一些说明：

1．图的m着色从1个颜色开始进行，直到某个颜色数找到了第一个解决方案然后停止。．

2．总结：回溯法扫描的点比分支限界法要多，并且在点多的情况下运行时间比分支限界法长（点越多长的越多），并且由于扫描方式不同，最佳路径不同（路径长度相等）

3．N皇后中，$t(n) = O(n^{2.8})$，线性表达式为$\lg(t(n) = 2.806882 * \lg(n) – 11.749752$

注：为了方便读取，将xls文件改为csv

注：图的m着色和旅行商问题在cpp文件下，主函数中依次进行各个算法，N皇后问题只修改了主函数

代码：

1. 图的 m 着色和旅行商问题

```
// Algorithm4.cpp : Designed by Xiao Yunming.
//

#include "stdafx.h" // VS projects head file

#include <iostream>
#include <fstream>
#include <string>
#include <vector>
#include <map>
#include <queue>
#include <algorithm>
#include <cmath>
#include <functional>
#include <time.h>

using namespace std;

// ---------------------------- M_COLOUR -----------------------------------

#define INF 99999

class MColor
{
public:
    int n, m;
    unsigned long long count[2];
    vector<vector<bool>> matrix;
    int *x;
    vector<vector<int>> result;
    unsigned long long sum;

    MColor(vector<vector<double>> matrix, int m, clock_t &start, clock_t &end);
    ~MColor();

    void BackTrack(int t);
    bool OK(int k);
};

MColor::MColor(vector<vector<double>> matrix, int m, clock_t &start, clock_t &end)
{
    n = matrix.size();
    this->m = m;
    this->x = new int[n];
    for (int i = 0; i < n; i++) {
        x[i] = -1;
        vector<bool> v;
        this->matrix.push_back(v);
        for (int j = 0; j < n; j++) {
            if (matrix[i][j] == INF)
                this->matrix[i].push_back(false);
            else
                this->matrix[i].push_back(true);
        }
    }
    this->sum = 0;
    this->count[0] = 0;
    this->count[1] = 0;

    start = clock();
    BackTrack(0);
    end = clock();
}

MColor::~MColor()
{
    this->matrix.clear();
    delete[] this->x;
}

void MColor::BackTrack(int t)
{
    if (this->sum > 0)
        return;
    if (t >= n) {
        this->sum++;
        vector<int>xx;
        for (int i = 0; i < n; i++)
            xx.push_back(x[i]);
        this->result.push_back(xx);
    }
    else {
        for (int i = 1; i <= m; i++) {
            x[t] = i;
            if (OK(t))
                BackTrack(t + 1);
            x[t] = -1;
        }
    }
}

bool MColor::OK(int k)
{
    for (int j = 0; j < n; j++) {
        this->count[0]++;
        if (this->count[0] == 0)
            this->count[1]++;
        if (matrix[k][j] == true && x[j] == x[k])
            return false;
    }
    return true;
}

// -------------------------- TSP_BACKTRACK --------------------------------

class BTTSP
{
public:
    int n, *x, *bestx, count;
    double best_cost, current_cost;
    vector<vector<double>> a;

    BTTSP(vector<vector<double>> matrix);
    ~BTTSP();

    void BackTrack(int t);
};

BTTSP::BTTSP(vector<vector<double>> matrix)
```

```
{
    this->a = matrix;
    n = this->a.size();
    x = new int[n];
    bestx = new int[n];
    for (int i = 0; i < n; i++) {
        x[i] = i;
        bestx[i] = i;
    }
    best_cost = INF;
    current_cost = 0;
    count = 0;

    BackTrack(1);
}

BTTSP::~BTTSP()
{
    delete[] x;
    delete[] bestx;
    a.clear();
}

void BTTSP::BackTrack(int i)
{
    if (i == n - 1) {
        if (a[x[n - 2]][x[n - 1]] < INF &&
            a[x[n - 1]][x[0]] < INF &&
            (best_cost == INF || current_cost + a[x[n - 2]][x[n - 1]] + a[x[n - 1]][x[0]] < best_cost))
        {
            for (int j = 0; j < n; j++)
                bestx[j] = x[j];
