动态规划

清华大学 张华清

斜率优化

- 什么样的题可以用斜率优化？其实特征也比较明显，我们也从一道经典例题来看：

Luogu 3195 [HNOI2008]玩具装箱

题目描述

复制Markdown 展开

P 教授要去看奥运，但是他舍不下他的玩具，于是他决定把所有的玩具运到北京。他使用自己的压缩器进行压缩，其可以将任意物品变成一堆，再放到一种特殊的一维容器中。

P 教授有编号为 $1 \cdots n$ 的 n 件玩具，第 i 件玩具经过压缩后的一维长度为 C_i。

为了方便整理，P教授要求：

- 在一个一维容器中的玩具编号是连续的。
- 同时如果一个一维容器中有多个玩具，那么两件玩具之间要加入一个单位长度的填充物。形式地说，如果将第 i 件玩具到第 j 个玩具放到一个容器中，那么容器的长度将为 $x = j - i + \sum_{k=i}^{j} C_k$。

制作容器的费用与容器的长度有关，根据教授研究，如果容器长度为 x，其制作费用为 $(x - L)^2$。其中 L 是一个常量。P 教授不关心容器的数目，他可以制作出任意长度的容器，甚至超过 L。但他希望所有容器的总费用最小。

$dp[i] = \min dp[k-1] + (i-k+ \text{sum}[i] - \text{sum}[k-1] - L)^2$

$k \quad i \quad [k, i]$ 在一起

$O(n^2)$

$j+1 \quad i$

$i-(j+1)$

$n \le 1000000$

Luogu 3195 [HNOI2008]玩具装箱

- 我们很容易就可以写出一个DP方程：
- f[i]=min{f[j]+(i-j-1+sum[i]-sum[j]-L)^2}。其中sum是前缀和
- 我们稍微整理一下呗，很常见的思路就是把和i有关的放一起，和j有关的放一起
- f[i]=min{f[j]+((i+sum[i])-(j+sum[j]+L+1))^2 }
- 都写到这了，我们就不妨让A[i]=i+sum[i]，让B[j]=j+sum[j]+L+1
- 那么就是f[i]=min{f[j]+(A[i]-B[j])^2}
- 再拆一拆：f[i]=min{ f[j]+A[i]^2+B[j]^2-2A[i]B[j] }

Luogu 3195 [HNOI2008]玩具装箱

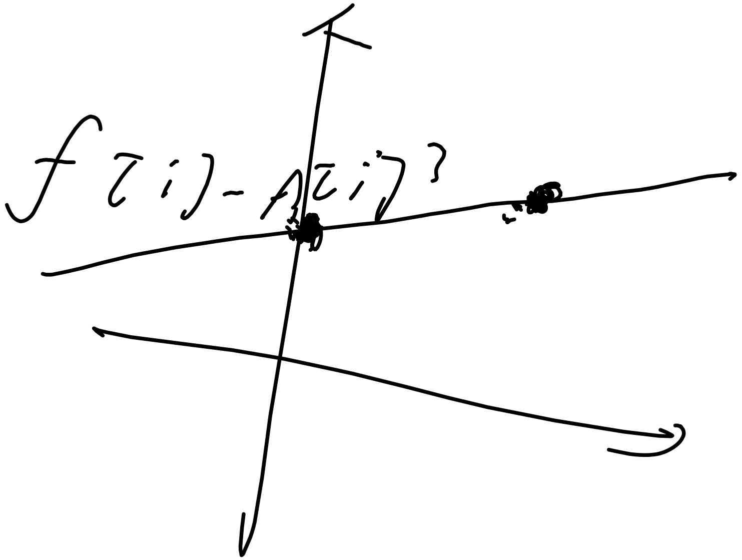
- $f[i]=\min\{ f[j]+B[j]^2+A[i]^2-2A[i]B[j] \}$
- 假设我们就从j转移，那先把min去掉，得到它满足：
- $f[i]-A[i]^2= f[j]+B[j]^2-2A[i]B[j]$ 这样一个方程。
- 形式差不多出来了。
- 注意此时的问题是：我们有好多个j，对应有好多个B[j],f[j]，我们每选一个j，带到方程中，就会求出一个f[i]。那到底选哪一个j，求出的f[i]是最小的呢？

