算法设计之回溯算法

(1)401. 二进制手表

Description

二进制手表是指，时（0-11）用 4 位二进制表示，分（0-59）用 6 位二进制表示
给定一个非负整数 n 代表二进制数1的个数，返回所有可能的时间。。

Solution

粗暴遍历一个时钟（12小时，每小时60分钟） 12 * 60 = 7200次
亮点是：int 转 string 的方法 : string s = to_string(val)
计算数转为二进制中会有几个1 : __builtin_popcount

```
#include <vector>
#include <iostream>
using namespace std;

class Solution {
private:
    string gao(int h, int m) {
        string result;
        result += to_string(h) + ":";
        if(m<10) result += "0";
        result += to_string(m);
        return result;
    }
public:
    vector<string> readBinaryWatch(int num) {
        vector<string> result;
```

```
            for (int h = 0; h < 12; h++) {
                for (int m = 0; m < 60; m++)
                    if (__builtin_popcount(h) + __builtin_popcount(m) == num)
                        result.push_back(gao(h, m));
            }
            return result;
        }
    };

    void output(vector<string> str){
        for(int i=0; i<str.size(); i++){
            cout<<str[i]<<endl;
        }
    }

    int main(){
        int n = 2;
        Solution solu;
        output(solu.readBinaryWatch(1));
        return 0;
    }
```

---

(2)78. 子集

Description

给定一组不含重复元素的整数数组 nums，返回该数组所有可能的子集（幂集）。说明：解集不能包含重复的子集。

Solution

不断迭代，在原有的行基础上复制改行,并在尾部加上新元素
例如拿【1,2,3】来说
第一次： {}
第二次(在原有的行基础加上新元素1) ：{}、{1}
第三次(在原有的行基础加上新元素2) ： {}、{1}、{2}、{1,2}
第四次(在原有的行基础加上新元素3) ： {}、{1}、{2}、{1,2}、{3}、{1,3}、{2,3}、{1,2,3}

```
#include <vector>
#include <iostream>
using namespace std;

class Solution {
```

```
public:
    vector<vector<int> > subsets(vector<int> nums) {
        vector<vector<int> > result;
        result.push_back({});
        for(auto elem : nums){
            for(int i=result.size()-1; i>=0; i--){
                result.push_back(result[i]);
                result.back().push_back(elem);
            }
        }
        return result;
    }
};

void output(vector<vector<int> > matrix){
    for(auto line : matrix){
        for(auto elem : line) cout << elem << ',';
        cout << endl;
    }
}

int main(){
    Solution solu;
    vector<int> nums = {1,2,3};
    output(solu.subsets(nums));
    return 0;
}
```

使用回溯递归法来解
递归作用和这句 for(auto elem : nums) 异曲同工

```
#include <vector>
#include <iostream>
using namespace std;

class Solution {
public:
    vector<vector<int> > subsets(vector<int> nums) {
        if(nums.empty())  return {{}}; //递归结束

        int elem = nums.back();nums.pop_back();
        vector<vector<int> > result = subsets(nums);
        for(int i=result.size()-1; i>=0; i--){
            result.push_back(result[i]);
            result.back().push_back(elem); //result最后一行增加元素 elem
        }
        return result;
    }
};
```

```
void output(vector<vector<int> > matrix){
    for(auto line : matrix){
        cout << '[';
        for(auto elem : line){
            cout << elem << ',';
        }
        cout << ']' << endl;
    }
}

int main(){
    Solution solu;
    vector<int> nums = {1,2,3};
    output(solu.subsets(nums));
    return 0;
}
```

---

(3)22. 括号生成

Description

给出 n 代表生成括号的对数，请你写出一个函数，使其能够生成所有可能的并且有效的括号组合。

Solution

迭代，核心代码是
for (auto e1 : p1) for (auto e2 : p2) {
result.push_back(“(” + e1 + “)” + e2);
}

- n=1时括号等于 (0,0) ,意思是p1为n=0的括号组合，p2为n=0的括号组合
- n=2时括号等于 (0,1) + (1,0) ,意思是p1为n=0的括号组合，p2为n=1的括号组合 + p1为n=1的括号组合，p2为n=0的括号组合
- n=3时括号等于 (0,2) + (1,1) + (2,0)
- n=4时括号等于 (0,3) + (1,2) + (2,1) + (3,0)

```
#include <vector>
#include <iostream>
using namespace std;

class Solution {
public:
```

```
    vector<string> generateParenthesis(int n) {
        vector<string> result;
        if (n == 0) {
            result.push_back("");
            return result;
        }
        for (int right = 1; right < 2 * n; right += 2) {
            //根据right，分成左右两部分，分别返回组合情况
            auto p1 = generateParenthesis(right - 1 >> 1);
            auto p2 = generateParenthesis(2 * n - right - 1 >> 1);
            //核心
            for (auto e1 : p1) for (auto e2 : p2)
                    result.push_back("(" + e1 + ")" + e2);
        }
        return result;
    }
};

void output(vector<string> str){
    for(auto elem : str) cout << elem << endl;
}

int main(){
    Solution solu;
    output(solu.generateParenthesis(3));
    return 0;
}
```

