算法分析第5次作业

小组编号：23

本次作业负责人:黄勖

1 frog_leap 问题

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。

解空间的定义:

每次跳跃只能与0相距1或者2的青蛙，最多有4只青蛙可能跳跃。因此，解空间是一个四叉树。

剪枝函数:

1. 青蛙不能往回跳，因此将往回跳的子树也剪去。
2. 如果当前结点下的青蛙跳的次数已经大于先前青蛙完成交换的最小跳数 `minJump` ，则进行剪枝。

终止条件:

左右各三只青蛙都已交换位置。

算法实现:

1. 从根结点出发，采用深度优先搜索策略。
2. 选择一个子结点，代表满足跳跃条件的一只青蛙跳跃。
3. 更新状态，将跳数 `Jump` 加1。
4. 根据剪枝函数，如果当前跳数 `Jump` 已经超过先前计算的最小跳数 `minJump` ，则剪去该结点的子树。
5. 递归选择当前子结点的子结点，以此类推，直到状态变成目标状态记录最小跳数 `minJump` 或无法再完成跳跃时终止。
6. 所有回溯搜索完毕后， `minJump` 的值即为完成交换的最小跳数。

时间复杂度分析:

由于问题的解空间是一颗完全4叉树，时间复杂度为$O(4^n)$。

本题分工：小组共同讨论，李嘉琪编写

2　算法分析题5-6 答案：

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）假设旅行售货员的费用为 `cost`，则可以得到前缀为 $x[1:i]$ 的旅行售货员的任一回路可以表示成排列 $(x[1], x[2], \ldots, x[i], \pi(i+1), \pi(i+2), \ldots, \pi(n))$，则当前回路的费用 `cost` 为

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

即

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

证明完毕。

（2）**解空间的定义：**

旅行售货员问题的解空间为 n 个元素组成的排列树，进一步优化解空间结构，每个结点可以走的路径为可以到达未被访问的结点。

剪枝函数：

假设已解出一个答案为 `mincost`，且当前搜索至第 i 层，如果当前费用加上剩下所有出发的最小边之和比 `mincost` 还要大，则说明在这个结点下不存在最优解，可剪去子树。同时，若当前费用已经大于 `mincost`，那也要剪去子树。

终止条件：

如果最终遍历的层数达到 n 说明递归结束，将当前的费用与 `mincost` 比较，小于的话更新 `mincost`，递归直至遍历完整棵排列树。

算法实现：

1. 从第一个元素开始，采用深度优先搜索策略。
2. 选择 n 个子结点中的一个，然后对选中的子树再进行搜索。
3. 在每一层，选择其中一个子结点，将问题规模减小为 n-1 个子结点中的一个，以此递归。

4. 在搜索的过程中，如果当前费用已经大于先前计算的最小费用 `mincost`，或者当前费用加上剩余结点的最小出度和大于 `mincost`，则剪去该结点的子树，提前回溯。
5. 若搜索到叶子结点，比较此时的费用和 `mincost`。如果当前费用更小，则更新 `mincost` 为此时的费用。
6. 当所有回溯遍历均完成后，`mincost` 即为所求的最小费用。

时间复杂度分析:

算法时间复杂度为 $O(n!)$，辅助空间复杂度为 $O(n^2)$。

本题分工：小组共同讨论，黄勖编写

3 算法实现题5-1 答案:

问题描述：子集和问题的一个实例为<S, t>。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!”。

解空间的定义:

解空间由 n 个元素组成的子集树构成，其中每个元素 i 可选择取出来构成子集 S1，形成解空间。

剪枝函数:

当搜索到 i 个元素时，若前 i 个元素中所取的元素之和已经超过了目标值 c，则无法到达目标答案，可以进行剪枝。

终止条件:

递归终止条件为递归到 n 层后，若当前 $nowSum = c$，则找到符合要求的解，结束递归。否则，结束当前层递归，回溯到上一层。

算法实现:

1. 从第一个元素开始，以深度优先搜索方式进行搜索。
2. 对于第一个元素，存在取和不取两种选择。
3. 对其子树进行搜索，选择取或不取，直到将该部分子树搜索完毕。
4. 若没有子树可搜索，则回溯到其父节点，进行与原选择相反的另一种选择，进入其另一个子树搜索。
5. 两边都搜索完后，再回溯到该父节点的父节点，以此类推，直至判断所有情况。

