[【编程之美】数组分割问题](http://blog.csdn.net/tianshuai11/article/details/7828907)

一，问题：

      1. 有一个无序、元素个数为2n的正整数数组，要求：如何能把这个数组分割为两个子数组，子数组的元素个数不限，并使两个子数组之和最接近。

      2. 有一个无序、元素个数为2n的正整数数组，要求：如何能把这个数组分割为元素个数为n的两个数组，并使两个子数组之和最接近。

二，分析：

         假设数组A[1..2N]所有元素的和是SUM。模仿动态规划解0-1背包问题的策略，令S(k, i)表示前k个元素中任意i个元素的和的集合。显然：  
                S(k, 1) = {A[i] | 1<= i <= k}  
                S(k, k) = {A[1]+A[2]+…+A[k]}  
                 S(k, i) = S(k-1, i) U {A[k] + x | x属于S(k-1, i-1) }  
          按照这个递推公式来计算，最后找出集合S(2N, N)中与SUM最接近的那个和，这便是答案。这个算法的时间复杂度是O(2^N).  
          因为这个过程中只关注和不大于SUM/2的那个子数组的和。所以集合中重复的和以及大于SUM/2的和都是没有意义的。把这些没有意义的和剔除掉，剩下的有意义的和的个数最多就是SUM/2个。所以，我们不需要记录S(2N,N)中都有哪些和，只需要从SUM/2到1遍历一次，逐个询问这个值是不是在S(2N,N)中出现，第一个出现的值就是答案。我们的程序不需要按照上述递推公式计算每个集合，只需要为每个集合设一个标志数组，标记SUM/2到1这个区间中的哪些值可以被计算出来。

二， 解法：

       由于对两个子数组和最接近的判断不太直观，我们需要对题目进行适当转化。我们知道当一个子数组之和最接近原数组之和sum的一半时，两个子数组之和是最接近的。所以转化后的题目是：**从2n个数中选出任意个数，其和尽量接近于给定值sum/2**。

       这个问题存储的是从前k个数中选取任意个数，且其和为s的取法是否存在dp[k][s]。之所以将选出的数之和放在下标中，而不是作为dp[k]的值，是因为那种做法不满足**动态规划**的前提——**最优化原理**，假设我们找到最优解有k个数p1p2...pk（选出的这k个数之和是最接近sum/2的），但最优解的前k-1个数p1p2...pk-1之和可能并不是最接近sum/2的，也就是说可能在访问到pk之前有另一组数q1q2....qk-1其和相比p1p2...pk-1之和会更接近sum/2，即最优解的子问题并不是最优的，所以不满足最优化原理。因此我们需要将dp[k]的值作为下标存储起来，将这个**最优问题转化为判定问题**，用带动态规划的思想的递推法来解。

       外阶段：在前k1个数中进行选择，k1=1,2...2\*n。  
       内阶段：从这k1个数中任意选出k2个数，k2=1,2...k1。

          状态：这k2个数的和为s，s=1,2...sum/2。

          决策：决定这k2个数的和有两种决策，一个是这k2个数中包含第k1个数，另一个是不包含第k1个数。

dp[k][s]表示取k个数，且这些数之和为s的取法是否存在，且这些数之和为s的取法是否存在。

**[html]** [view plaincopy](http://blog.csdn.net/tianshuai11/article/details/7828907)

1. #include **<iostream>**
2. #include **<algorithm>**
4. using namespace std;
6. #define MAXN 101
7. #define MAXSUM 100000
8. int A[MAXN];
9. bool dp[MAXN][MAXSUM];
11. // dp[k][s]表示从前k个数中去任意个数，且这些数之和为s的取法是否存在
12. int main()
13. {
14. int n, i, k1, k2, s, u;
15. cin **>>** n;
16. for (i=1; i**<**=2\*n; i++)
17. cin **>>** A[i];
18. int sum = 0;
19. for (i=1; i**<**=2\*n; i++)
20. sum += A[i];
21. memset(dp,0,sizeof(dp));
22. dp[0][0]=true;
23. // 外阶段k1表示第k1个数，内阶段k2表示选取数的个数
24. for (k1=1; k1**<**=2\*n; k1++)            // 外阶段k1
25. {
26. for (k2=k1; k2**>**=1; k2--)     // 内阶段k2
27. for (s=1; s**<**=sum/2; s++) // 状态s
28. {
29. //dp[k1][s] = dp[k1-1][s];
30. // 有两个决策包含或不包含元素k1
31. if (s**>**=A[k1] && dp[k2-1][s-A[k1]])
32. dp[k2][s] = true;
33. }
34. }
35. // 之前的dp[k][s]表示从前k个数中取任意k个数，经过下面的步骤后
36. // 即表示从前k个数中取任意个数
37. for (k1=2; k1**<**=2\*n; k1++)
38. for (s=1; s**<**=sum/2; s++)
39. if (dp[k1-1][s])
40. dp[k1][s]=true;
41. // 确定最接近的给定值sum/2的和
42. for (s=sum/2; s**>**=1 && !dp[2\*n][s]; s--)
43. ;
45. printf("the differece between two sub array is %d\n", sum-2\*s);
46. }

2. 解法：

     但本题还增加了一个限制条件，即选出的物体数必须为n，这个条件限制了内阶段k2的取值范围，并且dp[k][s]的含义也发生变化。这里的dp[k][s]表示从前k个数中取任意不超过n的k个数，且这些数之和为s的取法是否存在

**[sql]** [view plaincopy](http://blog.csdn.net/tianshuai11/article/details/7828907)

1. #include <iostream>
2. #include <algorithm>
4. using namespace std;
6. #define MAXN 101
7. #define MAXSUM 100000
8. **int** A[MAXN];
9. bool dp[MAXN][MAXSUM];
11. // 题目可转换为从2n个数中选出n个数，其和尽量接近于给定值sum/2
12. **int** main()
13. {
14. **int** n, i, k1, k2, s, u;
15. cin >> n;
16. **for** (i=1; i<=2\*n; i++)
17. cin >> A[i];
18. **int** sum = 0;
19. **for** (i=1; i<=2\*n; i++)
20. sum += A[i];
21. memset(dp,0,sizeof(dp));
22. dp[0][0]=**true**;
23. // 对于dp[k][s]要进行u次决策，由于阶段k的选择受到决策的限制，
24. // 这里决策选择不允许重复，但阶段可以重复，比较特别
25. **for** (k1=1; k1<=2\*n; k1++)                // 外阶段k1
26. **for** (k2=**min**(k1,n); k2>=1; k2--)      // 内阶段k2
27. **for** (s=1; s<=sum/2; s++) // 状态s
28. // 有两个决策包含或不包含元素k1
29. if (s>=A[k1] && dp[k2-1][s-A[k1]])
30. dp[k2][s] = **true**;
31. // 确定最接近的给定值sum/2的和
32. **for** (s=sum/2; s>=1 && !dp[n][s]; s--);
33. printf("the differece between two sub array is %d\n", sum-2\*s);
34. }

注意：如果数组中有负数的话，上面的背包策略就不能使用了（因为第三重循环中的s是作为数组的下标的，不能出现负数的），需要将数组中的所有数组都加上最小的那个负数的绝对值，将数组中的元素全部都增加一定的范围，全部转化为正数，然后再使用上面的背包策略就可以解决了。