            best_cost = current_cost + a[x[n - 2]][x[n - 1]] + a[x[n - 1]][x[0]];
        }
    }
    else {
        for (int j = i; j < n; j++) {
            if (a[x[i - 1]][x[j]] < INF &&
                (best_cost == INF || current_cost + a[x[i - 1]][x[j]] < best_cost))
            {
                count++;
                swap(x[i], x[j]);
                current_cost += a[x[i - 1]][x[i]];
                BackTrack(i + 1);
                current_cost -= a[x[i - 1]][x[i]];
                swap(x[i], x[j]);
            }
        }
    }
}

// ----------------------- TSP_BRANCH_BOUNDING ----------------------------

class BBTSP
{
public:
    class HeapNode
    {
    public:
        double low_cost, current_cost, rest_cost;
        int s;
        vector<int> x;

        HeapNode(double lc, double cc, double rc, int ss, vector<int> xx) {
            this->low_cost = lc;
            this->current_cost = cc;
            this->rest_cost = rc;
            this->s = ss;
            this->x = xx;
        }
        HeapNode(const HeapNode &h) {
            this->low_cost = h.low_cost;
            this->current_cost = h.current_cost;
            this->rest_cost = h.rest_cost;
            this->s = h.s;
            this->x = h.x;
        }
        ~HeapNode()
        {
            this->x.clear();
        }

        bool operator<(const HeapNode &h) const { return this->low_cost > h.low_cost; }; // here we do so to
change the priority_queue to min_heap
    };

    vector<vector<double>> a;
    int n;
    double best_cost;
    vector<int> bestx;
    int count;

    BBTSP(vector<vector<double>> matrix);
    ~BBTSP();
    BBTSP(const BBTSP &b);
};

BBTSP::BBTSP(vector<vector<double>> matrix)
{
    this->a = matrix;
    int n = a.size();
    priority_queue<HeapNode> heap;
    double *min_out = new double[n];
    double min_sum = 0;
    for (int i = 0; i < n; i++) {
        double min = INF;
        for (int j = 0; j < n; j++)
            if (a[i][j] < INF && a[i][j] < min)
                min = a[i][j];

        if (min == INF)
            return;
        min_out[i] = min;
        min_sum += min;
    }

    vector<int> x;
    for (int i = 0; i < n; i++)
        x.push_back(i);
    HeapNode enode(0, 0, min_sum, 0, x);
    heap.push(enode);
    this->best_cost = INF;
    this->count = 0;
    while (heap.empty() == false && enode.s < n - 1)
    {
        x = enode.x;
        if (enode.s == n - 2) {
            if (a[x[n - 2]][x[n - 1]] < INF &&
                a[x[n - 1]][x[0]] < INF &&
                enode.current_cost + a[x[n - 2]][x[n - 1]] + a[x[n - 1]][x[0]] < best_cost)
            {
                this->best_cost = enode.current_cost + a[x[n - 2]][x[n - 1]] + a[x[n - 1]][x[0]];
                this->bestx = x;
                enode.current_cost = best_cost;
                enode.low_cost = best_cost;
                enode.s++;
                heap.push(enode);
            }
        }
        else {
            for (int i = enode.s + 1; i < n; i++) {
                if (a[x[enode.s]][x[i]] < INF) {
                    double current_cost = enode.current_cost + a[x[enode.s]][x[i]];
                    double rest_cost = enode.rest_cost - min_out[x[enode.s]];
                    double b = current_cost + rest_cost;
                    if (b < best_cost) {
                        this->count++;
                        vector<int> xx = x;
                        xx[enode.s + 1] = x[i];
