最优化小论文

22120307 陈景龙

北京交通大学

日期：2022 年 12 月 30 日

摘　　要

关键词：优化问题，01 背包，TSP 问题

目录

1 01 背包

1.1 问题与格式介绍

01 背包是在 n 个物品中取出若干个物品，装入容量为 m 的背包。每个物品有其对应的体积和价值，求解将哪些物品装入背包可以使获得的总价值最大。

输入格式包含三行：第一行为两个整数 n, m，代表物品的个数和背包的容量；第二行包含 n 个整数 $w_1, w_2, \ldots, w_n$，代表每个物品的重量；第三行包含 n 个整数 $v_1, v_2, \ldots, v_n$，代表每个物品的价值。

输出为一行一个整数，代表最大价值。

也就是我们需要求解如下优化问题

$$\begin{cases} \max & \sum_{i=1}^{n} c_i v_i \\ s.t. & \sum_{i=1}^{n} c_i w_i \leq m \\ & c_i \in \{0, 1\}, \quad i = 1, \ldots, n \end{cases} \tag{1}$$

1.2 暴力枚举法 (Brute force)

1.2.1 基本思想

暴力枚举法对于每个物品枚举其选或者不选的情况，对于枚举出来的每种情况，判断其总重量是否超过 m 的限制。在所有重量小于等于 m 的方案中，选出价值最高的一个即可。

1.2.2 复杂度分析

由于每个物品都存在两种选择方案，因此时间复杂度为 $\mathcal{O}(C \cdot 2^n)$，视枚举方法的不同，C 也有不同的取值。如果是先枚举选或不选的情况，在扫描一遍统计答案，则 $C = n$，复杂度为 $\mathcal{O}(n2^n)$，一般采用的是二进制方式进行枚举。

1.3 回溯法 (Backtracking)

1.3.1 基本思想

回溯法也是暴力枚举的一种，不过其采取的是边枚举边统计的方式，一般采取 Dfs 的方式，搜到一种结果后便跳转至上一个解空间树，搜索另一个可行解。由于其在枚举过程中就记录了 $weight$ 和 $value$ 的值，因此如果在枚举的途中发现 $weight > m$，便可以直接中止后序的搜索，这样便可以减少一些无用的搜索状态，称为剪枝 (prune)。

1.3.2 复杂度分析

尽管这样剪枝能够在某些情况下减掉一些状态，但其无法动摇复杂度的本质，因此该剪枝算法我们只称其具有**较小的常数**，并不认为其时间复杂度发生了变化，故该算法时间复杂度仍然为 $\mathcal{O}(2^n)$。

1.4 贪心法 (Greedy)

1.4.1 基本思想

贪心法是一种非常直观的算法，也是非常符合人们常识的一种算法，尽管其可能无法得到01背包的正确解。贪心法按照物品的单位价值降序排序，然后依次考虑每个物品，能放则放，最后统计选择的物品价值。

贪心法可以求解背包问题，因为背包问题中每个物品可以只放一部分。但在01背包问题中，物品仅有放与不放两个状态，因此贪心法求解01背包可能无法填满整个背包，从而导致获取的价值无法达到最优解。

1.4.2 复杂度分析

该算法在开始时需要对物品进行排序，然后再依次枚举。一般而言，我们认为排序的最好复杂度为 $\mathcal{O}\left(n\log_2 n\right)$，而枚举的复杂度则为 $\mathcal{O}(n)$，因此该算法的时间复杂度为 $\mathcal{O}\left(n\log_2 n\right)$。

1.5 回溯法 + 分支界限法 (剪枝 plus)

1.5.1 基本思想

1.3中我们已经介绍了回溯法以及一种可行的剪枝算法，而分支界限法则是在其基础之上，进一步提升了剪枝的效果。

分支界限法首先确定一个合理的界限函数，并根据该函数确定解空间的上下界。然后按照广度优先策略搜索问题的解空间，依次扩展该节点的所有子节点，并且计算这些子节点目标函数的（上界）。由于本题是求最大价值，所以如果子节点的上界仍然低于最开始所确定下界，则将该子节点丢弃，因为其生成的解不会比当前节点更优；否则将其加入优先队列，然后每次从队列中取出目标函数价值最大的点进行上述搜索。当然，在搜到一个可行解之后，解空间的下界将改为该可行解，然后不断重复上述过程直到队列为空。

1.5.2 复杂度分析

分支界限法是一个非常优秀的剪枝算法，但是其仍然无法改变回溯法的本质复杂度，因此该算法的时间复杂度仍然为 $\mathcal{O}\left(2^n\right)$。

1.6 动态规划法

1.6.1 基本思想

动态规划算法将背包问题划分为若干个阶段，第 i 个阶段表示考了了前i个物品所得最优解，用二维数组表示，则 $F[i][j]$ 表示前 i 个物品凑出重量 j 的最大价值。每一次加入新的物品，我们就从 $F[i][0\sim m]$ 更新 $F[i+1][0\sim m]$，更新方程为 $F\left[i\right]\left[j\right]=\max\left(F\left[i-1\right]\left[j-W\left[i\right]\right]+V\left[i\right]\right)$。由于动态规划算法每一次都是当前状态的最优解，换句话说，各个子问题的解只和它前面的子问题的解相关，而且子问题的解都是当前情况的最优解。因此我们不管以什么样的顺序添加物品，最终状态都将是考虑 n 个物品的最优解。

1.6.2 复杂度分析

每次添加一个新的物品，都需要扫描 $0 \sim m$ 去求解，所以动态规划算法的时间复杂度为 $\mathcal{O}(nm)$。此外，我们如果在枚举 $0 \sim m$ 的时候采用倒序枚举更新的方法，那么我们就可以直接省去数组的第一维，并且可以发现在一次更新过程中，先更新的值不会影响未更新的值，所以空间复杂度可以降至 $\mathcal{O}(m)$。

此外，在枚举的过程中，我们并不一定要枚举到 m，可以加入一些小优化，在考虑第 i 个物品时，枚举到 $\min\left(m, \sum_{j=1}^{i} w_j\right)$ 即可。

2 TSP 问题

2.1 问题与格式介绍

TSP 问题又称旅行商问题，该问题是在寻求单一旅行者由起点出发，通过所有给定的需求点之后，最后再回到原点的最小路径成本。本题我们给定编号为 $0 \sim n$ 的 $n+1$ 个城市，求旅行商从 0 号城市出发，周游所有城市后回到 0 号城市的最小成本。

输入包含若干行，第一行一个整数 n，代表城市的个数；之后给出一个 $(n+1) \times (n+1)$ 的矩阵 (行和列都从 0 开始编号)，第 i 行第 j 列的数 $V_{i,j}$ 代表城市 i 和城市 j 之间的距离，数据保证 $V_{i,i} = 0, V_{i,j} = V_{j,i}$，特殊数据要求会在相应算法中说明。

输出包含一行一个整数，代表最小路径成本。

问题可以表示为

$$\begin{cases} \min & \sum_{i=1}^{n+1} V_{c_{i-1},c_i} \\ s.t. & c_0 = c_{n+1} = 0 \\ & \{c_1, \ldots, c_n\} = \{1, \ldots, n\} \end{cases} \tag{2}$$

其中 $c_1, \ldots, c_n$ 是一个长度为 n 的排列。

2.2 枚举法 (Enumerative)

2.2.1 基本思想

我们暴力枚举 TSP 问题中所有可能的城市路径，即对于 $1 \sim n$ 这 n 座城市，枚举其所有可能的排列情况，假定其排列情况为 $p_1, p_2, \cdots, p_n$，那么 TSP 问题的一条可行路径为 $0 \to p_1 \to p_2 \to \cdots \to p_n \to 0$，再按照这样的路径统计成本，最后选取一个最小的即可。

2.2.2 复杂度分析

由于每次排列的统计复杂度是 $\mathcal{O}(n)$ 的，枚举所有可能的情况为 $\mathcal{O}(n!)$，因此该算法的总时间复杂度为 $\mathcal{O}(n!n)$。

2.3 回溯法 (Backtracking)

2.3.1 基本思想

回溯法也是暴力的一种，枚举法是枚举 $1 \sim n$ 的所有全排列，而回溯法是依次在 p_i 位置填数，然后回溯填其他数。

2.3.2 复杂度分析

回溯法每个位置的枚举复杂度为 $\mathcal{O}(n)$，总情况数为 $\mathcal{O}(n!)$，所以总时间复杂度为 $\mathcal{O}(n!n)$。

不过值得一提的是，回溯法可以方便的进行剪枝操作，在获得了一个可行解后，如果搜索过程中的答案大于等于可行解，便可以直接剪枝，因为它不可能比该可行解更优了。

2.4 贪心法 (Greedy)

2.4.1 基本思想

贪心法并不是能够得到 TSP 问题准确解的一种做法，但是它非常符合人们对于 TSP 问题的直观想法。而且，当点数很多的时候，传统的 TSP 精确算法往往很难找到正确的解，那么此时贪心法便可以迅速地找到一个较好的可行解。