Luogu 3195 [HNOI2008]玩具装箱

- $f[i]-A[i]^2 \ = \ f[j]+B[j]^2 \ -2A[i]B[j]$
- 如果大家学过高中数学，那这是非常经典的一个问题。
- 如果没有学过，大家就可以学习一下这个套路：
- 我们看这个方程的每一项：
- f[i]-A[i]^2，只和i有关，是想要求最小值的（A[i]是定值，求f[i]最小值就是求f[i]-A[i]^2的最小值）
- f[j]+B[j]^2，只和j有关，是已经知道的
- 2A[i]B[j]，是（和i有关的）*（和j有关的），都是已经知道的
- 套路就是，把只和j有关的，已经知道的f[j]+B[j]^2，看做y_j
- 把（和i有关的）*（和j有关的）中，和j有关的，已经知道的B[j]，看作x_j
- 那么每个j，都对应一个点(x_j,y_j)

Luogu 3195 [HNOI2008]玩具装箱

- $f[i]-A[i]^2 = f[j]+B[j]^2 - 2A[i]B[j]$，换一下：
- $f[i]-A[i]^2 = y_j - 2A[i]x_j$
- 移个项，把y_j放一边：
- $y_j = 2A[i]x_j + f[i] - A[i]^2$
- 我们再给2A[i]起个名字吧！叫做k_i！
- 就得到$y_j = k_i x_j + f[i] - A[i]^2$
- 这时，我们发现f[i]-A[i]^2恰巧等于一个值！
- 它恰巧等于，过点(xj,yj)的，斜率为ki的直线，的纵截距！
- 就是说你考虑y=ki*x+b这个直线，它过(xj,yj)嘛，带进去：yj=ki*xj+b，那你对照一下，b不就是f[i]-A[i]^2嘛



Luogu 3195 [HNOI2008]玩具装箱

- 大家注意：点(xj,yj)，是此时已经算出了的一坨点，只要算出了，就是恒定不变的
- 而ki,是对于一个特定的i，给出的。
- 现在的问题是，我们有一坨不动的点，还有一个给出的斜率ki，我们要选一个点，让过这个点且斜率为ki的直线尽量小。
- 看我爬爬黑板！
- 发现了嘛？我们可以让这个直线从下往上切这个点集。第一次遇到哪个点，那就是说选这个点，纵截距最小
- ——也即！从这个点转移，f[i]最小！

斜率优化

- 好那我们现在可以梳理一下：
- 什么样的转移方程可以用斜率优化维护？
- 有一些项只和i有关，是我们想要最小/大化的
- 有一些项只和j有关，是已知的
- 有一个项，是（和i有关）*（和j有关）
- 符合这个条件的，就可以用斜率优化。

斜率优化

- 具体过程是：把只和j有关的看作纵坐标yj，（和i有关）*（和j有关）中，“和j有关”的看作横坐标xj，和i有关的看作斜率ki
- 这时我们恰巧发现，“只和i有关”的那些项，恰好等于直线的纵截距。我们要求纵截距的最小值，只需要用斜率为ki的直线去切点集，看看先切到谁就行。

斜率优化

- 怎么实现这个“切点集，看看先切到谁呢”？
- 计算几何经典结论：只可能会切到凸包上的点。
- 啥是凸包？考虑把一个弹性绳，包住所有点，就会形成凸包。

斜率优化

- 怎样维护凸包，并且找出用斜率为ki的直线去切，会切到谁?
- 有不同的情况:
- Case1：xj是单调递增的(也就是说我们只会往当前凸包的右边加点)，同时ki也是单调递增的，这时可以用单调队列来维护。
- “单调”的是啥？是斜率。 (具体一会再说
- Case2：xj是单调递增的，但ki不是。这时我们可以用单调栈维护凸包，但查询时需要二分
- Case3：xj都不是单调的，那需要CDQ分治/平衡树/二进制分组维护凸包
- 最常见的还是case1。

斜率优化

- Case1:
- （实际上只需要半个凸包。在本题中即下凸壳
- 单调队列里存的是下标。
- 要求斜率是单增的。
- 用一个斜率为k的直线去切?
- 正好切在凸包上满足这个条件的一个点：它左边那条边斜率小于k，右边那条边斜率大于k

Luogu 3195 [HNOI2008]玩具装箱

```
double a(int i){return sum[i]+i;}
double b(int i){return a(i)+w;}
double X(int i) {return a(i);}
double Y(int i){return 1ll*f[i]+b(i)*b(i); }
double slp(int a1,int a2){return 1.0*(Y(a2)-Y(a1))/(X(a2)-X(a1));}

int main(){
    scanf("%d%d",&n,&l);w=l+1;
    for(int i=1;i<=n;i++){
        scanf("%lld",&sum[i]);
        sum[i]+=sum[i-1];
    }
    int head=0,tail=-1;
    que[++tail]=0;//加入0这个点
    for(int i=1;i<=n;i++){
        while(head+1<=tail&&slp(que[head],que[head+1])<2*a(i))head++;//至少两个
        int j=que[head];
        f[i]=f[j]+1ll*(a(i)-b(j))*(a(i)-b(j));
        while(head+1<=tail&&slp(que[tail-1],que[tail])>slp(que[tail],i))tail--;
        que[++tail]=i;
    }
    printf("%lld\n",f[n]);
    return 0;
}
```

x(j) f(j)

f[i] k_i

(x_i, y_i)

