

**《操作系统》**

**实验课题十九**

|  |  |
| --- | --- |
|  |  |
| 学 号： |  |
| 姓 名： |  |
| 学 院： |  |
| 日 期： | 2022年 11 月 20 日 |

# 一、开发环境、运行环境、测试环境

使用vs 2022进行源代码的编写，使用vs 2022运行程序进行测试。

# 二、实验内容

（1）编程设计实现先来先服务调度算法、最短寻道时间优先调度算法、电梯调度算法、

循环式单向电梯调度算法、双队列单向电梯调度算法；

（2）编程设计实现移动头磁盘访问事件序列的随机发生机制，其中所生成的磁道号取值区间为[0, 199]；

（3）基于相同的一组移动头磁盘访问事件序列（长度为 100，即 100 个磁道号），进行有关算法的测试，并显示输出基于相应调度算法的磁道访问的先后次序及平均寻道数；

（4）变换上述移动头磁盘访问事件序列实施多次测试，统计分析和比较有关算法的性能（譬如平均寻道数）。

# 三、实验步骤

## 一、关键数据结构

求平均寻道长度函数：

double average (int\* length, int n)

先来先服务调度算法（FCFS）：

void First\_come\_first\_server(int\* a, int n, int now)

最短寻道时间优先调度算法（SSTF）：

void Shortest\_seek\_time\_first(int\* a, int n, int now)

电梯调度算法（SCAN）：

void Scan(int\* a, int n, int now)

循环式单向电梯调度算法（CSCAN）：

void Circle\_scan(int\* a, int n, int now)

双队列电梯调度算法（FSCAN）：

void F\_scan(int\* a, int n, int now)

## 二、算法流程

#### 1. 先来先服务调度算法（FCFS）

按访问请求到达的先后次序进行调度。

#### 2. 最短寻道时间优先调度算法（SSTF）

优先选择使磁头臂从当前位置开始移动最少的磁盘 I/O 请求进行调度。

#### 3. 电梯调度算法（SCAN）

要求磁头臂先沿一个方向移动，并在途中满足所有未完成的请求，直到它到达这个方向上的最后一个磁道，或者在这个方向上没有别的请求为止，后一种改进有时候称作LOOK 策略。然后倒转服务方向，沿相反方向扫描，同样按顺序完成所有请求。