贪心法从起点开始，每次找未经过且距离当前点最近的点，并且走到该位置，然后不断重复上述过程直至找到一条回路。

2.4.2 复杂度分析

由于每次找最小距离点的复杂度为 $\mathcal{O}(n)$，一共需要找 n 次，所以贪心法的时间复杂度为 $\mathcal{O}\left(n^2\right)$。

2.5 回溯 + 分支界限法

2.5.1 基本思想

分支界限法的重点在于如何确定界限函数。显然，我们可以通过贪心算法确定 TSP 问题的上界，那么我们该如何确定下界呢？对于无向图的代价矩阵，我们可以选取每一行最小的元素相加，可以得到一个简单的下界。但是，TSP 问题存在一个信息量更大的下界，考虑一个 TSP 问题的完整解，每个城市都有两条邻接边，一条为入边，一条为出边。那么，如果把矩阵中每一行最小的两个值相加，再除以 2（取上整），就可以得到一个合理的下界——尽管，这个解可能并不是一个可行解。

那么我们在搜索的时候，对于每个节点判断其下界，如果其下界仍然超过最开始设定的解上限，则舍去该节点。每次从队列中找到下界最小的点进行扩展，找到一个可行解后更新解上限，直到队列为空为止。

2.5.2 复杂度分析

分支界限法并不改变回溯法的复杂度，它只是进行了一个大力度的剪枝，故其时间复杂度为 $\mathcal{O}(n!n)$。

2.6 动态规划法 (Dynamic)

2.6.1 基本思想

首先，TSP 问题满足最优性原理，因此可以使用动态规划法求解。考虑如何设置状态，我们可以用 $F[i][S]$ 表示从起点出发，经过 S 集合中所有点，目前在 i 点的最小成本（显然起点一定在 S 中）。如何转移？我们枚举一个不在集合中的点 j，那么则有转移 $F[j][S+\{j\}] \leftarrow F[i][S]F[j][S+\{j\}]$ 取所有转移中的最小值即可。

2.6.2 复杂度分析

S 我们一般采用二进制状态压缩进行表示，枚举所有状态的复杂度是 $\mathcal{O}(2^n)$，枚举该状态中在集合中的点复杂度为 $\mathcal{O}(n)$，枚举一个不存在的点复杂度为 $\mathcal{O}(n)$，显然这三个复杂度是嵌套的，因此总时间复杂度为 $\mathcal{O}\left(n^2 2^n\right)$。

2.7 Clarke –Wright 算法

2.7.1 基本思想

作为一个 NP –Hard 问题，TSP 问题不存在多项式时间的精确解法。除了上述的贪心法之外，TSP 问题存在各种近似算法，它们旨在较为快速的得到一个可行的，较优的解，而并不是为了得到一个精确解——即，牺牲了准确度，换取了计算效率。

Clarke –Wright 启发式算法是求解 TSP 问题的一种近似解算法，算法首先选取一个点作为基准点（假定为 0 号点），记其他点到基准点的距离为 D_i，对于其他点两两计算一个 saving 值，即 $D_i + D_j - V_{i,j}$，这个值表示直接走这条边比从基准点走向这两个点所节约的值，因此叫做 saving 值。

首先我们对每个点 i 建一个 $0 \to i \to 0$ 的环，然后按照 saving 值降序排序考虑每个点对，如果点对 $\langle x, y\rangle$ 属于同一个环，则跳过；如果 $\langle x, y\rangle$ 的均在其各自所在环的顶/末端，则连接 $\langle x, y\rangle$，并且取消两者跟 0 的连线。重复上述操作直到所有点都在一个环上，所得的路径即为答案。

2.7.2 复杂度分析

该算法首先要对所有点对进行排序，故时间复杂度为 $\mathcal{O}(mlogm)$，即 $\mathcal{O}\left(n^2 logn\right)$，之后对每组点对进行枚举，操作的复杂度可视为常数，故总时间复杂度为 $\mathcal{O}\left(n^2 logn\right)$。

2.8 MST 启发式算法

2.8.1 基本思想

除了上述的 Clarke –Wright 启发式算法，MST 启发式算法也是求解 TSP 问题的一种启发式算法。算法首先生成一棵最小生成树 (Minimum Spanning Tree, MST)，然后在 MST 上指定一个根，从根节点开始按照前序遍历的方式（兄弟节点的顺序不管）生成一条路径，这样得到的路径即为所求的回路。

2.8.2 复杂度分析

求解 MST 可以用到 Kruskal 或者 Prim 算法，这里我选用了前者，复杂度为 $\mathcal{O}(mlogm)$，由于图为完全图，所以 $m = n^2$，故复杂度为 $\mathcal{O}\left(n^2logn\right)$；之后前序遍历的复杂度是 $\mathcal{O}(n)$ 的，所以总时间复杂度为 $\mathcal{O}\left(n^2logn\right)$。

2.9 Christofides 算法

2.9.1 基本思想

TSP 问题不存在多项式时间的常数近似，但如果其至于度量空间下，则存在多项式时间的常数近似算法（近似比可低于 3/2）。所谓度量空间，就是对于任意三个点 (u, v, w)，都满足 $V_{u,w} \leq V_{u,v} + V_{v,w}$，即满足三角不等式。也就是说，如果数据保证任意三点均满足三角不等式，我们便可以使用 Christofides 算法。

Christofides 算法首先计算图 G 的最小生成树 T，记 T 中所有度数为奇数的点为集合 $V^{'}$，根据握手定理可得 $V^{'}$ 的大小必定是偶数。然后从 G 的导出子图 $G(V^{'})$ 中求出最小权值完美匹配 M（可用带权带花树实现），然后记图 $G^{'} = M + T$，我们找到 $G^{'}$ 的任意一个欧拉回路 E，再根据 E 的点序构造一个哈密顿回路 H，那么所得到的 H 即为 Christofides 算法所求的解。在通过 E 构造 H 的过程中，如果 E_i 所表示的点在 $E_{1\sim i-1}$ 中出现过，则删去 E_i，最后得到的剩余序列 E 即为所求的 H 序列。

显然，如果我们直接按照 E 游览所有的点，那么耗费成本为 $W(T)+W(M)$，由于三角不等式的存在，所以我们按照 H 游览所有点时，若记其成本为 W，则必然有 $W \leq W(T) + W(M)$，因为删点的时候存在三角不等式的缩放。

假定旅行商问题最优解回路为 C，删除 C 中任意一条边则为 G 的一棵生成树 $T^{'}$，那么必然有 $W(T) \leq W(T^{'}) \leq W(C)$；考虑给 $V^{'}$ 中的点按 C 的路径顺序进行排序，得到 $V_1', V_2', \cdots, V_k'$，按该顺序生成一个回路 $C^{'}$，根据三角形不等式显然有 $W(C^{'}) \leq W(C)$。由于 $V^{'}$ 的大小为偶数，因此 $C^{'}$ 为一个二分图，故其存在两种二分图匹配方案 K_1, K_2，且必然有 $\min\{W(K_1), W(K_2)\} \leq W(C^{'})/2$，由于 M 是 $G(V^{'})$ 的最小完备匹配，故 $W(M) \leq \min\{W(K_1), W(K_2)\} \leq W(C^{'})/2 \leq W(C)/2$。

故综上所示，$W \leq W(T) + W(M) \leq W(C) + W(C)/2 = 3/2W(C)$。

2.9.2 复杂度分析

最小生成树采用 Kruskal 算法，复杂度为 $\mathcal{O}(m \log m)$，又因为该图为完全图，则 m 与 n^2 同阶，故 G 生成 T 的复杂度 $\mathcal{O}\left(n^2 logn\right)$。考虑 $G(V')$ 中求得 M 的复杂度，由于 $G(V')$ 中点数与 G 同阶，带花树实现一般图最大/小权匹配的复杂度为 $\mathcal{O}\left(mn^2\right)$，而本题代码实现了部分优化，所以复杂度降至 $\mathcal{O}\left(n^3\right)$。显然，图 T 与 M 中点数与边数均同阶，而 Euler 路径的算法复杂度为 $\mathcal{O}(n+m)$，即为 $\mathcal{O}(n)$。最终，总时间复杂度取 $\mathcal{O}\left(n^3\right)$。

2.10 OPT 算法

2.10.1 基本思想

2 -opt 算法是一个局部迭代优化算法，和上述启发式建立回路方式有所差距。2 -opt 算法首先随机生成一个回路，然后随机截取回路上的一段区间，将其翻转，得到一个新的回路序列。然后比较新旧回路序列的成本，如果更优则保留新的回路；如果重复上述操作若干次（通常会设置一个上限）仍然无法获得一个新的更优序列，则不再操作，以当前序列当做所求的答案。

由于这样每次操作只会改变两个点，因此称为 2 -opt 算法，故同理也有 3 -opt 或者 k -opt 算法，但这里不再赘述。

2.10.2 复杂度分析

2 -opt 算法每次操作的复杂度是 $\mathcal{O}(c)$ 的，该算法总复杂度较难分析，但可以确定的是，上限越高，操作次数越多，程序越慢，答案约精确；反之则越快，但是答案偏差也会越大。