5504 [JSOI2011] 柠檬

题目描述

复制Markdown 展开

Flute 很喜欢柠檬。它准备了一串用树枝串起来的贝壳，打算用一种魔法把贝壳变成柠檬。贝壳一共有 n $(1 \le n \le 100000)$ 只，按顺序串在树枝上。为了方便，我们从左到右给贝壳编号 $1..n$ 。每只贝壳的大小不一定相同，贝壳 i 的大小为 $s_i(1 \le s_i \le 10000)$ 。

变柠檬的魔法要求：Flute 每次从树枝一端取下一小段连续的贝壳，并选择一种贝壳的大小 s_0。如果这一小段贝壳中大小为 s_0 的贝壳有 t 只，那么魔法可以把这一小段贝壳变成 s_0t^2 只柠檬。Flute 可以取任意多次贝壳，直到树枝上的贝壳被全部取完。各个小段中，Flute 选择的贝壳大小 s_0 可以不同。而最终 Flute 得到的柠檬数，就是所有小段柠檬数的总和。

Flute 想知道，它最多能用这一串贝壳 变出多少柠檬。请你帮忙解决这个问题。

5504 [JSOI2011] 柠檬

- 发现性质：每一段的左右端点的贝壳大小一定是相等的，且这一段选定的s_0一定是左右端点的贝壳大小。
- 否则可以额外分出一段，答案会变得更优。
- 好处：去掉了“众数”这个难以处理的东西

5504 [JSOI2011] 柠檬

c_i：1---i 中有多少个贝壳 大小和i相同

- 于是我们有朴素DP：
- 状态设计：设f_i为前i个位置分成若干段的最大收益。
- 转移：$f_i = \max\limits_{j\le i, s_j = s_i} f_{j-1} + s_i * (c_i - c_j + 1)^2$（其中$s_i = s_j$可视为常数。
- 展成斜率优化的形式：$f_{j-1} + s_j c_j^2 = (2s_i(c_i - 1))c_j + f_i - s_i(c_i + 1)^2$
- 把$(c_j, f_{j-1} + s_j c_j^2)$看成点，$2s_i(c_i - 1)$看成斜率。

5504 [JSOI2011] 柠檬

- 需要对于每个大小分别维护。
- 使用单调队列维护…吗?
- 我们细细分析:
  - 求最大值，维护上凸壳
  - 横坐标单调递增
  - 斜率单调递增
- 凸壳往右长，斜率往左跑
- ——使用单调栈维护

决策单调性

$f_i = \min_{j} f_j + \cdots$

- 决策单调性：
- 一个解决最优性问题的DP题，如果设i的最优转移点是j，i’的最优转移点是j’，当i<i’时，有j<=j’，则称该DP问题具有决策单调性。
  - 值得注意的是，随转移点j的增大，f[i]并不是先变大后变小。也就是说，并不是单峰的
- 如何发现一道题目具有决策单调性？
- 打表！
- 下面，我们将通过例题，分别介绍三种决策单调性的类型。

决策单调性

- 分层决策单调性。
- 当DP问题形如：
- $f[i][j] = \min_{k<j} f[i-1][k] + cost(k,j)$
- 设k为f[i][j]的最优转移点，k'为f[i][j']的最优转移点，当j<j'时有k<=k'，则该DP具有决策单调性。

决策单调性

- 若直接DP，复杂度$O(nm^2)$。
- 我们使用分治的手段优化该DP。
- 还是一层一层的DP。对于第i层，我们先算$f[i][mid]$，其中$mid = \frac{m}{2}$。我们顺便求出f[i][mid]的最优转移点f[i-1][opt]。那么[1,i-1]的最优转移点只能在f[i-1][1...opt]中取，f[i-1][i+1...m]的最优转移点只能在[opt+1,m]中取。
- 递归下去，我们用solve(i,l,r,p,q)表示，下面要算f[i][l...r]，已知最优转移点只可能在f[i-1][p...q]中取。
- 先算f[i][mid]的值（枚举p到q），求出最优转移点opt，递归到solve(i,l,mid-1,p,opt)和solve(i,mid+1,r,opt,q)。
- 复杂度$O(nm\log m)$。