                        xx[i] = x[enode.s + 1];
                        HeapNode node(b, current_cost, rest_cost, enode.s + 1, xx);
                        heap.push(node);
                    }
                }
            }
        }
        enode = heap.top();
        heap.pop();
    }

    //this->bestx = x;
    delete[] min_out;
}

BBTSP::~BBTSP()
{
    a.clear();
    bestx.clear();
}

BBTSP::BBTSP(const BBTSP &b)
{
    this->a = b.a;
    this->n = b.n;
    this->count = b.count;
    this->best_cost = b.best_cost;
    this->bestx = b.bestx;
}

// ------------------------------ MAIN ------------------------------------

int main()
{
    ofstream fr("Result.txt");
    clock_t start, end;
    double duration;
    string temp;

    std::function<vector<double>(string, string)> StringParse; // to parse the long line to pieces with
"delim"
    StringParse = [](string s, string delim)
    {
        vector<string> str;
        vector<double>result_int;
        size_t pos = 0;
        size_t len = s.length();
        size_t delim_len = delim.length();
        while (pos < len) {
            int find_pos = s.find(delim, pos);
            if (find_pos < 0) {
                str.push_back(s.substr(pos, len - pos));
                break;
            }
            str.push_back(s.substr(pos, find_pos - pos));
            pos = find_pos + delim_len;
        }

        vector<string>::iterator it = str.begin();
        while (it != str.end()) {
            it->erase(0, it->find_first_not_of(' '));
            result_int.push_back(stod(*it));
            it++;
        }
        return result_int;
    };


    fstream f21("附件1-1.22基站图的邻接矩阵-v2-20170601.csv", ios::in | ios::out);
    fstream f22("附件1-1.30基站图的邻接矩阵-v2-20170601.csv", ios::in | ios::out);
    fstream f23("附件1-1.42基站图的邻接矩阵-v2-20170601.csv", ios::in | ios::out);
    vector<vector<double>> c21, c22, c23;
    map<int, int> m21, m22, m23;   // the order number (starting from 0) mapping to the enodebid

    getline(f21, temp);       getline(f21, temp);
    while (getline(f21, temp) && f21.good()) {
        vector<double> x = StringParse(temp, ",");
        m21.insert(map<int, int>::value_type(((int)x[0] - 1), (int)x[1]));
        x.erase(x.begin(), x.begin() + 2);
        c21.push_back(x);
    }
    getline(f22, temp);       getline(f22, temp);
    while (getline(f22, temp) && f22.good()) {
        vector<double> x = StringParse(temp, ",");
        m22.insert(map<int, int>::value_type(((int)x[0] - 1), (int)x[1]));
        x.erase(x.begin(), x.begin() + 2);
        c22.push_back(x);
    }
    getline(f23, temp);       getline(f23, temp);
    while (getline(f23, temp) && f23.good()) {
        vector<double> x = StringParse(temp, ",");
        m23.insert(map<int, int>::value_type(((int)x[0] - 1), (int)x[1]));
        x.erase(x.begin(), x.begin() + 2);
        c23.push_back(x);
    }

    fr << "22节点：" << endl;
    for (int i = 5; i <= 22; i++) {
        MColor M(c21, i, start, end);
        duration = (double)(end - start);
        fr << "m = " << i << ", solution = " << M.sum << ", L = ";
        if (M.count[1] != 0)
            fr << M.count[1] << "*2^64 + ";
        fr << M.count[0] << ", time cost = " << duration / CLOCKS_PER_SEC << endl;
        if (M.sum != 0) {
            auto it = M.result.begin();
            while (it != M.result.end()) {
