# 动态规划

## Distinct Subsequences

Given a string **S** and a string **T**, count the number of distinct subsequences of **T** in **S**.

A subsequence of a string is a new string which is formed from the original string by deleting some (can be none) of the characters without disturbing the relative positions of the remaining characters. (ie, "ACE" is a subsequence of "ABCDE" while "AEC" is not).

Here is an example:  
**S** = "rabbbit", **T** = "rabbit"

Return 3.

题意分析：

题目的意思呢，就是说给定两个字符串S和T，求S的子字符串和T相同的共有多少个。

既然是动态规划呢，自然有递推公式：

假设f(i, j)表示T[0, j]在S[0, i]里出现的次数。那么如果

* S[i] != T[j]，那么f(i, j) = f(i-1, j)
* S[i] == T[j]，那么f(i, j) = f(i-1, j) + f(i-1, j-1)

代码如下：

class Solution {

public:

int numDistinct(string S, string T) {

vector<int> f(T.size() + 1);

f[0] = 1;

for (int i = 0; i < S.size(); i++) {

for (int j = T.size() - 1; j >= 0; j--) {

f[j + 1] += (S[i] == T[j]) ? f[j] : 0;

}

}

return f[T.size()];

}

};

如果按照一般的动态规划，空间复杂度应该为O(m\*n)，但是在此段代码中，却只用了O(n).因为每推进一个i，都只与i-1有关。同时j从T.size()到0又保证了公式的公式的正确演进。

## Pow(x, n)

Implement pow(x, n).

题意分析：此题最容易想到的就是使用时间复杂度为O(n)的算法，但是显然不是最高效的。应该使用分治法来解决这个问题，可以把x^n分解为x^(n/2)\*x^(n/2)，这样就把效率提高了一倍。但是还应该特别考虑边界值，比如x=0, n<0时的情况。

class Solution {

public:

double pow(double x, int n) {

if (equal(x, 0.0) && n < 0) {

throw runtime\_error("invalid input");

}

if (n < 0 ) return 1.0 / power(x, -n);

return power(x,n);

}

private:

double power(double x, int n) {

if (n == 0) return 1;

if (n == 1) return x;

double half = power(x, n >> 1);

if (n & 0x1) return half\*half\*x;

return half \* half;

}

bool equal(double x, double y) {

if ( (x - y > -0.000001) && (x - y < 0.000001) ) {

return true;

}

return false;

}

};