更正

- DP作业题“高速公路”，转移方程中 k 的取值范围左端点应为 $\max\{1, i-j-1\}$ 而非 $\max\{0, i-j-1\}$。每次转移需要花的时间为 $O(j)$（可以理解为 $O(K)$）而非 $O(i)$。补充了对于base case的更详细的描述。
- DP作业题“DNA序列”，状态定义的 S 和 Q 写反了；三种需要添加gap的情况中，第三种“在 S_i 和 Q_j 后面各添加一个gap”是无意义的，不需要考虑。
- Bellman-Ford的转移方程 $\min$ 打成了 $\max$，已更正。
- 增加了一些有关最短路的内容，多数来自于作业和quiz。
- 附上了A*的复习提纲。

期末复习

- 最小生成树
- 动态规划
- 最短路

最小生成树 Minimum Spanning Trees

定义

- 无向连通图 $G = (V, E)$，边权 $w : E \to \mathbb{R}$
- 选出 $|V| - 1$ 条边 $T \subseteq E$，连通所有点，边权和最小
- 图不连通？
- 有自环？
- 有重边？

定义

- 无向连通图 $G = (V, E)$，边权 $w : E \to \mathbb{R}$
- 选出 $|V| - 1$ 条边 $T \subseteq E$，连通所有点，边权和最小
- 图不连通？最小生成森林
- 有自环？自环不会在最小生成树上
- 有重边？重边只选最轻的那条

性质

- Cut property: 对于结点集 V 的任何一个划分 $V_1 \cup V_2 = V$，令 $C \subseteq E$ 为所有一端在 V_1、一端在 V_2 的边的集合。假如 $e \in C$ 是 C 中边权**严格最小**的边，则 e 必然在最小生成树上。
  - 最小生成树需要连通整个图，必然要连通 V_1 和 V_2，其中最轻的边必然入选。
  - 如果 C 中有多条最轻边，其中至少有一条入选。
- Cycle property: 一个环 C 上如果有一条边 $e \in C$ 的边权**严格大于**环上其它所有边，e 必然不在任何最小生成树上。
  - 选 e 是不值的，可以用其它边替代。
  - 如果 C 中有多条最重边，就不好说了。

算法: Kruskal

```
T = empty set
for each vertex v in V
    MAKE_SET(v)
sort the edges of E into nondecreasing order by weight w
for each edge (u, v) in E, taken in nondecreasing order by weight
    if FIND_SET(u) != FIND_SET(v)
        T = T + {(u, v)}
        UNION(u, v)
return T
```

- 按边权从小到大枚举每条边，如果选它不构成环，就选，否则跳过
- 需要并查集来快速查询是否成环
- 时间复杂度一般认为是 $O\left(E\log E+E\alpha(V)\right)=O(E\log E)$，瓶颈在于**排序**。
    - 写成 $O(E\log V)$ 也对。

算法: Kruskal

- 按边权从小到大枚举每条边，如果选它不构成环，就选，否则跳过
- 需要并查集来快速查询是否成环
- 时间复杂度一般认为是 $O\left(E\log E+E\alpha(V)\right)=O(E\log E)$，瓶颈在于**排序**。
  - 写成 $O(E\log V)$ 也对。
- 自环会被并查集自动踢掉
- 重边也能正确处理，因为我们按边权从小到大枚举。
- 如果图不连通，将会返回一个最小生成森林。

算法: Prim

- 从一个起始点出发，每次连通一个新的结点，不断扩张，直到连通整个图
- 每次考察那些与当前的树只有一条边之隔的结点，选这些点中最近的那个
- 用一个数组 `dist[x]` 记录这个信息（如果与树相隔不止一条边就直接视为 ∞），每次找 `dist[x]` 最小的结点，将它加入树，并用它更新它邻居的 `dist` 值。
- 时间复杂度？

算法: Prim

时间复杂度？

- 邻接矩阵：妥妥的 $O\left(V^2\right)$（枚举邻居都需要 $O(V)$）
  - 但这不一定是最坏的
- 邻接表：$O\left(V^2+E\right)$，因为你可以 $O(\deg v)$ 地枚举 v 的邻居，所以更新邻居这一块的总复杂度是 $O\left(\sum_v \deg v\right)=O(E)$。
- 邻接表+优先队列优化：用优先队列快速找 `dist` 最小的结点，以及快速地更新。
  - 用二叉堆：找 `dist` 最小的结点需要 $O\left(\log V\right)$ 的时间，更新一个结点的 `dist` 值也需要 $O\left(\log V\right)$ 的时间。 $O\left(V\log V+E\log V\right)=O\left(E\log V\right)$。
    - 在 $E=\Theta\left(V^2\right)$ 时，这甚至比邻接矩阵更劣！
  - 用斐波那契堆：可以做到 $O(1)$ decrease-key，所以 $O(V\log V+E)$。