                auto itt = it->begin();
                while (itt != it->end()) {
                    fr << *itt << ", ";
                    itt++;
                }
                fr << endl;
                it++;
            }
            break;
        }
    }
    fr << endl << endl;

    fr << "30节点：" << endl;
    for (int i = 4; i <= 30; i++) {
        MColor M(c22, i, start, end);
        duration = (double)(end - start);
        fr << "m = " << i << ", solution = " << M.sum << ", L = ";
        if (M.count[1] != 0)
            fr << M.count[1] << "*2^64 + ";
        fr << M.count[0] << ", time cost = " << duration / CLOCKS_PER_SEC << endl;
        if (M.sum != 0) {
            auto it = M.result.begin();
            while (it != M.result.end()) {
                auto itt = it->begin();
                while (itt != it->end()) {
                    fr << *itt << ", ";
                    itt++;
                }
                fr << endl;
                it++;
            }
            break;
        }
    }
    fr << endl << endl;

    fr << "42节点：" << endl;
    for (int i = 5; i <= 42; i++) {
        MColor M(c23, i, start, end);
        duration = (double)(end - start);
        fr << "m = " << i << ", solution = " << M.sum << ", L = ";
        if (M.count[1] != 0)
            fr << M.count[1] << "*2^64 + ";
        fr << M.count[0] << ", time cost = " << duration / CLOCKS_PER_SEC << endl;
        if (M.sum != 0) {
            auto it = M.result.begin();
            while (it != M.result.end()) {
                auto itt = it->begin();
                while (itt != it->end()) {
                    fr << *itt << ", ";
                    itt++;
                }
                fr << endl;
                it++;
            }
            break;
        }
    }

    f21.close();
    f22.close();
    f23.close();


    fstream f31("附件1-2.15基站图的邻接矩阵-v4-20160613.csv", ios::in | ios::out);
    fstream f32("附件1-2.20基站图的邻接矩阵-v4-20160613.csv", ios::in | ios::out);
    fstream f33("附件1-2.22基站图的邻接矩阵-v4-20160613.csv", ios::in | ios::out);
    vector<vector<double>> c31, c32, c33;
    map<int, int> m31, m32, m33;   // the order number (starting from 0) mapping to the enodebid

    getline(f31, temp);       getline(f31, temp);
    int tempi31 = 0, tempi32 = 0, tempi33 = 0;
    while (getline(f31, temp) && f31.good()) {
        vector<double> x = StringParse(temp, ",");
        m31.insert(map<int, int>::value_type(tempi31, ((int)x[0])));
        x.erase(x.begin(), x.begin() + 2);
        c31.push_back(x);
        tempi31++;
    }
    getline(f32, temp);       getline(f32, temp);
    while (getline(f32, temp) && f32.good()) {
        vector<double> x = StringParse(temp, ",");
        m32.insert(map<int, int>::value_type(tempi32, ((int)x[0])));
        x.erase(x.begin(), x.begin() + 2);
        c32.push_back(x);
        tempi32++;
    }
    getline(f33, temp);       getline(f33, temp);
    while (getline(f33, temp) && f33.good()) {
        vector<double> x = StringParse(temp, ",");
        m33.insert(map<int, int>::value_type(tempi33, ((int)x[0])));
        x.erase(x.begin(), x.begin() + 2);
        c33.push_back(x);
        tempi33++;
    }
    f31.close();
    f32.close();
    f33.close();

    vector<vector<double>> temp3 = c31;
    c31[0] = temp3[12];
    c31[12] = temp3[0];
    for (int i = 0; i < temp3.size(); i++) {
