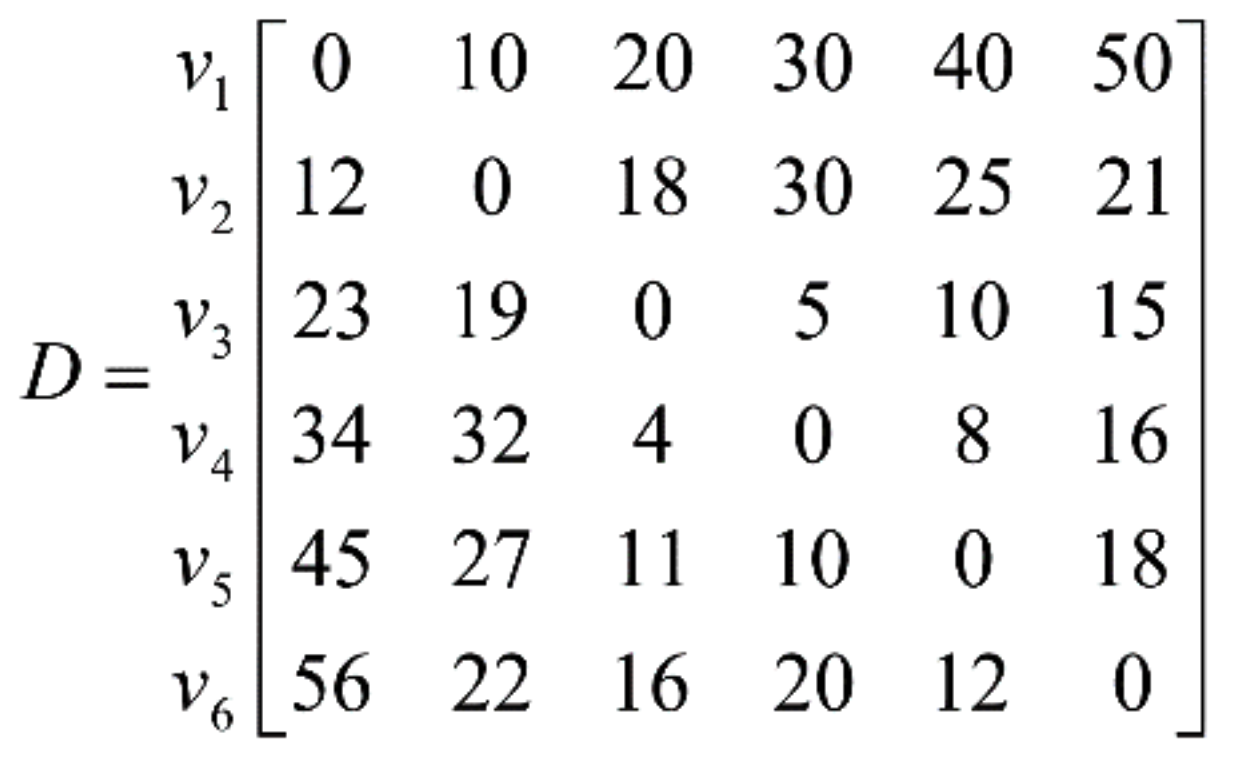
学号：**ZY2206117** 姓名：**黄海浪** 作业：**Assignment\_5**

1. **问题重述**

用模拟退火算法求解TSP问题。



◆ 状态表达:用顺序编码(V1 固定处于编码第一位);

◆ 邻域动作:交换任意两个城市(V1 不参与交换);

◆ 初 始 解:可任意给定初始解;

◆ 初始温度:以本人学号的最后 4 位作为初始温度（6117）;

◆ 降温控制:采用等比法，每次降低 40%的温度(假如初温 1000 度，则下一轮 600 度); t=0.6\*t

◆ 恒温过程:内循环 4 次。每次循环中，首先计算出转移概率，然后

自行模拟“轮盘赌”以决定是否从状态𝑖转移至状态𝑗;

◆ 结束条件:算法执行 4 轮。

1. **算法设计**

1. 定义初始温度，城市数量，城市间距离矩阵，初始解和领域动作产生的解。

2. 定义计算路径长度函数，用于计算给定路径的长度。

3. 定义领域动作函数，用于交换任意两个城市来产生新解。

4. 执行多次迭代。在每次迭代中：

1. 执行多次动作。在每次动作中：

1. 调用领域动作函数来产生新解。

2. 计算当前解和产生解的路径长度。

3. 如果产生解优于当前解，则直接接受；否则，以一定概率接受。

2. 降低温度。

5. 输出最短路径长度和最短路径。

算法使用模拟退火算法来解决旅行商问题，它通过不断迭代和降温来寻找最优解。

1. **算法实现和结果**

**伪代码**

定义初始温度，城市数量，城市间距离矩阵，初始解和领域动作产生的解

定义计算路径长度函数

定义路径长度为0

for i = 0 to 城市数量-2

路径长度 += 距离矩阵[路径[i]][路径[i+1]]