算法: Prim

- 有自环？无所谓，加过的点不会再加一遍
- 有重边？只看得见最短的那条
- 不连通？Prim算法只能找得出起始点所在的连通块的最小生成树。如果要求最小生成森林，需要为每个连通块都跑一遍Prim。

动态规划 Dynamic Programming

0-1 背包

一个背包载重为 V。现在有 n 件物品，第 i 件物品价值 v_i，重量为 w_i，问怎样使得装入背包的物品价值最大。

0-1 背包

“0-1”：每个物品只有“选”或者“不选”。

- $f(i,j)$ 表示将前 i 件物品中的某些物品装入载重为 j 的背包，最大价值是多少。
- 考虑第 i 件物品“选”或“不选”：
  - 选：则需要占用 w_i 的重量，我们还可以在前 $i-1$ 件物品中选出若干物品，重量不能超过 $j-w_i$。
  - 不选：相当于在前 $i-1$ 件物品中选出若干物品，重量不超过 j。
  - $$f(i,j)=\begin{cases}f(i-1,j), & j-w_i<0,\\ \max\{f(i-1,j),v_i+f(i-1,j-w_i)\}, & j-w_i\geqslant 0.\end{cases}$$
- 时间复杂度：$O(nV)$

0-1背包

$$f(i,j)=\begin{cases} f(i-1,j), & j-w_i<0,\\ \max\{f(i-1,j),v_i+f(i-1,j-w_i)\}, & j-w_i\geqslant 0. \end{cases}$$

“只要for一for就好了”

```
for (auto i = 1; i <= n; ++i)
  for (auto j = 0; j <= V; ++j) {
    f[i][j] = f[i - 1][j];
    if (v[i] + f[i - 1][j - w[i]] > f[i][j])
      f[i][j] = v[i] + f[i - 1][j - w[i]];
  }
```

完全背包

“完全”：每件物品有无穷多件。

- $f(i,j)$ 表示将前 i 件物品中的某些物品装入载重为 j 的背包，最大价值是多少。
- 第 i 件物品至多选 $\lfloor j/w_i \rfloor$ 件。
  - $\max\left\{kv_i + f(i-1, j-kw_i) \mid k = 0, \cdots, \lfloor j/w_i \rfloor\right\}$
  - 转移需要枚举 k，慢了。

完全背包

- $f(i,j)$ 表示将前 i 件物品中的某些物品装入载重为 j 的背包，最大价值是多少。
- 第 i 件物品至多选 $\lfloor j/w_i \rfloor$ 件。
  - 仍然考虑第 i 件物品“选”还是“不选”
  - 如果选，我们先放一件，占掉 w_i 的重量，然后再在前 i 个物品中选总重不超过 $j-w_i$ 的，贡献是 $v_i+f(i,j-w_i)$。注意，$f(i,j-w_i)$ 这个状态里可能也选了若干件 i，所以涵盖了所有情况。
  - 如果不选，就是 $f(i-1,j)$。
  - $$f(i,j)=\begin{cases} f(i-1,j), & j-w_i<0 \\ \max\{f(i-1,j),v_i+f(i,j-w_i)\}, & j-w_i\geqslant 0. \end{cases}$$
- 时间复杂度 $O(nV)$

“伪多项式时间” pseudo-polynomial time

- 多项式时间算法的“多项式”必须是关于**输入长度**的多项式
- $O(nV)$ 是关于输入长度和输入的**值**的多项式，不能认为是“多项式时间”。
  - 如果物品个数只有 10 个，但背包载重有 10^9，这个动态规划算法甚至还不如 $O(2^n)$ 的枚举子集。

注意事项

- 请书写**规范**、**正确**、**严谨**的数学语言。
  - 使用下标 a_i, f_{ij} 或函数 $p(i)$, $f(i,j)$，而不是 $f[i][j]$。
  - 所有记号必须**先定义后使用**。
  - 使用未声明的记号，你的得分将是undefined behavior。
- 注意交代清楚base case。