        c31[i][0] = temp3[i][12];
        c31[i][12] = temp3[i][0];
    }
    m31.erase(0);
    m31.erase(12);
    m31.insert(map<int, int>::value_type(0, 20));
    m31.insert(map<int, int>::value_type(12, 3));
    temp3 = c32;
    c32[0] = temp3[17];
    c32[17] = temp3[0];
    for (int i = 0; i < temp3.size(); i++) {
        c32[i][0] = temp3[i][17];
        c32[i][17] = temp3[i][0];
    }
    m32.erase(0);
    m32.erase(17);
    m32.insert(map<int, int>::value_type(0, 20));
    m32.insert(map<int, int>::value_type(17, 1));
    temp3 = c33;
    c33[0] = temp3[19];
    c33[19] = temp3[0];
    for (int i = 0; i < temp3.size(); i++) {
        c33[i][0] = temp3[i][19];
        c33[i][19] = temp3[i][0];
    }
    m33.erase(0);
    m33.erase(19);
    m33.insert(map<int, int>::value_type(0, 20));
    m33.insert(map<int, int>::value_type(19, 1));


    fr << "回溯法：" << endl;
    fr << "15节点：" << endl;
    start = clock();
    BTTSP B31(c31);
    end = clock();
    duration = (double)(end - start);
    fr << "最短路径为: " << m31[B31.bestx[0]];
    for (int i = 1; i < c31.size(); i++)
        fr << " -> " << m31[B31.bestx[i]];
    fr << endl << "路程为: " << B31.best_cost << "，访问节点数: " << B31.count << endl;
    fr << "耗时" << duration / CLOCKS_PER_SEC << "秒" << endl;
    fr << endl << endl;

    fr << "20节点：" << endl;
```

```
    start = clock();
    BTTSP B32(c32);
    end = clock();
    duration = (double)(end - start);
    fr << "最短路径为： " << m32[B32.bestx[0]];
    for (int i = 1; i < c32.size(); i++)
        fr << " -> " << m32[B32.bestx[i]];
    fr << endl << "路程为： " << B32.best_cost << "，访问节点数： "<< B32.count << endl;
    fr << "耗时" << duration / CLOCKS_PER_SEC << "秒" << endl;
    fr << endl << endl;

    fr << "22节点：" << endl;
    start = clock();
    BTTSP B33(c33);
    end = clock();
    duration = (double)(end - start);
    fr << "最短路径为： " << m33[B33.bestx[0]];
    for (int i = 1; i < c33.size(); i++)
        fr << " -> " << m33[B33.bestx[i]];
    fr << endl << "路程为： " << B33.best_cost << "，访问节点数： " << B33.count << endl;
    fr << "耗时" << duration / CLOCKS_PER_SEC << "秒" << endl;

    fr << endl << endl;


    fr << "分支限界法：" << endl;
    fr << "15节点：" << endl;
    start = clock();
    BBTSP B41(c31);
    end = clock();
    duration = (double)(end - start);
    fr << "最短路径为： " << m31[B41.bestx[0]];
    for (int i = 1; i < c31.size(); i++)
        fr << " -> " << m31[B41.bestx[i]];
    fr << endl << "路程为： " << B41.best_cost << "，访问节点数： " << B41.count << endl;
    fr << "耗时" << duration / CLOCKS_PER_SEC << "秒" << endl;

    fr << "20节点：" << endl;
    start = clock();
    BBTSP B42(c32);
    end = clock();
    duration = (double)(end - start);
    fr << "最短路径为： " << m32[B42.bestx[0]];
    for (int i = 1; i < c32.size(); i++)
        fr << " -> " << m32[B42.bestx[i]];
    fr << endl << "路程为： " << B42.best_cost << "，访问节点数： "<< B42.count << endl;
    fr << "耗时" << duration / CLOCKS_PER_SEC << "秒" << endl;

    fr << "22节点：" << endl;
    start = clock();
    BBTSP B43(c33);
    end = clock();
    duration = (double)(end - start);
    fr << "最短路径为： " << m33[B43.bestx[0]];
    for (int i = 1; i < c33.size(); i++)
        fr << " -> " << m33[B43.bestx[i]];
    fr << endl << "路程为： " << B43.best_cost << "，访问节点数： " << B43.count << endl;
    fr << "耗时" << duration / CLOCKS_PER_SEC << "秒" << endl;

    fr << endl << endl;

    fr.close();
}
```