6. 在搜索的过程中，若进行了选择或选择改变，更新当前和 `nowSum`，并修改该元素是否取入的标记。
7. 若 `nowSum` 等于目标值 t，则找到一个满足条件的子集，输出标记取入的元素，终止算法。
8. 若 `nowSum` 大于目标值 t，根据剪枝函数剪去接下来的子树，回溯。
9. 若 `nowSum` 小于目标值 t，继续搜索。

时间复杂度分析:

本题算法的时间复杂度为 $O(2^n)$，辅助空间复杂度为 O(n)。

本题分工：小组共同讨论，李嘉琪编写

4 算法实现题5-3 答案：

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。

解空间的定义:

问题的解空间为 n 个数字的排列树，每个数字取 1 到 m 中的一个数，组成为一颗深度为 n+1 的 m 叉完全树。

剪枝函数:

1. 若当前结点的总价大于 c，则剪去该结点下的子树。
2. 若当前结点的重量大于先前的最小重量 `minWeight`，则剪去该结点下的子树。

终止条件:

当已经递归了 n 层时，说明每个部件都找到了供应商。如果当前的 `nowWeight` 小于 `minWeight`，则更新 `minWeight`。

算法实现:

1. 采用深度优先搜索方式，从第一个结点开始选择，选择方案为 1 到 m 的正整数。
2. 第一个数选择完后，向下遍历，选择第二个数，以此类推搜索到叶子结点。
3. 在搜索过程中，使用 `nowWeight` 和 `nowValue` 分别表示当前结点的重量和价值，利用剪枝函数进行比较，剪去不满足条件的子树。
4. 遍历到叶子结点时，若 `nowWeight` 小于 `minWeight`，用 `nowWeight` 更新 `minWeight`。进行回溯。

时间复杂度分析:

由于解空间是深度为 n+1 的 m 叉完全树，搜索叶子结点个数为 m^n。每个结点的操作时间复杂度为 O(1)。每次更新最优解的时间复杂度为 $O(n)$。因此，总的时间复杂度为 $O(n \cdot m^n)$，辅助空间复杂度为 $O(n)$。

本题分工：小组共同讨论，黄勖编写

5 算法实现题5-6 答案：

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。

解空间的定义:

问题的解空间可表示为原集合分成 n 个子集的所有情况，形成一颗完全 n 叉树。

剪枝函数:

1. 往更深一层搜索结点时，即每加入一个新的数时，判断它与每个子集是否存在冲突。
2. 如果存在冲突，则剪去该结点的子树。

终止条件:

需要当所有情况遍历完毕，也就是搜索完所有 n 子集加入的选择，递归结束。

算法实现:

1. 采用深度优先搜索策略，选择第一个数，即 1，加入 n 个子集中的某个子集。
2. 选择下一个数，即 2，继续加入 n 个子集中的某个子集，以此类推。
3. 若到了一个数 k，它无法加入到当前任何集合中（即剪枝函数生效），比较 `k - 1` 和先前得出的最大 k 值 `maxK`。
4. 若 `k - 1 > maxK`，用 `k - 1` 更新 `maxK` 值。
5. 进行回溯，遍历其他情况。
6. 当回溯完成后，当前的 `maxK` 即为所求。

时间复杂度分析:

由于解空间是一颗完全 n 叉树，时间复杂度为 O(k · n^k)。

本题分工：小组共同讨论，李嘉琪编写

6 算法实现题5-13 答案：

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。

解空间的定义：

问题的解空间包括 1 到 n 共 n 个数字的排列，表示每个工作由第 j 个人承担。可组织为一颗排列树，根结点下的一层有 n 个结点，每个结点的下一层有 n-1 个结点，以此类推，直到叶子结点。

剪枝函数：

1. 用 `minValue` 表示先前各种排列下的最少费用。
2. 若搜索到当前结点时费用 `nowValue` 已经大于了 `minValue`，则剪去当前结点下的子树。

终止条件：

如果已经分配完了 n 个工作，考虑当前的 `nowValue` 值是否优于最优解 `minValue`。如果优于，则更新 `minValue`，然后回溯递归。递归完所有排列树后结束递归。

