优化DP

1.决策单调性优化DP(一维)

引理

四边形不等式

> *设 w(x,y) 是定义在整数集合上的二元函数。若对于定义域上的任意整数 a,b, 其中 $a \le b \le c \le d$，都有 $w(a,d)+w(b,c) \ge w(a,c)+w(b,d)$ 成立，则称函数 w 满足**四边形不等式**。*

四边形不等式的另一种形式

> *设 w(x,y) 是定义在整数集合上的二元函数。若对于定义域上的任意整数 a,b，其中 a<b，都有 $w(a,b+1)+w(a+1,b) \ge w(a,b)+w(a+1,b+1)$ 成立，则函数 w 满足**四边形不等式**。*

定理

> *在状态转移方程 F[i] = $min_{0\le j<i}${ F[i] + val(j,i) } 中，若函数 val 满足四边形不等式，则F具有决策单调性。*

当函数具有决策单调性时，可以把 F[i] = $min_{0\le j<i}${ F[i] + val(j,i) } 的计算时间从 O(N^2) 优化到 O(NlogN)

模板

(''')

```
#include <bits/stdc++.h>
using namespace std;
const int maxn = 1e5+10;
//单调队列三元组(l,r,j)
int l[maxn],r[maxn],p[maxn];
//计算 F[i] + val(j,i)
ll cal(int j,int i){
    return ...;
}
//DP
void solve(){
    //队头，队尾，初始化
    int Begin = 1, End = 1;
    l[End] = 1, r[End] = N, p[End] = 0;
    //DP循环
    for(int i=1;i<=N;i++)
    {
        //更新F[i]
        if(i>r[Begin]) Begin++;
        else l[Begin] = i;
        f[i] = cal(p[Begin],i);
        //更新队列
        int pos = -1;
        while(Begin<=End) {
            //若对于F[l[End]]来说，i是比p[End]更好的决策
            if(cal(i, l[End]) <= cal(p[End], l[End])) pos = l[End--];
            else {
                //若对于F[r[End]]来说，i不是比p[End]更好的决策
                if (cal(p[End], r[End]) <= cal(i, r[End])) break;
                //否则，二分寻找答案
                int L = l[End], R = r[End];
                while(L<=R){
                    int mid = (L+R)>>1;
```

```
                    if(cal(i,mid) <= cal(p[End],mid)) pos = mid, R = mid-1;
                    else L = mid + 1;
                }
                r[End] = pos-1;
                break;
            }
        }
        if(pos == -1) continue;
        l[++End] = pos, r[End] = N, p[End] = i;
    }
}
```

(''')

例题

诗人小G

题意

有一个长度为N的序列和常数 L,P，你需要将它分成若干段，每段的代价为 $|\sum(Ai) - L|^P$ ，求最小代价的划分案。n≤10^5,1≤P≤10。

代码

(''')

```
#include <bits/stdc++.h>
#define ll long double
using namespace std;
const int maxn = 1e5+10;
const long long limit = 1e18;
int N,L,P;
int l[maxn],r[maxn],p[maxn];
ll sum[maxn]{0}, f[maxn]{0};
char s[50];
ll qpow(ll a,int p){
    ll ret = 1;
    for(;p;p>>=1){
        if(p&1) ret = ret*a;
        a = a*a;
    }
    return ret;
}
ll cal(int j,int i){
    return f[j] + qpow(abs(sum[i]-sum[j]+i-j-1-L),P);
}
void solve(){
    int Begin = 1, End = 1;
    l[End] = 1, r[End] = N, p[End] = 0;
    for(int i=1;i<=N;i++)
    {
        if(i>r[Begin]) Begin++;
        else l[Begin] = i;
        f[i] = cal(p[Begin],i);
        int pos = -1;
        while(Begin<=End) {
            if(cal(i, l[End]) <= cal(p[End], l[End])) pos = l[End--];
            else {
                if (cal(p[End], r[End]) <= cal(i, r[End])) break;
                int L = l[End], R = r[End];
                while(L<=R){
                    int mid = (L+R)>>1;
                    if(cal(i,mid) <= cal(p[End],mid)) pos = mid, R = mid-1;
                    else L = mid + 1;
                }
                r[End] = pos-1;
                break;
            }
        }
        if(pos == -1) continue;
        l[++End] = pos, r[End] = N, p[End] = i;
    }
```

```
}
int main(){
    int T; cin>>T;
    while(T--){
        scanf("%d%d%d",&N,&L,&P);
        for(int i = 1; i<=N; i++){
            scanf("%s",s);
            sum[i] = strlen(s) + sum[i-1];
        }
        solve();
        if(f[N]>limit) puts("Too hard to arrange");
        else printf("%lld\n",(long long)f[N]);
        puts("--------------------");
    }
}
```

