模拟内存分配与回收

1. 实验目的

用高级语言编写和调试一个简单的内存分配与回收程序，模拟内存分配与回收的工作过程。从而对内存分配与回收的实质内容和执行过程有比较深入的了解。

1. 实验内容

任务：设计并实现一个简单的内存分配与回收程序

在动态分区存储管理方式中，主要的操作是**分配内存**和**回收内存**。

使用了四种内存分配算法：首次适应算法、循环首次适应算法、最佳适应算法、最坏适应算法。

代码：

os.h:

1. #ifndef \_OS\_H\_
2. #define \_OS\_H\_
4. #include <list>
5. #include <algorithm>
6. #include <iostream>
7. #include "memNode.h"
8. **using** **namespace** std;
10. **enum** flags{
11. SIZE\_ASC,
12. SIZE\_DESC,
13. ADDR\_ASC,
14. NO\_LOOP,
15. LOOP
16. };
17. /\*
18. const int SIZE\_DESC = 0;
19. const int SIZE\_ASC = 1;
20. const int ADDR\_ASC = 2;
21. const int NO\_LOOP = 3;
22. const int LOOP = 4;
23. \*/
24. **class** os
25. {
26. **public**:
27. os():\_initNum(0),\_minSize(0), \_empty(), \_use() {}
29. **void** init()
30. {
31. cout << "请输入最小不可分割大小:>";
32. cin >> \_minSize;
34. cout << "请输入初始内存块数:>";
35. cin >> \_initNum;
36. }
38. **void** create\_empty(){
39. **for**(**int** i = 0; i < \_initNum; ++i){
40. memNode \*newNode = **new** memNode;
41. **int** begin, end;
42. cout << "请输入第[" << i+1 << "]个空白内存块的首尾地址:>";
43. cin >> begin >> end;
44. newNode->set\_node(begin, end);
45. \_empty.push\_back(newNode);
46. }
47. }
49. //flag 1 降序　0 升序  2 地址升序
50. **void** list\_sort(**int** flag){
51. **switch**(flag){
52. **case** SIZE\_DESC:\_empty.sort(less\_than);**break**;
53. **case** SIZE\_ASC:\_empty.sort(greater);**break**;
54. **case** ADDR\_ASC:\_empty.sort(addr);**break**;
55. **default**:**return**;
56. }
57. }
59. **bool** alloc(**int** size, list<memNode\*>::iterator& start, **int** flag){
60. list<memNode\*>::iterator it = start;
62. **for**(; it != \_empty.end(); ++it){
63. **if**((\*it)->get\_size() - size > \_minSize){
64. memNode \*newNode = **new** memNode;
65. newNode->set\_node((\*it)->get\_begin(), (\*it)->get\_begin() + size);
66. (\*it)->cut\_node(size);
67. \_use.push\_back(newNode);
68. start = ++it;
69. **return** **true**;
70. }
71. **else** **if**((\*it)->get\_size() - size >= 0){
72. \_use.push\_back(\*it);
73. \_empty.erase(it);
74. start = ++it;
75. **return** **true**;
76. }
77. }
78. **if**(it == \_empty.end()){
79. **if**(flag == NO\_LOOP)
80. **return** **false**;
81. **else** {
82. **for**(it = \_empty.begin(); it != start; ++it){
83. **if**((\*it)->get\_size() - size > \_minSize){
84. memNode \*newNode = **new** memNode;
85. newNode->set\_node((\*it)->get\_begin(), (\*it)->get\_begin() + size);
86. (\*it)->cut\_node(size);
87. \_use.push\_back(newNode);
88. start = ++it;
89. **return** **true**;
90. }
91. **else** **if**((\*it)->get\_size() - size >= 0){
92. \_use.push\_back(\*it);
93. \_empty.erase(it);
94. start = ++it;
95. **return** **true**;
96. }
97. }
98. **if**(it == start)
99. **return** **false**;
100. }
101. }
103. **return** **true**;
104. }
105. **void** output\_list(**const** list<memNode\*>& list){
106. **for**(auto e:list){
107. printf("┌──────┬──────┐   ");
108. }
109. printf("┌──────┐");
110. cout << endl;
111. **for**(auto e:list){
112. printf("│%6d│%6d│——>", e->get\_begin(), e->get\_end());
113. }
114. printf("│ null │");
115. cout << endl;
116. **for**(auto e:list){
117. printf("└──────┴──────┘   ");
118. }
119. printf("└──────┘");
120. cout << endl;
121. }
123. **void** output(){
124. cout << endl;
125. cout << "空闲分区链:\n";
126. output\_list(\_empty);
128. cout << "已用分区链:\n";
129. output\_list(\_use);
130. cout << endl;
131. }
133. **void** distribution(**int** flag){
134. **int** size;
135. cout << "请输入要到来进程需要的内存大小:>";
136. cin >> size;
137. **static** auto sit = \_empty.begin();
138. auto it = \_empty.begin();
139. **bool** ret;
140. **if**(flag == LOOP){
141. ret = alloc(size, sit, flag);
