# 0.前言

由于做题进程中常会遇到需要优化的DP，于是惊讶地发现（2）（3）动态规划水集同时开工了。

DP优化分很多种，从**破环成链**这种不算优化的优化到**斜率优化**这种省选级别的优化应该都会提到..吧，首先我得先学会。

由于划水集（2）（3）我是同时写的，（3）里面涉及很多（2）的例题，请大家结合观看。

# 一些你初学DP就知道的优化

## 倒序优化

**01背包的一维形态**实质上就是倒序优化的前身，就是忽略不可控的起点所造成的的因素，从终点倒着来跑DP。

这个我也不知道啥时候可以用啊。。多做点题凭感觉呗。

### [Nike的任务](https://www.luogu.org/problemnew/show/P1280)

#### 题面： 见链接

#### 口胡分析：

是个人都会设f[i]为第i分钟所得的最大休息时间，但转移就很烦。

在第i秒时的结果是由你之前有没有接任务决定的，但是这个信息你并不能表示在DP中，所以就会很难受。

但我们可以想到接任务的那一秒是确定的，而且**往后的延续时长是确定的**，那么从后往前转移是很好做的。

那我们就倒着跑，设f[i]是i秒之后可得到的最大休息时间，那么有两个转移方案：

1. 这个时间点没工作，直接休息:  *f*[*i*]=*f*[*i*+1]+1
2. 有长为k时间工作，找最大的转移过来： *f*[*i*]=*max*(*f*[*i*],*f*[*i*+*k*])

f[1]*f*[1]即为答案。

*UPDATE*:学完期望DP的我来顺便修下锅。。我曾一度以为倒序优化没有那么重要。

但关于期望部分的DP，通常只能采用倒序才可以求解，可以去划水集（2）看看相关例题了解详细情况，比如**奖励关**。

## 破环成链

这个学模拟时候就用到了。。再说一遍也挺无聊的。。

*UPDATE*: 写了区间DP的我来补锅了，这是将环问题转换为链问题的重要途径，感觉还是要提一下的。。

如果题目中的区间是环形结构，那就不满足DP的无后效性了。

但我们在输入过程中把它倍长，然后跑 [1,*n*]就可以保证取到所有起点的区间，这样仍可取到环形结构中的所有情况，是等价的。

通常情况下在输出时候这样写：

for (int i=1;i<=n;i++) ans=max(ans,f[i][i+n-1]);

大家可以找[教主的花园](https://www.luogu.org/problemnew/show/P1133)，或[能量项链](https://www.luogu.org/problemnew/show/P1063)这类的题回顾一下。。

## 降维打击

其实这个受众面其实就很广了，通常都是你先写出的一个最基本的DP方程。后来通过对题目特有条件的分析，从而省略掉的其中的一维或者多维，**达到降低时空复杂度的目的**。

你要我举啥例题的话，**划水集（2）线性DP那章的**[**传纸条**](https://www.luogu.org/problemnew/show/P1006) 其实就可以用此方法优化。

由于行程时间相同，所以两个人走的总步数 *L*是一样的，又因为是曼哈顿距离只能写成 *x*+*y*，所以只需要维护一维 *x*，令一维就可用 *L*−*x*算出来，这样 *f*[*L*][*x*1​][*x*2​]就只有三维了。。

# 一些NOIP提高组常用优化

## 滚动数组

1. 一个最简单的例子就是**斐波那契数列**，如果要求 N=10^7左右的数据，你数组肯定存不下，但你DP可以跑啊！

因为斐波那契只和前两项有关，于是你就开个 *f*[3]，转移的时候*f*[*i*%3]=*f*[(*i*−1)%3]+*f*[(*i*−2)%3]就行了，刚才式子口胡不知道对不对。

1. 还有就是01背包，平时的*f*[*i*][*j*]表示到了第i个物品有j个容量所达到的最大价值，通常都将其倒序优化。

如果你死活都想正着写，那么根据转移方程*f*[*i*][*j*]=*max*(*f*[*i*][*j*],*f*[*i*−1][*j*−*w*[*i*]]+*v*[*i*])可知，每个状态只与第一维的前一行有关，那你就只用开个 *f*[2][*V*]，类似上面那样膜个2就可以正常的跑过大数据了。

