[[LeetCode] Wildcard Matching 外卡匹配](http://www.cnblogs.com/grandyang/p/4401196.html)

Implement wildcard pattern matching with support for '?' and '\*'.

'?' Matches any single character.

'\*' Matches any sequence of characters (including the empty sequence).

The matching should cover the **entire** input string (not partial).

The function prototype should be:

bool isMatch(const char \*s, const char \*p)

Some examples:

isMatch("aa","a") → false

isMatch("aa","aa") → true

isMatch("aaa","aa") → false

isMatch("aa", "\*") → true

isMatch("aa", "a\*") → true

isMatch("ab", "?\*") → true

isMatch("aab", "c\*a\*b") → false

**public** boolean isMatch(**String** s, **String** p) {

int i = 0, j = 0;

int stari = -1, starj = -1;

**while**(i < s.length()) {

// 1. match

**if**(j < p.length() && (s.charAt(i) == p.charAt(j) || p.charAt(j) == '?')) {

i++; j++;

}

// 2. star

**else** **if**(j < p.length() && p.charAt(j) == '\*') {

// first match 0

stari = i;

starj = ++j;

}

// different number that '\*' matches

**else** **if**(stari != -1) {

// match number +1

i = ++stari;

j = starj;

}

// 3. not match and no star

**else** **return** **false**;

}

// remove last '\*' in p

**while**(j < p.length() && p.charAt(j) == '\*') j++;

**return** j == p.length();

}

[Partition Array](http://www.lintcode.com/en/problem/partition-array/)

这道题目给了我们一个数组有2n integers， 需要我们把这个数组分成n对，然后从每一对里面拿小的那个数字，把所有的加起来，返回这个sum。并且要使这个sum 尽量最大。如何让sum 最大化呢，我们想一下，如果是两个数字，一个很小，一个很大，这样的话，取一个小的数字，就浪费了那个大的数字。所以我们要使每一对的两个数字尽可能接近。我们先把nums sort 一下，让它从小到大排列，接着每次把index： 0， 2， 4...偶数位的数字加起来就可以了。

[First Missing Positive 首个缺失的正数](http://www.cnblogs.com/grandyang/p/4395963.html)

Given an unsorted integer array, find the first missing positive integer.

For example,  
Given [1,2,0] return 3,  
and [3,4,-1,1] return 2.

Your algorithm should run in *O*(*n*) time and uses constant space.

这道题让我们找缺失的首个正数，由于限定了O(n)的时间，所以一般的排序方法都不能用，最开始我没有看到还限制了空间复杂度，所以想到了用哈希表来解，这个思路很简单，第一遍遍历数组把所有的数都存入哈希表中，并且找出数组的最大值，下次循环从1开始递增找数字，哪个数字找不到就返回哪个数字，如果一直找到了最大的数字，则返回最大值+1，代码如下：

[复制代码](javascript:void(0);)

// NOT constant space

class Solution {

public:

int firstMissingPositive(int A[], int n) {

if (n <= 0) return 1;

unordered\_map<int, int> m;

int mx = A[0];

for (int i = 0; i < n; ++i) {

if (A[i] > 0) {

m[A[i]] = 1;

mx = max(mx, A[i]);

}

}

for (int i = 1; i <= mx; ++i) {

if (m.find(i) == m.end()) return i;

}

return mx + 1;

}

};

[复制代码](javascript:void(0);)

但是上面的解法不是O(1)的时间复杂度，所以我们需要另想一种解法，既然不能建立新的数组，那么我们只能覆盖原有数组，我们的思路是把1放在数组第一个位置A[0]，2放在第二个位置A[1]，即需要把A[i]放在A[A[i] - 1]上，那么我们遍历整个数组，如果A[i] != i + 1, 而A[i]为整数且不大于n，另外A[i]不等于A[A[i] - 1]的话，我们将两者位置调换，如果不满足上述条件直接跳过，最后我们再遍历一遍数组，如果对应位置上的数不正确则返回正确的数，代码如下：

[复制代码](javascript:void(0);)

class Solution {

public:

int firstMissingPositive(int A[], int n) {

int i = 0;

while (i < n) {

if (A[i] != i + 1 && A[i] > 0 && A[i] <= n && A[i] != A[A[i] - 1]) {

swap(A[i], A[A[i] - 1]);

} else {

++ i;

}

}

for (i = 0; i < n; ++i) {

if (A[i] != i + 1) return i + 1;

}

return n + 1;

}

};

[复制代码](javascript:void(0);)

* [(365) Count 1 in Binary](http://www.lintcode.com/en/problem/count-1-in-binary/) 二进制中1 的个数

Count how many 1 in binary representation of a 32-bit integer.

Example

Given 32, return 1

Given 5, return 2

Given 1023, return 9

x & (x - 1) 的含义为 把二进制数中最右边的1变为0

public int countOnes(int num) {

int count = 0;

while (num != 0) {

num = num & (num - 1);

count++;

}

return count;

}