#### 4. 循环式单向电梯调度算法（CSCAN）

在磁盘调度时，把扫描限定在一个方向，当沿某个方向访问到最后一个磁道时，磁头臂返回到磁盘的另一端，并再次开始扫描。

#### 5. 双队列电梯调度算法（FSCAN）

将磁盘请求队列分成两个子队列，一个是当前请求的队列，一个是等待处理的队列，分别对其运用SCAN算法进行处理。

#### 6. 移动头磁盘访问事件序列随机发生机制

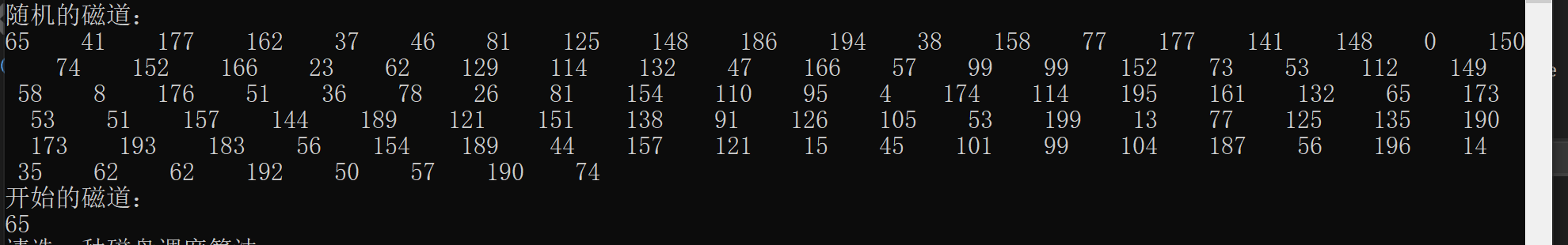
产生范围为[0,199]内的100个随机数，作为随机的访问序列。

## 三、测试结果及分析

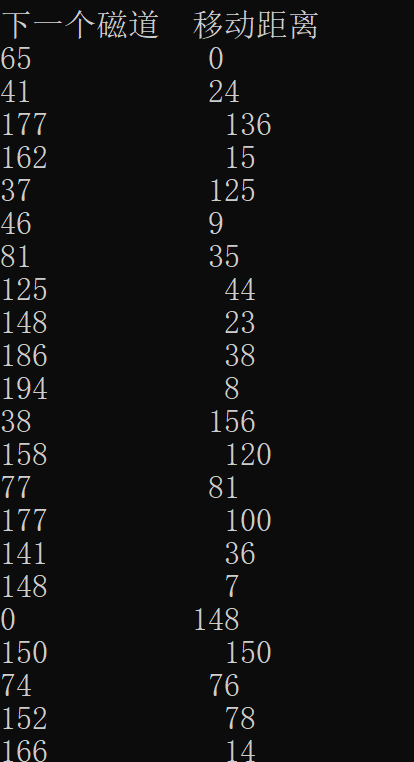
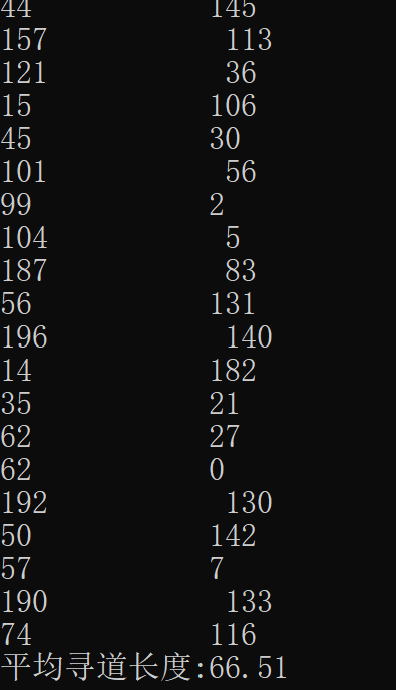
通过不断产生不同的随机访问序列，以此得到各种移动头磁盘调度算法的平均寻道数。

五个算法的部分测试结果如下图所示：

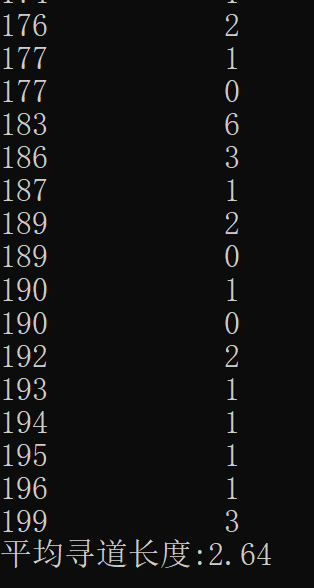
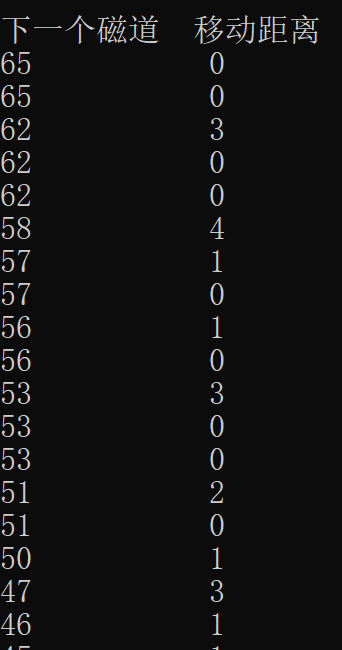
生成的随机访问序列