2. 改进后的NQueens主函数

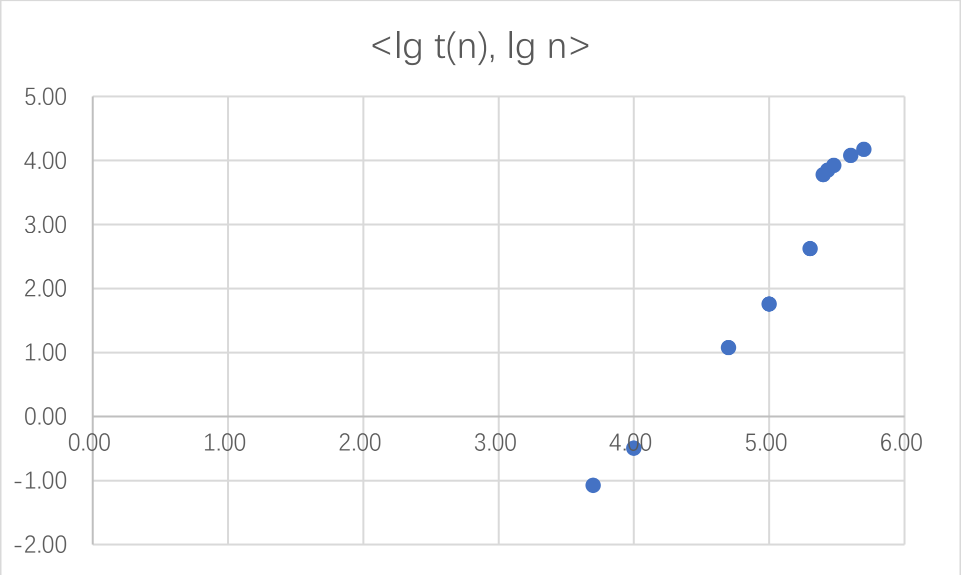
```
/*----------------------------------------------
主函数
 ----------------------------------------------*/
int main(int argc, char* argv[])
{
    clock_t start, finish;
    long m = 0, c = 0;
    start = clock();

    sscanf(argv[1],"%ld",&N);

    sqrt_N = (int) sqrt(N);

    // 分配解的空间
    pSolution = new long[N]; // 使用下标 0 - (N-1)

    // 分配正对角线上的数组空间，并赋初始值为0
    pPosDiagonal = new long[2*N-1];

    // 分配负对角线上的数组空间，并赋初始值为0
    pNegDiagonal = new long[2*N-1];

    do {
        // 产生一个随机解
        GeneratePermutation();
        m++;

        // 给两个对角线对应的皇后数数组赋值
        SetDiagonals();

        CountCollisions();

        bool flag = true;
        long gain;

        while (flag)
        {
            flag = false;
            for (long i = 0; i < N; i++)
                for (long j = i; j < N; j++)
                    // 若pSolution[i] 或 pSolution[j]对应的对角线有冲突
                    if (pPosDiagonal[i - pSolution[i] + N -1] > 1 || pNegDiagonal[i + pSolution[i]] > 1 ||
                        pPosDiagonal[j - pSolution[j] + N -1] > 1 || pNegDiagonal[j + pSolution[j]] > 1)
                    {
                        // 判断一下，如果交换pSolution[i]和pSolution[j]，是否可以降低冲突总数
                        gain = SwapEvaluate(i, j);

                        // 若交换会带来冲突的减少，则值得交换
                        if ( gain > 0)
                        {
                            flag = true; // 标记，证明本轮迭代做了交换

                            SwapQueens(i, j, gain); // 交换 2个皇后，并更新冲突总数
                            c++;
                        }
                    }
        }

    } while (NumCollisions > 0);

    finish = clock();

    double duration = (double) (finish-start)/CLOCKS_PER_SEC;
    FILE *fp = NULL;
    fp = fopen("result.txt", "a+");
    fprintf(fp, "N = %ld : time = %f seconds, m = %ld, c = %ld\n", N, duration, m, c);
    fclose(fp);
    fp = NULL;

    printf("N = %ld : time = %f seconds, m = %ld, c = %ld\n", N, duration, m, c);

    return 0;
}
```

