《计算机算法设计与分析》第五章作业

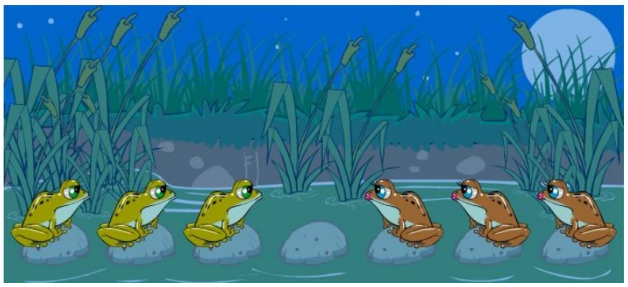
姓名：任宇　　学号：33920212204567

青蛙换位游戏

问题描述：

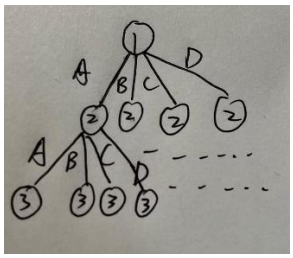
在7块石头上，有绿、红青蛙各3只，绿青蛙在左边面向右，红青蛙在右边面向左，中间是个空位。每次移动一只青蛙，青蛙只能往前跳一步，或隔着一只青蛙跳一步，将左边的绿青蛙移动到右边，将右边的红青蛙移动到左边。

http://www.yjlab.com/other/play01/



答：

解空间定义：分析题目，可以发现只有在空位左右各两只的总共四只青蛙能够进行跳跃操作。因此解空间是一个四叉树，如图：



剪枝：

- 不允许青蛙向其背对的方向跳。
- 不允许青蛙跳到一个已经被其他青蛙占据的位置或跳出界限。
- 如果当前移动导致后续没有任何合法移动可行，则停止当前分支的进一步搜索。

终止条件：左右各三只青蛙都已交换位置。

算法设计：

1. 初始化记录位置状态的数组 a，a[i]=0 表示空格，a[i]=1 表示青蛙面朝右边，a[i]=2 表示青蛙面朝左边，初始状态为 1110222 。
2. 选择空格左右四个石头进行搜索，每步记录改变后的位置状态，并记录每次操作的石头位置，逐层深度搜索。
3. 使用剪枝策略检查当前分支，如果不满足，则剪枝并回溯，还原位置状态数组和操作数组，递归搜索其他选择。
4. 当位置状态数组为 2220111 时，找到一个可行解，返回。

时间复杂度分析：每一层跳跃至多有 4 种可能，若需要 n 次操作才能完成交换，则时间复杂度为 $O(4^n)$，空间复杂度为 O(n)。

算法分析题 5-6

5-6 旅行售货员问题的上界函数。

设 G 是一个有 n 个顶点的有向图，从顶点 i 发出的边的最小费用记为 $\min(i)$。

（1）证明图 G 的所有前缀为 $x[1:i]$的旅行售货员回路的费用至少为：

$$\sum_{j=2}^{i} a(x_{j-1}, x_j) + \sum_{j=i}^{n} \min(x_j)$$

式中，$a(u, v)$是边(u, v)的费用。

（2）利用上述结论设计一个高效的上界函数，重写旅行售货员问题的回溯法，并与主教材中的算法进行比较。

答： （1）前缀为 x[1,i]的旅行商回路可以表示为 n 个顶点的一个排列，即（x[1],x[2],……,x[i], π(i+1), π(i+2),……, π(n))。这个回路的费用为：

$$h(\pi) = \sum_{j=2}^{i} a(x_{j-1}, x_j) + a(x_i, \pi(i+1)) + \sum_{j=i+1}^{n} a(\pi(j), \pi(j \bmod n + 1))$$