1. 以上都是很简单的滚动数组了，在划水集（2）的状压DP章节里，[**炮兵阵地**](https://www.luogu.org/problem/P2704) 也要用到滚动数组，大家可以去康康。

## 数据结构优化

对于形如*f*[*i*]=*maxj*∈[*x*,*y*]​(*f*[*j*])+*C* (*x*<=*y*<*i*)的转移方程式，如果想 *O*(*N*)枚举最值，会很慢。

但由于某些数据结构可以高效地维护区间最值，所以只要将 f[i]*f*[*i*]的序列丢到这类数据结构里面去，每次查询 [*x*,*y*]的相关信息，然后再把新的 f[i]*f*[*i*]插回维护用的数据结构里面，就可以将 *O*(*N^*2)降至 *O*(*N*∗*logN*)了~~

NOIP常用的基本上只有单调队列以及线段树，树状数组这类的基础数据结构，当然平衡树啥的也可以达到类似效果，但本文只会着重谈前面两个。

### 单调队列：

在NOIP比赛中，动规的优化部分往往不是考试重点，所以**单调队列**中具有更广泛的应用范围，而且并不难写。

既然是单调队列，那么其对于**决策取值范围单调变化**的题目也将更适用一些（比如下面这道例题）。如果 max的取值范围很诡异（比如说 *LIS*的 *a*[*i*]<*a*[*j*]，或者∑*k*=*ij*​*a*[*k*]≤*M*啥的），单调队列就很难有用武之地了。

限于篇幅原因，不方便在此细谈单调队列的具体写法。 对此，推荐大家可以参照 [**滑动窗口**](https://www.luogu.org/problem/P1886)**中的题解部分**，或找博客进行学习。顺便一提，笔者个人比较喜欢用STL<queue>库里的deque双向队列来维护，当然手写可能会更快。

#### 单调队列优化DP的例题:[*NOIP* *pj*的跳房子](https://www.luogu.org/problem/P3957)

##### 题面：

给你有n的格子的线段，每个格子距离原点 *x*[*i*]，且有价值 *s*[*i*]，给予g的花费即可从当前位置向前跳 [*max*(*d*−*g*,1),*d*+*g*]范围内任一长度但必须跳到格子上，可以在任意时候停止，求可达到k的价值所需的最小花费g。

##### 口胡分析：

既然要求最大值最小，想都不想直接二分答案，将问题转变为“给你mid的花费，问在游戏过程中的价值最大值”。

普及组都可以看出来这是个DP并写出来它的朴素形式。若设 f[i]*f*[*i*]为跳到第i格的最大价值，不难得出转移方程： *f*[*i*]=*x*[*i*]−*r*≤*x*[*k*]≤*x*[*i*]−*l*max​(*f*[*k*])+*s*[*i*]

其中*l*=*max*(*d*−*g*,1),*r*=*d*+*g*。

这个式子很明显满足本篇开头的那个形式，故可以对中间最值部分用单调队列维护即可，在此给出二分的check函数：

bool check(ll g)

{

deque<node> q; memset(f,0,sizeof(f));

ll l=max((ll)1,d-g),r=g+d,ans=0;

for (ll i=1,j=0;i<=n;i++)

{

while (x[j]<=x[i]-l) { //当f[k]进入右端范围，左端范围应另判。

while (!q.empty()&&f[j]>q.back().v) q.pop\_back();

//维护单调性。

q.push\_back((node){x[j],f[j]}); j++;

}

while (!q.empty()&&q.front().n<x[i]-r) q.pop\_front();

//当f[k]离开左端范围

if (q.empty()) f[i]=-21474836470;

else f[i]=q.front().v+s[i],ans=max(f[i],ans);

//直接赋值即可。

}

return (ans>=k);

}

### 线段树，树状数组

亦可适用于对本篇开头那个形式的优化，上面那道跳房子洛谷也有用线段树A掉的题解，但是对于普遍题目，**线段树需要占有更大的空间，甚至需要用到离散化。** 所以我们能用单调队列时候尽量不考虑用线段树。

但正如上文所说，单调队列仅适用于取值范围单调变化的题目。若**取值范围上下边界不明确，可能会影响到之前的状态时**，我们更倾向于用线段树或更高级的数据结构解题。

#### [USACO例题](https://www.luogu.org/problem/P4644)

##### 题面：

给你一个长为N的全0序列，有m个操作可以使 [l\_i,r\_i][*li*​,*ri*​]变为1，花费 c\_i*ci*​。问至少花多少钱才能使其变为全1序列。