路径长度 += 距离矩阵[路径[城市数量-1]][路径[0]]

返回路径长度

定义领域动作函数，用于交换任意两个城市

随机生成两个不同的整数i和j

复制路径到领域动作产生的解

交换领域动作产生的解中的第i个和第j个元素

for i = 0 to 3

for j = 0 to 3

调用领域动作函数来产生新解

计算当前解和产生解的路径长度

if 产生解优于当前解

直接接受产生解

else

随机生成一个实数sigma

计算概率p = exp((当前解长度 - 产生解长度) \* 100 / 温度)

if p > sigma

接受产生解

降低温度 \*= 0.6

输出最短路径长度和最短路径

**预处理与特殊处理**

预处理包括以下步骤：

1. 定义初始温度为6117（学号ZY2206117）。

2. 定义城市数量为6。

3. 定义一个6x6的城市间距离矩阵。

4. 定义一个初始解，表示旅行商的初始路径。

5. 定义一个领域动作产生的解，用于存储领域动作产生的新解。

这些步骤在代码中通过定义常量和变量来实现，它们为算法的执行提供了必要的输入数据和初始状态。

特殊处理包括：

exp((当前解长度 - 产生解长度) \* 100 / 温度)用于计算在当前温度下，产生解被接受的概率。这个概率与当前解和产生解的路径长度差，以及当前温度有关。在这段代码中，由于初始温度较高，因此将路径长度差乘以了100来放大概率。

sigma = dis(gen) \* (1 - 0.10 \* j)用于生成一个随机数sigma，作为接受新解的阈值。如果接受概率大于sigma，则接受新解；否则拒绝新解。sigma随着动作次数的增加而减小，越到后面，新解越容易被接受。

