1.

class Solution {

public:

int numJewelsInStones(string J, string S) {

int res = 0;

for (char s : S) {

for (char j : J) {

if (s == j) {

++res; break;

}

}

}

return res;

}

};

class Solution {

public:

int numJewelsInStones(string J, string S) {

int res = 0;

unordered\_set<char> s;

for (char c : J) s.insert(c);

for (char c : S) {

if (s.count(c)) ++res;

}

return res;

}

};

简洁的思路：

此问题为字符串的问题，我们依次判断J中的每个字符在S中出现几次。

故用两个for循环嵌套就可以完成，依次比较两个字符串中的每个字符。

注：区分大小写，即“z”与“Z”不同。

代码如下：

int numJewelsInStones(char\* J,char\* S) {

int i,j,count=0;

for(i=0;i<strlen(J);i++){

for(j=0;j<strlen(S);j++){

if(J[i]==S[j]){

count++;

}

}

}

return count;

}

2.

class Solution {

public:

vector<vector<int> > res;

void generateCombined(int n,int k,int start,vector<int> c){

if(c.size()==k){

res.push\_back(c);

return;

}

for(int i=start;i<n;i++){

c.push\_back(i);

generateCombined(n,k,i+1,c);

c.pop\_back();

}

}

vector<vector<int> > combine(int n, int k) {

if(n<=0||k<=0||k>n)

return res;

vector<int> c;

generateCombined(n,k,1,c);

}

};

题目解答：

方法1：回溯算法

循环加递归，记录前边存储的数字，当存储k个即可存储在结果中。

也可以先计算出总共用多少个结果，这样就可以直接给result申请对应大的空间。

运行时间24ms，代码如下。

/\*\*

\* Return an array of arrays of size \*returnSize.

\* The sizes of the arrays are returned as \*columnSizes array.

\* Note: Both returned array and \*columnSizes array must be malloced, assume caller calls free().

\*/

void dfs(int\*\*\* result, int\* size, int n, int k, int\* use, int used, int start) {

if(used == k) {

(\*size)++;

result[0] = (int\*\*)realloc(result[0], \*size \* sizeof(int\*));

result[0][\*size - 1] = (int\*)malloc(k \* sizeof(int));

memcpy(result[0][\*size - 1], use, k \* sizeof(int));

return;

}

int i = 0;

for(i = start; i < n; i++) {

use[used] = i + 1;

dfs(result, size, n, k, use, used + 1, i + 1);

}

}

int\*\* combine(int n, int k, int\*\* columnSizes, int\* returnSize) {

int i = 0;

int\*\* result = NULL;

int\* use = (int\*)malloc(k \* sizeof(int));

dfs(&result, returnSize, n, k, use, 0, 0);

columnSizes[0] = (int\*)malloc(\*returnSize \* sizeof(int));

for(i = 0; i < \*returnSize; i++)

columnSizes[0][i] = k;

free(use);

return result;

}

方法2：迭代法

也有点回溯的感觉，当后一个数字超过最大值，就对前一个数加一，然后继续重新递增下一个数字。

运行时间24ms，代码如下。

/\*\*

\* Return an array of arrays of size \*returnSize.

\* The sizes of the arrays are returned as \*columnSizes array.

\* Note: Both returned array and \*columnSizes array must be malloced, assume caller calls free().

\*/

int\*\* combine(int n, int k, int\*\* columnSizes, int\* returnSize) {

int i = 0;

int\* use = (int\*)calloc(k, sizeof(int));

int\*\* result = NULL;

while(i >= 0) {

use[i]++;

if(use[i] > n)

i--;

else if(i == k - 1) {

(\*returnSize)++;

result = (int\*\*)realloc(result, \*returnSize \* sizeof(int\*));

result[\*returnSize - 1] = (int\*)malloc(k \* sizeof(int));

memcpy(result[\*returnSize - 1], use, k \* sizeof(int));

}

else {

i++;

use[i] = use[i - 1];

}

}

columnSizes[0] = (int\*)malloc(\*returnSize \* sizeof(int));

for(i = 0; i < \*returnSize; i++)

columnSizes[0][i] = k;

free(use);

return result;

}