# 云某人的dp



## 背包

### 01背包

#### 表示容量，存储价值

```
for(int i=1;i<=m;++i)// 层数
{
    for(int j=t;j>=t[i];--j)// 容量
    {
        dp[j] = max(dp[j],dp[j-t[i]]+v[i]);
    }
}
```

#### 表示价值，存储容量

```
for(int i=1;i<=m;++i)// 层数
{
    for(int j=V;j>=t[i];--j)// 价值
    {
        if(dp[j-v[i]] + w[i] <= t)  dp[j] = min(dp[j],dp[j-v[i]]+w[i]);
    }
}
```

## 完全背包

```
for(int i=1;i<=m;++i)// 层数
{
    for(int j=t[i];j<=t;++j)// 容量
    {
        dp[j] = max(dp[j],dp[j-t[i]]+v[i]);
    }
}
```

# 区间dp

## 朴素区间dp

$O(n^3)$
$dp_{i,j}$ 表示从 $l$ 到 $r$ 的区间
每次枚举区间长度 - 起始点 - 间断点

> 枚举区间端点会比枚举起点+长度好实现一些

```
for(int i=2;i<=n;++i)
    {
        for(int j=1;j+i-1<=n;++j)
        {
            for(int k=j;k<j+i-1;++k)
            {
                dp[j][j+i-1] = min(dp[j][j+i-1],dp[j][k]+dp[k+1][j+i-1]+x);
            }
        }
    }
```

# 数据结构优化dp

## RMQ类

比如每次需要从前面最大/最小的数中转移而来，而 `dp` 值是动态的，就需要借助一些维护动态RMQ的数据结构，比如线段树/树状数组(只能维护$1 \sim x$的)

题目B
子序列/子串问题
LCS - 最长公共子序列

# 状态dp

其实这里记录的应该叫**状压dp**
但是状压通常指传统的二进制压缩，但这里有些类型无二进制压缩，而只是依托于其他状态转移。

## 状压dp

- `__builtin_popcount(x)` 二进制 $1$ 的个数，O(1)

## 子集dp

`sos_dp`
一个集合的状态，由其所以子集转移而来，为了防止重复记录，一般每次只新增一个元素。

## 多维状态DP

`dp` 多维表示为题目的条件/状态

### 题目A

在 $n$ 层的楼中，Pak Chanek可以操作三台电梯，他将在接下来的 $Q$ 天里工作。一开始，所有电梯都位于第 1 层，且均为开启状态。在每一天，恰好会发生以下两种事件之一：

- `1 x y`：当前在第 $x$ 层有一位乘客想要前往第 $y$ 层。（$1 \le x, y \le N$，且 $x \ne y$）
- `2 p`：在当天开始时，编号为 $p$ 的电梯状态发生切换：若之前是开启状态，则变为关闭状态；若之前是关闭状态，则变为开启状态。（$1 \le p \le 3$）

对于每一天，Pak Chanek 可以随意控制开启状态的电梯的移动。但在有乘客从 $x$ 层想要前往 $y$ 层的这一天，必须满足以下操作流程：

1. 有一部电梯移动至第 $x$ 层；
2. 乘客进入电梯；
3. 该电梯移动至第 $y$ 层；
4. 乘客从电梯中出来。

Pak Chanek 只能操作当前处于开启状态的电梯。注意，由于状态变更发生在一天的开始，这意味着当天被关闭的电梯从该天起就无法使用；而当天被开启的电梯从该天起即可使用。

已知每一天的电费不同。更具体地说，在第 $j$ 天，将一部电梯上下移动一个楼层所需的电费为 $A\_j$。

一开始，Pak Chanek 就知道未来 $Q$ 天中每一天的电费和事件序列，因此他可以提前规划电梯的操作方式。请你帮助他计算：在满足所有乘客需求的前提下，最小的电费总和是多少？

注：最终每部电梯不必返回第 1 层。
**sloution:**
$dp[d][i][j][k]$ 表示电梯分别在第几天被使用。
因为$N \le 10^5, Q \le 300$，而每天的操作是已知的，那么我们就可以根据电梯使用的日期，得知其停留在哪个位置。

设当前是 $p$ 天
$dp[d+1][p][j][k] \leftarrow dp[d][i][j][k]+x$
代表使用 $i$ 的电梯，那么对于$i,j,k$，共有三种后继状态。

我们可以发现，第一维完全没有必要，甚至不需要滚动数组。因为在第 $p$ 天，必然会从 $i,j,k=p-1$ 的其中一个状态转移。

那么我们可以把 $d$ 融入后三维。
使用电梯 $i$ 的转移如下：
$dp[d][j][k] \leftarrow dp[d-1][j][k]$
$dp[d][d-1][k] \leftarrow dp[i][d-1][k]$
$dp[d][j][d-1] \leftarrow dp[i][j][d-1]$