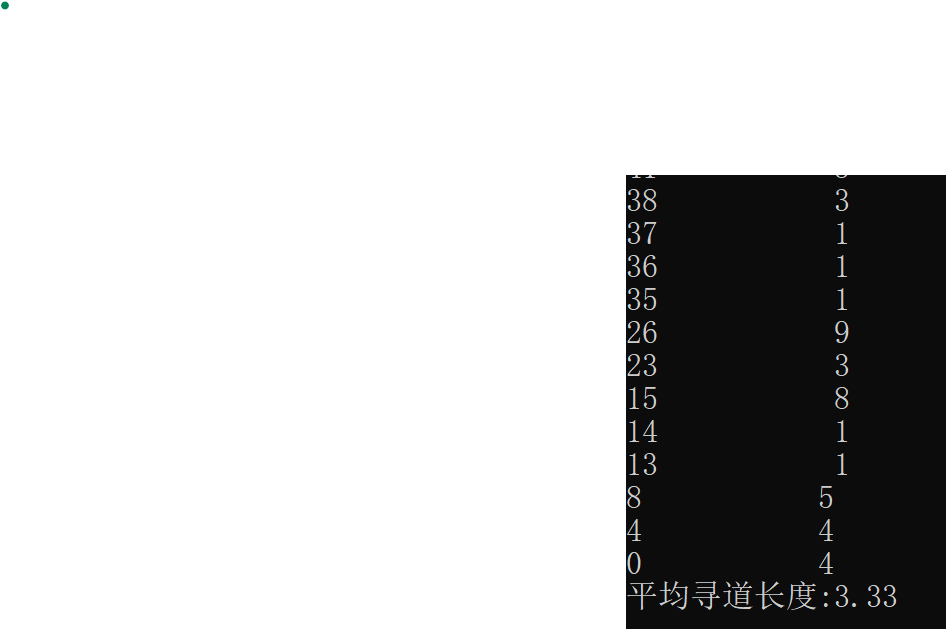
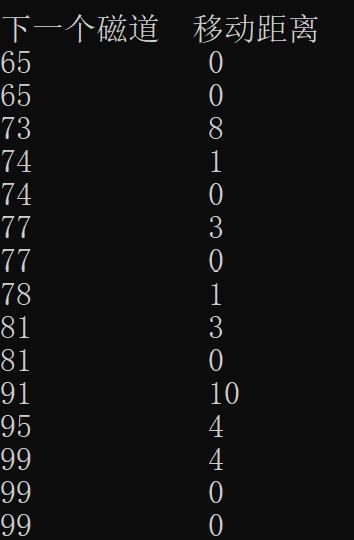
先来先服务调度算法（FCFS）：

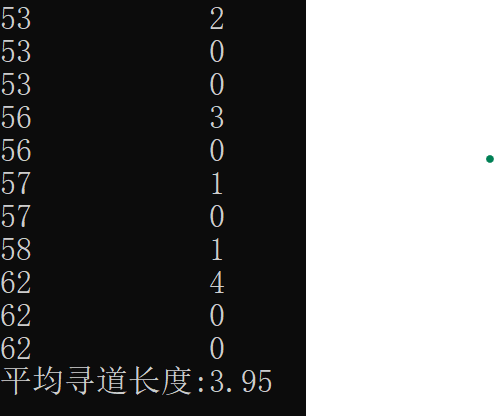
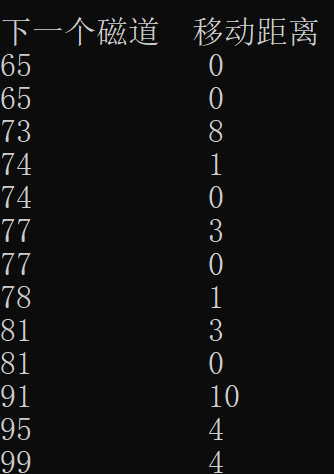
最短寻道时间优先调度算法（SSTF）：



