Hello world ---区间DP

/\*

区间DP 好题

其实我也还没理解这道题跟回文有什么不用一样。

正向其实可以看成j(区间右端)不断移动；逆向看成i(区间左端)不断移动。

然后转移的时候就可以根据方向来搞了。

注释掉的4条语句是逆向的想法。就是状态转移的时候i是从右往左。

\*/

//lightoj 1422

int main()

{

int cases, Cas = 0;

scanf("%d", &cases);

while(cases--) {

scanf("%d", &n);

for(int i = 1; i <= n; i++) scanf("%d", &a[i]);

for(int l = 1; l <= n; l++)

//for(int i = n; i > 0; i--) { //为什么要逆着？

for(int i = 1; i + l - 1 <= n; i++) {

int j = i + l - 1;

f[i][j] = f[i][j-1] + 1;

// f[i][j] = f[i+1][j] + 1;

// for(int k = i; k <= j; k++)

for(int k = i; k <= j; k++)

if(a[k] == a[j]) f[i][j] = min(f[i][j], f[i][k] + f[k+1][j-1]);

// if(a[k] == a[i]) f[i][j] = min(f[i+1][k-1]+f[k][j], f[i][j]); //TODO

}

printf("Case %d: %d\n", ++Cas, f[1][n]);

}

return 0;

}

/\*

区间DP 好题

题意：跟经典的关路灯一样

思路：因为这题之前做了hdu\_4283，“悟”除了区间DP的状态f[i,j]要保证满足局部独立性，然后做这题的时候一眼定义了f[i,j]表示关掉区间[i,j]的路灯花费最少的电费，而且严格控制是[i,j]这个区间，跟外面的灯无关。 然后接下来就不知道怎么转移了...因为历时时间无法计算也无法记录。

然后还是看了题解。发现又是自己的f[i,j]状态的含义定义不合适...=\_=

正解的f[i,j]状态定义为，关掉[i,j]这个区间的灯时，全局已经累积了多少电费。

这个状态的定义对“历时”比较好解决，如果当前人类关掉i灯后走向j灯，则全局电费增量为dis(i,j) \* sum\_of\_rest\_lights\_power，就是距离乘上剩下的灯的功率和。

状态方程就好解决了，略.

摘抄：先说下本题的模型，送餐的顺序是一个从x位置开始的从1...n的一个全排列，我们要做的是找一个全排列它的总坑爹值最小。

from http://blog.csdn.net/woshi250hua/article/details/7768744

\*/

// zoj 3469

int work(int s)

{

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

for(int j = 1; j <= n; j++) f[0][i][j] = f[1][i][j] = INF;

f[0][s][s] = f[1][s][s] = 0;

for(int i = s; i > 0; i--)

for(int j = s; j <= n; j++) if(i != j) { //TODO

int tmp = (Sum(1, i-1) + Sum(j+1, n));

f[0][i][j] = min(f[0][i][j], f[0][i+1][j] + (a[i+1].x - a[i].x) \* (tmp + a[i].b));

f[0][i][j] = min(f[0][i][j], f[1][i+1][j] + (a[j].x - a[i].x) \* (tmp + a[i].b));

f[1][i][j] = min(f[1][i][j], f[0][i][j-1] + (a[j].x - a[i].x) \* (tmp + a[j].b));

f[1][i][j] = min(f[1][i][j], f[1][i][j-1] + (a[j].x - a[j-1].x) \* (tmp + a[j].b));

}

return min(f[0][1][n], f[1][1][n]);

}

int main()

{

int V, X;

while(scanf("%d%d%d", &n, &V, &X) != EOF) { //V分钟跑1m，那就以V为单位

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

scanf("%d%d", &a[i].x, &a[i].b);

//a[i].x \*= V;

}

//X \*= V;

a[++n] = Person(X, 0);

sort(a+1, a+n+1, cmp);

for(int i = 1; i <= n; i++) sum[i] = sum[i-1] + a[i].b;

for(int i = 1; i <= n; i++) if(a[i].x == X && a[i].b == 0) {

printf("%d\n", V \* work(i)); //留到后面乘，之前在前面乘因为溢出WA了两次

break;

}

}

return 0;

}

/\*

区间DP 好题

题意：给定一个序列，序列内的人有屌丝值Di，是第然后将这个序列进栈，第i个人如果是第k个出栈，那么最后的屌丝总值增加Di \* (k-1), 求一个出栈序列使得总屌丝值最小。

思路：做完上一道hdu\_2476区间DP，然后还是想要那个思路来做这道题，就先预处理出区间[i,j]顺序放和逆序放的区间总愤怒值，然后再开一个DP计算全局最优值。我的区间f[i,j]是相对于全局(即n个人)的愤怒值。

然后，然后就跪了。因为计算的时候不知道怎么转移。

正解的f[i,j]定义：[i,j]区间的最小花费，只考虑[i,j]区间，跟全局无关。

这样定义，然后跟全局的关系在后面的DP计算时再考虑进去，这样思路就很明晰。

假设区间[i,j]里，第i个人是第k个进场的，即在i+1之后的k-1个人率先进场，则

区间[i,j]花费 = 在i号之前进场的([i+1 ~ i+1+k-1-1]) + i号第k个进场 + 在i号之后进场的([k+1 ~ j])

= f[i+1, i+1+k-1-1] + D[i]\*(k-i) + f[i+1, j] + (k-i+1)\*({k+1..j}的ds脾气和)