2.11 遗传算法 (GA)

2.11.1 基本思想

遗传算法是一种通过模拟自然进化过程搜索最优解的方法，通过数学的方式，利用计算机仿真运算，将问题的求解转化为类似生物进化中的染色体基因交叉、变异的过程，再迭代若干代种群后得到较好的优化结果。

遗传算法的效率与效果取决于参数设置、编码方式、以及染色体变换方式。本题中种群大小设置为 50(W)，迭代次数设置为 10000(G)，交换概率设置为 1，变异概率设置为 0.1；染色体编码为一段随机序列，将其升序排序后的原下标组成的序列即为一条 TSP 回路；染色体交换采取自然界中染色体交换方式，两两交换一段序列；染色体变异采取自交换方式，即自身两段染色体位置发生改变。

设置好了如上参数与规则，我们便可以按照自然界规则进行优胜劣汰模拟，每次迭代诞生新的种群，再从新旧种群中一起淘汰成本过大的染色体，留下成本较低的优秀染色体，再进行如上操作，迭代至上限即可。

2.11.2 复杂度分析

每次迭代的复杂度来自于染色体交换操作，染色体交换需要交换 $W/2$ 对，每次交换需要扫描整个染色体的长度，总共需要迭代 G 次，因此时间复杂度为 $\mathcal{O}(nWG)$。

2.12 模拟退火算法 (SA)

2.12.1 基本思想

在高温条件下, 粒子的能量较高, 可以自由运动和重新排列。在低温条件下, 粒子能量较低。如果从高温开始, 非常缓慢地降温 (这个过程被称为退火), 粒子就可以在每个温度下达到热平衡。当系统完全被冷却时, 最终形成处于低能状态的晶体。

如果温度下降十分缓慢, 而在每个温度都有足够多次的状态转移, 使之在每一个温度下达到热平衡, 则全局最优解将以概率 1 被找到。因此可以说模拟退火算法可以找到全局最优解。

对于本题而言，首先我们通过随机算法构造一个较好的在可行序列后，然后每次迭代，我们采用 2 −opt 算法，令 Δf 为原序列距离和随机变换后的序列距离的差值。如果 $\Delta f < 0$，则接受新的路径；否则以 $\exp\{(-\Delta f/T)\}$ 的概率接受新的路径。每操作一次，T 乘上降温系数 α，当温度降至临界值以下后，退出循环，此时我们认为找到了一个近似的全局最优解。

2.12.2 复杂度分析

模拟退火算法的时间复杂度取决于参数设置，温度初值，降温系数，临界值都会影响时间复杂度。假定温度初值为 T，降温系数为 α，临界值为 e，则通过计算可得最低迭代次数为 $\frac{lne-lnT}{lna}$。

附录

01 背包 Brute Force 解法

```cpp
/*program from Wolfycz*/
#include<map>
#include<cmath>
#include<queue>
#include<cstdio>
#include<vector>
#include<cstring>
#include<limits.h>
#include<iostream>
#include<algorithm>
#define MK make_pair
#define sqr(x) ((x)*(x))
#define pii pair<int,int>
#define UNUSED(x) (void)(x)
using namespace std;
typedef long long ll;
```

```
template<typename T>inline T read(T x) {
    int f = 1; char ch = getchar();
    for (; ch < '0' || ch>'9'; ch = getchar())  if (ch == '-')  f = -1;
    for (; ch >= '0' && ch <= '9'; ch = getchar())  x = (x << 1) + (x << 3) + ch - '0';
    return x * f;
}
int BruteForce(int* w, int* v, int n, int m) {
    int Ans = numeric_limits<int>::min();
    for (int sta = 0; sta < 1 << n; sta++) {
        int value = 0, weight = 0;
        for (int i = 0; i < n; i++) {
            if (!(sta >> i & 1))    continue;
            weight += w[i + 1];
            value += v[i + 1];
        }
        if (weight <= m)    //如果合法
            Ans = max(Ans, value);
    }
    return Ans;
}
const int N = 1e5;
int w[N + 10], v[N + 10];
int main() {
    int n = read(0), m = read(0);
    for (int i = 1; i <= n; i++)   w[i] = read(0);
    for (int i = 1; i <= n; i++)   v[i] = read(0);
    printf("%d\n", BruteForce(w, v, n, m));
    return 0;
}
```

01 背包回溯剪枝解法

```
/*program from Wolfycz*/
#include<map>
#include<cmath>
#include<queue>
#include<cstdio>
#include<vector>
#include<cstring>
#include<limits.h>
#include<iostream>
#include<algorithm>
#define MK make_pair
#define sqr(x) ((x)*(x))
#define pii pair<int,int>
#define UNUSED(x) (void)(x)
using namespace std;
typedef long long ll;
template<typename T>inline T read(T x) {
```

```
    int f = 1; char ch = getchar();
    for (; ch < '0' || ch>'9'; ch = getchar())  if (ch == '-')  f = -1;
    for (; ch >= '0' && ch <= '9'; ch = getchar())  x = (x << 1) + (x << 3) + ch - '0';
    return x * f;
}
void Dfs(int* w, int* v, int x, int weight, int value, int n, int m, int& Ans) {
    if (x > n) {
        if (weight <= m)
            Ans = min(Ans, value);
        return;
    }
    Dfs(w, v, x + 1, weight, value, n, m, Ans); //不选
    Dfs(w, v, x + 1, weight + w[x], value + v[x], n, m, Ans);   //选
}
int Backtracking_Prune(int* w, int* v, int n, int m) {
    int Ans = numeric_limits<int>::min();
    Dfs(w, v, 1, 0, 0, n, m, Ans);
    return Ans;
}
const int N = 1e5;
int w[N + 10], v[N + 10];
int main() {
    int n = read(0), m = read(0);
    for (int i = 1; i <= n; i++)   w[i] = read(0);
    for (int i = 1; i <= n; i++)   v[i] = read(0);
    printf("%d\n", Backtracking_Prune(w, v, n, m));
    return 0;
}
```

01 背包贪心法

```
/*program from Wolfycz*/
#include<map>
#include<cmath>
#include<queue>
#include<cstdio>
#include<vector>
#include<cstring>
#include<limits.h>
#include<iostream>
#include<algorithm>
#define MK make_pair
#define sqr(x) ((x)*(x))
#define pii pair<int,int>
#define UNUSED(x) (void)(x)
using namespace std;
typedef long long ll;
template<typename T>inline T read(T x) {
    int f = 1; char ch = getchar();
```

```
    for (; ch < '0' || ch>'9'; ch = getchar())  if (ch == '-')  f = -1;
    for (; ch >= '0' && ch <= '9'; ch = getchar())  x = (x << 1) + (x << 3) + ch - '0';
    return x * f;
}
int Greedy(int* w, int* v, int n, int m) {
    int id[n + 10];
    for (int i = 1; i <= n; i++)  id[i] = i;
    sort(id + 1, id + 1 + n, [=](int x, int y) {return v[x] * w[y] >= v[y] * w[x]; });  //按单位价值排序
    int weight = 0, value = 0;
    for (int i = 1; i <= n; i++) {  //贪心选取
        if (weight + w[id[i]] > m)  continue;
        weight += w[id[i]];
        value += v[id[i]];
    }
    return value;
}
const int N = 1e5;
int w[N + 10], v[N + 10];
int main() {
    int n = read(0), m = read(0);
    for (int i = 1; i <= n; i++)   w[i] = read(0);
    for (int i = 1; i <= n; i++)   v[i] = read(0);
    printf("%d\n", Greedy(w, v, n, m));
    return 0;
}
```