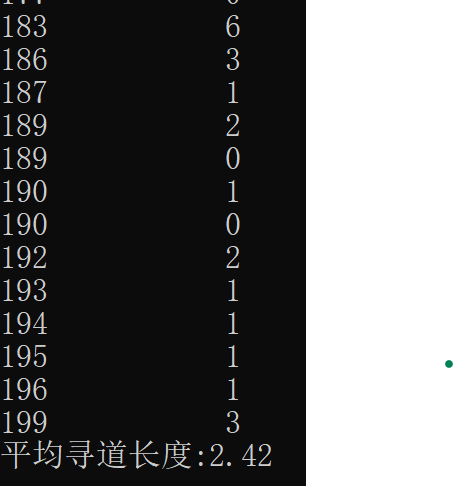
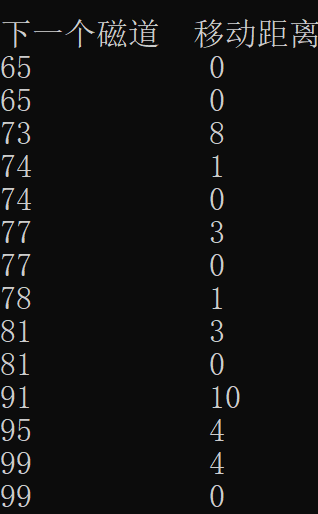
电梯调度算法（SCAN）：



循环式单向电梯调度算法（CSCAN）：



双队列电梯调度算法（FSCAN）：



通过不断设定不同的随机访问序列进行测试得到的结果如下：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 实验序号 | FCFS | SSTF | SCAN | CSCAN | FSCAN |
| 1 | 66.51 | 2.64 | 3.33 | 3.95 | 2.42 |
| 2 | 64.78 | 2.98 | 2.94 | 3.86 | 2.86 |
| 3 | 66.33 | 2.84 | 3.10 | 3.95 | 2.16 |
| 4 | 57.74 | 2.82 | 3.14 | 3.95 | 2.15 |
| 5 | 57.27 | 2.84 | 3.10 | 3.95 | 2.03 |

可见，显然FCFS的效果最差，平均寻道数最大，效果最好的是FSCAN，其作为SCAN的改进版，效果最好。

# 四、实验总结

这次实验主要研究移动头磁盘调度算法，通过课上知识与课下学习，我搞清楚了各种磁盘头调度算法的流程，在实验过程中，出现了很多bug，但通过自己DUBUG和参考网络上经典的代码最终成功完成了实验。并且在此次实验中我学会了很多新的知识和技能，收获颇多。

