这个问题又称为“子集和问题”（也就是给定一个整数集合和一个定值，从一个集合中选取一个子集，使得子集中所有数的和等于给定的值，具体的可以百度，google 子集和问题），这是一个NP完全问题，不存在多项式时间的解，所以没有好的算法。

回朔法（本质还是穷举法）

动态规划（两种）  
回朔法（第一种）  
#include <iostream>

#include <vector>

#include <bitset>

**using** **namespace** std;

void print(bitset<5> & X)

{

**for** (int i = 0; i < X.size(); ++i)

{

cout << X[i] << " ";

}

cout << endl;

}

void subsetSum(vector<int>& W, bitset<5>& X, int sum, int targetSum, int k)

{

X[k] = 1; // try one branch of tree

**if** (sum + W[k] == targetSum)

print(X); // we have a solution

**else** **if** (k + 1 < W.size() && sum + W[k] <= targetSum) // include the W[k]

subsetSum(W, X, sum + W[k], targetSum, k + 1); // the maxium k is W.size-2

**if** (k + 1 < W.size() && sum + W[k + 1] <= targetSum) {

X[k] = 0; // uniclude the W[k]

subsetSum(W, X, sum, targetSum, k + 1); // the maxium k is W.size-2

}

}

int main() {

vector<int> W = { 2, 4, 6, 8, 10 };

bitset<5> X;

subsetSum(W, X, 0, 20, 0);

system("pause");

**return** 0;

}

算法思想：考虑是否取第n个数，问题可以转化为前n-1个数和为sum-a[n-1]的问题，也可以转化为后n-1个数的求和问题。使用递归思想解决。

如果取第n个数，即求得前n-1个数满足和为sum-a[n-1]的问题

如果不取第n个数，即求得前n-1个数满足和为sum的问题

[**http://www.jianshu.com/p/3d1791cfba53(Ksum)**](http://www.jianshu.com/p/3d1791cfba53(Ksum))

**http://blog.csdn.net/a130737/article/details/44242047**

**三、k个和为定值的个数**

题目：给出[1,2,3,4]，k=2， target=5，[1,4] and [2,3]是2个符合要求的方案  
地址：<http://www.lintcode.com/zh-cn/problem/k-sum/>  
解析：dp[i][j][p] 表示从i个数中挑j个数和为p时的次数。  
dp[0][0][0]表示在一个空集中找出0个数，target为0，则有1个解，就是什么也不挑嘛！  
其实应该这样写，也就是说，找0个数，目标为0，则一定是有1个解：  
if (j == 0 && p == 0) {  
　　// select 0 number from i to the target: 0  
　　D[i][j][p] = 1;  
}

1. 状态表达式：

D[i][j][p] = D[i - 1][j][p]; //不包含第i个元素

if (p - A[i - 1] >= 0) { //可以包含第i个元素

D[i][j][p] += D[i - 1][j - 1][t - A[i - 1]];

}

意思就是：

（1）我们可以把当前A[i - 1]这个值包括进来，所以需要加上D[i - 1][j - 1][t - A[i - 1]]（前提是t - A[i - 1]要大于0）

（2）我们可以不选择A[i - 1]这个值，这种情况就是D[i - 1][j][t]，也就是说直接在前i-1个值里选择一些值加到target.

public class Solution {

/\*\*

\* @param A: an integer array.

\* @param k: a positive integer (k <= length(A))

\* @param target: a integer

\* @return an integer

\*/

public int kSum(int A[], int k, int target) {

// write your code here

int len = A.length;

int [][][] dp = new int[len+1][k+1][target+1];

if(target < 0)

return 0;

for(int i = 0; i <= len; ++i)

for(int j = 0; j <= k; ++j)

for(int p = 0; p <= target; ++p)

{

if(j == 0 && p == 0)

dp[i][j][p] = 1;

else if(i != 0 && j != 0 && p!= 0)

{

dp[i][j][p] = dp[i-1][j][p];

if(p >= A[i-1])

dp[i][j][p] += dp[i-1][j-1][p-A[i-1]];

}

}

return dp[len][k][target];

}

}

3sum

<http://blog.csdn.net/nomasp/article/details/49209523>

https://www.jiuzhang.com/solutions/3sum/

class Solution {

public:

vector<vector<int>> threeSum(vector<int>& nums) {

sort(nums.begin(), nums.end());

vector<vector<int>> result;

int len = nums.size();

for (int current = 0; current < len - 2&&nums[current]<=0;current++)

{

int front = current + 1, back = len - 1;

while (front < back)

{

if (nums[current] + nums[front] + nums[back] < 0)

front++;

else if (nums[current] + nums[front] + nums[back] > 0)

back--;

else

{

vector<int> v(3);

v.push\_back(nums[current]);

v.push\_back(nums[front]);

v.push\_back(nums[back]);

result.push\_back(v);

v.clear();

do {

front++;

} while (front < back&&nums[front - 1] == nums[front]);

do {

back--;

} while (front < back&&nums[back + 1] == nums[back]);

}

}

while (current < len - 2 && nums[current + 1] == nums[current])

current++;

}

return result;

}

};

4sum

<http://blog.csdn.net/NoMasp/article/details/49226833>

<https://www.jiuzhang.com/solutions/4sum/>

**class** Solution {

**public**:

/\*

题意：找到数列中所有和等于目标数的四元组，需去重

多枚举一个数后，参照3Sum的做法，O(N^3)

\*/

vector<vector<int>> fourSum(vector<int>& nums, **int** target) {

**int** len = nums.size();

**int** left, right, sum;

sort(nums.**begin**(), nums.**end**());

vector<vector<int>> res;

vector<int> tmp;

**for** (**int** i = 0; i < len - 3; i++) {

**if** (i && nums[i] == nums[i - 1]) **continue**;

**for** (**int** j = i + 1; j < len - 2; j++) {

**if** (j != i + 1 && nums[j] == nums[j - 1]) **continue**;

sum = target - nums[i] - nums[j];

left = j + 1;

right = len - 1;

**while** (left < right) {

**if** (nums[left] + nums[right] == sum) {

tmp.clear();

tmp.push\_back(nums[i]);

tmp.push\_back(nums[j]);

tmp.push\_back(nums[left]);

tmp.push\_back(nums[right]);

res.push\_back(tmp);

left++;

right--;

**while** (left < right && nums[left] == nums[left - 1]) left++;

**while** (left < right && nums[right] == nums[right + 1]) right--;

} **else**

**if** (nums[left] + nums[right] > sum) right--;

**else** left++;

}

}

}

**return** res;

}

};