也就是 上一天 $\times$ 这一天

那么对于另外两个电梯，也是三种，总共九条转移。

复杂度$O(Q^3)$

新增代价就不做赘述了，随便用个数组存下来，复杂度 $\leq Q^3$ 即可。

**code:**

```
struct node
{
    int op,l,r,v;
}q[N];

int n,m;
int a[3][N], dp[N][N][N], mn[3][N][N];
int use[3] = {0,0,0};

void qmin(int &a,int b)
{
    a = min(a,b);
}

void func(void)
{
    cin >> n >> m;
    for(int i=1;i<=m;++i)
    {
        cin >> a[0][i];
        a[1][i] = a[2][i] = a[0][i];
    }
    for(int i=1;i<=m;++i)
    {
        cin >> q[i].op;
        if(q[i].op == 1)
        {
            cin >> q[i].l >> q[i].r;
            q[i].v = abs(q[i].r-q[i].l);
        }
        else
        {
            cin >>  q[i].v;
            use[q[i].v-1] ^= 1;
```

```
        }
        for(int j=0;j<3;++j)
        {
            if(use[j])  a[j][i] = inf;
        }
    }
    a[0][0] = a[1][0] = a[2][0] = inf;
    for(int i=0;i<3;++i)
    {
        for(int j=0;j<=m;++j)
        {
            int tmp = inf;
            for(int k=j;k<=m;++k)
            {
                tmp = min(tmp,a[i][k]);
                mn[i][j][k] = tmp;
            }
        }
    }
    memset(dp,0x3f,sizeof dp);
    int p = 0;
    dp[0][0][0] = 0;
    q[0] = {1,1,1,0};
    for(int i=1;i<=m;++i)
    {
        if(q[i].op == 2)    continue;
        for(int j=0;j<i;++j)
        {
            for(int k=0;k<i;++k)
            {
                if(q[j].op == 2 || q[k].op == 2)    continue;
                // p j k
                qmin(dp[i][j][k],dp[p][j][k]+abs(q[i].l-q[p].r)*mn[0][p]
[i]+q[i].v*a[0][i]);
                qmin(dp[p][i][k],dp[p][j][k]+abs(q[i].l-q[j].r)*mn[1][j]
[i]+q[i].v*a[1][i]);
                qmin(dp[p][j][i],dp[p][j][k]+abs(q[i].l-q[k].r)*mn[2][k]
[i]+q[i].v*a[2][i]);
                // j k p
                qmin(dp[i][k][p],dp[j][k][p]+abs(q[i].l-q[j].r)*mn[0][j]
[i]+q[i].v*a[0][i]);
                qmin(dp[j][i][p],dp[j][k][p]+abs(q[i].l-q[k].r)*mn[1][k]
[i]+q[i].v*a[1][i]);
                qmin(dp[j][k][i],dp[j][k][p]+abs(q[i].l-q[p].r)*mn[2][p]
[i]+q[i].v*a[2][i]);
                // j p k
                qmin(dp[i][p][k],dp[j][p][k]+abs(q[i].l-q[j].r)*mn[0][j]
[i]+q[i].v*a[0][i]);
                qmin(dp[j][i][k],dp[j][p][k]+abs(q[i].l-q[p].r)*mn[1][p]
[i]+q[i].v*a[1][i]);
                qmin(dp[j][p][i],dp[j][p][k]+abs(q[i].l-q[k].r)*mn[2][k]
[i]+q[i].v*a[2][i]);
            }
        }
        p = i;
    }
```

```
    int ans = 1e18;
    for(int i=0;i<=p;++i)
    {
        for(int j=0;j<=p;++j)
        {
            for(int k=0;k<=p;++k)
            {
                if(i == p || j == p || k == p)  ans = min(ans,dp[i][j][k]);
            }
        }
    }
    cout << ans << '\n';
}
```

## 题目B

上一题的升级版。
新增了数组维度的平移，使得九个状态不用一起列出来，而可以用循环计算。

P9676 [ICPC 2022 Jinan R] Skills
庞博士有 $3$ 项技能：喝汽水、猎狐和炒股，编号分别为 $1, 2, 3$。初始时，每项技能的熟练度为 $0$。

接下来有 $n$ 天。在第 $i$ 天，庞博士可以选择一项技能（假设是第 $j$ 项）进行练习，然后在这天结束时让这项技能的熟练度增加 $a_{i,j}(0 \leq a_{i,j} \leq 10000)$。同时，如果某一项技能（假设是第 $k$ 项）已经有 $x$ 天没有练习，那么在这天结束时，这项技能的熟练度会减少 $x$。当然，任何一项技能的熟练度都不可能小于 $0$。