##### 口胡题解：

我们先通过对操作的 r\_i*ri*​排序使其单调，再设 f[i]*f*[*i*]表示使 [1,i][1,*i*]范围变为全1序列的最小花费。

易推得：f[r\_i]=\max\_{l\_i-1\leq k\leq r\_i}(f[k])+c\_i*f*[*ri*​]=*li*​−1≤*k*≤*ri*​max​(*f*[*k*])+*ci*​

显然修改后的 f[i]*f*[*i*]会影响到 f[k]*f*[*k*]的最值取值，于是我们对 f[i]*f*[*i*]用线段树维护，支持区间求最值，单点修改，然后就是板子了。

### 倍增优化

# 省选才考得到的优化

* 斜率优化
* 四边形不等式优化

# 一些你想不到的玄学优化

由于做题先后问题，先把这一天赋点开了。。

既然是个例的玄学优化，那么就以例题的形式给出了。

## [NOIP原题，过河](https://www.luogu.org/problemnew/show/P1052)

### 题面：

给你一个L（ L<=10^9*L*<=109）的路上面有N（ N<=100*N*<=100）个刺儿，给定跳远范围 [S,T][*S*,*T*]，问要跳到终点最少得扎多少刺儿？

### 口胡分析：

首先是个人都想得出30分DP（ L<=300*L*<=300）：

设f[i]为长为i时最小踩刺次数，则 f[i]=f[i-(s->t)]+(cier[i]?)*f*[*i*]=*f*[*i*−(*s*−>*t*)]+(*cier*[*i*]?)，大概这意思。。

但L太大辣！于是我们就想想办法优化一下。

然后因为刺的个数很少，但是也不能以刺为状态写DP，于是看看刺与刺之间的路能不能缩进。

**于是想到我们一定能跳到 dis=lcm(s,s+1,..,t-1,t)*dis*=*lcm*(*s*,*s*+1,..,*t*−1,*t*)处，于是把这段路对 dis*dis*取个模就行了？？**

。。。不是，这个优化是怎么想到的，怎么证明的啊

这道题优化就玄学，还有其他方法可以把距离缩成71以内，真的不是很懂。

## [NOIP2018普及，摆渡车](https://www.luogu.org/problem/P5017)

### 题意：

一辆车一队人，给定每人的到达时间和一次来回的时间，问最少等待时间？

### 口胡分析：

这题有两种思路：对于小范围的 n,m*n*,*m*列转移方程，或是对 T*T*列转移方程。我两种都写了一下，各有特色，这里只讲后者。

首先我们可以在时间链上标记出每个人的时刻，设 f[i]*f*[*i*]为到了第 i*i*时刻为止所有人的最小等待时间。

一个好想的转移思路就是我从某一时刻开始就在起点等，一直等到当前时刻才把所有人一窝子全送走，若利用前缀和啥的处理一下，这明显是一个 O(T^2)*O*(*T*2)的算法。

* 考虑第一个玄学优化：我们可以先考虑一个人最多会等多久。

如果他倒霉，卡在车子走的下一秒到达起点，那么就必须等 m-1*m*−1秒让车子开回来；如果下一个 m*m*内还有贡献更多的人群，那么他还要继续等。当其贡献更多人群超过 m*m*秒后，就完全可以先把那个倒霉的人送走再回来接贡献大的。所以最劣情况是 2m2*m*。

基于此，一个人产生贡献最优区间可以锁定在 (i-2m,i-m](*i*−2*m*,*i*−*m*]内，我们可以缩小第二次枚举的长度，使其 O(T^2)\to O(Tm)*O*(*T*2)→*O*(*Tm*)，已经优化很大了。

* 考虑第二个玄学优化：大量的空白时间。

由于200个人占有五个数量级的时间轴，空白时间一定是很多的。对于 [i-m,i][*i*−*m*,*i*]区间若没有人的情况，我们完全可以通过 f[i]=f[i-m]*f*[*i*]=*f*[*i*−*m*]对其加速。但这个其实很玄学，若 m*m*较大且分布均匀很可能卡掉，但显然 CCF*CCF*没有那么做。于是就玄学 AC*AC*了。