决策单调性 & 四边形不等式优化
- 区间dp
- 四边形不等式优化
- 决策单调性
- 板子题
  - P3515 [POI2011]Lightning Conductor

斜率优化
- 例题引入
- 二分/CDQ/平衡树优化 DP
- 总结
- 代码
- 一些题目
  - BZOJ2726任务安排
  - bzoj3675序列分割
  - bzoj4518征途
  - [bzoj2149]拆迁队
  - bzoj1492货币兑换Cash
  - bzoj3672购票

决策单调性 & 四边形不等式优化

区间dp

区间 dp 的特点在于，能将问题转化为 **两两合并** 的形式。一般通过 **枚举合并点** 的方式，来确定原问题的最优解。

$$f(i,j)=\max\{f(i,k)+f(k+1,j)+cost\}$$

其中 k 代表合并点。

四边形不等式优化

当函数 $f(i,j)$ 满足 $f(a,c)+f(b,d)\leq f(b,c)+f(a,d)$, 且 $a\leq b<c\leq d$ 时，称 $f(i,j)$ 满足四边形不等式。

当函数 $f(i,j)$ 满足 $f\left(i',j'\right)\leq f(i,j)$, 且 $i\leq i'<j'\leq j$ 时, 称 f 关于区间包含关系单调。

$s(i,j)=k$ 是指 $f(i,j)$ 这个状态的最优决策。

如果一个区间 DP 方程满足四边行不等式，那么求 k 值的时候 $s(i,j)$ 只和 $s(i+1,j)$ 和 $s(i,j-1)$ 有关, 所以可以以 $i-j$ 递增为顺序递推各个状态值最终求得结果, 将 $O\left(n^3\right)$ 降为 $O\left(n^2\right)$。

怎么证明 f 满足四边形不等式？$O(n^3+n^4)$ 打表验证（求 f 与枚举 a,b,c,d）。一般 opt = max 时不成立（但一般有其他性质，比如在两端点取最值）。最好都验证一下。

```
for (int k=i ; k<j ; k++ );//O(n^3)

for (int k=s[i][j-1] ; k<=s[i+1][j] ; k++);//O(n^2)
```

决策单调性

当有状态转移方程：

$$f_i=\min_{j\in[1,i)}\{f_j+cost_{j,i}\}$$

其中， cost(j,i) 满足 **四边形不等式**，则 f_i 具 有 **决策单调性。**

决策单调性：当状态 i 相对于 j < i 的所有 j 在状态 k (i < k) 都更优时，那么对于 k' (k < k') ，选 i 均比选 j 要更优。

一般来说，会以某段连续的区间 (l,r) 选 j 为当前最优方案来更新答案，即一个三元组 (j,l,r)。

具体的，

- 初始化决策集（用单调队列）加入 (0,1,n)，代表对于 [1,n] 区间的所有状态，选状态 0 是最优的。
- 顺序遍历每个状态 i，当前状态 i 从最优状态 j 转移而来（即队头元素的 j）。
- 若队头的区间 [l,r] ，r == i 时，将队头弹出，代表 **[l,r] 的所有状态更新完毕** 。将每次新队头的 l 更新成 i + 1 （由于 i 由上步更新完毕）。
- 插入状态 i。

- 对于插入来说，若当前状态 i 对于队尾的 (j,l,r)，对于 l 来说，选 i 比选 j 更优，那意味着整段 [l,r] 选 i 比选 j 更优，此时把队尾剔除。
- 若队列还有元素 (j,l,r)：
  - 对于 r 来说，选 i 比选 j 更优，那意味着 [l,r] 之间存在一个位置 p，在 [l,p-1] 时选 j 更优，在 [p,r] 时选 i 更优。二分出 p 的位置，将队尾的 (j,l,r) 修改为 (j,l,p-1)。那剩下的，直到 n 的状态，选 i 比选之前的 j 都更优。此时把状态 (i,p,n) 推进队尾。
  - 对于 r 来说，选 i 比选 j 劣，则判断 r 是否为 n。如果不为 n，则推入 (i,r+1,n)；否则不推入元素。
- 否则推入 (i,i+1,n)。