由此公式可知：

$$h(\pi) \geq \sum_{j=2}^{i} a(x_{j-1}, x_j) + \min(x_i) + \sum_{j=i+1}^{n} min(\pi(j))$$

$$h(\pi) \geq \sum_{j=2}^{i} a(x_{j-1}, x_j) + \sum_{j=i}^{n} min(x_i)$$

（2）解空间定义：旅行售货员问题的解空间是一颗排列树，开始时 x=[1,2,……,n],则对应的排列树由 x[1:n]的所有排列构成。

可以发现，若从顶点 i 发出的边的最大费用记为 max(i)，最小费用记为 min(i)，那么可以发现旅行售货员回路的费用不超过$\sum_{i=1}^{n} max(x_i) + 1$,同时也不低于$\sum_{i=1}^{n} min(x_i)$，所以可以先对图 G 进行简单遍历，计算出这两个值以用来估计未完成部分的上界。在算法的每一步中更新它，帮助算法避免对那些不可能产生比当前已知解更好的路径的分支进行不必要的搜索。

与主教材的算法相比，我们在每次搜索前简单的遍历图以估计未完成部分的上界，对原有的剪枝方案进行了优化，可以降低一定的时间复杂度。

算法实现题 5-1

5-1　子集和问题。

问题描述：子集和问题的一个实例为<S, c>。其中，$S=\{x_1, x_2, \cdots, x_n\}$是一个正整数的集合，$c$ 是一个正整数。子集和问题判定是否存在 S 的一个子集 $S1$，使得 $\sum_{x\in S1} x=c$。试设计一个解子集和问题的回溯法。

算法设计：对于给定的正整数的集合 $S=\{x_1, x_2, \cdots, x_n\}$和正整数 c，计算 S 的一个子集 S_1，使得 $\sum_{x\in S_1} x=c$。

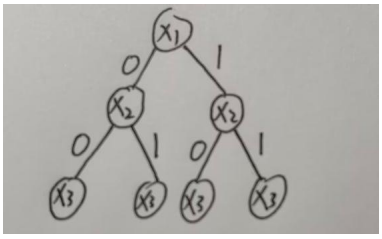
数据输入：由文件 input.txt 提供输入数据。文件第 1 行有 2 个正整数 n 和 c，n 表示 S 的大小，c 是子集和的目标值。接下来的 1 行中，有 n 个正整数，表示集合 S 中的元素。

结果输出：将子集和问题的解输出到文件 output.txt。当问题无解时，输出“No Solution!”。

输入文件示例	输出文件示例
input.txt	output.txt
5 10	2 2 6
2 2 6 5 4	

答：

解空间定义：解空间是所有可能数字的集合，是子集树问题，对于一个数字，可以选择加入集合，也可以选择不加入集合，即解空间可以表示为一颗高度为 n+1 的完全二叉树，如图所示：



剪枝：如果当前部分解已经无法满足条件，就剪掉这个分支。在本问题中，剪枝条件是当前子集和大于目标和 c，或者已经考虑了所有元素。

终止条件：找到一个子集其和等于 c，或者所有可能的子集都已探索完毕。

算法设计：

5. 开始时，子集为空，总和为 0。
6. 对于每个数字，有两种选择，加入子集或不加入。
7. 递归探索两种选择，每次递归时更新当前和以及当前索引。
8. 如果当前和等于目标和 c，记录解并返回。
9. 如果当前和大于目标和 c，回溯到上一层。
10. 如果所有可能的子集都探索完毕，返回。

时间复杂度分析：在最坏的情况下，算法将探索所有可能的子集。对于每个元素，我们都有两种选择(包含或不包含在子集中)，所以时间复杂度为 $O(2^n)$,其中 n 是集合中元素的数量。

算法实现题 5-3

5-3　最小重量机器设计问题。

问题描述：设某一机器由 n 个部件组成，每种部件都可以从 m 个不同的供应商处购得。设 w_{ij} 是从供应商 j 处购得的部件 i 的重量，c_{ij} 是相应的价格。试设计一个算法，给出总价格不超过 c 的最小重量机器设计。

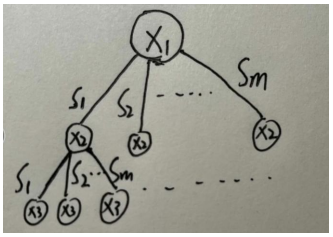
算法设计：对于给定的机器部件重量和机器部件价格，计算总价格不超过 d 的最小重量机器设计。

