DP板子

背包

01背包

“只使用了前i个物品的情况下容量为j的背包所能达到的最大价值”

```
dp[curv] = max(dp[curv], dp[curv - curw]+vali)
遍历方式:
    物品 0->n
    体积 v->wi(若反过来，则可能使用同一个物品多次)
```

完全背包

“只使用一定数量前i个物品，容量为j的背包所能达到的最大价值”

```
dp[volume - k*wi] = max(dp[volume] + gain, self)
遍历方式:
    物品 0->n
    体积 0->v
    while EnougthSpace k->∞
```

多重背包

二进制分组后转为01背包

二进制分组

```
int item_count = 0;
int item[MAXN*32];

void divide(int num,llint value)
{
    int k = 1;
    for(int i=0;i<32;i++)
    {
        if(k&num)
            item[item_count++] = value*k;

        k = k<<1;
    }
}
```

构造答案

记录转移到容量v的物品，可以从最大向前构造出答案。

对于多重背包在二进制分组时就需要同时记录一捆里面的物品数量与标号

其他背包

多维价值：在状态里面加入更多维

混合背包：因为“容量为j的背包所能达到的最大价值”这个定义互通，不同背包问题的dp数组实际上可以混用，只要根据物品类型选择合适的转移方程即可

并且有一个取巧的做法是将完全背包设定为一个很大的多重背包

分组背包：每组产生冲突，遍历方式类似完全背包，只不过这回选择组内物品而非物品个数

```
for volume
for group
for item
dp[volume + ifEnougthSpace(w[item])] = max(self,dp[volume] + gain)
```

有依赖背包：把相互依赖的物体捆绑为一个物体

区间DP

构造最优解的方法是使用一个辅助表记录分割点信息

流水线调度

dp数组存第i级最短时间

构造最优解的方法是记录上一级来源

最长子序列

最长的xx子序列因为其长度关于位置单调变化，故在朴素n^2算法以外存在使用二分查找的nlogn算法

最长上升子序列

状态：

```
长度为i的子序列最小的结尾（nlogn）；初值置inf
以第i个元素结尾的子序列的最大长度（n²）
```

转移方程：

```
i 0->maxlen : dp[lb(data[i])] = data[i]   (nlogn)
i 0->maxlen :                             (n²)(需要记录全局dp最大的下标)
    j 0->i-1 :
        if(data[i]<data[j])
            dp[i] = max(self,dp[j]+1)
```

只有n^2的做法可以较为方便地统计方案数、回溯子序列:

```
在max更新有效时记录j，由此直到子序列的结尾就能回溯到开头
在dp处增添加法计数可以获得最优方案的总方案数。
    等效可加的条件是dp[i]-1 == dp[j]&&data[i]<data[j]
    置零的条件是更新成功 dp[i]-1<dp[j]
统计方案数时需要根据最大长度反向遍历，输出所有方案数，
或是在原数组末尾放一个无穷大的元素统计其方案数
```

最长公共子序列

状态:

```
以i，j结尾的子串的最长公共子序列长度（n²）
在特殊情形（A、B均为一个序列的排列）时有nlogn解法
    AToB记录B每个元素在A中位置（输入B时按照值记录下标，再将a[i]代入B即可求得AToB[i]）
    for1:b[value]=i; for2:a_to_b[i]=b[a[i]];
AtoB的最长上升子序列长度就是A、B最长公共子序列的长度
```

转移方程:

```
能更新就更新，不能更新就继承
if(a[i]==b[j])
    dp[i][j] = max(self,dp[i-1][j-1]+1);
else
    dp[i][j] = max(dp[i-1][j],dp[i][j-1])
```

统计方案数、回溯子序列:

```
统计方案数仍是在等于且max成立时+=，不等于时继承，max更新时置1
```

编辑距离

为了防止出界，字符串从下标1开始有效

dp状态定义为：从子串Ai->Bj的编辑距离（所需要的最少改动）

初始化 dp[i][0] = dp[0][i] = i;其余inf

转移： a[i]==b[j] dp[i][j] = min(dp[i][j],dp[i-1][j-1]); 否则dp[i][j] = min(self,dp[i-1][j-1] + 1,dp[i-1][j] + 1,dp[i][j-1] + 1) 分别对应修改、删除、插入（插入A等效于删除B）

遍历过程是下标i、j循环