作业题：Highway Speed-Limit Sign

一条高速公路长度为 L，在 $0 = x_1 < x_2 < \cdots < x_n < L$ 处设置了限速，在 $[x_i, x_{i+1}]$ 区间的速度不得超过 v_i。移除至多 K 个限速，最小化从 0 走到 L 的时间。

和背包类似的套路：“前缀”

$f(i,j)$ 表示 $[0,x_i]$ 这一段上移除了 j 个限速标志之后（不包括 x_i 处的），通过 $[0,x_i]$ 的最短时间

- $j \leqslant i-2$
- 考虑第 $i-1$ 个限速标志“移除”还是“不移除”？

和背包类似的套路："前缀"

$f(i,j)$ 表示 $[0,x_i]$ 这一段上移除了 j 个限速标志之后（不包括 x_i 处的），通过 $[0,x_i]$ 的最短时间

- $j \leqslant i-2$
- 考虑第 $i-1$ 个限速标志"移除"还是"不移除"？
- 如果仅仅知道移除了第 $i-1$ 个标志，在 $[?,x_i]$ 这一段上的速度又是多少？

和背包类似的套路："前缀"

$f(i,j)$ 表示 $[0,x_i]$ 这一段上移除了 j 个限速标志（不包括 x_i 处的）之后，通过 $[0,x_i]$ 的最短时间（$j \leqslant i-2$）

- 考虑上一个**没有移除**的限速标志是谁
- 假设是 x_k，也就是说我们移除了 $x_{k+1},\cdots,x_{i-1}$ 处的限速标志，共 $i-k-1$ 个。
- 需要 $i-k-1 \leqslant j$，并且 $k \geqslant 1$
- 那么在 $[0,x_k]$ 这一段上的最短通过时间是 $f(k,j-(i-k-1))$，在 $[x_k,x_i]$ 上的最短通过时间是 $\dfrac{x_i-x_k}{v_k}$，所以

$$f(i,j)=\min_{\substack{1\leqslant k\leqslant i-1\\ i-k-1\leqslant j}}\left\{f(k,j-(i-k-1))+\frac{x_i-x_k}{v_k}\right\}$$

Base cases

$f(i,j)$ 仅在 $j \leqslant i-2$，且 $i \geqslant 1, 0 \leqslant j \leqslant K$ 时有意义。为了方便，其他情况下 $f(i,j) = +\infty$。

$$f(i,j) = \min_{\substack{1 \leqslant k \leqslant i-1 \\ i-k-1 \leqslant j}} \left\{ f(k, j-(i-k-1)) + \frac{x_i - x_k}{v_k} \right\}$$

- 当 $j \leqslant i-3$ 时，转移方程中的 $j-(i-k-1) \leqslant k-2$，$f(k, j-(i-k-1))$ 是合法状态。
- 但是当 $j = i-2$ 时，$f(i,j)$ 的含义是将 $(0, x_i)$ 上的所有限速标志都移除，这意味着 k 只能取 1
  - 转移方程中的 $f(k, j-(i-k-1))$ 在 $k > 1$ 时都是无穷大，没问题
  - $k = 1$ 对应的式子是 $f(1,0) + x_i / v_1$，补充定义 $f(1,0) = 0$ 后问题解决。

时间复杂度

$$f(i,j)=\begin{cases}0, & \text{if } i=1,j=0,\\ \min\limits_{\substack{1\leqslant k\\ i-k-1\leqslant j}}\left\{f(k,j-(i-k-1))+\dfrac{x_i-x_k}{v_k}\right\}, & \text{if } i\geqslant 1,0\leqslant j\leqslant K,j\leqslant i-2,\\ +\infty, & \text{otherwise.}\end{cases}$$

状态数 nK，每次转移需要花 $O(j)$（或理解为 $O(K)$）的时间枚举 k，因此总复杂度为 $O(nK^2)$。

所以，到底怎么设计状态、转移？

有很多“触及本质”的解释

- 南京大学蒋炎岩：“动态规划的状态是对搜索（枚举）的状态空间的概括”
- 钟皓曦：动态规划可以看做一个图论问题，“状态”就是图上的结点，“转移”就是图上的边，解决动态规划问题就是在这样的图上做遍历。
  - 需要按照拓扑序遍历
  - 许多简单的问题的拓扑序非常清晰，以至于可以直接 `for i = 1 to n`
  - 对于复杂的问题，需要拓扑排序、记忆化搜索，或者用图论算法等等。

但这些远远超出了CS101的范围...

