1. 贪心

贪心其实是一种常用的思想，就是当一种决策在某种情况下一定优于其他决策时，就说这种决策具有贪心选择性。

- **P1016 旅行家的预算**

  题目描述 展开

  一个旅行家想驾驶汽车以最少的费用从一个城市到另一个城市（假设出发时油箱是空的）。给定两个城市之间的距离$D1$、汽车油箱的容量C（以升为单位）、每升汽油能行驶的距离$D2$、出发点每升汽油价格P和沿途油站数N（N可以为零），油站i离出发点的距离Di、每升汽油价格Pi（$i = 1, 2, \ldots, N$）。计算结果四舍五入至小数点后两位。如果无法到达目的地，则输出"No Solution"。

  - 本质上是一个找油价更低加油站的过程。分两种情况讨论：
    - 当前加油站i加满油可以到达下一个价格更低的j加油站：$P_j \leq P_i$且$D_j - D_i \geq D2 * C$
      这种情况直接加到刚好走到最近的j加油站即可
    - ELSE
      加满油

- **P1080 国王的游戏**

  题目描述 展开

  恰逢 H国国庆，国王邀请n 位大臣来玩一个有奖游戏。首先，他让每个大臣在左、右手上面分别写下一个整数，国王自己也在左、右手上各写一个整数。然后，让这 n 位大臣排成一排，国王站在队伍的最前面。排好队后，所有的大臣都会获得国王奖赏的若干金币，每位大臣获得的金币数分别是：排在该大臣前面的所有人的左手上的数的乘积除以他自己右手上的数，然后向下取整得到的结果。

  国王不希望某一个大臣获得特别多的奖赏，所以他想请你帮他重新安排一下队伍的顺序，使得获得奖赏最多的大臣，所获奖赏尽可能的少。注意，国王的位置始终在队伍的最前面。

  - 考虑国王(a_0, b_0)第一个人(a_1, b_1)和第二个人(a_2, b_2)
    的位置，只要满足

    $$max(a_0/b_1, a_0a_1/b_2) \leq max(a_0/b_2, a_0a_2/b_1)$$

    化简

    $$max(1/b_1, a_1/b_2) \leq max(1/b_2, a_2/b_1)$$

  - 只要$a_1/b_2 \leq a_2/b_1$即$a_1b_1 < a_2b_2$
  - 以此类推，只要按照a_ib_i排序即可

2. 二分

在某个自变量x单调增长的情况下，函数$f(x)$是单调且易验证的，那么此时可以对x进行二分，求解满足$f(x)=y$的那个最大/最小的x

```
int l=0, r=mx, ans=0;
while (l<=r) {
    int mid=(l+r)>>1;
    if (judge(mid)){
        l=mid+1;
        ans=mid;
    } else {
        r=mid-1;
    }
}
```

- **P1102 A-B数对**

  题目描述 展开

  出题是一件痛苦的事情！

  相同的题目看多了也会有审美疲劳，于是我舍弃了大家所熟悉的 A+B Problem，改用 A-B 了哈哈！

  好吧，题目是这样的：给出一串数以及一个数字 C，要求计算出所有 $A-B=C$ 的数对的个数（不同位置的数字一样的数对算不同的数对）。

  - 本质上是寻找$B=A-C$的个数，可以用map查找X-C出现的次数。然后其实也对所有数排序后，手写二分去查找，或者直接lower_bound(x.begin(), x.end(), x[i]-C)

- **P2678 跳石头**

  题目描述

  这项比赛将在一条笔直的河道中进行，河道中分布着一些巨大岩石。组委会已经选择好了两块岩石作为比赛起点和终点。在起点和终点之间，有 N 块岩石（不含起点和终点的岩石）。在比赛过程中，选手们将从起点出发，每一步跳向相邻的岩石，直至到达终点。

  为了提高比赛难度，组委会计划移走一些岩石，使得选手们在比赛过程中的最短跳跃距离尽可能长。由于预算限制，组委会至多从起点和终点之间移走 M 块岩石（不能移走起点和终点的岩石）。

  - 二分答案其实就是逆向思维，在我们已知答案的法时候，对答案进行验证有时是相对容易的，所以可以先对答案进行假设，利用二分的方法去缩小答案区间。
  - 二分完距离我们需要验证是否满足至多取走M块石头，这里要用到贪心的思想。就是从左往右考虑，如果两个石头距离小于X，那一定是取走右边的那个石头更好。

3. 前缀和和差分

前缀和是常用的静态区间和的求解方法。

$$\sum_{i=l}^{r} a_i = s_r - s_{l-1}$$

$$s_1 = a_1, s_i = s_{i-1} + a_i$$

- **P1115 最大子段和**

  题目描述 展开

  给出一个长度为 n 的序列 a，选出其中连续且非空的一段使得这段和最大。

  - 在递推求前缀和的过程中，如果前缀和变成负数了，那么下一个数就做一个截断就好了。
  - f[i]表示以i为结尾的最大子段和
    f[i]=max(f[i-1], 0)+a[i]