142. }
143. **else** {
144. ret = alloc(size, it, flag);
145. }
146. **if**(ret == **false**){
147. cout << "可用内存不足，分配失败!!" << endl;
148. **return**;
149. }
150. }
152. **void** freed(){
153. **int** addr;
154. cout << "请输入要释放内存块的起始地址:";
155. cin >> addr;
156. auto it = \_use.begin();
157. **for**( ;it != \_use.end(); ++it){
158. **if**((\*it)->get\_begin() == addr){
159. \_empty.push\_back(\*it);
160. \_use.erase(it);
161. **return**;
162. }
163. }
164. **if**(it == \_use.end()){
165. output();
166. cout << "请输入正确的起始地址!" << endl;
167. }
168. }
170. **void** merge(list<memNode\*> &l){
171. auto it = l.begin();
172. **for**( ;it != l.end(); ++it){
173. auto tmp = it;
174. ++it;
175. **if**(it == l.end())**break**;
176. **if**((\*tmp)->get\_end() == (\*it)->get\_begin()){
177. (\*tmp)->set\_node((\*tmp)->get\_begin(), (\*it)->get\_end());
178. l.erase(it);
179. }
180. it = tmp;
181. }
182. }
184. **void** FF(){
185. merge(\_empty);
186. list\_sort(ADDR\_ASC);
187. output();
188. **int** c;
189. **while**(1){
190. cout << "1.分配　　2.释放    0.退出\n请选择:>";
191. cin >> c;
192. **switch**(c){
193. **case** 1:distribution(NO\_LOOP);**break**;
194. **case** 2:freed();**break**;
195. **case** 0:**return**;
196. **default**:**continue**;
197. }
198. list\_sort(ADDR\_ASC);
199. merge(\_empty);
200. output();
201. }
202. }
204. **void** NF(){
205. merge(\_empty);
206. list\_sort(ADDR\_ASC);
207. output();
208. **int** c;
209. **while**(1){
210. cout << "1.分配　　2.释放    0.退出\n请选择:>";
211. cin >> c;
212. **switch**(c){
213. **case** 1:distribution(LOOP);**break**;
214. **case** 2:freed();**break**;
215. **case** 0:**return**;
216. **default**:**continue**;
217. }
218. list\_sort(ADDR\_ASC);
219. merge(\_empty);
220. output();
221. }
222. }
224. **void** BF(){
225. list\_sort(ADDR\_ASC);
226. merge(\_empty);
227. list\_sort(SIZE\_ASC);
228. output();
229. **int** c;
230. **while**(1){
231. cout << "1.分配　　2.释放    0.退出\n请选择:>";
232. cin >> c;
233. **switch**(c){
234. **case** 1:distribution(NO\_LOOP);**break**;
235. **case** 2:freed();**break**;
236. **case** 0:**return**;
237. **default**:**continue**;
238. }
239. list\_sort(ADDR\_ASC);
240. merge(\_empty);
241. list\_sort(SIZE\_ASC);
242. output();
243. }
244. }
246. **void** WF(){
247. list\_sort(ADDR\_ASC);
248. merge(\_empty);
249. list\_sort(SIZE\_DESC);
250. output();
251. **int** c;
252. **while**(1){
253. cout << "1.分配　　2.释放    0.退出\n请选择:>";
254. cin >> c;
255. **switch**(c){
256. **case** 1:distribution(NO\_LOOP);**break**;
257. **case** 2:freed();**break**;
258. **case** 0:**return**;
259. **default**:**continue**;
260. }
261. list\_sort(ADDR\_ASC);
262. merge(\_empty);
263. list\_sort(SIZE\_DESC);
264. output();
265. }
266. }
268. ~os() {
269. **for**(auto e:\_empty){
270. **delete** e;
271. }
272. **for**(auto e:\_use){
273. **delete** e;
274. }
275. }
277. **private**:
278. **int** \_initNum;
279. **int** \_minSize;
280. list<memNode\*> \_empty;
281. list<memNode\*> \_use;
282. **struct** less\_than{
283. **bool** operator()(**const** memNode\* n1,**const** memNode\* n2){
284. **if**(n1->get\_size() > n2->get\_size())**return** **true**;
285. **return** **false**;
286. }
287. }less\_than;
289. **struct** greater{
290. **bool** operator()(**const** memNode\* n1,**const** memNode\* n2){
291. **if**(n1->get\_size() > n2->get\_size())**return** **true**;
292. **return** **false**;
293. }
294. }greater;
296. **struct** addr{
297. **bool** operator()(**const** memNode\* n1, **const** memNode\* n2){
298. **if**(n1->get\_begin() < n2->get\_begin())**return** **true**;
299. **return** **false**;
300. }
301. }addr;
302. };
304. #endif