对于上述过程，对于每个状态 i，仅会被插入到队列中 1 次，同时也只会被删除 1 一次，故复杂度瓶颈在 $O(n)$ 次二分。

复杂度 $O(n \log n)$ 。

板子题

[NOI1995] 石子合并 - 四边形不等式

P3515 [POI2011]Lightning Conductor

「POI2011」Lightning Conductor - 决策单调性

斜率优化

例题引入

有 n 个玩具，第 i 个玩具价值为 c_i。要求将这 n 个玩具排成一排，分成若干段。对于一段 $[l,r]$，它的代价为 $(r-l+\sum_{i=L}^{R}c_i-L)^2$。求分段的最小代价。

$1\le n\le 5\times 10^4,1\le L,0\le c_i\le 10^7$

令 f_i 表示前 i 个物品，分若干段的最小代价。

状态转移方程：$f_i=\min_{j<i}\{f_j+(pre_i-pre_j+i-j-1-L)^2\}$。

其中 pre_i 表示前 i 个数的和，即 $\sum_{j=1}^{i}c_j$。

简化状态转移方程式：令 $s_i=pre_i+i,L'=L+1$，则 $f_i=\min_{j<i}\{f_j+(s_i-s_j-L')^2\}$。

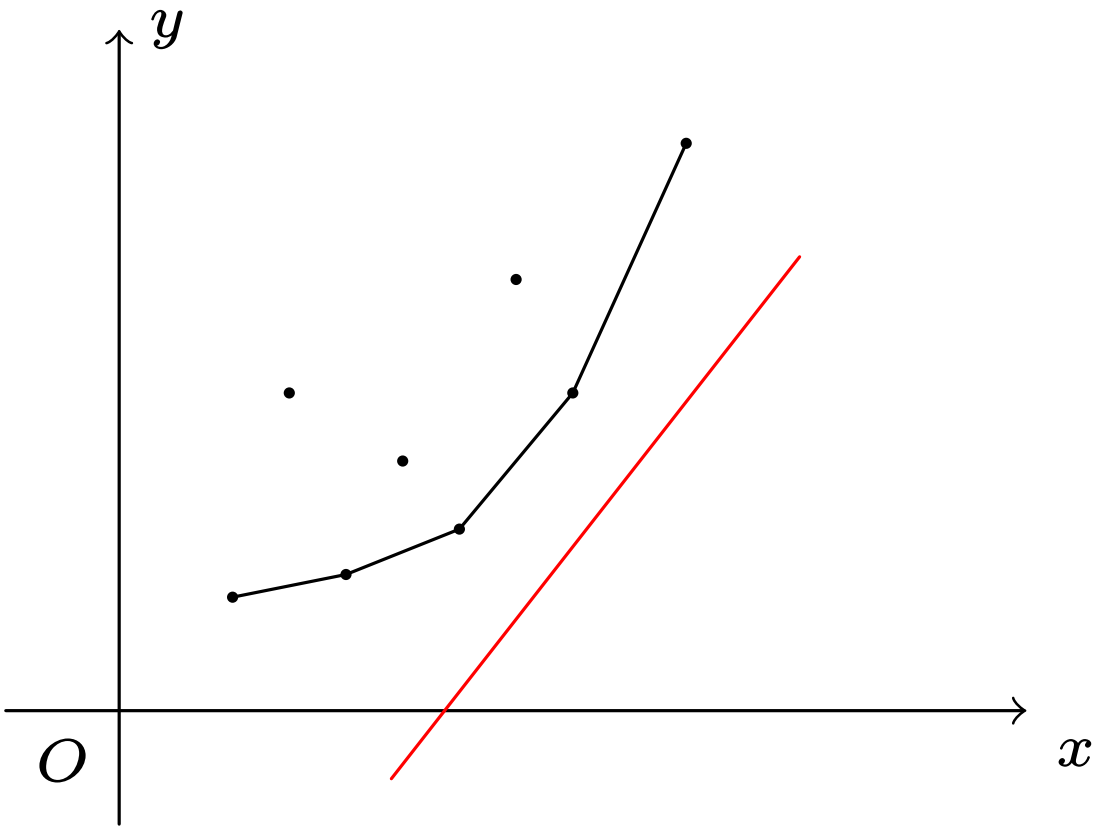
将与 j 无关的移到外面，我们得到

$$f_i-(s_i-L')^2=\min_{j<i}\{f_j+s_j^2+2s_j(L'-s_i)\}$$

考虑一次函数的斜截式 $y=kx+b$，将其移项得到 $b=y-kx$。我们将与 j 有关的信息表示为 y 的形式，把同时与 i,j 有关的信息表示为 kx，**把要最小化的信息（与 i 有关的信息）表示为 b**，也就是截距。具体地，设