现在，庞博士想知道：在第 $n$ 天结束后，这 $3$ 项技能的熟练度之和最大为多少。由于他非常忙，而且他的日程和对习惯的适应程度可能有变，所以庞博士把这 $T$ 个问题交给了你——每个问题的内容都一样，只是给出的数据可能有所不同而已。

**sloution**
$dp[d][i][j][k]$ 表示在第 $d$ 天，$i, j, k$ 技能分别有多少天没有练习。

那么我们可以枚举时间，然后枚举当前练习的技能，后枚举可以转移到哪天（每个状态只有一个后继，但有若干个前驱）。
**对于练习第 $i$ 天**
$dp[d+1][0][j+1][k+1] \leftarrow dp[d][0][j][k]$
$dp[d+1][0][1][k+1] \leftarrow dp[d][i][0][k]$
$dp[d+1][0][j+1][1] \leftarrow dp[d][i][j][0]$

这样很明显会超时，所以思考优化。

首先是 `T()` 方面:
$n \times n^3$ 绝对不可能接受。
但是我们计算发现，未练习的代价是 $n^2$ 级别，在一个技能超过 $2\sqrt{a_{\max}} = 200$ 未练习，不如一开始就不练习，那么 $i, j, k \leq 200$。
因为我们按时间转移，每次必然从 $i, j, k = 0$ 之一转移，那么对于每个技能的计算，只有两个维度$\neq 1$
。
那么复杂度降低为 $O(n \times a_{\max})$

然后是 `M()` 方面
$n \times n^3$ 绝对不可能接受。
因为每次都从前一天的状态转移，那么我们用滚动数组优化，第一维降为 $2$
又因为每次的转移必然从 $i, j, k = 0$ 之一转移，那么我们可以对数组进行位移，让第二个维度表示为当

前联系的技能，这样第二维降为 $3$
那么复杂度降低为 $O(\max(n, a_{\max}))$

因为进行了数组位移，那么转移方程和过程都要变化。
每次转移将每个练习的第 $1, 2, 3$ 中操作整合到一起，具体可以看下列代码。

### code

```
const int N = 1e3+3;
const int L = 210;

int n,op;
int a[N][3];
int dp[2][3][L][L];

void func(void)
{
    cin >> n;
    for(int i=1;i<=n;++i)
    {
        for(int j=0;j<3;++j)    cin >> a[i][j];
    }
    memset(dp,0,sizeof dp);
    for(int i=1;i<=n;++i)
    {
        op ^= 1;
        for(int j=0;j<3;++j)
        {
            for(int x=0;x<L&&x<=i;++x)
            {
                for(int y=0;y<L&&y<=i;++y)
                {
                    int dx = (x ? x+1 : 0), dy = (y ? y+1 : 0);
                    int d1 = (j+1)%3, d2 = (j+2)%3;
                    if(dx < L && dy < L)    dp[op][j][dx][dy] = max(dp[op][j][x][y],dp[op^1][j][x][y]-dx-dy+a[i][j]);
                    if(dy < L)  dp[op][d1][dy][1] = max(dp[op][d1][dy][1],dp[op^1][j][x][y]-1-dy+a[i][d1]);
                    if(dx < L)  dp[op][d2][1][dx] = max(dp[op][d2][1][dx],dp[op^1][j][x][y]-dx-1+a[i][d2]);
                }
            }
        }
    }
    int ans = 0;
    for(int i=0;i<3;++i)
    {
        for(int x=0;x<L;++x)
        {
            for(int y=0;y<L;++y)    ans = max(ans,dp[op][i][x][y]);
        }
    }
    cout << ans << '\n';
}
```

# 子序列/子串问题

| 子序列 | 子串 |
| --- | --- |
| 不连续 | 连续 |

## LCS - 最长公共子序列

求长度正反皆可，但如果需要还原，或许反向更好，因为需要反向溯源求串。

```
// 求长度部分
int dp[N][N];
for(int i=L1;i>=1;--i)
{
    for(int j=L2;j>=1;--j)
    {
        // 如果两个字符相同，则可以尝试让串长度 +1
        if(s1[i] == s2[j])  dp[i][j] = dp[i+1][j+1] + 1;
        // 不同则继承上一个串
        dp[i][j] = max({dp[i][j],dp[i+1][j],dp[i][j+1]});
    }
}
// 反向溯源
int i=1,j=1;
string ans;
while(i <= L1 && j <= L2)
{
    // 如果两个字符相同，则加入答案
    if(s1[i] == s2[j])
    {
        ans += s1[i];
        i ++, j ++;
    }
    // 否则回归较大的数
    else (dp[i+1][j] > dp[i][j+1] ? i ++ : j ++);
}
```