算法实现：

1. 采用深度优先搜索策略，首先选择根结点下的一个子结点，让它干第一个工作。
2. 在选出的子结点中选择一个结点来干第二个工作，以此类推，直到搜索到叶子结点，即所有工作都被分配完。
3. 从叶子结点进行回溯，形成其他种类的工作分配排列顺序，直到所有顺序都被遍历过。
4. 在每次进入一个结点时，更新当前所需价值 `nowValue`，比较 `nowValue` 和先前各种排列中的最少费用 `minValue`。
5. 若 `nowValue > minValue`，根据剪枝函数，减掉该结点下的子树。
6. 当每次搜索到叶子结点时，比较 `nowValue` 和 `minValue`。若 `nowValue < minValue`，则更新 `minValue` 为 `nowValue`。
7. 完成所有搜索后，算法终止，输出 `minValue` 即为所求。

时间复杂度分析：

该问题的解空间是一颗排列树，共有 n! 种排列情况，时间复杂度为 $O(n!)$。

本题分工：小组共同讨论，黄勖编写

7　算法实现题5-16 答案：

5-16　无优先级运算问题。

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。

解空间的定义：

问题的解空间包括 n 个正整数的排列树和长度为 n-1 的运算符的排列树。其中，正整数排列树每层的子结点数比上一层少 1，而运算符排列树的每个非叶子结点均有 4 个子结点。

剪枝函数：

1. 当搜索正整数排列树时，如果当前结点在正整数排列树中的位次已经超过了最少运算次数 `minNum`，剪去该结点的子树。
2. 在搜索正整数排列树的过程中，若当前结点利用 `calculate(int x)` 函数判断该算式结果等于 m，剪去该结点的子树。

终止条件：

遍历完整个搜索树结束递归，得到最优解。

算法实现：

1. 定义 `calculate(int x)` 函数用于判断，通过按顺序的 x 个正整数和 x-1 个运算符的运算结果是否为 m。
2. 采用深度优先搜索策略，选择一次运算中的第一个正整数，然后在其子结点中选择第二个正整数，同时选择一个运算符，以此类推，直到选择到叶子结点，进行回溯，依次遍历所有情况。
3. 当搜索到一个正整数结点时，根据剪枝函数，如果当前结点的深度已经超过了 `minNum`，剪去子树，直接回溯。如果当前结点利用 `calculate(int x)` 函数判断该算式结果等于 m，用该式子的正整数个数更新 `minNum`，剪去子树，直接回溯。
4. 完成所有搜索后，`minNum` 即为实现算式等于 m 的最少运算次数。

时间复杂度分析：

计算正整数解空间的排列树的时间复杂度为 $O(n!)$，计算运算符解空间的排列树的时间复杂度为 $O(4^{n-1})$。因此，算法总的时间复杂度为 $O(4^{n-1} \cdot n!)$。

本题分工：小组共同讨论，李嘉琪编写

8 算法实现题5-20 答案：

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。

解空间的定义：

问题的解空间是由 n 个 0 或 1 字符组成的排列树，其中 1 表示加入部落卫队，0 表示不加入。该解空间可以组织成一颗子集树。

剪枝函数：

1. 每次选择加入一个居民到部落卫队时，判断他和当前卫队中的成员有无矛盾，若有则不符合条件，剪去当前结点下的子树。
2. 如果后面所有队员都能加入卫队也无法取得更大的人数，则剪去当前结点的子树。

终止条件：

遍历完整个搜索树结束递归，得到最优解。

算法实现：

1. 采用深度优先策略，选择第一个居民是否入护卫队，然后再选择第二个居民是否入护卫队，以此类推，直到叶子结点。搜索到叶子结点再进行回溯，遍历所有情况。
2. 在进入到一个结点时，根据剪枝函数，若有 `已选的人数 + 剩余的人数 < maxNum`，剪去该结点的子树。同时，当选择该结点表示的居民加入卫队时，要判断他能否与当前卫队中所有人都不结仇，若有结仇的，则不满足题意，剪去该结点的子树。
3. 每次搜索到叶子结点时，比较该选取集合中的人数 `nowNum` 与 `maxNum` 的大小，若 `nowNum > maxNum`，用 `nowNum` 更新 `maxNum`。搜索完所有情况时，`maxNum` 即为所求。

时间复杂度分析：

有 2^n 种集合的选择，不剪枝下，共会遍历 2^n 个结点，遍历每个时要进行 O(n) 的时间复杂度判断是否能加入卫队结合，因此该问题的时间复杂度为 $O(n \cdot 2^n)$。

本题分工：小组共同讨论，黄勖编写