而且熟练背诵任何的概念和名人名言都不能帮助你做出题。

- 理解例题
- 熟悉套路

CF908G CF908D CF891C CF786B CF785E CF689D CF613D CF600E CF487E CF375C
CF340E CF280C CF258D CF235E CF79D CF960F CF1093E CF1111E CF1111C CF1175E
CF1207G P1110 P1251 P1295 P1361 P1398 P1402 P1447 P1500 P1501 P1505
P1600 P1712 P1829 P1848 P1856 P1975 P2042 P2045 P2046 P2048 P2050 P2053
P2272 P2172 P2173 P2195 P2144 P2147 P2153 P2161 P2221 P2260 P2300 P3745
P3747 P3746 P3750 P2350 P3749 P2120 P3705 P3707 P3755 P2371 P2387 P3759
P2617 P2402 P2408 P2414 P3648 P3703 P2444 P2468 P2469 P2472 P2480 P2486
P2491 P2495 P2501 P2505 P2508 P2511 P2518 P2519 P2523 P2542 P2572 P2596
P2598 P2604 P2607 P3943 P2757 P2633 P3953 P2680 P3702 P2711 P2710 P2762
P2764 P2765 P2766 P2774 P2770 P2754 P3620 P3622 P3623 P3628 P1231 P3380
P2824 P2825 P2839 P2852 P2944 P3959 P3402 P3960 P3401 P3153 P3154 P3157
P3158 P3159 P3163 P3168 P3171 P3181 P3195 P3196 P3199 P3203 P3206 P3224
P3227 P3246 P3267 P3285 P3287 P3288 P3292 P3302 P3304 P3305 P3311 P3312
P3313 P3320 P3321 P3324 P3327 P3329 P3332 P3338 P3355 P3356 P3357 P3358
P3410 P2783 P3413 P3454 P3515 P3521 P3565 P5154 P3676 P3674 P3709 P3806
P3690 P3768 P3778 P3792 P3794 P3809 P3810 P3829 P3830 P3835 P3866 P3899
P3911 P4026 P3950 P3973 P3978 P3979 P3980 P4001 P4000 P4009 P4012 P4013
P4027 P4036 P4068 P4072 P4049 P4093 P4103 P4127 P4149 P4134 P4168 P4135
P4137 P4146 P4171 P4151 P4172 P4174 P4175 P4176 P4177 P4178 P4180 P4197
P4198 P4208 P4211 P4213 P4219 P4234 P4230 P4238 P4322 P4248 P4251 P4319
P4320 P4271 P4123 P4314 P4298 P4311 P4312 P4304 P4329 P2726 P4340 P4381
P4390 P4363 P4449 P4461 P4462 P4714 P4479 P4491 P4492 P4495 P4602 P4512
P4514 P4526 P4516 P4556 P4576 P4591 P4592 P4606 P4777 P4313 P4630 P4655
P4719 P4735 P4721 P4722 P4725 P4841 P4859 P4751 P4767 P4768 P5029 P4847
P4782 P4883 P4921 P4884 P5112 P4897 P4983 P5030 P5023 P5024 P5025 P5110
P5055 P5060 P4980 P5245 P5221 P5283 P5290 P5300 P5303 P5304 P5339 P5369
P5395 P5468 P5471 SP1557 SP10628 SP2916 SP4155 SP4487 SP6779 SP16549
SP16580 SP8791 SP9070 SP19543 UVA11082 UVA11426

NOI/NOI+/CTSC

CF896E CF809E CF70D CF923E P1587 P2081 P2179 P3704 P2605 P3238 P4240
P4091 P4249 P4364 P4382 P4383 P4463 P4466 P4546 P4619 P4827 P4899 P4931
P5050 P5284 P5295 P5305 P5331 P2791 P5401 P5470

难易度统计

暂无评定	0题
入门	17题
普及-	42题
普及/提高-	77题
普及+/提高	91题
提高+/省选-	180题
省选/NOI-	323题
NOI/NOI+/CTSC	31题

