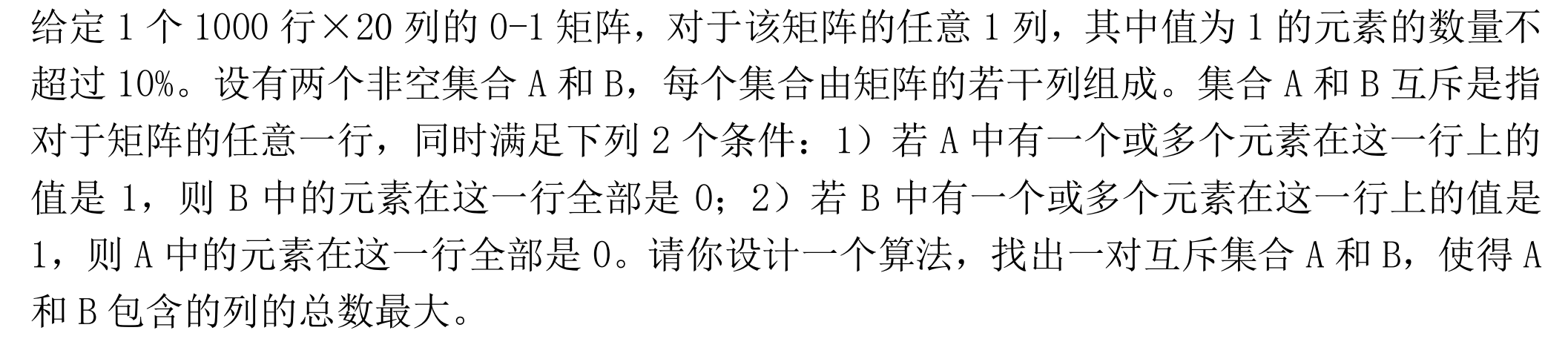
西安交通大学实验报告

课程名称：算法设计与分析 实验名称： 回溯算法

学 院：\_\_\_\_\_\_\_电信学部\_\_\_\_\_\_\_\_\_\_\_\_ 实 验 日 期 2022年4 月 24 日

|  |  |  |
| --- | --- | --- |
| **刘沁宇** | **计算机002** | **2203613019** |

#### 问题描述



#### 问题分析

此问题的输入的数据规模较大，采用暴力算法势必会导致算法的执行效率较低，且求解一对互斥集合，不具备最优子结构性质，因此可考虑用分支限界法或者回溯法加剪枝太降低时间复杂度。

