动态规划

思路

- 将问题分为若干个子问题，这些子问题的解决之间有所重叠，即规模较大的子问题能够通过规模较小的子问题的解容易得到；
- 通过记录子问题的解，期待将指数级别的时间复杂度降到多项式时间；
- 一种思路是：OPT 选择当前的点意味着什么；
- **注意如何初始化数组；**
- **解决多维 DP 问题时，注意循环顺序。**

问题

1. Weighted Interval Scheduling

问题描述: 一组课程，有开始时间、结束时间和权重，找出一种安排方案使得安排下的课程权重最大

解决方法:

1. 定义 `p[j]` = job i 和 j 相容时最大的 i，`OPT[j]` = 把课程范围缩小到 1 ~ j 时的解;
2. 第一种情况: `OPT` 选择了第 `j` 个 job，此时不能选择 `p[j]` 后的所有 job，即 `OPT[p[j]]`;
3. 第二种情况: `OPT` 没有选择第 `j` 个 job，则解维持在 `OPT[j - 1]`.

则递推公式为:

```
OPT[j] = if (j == 0) {
  0
} else {
  max(vj + OPT[p[j]], OPT[j - 1])
}
```

2. Segmented Least Squares

这个，先略过吧... TODO

3. 背包问题及其变种

0-1 背包问题

问题描述: 有 `n` 个物品和 `1` 个背包，每个物体有重量 `wi` 和价值 `vi`，背包的容量是 `W`，尽可能装满背包，达到最大的价值。

特点: 每个物品只能有放或者不放两种选择。

解决方法: 与问题 1 类似，同样考虑放不放第 i 个物品：

1. 定义 `OPT[i][w]` = 在只有物品 `1 ~ i`、背包容量为 `w` 的情况下的最大价值;
2. 第一种情况: `OPT` 没有选择第 `i` 个物品，则解为 `OPT[i - 1][w]`;

3. 第二种情况: `OPT` 选择了第 `i` 个物品，则意味着 `1 ~ i - 1` 个物品只能被放入容量为 `W - wi` 的背包里，即解为 `OPT[i - 1][W - wi]`.

则递推公式为:

```
OPT[i][W] = if (i == 0) {
  0
} else if (wi > W) {
  OPT[i - 1][W] // 背包根本装不下第 i 个物品
} else {
  max(OPT[i - 1][W], vi + OPT[i - 1][W - wi])
}
```

优化: 可以将二维表转化为一维数组:

```
for (i in 1..N) {
  for (w in W..0) {
    f[w] = max(f[w], f[w - w[i]] + v[i])
  }
}
```

完全背包问题

特点: 每个物品可以不放，也可以放任意多个。

解决方法: 递推公式为:

```
OPT[i][w] = max(OPT[i - 1][w], v[i] + OPT[i][W - w[i])
```

```
for (i in 1..N) {
  for (w in 0..W) {
    f[w] = max(f[w], f[w - w[i]] + v[i])
  }
}
```

如何初始化 f

- 如果不需要装满背包，则 `f[1..N] = 0`;
- 否则，`f[1] = 0`，`f[2..N] = -INF`.

RNA 配对

问题描述: 一条数轴上有 `N` 个点，点的取值是 A, C, U, G，它们中有一些可以配对（至于谁和谁配对？哦，那是生物学的事情，而我的生物很烂）；还需要满足如下两个条件:

1. 如果 `i` 和 `j` 配对，则必须有 `i <; j - 4`;
2. 两个配对间不能有交叉，即如果 `i` 和 `j` 配对，`m` 和 `n` 配对，不能有 `i <; m <; j <; n`.

求最大配对个数。

解决方法:

1. 定义 `OPT[i][j]` = 区间 `[i, j]` 中最大配对个数;
2. 第一种情况: `i >;= j - 4`， 则 `OPT[i][j] = 0`;
3. 第二种情况: 如果 j 不是一个配对中的一个端点，则 `OPT[i][j] = OPT[i][j - 1]`;
4. 第三种情况: j 和 某个符合 `i <;= t <; j - 4` 的 t 配对了，因此遍历 t ，计算 `max(OPT[i][t - 1] + OPT[t + 1][j - 1]`，再加上 1，就是 `OPT[i][j]`.

序列对齐

问题描述: 给定两个字符串 `x1x2...xm` 和 `y1y2...yn`，找出最小的对齐代价。在对齐字符串的时候，x1 到 xm 以及 y1 到 ym 的顺序不能发生变化，但可以有空格。

例子: `CTGACCTACG` 和 `CTGCACGAACG` 两个字符串，可以对齐为 ("+" 表示空格):

C T **+** G A C **C T** A C G

C T **G** G A C **G A** A C G

则对齐代价是 `空格代价 b + CG 错误代价 + TA 错误代价`.

解决方法:

1. 定义 `OPT[i][j]` = 对齐字符串 `x1x2...xi` 和 `y1y2...yj` 的最小代价;
2. 第一种情况: `OPT` 匹配了 `xi` 和 `yj`，则最小代价是 `xi-yj` 错误代价(如果相等，则这个代价为 0) 加上 `OPT[i - 1][j - 1]`;
3. 第二种情况: `OPT` 不匹配 `xi`，也就是说选择使用空格和 `yj` 进行匹配，则最小代价是 `b + OPT[i - 1][j]`;
4. 第三种情况: `OPT` 不匹配 `yj`，也就是说选择使用空格和 `xi` 进行匹配，则最小代价是 `b + OPT[i][j - 1]`;
5. 此外，对于 `i` 和 `j` 为 0 的情况来说，`OPT[i][j]` 分别为 `b * j` 和 `b * i`.

优化: 可以将此问题转化为图的最短路径问题，详情请见 PPT~ (06-DP-II, P12)

其它问题

- Bellman-Ford 算法找有负权重的最短路;
- 采用 Bellman-Ford 算法思想找出负环;
- ......