**实验7-装载问题**

**问题分析**

* 装载问题类似于背包问题。假设一个装载问题有解，则可采用的策略是：
  + 先将第一艘船尽可能装满
  + 将剩余的集装箱装上第二艘船（如果不能把剩余所有货物装上第二艘船，那么该问题无解）

其中将第一艘船尽可能装满等价于面向全体集装箱选法的一个子集，且该子集是所有子集中重量之和最接近于C1的。

* 定义string assign为当前分配方案，string best为最佳分配方案。Int cargo1max为第一艘船最大装载量
* 算法：采用深度优先搜索算法+回溯法。每次递归DFS，有两种状态选择：
  + 如果第一艘船能够装下当前集装箱，则放在了第一艘船上。
  + 当前集装箱选择放在第二艘船上。

其中第一艘船选上该集装箱时，对应的assign字符串相应的位置赋值为1，回溯的时候应恢复现场，即退出递归时再赋值为0 。

1. **if** (w[count] + cargos1 <= c1)
2. {
3. assign[count] = '1';
4. dfs(count + 1, w[count] + cargos1);
5. assign[count] = '0';//回溯
6. }
7. dfs(count + 1, cargos1);//不选第count件货物

* 终止判断

当 count == n+1时代表到达搜索终点（count对应集装箱数量）

* + 若第二艘船无法装载剩余货物，代表当前方案不可行
  + 若当前第一艘船能装载的重量大于第一艘船能装的最大装载量cargo1max，则更新最大装载量。
  + 如果存在多种最大的方案，直接通过比较字符串大小选择字典序更大的。

1. **if** (count == n + 1)
2. {
3. **if** (cargos1 > cargo1max)
4. {
5. cargo1max = cargos1;
6. //cout << cargo1max << endl;
7. best = assign;
8. }
9. **if** (cargos1 == cargo1max && assign > best)//选择字典序大的
10. {
11. best = assign;
12. }
13. **return** ;
14. }

* 当两艘船的承载量之和都小于集装箱总量时，则不能装载

1. **if** (cargo1max + c2 < sum)
2. {
3. cout << cargo1max << " " << c2<<endl;
4. cout << "No" << endl;
5. }

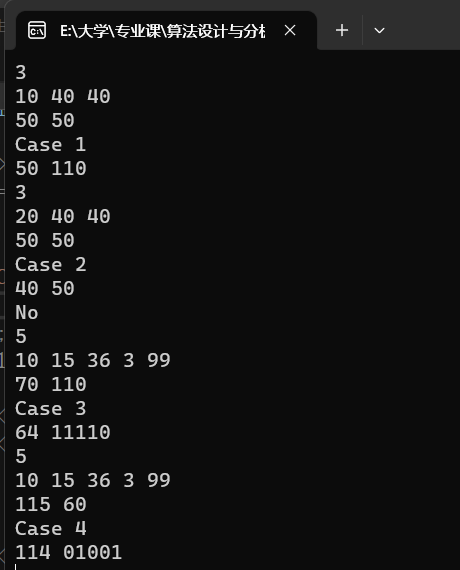
**进阶任务**

剪枝处理

1. **int** r;//剩余重量
2. **void** dfs(**int** count,**int** cargos1)
3. {
4. ... ...
5. r -= w[count];
6. ... ...
7. **if** (cargos1 + r > cargo1max)//剪枝
8. {
9. dfs(count + 1, cargos1);
10. r += w[count];
11. }
12. }

该算法的解空间是一颗子集树，其中最多含有个节点，对应的算法时间复杂度为

**运行结果截图**



**设计调试中的问题**

* 初始化问题

由于有多组测试数据，除了每一次输入要将基本的sum和cargo1max清零之外，还需要将string类型字符串重新设置大小以防止数组越界。

1. assign.resize(n + 1, '0');
2. best.resize(n + 1, '0');
3. **int** sum = 0;
4. cargo1max = 0;

**实验体会**

此次实验相比一般的背包问题较为简单，虽然没运用动态规划的理念，但是却加深了我对于深度优先搜索的理解，也让我体会到了回溯法的巧妙。在每一层递归结束后都恢复到原始状态，这保证了情况分析的不重不漏，在小规模数据中配合深度优先搜索算法，能够快速地得出最优解。

这是本课程的最后一次实验，回顾这这七次实验涉及到了递归分治，动态规划，广度优先搜索，深度优先搜索等等这其中大部分算法都是初次接触，让我体会到了算法的奥妙所在，也希望日后继续研究算法，并能在恰当的地方将其运用。

**程序源码**

1. #include<iostream>
2. #include<bits/stdc++.h>
3. **using** **namespace** std;
4. **int** n;
5. **int** w[21];
6. **int** c1, c2;
7. string assign, best;//当前分配方案 和 最佳分配方案
8. **int** cargo1max;//第一艘船最大装载量
9. **int** r;//剩余重量
10. **void** dfs(**int** count,**int** cargos1)//cargos1第一艘船装在重量
11. {
12. **if** (count == n + 1)
13. {
14. **if** (cargos1 > cargo1max)
15. {
16. cargo1max = cargos1;
17. //cout << cargo1max << endl;
18. best = assign;
19. }
20. **if** (cargos1 == cargo1max && assign > best)//选择字典序大的
21. {
22. best = assign;
23. }
24. **return** ;
25. }
26. r -= w[count];
27. **if** (w[count] + cargos1 <= c1)
28. {
29. assign[count] = '1';
30. dfs(count + 1, w[count] + cargos1);
31. assign[count] = '0';//回溯
32. }
33. **if** (cargos1 + r > cargo1max)
34. {
35. dfs(count + 1, cargos1);//不选第count件货物
36. r += w[count];
37. }
38. }
40. **int** main()
41. {
42. **int** cas = 1;
43. **while** (cin >> n)
44. {
45. assign.resize(n + 1, '0');
46. best.resize(n + 1, '0');
47. **int** sum = 0;
48. cargo1max = 0;
49. **for** (**int** i = 1; i <= n; i++)
50. {
51. cin >> w[i];
52. sum += w[i];
53. }
54. r = sum;
55. cin >> c1 >> c2;
56. cout << "Case " << cas << endl;
57. cas++;
58. dfs(1, 0);
59. **if** (cargo1max + c2 < sum)
60. {
61. cout << cargo1max << " " << c2<<endl;
62. cout << "No" << endl;
63. }
64. **else**
65. {
66. cout << cargo1max << " " << best.substr(1,n) << endl;
67. }
68. }
69. }