memNode.h

1. #ifndef \_MEMNODE\_H\_
2. #define \_MEMNODE\_H\_
4. **class** memNode
5. {
6. **public**:
7. memNode() {}
9. **int** get\_begin()**const** { **return** \_begin; }
11. **int** get\_end()**const** { **return** \_end; }
13. **int** get\_size()**const** { **return** \_size; }
15. **void** set\_node(**int** begin, **int** end){
16. \_begin = begin;
17. \_end = end;
18. \_size = \_end - \_begin;
19. };
21. **void** cut\_node(**int** n){
22. \_begin += n;
23. \_size = \_end- \_begin;
24. }
26. ~memNode() {}
27. **private**:
28. **int** \_begin;
29. **int** \_end;
30. **int** \_size;
31. };

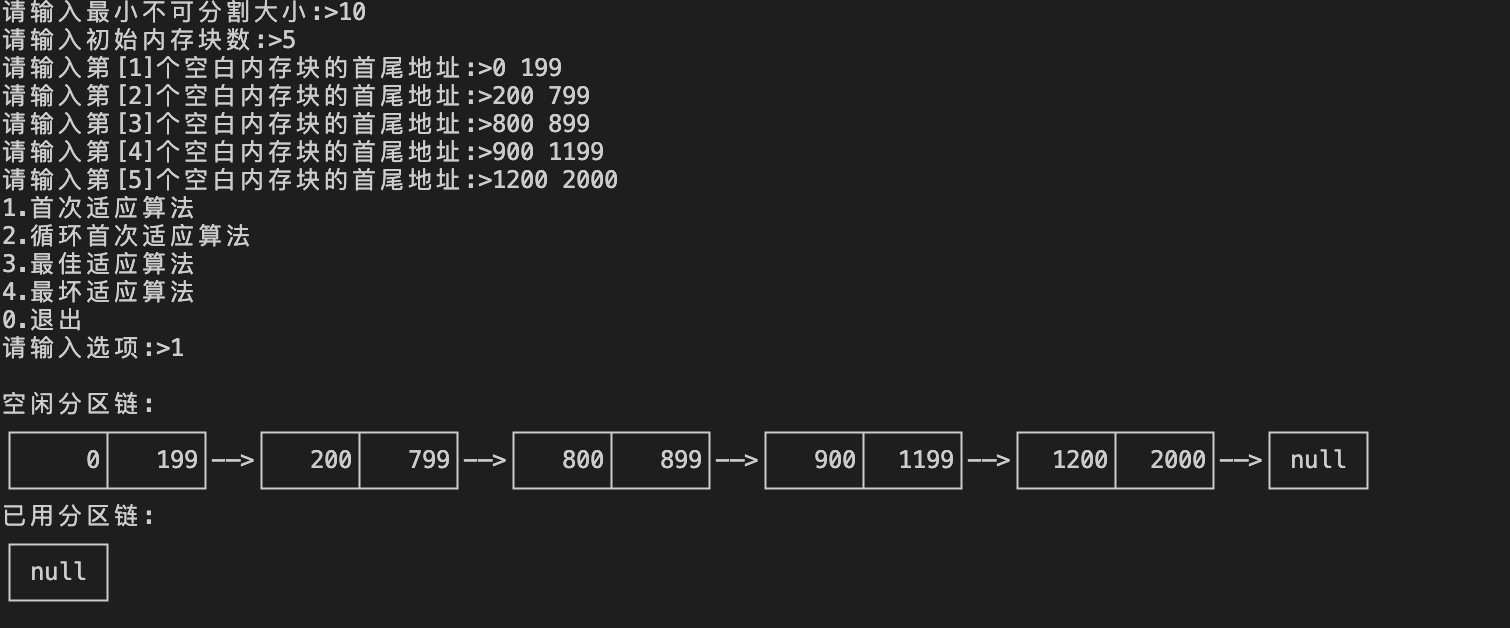
34. #endif