数据输入：由文件 input.txt 给出输入数据。第一行有 3 个正整数 n、m 和 d。接下来的 $2n$ 行，每行 n 个数。前 n 行是 c，后 n 行是 w。

结果输出：将计算的最小重量及每个部件的供应商输出到文件 output.txt。

答：

解空间定义：对于每一个部件，它可以选择分配给 m 个供应商中的任何一个，所以问题的解空间树是一颗 n+1 层的 m 叉树：



限界函数设计：限界函数用来判断当前分配是否有可能比已知的最佳方案更优。如果当前总重量已经超过了已知的最小总重量时，则不需要继续探索这个方案。

剪枝：题目中提到一个约束条件，即总价格不能超过 c，所以，如果当前方案的总价格超过 c 时，剪枝。在满足这个约束的前提下，如果当前分配的总重量超过已知最小总重量时，剪枝。

终止条件：如果找到了一个分配方案，其总重量低于或等于目前已知的最佳方案，那么记录这个方案并更新最小总重量。如果已经考虑了所有分配并且没有找到更好的方案，则回溯。

算法设计：

1. 初始化总价格为 0 和最小总重量为无穷大。
2. 从第一个部件开始，尝试分配给每个供应商，并计算总价格和总重量。
3. 如果当前分配的总价格小于 c，递归地对下一个部件进行分配。
4. 如果当前分配的总价格超过 c，剪枝并回溯。
5. 如果所有部件都已分配，并且当前总重量小于或等于最小总重量，更新最小总重量并记录分配方案。
6. 如果当前部件分配的总重量已超过最小总重量，剪枝并回溯。
7. 继续这个过程，直到所有可能的分配都已经探索。

时间复杂度分析：每个部件有 m 种可能的分配方式，所以时间复杂度为 $O(m^n)$，其中 m 是供应商数量，n 是部件数量。

算法实现题 5-6

5-6　无和集问题。

问题描述：设 S 是正整数集合。S 是一个无和集，当且仅当 $x, y \in S$ 蕴含 $x+y \notin S$。对于任意正整数 k，如果可将 $\{1, 2, \cdots, k\}$ 划分为 n 个无和子集 $S_1, S_2, \cdots, S_n$，则称正整数 k 是 n 可分的。记 $F(n)=\max\{k \mid k$ 是 n 可分的$\}$。试设计一个算法，对任意给定的 n，计算 $F(n)$的值。

算法设计：对任意给定的 n，计算 $F(n)$的值。

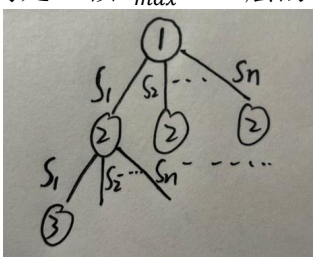
数据输入：由文件 input.txt 给出输入数据。第 1 行有 1 个正整数 n。

结果输出：将计算的 $F(n)$的值以及 $\{1, 2, \cdots, F(n)\}$的一个 n 划分输出到文件 output.txt。文件的第 1 行是 $F(n)$的值。接下来的 n 行，每行是一个无和子集 S_i。

输入文件示例	输出文件示例
input.txt	output.txt
2	8
	1 2 4 8
	3 5 6 7

答：

解空间定义：对于每一个集合元素，它可以选择分配给 n 个无和子集当中的任何一个，所以问题的解空间树是一颗$k_{max}+1$ 层的 n 叉树，是个子集树问题：



剪枝：对于本问题，如果加入集合中后，当前集合中存在两个元素之和等于集合中的另一个元素，也就是当前集合加入新元素后不再满足无和集的要求，那么就应该停止进一步扩展这个节点，并回溯到上一个节点。

终止条件：如果当前元素无法加入任何一个无和子集，那么记录当前深度并返回，如果当前深度大于记录的 F(n)，那么就更新 F(n)为当前深度，直到遍历完所有可能解，得到最后的 F(n)。

