**Problem Name:** Cracking the safe

**Topics:**

**Companies:**

**Level:** Hard

**Language:** C++

**Problem Statement:**

**Input Format:**

**Output Format:**

**Constraints:**

**Examples:**

**Brute force Solution:**

**Explanation:**

**Code:**

**Time Complexity**: O(N2)

**Space Complexity:** O(1)

**Optimized Solution:**

**Explanation:**

In order to guarantee to open the box at last, the input password ought to contain all length-n combinations on digits [0..k-1] - there should be k^n combinations in total.

To make the input password as short as possible, we'd better make each possible length-n combination on digits [0..k-1] occurs **exactly once** as a substring of the password. The existence of such a password is proved by [De Bruijn sequence](https://en.wikipedia.org/wiki/De_Bruijn_sequence#Uses):

A de Bruijn sequence of order n on a size-k alphabet A is a cyclic sequence in which every possible length-n string on A occurs exactly once as a substring. It has length k^n, which is also the number of distinct substrings of length n on a size-k alphabet; de Bruijn sequences are therefore optimally short.

We reuse last n-1 digits of the input-so-far password as below:

e.g., n = 2, k = 2

all 2-length combinations on [0, 1]:

00 (`00`110),

01 (0`01`10),

11 (00`11`0),

10 (001`10`)

the password is 00110

We can utilize **DFS** to find the password:

**goal**: to find the shortest input password such that each possible n-length combination of digits [0..k-1] occurs exactly once as a substring.

**node**: current input password

**edge**: if the last n - 1 digits of node1 can be transformed to node2 by appending a digit from 0..k-1, there will be an edge between node1 and node2

**start node**: n repeated 0's  
**end node**: all n-length combinations among digits 0..k-1 are visited

**visitedComb**: all combinations that have been visited

**Code:**

int n, k, total;

bool backtrack(string& ans, unordered\_set<string>& visited){

if(visited.size() == total){

return true;

}else{

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

ans.push\_back('0'+i);

string cur = ans.substr(ans.size()-n);

if(visited.find(cur) == visited.end()){

visited.insert(cur);

if(backtrack(ans, visited)) return true;

visited.erase(cur);

}

ans.pop\_back();

}

return false;

}

}

string crackSafe(int n, int k) {

this->n = n;

this->k = k;

total = pow(k, n);

string ans(n, '0');

unordered\_set<string> visited = {ans};

backtrack(ans, visited);

return ans;

}

**Time Complexity**: O(k^(2n))

**Space Complexity:** O(k^n)