CF321E Ciel and Gondolas

Ciel狐狸在游乐园里排队等待上摩天轮。现在有n 个人在队列里，有k 条船，第i条船到时，前q_i 个人可以上船。最后一条船将载走剩下的所有人，则q_k 此时载走的人数。 人总是不愿意和陌生人上同一条船的，当第i 个人与第j 个人处于同一条船上时，会产生$u_{i,j}$ 的沮丧值。请你求出最小的沮丧值和。 一条船上的人两两都会产生沮丧值。

输入格式:

第一行两个数代表n ,k ， 接下来n 行每行n 个数，第i 行第j 个数表示$u_{i,j}$ 。 请使用快速读入的技巧，防止读入导致的TLE。

输出格式:

一行一个整数表示最小沮丧值。 贡献者：MSF_Akatsuki

CF321E Ciel and Gondolas

- 记f[x][i]为前x艘船，上了i个人。
- $f[x][i] = \min_j f[x-1][j] + sum(j+1, i)$，其中sum(x,y)是(x,x)到(y,y)的子矩阵和（再除2）
- 然后直接看代码：

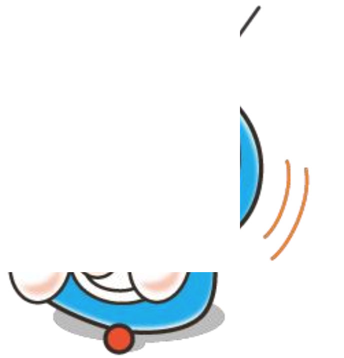
CF321E Ciel and Gondolas

- 一共logn层。每一层总复杂度O(n)
- 复杂度nklogn

f[i][mid]

```
void solve(int i,int l,int r,int p,int q){
    if(l>r)return;
    int mid=(l+r)>>1;
    int opt;
    for(int h=p;h<=q;h++){
        if(f[i][mid]>f[i-1][h]+get_sum(h+1,mid)){
            f[i][mid]=f[i-1][h]+get_sum(h+1,mid);
            opt=h;
        }
    }
    solve(i,l,mid-1,p,opt);
    solve(i,mid+1,r,opt,q);
}

int main(){
    scanf("%d%d",&n,&m);
    for(int i=1;i<=n;i++){
    for(int i=1;i<=n;i++){
    memset(f,0x3f,sizeof(f));
    f[0][0]=0;
    for(int i=1;i<=m;i++){
        solve(i,0,n,0,n);
    }
    printf("%d\n",f[m][n]);
    return 0;
}
```



决策单调性

- 单层决策单调性。
- 一维DP，形如
- $f_r = \min\limits_{l=1}^{r-1}\{f_l + cost(l,r)\}$ 。
- 设k为f[i]的最优转移点，k’为f[i’]的最优转移点，当i<i’时有k<=k’，则该DP具有决策单调性。

决策单调性

- 有一个性质：只考虑前若干个转移点，也有决策单调性：
- 用i位置的数字，表示i的当前最优转移。那会是：
- 只考虑j<=1：1111111111
- 只考虑j<=2：112222222
- 只考虑j<=3：1122223333
- 只考虑j<=4：2444444
- 只考虑j<=5：444555

决策单调性

- 算法大概就是：单调队列里放一个三元组：表示当前,[l,r]区间由j这个转移点转移过来最优
- 一开始：取出队首，就知道i从哪转移了。
- 然后算出f[i]，现在加入i这个转移点。
- 开始判断，能不能把队尾整个弹出。如果能，就弹
- 直到不再能整个弹出，就得看看从哪断开。一个一个判？T飞了。
- 只需要搞一个二分即可。

1912 [NOI2009] 诗人小G

题目描述

复制Markdown 展开

小 G 是一个出色的诗人，经常作诗自娱自乐。但是，他一直被一件事情所困扰，那就是诗的排版问题。

一首诗包含了若干个句子，对于一些连续的短句，可以将它们用空格隔开并放在一行中，注意一行中可以放的句子数目是没有限制的。小 G 给每首诗定义了一个行标准长度（行的长度为一行中符号的总个数），他希望排版后每行的长度都和行标准长度相差不远。显然排版时，不应改变原有的句子顺序，并且小 G 不允许把一个句子分在两行或者更多的行内。在满足上面两个条件的情况下，小 G 对于排版中的每行定义了一个不协调度, 为这行的实际长度与行标准长度差值绝对值的 P 次方，而一个排版的不协调度为所有行不协调度的总和。

