r+1) \cdots n]$ . 求能得到的字典序最大的序列. 数据保证所有测试数据的 $n$ 之和不超过 $2000$ .

### 思路

显然应贪心地将值 $n$ 或值 $(n-1)$ 移动到下标 $1$ 处. 因不易找到同时贪心操作区间 $[l, r]$ 的两端点的策略, 考虑贪心 $r$ 并枚举 $l$ , 暴力比较字典序.

根据值 $n$ 的位置分类:

① 若值 $n$ 不在下标 $1$ 处, 则可使得值 $n$ 移动到下标 $1$ 处.

设其在原序列中的下标为 $pos$ , 则取 $r = pos - 1$ .

② 若值 $n$ 在下标 $1$ 处, 则只能使得值 $(n-1)$ 移动到下标 $1$ 处.

设其在原序列中的下标为 $pos$ , 则取 $r = pos - 1$ .

特别地, $pos = n$ 时, 取 $r = n$ 也可使得 $p_n$ 移动到下标 $1$ 处, 故取 $r = pos$ .

时间复杂度 $O(n^2)$ .

### 代码

```cpp
void solve() {
    int n; cin >> n;
    vector<int> a(n + 1);
    for (int i = 1; i <= n; i++) cin >> a[i];

    int pos = find(all(a), n) - a.begin();
    if (pos == 1) pos = find(all(a), n - 1) - a.begin();

    auto get = [&](int l, int r) {
        vector<int> res;
        for (int i = r + 1; i <= n; i++)
            res.push_back(a[i]);
        for (int i = r; i >= l; i--)
            res.push_back(a[i]);
        for (int i = 1; i < l; i++)
            res.push_back(a[i]);
        return res;
    };

    vector<int> ans(n, 1);  // 注意初始化为1, 否则需特判n = 1
    int r = pos == n ? pos : pos - 1;  // pos = n时a[n]可到下标1处
    for (int l = 1; l <= r; l++)
        ans = max(ans, get(l, r));

    for (int i = 0; i < n; i++)
        cout << ans[i] << " \n"[i == n - 1];
}

int main() {
    CaseT
    solve();
}
```

---

## 8.7.21 Heating

原题指路: https://codeforces.com/problemset/problem/1260/A

### 题意

有编号 $1 \sim n$ 的 $n$ $(1 \le n \le 1000)$ 个房间, 其中 $i$ $(1 \le i \le n)$ 号房间至多可安装 $c_i$ $(1 \le c_i \le 1e4)$ 个暖气, 温暖度之和要求 $\ge sum_i$ $(1 \le sum_i \le 1e4)$ . 安装一个温暖度之和为 $k$ 的暖气花费 $k^2$ 元. 求使得所有房间满足要求所需的最小花费.

### 思路

[**定理**] 每个房间中的温暖度 $x_1, \cdots, x_m$ 满足 $\max\limits_{1\le i\le m} x_i - \min\limits_{1\le i\le m} x_i < 2$ .

[**证**] 若不然, 设有两个温暖度 $(x, y)\ s.t.\ y \ge x+2$ . 考察将两个温暖度替换为 $(x+1, y-1)$ 的代价.

注意到 $(x+1)^2 + (y-1)^2 = x^2 + 2x + 1 + y^2 - 2y + 1 = (x^2+y^2) + 2(x-y+1)$

$\le (x^2+y^2) + 2(x-x-2+1) < x^2 + y^2$ , 方案更优.

故尽量均匀地分配每个房间的温暖度即可.

### 代码

```python
def solve():
    c, sum = map(int, input().split())

    d, r = sum // c, sum % c
    print((c - r) * d * d + r * (d + 1) * (d + 1))

def main():
    t = 1
    t = int(input())
    for i in range(t):
        solve()
```

---

## 8.7.22 Deadline

原题指路: https://codeforces.com/problemset/problem/1288/A

### 题意 ($2$ s)

有 $t$ $(1 \le t \le 50)$ 组测试数据. 每组测试数据给定两个整数 $n, d$ $(1 \le n, d \le 1e9)$ , 问是否 $\exists x \in \mathbb{N}\ s.t.\ x + \left\lceil \dfrac{d}{x+1} \right\rceil \le n$ .

### 思路I

因 $x + \left\lceil \dfrac{d}{x+1} \right\rceil = \left\lceil x + \dfrac{d}{x+1} \right\rceil$ , 考察函数 $f(x) = x + \dfrac{d}{x+1} = (x+1) + \dfrac{d}{x+1} - 1$ .

由均值不等式: $(x+1) + \dfrac{d}{x+1} \ge 2\sqrt{d}$ , 等号取得 iff $x + 1 = \dfrac{d}{x+1}$ , 解得: $x = \sqrt{d} - 1$ .

$x \ge \sqrt{d}$ 时, 有 $\lceil f(x) \rceil \le \lceil f(x+1) \rceil$ , 故只需遍历 $x \in \left[0, \left\lfloor \sqrt{d} \right\rfloor\right]$ 并检查 $\lceil f(x) \rceil \le n$ 是否成立.

时间复杂度 $O\left(\sqrt{d}\right)$ .

### 代码I

```cpp
void solve() {
    int n, d; cin >> n >> d;

    for (int x = 0; x * x <= d; x++) {
```

```
        if (x + (d + x + 1 - 1) / (x + 1) <= n) {
            cout << "YES" << endl;
            return;
        }
    }
    cout << "NO" << endl;
}

int main() {
    CaseT
    solve();
}
```

## 思路II

由**思路I**: 只需检查 $x = \left\lfloor \sqrt{d} - 1 \right\rfloor$ 和 $x = \left\lceil \sqrt{d} - 1 \right\rceil$ 处是否有 $f(x) \le n$ 即可.

时间复杂度 $O(1)$ .

## 代码II

```
const double eps = 0.1;

void solve() {
    int n, d; cin >> n >> d;

    auto f = [&](int x) {
        return x + (d + x + 1 - 1) / (x + 1);
    };

    cout << (f(floor(sqrt(d + eps)) - 1) <= n || f(ceil(sqrt(d + eps)) - 1) <= n ? "YES" : "NO") << endl;
}

int main() {
    CaseT
    solve();
}
```

## 思路III

$$x + \left\lceil \frac{d}{x+1} \right\rceil \le n \Leftrightarrow \left\lceil \frac{d}{x+1} \right\rceil \le n - x$$

$$\Leftrightarrow d \le (n-x)(x+1) = nx - x^2 + n - x \Leftrightarrow x^2 - (n-1)x + (d-n) \le 0 .$$

因抛物线 $y = x^2 - (n-1)x + (d-n)$ 开口向上, 对称轴 $x = -\frac{b}{2a} = \frac{n-1}{2} \ge 0$ ,

则只需 $\Delta = (n-1)^2 - 4(d-n) \ge 0$ 即可.

时间复杂度 $O(1)$ .

## 代码III

```
void solve() {
    int n, d; cin >> n >> d;
    cout << ((ll)(n - 1) * (n - 1) - (ll)4 * (d - n) >= 0 ? "YES" : "NO") << endl;
}

int main() {
    CaseT
    solve();
}
```