结果部分：

1. N皇后问题

```
N = 5000 : time = 0.085000 seconds, m = 1, c = 2413
N = 10000 : time = 0.323000 seconds, m = 1, c = 4997
N = 50000 : time = 11.935000 seconds, m = 1, c = 24552
N = 100000 : time = 57.275000 seconds, m = 1, c = 49460
N = 200000 : time = 422.781000 seconds, m = 1, c = 98782
N = 250000 : time = 6010.142000 seconds, m = 1, c = 123277
N = 270000 : time = 7068.152000 seconds, m = 1, c = 132578
N = 300000 : time = 8371.916000 seconds, m = 1, c = 147634
N = 400000 : time = 12018.889000 seconds, m = 1, c = 197453
N = 500000 : time = 15005.800000 seconds, m = 1, c = 246295
```

n	5,000	10,000	50,000	100,000	200,000	250,000	270,000	300,000	400,000	500,000
m	2413	4997	24552	49460	98782	123277	132578	147634	197453	246295
t(n)	0.085	0.323	11.935	57.275	422.781	6010.142	7068.152	8371.916	12018.889	15005.8
lg t(n)	-1.07	-0.49	1.08	1.76	2.63	3.78	3.85	3.92	4.08	4.18
2 lg n	7.40	8.00	9.40	10.00	10.60	10.80	10.86	10.95	11.20	11.40
3 lg n	11.10	12.00	14.10	15.00	15.90	16.19	16.29	16.43	16.81	17.10



线性分析表达式：$\lg(t(n)) = 2.806882 * \lg(n) – 11.749752$

因此，$t(n) = O(n^{2.8})$

2. 图的 m 着色问题：

22 节点：

m = 1, solution = 0, L = 46, time cost = 0

m = 2, solution = 0, L = 856, time cost = 0

m = 3, solution = 0, L = 26544, time cost = 0
m = 4, solution = 0, L = 4097896, time cost = 0.024
m = 5, solution = 1, L = 253116, time cost = 0.003
涂色方案：1, 1, 2, 2, 2, 1, 1, 3, 3, 4, 4, 3, 4, 2, 2, 5, 4, 5, 2, 5, 5, 3,

30 节点：
m = 1, solution = 0, L = 31, time cost = 0
m = 2, solution = 0, L = 646, time cost = 0
m = 3, solution = 0, L = 6879, time cost = 0
m = 4, solution = 1, L = 1016083, time cost = 0.007
涂色方案：1, 2, 1, 2, 2, 3, 4, 1, 2, 3, 1, 4, 4, 4, 3, 2, 1, 3, 1, 1, 1, 2, 3, 3, 2, 3, 4, 4, 4, 1,

42 节点：
m = 1, solution = 0, L = 343, time cost = 0
m = 2, solution = 0, L = 31660, time cost = 0
m = 3, solution = 0, L = 13414995, time cost = 0.05
m = 4, solution = 0, L = 592815352264, time cost = 2531.11
m = 5, solution = 1, L = 6625, time cost = 0.001
涂色方案：1, 1, 1, 1, 1, 1, 1, 1, 2, 3, 1, 2, 3, 2, 4, 2, 2, 3, 4, 1, 2, 1, 3, 4, 3, 1, 4, 3, 3, 5, 2, 3, 2, 4, 4, 5, 1, 5, 2, 2, 5, 4,

问题	用到的颜色总数 *m* （色数）	搜索过的结点总数 *L*	程序运行时间 *T* （单位：s）
22 个基站	5	253116	0.003
30 个基站	4	1016083	0.007
42 个基站	5	6625	0.001

3. 旅行商问题：

回溯法：

15 节点：
最短路径为: 20 -> 9 -> 7 -> 16 -> 3 -> 13 -> 12 -> 21 -> 10 -> 8 -> 19 -> 11 -> 22 -> 5 -> 17 -> 20
路程为: 5506.88，访问节点数: 254515
耗时 0.019 秒