$$\begin{aligned}x_j&=s_j\\y_j&=f_j+s_j^2\\k_i&=-2(L'-s_i)\\b_i&=f_i-(s_i-L')^2\end{aligned}$$

则转移方程就写作 $b_i=\min_{j<i}\{y_j-k_ix_j\}$。我们把 (x_j,y_j) 看作二维平面上的点，则 k_i 表示直线斜率，b_i 表示一条过 (x_j,y_j) 的斜率为 k_i 的直线的截距。问题转化为了，选择合适的 j（$1\le j<i$），最小化直线的截距。



如图，我们将这个斜率为 k_i 的直线从下往上平移，直到有一个点 (x_p, y_p) 在这条直线上，则有 $b_i = y_p - k_i x_p$，这时 b_i 取到最小值。算完 f_i，我们就把 (x_i, y_i) 这个点加入点集中，以做为新的 DP 决策。那么，我们该如何维护点集？

容易发现，可能让 b_i 取到最小值的点一定在下凸壳上。因此在寻找 p 的时候我们不需要枚举所有 $i-1$ 个点，只需要考虑凸包上的点。而在本题中 k_i 随 i 的增加而递增，因此我们可以单调队列维护凸包。

具体地，设 $K(a, b)$ 表示过 (x_a, y_a) 和 (x_b, y_b) 的直线的斜率。考虑队列 $q_l, q_{l+1}, \ldots, q_r$，维护的是下凸壳上的点。也就是说，对于 $l < i < r$，始终有 $K(q_{i-1}, q_i) < K(q_i, q_{i+1})$ 成立。

我们维护一个指针 e 来计算 b_i 最小值。我们需要找到一个 $K(q_{e-1}, q_e) \le k_i < K(q_e, q_{e+1})$ 的 e（特别地，当 $e = l$ 或者 $e = r$ 时要特别判断），这时就有 $p = q_e$，即 q_e 是 i 的最优决策点。由于 k_i 是单调递减的，因此 e 的移动次数是均摊 $O(1)$ 的。

在插入一个点 (x_i, y_i) 时，我们要判断是否 $K(q_{r-1}, q_r) < K(q_r, i)$，如果不等式不成立就将 q_r 弹出，直到等式满足。然后将 i 插入到 q 队尾。

这样我们就将 DP 的复杂度优化到了 $O(n)$。

概括一下上述斜率优化模板题的算法：

1. 将初始状态入队。
2. 每次使用一条和 i 相关的直线 $f(i)$ 去切维护的凸包，找到最优决策，更新 dp_i。
3. 加入状态 dp_i。如果一个状态（即凸包上的一个点）在 dp_i 加入后不再是凸包上的点，需要在 dp_i 加入前将其剔除。

接下来我们介绍斜率优化的进阶应用，将斜率优化与二分/分治/数据结构等结合，来维护性质不那么好（缺少一些单调性性质）的 DP 方程。

二分/CDQ/平衡树优化 DP

当我们在 i 这个点寻找最优决策时，会使用一个和 i 相关的直线 $f(i)$ 去切我们维护的凸包。切到的点即为最优决策。

在上述例题中，直线的斜率随 i 单调变化，但是对于有些问题，斜率并不是单调的。这时我们需要维护凸包上的每一个节点，然后每次用当前的直线去切这个凸包。这个过程可以使用二分解决，因为凸包上相邻两个点的斜率是有单调性的。

