

《算法分析与设计》0525 课后作业

2154312 郑博远

1. 圆排列问题

圆排列问题的进一步优化，完成代码分析复杂度。

算法尚有许多改进的余地。例如，像 $1, 2, \dots, n-1, n$ 和 $n, n-1, \dots, 2, 1$ 这种互为镜像的排列具有相同的圆排列长度，只计算一个就够了，可减少约一半的计算量。另一方面，如果所给的 n 个圆中有 k 个圆有相同的半径，则这 k 个圆产生的 $k!$ 个完全相同的圆排列，只计算一个就够了。

思路：

1. 镜像排列优化

在未剪枝的排列组合方式下，每种排列方式均有一个互为镜像的队列；因此优化时应该规定某种顺序，使最后遍历到的排列方式种仅保留二者之一。规定仅保留第一个圆的半径小于等于最后一个圆的半径的排列方式。在不考虑有半径相同的圆的前提下，容易发现这样能够剪枝掉正好一半的冗余序列。

但在实际操作时，不能在每次组合时都粗暴地采用 $r[1] \leq r[n]$ 的方式剪枝。例如：2134 \rightarrow 2431 \rightarrow 2413 的遍历过程中，会由于 2431 被剪枝而遗漏了 2413 这一合法情况。因此，在每次递归的第一个下标为 `index` 的情况下，应该遍历 `index` 开始的所有下标。若所有元素全部小于 $r[1]$ ，则无论如何排列组合都无法产生合法序列，该情况下可以被剪枝。

2. 半径相同优化

该情况下的剪枝较为简单。在每次从下标 `index` 开始遍历之后的每个元素，交换产生新序列可能时，记录当前下标 i 对应的圆半径是否出现过。若已经出现过，则意味着在 `index` 位置的该排列已经存在过，可以剪枝。

代码:

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

int N;
double minLen = 1e3;

// 计算当前结果
void callen(vector <double>& ans, vector <double>& x, vector <double>&
r)
{
    double minX = 0, maxX = 0;
    for (int i = 1; i <= N; i++) {
        minX = min(minX, x[i] - r[i]);
        maxX = max(maxX, x[i] + r[i]);
    }
    if (maxX - minX < minLen) {
        minLen = maxX - minX;
        for (int i = 1; i <= N; i++)
            ans[i] = r[i];
    }
}

// 计算圆心坐标
double getcenterX(int cur, vector <double>& ans, vector <double>& x,
vector <double>& r)
{
    double x_max = 0;
    for (int j = 1; j < cur; j++) {
        double newx;
        newx = x[j] + 2.0 * sqrt(r[cur] * r[j]);
        x_max = max(x_max, newx);
    }
    return x_max;
}
```

```

// 枚举排列情况
void permutation(int index, vector <double>& ans, vector <double>& x,
vector <double>& r)
{
    if (index == N + 1) {
        callen(ans, x, r);
    }
    else {
        // 在此部分进行对称的剪枝
        bool invalid = false;
        for (int i = index; i <= N; i++)
            // 从index开始的值是否有比数组第一个元素更大的
            if (r[i] >= r[1])
                invalid = true;
        // 若之后没有比数组第一个元素更大的值 则排列反转后已经出现过
        if (!invalid)
            return;

        // 用于记录是否重复 去除半径相同的
        map<double, int> occured;
        for (int i = index; i <= N; i++) {
            // 出现过 就跳过
            if (occured[r[i]])
                continue;
            occured[r[i]]++;

            swap(r[index], r[i]);
            double centerX = getcenterX(index, ans, x, r);
            // 有更小的总长度 提前结束
            if (centerX + r[index] + r[1] < minLen) {
                x[index] = centerX;
                permutation(index + 1, ans, x, r);
            }
            swap(r[index], r[i]);
        }
    }
}

int main()
{
    cin >> N;
    vector <double> ans(N + 1), x(N + 1), r(N + 1);
    for (int i = 1; i <= N; i++)

```

```

        cin >> r[i];

    permutation(1, ans, x, r);

    cout << minLen << endl;
    for (int i = 1; i <= N; i++)
        cout << ans[i] << " ";

    return 0;
}

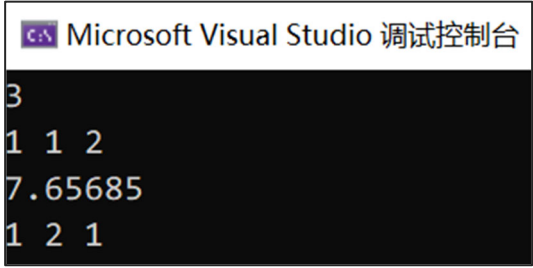
```

时间复杂度分析：

原先算法的时间复杂度为 $O((n+1)!)$ 。镜像排列优化可以剪枝一半的排列情况；在假设有 j 组圆半径相等，其中每组有 c_i 个圆半径相同。在此情况下，时间复杂度可以被优化为：

$$O\left(\frac{(n+1)!}{2 \prod_{i=1}^j c_i!}\right) = O\left(\frac{(n+1)!}{\prod_{i=1}^j c_i!}\right)$$

结果运行：



```

Microsoft Visual Studio 调试控制台
3
1 1 2
7.65685
1 2 1

```