**代码运行环境和结果**

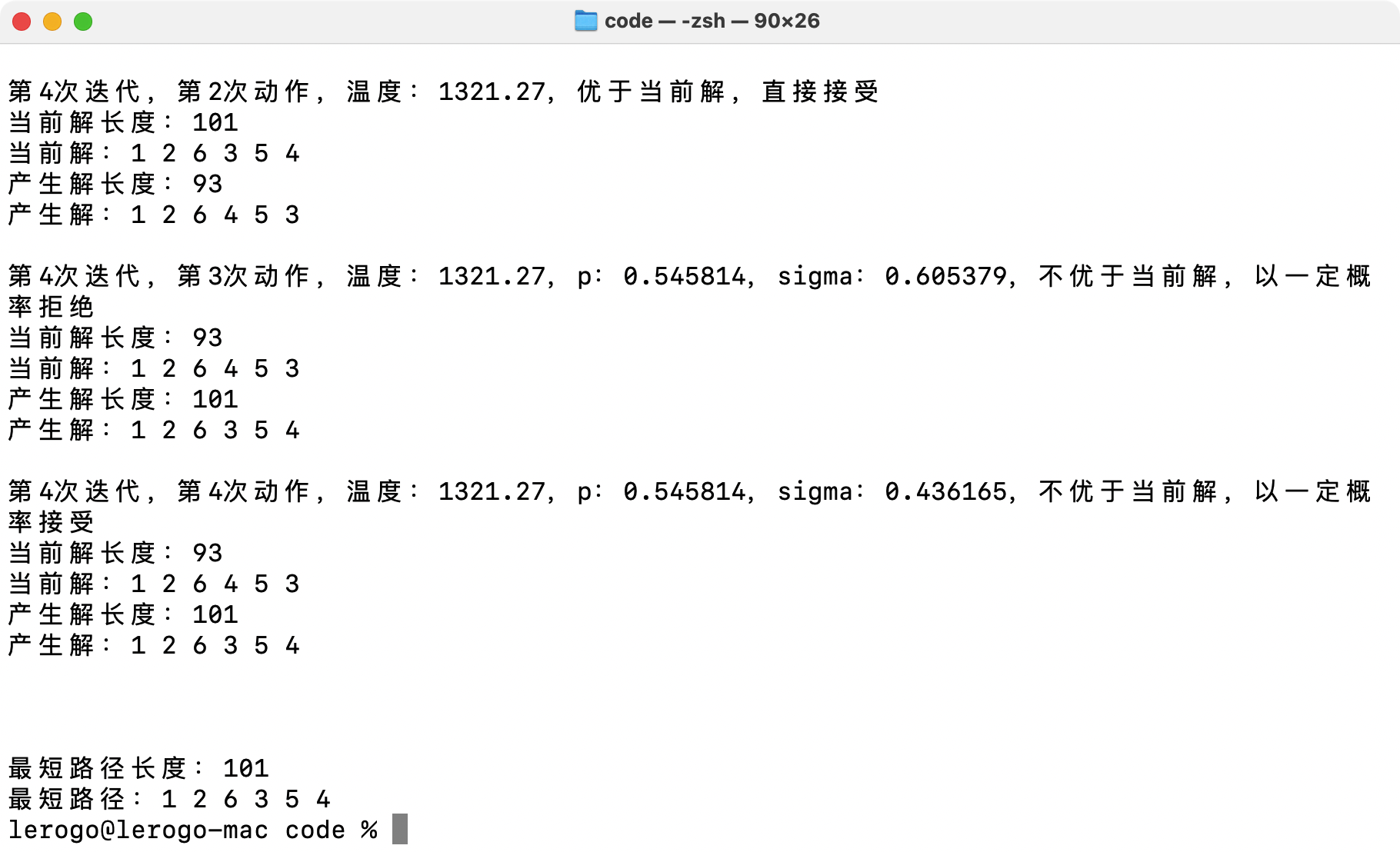
运行环境：

* 操作系统：macOS Ventura13.4
* 构建工具：cmake version 3.25.2；GNU Make 3.81
* 构建命令：cmake CMakeLists.txt && make
* 运行命令：./algorithm\_assignment\_5

