**斜率优化dp小结**

单调队列优化

在写斜率优化之前，我们来回顾一下单调队列优化的dp

1. 对于如下形式的dp方程  dp[i]=min{dp[j]+f(j)}(0<j<i)

我们直接用一个变量维护(0, i)中dp[j] + f(j)的最小值即可

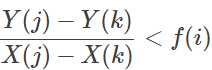
2.对于如下形式的dp方程  dp[i]=min{dp[j]+f(j)}(i−m<j<i)

我们可以用一个单调队列维护一个(i - m, j)中dp[j] + f(j)的最小值，然后做到O(1)转移。

斜率优化

基本形式

但是对于形如  dp[i]=min{dp[j]+f(i,j)}

的方程，无法做到O(1)计算dp[j]+f(i,j)的最小值，这时就需要斜率优化这个技巧来解决这个问题了。 令k < j < i，当我们更新dp[i]时，如果有dp[j] + f(i, j) 比dp[k] + f(i, k)更优，则有dp[j] + f(i, j) - (dp[k] + f(i, k) < 0，对于这个不等式如果能够化解成如下形式 

我们就能通过斜率优化这个dp了。

HDU 3507，很适合的一个入门题。<http://acm.hdu.edu.cn/showproblem.php?pid=3507>

大概题意就是要输出N个数字a[N]，输出的时候可以连续连续的输出，每连续输出一串，它的费用是 “这串数字和的平方加上一个常数M”。

我们设dp[i]表示输出到i的时候最少的花费，sum[i]表示从a[1]到a[i]的数字和。于是方程就是：dp[i]=min{dp[j]+M+(sum[i]−sum[j])2 (0<j<i)}；

很显然这个是一个二维的。题目的数字有500000个，不用试了，二维铁定超时了。那我们就来试试斜率优化吧，看看是如何做到从O(n^2)复杂度降到O(n)的。

令k < j < i，当有  dp[j]+M+(sum[i]−sum[j])2−(dp[k]+M+(sum[i]−sum[k])2)<0

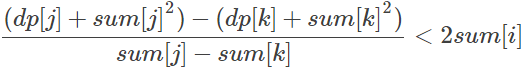
dp[j]+M+(sum[i]-sum[j])^2<dp[k]+M+(sum[i]-sum[k])^2。dp[j]+M+sum[i]^2-2\*sum[i]\*sum[j]+sum[j]^2<dp[k]+M+sum[i]^2-2\*sum[i]\*sum[k]+sum[k]^2

dp[j]+ sum [j]^2-(dp[k]+ sum [k]^2) < 2\* sum [j]\*sum[i] - 2\*sum [k] \*sum[i]

dp[j]+ sum [j]^2-(dp[k]+ sum [k]^2) <sum[i]\* (2\*( sum [j]- sum [k]))

(dp[j]+ sum [j]^2-(dp[k]+ sum [k]^2))/(2\*( sum [j]- sum [k]))<sum[i]。

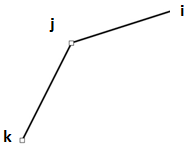
从j转移到i, 比从k转移到i更优，变换此不等式可得:

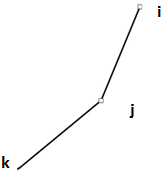


令Y(i)=dp[i]+sum[i]2, X(i)=sum[i], f(i)=2sum[i]则将此不等式化解为上述形式。

优化方法: 

可以发现，若满足则j转移到i，比k转移到i更优，如果我们把(X(j), Y(j)), (X(k), Y(k))当成平面上的两个点Pj, Pk，这个不等式的含义即为若的斜率＜f(i)则，从j转移更优。

令grad(i, j)表示的斜率，现在我们假设grad(i,j)<grad(j,k)，若grad(i, j)<f(a),则i比j更优，若grad(i, j) > f(a), 则grad(j, k) > f(a),那么从k转移比从j转移更优，当grad(i, j) < grad(j, k)的时候，无论如何j转移到a都不会是最优。而这种情况恰好对应下图   
   
所以这种情况时，我们可以直接把j点删除，最后能够转移的点集只会存在这种图形，



所以最后我们维护一个上凸集即可。

但是此时我们还是没有解决最终问题，如何才能找到转移到i点的最优的点呢。可以发现最后的点集一定是一个凸集，也就是斜率单调！！这样对于k < j, grad(j,k) < f(i),时更优，从图形特点我们可以发现如果j比k优，那么j点比所有比k小的点都优，所以对于每一个f(i),我们维护一个所有比i点小的凸集，二分查找斜率比f(i)小的编号最大的点，就是最优的转移点。如果f(i)也满足单调性，比如这道题，我们还可以直接维护一个单调队列就能解决这个问题。

于是对于这题我们对于斜率优化做法可以总结如下：

1，用一个单调队列来维护解集。

2，假设队列中从头到尾已经有元素a b c。那么当d要入队的时候，我们维护队列的上凸性质，即如果g[d,c]<g[c,b]，那么就将c点删除。直到找到g[d,x]>=g[x,y]为止，并将d点加入在该位置中。