算法设计：

1. 从空集开始，逐渐添加元素构建解集合。
2. 对于每个新元素，检查加入该元素后集合是否仍满足无和集条件，如果不满足，则舍弃加入当前无和子集，剪枝并回溯，尝试加入下一个无和子集。
3. 如果新元素无法加入任何一个无和子集，记录当前深度并返回，如果当前深度大于记录的 F(n)，那么就更新 F(n)为当前深度。
4. 继续这个过程，直到所有可能的方案都已经探索。

时间复杂度分析：对于不同的 n，解空间的层数并不确定，在最坏的情况下，需要遍历 $O(n^{k_{max}+1})$。

算法实现题 5-13

5-13　工作分配问题。

问题描述：设有 n 件工作分配给 n 个人。将工作 i 分配给第 j 个人所需的费用为 c_{ij}。试设计一个算法，为每个人都分配 1 件不同的工作，并使总费用达到最小。

算法设计：设计一个算法，对于给定的工作费用，计算最佳工作分配方案，使总费用达到最小。

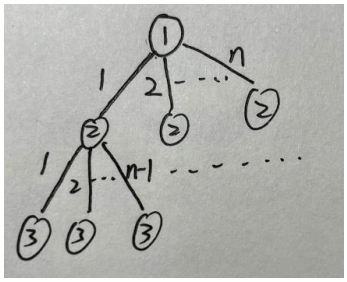
数据输入：由文件 input.txt 给出输入数据。第 1 行有 1 个正整数 n（$1 \leqslant n \leqslant 20$）。接下来的 n 行，每行 n 个数，表示工作费用。

结果输出：将计算的最小总费用输出到文件 output.txt。

输入文件示例	输出文件示例
input.txt	output.txt
3	9
10 2 3	
2 3 4	
3 4 5	

答：

解空间定义：分析题目可以知道，本问题是一个典型的排列树问题，第一层有 n 个选择，逐层递减，直到叶子结点：



限界函数设计：限界函数用于计算当前分配方案的总费用，并与已知的最小费用进行比较。如果当前总费用已经超过已知的最小费用，则没有继续探索该分支的必要，因为它不可能产生更优的解。

剪枝：剪枝基于限界函数的结果来决定是否剪枝。如果当前节点的总费用已经超过已知的最优解，则应被剪枝。

终止条件：遍历整个排列树中所有可能的解，找到最优解。

算法设计：

1. 初始化一个数组，用于记录工作分配的情况。
2. 从第一个工作开始，为每个工作尝试分配给不同的人。
3. 使用限界函数和剪枝函数检查当前分配是否可行。
4. 如果当前分配方案可行，递归地为下一个工作分配人员。
5. 如果达到一个完整的分配方案，比较并更新最小总费用。
6. 回溯到上一步，尝试不同的分配方案。
7. 重复步骤 2 到 6，直到探索完所有可能的分配方案。

时间复杂度分析：最坏情况的时间复杂性为 O(n!)，但是剪枝可以减少不必要的搜索，优化搜索时间。

算法实现题 5-16

问题描述：给定 n 个正整数和 4 个运算符+、-、*、/，且运算符无优先级，如 2+3×5=25。对于任意给定的整数 m，试设计一个算法，用以上给出的 n 个数和 4 个运算符，产生整数 m，且用的运算次数最少。给出的 n 个数中每个数最多只能用 1 次，但每种运算符可以任意使用。

算法设计：对于给定的 n 个正整数，设计一个算法，用最少的无优先级运算次数产生整数 m。

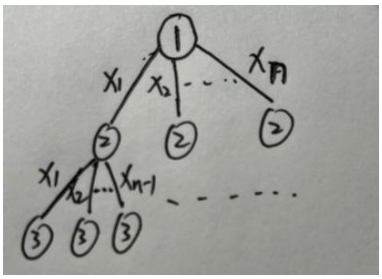
数据输入：由文件 input.txt 给出输入数据。第 1 行有 2 个正整数 n 和 m。第 2 行是给定的用于运算的 n 个正整数。