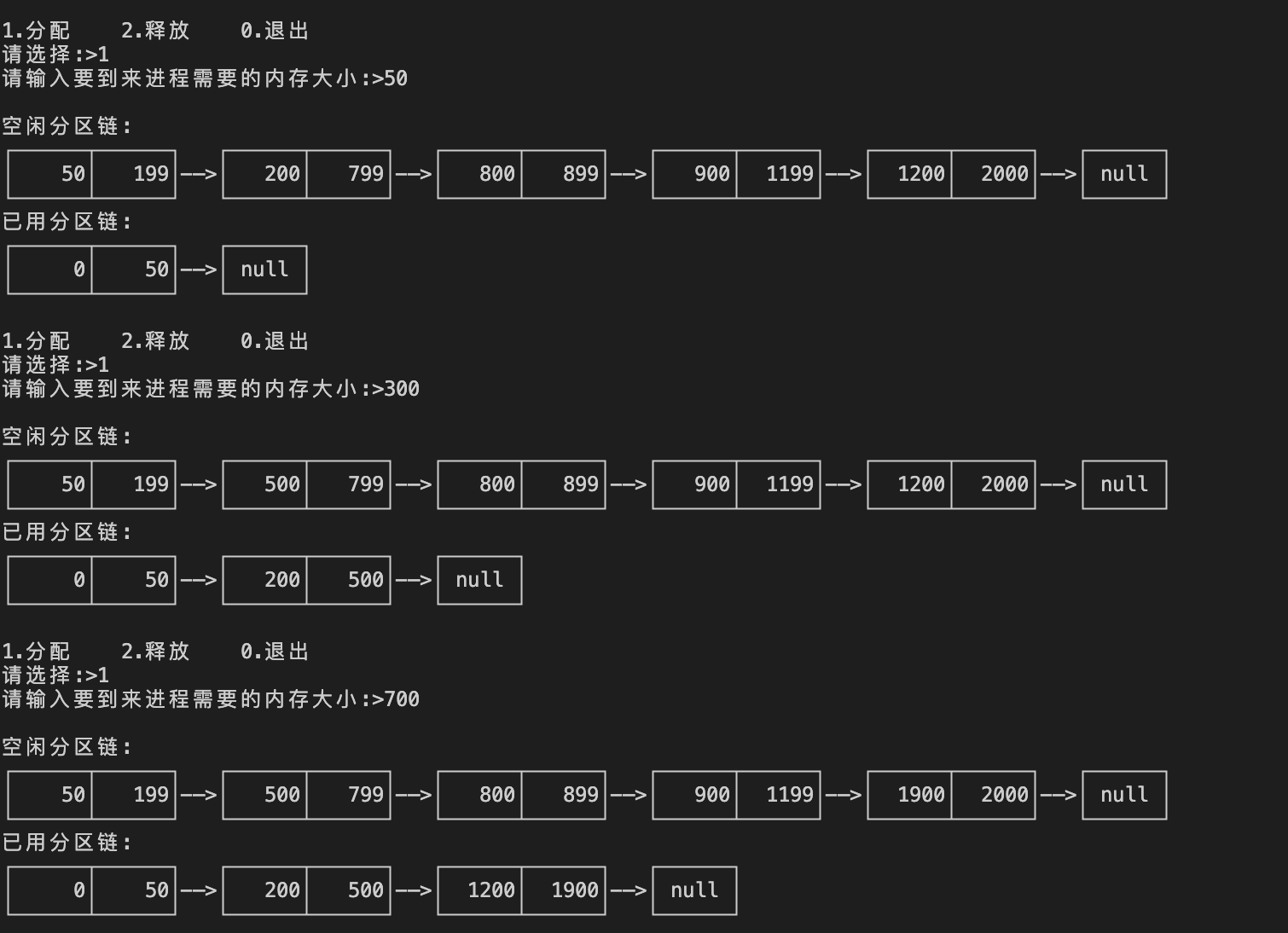
main.cpp

1. #include "os.h"
3. **void** menu()
4. {
5. **int** chose = 0;
6. **while**(1)
7. {
8. os s;
9. s.init();
10. s.create\_empty();
11. cout << "1.首次适应算法" << endl;
12. cout << "2.循环首次适应算法" << endl;
13. cout << "3.最佳适应算法" << endl;
14. cout << "4.最坏适应算法" << endl;
15. cout << "0.退出" << endl;
17. cout << "请输入选项:>";
18. cin >> chose;
19. **switch**(chose){
20. **case** 0:exit(0);
21. **case** 1:s.FF();**break**;
22. **case** 2:s.NF();**break**;
23. **case** 3:s.BF();**break**;
24. **case** 4:s.WF();**break**;
25. **default**:**continue**;
26. }
27. }
28. }
30. **int** main()
31. {
32. menu();
33. **return** 0;
34. }