有 n 个玩具，第 i 个玩具价值为 c_i。要求将这 n 个玩具排成一排，分成若干段。对于一段 $[l,r]$，它的代价为 $(r-l+\sum_{i=L}^{R}c_i-L)^2$。求分段的最小代价。

$$1\le n\le 5\times 10^4, 1\le L, -10^7\le c_i\le 10^7$$

本题与「玩具装箱」问题唯一的区别是，玩具的价值可以为负。延续之前的思路，令 f_i 表示前 i 个物品，分若干段的最小代价。

状态转移方程：$f_i=\min_{j<i}\{f_j+(pre_i-pre_j+i-j-1-L)^2\}$。

其中 $pre_i=\sum_{j=1}^{i}c_j$。

将方程做相同的变换

$$f_i-(s_i-L')^2=\min_{j<i}\{f_j+s_j^2+2s_j(L'-s_i)\}$$

然而这时有两个条件不成立了：

1. 直线的斜率不再单调；
2. 每次加入的决策点的横坐标不再单调。

仍然考虑凸壳的维护。

在寻找最优决策点，也就是用直线切凸壳的时候，我们将单调队列找队首改为：凸壳上二分。我们二分出斜率最接近直线斜率的那条凸壳边，就可以找到最优决策。

在加入决策点，也就是凸壳上加一个点的时候，我们有两种方法维护：

1. 直接用平衡树维护凸壳。那么寻找决策点的二分操作就转化为在平衡树上二分，插入决策点就转化为在平衡树上插入一个结点，并删除若干个被踢出凸壳的点。此方法思路简洁但实现繁琐。
2. 考虑 CDQ 分治。

$\mathrm{CDQ}(l,r)$ 代表计算 $f_i, i\in[l,r]$。考虑 $\mathrm{CDQ}(1,n)$：

- 我们先调用 $\mathrm{CDQ}(1,mid)$ 算出 $f_i, i\in[1,mid]$。然后我们对 $[1,mid]$ 这个区间内的决策点建凸壳，然后使用这个凸壳去更新 $f_i, i\in[mid+1,n]$。这时我们决策点集是固定的，不像之前那样边计算 DP 值边加入决策点，那么我们就可以把 $i\in[mid+1,n]$ 的 f_i 先按照直线的斜率 k_i 排序，然后就可以使用单调队列来计算 DP 值了。当然，也可以在静态凸壳上二分计算 DP 值。
- 对于 $[mid+1,n]$ 中的每个点，如果它的最优决策的位置是在 $[1,mid]$ 这个区间，在这一步操作中他就会被更新成最优答案。当执行完这一步操作时，我们发现 $[1,mid]$ 中的所有点已经发挥了全部的作用，凸壳中他们存不存在已经不影响之后的答案更新。因此我们可以直接舍弃这个区间的决策点，并使用 $\mathrm{CDQ}(mid+1,n)$ 解决右区间剩下的问题。

时间复杂度 $n\log^2 n$。

对比「玩具装箱」和「玩家装箱 改」，可以总结出以下两点：

- 二分/CDQ/平衡树等能够优化 DP 方程的计算，于一定程度上降低复杂度，但不能改变这个方程本身。

- DP 方程的性质会取决于数据的特征，但 DP 方程本身取决于题目中的数学模型。

总结

斜率优化 DP 需要灵活运用，其宗旨是将最优化问题转化为二维平面上与凸包有关的截距最值问题。遇到性质不太好的方程，有时需要辅以数据结构来加以解决，届时还请就题而论。

维护下凸包当且仅当 opt = min。

快半个圆的凸包经常存在。正确求法应用斜率的单调性。

关于 X(i) 的设值：让其都单增（随着决策点的顺序添加单增），然后直接套 slope。故当斜率时，K(i) 取什么值，是否单调都无所谓了，反正求得是截距最小(最大)，而不管正负，都是切点最小（自己可以比划一下）。