(''')

2.决策单调性优化DP（二维）

引理

> *在区间DP问题，遇到类似下面这样的转移方程*
>
> *F[i,j] = $min_{i \le k < j}${F[i,k] + F[k+1,j] + w(i,j)}*
>
> *利用四边形不等式，可以进行优化*

定理

> *在状态转移方程F[i,j] = $min_{i \le k < j}${F[i,k] + F[k+1,j] + w(i,j)} 中，如果下面两个条件成立：*
>
> *1.w满足四边形不等式*
>
> *2.对于任意的 $a \le b \le c \le d$，都有 $w(a,d) \ge w(b,c)$。*
>
> *那么F也满足四边形不等式*

定理（二维决策单调性）

> *在状态转移方程F[i,j] = $min_{i \le k < j}${F[i,k] + F[k+1,j] + w(i,j)} 中，记P[i,j]为令F[i,j]取最小值的k值。*
>
> *如果F满足四边形不等式，那么对于任意i<j, 有 $P[i,j-1] \le P[i,j] \le P[i+1,j]$。*

根据上面定理，我们只需要在 P[l,r-1]≤k≤P[l+1,r]范围内对k进行枚举，求出F[l,r]和P[l,r]。算法时间复杂度为

$O(\sum_{i=1}^{N-1}\sum_{j=i+1}^{N}(P[i+1,j]-P[i,j-1]+1))$ = $O(\sum_{i=1}^{N-1}(P[i+1,N]-P[1,N-i]+N-i))$ = $O(N^2)$

例题

再探石子合并

题意

与“石子合并”相同，数据范围$N \le 5000$。

代码

(''')

```
#include<bits/stdc++.h>
using namespace std;
const int maxn =  5000+10;
typedef long long ll;
int n,p[maxn][maxn];
ll a[maxn],sum[maxn],f[maxn][maxn];
```

```
int main(){
    while(scanf("%d",&n) == 1 && n) {
        for (int i = 1; i <= n; i++)scanf("%lld", &a[i]);
        //初始化
        memset(f,0x3f,sizeof(f));
        for (int i = 1; i <= n; i++) {
            f[i][i] = 0;
            sum[i] = sum[i - 1] + a[i];
            p[i][i + 1] = i;
        }
        for(int i = 1; i<n; i++){
            f[i][i+1] = sum[i+1]-sum[i-1];
        }
        //阶段len，状态l，决策k
        for (int len = 2; len <= n; len++){
            for(int l = 1; l <= n - len + 1; l++){
                int r = l + len - 1;
                //在 P[l,r-1]$\leq$k$\leq$P[l+1,r]范围内对k进行枚举
                for(int k = p[l][r-1]; k<=p[l+1][r];k++){
                    //F[i,k] + F[k+1,j] + w(i,j)
                    ll t = f[l][k]+f[k+1][r] + sum[r] - sum[l-1];
                    if(t < f[l][r]){
                        p[l][r] = k, f[l][r] = t;
                    }
                }
            }
        }
        printf("%lld\n", f[1][n]);
    }
}
```

(''')