(4)46. 全排列

Description

给定一个没有重复数字的序列，返回其所有可能的全排列。

Solution

通过 递归 + 交换(第i位和其后nums所有数都交换)
第1位和其后所有数交换 第2位和其后所有数交换
[1,2,3]->[1,2,3]->[1,2,3]
->[1,3,2]
-> [2,1,3] -> [2,1,3]
-> [2,3,1]
-> [3,2,1] -> [3,2,1]
-> [3,1,2]

```
#include <vector>
#include <iostream>
using namespace std;

class Solution{
public:
    void dfs(vector<int> nums, vector<vector<int>>& result, int cur){
        if(cur == nums.size()) { //结束条件
            result.push_back(nums);
            return;
        }
        for(int i=cur; i<nums.size(); i++){
            swap(nums[cur], nums[i]);
            dfs(nums, result, cur+1);
            swap(nums[i], nums[cur]); //很重要，恢复原状
        }
    }
    vector<vector<int>> permute(vector<int>& nums) {
        vector<vector<int>> result;
        dfs(nums, result, 0);
        return result;
    }
}


void output(vector<vector<int> > matrix){
    for(auto line : matrix) {
        for(auto elem : line) cout << elem << ' ';
        cout << endl;
    }
}

int main(){
    Solution solu;
    vector<int> nums = {1,2,3};
    output(solu.permute(nums));
    return 0;
}
```

---

(5)89. 格雷编码

Description

给定一个非负整数 n，代表格雷编码的位数，要求打印其格雷编码序列。格雷编码序列必须以 0 开头。

Solution

有如下规律：长度 1 的格雷码序列是 0、1

把长度 1 的格雷码镜像复制一份，并在所有复制上最高位补一个0，这个序列为 00、01

把长度 1 的格雷码镜像复制一份，并在所有复制上最高位补一个1，序列改为倒序，这个序列为 11、10

所以长度为 2 的格雷码序列 00、01、11、10

所以长度为 3 的格雷码序列 000、001、011、010、110、111、101、100

… …

根据这个规律不断迭代

```
#include <vector>
#include <iostream>
using namespace std;

class Solution {
public:
    vector<int> grayCode(int n) {
        if(n==0){
            return {0};
        }
        auto result = grayCode(n - 1);
        for(int i=result.size()-1; i>=0; i--) {
            result.push_back(1<<(n-1) | result[i]); //含义是result[i]的最高位补一个1
        }
        return result;
    }
};

void output(vector<int> nums){
    for(auto elem : nums) cout << elem << ' ';
}

int main(){
    Solution solu;
    output(solu.grayCode(3));
    return 0;
}
```

(6)17. 电话号码的字母组合

Description

给定一个仅包含数字 2-9 的字符串，返回所有它能表示的字母组合。

Solution

思路很简单，排序组合： 2-abc 3-def

a -> ad
ae
af
b -> bd
be
bf
c -> cd
ce
cf

```
#include <vector>
#include <string>
#include <iostream>
#include <algorithm>
using namespace std;

class Solution {
public:
    void dfs(string digits, vector<string> &phoneletter, string phrase, v
ector<string> &result){
        if(digits.empty()) {
            result.push_back(phrase);
            return;
        }

        int i=digits.front()-'0'; digits.erase(digits.begin());//从头部弹出
元素
        for(auto c : phoneletter[i]){
            dfs(digits, phoneletter, phrase+c, result);
        }
    }

    vector<string> letterCombinations(string digits){
        if(digits.empty()) return {};

        vector<string> phoneletter = {"","", "abc", "def", "ghi", "jkl",
"mno", "pqrs", "tuv", "wxyz"};
        vector<string> result;
        dfs(digits, phoneletter, "", result);
```

```
        return result;
    }
};

void output(vector<string> phrase){
    for(auto elem : phrase)  cout << elem << ' ';
}

int main(){
    Solution solu;
    output(solu.letterCombinations("23"));
    return 0;
}
```