cdq分治会把斜率给整成随机的，若套了cdq，则只能在凸包上二分了。

怕爆 long long，用 long double 的斜率式。加个eps，呈直线的也踢掉，不会是最优的。

代码

```
#include <bits/stdc++.h>
using namespace std;

using i64 = long long;
using f64 = long double;
constexpr i64 inf = 0x3f3f3f3f3f3f3f3f;
constexpr f64 eps = 1e-9;  ///与delta X的取值直接相关

vector<i64> f, A, B, g;  //f是最终取值

struct SlopeDP {
    vector<int> stk;
    SlopeDP() {}
    i64 getans(int i, int j) {
        return f[j] - A[j] - A[j] * j + (i64)j * (j + 1) / 2 + A[j] * i -
               (i64)i * j + B[i] + A[i] + (i64)i * (i - 1) / 2;  //f[i]关于j的表达式
    }
    i64 Y(int j) { return f[j] - A[j] - A[j] * j + (i64)j * (j + 1) / 2; }
    i64 X(int j) { return A[j] - j; }
    i64 K(int i) { return -i; }  //-i表示y=kx+b的k
    f64 slope(int i, int j) { return (f64)(Y(j) - Y(i)) / (X(j) - X(i)); };
    void add(int j) {  //这里维护下凸包，opt=min。opt=max时，调转比较符
        while (stk.size() > 1) {
            int q = *prev(prev(stk.end())), p = *prev(stk.end());
            if (slope(q, p) + eps > slope(p, j)) {
                stk.pop_back();
            } else {
                break;
            }
        }
        stk.emplace_back(j);
    }
    i64 query(int i) {
```

```
        if (stk.size() == 0) {
            return inf;
        }
        int id = 0;
        int l = 0, r = (int)stk.size() - 1;
        while (l < r) {
            int m = l + r >> 1;
            if (slope(stk[m], stk[m + 1]) < K(i) + eps) {
                id = m + 1;
                l = m + 1;
            } else {
                r = m;
            }
        }
        return getans(i, stk[id]);
    }
};

struct Cdq {
    vector<int> id;
    Cdq(vector<int> id_)
        : id(id_) {
        sort(id.begin(), id.end());  //这里默认第一关键字是下标，即保证cdq之前i<j,id[i]
<id[j]
    }
    function<void(int, int)> cdq = [&](int l, int r) {
        if (l < r) {
            int m = l + r >> 1;
            cdq(l, m);
            SlopeDP sdp;
            vector<int> id1, id2;
            for (int i = l; i <= m; i++) {
                id1.emplace_back(id[i]);  //这里默认左半区间的点都可以贡献给右半区间，具体
需要看题目的dp结构
            }
            for (int i = m + 1; i <= r; i++) {
                id2.emplace_back(id[i]);
            }
            function<bool(int, int)> cmp = [&](int x, int y) {  //这里按照第二关键字，
也就是y=kx+b的x排序
                return sdp.X(x) <= sdp.X(y);
            };
            sort(id1.begin(), id1.end(), cmp);
            sort(id2.begin(), id2.end(), cmp);
            /*下列所有注释是调试，根据需要解注释看转移过程*/
            // cout << l << " " << r << ",id1:";
            // for (auto x : id1) {
            //     cout << x << " ";
            // }
            // cout << ",id2:";
            // for (auto x : id2) {
            //     cout << x << " ";
            // }
```

```
            // cout << ".";
            // cout << l << " " << r << "\n";
            int p = 0, q = 0;
            while (q < id2.size()) {
                while (p < id1.size() &&
                       sdp.x(id[p]) <= sdp.x(id[q])) {
                    sdp.add(id1[p]);
                    p++;
                }

                int i = id2[q];
                i64 buf;  //调试用
                // cout << "[p=" << p << "],";
                f[i] = min(f[i], buf = sdp.query(i));
                // cout << "f[" << i << "]:" << buf << ",";
                q++;
                // cout << ")";
            }
            // cout << "\n";
            // cout << l << " " << r << "\n";
            cdq(m + 1, r);
        }
    };
};
```

一些题目

[BZOJ2726][SDOI2012]任务安排

$$f[i]=min\{f[j]+(sf[i]-sf[j])*st[i]+S*(sf[n]-sf[j])\}$$

即

$$f[i]=min\{f[j]+st[i]*sf[i]-st[i]*sf[j]+S*sf[n]-S*sf[j]\}$$

> $Y(j)=f[j]-S*sf[j],X(j)=sf[j],K(i)=st[i]$。斜率不一定单增，$st[i+1]$可能小于$st[i]$。而$X(j)$妥妥单增，故在下凸包上二分答案：由getans来比较，slope来维护下凸包，解决。