3，求解时候，从队头开始，如果已有元素a b c，当i点要求解时，如果g[b,a]<sum[i]，那么说明b点比a点更优，a点可以排除，于是a出队。最后dp[i]=getDp(q[head])。

#include<iostream>

#include<string>

using namespace std;

int dp[500005];

int q[500005];

int sum[500005];

int head,tail,n,m;

int getDP(int i,int j){

return dp[j]+m+(sum[i]-sum[j])\*(sum[i]-sum[j]);

}

int getUP(int j,int k) //yj-yk的部分{

return dp[j]+sum[j]\*sum[j]-(dp[k]+sum[k]\*sum[k]);

}

int getDOWN(int j,int k) //xj-xk的部分{

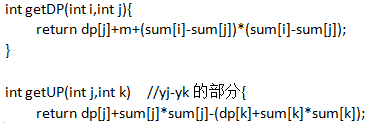
return 2\*(sum[j]-sum[k]);

}

int main(){

int i;

freopen("D:\\in.txt","r",stdin);

 while(scanf("%d%d",&n,&m)==2) {

for(i=1;i<=n;i++)

scanf("%d",&sum[i]);

sum[0]=dp[0]=0;

for(i=1;i<=n;i++)

sum[i]+=sum[i-1];

head=tail=0;

q[tail++]=0;

for(i=1;i<=n;i++){

while(head+1<tail && getUP(q[head+1],q[head])<=sum[i]\*getDOWN(q[head+1],q[head]))

head++;

dp[i]=getDP(i,q[head]);

while(head+1<tail && getUP(i,q[tail-1])\*getDOWN(q[tail-1],q[tail-2])<=getUP(q[tail-1],q[tail-2])\*getDOWN(i,q[tail-1]))

tail--;

q[tail++]=i;

}

printf("%d\n",dp[n]);

}

return 0;

}

分治做法

对于f(i)单调的这种情况，除了使用单调队列优化的斜率优化做，我们还有另外一种分治的做法，但是复杂度会变成O(nlogn) 比O(n)差。   
当f(i)单调的时候，我们可以发现若a > b,则f(a) > f(b),设转移到a的最优点是c，转移到b的最优点是d，一定有c > d。也就是转移到a的最优点一定大于等于转移到b的最优点。考虑这样的分治

void dfs(int l, int r, int dl, int dr) {

//[l,r]表示现在更新[l,r]区间dp[i]的最优值

//用j -> f(i),表示j是更新f(i)最优值的最优点

//那么[dl,dr]表示更新dp([l,r])的点，一定在[dl,dr]范围内

int mid = (l + r) >> 1;

int dm = dl;

int g = inf;

for (int i = dl; i <= dr; i++) {

if(g < dp[i] + f(i, mid)) {

g = dp[i] + f(i, mid);//记录更新dp[mid]的最优

dm = i;//记录更新dp[mid]的最优点

}

}

dp[mid] = g; //更新dp[mid]的值

//因为上文叙述的单调性，

//更新[l,mid-1]的最优点，一定在[dl,dm]范围内

if(l < mid) dfs(l, mid - 1, dl, dm);

//更新[mid+1,r]的最优点，一定在[dm,dr]范围内

if(mid < r) dfs(mid + 1, r, dm, dr);

//此份代码dfs顺序有点问题，并不正确，但是并不影响理解

}

可以发现这个分治比起斜率优化，不仅写起来方便很多，并且适用的范围也更广。这个做法不局限于斜率单调，可以发现只要满足c是更新f(a)的最优点，d是更新f(b)的最优点，若a > b 一定有 c > d，则可以有这个分治做。

**HDU 3507 Print Article（CDQ分治+分治DP）**

【题目链接】 <http://acm.hdu.edu.cn/showproblem.php?pid=3507>

 【题目大意】

　　将长度为n的数列分段，最小化每段和的平方和。

【题解】

　　根据题目很容易得到dp[j]=min(dp[k]+(s[j]-s[k])2)，因为是从前往后转移，且决策单调，因此在CDQ分治的同时进行分治DP即可。

【代码】

#include <cstdio>

typedef long long LL;

const int N=500005;

int n,M,t;

LL f[N],g[N],a[N],s[N],INF=1LL<<60;

void DP(int l,int r,int dl,int dr){

int m=(l+r)>>1,i,dm=0;

LL &ret=g[m]; ret=INF;

for(i=dl;i<=dr&&i<m;i++){

LL t=f[i]+(s[m]-s[i])\*(s[m]-s[i])+M;

if(t<ret)ret=t,dm=i;

}if(l<m)DP(l,m-1,dl,dm);

if(r>m)DP(m+1,r,dm,dr);

}

void CDQ(int l,int r){

if(l==r)return;

int mid=(l+r)>>1;

CDQ(l,mid);

DP(mid+1,r,l,mid);

for(int i=r;i>mid;i--)if(g[i]<f[i])f[i]=g[i];

CDQ(mid+1,r);

}

int main(){

while(~scanf("%d%d",&n,&M)){

for(int i=1;i<=n;i++){scanf("%lld",&s[i]);s[i]+=s[i-1];}

for(int i=1;i<=n;i++)f[i]=s[i]\*s[i]+M;CDQ(0,n);

printf("%lld\n",f[n]);

}return 0;

}