运行结果：

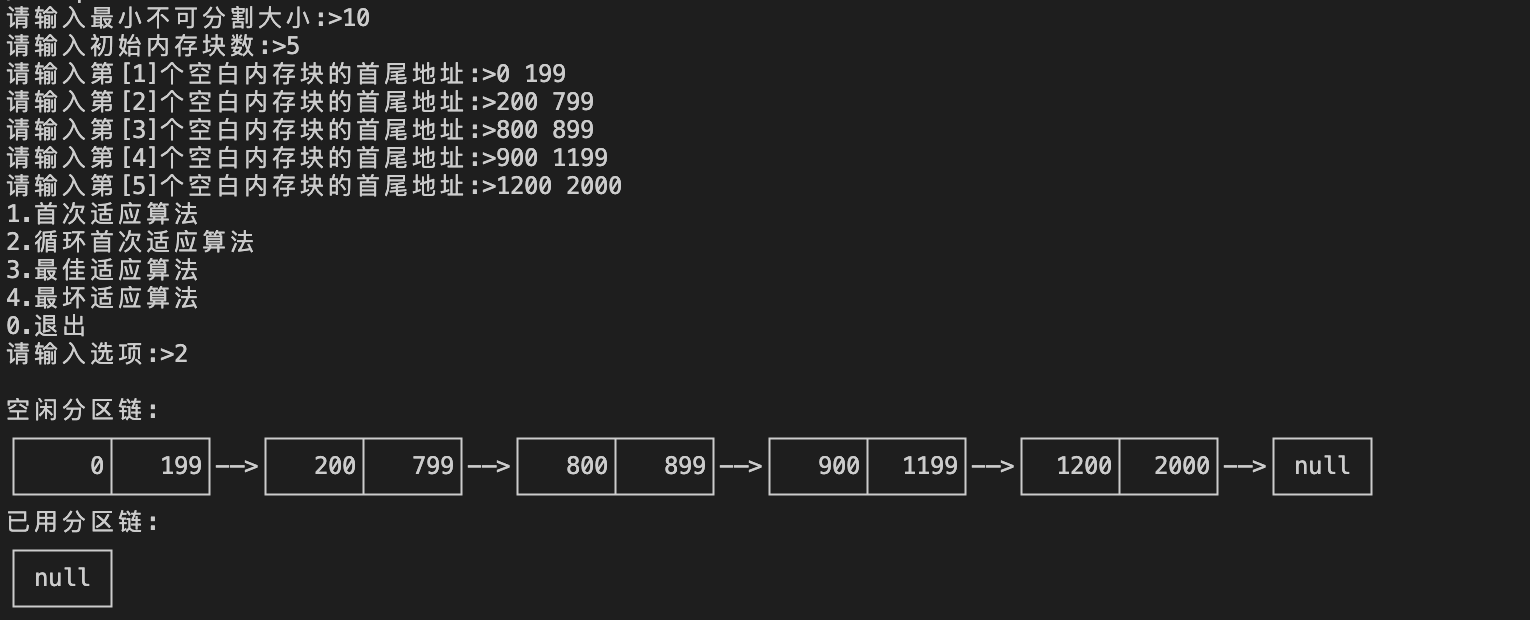
首次适应算法：