---

(7)93. 复原IP地址

Description

给定一个只包含数字的字符串，复原它并返回所有可能的 IP 地址格式。(分四段，0~255)
输入: “25525511135” , 输出: [“255.255.11.135”, “255.255.111.35”]

Solution

用递归遍历例举所有组合，满足条件的组合加入
但其实没有遍历所有，算法中含有剪枝
就是说不是分割完四段后再判断是否符合Ip地址格式
而是每分一段（part），都要判断该段是否符合格式 if(ok(part))
符合格式的才拿来再往下组合
直到分到第四段，。。。

```
#include <vector>
#include <iostream>
using namespace std;

class Solution {
public:
    bool ok(string part) { //判断是否能作为ip地址子串 0~255
        if(part.empty() || part.size()>3 || (part.size()>1&&part[0]=='0'
)) return false;
        return stoi(part) <= 255;
    }
    void dfs(int step, string s, string temp, vector<string> &result) {
        if(step == 4) {
```

```
            if(ok(s)) result.push_back(temp + s);
            return;
        }
        for(int len=1; len<=min(3, (int)s.size()); len++) {
            string part = s.substr(0, len);
            if(ok(part)) dfs(step+1, s.substr(len), temp+part+".", resul
t);
        }
    }
    vector<string> restoreIpAddresses(string s){
        vector<string> result;
        dfs(1, s, "", result);
        return result;
    }
};

void output(vector<string> text){
    for(auto line : text)  cout << line << endl;
}

int main(){
    Solution solu;
    output(solu.restoreIpAddresses("25525511135"));
    return 0;
}
```

---

(8)357. 计算各个位数不同的数字个数

Description

给定一个非负整数 n，计算各位数字都不同的数字 x 的个数，其中 0 ≤ x < 10n 。

Solution

本题是概率统计中的排列问题
有点类似于 从0~9这 10 个数中挑出 n 个组成一个n位数，有几种排列方案， 但又不完全一样，

n=0，即[0,1)：1个，
n=1，即[0,10)：10个，1 + 9 = 10
n=2，即[0,100)：91个， 1 + 9 + 9*9 = 91
n=3，即[0,1000)：739个，1 + 9 + 9*9 + 9*9*8 = 739
n=4，即[0,10000)：5275个，1 + 9 + 9*9 + 9*9*8 + 9*9*8*7 = 5275
... ...
利用

```
f(1)=9
f(2)=9*9=f(1)*9
f(3)=9*9*8=f(2)*8
```

```
#include <vector>
#include <cmath>
#include <iostream>
using namespace std;

class Solution {
public:
    int countNumbersWithUniqueDigits(int n) {
        int result = 1;
        if(n>=1) result += 9;
        int f = 9;//f(1)=9  f(2)=9*9   f(3)=9*9*8
        for(int i = 2; i <= n; i++){
            f *= 11-i;
            result += f;
        }
        return result;
    }
};

int main(){
    Solution solu;
    cout << solu.countNumbersWithUniqueDigits(9);
    return 0;
}
```

(9)90. 子集 II

Description

给定一个可能包含重复元素的整数数组 nums，返回该数组所有可能的子集（幂集）。说明：解集不能包含重复的子集。

Solution

这题与《78. 子集》思路有类似。
因为 《78. 子集》所给nums是不重复的，而本题所给nums是有重复的
解本题很重要的点是要排序
然后把相同的元素划分成一组
然后加元素时：一组一组的考虑。《78. 子集》是一个元素一个元素考虑

```cpp
#include <vector>
#include <iostream>
#include <algorithm>
using namespace std;

class Solution {
public:
    vector<vector<int>> subsetsWithDup(vector<int>& nums) {
        sort(nums.begin(), nums.end());
        if (nums.empty())  return {{}};

        int elem = nums.back(), cnt = 0;
        while (!nums.empty() && nums.back() == elem) { //重复的划成一组
            cnt++;
            nums.pop_back();
        }

        auto result = subsetsWithDup(nums);
        for (int i=result.size()-1; i>=0; i-- ) {
            auto temp = result[i];
            for (int j=1; j<=cnt; j++) {
                temp.push_back(elem);
                result.push_back(temp);
            }
        }
        return result;
    }
};

void output(vector<vector<int> > matrix){
    for(auto line : matrix){
        for(auto elem : line) cout << elem << ' ';
        cout << endl;
    }
}

int main(){
    Solution solu;
    vector<int> nums = {1,2,2};
    output(solu.subsetsWithDup(nums));
    return 0;
}
```