01 背包分支界限法

```
/*program from Wolfycz*/
#include<map>
#include<cmath>
#include<queue>
#include<cstdio>
#include<vector>
#include<cstring>
#include<limits.h>
#include<iostream>
#include<algorithm>
#define MK make_pair
#define sqr(x) ((x)*(x))
#define pii pair<int,int>
#define UNUSED(x) (void)(x)
using namespace std;
typedef long long ll;
template<typename T>inline T read(T x) {
    int f = 1; char ch = getchar();
    for (; ch < '0' || ch>'9'; ch = getchar())  if (ch == '-')  f = -1;
    for (; ch >= '0' && ch <= '9'; ch = getchar())  x = (x << 1) + (x << 3) + ch - '0';
```

```
    return x * f;
}
int calc(int* w, int* v, int* id, int x, int weight, int value, int n, int m) {
    return value + ceil(1.0 * (m - weight) * v[id[x]] / w[id[x]]);
}   //目标函数
struct node {
    int x, w, v, Max;
    node(int x = 0, int w = 0, int v = 0) :x(x), w(w), v(v) { Max = 0; }
    bool operator<(const node& ots)const { return Max < ots.Max; }
};
int Bfs(int* w, int* v, int* id, int x, int n, int m) {
    priority_queue<node>H;  //优先队列，优先扩展目标函数值更大的节点
    H.push(node(x, 0, 0));
    int weight = 0, Min = 0;
    for (int i = 1; i <= n; i++) {  //计算下界
        if (weight + w[id[i]] > m)  continue;
        weight += w[id[i]];
        Min += v[id[i]];
    }
    while (!H.empty()) {
        node Now = H.top();
        H.pop();
        if (Now.x > n) {    //找到一个可行解，更新下界
            Min = max(Min, Now.v);
            continue;
        }
        node ls(Now.x + 1, Now.w, Now.v);   //不选
        node rs(Now.x + 1, Now.w + w[id[Now.x]], Now.v + v[id[Now.x]]); //选
        ls.Max = calc(w, v, id, ls.x, ls.w, ls.v, n, m);
        rs.Max = calc(w, v, id, rs.x, rs.w, rs.v, n, m);
        if (ls.w <= m && (ls.x > n || ls.Max >= Min)) H.push(ls);
        if (rs.w <= m && (rs.x > n || rs.Max >= Min)) H.push(rs);
        //满足限制，且目标值不低于下界，则加入队列
    }
    return Min;
}
int babm(int* w, int* v, int n, int m) {
    int id[n + 10];
    for (int i = 1; i <= n; i++)  id[i] = i;
    sort(id + 1, id + 1 + n, [=](int x, int y) {return v[x] * w[y] >= v[y] * w[x]; });
    return Bfs(w, v, id, 1, n, m);
}
const int N = 1e5;
int w[N + 10], v[N + 10];
int main() {
    int n = read(0), m = read(0);
    for (int i = 1; i <= n; i++)   w[i] = read(0);
    for (int i = 1; i <= n; i++)   v[i] = read(0);
```

```
    printf("%d\n", babm(w, v, n, m));
    return 0;
}
```

01 背包动态规划法

```
/*program from Wolfycz*/
#include<map>
#include<cmath>
#include<queue>
#include<cstdio>
#include<vector>
#include<cstring>
#include<limits.h>
#include<iostream>
#include<algorithm>
#define MK make_pair
#define sqr(x) ((x)*(x))
#define pii pair<int,int>
#define UNUSED(x) (void)(x)
using namespace std;
typedef long long ll;
template<typename T>inline T read(T x) {
    int f = 1; char ch = getchar();
    for (; ch < '0' || ch>'9'; ch = getchar())  if (ch == '-')  f = -1;
    for (; ch >= '0' && ch <= '9'; ch = getchar())  x = (x << 1) + (x << 3) + ch - '0';
    return x * f;
}
int Dynamic(int* w, int* v, int n, int m) {
    int F[m + 10], weight = 0;
    memset(F, 0, sizeof(F));
    for (int i = 1; i <= n; i++) {  //枚举每个物品
        weight += w[i];
        for (int j = min(weight, m); j >= w[i]; j--)    //倒序枚举更新
            F[j] = max(F[j], F[j - w[i]] + v[i]);
    }
    int Ans = 0;
    for (int i = m; i; i--)
        Ans = max(Ans, F[i]);
    return Ans;
}
const int N = 1e5;
int w[N + 10], v[N + 10];
int main() {
    int n = read(0), m = read(0);
    for (int i = 1; i <= n; i++)   w[i] = read(0);
    for (int i = 1; i <= n; i++)   v[i] = read(0);
    printf("%d\n", Dynamic(w, v, n, m));
    return 0;
```

```
}
```

TSP 枚举法

```
/*program from Wolfycz*/
#include<map>
#include<cmath>
#include<queue>
#include<cstdio>
#include<vector>
#include<cstring>
#include<limits.h>
#include<iostream>
#include<algorithm>
#define MK make_pair
#define sqr(x) ((x)*(x))
#define pii pair<int,int>
#define UNUSED(x) (void)(x)
using namespace std;
typedef long long ll;
template<typename T>inline T read(T x) {
    int f = 1; char ch = getchar();
    for (; ch < '0' || ch>'9'; ch = getchar())  if (ch == '-')  f = -1;
    for (; ch >= '0' && ch <= '9'; ch = getchar())  x = (x << 1) + (x << 3) + ch - '0';
    return x * f;
}
int Enumerative(const vector<vector<int>>& Dis, int n) {
    int p[n + 10], Ans = numeric_limits<int>::max();
    for (int i = 1; i <= n; i++)  p[i] = i;
    p[0] = p[n + 1] = 0;
    do {
        int temp = 0;
        for (int i = 0; i <= n; i++)
            temp += Dis[p[i]][p[i + 1]];
        Ans = min(Ans, temp);
    } while (next_permutation(p + 1, p + 1 + n));
    //next_permutation生成下一个全排列，复杂度是均摊O(1)的
    return Ans;
}
int main() {
    int n = read(0);
    vector<vector<int>>Dis;
    for (int i = 0; i <= n; i++) {
        vector<int>temp;
        for (int j = 0; j <= n; j++)
            temp.emplace_back(read(0));
        Dis.emplace_back(temp);
    }
    printf("%d\n", Enumerative(Dis, n));
```

```
    return 0;
}
```

TSP 回溯法

```
/*program from Wolfycz*/
#include<map>
#include<cmath>
#include<queue>
#include<cstdio>
#include<vector>
#include<cstring>
#include<limits.h>
#include<iostream>
#include<algorithm>
#define MK make_pair
#define sqr(x) ((x)*(x))
#define pii pair<int,int>
#define UNUSED(x) (void)(x)
using namespace std;
typedef long long ll;
template<typename T>inline T read(T x) {
    int f = 1; char ch = getchar();
    for (; ch < '0' || ch>'9'; ch = getchar())  if (ch == '-')  f = -1;
    for (; ch >= '0' && ch <= '9'; ch = getchar())  x = (x << 1) + (x << 3) + ch - '0';
    return x * f;
}
void Dfs(int x, const vector<vector<int>>& Dis, map<int, bool>& Vis, int Last, int temp, int& Ans,
    int n) {
    if (temp >= Ans)   return;  //剪枝
    if (x > n) {
        Ans = min(Ans, temp + Dis[Last][0]);
        return;
    }
    for (int i = 1; i <= n; i++) {  //枚举当前位置填的数
        if (Vis[i]) continue;
        Vis[i] = true;
        Dfs(x + 1, Dis, Vis, i, temp + Dis[Last][i], Ans, n);
        Vis[i] = false;
    }
}
int Backtracking(const vector<vector<int>>& Dis, int n) {
    int Ans = numeric_limits<int>::max();
    map<int, bool>Vis;
    Dfs(1, Dis, Vis, 0, 0, Ans, n);
    return Ans;
}
int main() {
    int n = read(0);
```

```
    vector<vector<int>>Dis;
    for (int i = 0; i <= n; i++) {
        vector<int>temp;
        for (int j = 0; j <= n; j++)
            temp.emplace_back(read(0));
        Dis.emplace_back(temp);
    }
    printf("%d\n", Backtracking(Dis, n));
    return 0;
}
```

TSP 贪心法

```
/*program from Wolfycz*/
#include<map>
#include<cmath>
#include<queue>
#include<cstdio>
#include<vector>
#include<cstring>
#include<limits.h>
#include<iostream>
#include<algorithm>
#define MK make_pair
#define sqr(x) ((x)*(x))
#define pii pair<int,int>
#define UNUSED(x) (void)(x)
using namespace std;
typedef long long ll;
template<typename T>inline T read(T x) {
    int f = 1; char ch = getchar();
    for (; ch < '0' || ch>'9'; ch = getchar())  if (ch == '-')  f = -1;
    for (; ch >= '0' && ch <= '9'; ch = getchar())  x = (x << 1) + (x << 3) + ch - '0';
    return x * f;
}
void Greedy(const vector<vector<int>>& Dis, map<int, bool>& Vis, int x, int Last, int n, int& Ans) {
    if (x > n) {    //搜索到底
        Ans += Dis[Last][0];
        return;
    }
    int Min = numeric_limits<int>::max(), ID = 0;
    for (int i = 1; i <= n; i++) {  //找到未访问过的最近点的
        if (Vis[i]) continue;
        if (Dis[Last][i] < Min) {
            Min = Dis[Last][i];
            ID = i;
        }
    }
    Vis[ID] = true;
```

```
    Ans += Dis[Last][ID];
    Greedy(Dis, Vis, x + 1, ID, n, Ans);
}
int Greedy(const vector<vector<int>>& Dis, int n) {
    map<int, bool>Vis;
    int Ans = 0;
    Greedy(Dis, Vis, 1, 0, n, Ans);
    //这里使用了函数同名不同参的重载
    return Ans;
}
int main() {
    int n = read(0);
    vector<vector<int>>Dis;
    for (int i = 0; i <= n; i++) {
        vector<int>temp;
        for (int j = 0; j <= n; j++)
            temp.emplace_back(read(0));
        Dis.emplace_back(temp);
    }
    printf("%d\n", Greedy(Dis, n));
    return 0;
}
```