# 五、源代码附录

1. #include <iostream>
2. #include <algorithm>
3. #include <cstdio>
5. #define MAX 200
6. **using** **namespace** std;
8. //求平均寻道长度
9. **double** average(**int**\* length, **int** n) {
10. **double** sum = 0;
11. **int** i;
12. **for** (i = 0; i < n; i++)
13. sum += length[i];
14. **return** sum / n;
15. }
17. //先来先服务调度算法
18. **void** First\_come\_first\_server(**int**\* a, **int** n, **int** now) {
19. **int** i, k = 0;
20. **int** t = n;
21. **int** next[MAX], length[MAX];
22. **int** visit[MAX] = { 0 };
23. **while** (t--)
24. {
25. **for** (i = 0; i < n; i++) {
26. **if** (visit[i])
27. **continue**;
28. **else** {
29. length[k] = abs(a[i] - now);
30. next[k++] = a[i];
31. visit[i] = 1;
32. now = a[i];
33. }
34. }
35. }
36. cout << endl << "下一个磁道  " << "移动距离" << endl;
37. **for** (i = 0; i < n; i++)
38. cout << next[i] << "           " << length[i] << endl;
39. cout << "平均寻道长度:";
40. printf("%.2f\n\n", average(length, n));
41. }
42. //最短寻道时间优先算法
43. **void** Shortest\_seek\_time\_first(**int**\* a, **int** n, **int** now) {
44. **int** i, min, k = 0;
45. **int** flag;//最短路径的下标
46. **int** t = n;
47. **int** next[MAX], length[MAX];
48. **int** visit[MAX] = { 0 };//访问数组，已经访问过的为1
49. //寻道n次
50. **while** (t--) {
51. min = 999999;
52. //找没有访问过的中最近的
53. **for** (i = 0; i < n; i++) {
54. **if** (visit[i])
55. **continue**;
56. **if** (abs(a[i] - now) < min) {
57. flag = i;
58. min = abs(a[i] - now);
59. }
60. }
61. length[k] = min;//移动的距离
62. next[k++] = a[flag];//下一个被访问的磁道号
63. visit[flag] = 1;//访问过的置1
64. now = a[flag];//磁道移动到当前位置
65. }
66. cout << endl << "下一个磁道  " << "移动距离" << endl;
67. **for** (i = 0; i < n; i++)
68. cout << next[i] << "           " << length[i] << endl;
69. cout << "平均寻道长度:";
70. printf("%.2f\n\n", average(length, n));
71. }
73. //电梯调度算法
74. **void** Scan(**int**\* a, **int** n, **int** now) {
75. **int** i, flag = 0, k = 0;
76. **int** next[MAX], length[MAX];
77. sort(a, a + n);//从小到大排序
78. //寻找开始磁道
79. **for** (i = 0; i < n; i++) {
80. **if** (a[i] >= now) {
81. flag = i;
82. **break**;
83. }
84. }
86. //往递增方向访问
87. **for** (i = flag; i < n; i++) {
88. next[k] = a[i];
89. length[k++] = abs(a[i] - now);
90. now = a[i];
91. }
93. //回来往递减方向访问
94. **for** (i = flag - 1; i >= 0; i--) {
95. next[k] = a[i];
96. length[k++] = abs(a[i] - now);
97. now = a[i];
98. }
100. cout << endl << "下一个磁道  " << "移动距离" << endl;
101. **for** (i = 0; i < n; i++)
102. cout << next[i] << "           " << length[i] << endl;
103. cout << "平均寻道长度:";
104. printf("%.2f\n\n", average(length, n));
105. }
107. **void** Circle\_scan(**int**\* a, **int** n, **int** now) {
108. **int** i, flag = 0, k = 0;
109. **int** next[MAX], length[MAX];
110. sort(a, a + n);//从小到大排序
111. **for** (i = 0; i < n; i++) {
112. **if** (a[i] >= now) {
113. flag = i;
114. **break**;
115. }
116. }
118. //往递增方向访问
119. **for** (i = flag; i < n; i++) {
120. next[k] = a[i];
121. length[k++] = abs(a[i] - now);
122. now = a[i];
123. }
125. //循环往递增方向访问
126. **for** (i = 0; i < flag; i++) {
127. next[k] = a[i];
128. length[k++] = abs(a[i] - now);
129. now = a[i];
130. }
132. cout << endl << "下一个磁道  " << "移动距离" << endl;
133. **for** (i = 0; i < n; i++)
134. cout << next[i] << "           " << length[i] << endl;
135. cout << "平均寻道长度:";
136. printf("%.2f\n\n", average(length, n));
137. }
139. **void** F\_scan(**int**\* a, **int** n, **int** now) {
140. **int** i, flag = 0, k = 0;
141. **int** next1[MAX], length1[MAX];
142. **int** next2[MAX], length2[MAX];
143. **int** a1[MAX], a2[MAX];