小 G 最近又作了几首诗，现在请你对这首诗进行排版，使得排版后的诗尽量协调（即不协调度尽量小），并把排版的结果告诉他。

1912 [NC

```cpp
bool jdg(int i,int j1,int j2){//i从j2转移是不是比从j1转移好
    return f[j1]+get_pow(abs(sum[i]-sum[j1]+(i-j1-1)-L),p)>\
    f[j2]+get_pow(abs(sum[i]-sum[j2]+(i-j2-1)-L),p);
}
int bnd(Node hh,int j2){
    int l=hh.l,r=hh.r,j1=hh.x;
    if(jdg(r,j1,j2)==false)return r+1;//如果最右边的点都不能更新最优决策点
    while(l+1<r){
        int mid=(l+r)>>1;
        if(jdg(mid,j1,j2))r=mid;
        else l=mid+1;}
    }
    if(jdg(l,j1,j2))return l;
    return r;
}
void DP(){
    head=0,tail=-1;
    for(int i=1;i<=n;i++)f[i]=INF;
    f[0]=0;
    que[++tail]=Node(1,n,0);
    for(int i=1;i<=n;i++){
        if(que[head].r<i)head++;

        int j=que[head].x;
        pre[i]=j;//记录i从j转移过来
        f[i]=f[j]+get_pow(abs(sum[i]-sum[j]+(i-j-1)-L),p);//别忘了还有空格

        int w=n+1;
        while(head<=tail&&jdg(que[tail].l,que[tail].x,i)){//判断能不能弹队尾
            w=que[tail].l;
            tail--;
        }
        if(head<=tail)w=bnd(que[tail],i);
        if(w<=n)if(head<=tail&&w<=que[tail].r){
            que[tail]=Node(que[tail].l,w-1,que[tail].x);
        }
        que[++tail]=Node(w,n,i);
    }
}

void Print(){
```

决策单调性

- 区间DP决策单调性。
- 这类决策单调性有点特殊：对于最优化的区间DP，若有$d[l][r-1] \le d[l][r] \le d[l+1][r]$，其中 d[l][r]是f[l][r]的最优转移点。那我们就这样处理：
- 先按长度枚举区间。
- 然后计算f[l][r]的时候就直接从d[l][r-1]枚举到d[l+1][r]。
- 复杂度即为$O(n^2)$。
  - 为什么？斜着看！。

(1,3)

d(1,2) − d(2,3)

d(2,3) − d(3,4)

d(3,4) − d(4,4)

(1,2) (2,3) (3,4) (4,4)

O(n)

HDU3516 Tree Construction

Problem Description

Consider a two-dimensional space with a set of points (xi, yi) that satisfy xi < xj and yi > yj for all i < j. We want to have them all connected by a directed tree whose edges go toward either right (x positive) or upward (y positive). The figure below shows an example tree.

Figure 1: An example tree

Write a program that finds a tree connecting all given points with the shortest total length of edges.

Input

The input begins with a line that contains an integer n (1 <= n <= 1000), the number of points. Then n lines follow. The i-th line contains two integers xi and yi (0 <= xi, yi <= 10000), which give the coordinates of the i-th point.

HDU3516 Tree Construction

- 先想DP怎么写。
- 区间DP。
- $dp[i][j] = \min_k dp[i][k] + dp[k+1][j] + abs(y[j] - y[k]) + abs(x[i] - x[k+1])$
- 套上前面说的方法即可。

HDU3516 Tree Construction

- 先想DP怎么写。
- 区间DP。
- $dp[i][j] = \min_{k} dp[i][k] + dp[k+1][j] + abs(y[j] - y[k]) + abs(x[i] - x[k+1])$
- 套上前面说的方法即可。

HDU3516 Tree Construction

```
for(len=3;len<=n;len++){
    for(i=1;i+len-1<=n;i++){
        j=i+len-1;
        dp[i][j]=inf;

        for(k=s[i][j-1];k<=s[i+1][j];k++){
            if(dp[i][j]>dp[i][k]+dp[k+1][j]+abs(y[j]-y[k])+abs(x[i]-x[k+1]) ){
                dp[i][j]=dp[i][k]+dp[k+1][j]+abs(y[j]-y[k])+abs(x[i]-x[k+1]);
                s[i][j]=k;
            }
        }
    }
}
```