20 节点：
最短路径为: 20 -> 17 -> 5 -> 22 -> 11 -> 19 -> 18 -> 8 -> 1 -> 10 -> 21 -> 14 -> 12 -> 15 -> 2 -> 13 -> 3 -> 16 -> 7 -> 9 -> 20
路程为: 6987.51，访问节点数: 76201708
耗时 3.473 秒

22 节点：
最短路径为: 20 -> 17 -> 5 -> 22 -> 11 -> 19 -> 8 -> 18 -> 6 -> 4 -> 1 -> 10 -> 21 -> 14 -> 12 -> 15 -> 2 -> 13 -> 3 -> 16 -> 7 -> 9 -> 20
路程为: 7690.8，访问节点数: 486666892
耗时 26.827 秒

分支限界法：

15 节点：
最短路径为: 20 -> 17 -> 5 -> 22 -> 11 -> 19 -> 8 -> 10 -> 21 -> 12 -> 13 -> 3 -> 16 -> 7 -> 9 -> 20
路程为: 5506.88，访问节点数: 48222
耗时 0.059 秒

```
20 节点：
最短路径为: 20 -> 9 -> 7 -> 16 -> 3 -> 13 -> 2 -> 15 -> 12 -> 14 -> 21 -> 10 -> 1 -> 8 -> 18 -> 19 -> 11 -> 22 -> 5 -> 17 -> 20
路程为: 6987.51，访问节点数: 1270734
耗时 2.188 秒

22 节点：
最短路径为: 20 -> 9 -> 7 -> 16 -> 3 -> 13 -> 2 -> 15 -> 12 -> 14 -> 21 -> 10 -> 1 -> 4 -> 6 -> 18 -> 8 -> 19 -> 11 -> 22 -> 5 -> 17 -> 20
路程为: 7690.8，访问节点数: 1701067
耗时 3.662 秒
```

问题	求解算法	最短回路	路径总长度（单位：m）	搜索过的节点数	程序运行时间（单位：s）
15 个基站	回溯	20 -> 9 -> 7 -> 16 -> 3 -> 13 -> 12 -> 21 -> 10 -> 8 -> 19 -> 11 -> 22 -> 5 -> 17 -> 20	5506.88	254515	0.019
	分支限界	20 -> 17 -> 5 -> 22 -> 11 -> 19 -> 8 -> 10 -> 21 -> 12 -> 13 -> 3 -> 16 -> 7 -> 9 -> 20	5506.88	48222	0.059
20 个基站	回溯	20 -> 17 -> 5 -> 22 -> 11 -> 19 -> 18 -> 8 -> 1 -> 10 -> 21 -> 14 -> 12 -> 15 -> 2 -> 13 -> 3 -> 16 -> 7 -> 9 -> 20	6987.51	76201708	3.473
	分支限界	20 -> 9 -> 7 -> 16 -> 3 -> 13 -> 2 -> 15 -> 12 -> 14 -> 21 -> 10 -> 1 -> 8 -> 18 -> 19 -> 11 -> 22 -> 5 -> 17 -> 20	6987.51	1270734	2.188
22 个基站	回溯	20 -> 17 -> 5 -> 22 -> 11 -> 19 -> 8 -> 18 -> 6 -> 4 -> 1 -> 10 -> 21 -> 14 -> 12 -> 15 -> 2 -> 13 -> 3 -> 16 -> 7 -> 9 -> 20	7690.8	486666892	26.827
	分支限界	20 -> 9 -> 7 -> 16 -> 3 -> 13 -> 2 -> 15 -> 12 -> 14 -> 21 -> 10 -> 1 -> 4 -> 6 -> 18 -> 8 -> 19 -> 11 -> 22 -> 5 -> 17 -> 20	7690.8	1701067	3.662