[bzoj3675][Apio2014]序列分割

$f[i][k]=max\{f[j][k-1]+(s[i]-s[j])*(s[n]-s[i])\}$
即
$f[i][k]=max\{f[j][k-1]+s[j]*s[i]-s[j]*s[n]+s[i]*s[n]-s[i]*s[i]\}$

> $Y(j,k)=f[j][k]-s[j]*s[n],X(j,k)=s[j],K(i)=-s[i]$。斜率单减，X(j)单增，单调队列维护上凸包。空间小，故X，Y由表达式来求，且使用滚动dp。

[bzoj4518][Sdoi2016]征途

$f[i][k]=min\{f[j][k-1]+m*(s[i]-s[j])*(s[i]-s[j])\}$(最后加上最后一段$m*(s[n]-s[i])^2$，减去$s[n]*s[n]$)
即
$f[i][k]=min\{f[j][k-1]+m*s[i]*s[i]+m*s[j]*s[j]-2*s[i]*s[j]\}$

> $Y(j,k)=f[j][k]+m*s[j]*s[j],X(j)=s[j],K(i)=2*s[i]$。斜率单增，$X(j)$单增，单调队列维护下凸包。记得将最后一段加上。

[bzoj2149]拆迁队

几个问题：
1、ij项为：$-d[j]*i$。
试令$X[j]=d[j]$，维护$X[j]$单增的下凸包。
2、$d[L[cl]]<=d[R[cr]]$加点
看清楚，d[]作为X轴，（直觉上）决策点不能在状态点右侧。
3、cmp。
让不这么优的放前面。
4、爆$longlong$
有可能。在斜率那。改换成long double
$g[i]=g[j]+(2*a[j]+i-j)*(i-j-1)/2+a[i]+b[i]$

强约束：$f[j]+1=f[i]$，$d[j]<=d[i](f[j]+1=f[i]$不代表j就能转移给i，可能j的$a[j]$值很高，但排的前而已。）
负数的问题也解决了，原先的solve里当栈空的时候也会更新值。。。结果出现了负数。栈空时不更新。同时栈顶为$stk[top-1]$。。不要搞错了。

> $Y(j)=g[j]-(j+1)*a[j]+j*(j+1)/2,X(j)=d[j],K(i)=-i$，cdq按X(j)单增下凸包维护。三分靠getans评判。

[bzoj1492][NOI2007]货币兑换Cash

为什么不卖掉一部分金券来增加现金，或者花掉一部分现金换取金券：

1. 首先倘若分天卖金券，必定存在一天卖，以同等的比例，能卖更高的价格。即$A_i*tot+B_i*tot$最大。同时分天买金券，必定有一天花相同价钱买入金券的量最大的。
2. 由于一天可以多次交易，而且交易的价格都是按照 A_i,B_i 来计算的，即现金与期货等价交换。第i天能赚到的最多钱等价于第 i 天能换到最多的金券数量。故直接用f[i]表示金额的大小即可。

总结：可以这么理解，f[i]表示当前能赚的最多的钱，可以把第j天的所有钱换成金券（那么中间就不操作），然后在第i天全部卖掉。

$f[i]=X[j]*A[i]+Y[j]*B[i].$

> $Y(j)=\frac{f[j]}{A[j]*rate[j]+B[j]},X(j)=\frac{rate[j]*f[j]}{A[j]*rate[j]+B[j]},opt=max$.cdq维护上凸包。

[bzoj3672][Noi2014]购票

树结构+斜率优化的题目。

假定每个点的可走距离lim[i]为无穷大，则从该点沿祖先的所有点的决策均可以做，复杂度$O(n^2)$。考虑到**以u节点为根节点的所有子树，均可以从u->fa[u]->...->1的一条链状态转移过来**，便考虑点分治。
先继续对u节点的祖先连通块进行分治，即让祖先的所有状态求出。后把u节点的子树加入决策中，按dis[u] - lim[u]从大到小排序（该值越大，可选择的决策点就越少）。遍历u上的祖先链，维护决策下凸包，让子树的状态更新完后，再对子树进行分治。推荐将根节点也划入待更新的状态中，代码写的简洁。