TSP 分治界限法

```
/*program from Wolfycz*/
#include<map>
#include<cmath>
#include<queue>
#include<cstdio>
#include<vector>
#include<cstring>
#include<limits.h>
#include<iostream>
#include<algorithm>
#define MK make_pair
#define sqr(x) ((x)*(x))
#define pii pair<int,int>
#define UNUSED(x) (void)(x)
using namespace std;
typedef long long ll;
template<typename T>inline T read(T x) {
    int f = 1; char ch = getchar();
    for (; ch < '0' || ch>'9'; ch = getchar())  if (ch == '-')  f = -1;
    for (; ch >= '0' && ch <= '9'; ch = getchar())  x = (x << 1) + (x << 3) + ch - '0';
    return x * f;
}
void Greedy(const vector<vector<int>>& Dis, map<int, bool>& Vis, int x, int Last, int n, int& Ans) {
    if (x > n) {
```

```
25          Ans += Dis[Last][0];
26          return;
27      }
28      int Min = numeric_limits<int>::max(), ID = 0;
29      for (int i = 1; i <= n; i++) {
30          if (Vis[i]) continue;
31          if (Dis[Last][i] < Min) {
32              Min = Dis[Last][i];
33              ID = i;
34          }
35      }
36      Vis[ID] = true;
37      Ans += Dis[Last][ID];
38      Greedy(Dis, Vis, x + 1, ID, n, Ans);
39  }
40  struct node {
41      vector<int>list;
42      map<int, bool>Vis;
43      int Min;
44      node() { Min = 0; }
45      node(vector<int>list, map<int, bool>Vis, int Min) :list(list), Vis(Vis), Min(Min) {}
46      bool operator<(const node& ots)const { return Min < ots.Min; }
47      bool operator>(const node& ots)const { return Min > ots.Min; }
48  };
49  struct Mininums {   //存储两个最小数
50      int M1, M2;
51      Mininums(int M1 = numeric_limits<int>::max(), int M2 = numeric_limits<int>::max()) :M1(M1), M2(
            M2) {}
52      void insert(int v) {
53          if (v < M1)   M2 = M1, M1 = v;
54          else    if (v < M2)   M2 = v;
55      }
56  };
57  int calc(const vector<vector<int>>& Dis, vector<int>& list, map<int, bool>Vis, int n) {
58      int Ans = 0;
59      for (int i = 1; i < (int)list.size(); i++)
60          Ans += Dis[list[i - 1]][list[i]];
61      Ans <<= 1;
62      for (int i = 0; i <= n; i++) {
63          if (Vis[i] && i != list.front() && i != list.back())
64              continue;
65          Mininums temp;
66          for (int j = 0; j <= n; j++) {
67              if (i == j)   continue;
68              temp.insert(Dis[i][j]);
69          }
70          Ans += temp.M1 + temp.M1;   //不在路径中的点计算该行最小的两个数
71          if (i == list.front() || i == list.back())
```

```
            Ans -= temp.M2; //路径的两端记录最小的数
    }
    return (Ans + 1) >> 1;
}
int babm(const vector<vector<int>>& Dis, int n) {
    map<int, bool>Vis;
    int Max = 0;
    Greedy(Dis, Vis, 1, 0, n, Max); //贪心法计算上限
    priority_queue<node, vector<node>, greater<node>>H;
    H.push(node(vector<int>{0}, map<int, bool>{MK(0, true)}, 0));
    while (!H.empty()) {
        node Now = H.top();
        H.pop();
        if (Now.Min > Max)    continue;
        if ((int)Now.list.size() == n + 1) {
            int Ans = 0;
            for (int i = 1; i < (int)Now.list.size(); i++)
                Ans += Dis[Now.list[i - 1]][Now.list[i]];
            Ans += Dis[Now.list.back()][0];
            Max = min(Max, Ans);
            continue;
        }
        for (int i = 0; i <= n; i++) {  //枚举当前位置填的数
            if (Now.Vis[i]) continue;
            Now.list.emplace_back(i);
            Now.Vis[i] = true;
            int Min = calc(Dis, Now.list, Now.Vis, n);
            if (Min <= Max) //目标函数低于上界则加入队列
                H.push(node(Now.list, Now.Vis, Min));
            Now.list.pop_back();
            Now.Vis[i] = false;
        }
    }
    return Max;
}
int main() {
    int n = read(0);
    vector<vector<int>>Dis;
    for (int i = 0; i <= n; i++) {
        vector<int>temp;
        for (int j = 0; j <= n; j++)
            temp.emplace_back(read(0));
        Dis.emplace_back(temp);
    }
    printf("%d\n", babm(Dis, n));
    return 0;
}
```

TSP 动态规划法

```cpp
/*program from Wolfycz*/
#include<map>
#include<set>
#include<cmath>
#include<queue>
#include<cstdio>
#include<vector>
#include<cstring>
#include<limits.h>
#include<iostream>
#include<algorithm>
#define MK make_pair
#define sqr(x) ((x)*(x))
#define pii pair<int,int>
#define UNUSED(x) (void)(x)
#define lowbit(x) ((x)&(-x))
using namespace std;
typedef long long ll;
template<typename T>inline T read(T x) {
    int f = 1; char ch = getchar();
    for (; ch < '0' || ch>'9'; ch = getchar())  if (ch == '-')  f = -1;
    for (; ch >= '0' && ch <= '9'; ch = getchar())  x = (x << 1) + (x << 3) + ch - '0';
    return x * f;
}
int Dynamic(const vector<vector<int>>& Dis, int n) {
    int F[n + 1][1 << (n + 1)];
    memset(F, 0x7f, sizeof(F));
    F[0][1] = 0;    //初始在0号点
    for (int sta = 0; sta < 1 << (n + 1); sta++)
        for (int i = 0; i <= n; i++)
            if (sta >> i & 1)   //枚举集合中的点i
                for (int k = 0; k <= n; k++)
                    if (sta >> k & 1 && i != k)   //枚举集合中另一个点k，从k走到i
                        F[i][sta] = min(F[i][sta], F[k][sta ^ (1 << i)] + Dis[k][i]);
    int Ans = numeric_limits<int>::max();
    for (int i = 1; i <= n; i++)
        Ans = min(Ans, F[i][(1 << (n + 1)) - 1] + Dis[i][0]);
    return Ans;
}
int main() {
    int n = read(0);
    vector<vector<int>>Dis;
    for (int i = 0; i <= n; i++) {
        vector<int>temp;
        for (int j = 0; j <= n; j++)
            temp.emplace_back(read(0));
```

```
        Dis.emplace_back(temp);
    }
    printf("%d\n", Dynamic(Dis, n));
    return 0;
}
```

TSP Clarke -Wright 算法

```
/*program from Wolfycz*/
#include<map>
#include<cmath>
#include<queue>
#include<cstdio>
#include<vector>
#include<cstring>
#include<limits.h>
#include<iostream>
#include<algorithm>
#define MK make_pair
#define sqr(x) ((x)*(x))
#define pii pair<int,int>
#define UNUSED(x) (void)(x)
using namespace std;
typedef long long ll;
template<typename T>inline T read(T x) {
    int f = 1; char ch = getchar();
    for (; ch < '0' || ch>'9'; ch = getchar())  if (ch == '-')  f = -1;
    for (; ch >= '0' && ch <= '9'; ch = getchar())  x = (x << 1) + (x << 3) + ch - '0';
    return x * f;
}
int ClarkeWright(vector<vector<int>>& Dis, int n) {
    struct node {
        int x, y, f;
        node(int x = 0, int y = 0, int f = 0) :x(x), y(y), f(f) {}
        bool operator<(const node& ots)const { return f > ots.f; }
    };
    vector<node>pairs;
    for (int i = 1; i <= n; i++)    //统计点对
        for (int j = 1; j < i; j++)
            pairs.emplace_back(node(i, j, Dis[0][i] + Dis[j][0] - Dis[i][j]));
    sort(pairs.begin(), pairs.end());
    vector<int>list[n + 1], Frm(n + 1);
    for (int i = 1; i <= n; i++)    //每个序列初始只包含自己
        list[Frm[i] = i].emplace_back(i);
    for (auto pair : pairs) {
        int Fx = Frm[pair.x], Fy = Frm[pair.y];
        if (Fx == Fy) continue; //如果在同一个序列中则跳过
        if (list[Fx].front() != pair.x && list[Fx].back() != pair.x)    continue;
        if (list[Fy].front() != pair.y && list[Fy].back() != pair.y)    continue;
```

```
        //不在序列头/尾则跳过
        if (list[Fx].front() == pair.x) reverse(list[Fx].begin(), list[Fx].end());
        if (list[Fy].back() == pair.y)  reverse(list[Fy].begin(), list[Fy].end());
        //同一方向
        for (auto p : list[Fy]) {   //将y所在序列加入x中
            Frm[p] = Fx;
            list[Fx].emplace_back(p);
        }
        list[Fy].clear();
    }
    Frm[0] = Frm[1];
    list[Frm[0]].insert(list[Frm[0]].begin(), 0);
    list[Frm[0]].emplace_back(0);   //在序列头尾加上0，统计答案
    int Ans = 0;
    for (int i = 1; i < (int)list[Frm[0]].size(); i++)
        Ans += Dis[list[Frm[0]][i - 1]][list[Frm[0]][i]];
    return Ans;
}
int main() {
    int n = read(0);
    vector<vector<int>>Dis;
    for (int i = 0; i <= n; i++) {
        vector<int>temp;
        for (int j = 0; j <= n; j++)
            temp.emplace_back(read(0));
        Dis.emplace_back(temp);
    }
    printf("%d\n", ClarkeWright(Dis, n));
    return 0;
}
```