- **P1719 最大加权矩形**

题目描述

展开

为了更好的备战NOIP2013，电脑组的几个女孩子LYQ,ZSC,ZHQ认为，我们不光需要机房，我们还需要运动，于是就决定找校长申请一块电脑组的课余运动场地，听说她们都是电脑组的高手，校长没有马上答应他们，而是先给她们出了一道数学题，并且告诉她们：你们能获得的运动场地的面积就是你们能找到的这个最大的数字。

校长先给他们一个N*N矩阵。要求矩阵中最大加权矩形，即矩阵的每一个元素都有一权值，权值定义在整数集上。从中找一矩形，矩形大小无限制，是其中包含的所有元素的和最大 。矩阵的每个元素属于[-127,127],例如

```
 0 -2 -7  0
 9  2 -6  2
-4  1 -4  1
-1  8  0 -2
```

在左下角：

```
9  2
-4  1
-1  8
```

和为15。

几个女孩子有点犯难了，于是就找到了电脑组精打细算的HZH，TZY小朋友帮忙计算，但是遗憾的是他们的答案都不一样，涉及土地的事情我们可不能含糊，你能帮忙计算出校长所给的矩形中加权和最大的矩形吗?

- 相当于刚才的问题变成二维的了，直接二维前缀和暴力求解即可。

差分可以用来快速进行区间加减，然后求某个点的值。

$$d_i = a_i - a_{i-1}$$

当给$a_l ... a_r$加一个数x时，不需要对每个数都进行操作，而是可以替代为对d_l加x，然后对d_{r+1}减x，然后通过对d进行前缀和求出每个a的值。其原理是当给一个区间加同一个数的时候，并没有代表这个区间内部所有数的差分，而是只改变了区间两个端点的差分值。

- **P1083 借教室**

题目描述

展开

在大学期间，经常需要租借教室。大到院系举办活动，小到学习小组自习讨论，都需要向学校申请借教室。教室的大小功能不同，借教室人的身份不同，借教室的手续也不一样。

面对海量租借教室的信息，我们自然希望编程解决这个问题。

我们需要处理接下来n天的借教室信息，其中第i天学校有r_i个教室可供租借。共有m份订单，每份订单用三个正整数描述，分别为d_j, s_j, t_j，表示某租借者需要从第s_j天到第t_j天租借教室（包括第s_j天和第t_j天），每天需要租借d_j个教室。

我们假定，租借者对教室的大小、地点没有要求。即对于每份订单，我们只需要每天提供d_j个教室，而它们具体是哪些教室，每天是否是相同的教室则不用考虑。

借教室的原则是先到先得，也就是说我们要按照订单的先后顺序依次为每份订单分配教室。如果在分配的过程中遇到一份订单无法完全满足，则需要停止教室的分配，通知当前申请人修改订单。这里的无法满足指从第s_j天到第t_j天中有至少一天剩余的教室数量不足d_j个。

现在我们需要知道，是否会有订单无法完全满足。如果有，需要通知哪一个申请人修改订单。

- 二分最小的无法满足的订单号k，然后用差分的方法对前k个订单进行快速操作，验证的时候求一下前缀和看有没有小于0的数即可。

4. 单调队列/单调栈

单调队列/栈就是按照单调顺序排列的一种数据结构，满足先进先出/后进先出的同时，维护单调性，一般用在动态规划的优化上。

- **P1886 滑动窗口**

题目描述

展开

有一个长为 n 的序列 a，以及一个大小为 k 的窗口。现在这个从左边开始向右滑动，每次滑动一个单位，求出每次滑动后窗口中的最大值和最小值。

例如：

The array is $[1, 3, -1, -3, 5, 3, 6, 7]$, and $k = 3$。

Window position	Minimum value	Maximum value
[1 3 -1] -3 5 3 6 7	-1	3
1 [3 -1 -3] 5 3 6 7	-3	3
1 3 [-1 -3 5] 3 6 7	-3	5
1 3 -1 [-3 5 3] 6 7	-3	5
1 3 -1 -3 [5 3 6] 7	3	6
1 3 -1 -3 5 [3 6 7]	3	7

- 如果$i < j$ 并且 $a[i] < a[j]$ 那么$a[i]$永远都不会成为最大值，所以只需维护一个单调递减队列，每次滑动后，将新的数字加入到队列中维护单调性，同时，需检查队列中队首元素是否还在窗口中，否则不断删除队首元素。

```
for (int i = 1; i <= n; i++)
{
    cin >> a[i];
    while (!q_up.empty() && a[i] < q_up.back().first)
        q_up.pop_back();
    q_up.emplace_back(a[i], i);
    while (i - q_up.front().second >= m)
        q_up.pop_front();
    if (i >= m)
        cout << q_up.front().first << ' ';
}
```

