**未经优化的圆排列问题：**

1. **代码**

#include <iostream>

#include <algorithm>

#include <math.h>

#include <time.h>

using namespace std;

int n;

double minlen, x[100], r[100], b[100];

double center (int j) {

double x\_j = 0;

for (int i = 1; i < j; ++i) {

x\_j = max(x\_j, x[i] + 2 \* sqrt(r[i] \* r[j]));

}

return x\_j;

}//计算某个圆的圆心位置

void compute () {

double left = 0, right = 0;

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

if (x[i] - r[i] < left) {

left = x[i] - r[i];

}

if (x[i] + r[i] > right) {

right = x[i] + r[i];

}

}

if (right - left < minlen) {

minlen = right - left;

for (int i = 1; i <= n; ++i) {

b[i] = r[i];

}

}

}//在排列完后计算整个排列的长度

void dfs (int j) {

if (j > n) {

compute();

}

else {

for (int i = j; i <= n; ++i) {

swap (r[j], r[i]);

double x\_tmp = center(j);

if (x\_tmp + r[1] + r[j] < minlen) {

x[j] = x\_tmp;

dfs (j + 1);

}

swap (r[j], r[i]);

}

}

}//使用回溯法深度优先遍历解空间

int main () {

clock\_t start,end;

minlen = 1e20;

cin >> n;

for (int i = 1; i <= n; ++i) {

cin >> r[i];

}

start = clock();

dfs (1);

cout << minlen << endl;

for (int i = 1; i <= n; ++i) {

cout << b[i] << " ";

}

cout << endl;

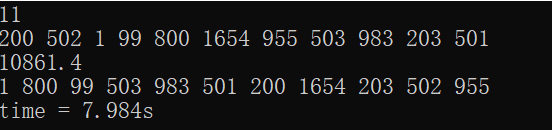
end = clock();

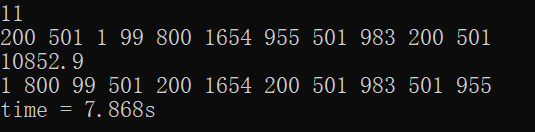
cout << "time = " << double(end-start)/CLOCKS\_PER\_SEC << "s" << endl;

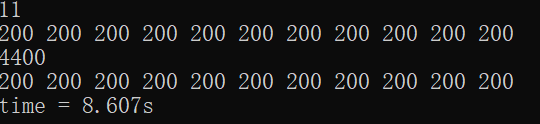
return 0;

}

1. **结果**







1. **复杂度分析**

最坏的情况下需要计算n!数量级排列长度，而计算排列长度的时间复杂度为O(n)，所以时间复杂度为O((n+1)!)

**优化相同半径的的圆的全排列只计算一次**

1. **修改代码**

将回溯的递归函数修改为如下：

void dfs (int j) {

if (j > n) {

compute();

}

else {

int rec[20];

int p = 0;

for (int i = j; i <= n; ++i) {

bool flag = 1;

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

if (rec[k] == r[i]) {

flag = 0;

}

}//记录已经在这个地方安置过的半径，如果相同，则不交换

if (flag == 1) {

swap (r[j], r[i]);

double x\_tmp = center(j);

if (x\_tmp + r[1] + r[j] < minlen) {

x[j] = x\_tmp;

dfs (j + 1);

}

swap (r[j], r[i]);

rec[p] = r[i];

++p;

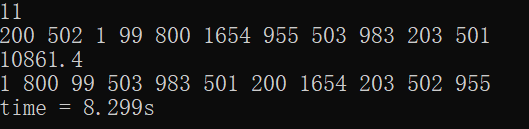
}

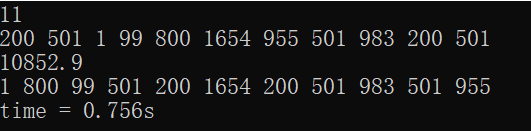
}

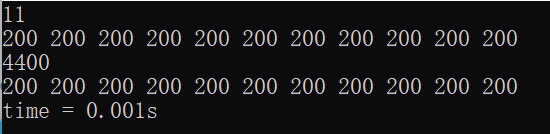
}

}

1. **结果**







可以看出当没有相同半径时，消耗时间与未优化时几乎相同，而相同的半径越多，其相对于未优化时的速度就越快

1. **复杂度分析**

和圆的半径的重复数量有关，是未优化的时间复杂度/（k1! \* k2! \* … \* kn!），kn为重复元素的重复次数。

**优化镜像排列**

1. **代码修改**

我未能找到排除全部镜像序列的方法，经分析，我尝试排除一部分镜像序列以达到减少运算量的目的，即将某个圆开头的序列全部排除，因为其序列的镜像全部包含在了其它圆开头的序列中了，在优化1的基础上，修改代码如下：

void dfs (int j) {

if (j > n) {

compute();

}

else {

int rec[20];

int p = 0;

for (int i = j; i <= n; ++i) {

if (j != 1 || i != n) {//排除最后一个圆开头的序列

bool flag = 1;

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

if (rec[k] == r[i]) {

flag = 0;

}

}

if (flag == 1) {

swap (r[j], r[i]);

double x\_tmp = center(j);

if (x\_tmp + r[1] + r[j] < minlen) {

x[j] = x\_tmp;

dfs (j + 1);

}

swap (r[j], r[i]);

rec[p] = r[i];

++p;

}

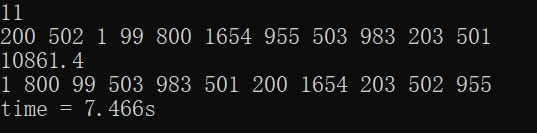
}

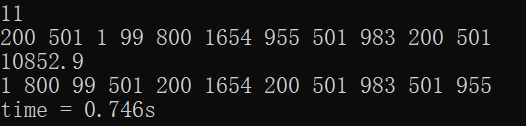
}

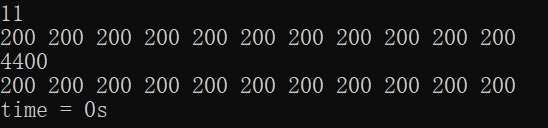
}

}

1. **结果**







1. **复杂度分析**

在优化1的基础上，优化2只做了非常有限的优化，可以看出速度虽然更快，但是快的并不明显，这种算法只是省去了最后一个圆开头的排列，是优化2的时间的复杂度的(n-1)/n