#### 算法设计

在利用回溯法对每列考虑三种情况

（1）.该列加入数组A

（2）.该列加入数组B

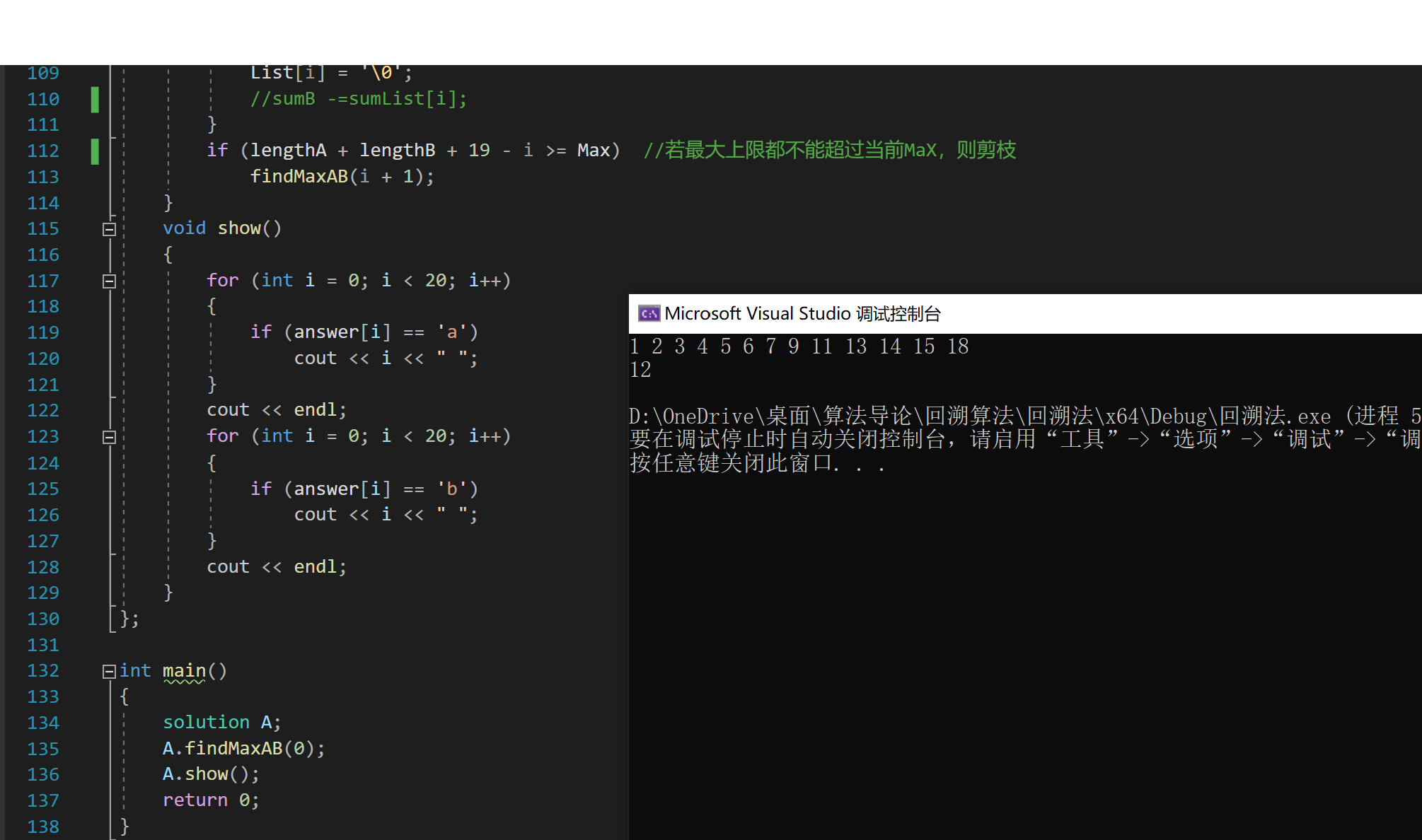
（3）.该列既不加入数组A也不加入数组B

利用二维数组judge来判断任意两列中是否有同一行都是1（即是否冲突），利用inA和inB函数判断当前列是否能加入到A或B中，再套用回溯的框架即可得到答案。

#### 算法实现

|  |
| --- |
| 1. #include <iostream> 2. #include <vector> 3. #include <string> 4. #include <fstream> 5. **using** **namespace** std; 6. **class** solution 7. { 8. **int** matrix[1000][20]; 9. **int** Max; 10. **bool** judge[20][20];      //判断i,j列是否冲突 11. **int** lengthA, lengthB;    // sumA, sumB; 12. **int** dif; 13. **char** List[20]; 14. **char** answer[20]; 15. **int** sumList[20]; 16. **public**: 17. solution( )   //构造列表 18. { 19. fstream infile; 20. infile.open("exp\_in.txt"); 21. **for** (**int** i = 0; i < 20; i++) 22. { 23. sumList[i] = 0; 24. } 25. **for** (**int** i = 0; i < 1000; i++) 26. { 27. **for** (**int** j = 0; j < 20; j++) 28. { 29. infile >> matrix[i][j]; 30. **if** (matrix[i][j]==1) 31. sumList[j] += 1; 32. } 33. } 34. **for** (**int** i = 0; i < 20; i++)   //初始化judge矩阵 35. { 36. List[i] = '\0'; 37. answer[i] = '\0'; 38. **for** (**int** j = 0; j < 20 ; j++) 39. { 40. judge[i][j] = judge[j][i] = **true**;           //true表示不冲突 41. **for** (**int** k = 0; k <1000;k++) 42. { 43. **if** (matrix[k][i] == 1 && matrix[k][j] == 1) 44. { 45. judge[i][j] = judge[j][i] = **false**;   //false表示冲突 46. **break**; 47. } 48. } 49. } 50. } 51. lengthA = lengthB = Max = 0; 52. dif = 0; 53. } 54. **bool** inA(**int** n)   //判断第n行是否能加入集合A中 55. { 56. **for** (**int** i = 0; i < 20; i++) 57. { 58. **if** (List[i] == 'b' && !judge[i][n]) 59. { 60. **return** **false**; 61. } 62. } 63. **return** **true**; 64. } 65. **bool** inB(**int** n)  //判断第n行是否能加入集合B中 66. { 67. **for** (**int** i = 0; i < 20; i++) 68. { 69. **if** (List[i] == 'a' && !judge[i][n]) 70. { 71. **return** **false**; 72. } 73. } 74. **return** **true**; 75. } 76. **void** findMaxAB(**int** i) 77. { 78. **if** (i > 19) 79. { 80. **if** (lengthA + lengthB >= Max && lengthA != 0 && lengthB != 0 && lengthA-lengthB > 0 && lengthA-lengthB > dif ) 81. { 82. Max = lengthA + lengthB; 83. dif = lengthA - lengthB; 84. **for** (**int** i = 0; i < 20; i++) 85. { 86. answer[i] = List[i]; 87. } 89. } 90. **return**; 91. } 92. **if** (inA(i)) 93. { 94. lengthA++; 95. List[i] = 'a'; 96. findMaxAB(i + 1);  //判断下一列 97. lengthA--;            //撤销选择 98. List[i] = '\0'; 99. } 100. **if** (inB(i)) 101. { 102. lengthB++; 103. List[i] = 'b'; 104. findMaxAB(i + 1);  //判断下一列 105. lengthB--;            //撤销选择 106. List[i] = '\0'; 107. } 108. **if** (lengthA + lengthB + 19 - i >= Max)  //若最大上限都不能超过当前MaX，则剪枝 109. findMaxAB(i + 1); 110. } 111. **void** show() 112. { 113. **for** (**int** i = 0; i < 20; i++) 114. { 115. **if** (answer[i] == 'a') 116. cout << i << " "; 117. } 118. cout << endl; 119. **for** (**int** i = 0; i < 20; i++) 120. { 121. **if** (answer[i] == 'b') 122. cout << i << " "; 123. } 124. cout << endl; 125. } 126. }; 128. **int** main() 129. { 130. solution A; 131. A.findMaxAB(0); 132. A.show(); 133. **return** 0; 134. } |

#### 实验结果



图表 1 所给用例输出与期望输出相同