TSP MST 启发式算法

```
/*program from Wolfycz*/
#include<map>
#include<cmath>
#include<queue>
#include<cstdio>
#include<vector>
#include<cstring>
#include<limits.h>
#include<iostream>
#include<algorithm>
#define MK make_pair
#define sqr(x) ((x)*(x))
#define pii pair<int,int>
#define UNUSED(x) (void)(x)
using namespace std;
typedef long long ll;
```

```cpp
template<typename T>inline T read(T x) {
    int f = 1; char ch = getchar();
    for (; ch < '0' || ch>'9'; ch = getchar())  if (ch == '-')  f = -1;
    for (; ch >= '0' && ch <= '9'; ch = getchar())  x = (x << 1) + (x << 3) + ch - '0';
    return x * f;
}
int MST(vector<vector<int>>& Dis, int n) {
    struct Line {
        int x, y, v;
        Line(int x = 0, int y = 0, int v = 0) :x(x), y(y), v(v) {}
        bool operator<(const Line& ots)const { return v < ots.v; }
    };
    vector<Line>lines;
    vector<int>Fa, Vert[n + 1];
    function<int(int)>Find = [&](int x) {return x == Fa[x] ? x : Fa[x] = Find(Fa[x]); };
    //并查集
    for (int i = 0; i <= n; i++)    Fa.emplace_back(i);
    for (int i = 0; i <= n; i++)
        for (int j = 0; j <= n; j++)
            lines.emplace_back(Line(i, j, Dis[i][j]));
    sort(lines.begin(), lines.end());
    for (auto line : lines) {   //Kruskal算法求解MST
        int Fx = 0, Fy = 0;
        if ((Fx = Find(line.x)) != (Fy = Find(line.y))) {
            Fa[Fx] = Fy;
            Vert[line.x].emplace_back(line.y);
            Vert[line.y].emplace_back(line.x);
        }
    }
    vector<int>list;
    function<void(int, int)>Dfs = [&](int x, int fa) {  //Dfs查找前序遍历
        list.emplace_back(x);
        for (auto son : Vert[x]) {
            if (son == fa)    continue;
            Dfs(son, x);
        }
    };
    Dfs(0, 0);
    list.emplace_back(list.front());
    int Ans = 0;
    for (int i = 1; i < (int)list.size(); i++)
        Ans += Dis[list[i - 1]][list[i]];
    return Ans;
}
int main() {
    int n = read(0);
    vector<vector<int>>Dis;
    for (int i = 0; i <= n; i++) {
```

```
        vector<int>temp;
        for (int j = 0; j <= n; j++)
            temp.emplace_back(read(0));
        Dis.emplace_back(temp);
    }
    printf("%d\n", MST(Dis, n));
    return 0;
}
```

