1. malloc前要加相应的指针类型

```
   int* res=(int *)malloc((m+n)>>2);
```

2. majority element

每找出两个不同的element，则成对删除。最终剩下的一定就是所求的。

可扩展到[n/k]的情况，每k个不同的element进行成对删除。

3. single number II 出现三次

利用三个变量分别保存各个二进制位上 1 出现一次、两次、三次的分布情况，最后只需返回变量一就行了

```
      int singleNumber(int A[], int n) {
      int one,two,three;
      one=two=three=0;
      for(int i=0;i<n;i++)
      {//一定是出现3次，2次，1次这样的顺序，如果反过来的话，先更新了one的话，会影响到two和three的
         three =  two & A[i];//已经出现了两次，还出现了一次
         two = two | one & A[i];//出现了1次又出现了1次，在加上以前已经出现了2次的，为新的出现了2次的
         one = one | A[i];//出现了1次
         //将出现3次的其出现1次2次全部抹去
         one = one & ~three;
         two = two & ~three;
      }
       return one;
   }
```

4. unique path II f定义成全局变量的问题

5. clone graph 注意遍历时利用map有效去除已遍历过的

再次遍历时通过节点的邻接表是否为空判断

6. Contains Duplicate III

```
       for (i = 0; i < nums.size(); i++) {
         if (i > k) window.erase(nums[i-k-1]); // keep the set contains nums i j at most k
         // -t <= x - nums[i] <= t;
         auto pos = window.(nums[i] - t); // lower_boundx >= nums[i] - t
         if (pos != window.end() && *pos - nums[i] <= t) // x <= nums[i] + t
              return true;
         window.insert(nums[i]);
      }
      return false;
```

7. Unique Binary Search Trees

```
     int numTrees(int n) {
         int* f=new int[n+1];
         memset(f,0,(n+1)*sizeof(int));
         f[0]=1;f[1]=1;f[2]=2;
         for(int i=3;i<=n;i++){
             for(int j=1;j<=i;j++)
                 f[i]+=f[j-1]*f[i-j];
       }
       return f[n];
     }
```

8. Gas Station

```
解题思路：

1：假设出发车站为0，初始化车内油量0

2：车内油量＝车站油量－消耗

3：如果车内油量大于0，车开到下一车站，否则出发车站前移一个车站

重复2，3步，直到所有车站遍历完。如果车内油量剩余大于等于0，返回出发车站，否则返回－1.
```

9. Add Digits Total Accepted: 3839Total Submissions: 8437

```
Given a non-negative integer num, repeatedly add all its digits until the result has only one digit.

For example:

Given num = 38, the process is like: 3 + 8 = 11, 1 + 1 = 2. Since 2 has only one digit, return it.

int addDigits(int num) {

  return -num % 9 + 1;

}
```

10. Write a C function to remove spaces from a string. The function header should be void removeSpaces(char *str)

```
void removeSpace(char *str) {

  char *p1 = str, *p2 = str;

  do

    while (*p2 == ' ')

      p2++;

  while (*p1++ = *p2++);

}

其中str必须为  char[] str;
```

11. Validate if a given string is numeric.

```
  Some examples:

  "0" => true

  "   0.1  " => true

  "abc" => false

  "1 a" => false

  "2e10" => true

  Note: It is intended for the problem statement to be ambiguous. You should gather all requirements up front before implementing one. 来源： <https://leetcode.com/problems/valid-number/>
```

用自动机DFA做

```
 //   +/- d . e/E space can_accept

int trans[][6] = {

    { 1, 2, 8, -1, 0, 0 },

    { -1, 2, 8, -1, -1, 0 },

    { -1, 2, 3, 5, 10, 1 },

    { -1, 4, -1, 5, 10, 1 },

    { -1, 4, -1, 5, 10, 1 },

    { 6, 7, -1, -1, -1, 0 },

    { -1, 7, -1, -1, 10, 0 },

    { -1, 7, -1, -1, 10, 1 },

    { -1, 9, -1, -1, -1, 0 },

    { -1, 9, -1, 5, 10, 1 },

    { -1, -1, -1, -1, 10 ,1}

};
```

```
bool isNumber(char* s) {

    char pos[128], c;

    int state = 0;

    memset(pos, -1, 128);

    pos['e'] = pos['E'] = 3;

    pos['+'] = pos['-'] = 0;

    pos['.'] = 2;
    pos[' '] = 4;
    memset(pos + 48, 1, 10);
    while (c=*s++)
        if (pos[c] >= 0){
          state = trans[state][pos[c]];
          if(state < 0)  return false;
        }
        else return false;
    if(trans[state][5]) return true;
    return false;
}
```