统计数据非实时更新。

作业：DNA序列

仍然是“前缀”的套路：$f(i,j)$ 表示序列 S 的前 i 项与序列 Q 的前 j 项匹配，最多得分是多少

考虑是否将 S_i 与 Q_j 匹配：

- 如果将 S_i 与 Q_j 匹配，则贡献是 $f(i-1,j-1)+Score(S_i,Q_j)$
- 如果不对齐
  - 在 S_i 后面补一个gap，让 $S_{1..i}$ 与 $Q_{1..j-1}$ 匹配，让 Q_j 与gap匹配，贡献是 $f(i,j-1)+Penalty(-,Q_j)$
  - 在 Q_j 后面补一个gap，让 $S_{1..i-1}$ 与 $Q_{1..j}$ 匹配，让 S_i 与gap匹配，贡献是 $f(i-1,j)+Penalty(S_i,-)$
  - 在 S_i 后面和 Q_j 后面都补一个gap？gap与gap对齐没有意义。

作业：DNA序列

$$f(i,j) = \begin{cases} \text{base cases} & i = 0 \text{ or } j = 0 \\ \max \begin{cases} f(i-1, j-1) + Score(S_i, Q_j) \\ f(i-1, j) + Penalty(S_i, -) \\ f(i, j-1) + Penalty(-, Q_j) \end{cases} & i, j > 0 \end{cases}$$

时间复杂度？

作业：DNA序列

$$f(i,j)=\begin{cases} j\cdot Penalty(-,Y), & i=0 \\ i\cdot Penalty(X,-), & j=0 \\ \max\begin{cases} f(i-1,j-1)+Score(S_i,Q_j) \\ f(i-1,j)+Penalty(S_i,-) \\ f(i,j-1)+Penalty(-,Q_j) \end{cases} & i,j>0 \end{cases}$$

时间复杂度？

状态数 $O(mn)$，转移 $O(1)$，所以复杂度 $O(mn)$

最短路

Bellman-Ford

设 $dist(x, i)$ 表示从起点到 x 经过**不超过** i 条边的最短路长度。

考虑"最后一条边"是谁

- 假设是 (v, x)，那么路径长度是 $dist(v, i-1) + w(v, x)$
- "**不超过**"意味着 $dist(x, i-1)$ 也要算上

$$dist(x, i) = \min\{dist(x, i-1), dist(v, i-1) + w(v, x) \mid (v, x) \in E\}$$

Bellman-Ford

$$dist(x,i) = \min\{dist(x,i-1), dist(v,i-1) + w(v,x) \mid (v,x) \in E\}$$

实现：难以枚举 v，难道需要图转置？

Bellman-Ford

$$dist(x,i) = \min\{dist(x,i-1), dist(v,i-1) + w(v,x) \mid (v,x) \in E\}$$

实现：难以枚举 v，难道需要图转置？

直接枚举所有 v，尝试用 $dist(v,i-1) + w(v,x)$ 更新 $dist(x,i)$

```
for (auto i = 1; i <= n - 1; ++i) {
  for (auto x = 1; x <= n; ++x)
    dist[x][i] = dist[x][i - 1];
  for (auto v = 1; v <= n; ++v)
    for (auto [x, w] : G[v])
      dist[x][i] = std::min(dist[x][i], dist[v][i - 1] + w);
}
```

Bellman-Ford

$$dist(x,i) = \min\{dist(x,i-1), dist(v,i-1) + w(v,x) \mid (v,x) \in E\}$$

发现 $dist(\cdot,i)$ 只依赖于 $dist(\cdot,i-1)$，所以可以滚动数组

```
for (auto i = 1; i <= n - 1; ++i) {
  for (auto x = 1; x <= n; ++x)
    dist[x][i % 2] = dist[x][(i - 1) % 2];
  for (auto v = 1; v <= n; ++v)
    for (auto [x, w] : G[v])
      dist[x][i % 2] = std::min(dist[x][i % 2], dist[v][(i - 1) % 2] + w);
}
```

空间复杂度降到了 $O(V)$。

Bellman-Ford

再经过仔细的思考和优化，就有了这样的代码

```
for (auto i = 1; i <= n - 1; ++i)
  for (auto v = 1; v <= n; ++v)
    for (auto [x, w] : G[v])
      dist[x] = std::min(dist[x], dist[v] + w);
```

Dijkstra

贪心，建立在“绕路不会有好处”的基础上：

- 假设起点 S 有三条出边 $S \to A, S \to B, S \to C$，边权分别为 $2, 5, 3$。
- 我们断言：从 S 沿边 (S, A) 直接走到 A 一定是从 S 到 A 的最短路径，从 B 或 C 绕道不可能更短。
- 边权必须非负，否则绕道有可能更短。