- **P1950 长方形**

题目描述

展开

小明今天突发奇想，想从一张用过的纸中剪出一个长方形。

为了简化问题，小明做出如下规定：

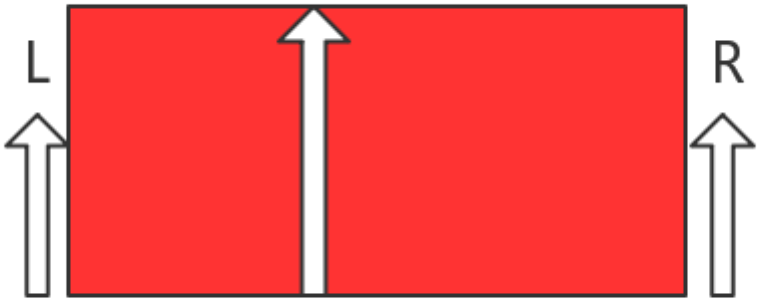
（1）这张纸的长宽分别为 n, m。小明讲这张纸看成是由$n \times m$个格子组成，在剪的时候，只能沿着格子的边缘剪。

（2）这张纸有些地方小明以前在上面画过，剪出来的长方形不能含有以前画过的地方。

（3）剪出来的长方形的大小没有限制。

小明看着这张纸，想了好多种剪的方法，可是到底有几种呢？小明数不过来，你能帮帮他吗？

- 给出一个n*m的01矩阵，求全1的矩阵有多少个
- 首先对每一行单独考虑，定义h_{ij}为第i行j列的格子，向上能扩展多少。
  易得，若$a_{ij} = 0$,$h_{ij} = 0$，否则$h_{ij} = h_{i-1j} + 1$
- 然后，可以将每一行的h_{ij}看做一排高度不等的木棍，现在求这些木棍能组成多少矩形。为了避免重复的计算，我们只考虑以每个木棍h_{ij}为高的矩形框，这个矩形框内，在这个木棍左右任选两个点，都可以组成一个符合条件的矩形。
- 那么怎么求每个矩形框呢，实质上就是求每个木棍可以向左右扩展多少格，也就是找到左边右边第一个比它矮的木棍(这里为了避免等高重复的问题，我们定义左边找第一个小于等于，右边找小于，矩形框为左开右闭)



- 快速求一个木棍左边第一个小于等于它的木棍，可以维护一个单调递增的单调栈来实现。

```
for (int i = 1; i <= n; i++)
{
    stack<pii> s;
    s.emplace(0, 0);
    for (int j = 1; j <= m; j++)
    {
        while (!s.empty() && h[i][j] < s.top().first)
            s.pop();
        l[j] = s.top().second;
        s.emplace(h[i][j], j);
    }
    stack<pii> t;
    t.emplace(-1, m + 1);
    for (int j = m; j >= 1; j--)
    {
        while (!t.empty() && h[i][j] <= t.top().first)
            t.pop();
        r[j] = t.top().second;
        t.emplace(h[i][j], j);
    }
    for (int j = 1; j <= m; j++)
        ans += (LL)(j - l[j]) * (r[j] - j) * h[i][j];
}
```

5. 分治

分治实际上是一种更复杂的二分过程。之前的二分通常是对答案x进行二分，而分治是对过程进行二分。分治的关键是想好把一个大过程划分成两半子过程后，是否可以化简求解过程。

- **P1908 逆序对**
  - 总体的逆序对数=左边的逆序对数+右边的逆序对数+跨区间的逆序对数

- 这里用到了归并排序的过程。排序的过程中，不断划分区间，直到划分到孤立的点。然后回溯，每一次回溯的过程我们都可以通过合并两个有序区间得到一个大的有序区间。跨区间的逆序对数可以在合并区间的时候顺便求出

```
void solve(int l, int r)
{
    if (l==r) return ;
    int mid=(l+r)>>1;
    solve(l, mid);
    solve(mid+1, r);
    int i=l, j=mid+1, k=0;
    while (i<=mid || j<=r)
    {
        if (j>r || i<=mid && q[i]<=q[j]) nq[k++]=q[i++],
        sum+=(j-mid-1);
        else
        if (i>mid || j<=r && q[i]>q[j]) nq[k++]=q[j++];
    }
    for (int i=0;i<k;i++) q[l+i]=nq[i];
}
```

6. 倍增

- **P1440求m区间内的最小值**

题目描述

展开

一个含有 n 项的数列，求出每一项前的 m 个数到它这个区间内的最小值。若前面的数不足 m 项则从第 1 个数开始，若前面没有数则输出 0。

```
void rmq()
{
    for (int i=1;i<=n;i++)
        st[i][0]=a[0];
    for (int k=1;(1<<k)<=n;k++)
        for (int i=1;i+(1<<k)-1<=n;i++)
            st[i][k]=min(st[i][k-1], st[i+(1<<k-1)][k-1]);

}
```