还是挺有收获的，我原来思考状态的定义时习惯考虑“搞完区间(i,j)的花费”，但是现在做了两题区间DP都是定义为“搞完区间[i,j]时全局的花费”，其实就等于“搞区间[i,j]之前的全局花费 + 搞区间[i,j]的全局花费增量”。

之前YY出了个“局部独立性”...我觉得这个概念可以帮助思考dp状态以及状态转移，就是推测状态的定义时，最好满足“局部独立性”，不然转移时会比较麻烦。

\*/

// hdu 4283

int main()

{

int cases, Cas = 0;

scanf("%d", &cases);

while(cases--) {

scanf("%d", &n);

for(int i = 1; i <= n; i++) scanf("%d", &d[i]);

for(int i = 1; i <= n; i++) sum[i] = sum[i-1] + d[i];

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

//f[i][i] = 0;

//for(int j = i+1; j <= n; j++) f[i][j] = INF;

}

for(int l = 1; l <= n; l++)

for(int i = 1; i + l - 1 <= n; i++) {

int j = i + l - 1;

f[i][j] = f[i][j-1] + d[j] \* (j - 1);

for(int k = i; k <= j; k++)

f[i][j] = min(f[i][j], f[i+1][k] + f[k+1][j] + d[i]\*(k-i) + (k-i+1) \* (sum[j]-sum[k]));

}

printf("Case #%d: %d\n", ++Cas, f[1][n]);

}

return 0;

}

/\*

区间DP预处理 好题

题意：一步可以改变A串的一段为同一个字母，问用最少步数把A串变成B串。

思路：代码里面细节用 //TODO 注释了

\*/

//hdu 2476

int main()

{

A[0] = B[0] = '-';

while(scanf("%s%s", A+1, B+1) == 2) {

int len = strlen(A);

memset(f, 0, sizeof(f));

for(int i = 1; i < len; i++) //初始化简单，可以初始化为INF, 但千万不能用memset，因为后面对于i>j的情况需要等于0

for(int j = i; j < len; j++)

f[i][j] = j - i + 1;

for(int l = 1; l < len; l++)

for(int i = 1; i + l - 1< len; i++) {

int j = i + l - 1;

for(int k = i+1; k <= j; k++) //TODO k是小于等于j

if(B[i] == B[k]) //TODO 是用B串预处理

f[i][j] = min(f[i][j], f[i][k-1] + f[k+1][j]);

else f[i][j] = min(f[i][j], f[i+1][j] + 1); //TODO

}

memset(ans, 0, sizeof(ans));

for(int i = 1; i < len; i++) {

ans[i] = f[1][i]; //TODO

if(A[i] == B[i]) ans[i] = ans[i-1];

for(int k = 1; k < i; k++)

ans[i] = min(ans[i], ans[k] + f[k+1][i]);

}

printf("%d\n", ans[len-1]);

}

return 0;

}

/\*

题意：给你n个数，A、B两个人从两端取数，每一次可以取连续的一段数，而且每个人都按让自己最优的方案去取数（即取的数的和尽可能大）

问最后A与B的差值最大是多少

思路：有一道区间两端取数的初始版本：USACO / A Game http://www.cnblogs.com/AbandonZHANG/archive/2012/08/03/2620838.html

初始版本"a game"是每次可以取两端的一个数字，本题是每次可以取两端连续的一段数字。我是看了前者的解析才理解后者的题解的=\_=

先来讲初始版本的：f[i, j]表先手面对这样一个序列可以保证取道的最大值。

f[i, j] = sum(i,j) - min{f[i+1, j], f[i, j-1]}; //面对区间[i,j]，我们有两种选择，然后轮到对手，对手也会采取最优策略拿到f[i+1, j]或者f[i, j-1]的值，所以我们只需要使得区间[i,j]被对手拿到最少即可。这有点像“极大极小搜索”的想法，将自己的所得极大化相当于将对手的所得极小化。

选了一个后转化成的子问题，第二个人是先选，所以第一个人只能拿到子问题的后选的人的解，即sum[i][j]-f[i][j]。

本题其实就是扩展一下嘛，可以仿照a game的写出方程，对于一段区间，枚举分隔点，要么取左边，要么取右边，对应于:

f[i][j] = max(f[i][j], Sum(i, k) - f[k+1][j]);

f[i][j] = max(f[i][j], Sum(k, j) - f[i][k-1]);

这是道博弈题，双方都采取最优策略，所以对于区间(l,r)，如果先手A取(l..k)，则就转化成了子问题，就是后手B对于区间(k+1,r)最大为多少，注意，这个时候的区间(k+1,r)，先手是B！所以这就是方程得以成立的巧妙之处哈~ diff = sum(l..k) - f[k+1, r].

因为对于每个区间，先手是轮换着来的，而f[i,j]表先手能取到的最大值，就很好地适合“先手是轮换着的”.

\*/

//lightoj 1031

int main()

{

int Cas = 0, cases;

scanf("%d", &cases);

while(cases--) {

scanf("%d", &n);

for(int i = 1; i <= n; i++) scanf("%d", &sum[i]), sum[i] += sum[i-1];

for(int l = 1; l <= n; l++)

for(int i = 1; i + l - 1 <= n; i++) {

int j = i + l - 1;

f[i][j] = Sum(i, j);

for(int k = i; k <= j; k++) {

f[i][j] = max(f[i][j], Sum(i, k) - f[k+1][j]);

f[i][j] = max(f[i][j], Sum(k, j) - f[i][k-1]);

}

}

printf("Case %d: %d\n", ++Cas, f[1][n]);

}

return 0;

}