带权二分

- 其实并不一定用于优化DP，也可能用于优化贪心等最优化的算法。
- 什么样的问题可以尝试使用带权二分优化？特征很明显。我们先来看两个经典题的题干：

2619 [国家集训队]Tree I4072

- 给你一个无向带权连通图，每条边是黑色或白色。让你求一棵最小权的恰好有 k 条白色边的生成树。

[SDOI2016]征途

- 简要题意：给一个序列，要求分成恰好k段，使得每段和方差最小。
- n<=3000

带权二分

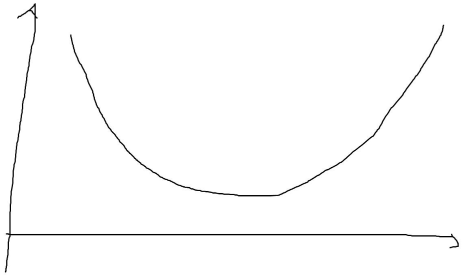
- 大家发现特征了嘛?
- 要解决一个最优化问题（求最大/最小值）
- 同时有一个限制，就是某个东西一定要恰好是k。
- 比如说，Tree，求最小生成树，同时要求白边恰好k条
- 征途，求方差的最小值，同时要求分成恰好k段

带权二分

- 大家可以想想，我们DP时，如果有“恰好k个”的限制，那我们往往要在dp数组里多加一个状态，表示当前已经选了k个，这样时间复杂度就上来了
- 比如征途，设dp[i][j]为1...i，已经选了j段；转移枚举最后一段的左端点k，复杂度n^3。
- 至于Tree这个题（求最小生成树类似贪心吧），可能连“多记一个状态”都拯救不了你

带权二分

- 这时，我们就可以尝试用带权二分来解决“恰好k个”的限制。具体来说：
- 如果最小(大)化的值，随着k，是一个凸函数，那么就可以用带权二分
- 以Tree举例子，大概是这样：
- 横轴是白边个数，纵轴是白边个数为横坐标的情况下，
- 最小生成树的值



带权二分

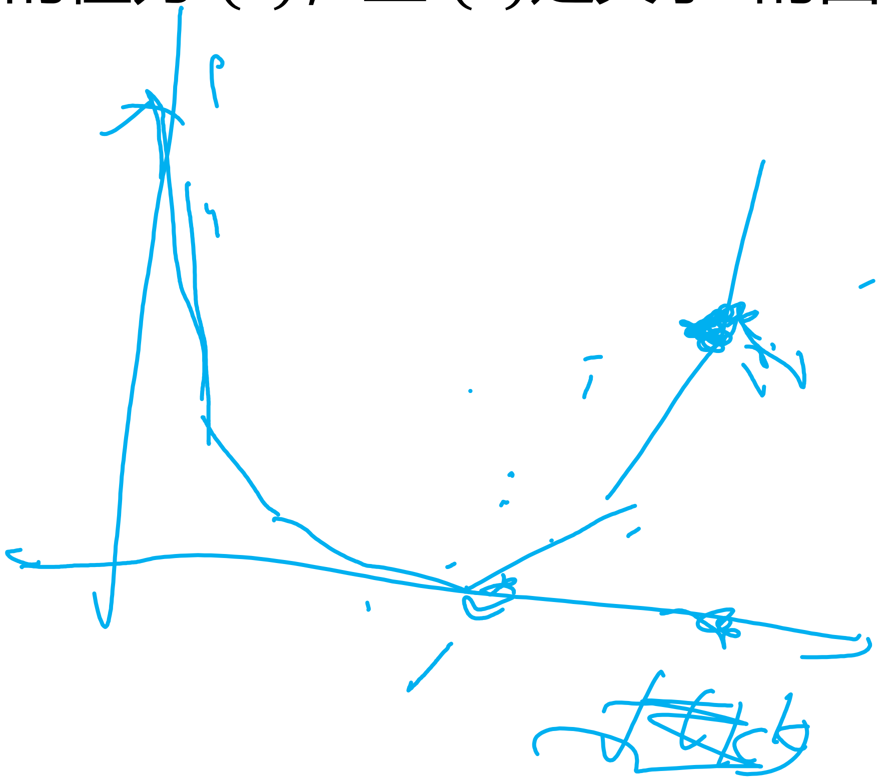
- 什么是凸呢？差分是单调递减（或递增）的（就f[x+1]-f[x]单调）
- 怎么判断是不是凸的呢？
- 一般不要去证明啊！
- 要么打表试一试。
- 要么假装是凸的，写一个带权二分上去，看看对不对。（大部分（当然不是所有）涉及“恰好k个”的问题，往往具有凸性，可以用带权二分。

带权二分

- 好了，刚刚我们以两道题为例，简单介绍了一下什么时候可以使用带权二分。
- 接下来我们介绍一下带权二分的原理。

带权二分

- 我们形式化地表述一下我们要解决的问题：
- 设选x个东西的情况下，最优的值为f(x)，且f(x)是关于x的凸函数（当然其实是一些离散的点
- 目标是求f(k)