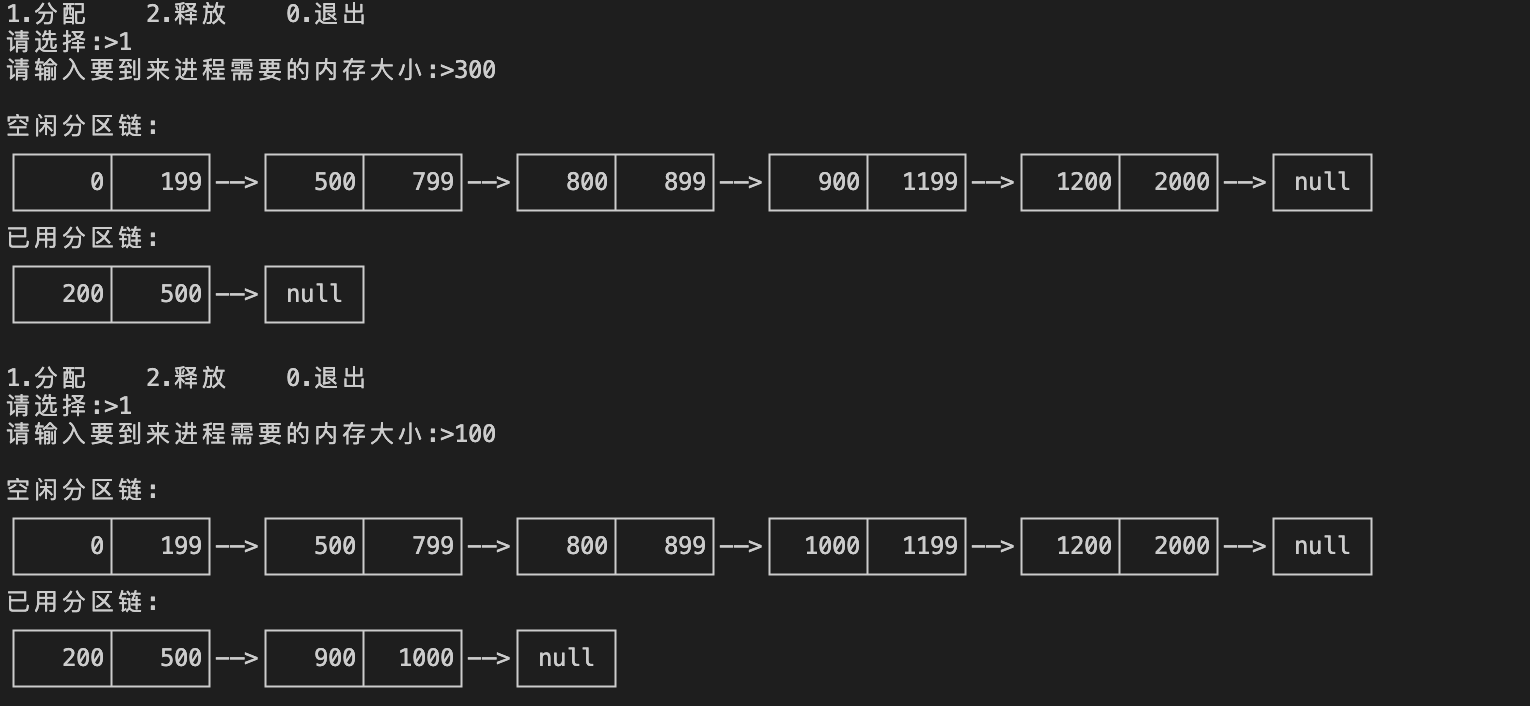


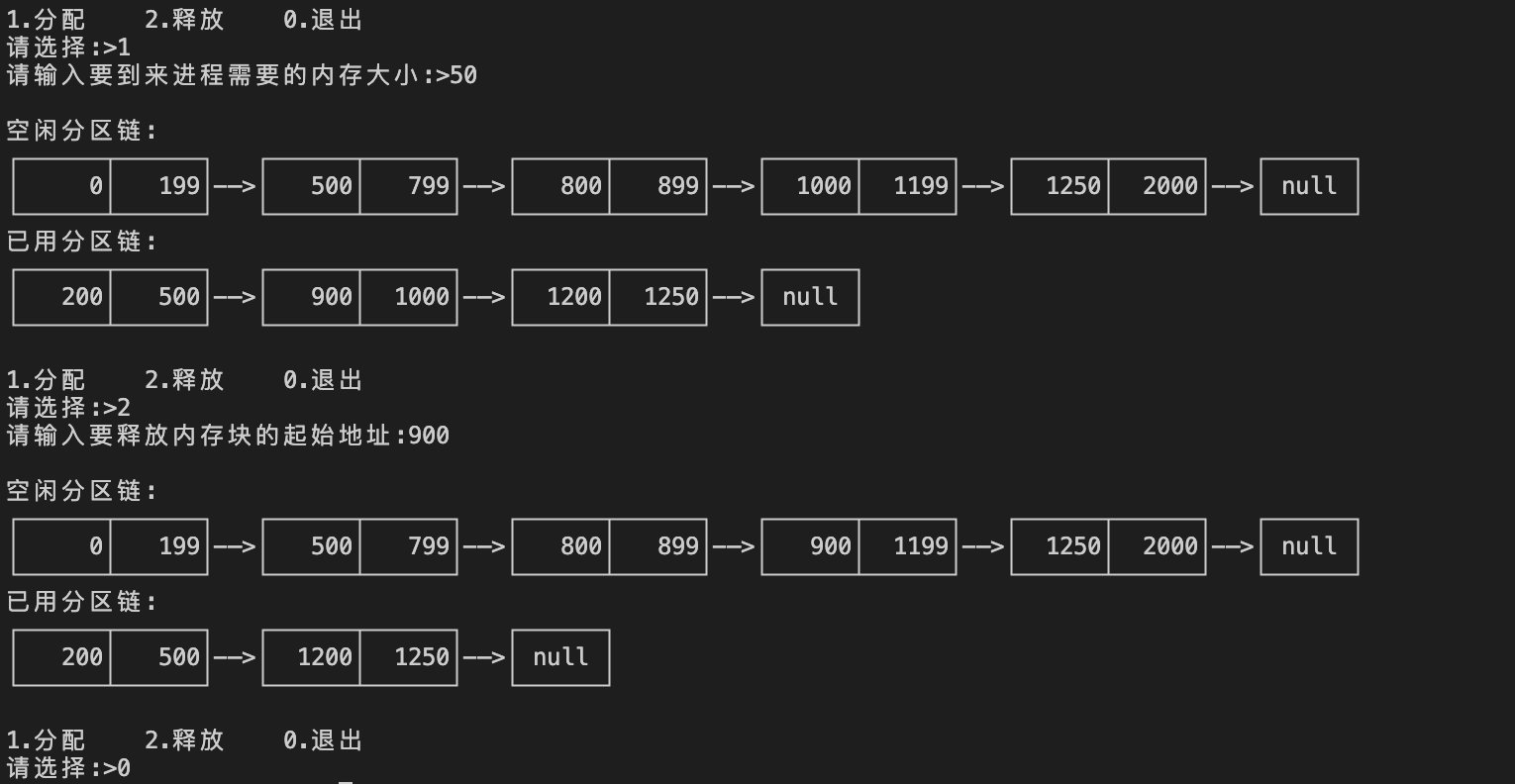