运行结果：

**请见附录**

结果截图（随机算法，结果与附录不同）：



1. **附录-结果**

第1次迭代，第1次动作，温度：6117，优于当前解，直接接受

当前解长度：141

当前解：1 4 6 2 3 5

产生解长度：127

产生解：1 4 6 2 5 3

第1次迭代，第2次动作，温度：6117，优于当前解，直接接受

当前解长度：127

当前解：1 4 6 2 5 3

产生解长度：125

产生解：1 4 5 2 6 3

第1次迭代，第3次动作，温度：6117，p：0.891870，sigma：0.473713，不优于当前解，以一定概率接受

当前解长度：125

当前解：1 4 5 2 6 3

产生解长度：132

产生解：1 3 5 2 6 4

第1次迭代，第4次动作，温度：6117，p：0.808542，sigma：0.352791，不优于当前解，以一定概率接受

当前解长度：132

当前解：1 3 5 2 6 4

产生解长度：145

产生解：1 5 3 2 6 4

第2次迭代，第1次动作，温度：3670.2，p：0.479192，sigma：0.889830，不优于当前解，以一定概率拒绝

当前解长度：145

当前解：1 5 3 2 6 4

产生解长度：172

产生解：1 5 3 2 4 6

第2次迭代，第2次动作，温度：3670.2，p：0.479192，sigma：0.782212，不优于当前解，以一定概率拒绝

当前解长度：145

当前解：1 5 3 2 6 4

产生解长度：172

产生解：1 5 3 2 4 6

第2次迭代，第3次动作，温度：3670.2，p：0.826360，sigma：0.677395，不优于当前解，以一定概率接受

当前解长度：145

当前解：1 5 3 2 6 4

产生解长度：152

产生解：1 5 3 6 2 4

第2次迭代，第4次动作，温度：3670.2，优于当前解，直接接受

当前解长度：152

当前解：1 5 3 6 2 4

产生解长度：145

产生解：1 5 3 2 6 4

第3次迭代，第1次动作，温度：2202.12，p：0.664516，sigma：0.935455，不优于当前解，以一定概率拒绝

当前解长度：145

当前解：1 5 3 2 6 4

产生解长度：154

产生解：1 5 2 3 6 4

第3次迭代，第2次动作，温度：2202.12，p：0.293437，sigma：0.700589，不优于当前解，以一定概率拒绝

当前解长度：145

当前解：1 5 3 2 6 4

产生解长度：172

产生解：1 5 3 2 4 6

第3次迭代，第3次动作，温度：2202.12，p：0.664516，sigma：0.514863，不优于当前解，以一定概率接受

当前解长度：145

当前解：1 5 3 2 6 4

产生解长度：154

产生解：1 6 3 2 5 4

第3次迭代，第4次动作，温度：2202.12，优于当前解，直接接受

当前解长度：154

当前解：1 6 3 2 5 4

产生解长度：144

产生解：1 6 2 3 5 4

第4次迭代，第1次动作，温度：1321.27，p：1.000000，sigma：0.941918，不优于当前解，以一定概率接受

当前解长度：144

当前解：1 6 2 3 5 4

产生解长度：144

产生解：1 6 2 4 5 3

第4次迭代，第2次动作，温度：1321.27，p：0.588727，sigma：0.710609，不优于当前解，以一定概率拒绝

当前解长度：144

当前解：1 6 2 4 5 3

产生解长度：151

产生解：1 3 2 4 5 6

第4次迭代，第3次动作，温度：1321.27，优于当前解，直接接受

当前解长度：144

当前解：1 6 2 4 5 3

产生解长度：93

产生解：1 2 6 4 5 3

第4次迭代，第4次动作，温度：1321.27，p：0.545814，sigma：0.610212，不优于当前解，以一定概率拒绝

当前解长度：93

当前解：1 2 6 4 5 3

产生解长度：101

产生解：1 2 6 3 5 4

最短路径长度：93

最短路径：1 2 6 4 5 3

1. **附录-代码**

#include <iostream>

#include <random>

#include <cstring>

// ZY2206117 黄海浪

const double initTemperature = 6117;

// 是否打印过程日志

bool isLog = true;

// 6 cities

const int n = 6;

// 6x6 cost matrix

const int D[n][n] = {

{0, 10, 20, 30, 40, 50},

{12, 0, 18, 30, 25, 21},

{23, 19, 0, 5, 10, 15},

{34, 32, 4, 0, 8, 16},

{45, 27, 11, 10, 0, 18},

{56, 22, 16, 20, 12, 0}

};

// 解的编码（顺序编码） 下面为初始解

int ans[n] = {0, 3, 5, 1, 2, 4};

// 领域动作产生的解

int gAns[n];

// 计算路径长度

int getLength(const int \*path) {

int length = 0;

for (int i = 0; i < n - 1; ++i) {

length += D[path[i]][path[i + 1]];

}

length += D[path[n - 1]][path[0]];

return length;

}

// 领域动作，换任意两个城市(V1 不参与交换)

void getNeighbor(const int \*path) {

std::random\_device rd;

std::mt19937 gen(rd());

std::uniform\_int\_distribution<> dis(1, n - 1);

int i = dis(gen);

int j = dis(gen);

while (i == j) {

j = dis(gen);

}

memcpy(gAns, path, sizeof(int) \* n);

std::swap(gAns[i], gAns[j]);

}

void logData(int i, int j, double t, const std::string &info) {

std::cout << "第" << i << "次迭代，第" << j << "次动作，温度：" << t << "，" << info << std::endl;

std::cout << "当前解长度：" << getLength(ans) << std::endl;

std::cout << "当前解：";

for (int an: ans) {

std::cout << an + 1 << " ";

}

std::cout << std::endl;

std::cout << "产生解长度：" << getLength(gAns) << std::endl;

std::cout << "产生解：";

for (int an: gAns) {

std::cout << an + 1 << " ";

}

std::cout << std::endl << std::endl;

}

int main() {

double t = initTemperature;

for (int i = 0; i < 4; ++i) {

for (int j = 0; j < 4; ++j) {

getNeighbor(ans);

int ansLength = getLength(ans);

int gAnsLength = getLength(gAns);

if (ansLength > gAnsLength) {

if (isLog) {

logData(i + 1, j + 1, t, "优于当前解，直接接受");

}

// 优于当前解，直接接受

memcpy(ans, gAns, sizeof(int) \* n);

} else {

// 以一定概率接受

std::random\_device rd;

std::mt19937 gen(rd());

std::uniform\_real\_distribution<> dis(0.5, 1);

// 由于用学号后4位作为温度，直接用目标函数值作为概率不行，因此放大100倍

double p = exp((ansLength - gAnsLength) \* 100 / t);

// 越到后面，越容易被接受

double sigma = dis(gen) \* (1 - 0.10 \* j);

std::string logParam = "p：" + std::to\_string(p) + "，sigma：" + std::to\_string(sigma);

if (p > sigma) {

if (isLog) {

logData(i + 1, j + 1, t, logParam + "，不优于当前解，以一定概率接受");

}

memcpy(ans, gAns, sizeof(int) \* n);

} else {

if (isLog) {

logData(i + 1, j + 1, t, logParam + "，不优于当前解，以一定概率拒绝");

}

}

}

}

// 降温

t \*= 0.6;

}

std::cout << std::endl << std::endl;

std::cout << "最短路径长度：" << getLength(ans) << std::endl;

std::cout << "最短路径：";

for (int an: ans) {

std::cout << an + 1 << " ";

}

std::cout << std::endl;

return 0;

}