TSP Christofides 算法

```
/*program from Wolfycz*/
#include<map>
#include<cmath>
#include<queue>
#include<cstdio>
#include<vector>
#include<cstring>
#include<limits.h>
#include<iostream>
#include<algorithm>
#define MK make_pair
#define sqr(x) ((x)*(x))
#define pii pair<int,int>
#define UNUSED(x) (void)(x)
using namespace std;
typedef long long ll;
template<typename T>inline T read(T x) {
    int f = 1; char ch = getchar();
    for (; ch < '0' || ch>'9'; ch = getchar())  if (ch == '-')  f = -1;
    for (; ch >= '0' && ch <= '9'; ch = getchar())  x = (x << 1) + (x << 3) + ch - '0';
    return x * f;
}
struct FLOWER {
    const static int N = 5e2;
    struct edge {
        int u, v, w;
        edge(int u = 0, int v = 0, int w = 0) :u(u), v(v), w(w) {}
    }g[N][N];
    int lab[N], slack[N], F[N], pa[N], flower_from[N][N], S[N], Vis[N], match[N];
    int n, m, n_x;
    vector<int>flower[N];
    deque<int>q;
    int dist(const edge& e) { return lab[e.u] + lab[e.v] - (g[e.u][e.v].w << 1); }
    void update_slack(int u, int x) {
        if (!slack[x] || dist(g[u][x]) < dist(g[slack[x]][x]))
            slack[x] = u;
    }
    void set_slack(int x) {
```

```
        slack[x] = 0;
        for (int u = 1; u <= n; u++)
            if (g[u][x].w > 0 && F[u] != x && !S[F[u]])
                update_slack(u, x);
    }
    void push(int x) {
        if (x <= n) return q.push_back(x);
        for (int i = 0; i < (int)flower[x].size(); i++)
            push(flower[x][i]);
    }
    void set_st(int x, int b) {
        F[x] = b;
        if (x <= n)   return;
        for (auto p : flower[x])
            set_st(p, b);
    }
    int get_pr(int x, int v) {
        int pr = find(flower[x].begin(), flower[x].end(), v) - flower[x].begin();
        if (pr & 1) {
            reverse(flower[x].begin() + 1, flower[x].end());
            return (int)flower[x].size() - pr;
        }
        else   return pr;
    }
    void set_match(int x, int y) {
        match[x] = g[x][y].v;
        if (x <= n)   return;
        edge e = g[x][y];
        int xr = flower_from[x][e.u], pr = get_pr(x, xr);
        for (int i = 0; i < pr; i++)
            set_match(flower[x][i], flower[x][i ^ 1]);
        set_match(xr, y);
        rotate(flower[x].begin(), flower[x].begin() + pr, flower[x].end());
    }
    void augment(int x, int y) {
        int temp = F[match[x]];
        set_match(x, y);
        if (!temp)  return;
        set_match(temp, F[pa[temp]]);
        augment(F[pa[temp]], temp);
    }
    int get_lca(int x, int y) {
        static int t = 0;
        for (++t; x || y; swap(x, y)) {
            if (!x) continue;
            if (Vis[x] == t)  return x;
            Vis[x] = t;
            x = F[match[x]];
```

```
87              if (x)  x = F[pa[x]];
88          }
89          return 0;
90      }
91      void add_blossom(int u, int lca, int v) {
92          int b = n + 1;
93          while (b <= n_x && F[b])    b++;
94          if (b > n_x)  ++n_x;
95          lab[b] = S[b] = 0;
96          match[b] = match[lca];
97          flower[b].clear();
98          flower[b].emplace_back(lca);
99          for (int x = u, y; x != lca; x = F[pa[y]]) {
100             flower[b].emplace_back(x);
101             flower[b].emplace_back(y = F[match[x]]);
102             push(y);
103         }
104         reverse(flower[b].begin() + 1, flower[b].end());
105         for (int x = v, y; x != lca; x = F[pa[y]]) {
106             flower[b].emplace_back(x);
107             flower[b].emplace_back(y = F[match[x]]);
108             push(y);
109         }
110         set_st(b, b);
111         for (int x = 1; x <= n_x; x++)  g[b][x].w = g[x][b].w = 0;
112         for (int x = 1; x <= n; x++)    flower_from[b][x] = 0;
113         for (int i = 0; i < (int)flower[b].size(); i++) {
114             int temp = flower[b][i];
115             for (int x = 1; x <= n_x; x++)
116                 if (g[b][x].w == 0 || dist(g[temp][x]) < dist(g[b][x]))
117                     g[b][x] = g[temp][x], g[x][b] = g[x][temp];
118             for (int x = 1; x <= n; x++)
119                 if (flower_from[temp][x])
120                     flower_from[b][x] = temp;
121         }
122         set_slack(b);
123     }
124     void expand_blossom(int b) {
125         for (int i = 0; i < (int)flower[b].size(); i++)
126             set_st(flower[b][i], flower[b][i]);
127         int xr = flower_from[b][g[b][pa[b]].u], pr = get_pr(b, xr);
128         for (int i = 0; i < pr; i += 2) {
129             int xs = flower[b][i], xns = flower[b][i + 1];
130             pa[xs] = g[xns][xs].u;
131             S[xs] = 1, S[xns] = 0;
132             slack[xs] = 0, set_slack(xns);
133             push(xns);
134         }
```

```
135         S[xr] = 1, pa[xr] = pa[b];
136         for (int i = pr + 1; i < (int)flower[b].size(); i++) {
137             int xs = flower[b][i];
138             S[xs] = -1, set_slack(xs);
139         }
140         F[b] = 0;
141     }
142     bool on_found_edge(const edge& e) {
143         int Fu = F[e.u], Fv = F[e.v];
144         if (!~S[Fv]) {
145             pa[Fv] = e.u, S[Fv] = 1;
146             int nu = F[match[Fv]];
147             slack[Fv] = slack[nu] = 0;
148             S[nu] = 0, push(nu);
149         }
150         if (!S[Fv]) {
151             int lca = get_lca(Fu, Fv);
152             if (!lca) {
153                 augment(Fu, Fv);
154                 augment(Fv, Fu);
155                 return true;
156             }
157             else   add_blossom(Fu, lca, Fv);
158         }
159         return false;
160     }
161     bool matching() {
162         memset(S, 0xff, sizeof(S));
163         memset(slack, 0x00, sizeof(slack));
164         q.clear();
165         for (int i = 1; i <= n_x; i++) {
166             if (F[i] == i && !match[i]) {
167                 pa[i] = S[i] = 0;
168                 push(i);
169             }
170         }
171         if (q.empty())  return false;
172         while (true) {
173             while (!q.empty()) {
174                 int u = q.front(); q.pop_front();
175                 if (S[F[u]] == 1)   continue;
176                 for (int v = 1; v <= n; v++) {
177                     if (g[u][v].w > 0 && F[u] != F[v]) {
178                         if (dist(g[u][v]) == 0) {
179                             if (on_found_edge(g[u][v]))
180                                 return true;
181                         }
182                         else update_slack(u, F[v]);
```

```
                    }
                }
            }
            int d = numeric_limits<int>::max();
            for (int i = n + 1; i <= n_x; i++)
                if (F[i] == i && S[i] == 1)
                    d = min(d, lab[i] >> 1);
            for (int i = 1; i <= n_x; i++) {
                if (F[i] == i && slack[i]) {
                    if (!~S[i]) d = min(d, dist(g[slack[i]][i]));
                    if (!S[i])  d = min(d, dist(g[slack[i]][i]) >> 1);
                }
            }
            for (int i = 1; i <= n; i++) {
                if (!S[F[i]]) {
                    if (lab[i] <= d)  return false;
                    lab[i] -= d;
                }
                if (S[F[i]] == 1) lab[i] += d;
            }
            for (int i = n + 1; i <= n_x; i++) {
                if (F[i] == i) {
                    if (!S[F[i]])       lab[i] += d << 1;
                    if (S[F[i]] == 1)   lab[i] -= d << 1;
                }
            }
            q.clear();
            for (int i = 1; i <= n_x; i++)
                if (F[i] == i && slack[i] && F[slack[i]] != i && !dist(g[slack[i]][i]))
                    if (on_found_edge(g[slack[i]][i]))
                        return true;
            for (int i = n + 1; i <= n_x; i++)
                if (F[i] == i && S[i] == 1 && !lab[i])
                    expand_blossom(i);
        }
        return false;
    }
    void weight_blossom() {
        memset(match, 0x00, sizeof(match));
        n_x = n;
        for (int i = 0; i <= n; i++)
            F[i] = i, flower[i].clear();
        int wMax = 0;
        for (int i = 1; i <= n; i++) {
            for (int j = 1; j <= n; j++) {
                flower_from[i][j] = i == j ? i : 0;
                wMax = max(wMax, g[i][j].w);
            }
```

```
        }
        for (int i = 1; i <= n; i++)  lab[i] = wMax;
        while (matching());
        return;
    }
}fl;    //带花树
struct GRAPH {
    const static int N = 5e2, M = N << 1;
    int pre[(M << 1) + 10], now[N + 10], child[(M << 1) + 10], Degree[N + 10], tot;
    bool Vis[(M << 1) + 10];
    GRAPH() { tot = 1; }
    void link(int x, int y) { pre[++tot] = now[x], now[x] = tot, child[tot] = y, Degree[y]++; }
    void connect(int x, int y) { link(x, y), link(y, x); }
    void Dfs(vector<int>& res, int x) { //Dfs暴力寻找欧拉回路
        res.push_back(x);
        if (res.size() > 1 && res.front() == res.back())    return;
        for (int p = now[x]; p; p = pre[p]) {
            if (Vis[p]) continue;
            Vis[p] = Vis[p ^ 1] = 1;
            int son = child[p];
            Degree[x]--, Degree[son]--;
            Dfs(res, son);
            if (res.front() == res.back())  return;
        }
    }
    void euler(int x, vector<int>& list) {  //欧拉回路
        vector<int>res;
        Dfs(res, x);
        for (auto p : res) {
            if (Degree[p])  euler(p, list);
            else    list.push_back(p);
        }
    }
}Graph;
struct Line {
    int x, y, v;
    Line(int x = 0, int y = 0, int v = 0) :x(x), y(y), v(v) {}
    bool operator<(const Line& ots)const { return v < ots.v; }
};
int Christofides(const vector<vector<int>>& Dis, int n) {
    vector<Line>lines;
    vector<int>Fa, Degree(n + 1);
    function<int(int)>Find = [&](int x) {return x == Fa[x] ? x : Fa[x] = Find(Fa[x]); };
    for (int i = 0; i <= n; i++)
        Fa.emplace_back(i);
    int Max = numeric_limits<int>::min();
    for (int i = 0; i <= n; i++) {
        for (int j = 0; j <= n; j++) {
```

```
279             lines.emplace_back(Line(i, j, Dis[i][j]));
280             Max = max(Max, Dis[i][j]);
281         }
282     }
283     sort(lines.begin(), lines.end());
284     for (auto line : lines) {   //生成MST，并建图
285         int Fx = 0, Fy = 0;
286         if ((Fx = Find(line.x)) != (Fy = Find(line.y))) {
287             Fa[Fx] = Fy;
288             Degree[line.x]++;
289             Degree[line.y]++;
290             Graph.connect(line.x, line.y);
291         }
292     }
293     int list[n + 1], m = 0;
294     for (int i = 0; i <= n; i++)
295         if (Degree[i] & 1)
296             list[++m] = i;
297     for (int i = 1; i <= m; i++)    //找到奇度数点加入带花树
298         for (int j = 1; j <= m; j++)
299             fl.g[i][j] = FLOWER::edge(i, j, Max - Dis[list[i]][list[j]]);
300     fl.n = m;
301     fl.weight_blossom();    //求最大带权匹配
302     for (int i = 1; i <= m >> 1; i++)
303         Graph.connect(list[i], list[fl.match[i]]);  //最大匹配与MST加入同一个图
304     vector<int>Euler, Hamilton;
305     Graph.euler(0, Euler);  //找到新图的欧拉回路
306     map<int, bool>Vis;
307     for (auto p : Euler) {  //通过欧拉回路生成哈密顿回路
308         if (!Vis[p])    Hamilton.emplace_back(p);
309         Vis[p] = true;
310     }
311     Hamilton.emplace_back(Hamilton.front());
312     int Ans = 0;
313     for (int i = 0; i < (int)Hamilton.size(); i++)
314         Ans += Dis[Hamilton[i]][Hamilton[i + 1]];
315     return Ans;
316 }
317 int main() {
318     int n = read(0);
319     vector<vector<int>>Dis;
320     for (int i = 0; i <= n; i++) {
321         vector<int>temp;
322         for (int j = 0; j <= n; j++)
323             temp.emplace_back(read(0));
324         Dis.emplace_back(temp);
325     }
326     printf("%d\n", Christofides(Dis, n));
```

```
    return 0;
}
```

TSP K-OPT 算法

```
/*program from Wolfycz*/
#include<map>
#include<cmath>
#include<ctime>
#include<queue>
#include<cstdio>
#include<vector>
#include<cstring>
#include<limits.h>
#include<iostream>
#include<algorithm>
#define MK make_pair
#define sqr(x) ((x)*(x))
#define pii pair<int,int>
#define UNUSED(x) (void)(x)
using namespace std;
typedef long long ll;
template<typename T>inline T read(T x) {
    int f = 1; char ch = getchar();
    for (; ch < '0' || ch>'9'; ch = getchar())  if (ch == '-')  f = -1;
    for (; ch >= '0' && ch <= '9'; ch = getchar())  x = (x << 1) + (x << 3) + ch - '0';
    return x * f;
}
int K_OPT(vector<vector<int>>& Dis, int n) {
    srand(time(NULL));
    vector<int>list(1, 0);
    for (int i = 1; i <= n; i++)  list.emplace_back(i);
    list.emplace_back(0);
    random_shuffle(list.begin() + 1, list.end() - 1);   //生成随机序列
    int Limit = 1e3, Times = 0;
    while (true) {
        if (Times > Limit)    break;
        int l = rand() % n, r = rand() % n; //随机生成两个断点
        while (l == r)  //断点不能相同
            l = rand() % n, r = rand() % n;
        if (l > r)    swap(l, r);
        l++, r++;
        int Saving = Dis[list[l - 1]][list[l]] + Dis[list[r]][list[r + 1]] - Dis[list[l - 1]][list[r
            ]] - Dis[list[l]][list[r + 1]];
        if (Saving > 0) {   //判断是否交换
            Times = 0;
            reverse(list.begin() + l, list.begin() + r + 1);
        }
        else Times++;
```

```
    }
    int Ans = 0;
    for (int i = 1; i < (int)list.size(); i++)
        Ans += Dis[list[i - 1]][list[i]];
    return Ans;
}
int main() {
    int n = read(0);
    vector<vector<int>>Dis;
    for (int i = 0; i <= n; i++) {
        vector<int>temp;
        for (int j = 0; j <= n; j++)
            temp.emplace_back(read(0));
        Dis.emplace_back(temp);
    }
    printf("%d\n", K_OPT(Dis, n));
    return 0;
}
```