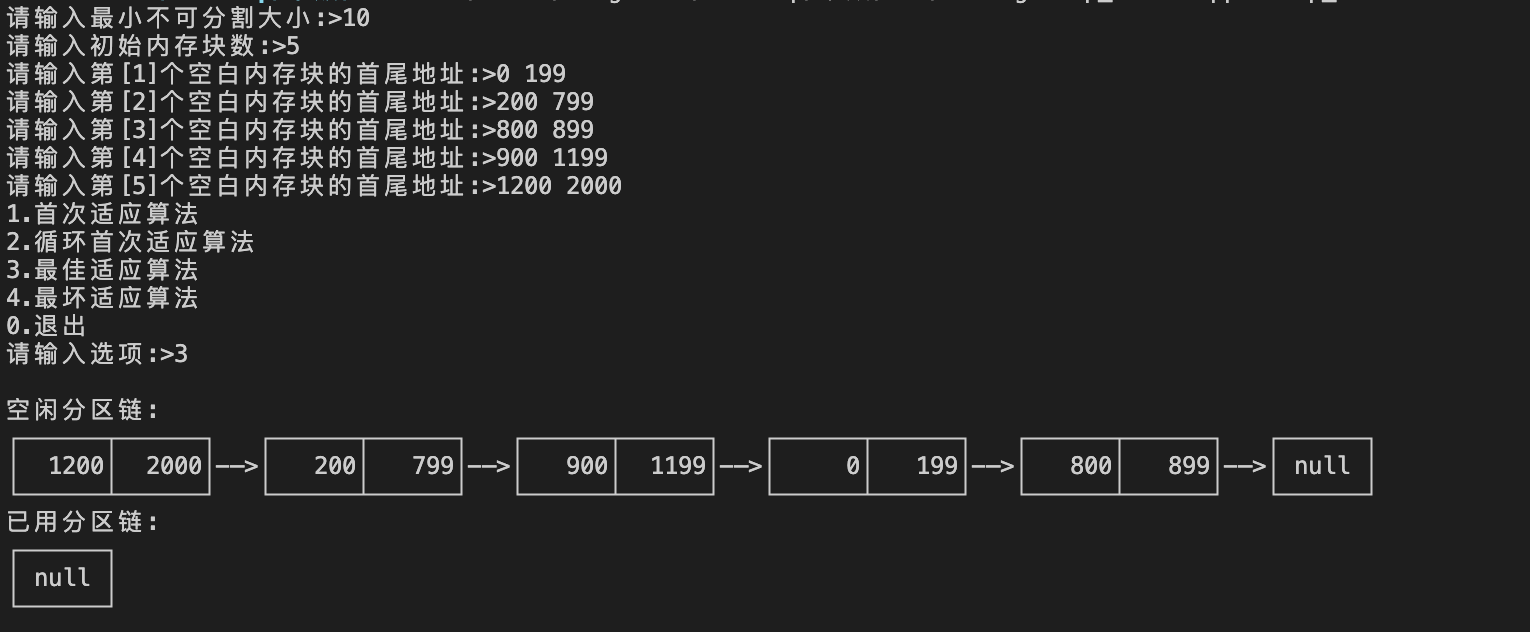
循环首次适应算法