结果输出：将计算的产生整数 m 的最少无优先级运算次数以及最优无优先级运算表达式输出到文件 output.txt。

输入文件示例	输出文件示例
input.txt	output.txt
5 25	2
5 2 3 6 7	2+3*5

答：

解空间定义：分析题目可以知道，本问题是一个排列树问题，第一层有 n 个选择，逐层递减，直到叶子结点：



限界函数设计：如果当前层数已经超过已知的最小运算次数，则没有继续探索该分支的必要，因为它不可能产生更优的解。

剪枝：剪枝基于限界函数的结果来决定是否剪枝。如果当前节点的运算次数已经超过已知的最小运算次数，或者当前节点对应的数字已被选择，则应被剪枝。

终止条件：遍历整个排列树中所有可能的解，找到最优解。

算法设计：

1. 从空序列开始。
2. 选择一个尚未使用的数字添加到当前序列。
3. 使用限界函数和剪枝策略检查当前序列。
4. 如果当前序列可能导致解，那么递归地继续添加新的数字。
5. 如果当前序列已经达到目标数 m，记录下运算次数，并与当前最小运算次数比较，更新最小次数。
6. 如果当前序列不可能得到解，或者运算次数已经超过当前最小值，则回溯到上一步。
7. 重复步骤 2 到 6，直到探索完所有可能的序列。

时间复杂度分析：最坏情况的时间复杂性为 O(n!)，但是限界函数和剪枝策略可以减少不必要的搜索，优化搜索时间。

算法实现题 5-20

5-20　部落卫队问题。

问题描述：原始部落 byteland 中的居民们为了争夺有限的资源，经常发生冲突。几乎每个居民都有他的仇敌。部落酋长为了组织一支保卫部落的队伍，希望从部落的居民中选出最多的居民入伍，并保证队伍中任何 2 个人都不是仇敌。

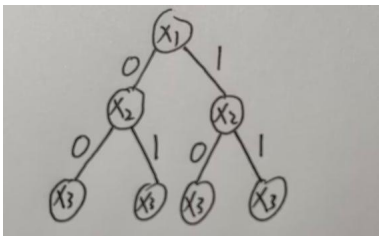
算法设计：给定 byteland 部落中居民间的仇敌关系，计算组成部落卫队的最佳方案。

数据输入：由文件 input.txt 给出输入数据。第 1 行有 2 个正整数 n 和 m，表示 byteland 部落中有 n 个居民，居民间有 m 个仇敌关系。居民编号为 $1, 2, \cdots, n$。接下来的 m 行中，每行有 2 个正整数 u 和 v，表示居民 u 与居民 v 是仇敌。

结果输出：将计算的部落卫队的最佳组建方案输出到文件 output.txt。文件的第 1 行是部落卫队的人数；第 2 行是卫队组成 x_i（$1 \leqslant i \leqslant n$）。$x_i$=0 表示居民 i 不在卫队中，x_i=1 表示居民 i 在卫队中。

答：

解空间定义：分析问题，发现是一个子集树问题：对于每一个居民，可以选择加入卫队，也可以选择不加入卫队，即解空间可以表示为一颗高度为 n+1 的完全二叉树，如图所示：



剪枝：

1. 如果添加一个新的居民到当前卫队会导致与卫队中已有成员存在仇敌关系，则这条分支不再继续。
2. 如果即使将所有剩余的居民都加入卫队，人数也不会超过当前已知的最大卫队人数，则该分支不再继续。

终止条件：遍历子集树中所有可能的解，并找到最优解，即卫队人数最多。

算法设计：

1. 从空卫队开始，设置当前最大卫队人数为 0。
2. 逐个考虑每个居民是否加入卫队。
3. 对于每个居民，使用剪枝策略检查是否可以加入。
4. 如果可以加入，则更新当前卫队组合，并递归地考虑下一个居民。
5. 在每个递归步骤后，检查当前卫队人数是否超过已知的最大值，如果超过，则更新最大值。
6. 如果当前居民不能加入，尝试下一个居民。
7. 当所有居民都被考虑过后，得出当前最大卫队人数和组合。
8. 回溯到上一步，尝试不同的组合。

时间复杂度分析：在最坏的情况下，算法将探索所有可能的子集，所以时间复杂度为 $O(2^n)$,其中 n 是居民数量。