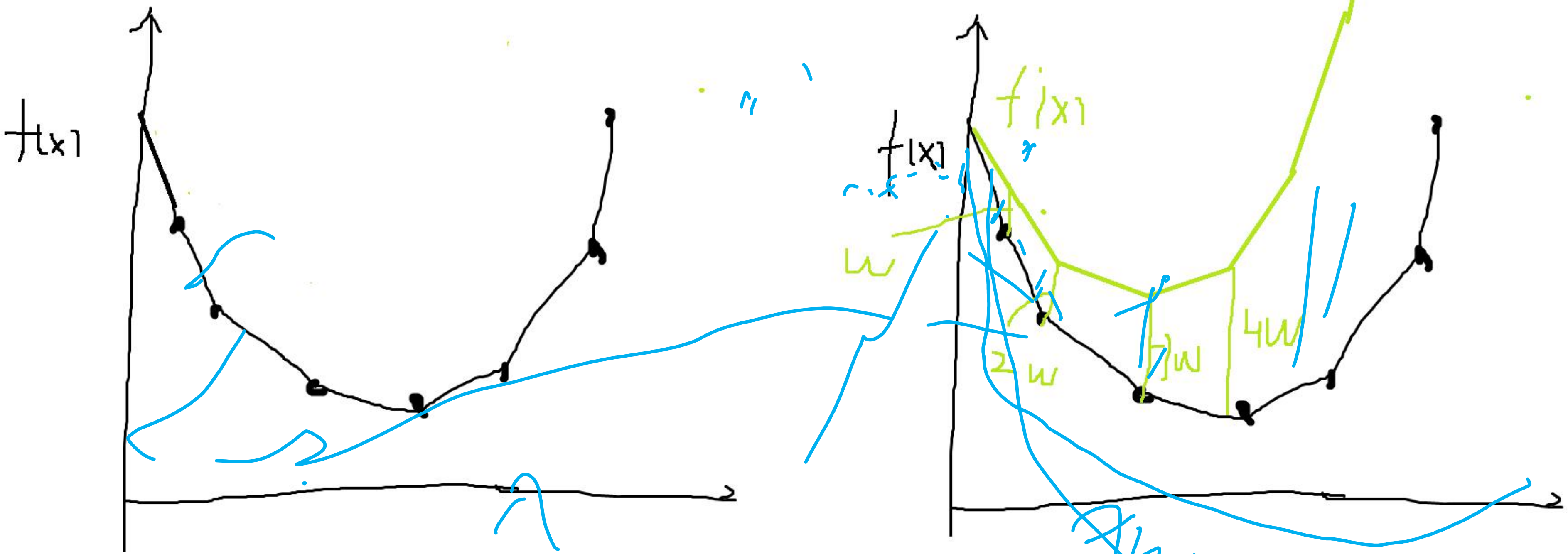
带权二分

- 我们能干什么？首先任意位置的f(x)我们是不会求的，所以f(k)没法直接求。
- 我们会求（或者说，能比较快地求出）全局的最小值，同时也可以顺便求出，取到全局最小值的x是多少
- 比如说，我们会求最小生成树，顺便就能记录选了几条白边。
- 我们也会n^2（相较原做法的n^3，比较快）求出，在没有“k段”限制下的最小方差（设dp[i]，O(n)枚举左端点转移即可））。

带权二分

- 那咋办?
- ——想个办法，让k处取最小值
- 啥办法?
- ——每选一个东西，把值额外加上一个w（可以是负的）。
- 也就是让$f'(x)=f(x)+x*w$
- 大家可以发现，随着w的变化，有一天，最小值会在k处取到!
- 请看图感受:

带权二分



- 大家可以发现，我给f(x)加上一个x*w，最小值点从4变成了3

带权二分

- 可以想见，如果w继续增大，那么最小值点x0会继续变小；
- 如果w减小以至于变成负数，那么最小值点x0则 会不断变大
- 那么总会有一个w，使得最小值在k处取到。
- ——那不就可以二分了嘛
- x0<k，那就让w变小点
- x0>k，那就让w变大点
- 欸，现在问题不就只需要求，f'(x)的最小值点x0了嘛！

r = w

l = w

带权二分

- 刚刚说了，我们会求f(x)的最小值和最小值点。
- 那么f'(x)的最小值和最小值点，往往也是不难求的。
- 比如说，在Tree中，只需要让白边边权增加一个w
- 在归程中，只需要每加一段，给花费加上一个w，即
- f[i]=min(f[j]+cost(i,j))+w
- 大概先感受一下，cost是啥一会讲题时再说
- 当然，顺便就可以在求出f'(x)的最小值的过程中，求出取到最小值的那个点x0了。（顺便记录白边个数，或是在dp时顺便记录当前找了几段。