TSP 遗传算法

```
/*program from Wolfycz*/
#include<map>
#include<cmath>
#include<ctime>
#include<queue>
#include<cstdio>
#include<vector>
#include<cstring>
#include<limits.h>
#include<iostream>
#include<algorithm>
#define MK make_pair
#define sqr(x) ((x)*(x))
#define pii pair<int,int>
#define UNUSED(x) (void)(x)
using namespace std;
typedef long long ll;
template<typename T>inline T read(T x) {
    int f = 1; char ch = getchar();
    for (; ch < '0' || ch>'9'; ch = getchar())  if (ch == '-')  f = -1;
    for (; ch >= '0' && ch <= '9'; ch = getchar())  x = (x << 1) + (x << 3) + ch - '0';
    return x * f;
}
vector<int>randperm(int n) {
    vector<int>list;
    for (int i = 0; i < n; i++) list.emplace_back(i);
    random_shuffle(list.begin(), list.end());
    return list;
```

```
}
int GA(vector<vector<int>>& Dis, int n) {
    srand(time(NULL));
    int W = 50, G = 10000;  //种群大小，迭代次数
    double pm = 0.1;    //变异概率
    vector<vector<int>>J; J.resize(W);
    for (int k = 0; k < W; k++) {   //通过最小环法生成W个初始染色体
        vector<int>c = randperm(n), _c;
        _c.emplace_back(0);
        for (auto p : c)    _c.push_back(p + 1);
        _c.emplace_back(0);
        J[k].resize(_c.size());
        while (true) {
            bool Flag = 0;
            for (int i = 1; i < n; i++)
                for (int j = i + 1; j <= n; j++)
                    if (Dis[_c[i - 1]][_c[j]] + Dis[_c[i]][_c[j + 1]] < Dis[_c[i - 1]][_c[i]] + Dis[
                        _c[j]][_c[j + 1]])
                        reverse(_c.begin() + i, _c.begin() + j + 1), Flag |= true;
                            //如果更优则交换，直到不能交换为止
            if (!Flag) {
                _c.back() = n + 1;
                for (int i = 0; i < (int)_c.size(); i++)
                    J[k][_c[i]] = i;    //编码
                break;
            }
        }
    }
    double Ans = 0;
    for (int k = 0; k < G; k++) {
        vector<vector<int>>A = J;   //染色体交换种群
        vector<int>c = randperm(W);
        for (int i = 0; i < W; i += 2) {    //随机匹配，随机找断点
            int F = rand() % n + 1;
            for (int j = F; j <= n; j++)
                swap(A[c[i]][j], A[c[i + 1]][j]);
        }
        vector<int>by;  //随机找到一些变异的染色体编号
        while (by.empty())
            for (int i = 0; i < W; i++)
                if (1.0 * rand() / RAND_MAX < pm)
                    by.emplace_back(i);
        vector<vector<int>>B;
        for (auto p : by)   B.emplace_back(A[p]);
        for (int j = 0; j < (int)by.size(); j++) {
            vector<int>bw;
            while (true) {
                bw.clear();
```

```
                bw.emplace_back(0);
                bw.emplace_back(n + 2);
                for (int i = 0; i < 3; i++) //寻找断点
                    bw.emplace_back(rand() % n + 1);
                bool Flag = 0;
                sort(bw.begin(), bw.end());
                Flag |= (bw[0] == bw[1]) | (bw[1] == bw[2]);
                if (!Flag)  break;
            }
            vector<int>temp;    //染色体变异（自交换）
            for (int i = bw[0]; i < bw[1]; i++)     temp.emplace_back(B[j][i]);
            for (int i = bw[2]; i < bw[3]; i++)     temp.emplace_back(B[j][i]);
            for (int i = bw[1]; i < bw[2]; i++)     temp.emplace_back(B[j][i]);
            for (int i = bw[3]; i < bw[4]; i++)     temp.emplace_back(B[j][i]);
            B[j] = temp;
        }
        vector<vector<int>>G;   //新的种群 = 原种群 + 染色体交换 + 染色体变异
        for (auto p : J)    G.emplace_back(p);
        for (auto p : A)    G.emplace_back(p);
        for (auto p : B)    G.emplace_back(p);
        vector<int>VF;
        for (auto g : G) {
            vector<pair<int, int>>Index1;
            for (int i = 0; i < (int)g.size(); i++)
                Index1.emplace_back(MK(g[i], i));
            sort(Index1.begin(), Index1.end()); //解码
            int res = 0;
            for (int i = 1; i < (int)Index1.size(); i++)
                res += Dis[Index1[i - 1].second][Index1[i].second];
            VF.emplace_back(res);   //记录每个个体的代价
        }
        vector<pair<int, int>>Index2;   //给所有个体按照代价排序
        for (int i = 0; i < (int)VF.size(); i++)
            Index2.emplace_back(MK(VF[i], i));
        sort(Index2.begin(), Index2.end());
        J.clear();
        for (int i = 0; i < W; i++) //优胜劣汰，保留W个个体进行下一次迭代
            J.emplace_back(G[Index2[i].second]);
        Ans = Index2[0].first;  //记录最优的个体
    }
    return Ans;
}
int main() {
    int n = read(0);
    vector<vector<int>>Dis;
    for (int i = 0; i <= n; i++) {
        vector<int>temp;
        for (int j = 0; j <= n; j++)
```

```
            temp.emplace_back(read(0));
        Dis.emplace_back(temp);
    }
    printf("%d\n", GA(Dis, n));
    return 0;
}
```

TSP 模拟退火算法

```
/*program from Wolfycz*/
#include<map>
#include<cmath>
#include<ctime>
#include<queue>
#include<cstdio>
#include<vector>
#include<cstring>
#include<limits.h>
#include<iostream>
#include<algorithm>
#define MK make_pair
#define sqr(x) ((x)*(x))
#define pii pair<int,int>
#define UNUSED(x) (void)(x)
using namespace std;
typedef long long ll;
template<typename T>inline T read(T x) {
    int f = 1; char ch = getchar();
    for (; ch < '0' || ch>'9'; ch = getchar())  if (ch == '-')  f = -1;
    for (; ch >= '0' && ch <= '9'; ch = getchar())  x = (x << 1) + (x << 3) + ch - '0';
    return x * f;
}
vector<int>randperm(int n) {
    vector<int>list;
    for (int i = 1; i <= n; i++) list.emplace_back(i);
    random_shuffle(list.begin(), list.end());
    return list;
}
int SA(vector<vector<int>>& Dis, int n) {
    srand(time(NULL));
    vector<int>list;
    int Min = numeric_limits<int>::max(), L = 1e3;
    for (int i = 1; i <= L; i++) {  //随机生成L个随机序列，取最优的一个作为初始序列
        vector<int>temp = randperm(n);
        temp.insert(temp.begin(), 0);
        temp.emplace_back(0);
        int res = 0;
        for (int j = 1; j < (int)temp.size(); j++)
            res += Dis[temp[j - 1]][temp[j]];
```

```
        if (res < Min) {
            Min = res;
            list = temp;
        }
    }
    double T = 1, a = 0.99999, e = 1e-30;   //T是初始问题，a是降温系数，e是临界值
    while (T >= e) {    //模拟退火的交换采用2-opt算法
        int l = rand() % n, r = rand() % n;
        while (l == r)
            l = rand() % n, r = rand() % n;
        if (l > r)    swap(l, r);
        l++, r++;
        int Delta = Dis[list[l - 1]][list[r]] + Dis[list[l]][list[r + 1]] - Dis[list[l - 1]][list[l
            ]] - Dis[list[r]][list[r + 1]];
        if (Delta < 0 || 1.0 * rand() / RAND_MAX <= exp(-Delta / T))    //如果更优，或者更劣但是有概
            率进行旋转
            reverse(list.begin() + l, list.begin() + r + 1);
        T *= a; //降温
    }
    int Ans = 0;
    for (int i = 1; i < (int)list.size(); i++)
        Ans += Dis[list[i - 1]][list[i]];
    return Ans;
}
int main() {
    int n = read(0);
    vector<vector<int>>Dis;
    for (int i = 0; i <= n; i++) {
        vector<int>temp;
        for (int j = 0; j <= n; j++)
            temp.emplace_back(read(0));
        Dis.emplace_back(temp);
    }
    printf("%d\n", SA(Dis, n));
    return 0;
}
```