144. **for** (**int** j = 0; j < n / 2; j++) {
145. a1[j] = a[j];
146. }
147. **for** (**int** j = n / 2,i=0; j < n; j++) {
148. a2[i++] = a[j];
149. }
150. **if** (n % 2 == 0) {
151. sort(a1, a1 + n / 2);//从小到大排序
152. sort(a2, a2 + n / 2);
153. //寻找开始磁道
154. **for** (i = 0; i < n / 2; i++) {
155. **if** (a1[i] >= now) {
156. flag = i;
157. **break**;
158. }
159. }
161. //往递增方向访问
162. **for** (i = flag; i < n / 2; i++) {
163. next1[k] = a1[i];
164. length1[k++] = abs(a1[i] - now);
165. now = a1[i];
166. }
168. //回来往递减方向访问
169. **for** (i = flag - 1; i >= 0; i--) {
170. next1[k] = a1[i];
171. length1[k++] = abs(a1[i] - now);
172. now = a1[i];
173. }
174. flag = 0; k = 0;
175. now = a[n / 2];
176. //寻找开始磁道
177. **for** (i = 0; i < n / 2; i++) {
178. **if** (a2[i] >= now) {
179. flag = i;
180. **break**;
181. }
182. }
184. //往递增方向访问
185. **for** (i = flag; i < n / 2; i++) {
186. next2[k] = a2[i];
187. length2[k++] = abs(a2[i] - now);
188. now = a2[i];
189. }
191. //回来往递减方向访问
192. **for** (i = flag - 1; i >= 0; i--) {
193. next2[k] = a2[i];
194. length2[k++] = abs(a2[i] - now);
195. now = a2[i];
196. }
197. }
198. **else** {
199. sort(a1, a1 + n / 2);//从小到大排序
200. sort(a2, a2 + n / 2 + 1);
201. //寻找开始磁道
202. **for** (i = 0; i < n / 2; i++) {
203. **if** (a1[i] >= now) {
204. flag = i;
205. **break**;
206. }
207. }
209. //往递增方向访问
210. **for** (i = flag; i < n / 2; i++) {
211. next1[k] = a1[i];
212. length1[k++] = abs(a1[i] - now);
213. now = a1[i];
214. }
216. //回来往递减方向访问
217. **for** (i = flag - 1; i >= 0; i--) {
218. next1[k] = a1[i];
219. length1[k++] = abs(a1[i] - now);
220. now = a1[i];
221. }
222. flag = 0; k = 0;
223. now = a[n / 2];
224. //寻找开始磁道
225. **for** (i = 0; i < n / 2 + 1; i++) {
226. **if** (a2[i] >= now) {
227. flag = i;
228. **break**;
229. }
230. }
232. //往递增方向访问
233. **for** (i = flag; i < n / 2 + 1; i++) {
234. next2[k] = a2[i];
235. length2[k++] = abs(a2[i] - now);
236. now = a2[i];
237. }
239. //回来往递减方向访问
240. **for** (i = flag - 1; i >= 0; i--) {
241. next2[k] = a2[i];
242. length2[k++] = abs(a2[i] - now);
243. now = a2[i];
244. }
245. }
246. cout << endl << "下一个磁道  " << "移动距离" << endl;
247. **for** (i = 0; i < n / 2; i++)
248. cout << next1[i] << "           " << length1[i] << endl;
249. **for** (i = 0; i < n / 2; i++)
250. cout << next2[i] << "           " << length2[i] << endl;
251. cout << "平均寻道长度:";
252. printf("%.2f\n\n", (average(length1, n/2)+ average(length2, n / 2))/2);
253. }
255. **int** main() {
256. **int** a[MAX];//存放要用到的磁道号
257. **int** start;//开始的磁道
258. **int** n, i, flag = 1;
259. cout << "一共要用到的磁道数：" << endl;
260. cin >> n;
261. cout << "随机的磁道：" << endl;
262. srand((**int**)time(0));
263. **for** (i = 0; i < n; i++) {
264. a[i] = rand() % 200;
265. cout << a[i] << "    ";
266. }
267. cout << endl << "开始的磁道：" << endl;
268. cin >> start;
270. //磁盘调度目录
271. **while** (flag) {
272. cout << "请选一种磁盘调度算法：" << endl;;
273. cout << "1.先来先服务调度算法" << endl;
274. cout << "2.最短寻道时间优先算法" << endl;
275. cout << "3.电梯调度算法" << endl;
276. cout << "4.循环式单向电梯调度算法" << endl;
277. cout << "5.双队列电梯调度算法" << endl;
278. cout << "0.退出" << endl;
279. cin >> flag;
280. **switch** (flag) {
281. **case** 1:
282. First\_come\_first\_server(a, n, start);
283. **break**;
284. **case** 2:
285. Shortest\_seek\_time\_first(a, n, start); **break**;
286. **case** 3:
287. Scan(a, n, start); **break**;
288. **case** 4:
289. Circle\_scan(a, n, start); **break**;
290. **case** 5:
291. F\_scan(a, n, start); **break**;
292. **case** 0:
293. **break**;
294. }
295. }
296. **return** 0;
297. }