Dijkstra

实现和Prim非常类似，时间复杂度分析也是一样的：

- 邻接矩阵：$O\left(V^2\right)$
- 邻接表朴素：$O\left(V^2+E\right)$
- 邻接表+二叉堆：$O\left(V\log V+E\log V\right)=O\left(E\log V\right)$
- 邻接表+斐波那契堆：$O\left(V\log V+E\right)$

Floyd-Warshall

所有点对之间的最短路径，动态规划算法。

考虑 $f_k(i,j)$ 表示从 i 到 j 只经过 $\{1,\cdots,k\}$ 中的结点的最短路径长度。

Floyd-Warshall

所有点对之间的最短路径，动态规划算法。

- 考虑 $f_k(i,j)$ 表示从 i 到 j 只经过 $\{1,\cdots,k\}$ 中的结点的最短路径长度。
- 在所有只经过 $\{1,\cdots,k-1\}$ 中的结点的路径里尝试加入结点 k
- 如果从 i 先走到 k 再走到 j 更短，就更新
- $$f_k(i,j)=\min\{f_{k-1}(i,j),f_{k-1}(i,k)+f_{k-1}(k,j)\}$$

Floyd-Warshall

$$f_k(i,j) = \min\{f_{k-1}(i,j), f_{k-1}(i,k) + f_{k-1}(k,j)\}$$

类似于Bellman-Ford的实现，k 这一维不需要存，可以直接滚动。

```
for (auto k = 1; k <= n; ++k)
  for (auto i = 1; i <= n; ++i)
    for (auto j = 1; j <= n; ++j)
      if (dist[i][k] + dist[k][j] < dist[i][j])
        dist[i][j] = dist[i][k] + dist[k][j];
```

记路径

对于单源最短路径，在松弛的时候记录前驱结点

```
for (auto [v, w] : G[x])
  if (dist[x] + w < dist[v]) {
    dist[v] = dist[x] + w;
    pre[v] = x;
  }
```

之后顺着 `pre` 数组往前走，就相当于是在最短路径上倒着走，直到走到起点 `s`。

```
std::vector<int> reversed_path;
while (x != s) {
  reversed_path.push_back(x);
  x = pre[x];
}
reversed_path.push_back(s);
```

记路径

对于Floyd-Warshall，为每一对结点记录最后一次更新用的中间结点 `p[i][j]`。

```
if (dist[i][k] + dist[k][j] < dist[i][j]) {
  dist[i][j] = dist[i][k] + dist[k][j];
  p[i][j] = k;
}
```

递归地找出路径：从 `i` 到 `j` 的最短路就是从 `i` 到 `p[i][j]` 的最短路 + 从 `p[i][j]` 到 `j` 的最短路。

负环

边权**和**为负的环。它会让最短路长度变为 $-\infty$。

Bellman-Ford和Floyd-Warshall都可以判负环：

- Bellman-Ford运行完之后，枚举每一条边，如果还能松弛，说明有负环
- Floyd-Warshall运行完之后，如果出现 `dist[i][i] < 0`，说明有负环。

计数

最短路径可能有多条，统计最短路径的数量。

对于单源最短路径，记录 `cnt[x]` 表示从起点到 `x` 的最短路径数量。松弛时

- 如果 `dist[x] + w < dist[v]`，说明找到了一条从起点到 `v` 的更短的路径，则更新 `cnt[v] = cnt[x]`。
- 如果 `dist[x] + w == dist[v]`，说明找到了一条和已知的最短路径一样短的路径，则累加 `cnt[v] += cnt[x]`。

计数

Floyd-Warshall同理，记录 `cnt[i][j]` 表示从 `i` 到 `j` 的最短路径数量，松弛时更新。

乘法原理：`i` 到 `k` 的最短路径数量乘以 `k` 到 `j` 的最短路径数量。

```
if (dist[i][k] + dist[k][j] < dist[i][j]) {
  dist[i][j] = dist[i][k] + dist[k][j];
  cnt[i][j] = cnt[i][k] * cnt[k][j];
} else if (dist[i][k] + dist[k][j] == dist[i][j])
  cnt[i][j] += cnt[i][k] * cnt[k][j];
```

A*

- 本质是搜索算法，而CS101从最短路的角度讲A*。
- 有兴趣可以修读CS181：人工智能1
- 复习提纲：
  - 算法流程：Tree search, Graph search
  - heuristic的特征：admissibility, ($\Leftarrow$) consistency。
  - 使用何种heuristic配合何种算法可以optimal？