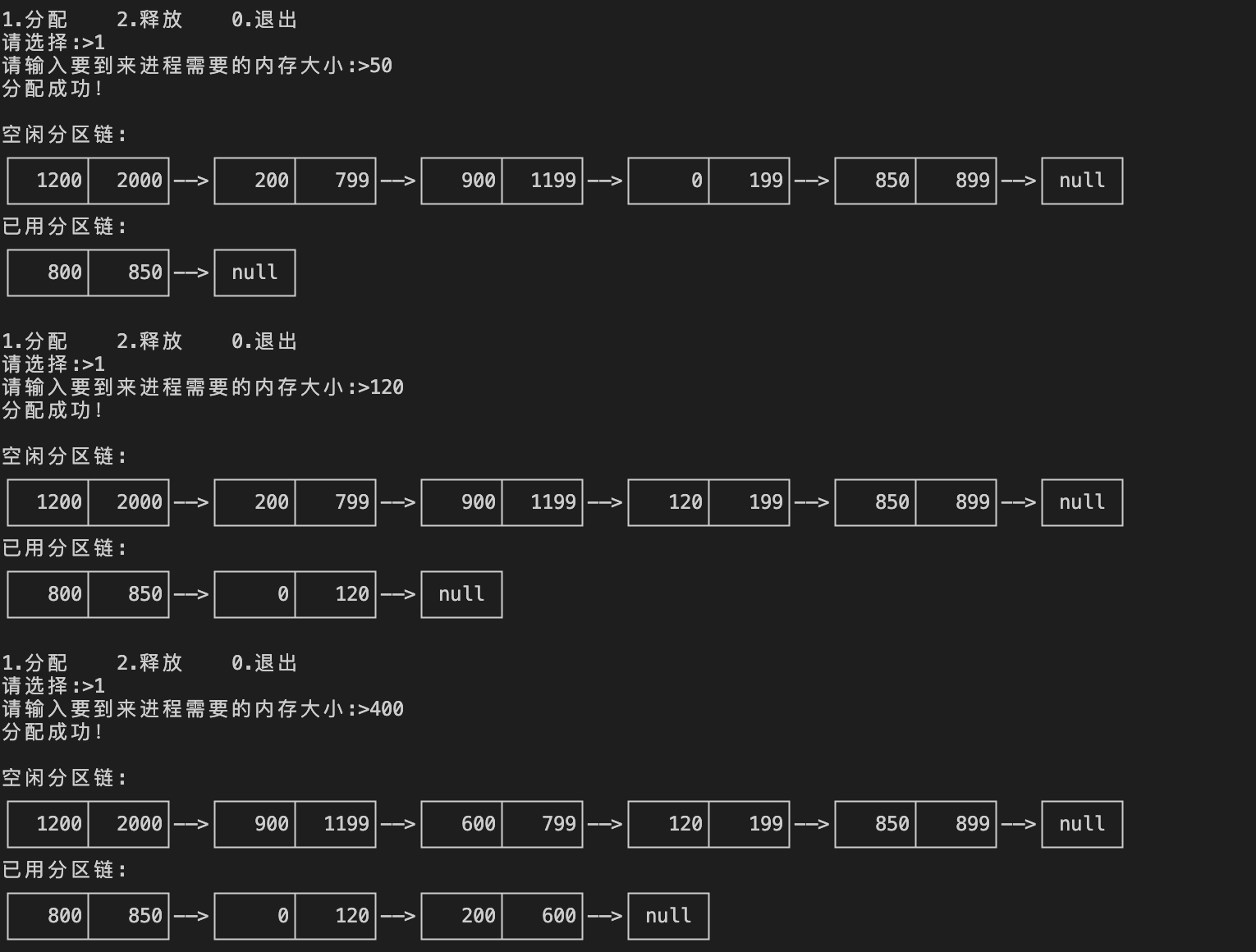






最佳适应算法：







最坏适应算法