带权二分

- 那就简单了!重述一下整个流程：
- 我们二分一个值w(边界可以设置地大一些，从-1e9到+1e9这样，当然也可能得根据题目的数据范围调一调)
- 然后，我们去掉“恰好有k个东西”的限制，但是每多有一个东西，就额外加上w的花费。即令f'(x)=f(x)+x*w，其中x东西数
- 然后，不管用DP还是贪心啥的方法，求出f'(x)的最小值f'(x0)，顺便求出此时的最小值点x0。
- 比较一下x0和k的大小关系。如果x0<k了，那就让w小一点（二分到左边），否则二分到右边。
- 最终，我们就能让x0=k，即我们求出了f'(x)的最小值f'(k)。

带权二分

- 再怎样求答案？那简单啦！
- f(k)不就等于f’(k)-w*k 嘛！

带权二分

- 看看代码：

```
int l=-110,r=110;
while(l<=r){
    int mid=(l+r)>>1;
    Kruscal(mid);
    if(now0>=K){
        ans=ansn-K*mid;
        l=mid+1;
    }
    else r=mid-1;
}
```

- mid就是上面的w啦！

[SDOI2016]征途

- Tree应该不用讲了吧！我们再稍微讲一下征途这个题

m: $x_1, x_2 \cdots x_m$

$$\bar{x} = \frac{1}{m}\sum_{i=1}^{m} x_i$$

$$s^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \bar{x})^2$$

$$s = \sqrt{s^2}$$

[SDOI2016]征途

$f_{i,j}$ $-$ $t[k,j]$

n^3

- 咱先不考虑带权二分的事哈，
- 首先，方差有一个很常用的公式：
- $$s^2 \times m^2 = -\left(\sum_{i=1}^{m} v_i\right)^2 + m \times \left(v_1^2 + v_2^2 + ... + v_m^2\right)$$
- 然后发现左边不就是所有元素的和的平方嘛，是个定值呀！
- 那就是最小化右边这一坨。
- 也就是说，现在问题是：给定n个数，要求划分成恰好m段，最小化每段的和的平方的和。
- （换句话说，前面PPT里出现的cost(i,j)，就是(sum[j]-sum[i-1])^2啦（其中sum是前缀和

[SDOI2016]征途

- 转移时顺便记录当前分了几段，用于wqs二分：

```
for(int i=1;i<=n;i++){
    for(int j=0;j<i;j++){
        if(f[j]+(sum[i]-sum[j])*(sum[i]-sum[j])+md<f[i
        ||(f[j]+(sum[i]-sum[j])*(sum[i]-sum[j])+md==f[
            f[i]=f[j]+(sum[i]-sum[j])*(sum[i]-sum[j])+
            d[i]=d[j]+1;
        }
    }
}
```

mid

- d[]就是记录分了几段啦
- 为啥if里面的条件这么复杂？我们一会再说。

[SDOI2016]征途

- 把这个DP外面套上wqs二分，就搞完啦！
- 甚至可以再加上斜率优化！
- 就是说这题斜率优化/wqs二分，用上任意一种都可以过
- 两个都用就可以做到nlogV，巨大快。

带权二分的一个代码细节

- (以Tree为例:
- 有这样一个问题：当你二分到一个w的时候，可能选3/4/5/6条黑边都可以保证值最小。那...取哪一个?
- 一个处理方法:
- 先尽量让选的黑边尽量少，算出一个下界;
- 再尽量让选的黑边尽量多，算出一个上界
- 如果k在下界和上界之间，就说明找到了。

带权二分的一个代码细节

- 也可以这么做：如果尽量选白边的话，相当于求出了上界，二分需要这么写：
- 就是说，当now0>k的时候，也统计一下答案

```
int l=-110,r=110;
while(l<=r){
    int mid=(l+r)>>1;
    Kruscal(mid);
    if(now0>=K){
        ans=ansn-K*mid;
        l=mid+1;
    }
    else r=mid-1;
}
```

带权二分的一个代码细节

- 也可以尽量选黑边，相当于求出了下界，那么二分则需要这么写：
- 就是说，当now0<k的时候，也统计一下答案

```
int l=-110,r=110;
while(l<=r){
    int mid=(l+r)>>1;
    Kruscal(mid);
    if(now0>K){
        l=mid+1;
    }
    else{
        ans=ansn-K*mid;
        r=mid-1;
    }
}
```

谢谢大家！