12. Dungeon Game

```
  int m, n, i, j;
  int calculateMinimumHP(vector<vector<int>>& dungeon) {
    m = dungeon.size();
    n = dungeon[0].size();
    vector<int> dp(m + 1, 0xffff);
    dp[m - 1] = 1;
    for (i = n - 1; i >= 0; i--)
      for (j = m - 1; j >= 0; j--)
        dp[j] = max(1, min(dp[j + 1], dp[j]) - dungeon[j][i]);
    return dp[0];
  }
```

13. 排列对应序号

```
int  fac[] = { 1, 1, 2, 6, 24, 120, 720, 5040, 40320, 362880, 3628800, 39916800 }; //i的阶乘为fac[i]
/*  康托展开.
{1...n}的全排列由小到大有序，s[]为第几个数  */
int KT(int n, char s[]) {
  int i, j, t, sum;
  sum = 0;
  for (i = 0; i < n; i++) {
    t = 0;
    for (j = i + 1; j < n; j++)
    if (s[j] < s[i])
      t++;
    sum += t*fac[n - i - 1];
  }
  return sum + 1;
}
```

14. Find the Duplicate Number

```
int findDuplicate(int* nums, int numsSize) {
  int slow, fast;
  for (slow = nums[0], fast = nums[slow]; slow != fast; slow = nums[slow])
    fast = nums[nums[fast]];
  for (fast = 0; slow != fast; fast = nums[fast])
    slow = nums[slow];
  return slow;
}
```

15. Find Median from Data Stream

```
class MedianFinder {
public:
    priority_queue<int> large;
    priority_queue<int, vector<int>, greater<int>> small;
    // Adds a number into the data structure.
    void addNum(int num) {
        if (!large.empty() && num < large.top()) {
            large.push(num);
            if (large.size() - small.size() == 1) {
                small.push(large.top());
                large.pop();
            }
        }
        else {
            small.push(num);
            if (small.size() - large.size() == 2) {
                large.push(small.top());
                small.pop();
            }
        }
    }

    // Returns the median of current data stream
    double findMedian() {
        if (small.size() == large.size())
            return (small.top() + large.top()) / 2.0;
        return small.top();
    }
};
```

16. 开关灯问题

```
//We are toggling the nth bulb that much number of times as much there are factors of it. Thus number i with even number of factors will be off and bulb at number j with odd number of factors will be on.
//And only square numbers have odd number of factors. E.g 1(only 1), 4(1,2,4) ,  9(1,3,9), 16(1,2,4,8,16)...and so on...thus we have to find number of perfect squares within n which can be simply reduced to square root of n.
return (int)sqrt(n);
```

17 编辑距离

问题：

给定两个字符串 A和B，由A转成B所需的最少编辑操作次数。允许的编辑操作包括将一个字符替换成另一个字符，插入一个字符，删除一个字符。
例如将A(kitten)转成B(sitting)：
sitten （k→s）替换
sittin （e→i）替换
sitting （→g）插入

思路：

如果我们用 i 表示当前字符串 A 的下标，j 表示当前字符串 B 的下标。 如果我们用d[i, j] 来表示A[1, ... , i] B[1, ... , j] 之间的最少编辑操作数。那么我们会有以下发现：

1. d[0, j] = j;

2. d[i, 0] = i;

3. d[i, j] = d[i-1, j - 1] if A[i] == B[j]

4. d[i, j] = min(d[i-1, j - 1], d[i, j - 1], d[i-1, j]) + 1 if A[i] != B[j]

```
class Solution {
public:
    int mymin (int x, int y, int z) {
```

~~~
        if (x < y) {
            if (x < z)  return x;
            return z;
        }
        if (y < z)  return y;
        return z;
    }


    int minDistance(string word1, string word2) {
        int m, n, i, j;
        m = word1.size();
        n = word2.size();
        vector<vector<int>> dis(m+1, vector<int>(n+1));
        for (i = 0; i <= m; i++)
            dis[i][0] = i;
        for (j = 0; j <= n; j++)
            dis[0][j] = j;
        for (i = 0; i < m; i++)
            for (j = 0; j < n; j++)
                dis[i+1][j+1] = word1[i] == word2[j]? dis[i][j]: 1 + mymin(dis[i][j+1], dis[i+1][j], dis[i][j]);
        return dis[m][n];
    }
};
```
~~~