模拟内存分配与回收

一、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。

二、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:

```
#ifndef _OS_H_
#define _OS_H_

#include <list>
#include <algorithm>
#include <iostream>
#include "memNode.h"
using namespace std;

enum flags{
    SIZE_ASC,
    SIZE_DESC,
    ADDR_ASC,
    NO_LOOP,
    LOOP
};
/*
const int SIZE_DESC = 0;
const int SIZE_ASC = 1;
const int ADDR_ASC = 2;
const int NO_LOOP = 3;
const int LOOP = 4;
*/
class os
{
```

```
26.     public:
27.         os():_initNum(0),_minSize(0), _empty(), _use() {}
28.
29.         void init()
30.         {
31.             cout << "请输入最小不可分割大小:>";
32.             cin >> _minSize;
33.
34.             cout << "请输入初始内存块数:>";
35.             cin >> _initNum;
36.         }
37.
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.         }
48.
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.         }
58.
59.         bool alloc(int size, list<memNode*>::iterator& start, int flag){
60.             list<memNode*>::iterator it = start;
61.
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() + s
ize);
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_beg
in() + 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.            }
102.
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.         }
122.
123.         void output(){
124.             cout << endl;
125.             cout << "空闲分区链:\n";
126.             output_list(_empty);
127.
128.             cout << "已用分区链:\n";
129.             output_list(_use);
130.             cout << endl;
131.         }
132.
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.         }
151.
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.        }
169.
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.        }
183.
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.        }
203.
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.        }
223.
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.        }
245.
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.        }
267.
268.        ~os() {
269.            for(auto e:_empty){
270.                delete e;
271.            }
272.            for(auto e:_use){
273.                delete e;
274.            }
275.        }
276.
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;
```

```
288.
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;
295.
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.    };
303.
304.    #endif
```

memNode.h

```
1.      #ifndef _MEMNODE_H_
2.      #define _MEMNODE_H_
3.
4.      class memNode
5.      {
6.      public:
7.          memNode() {}
8.
9.          int get_begin()const { return _begin; }
10.
11.         int get_end()const { return _end; }
12.
13.         int get_size()const { return _size; }
14.
15.         void set_node(int begin, int end){
16.             _begin = begin;
17.             _end = end;
18.             _size = _end - _begin;
19.         };
20.
21.         void cut_node(int n){
22.             _begin += n;
23.             _size = _end- _begin;
24.         }
25.
26.         ~memNode() {}
```

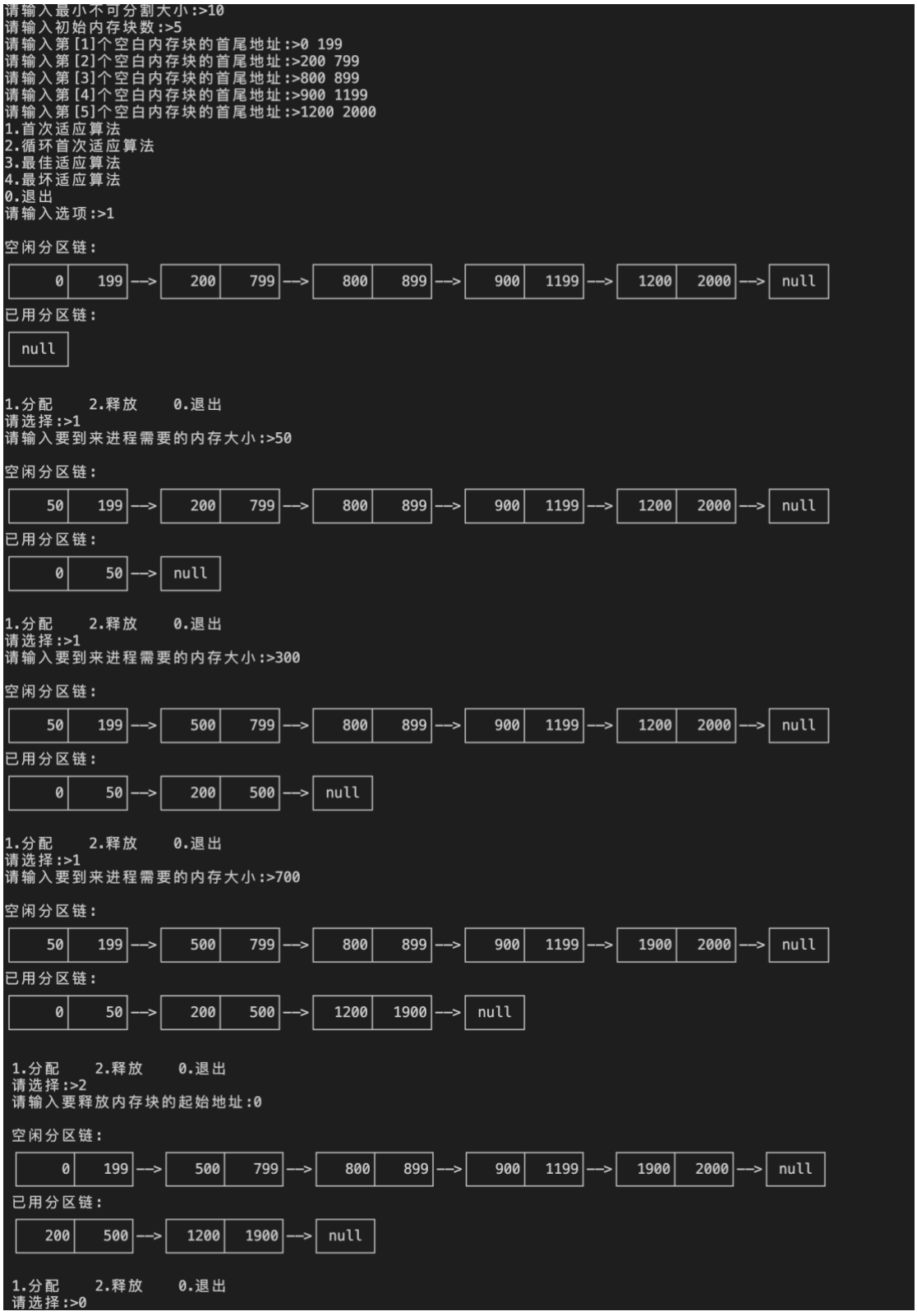
```
    private:
        int _begin;
        int _end;
        int _size;
    };


    #endif
```

main.cpp

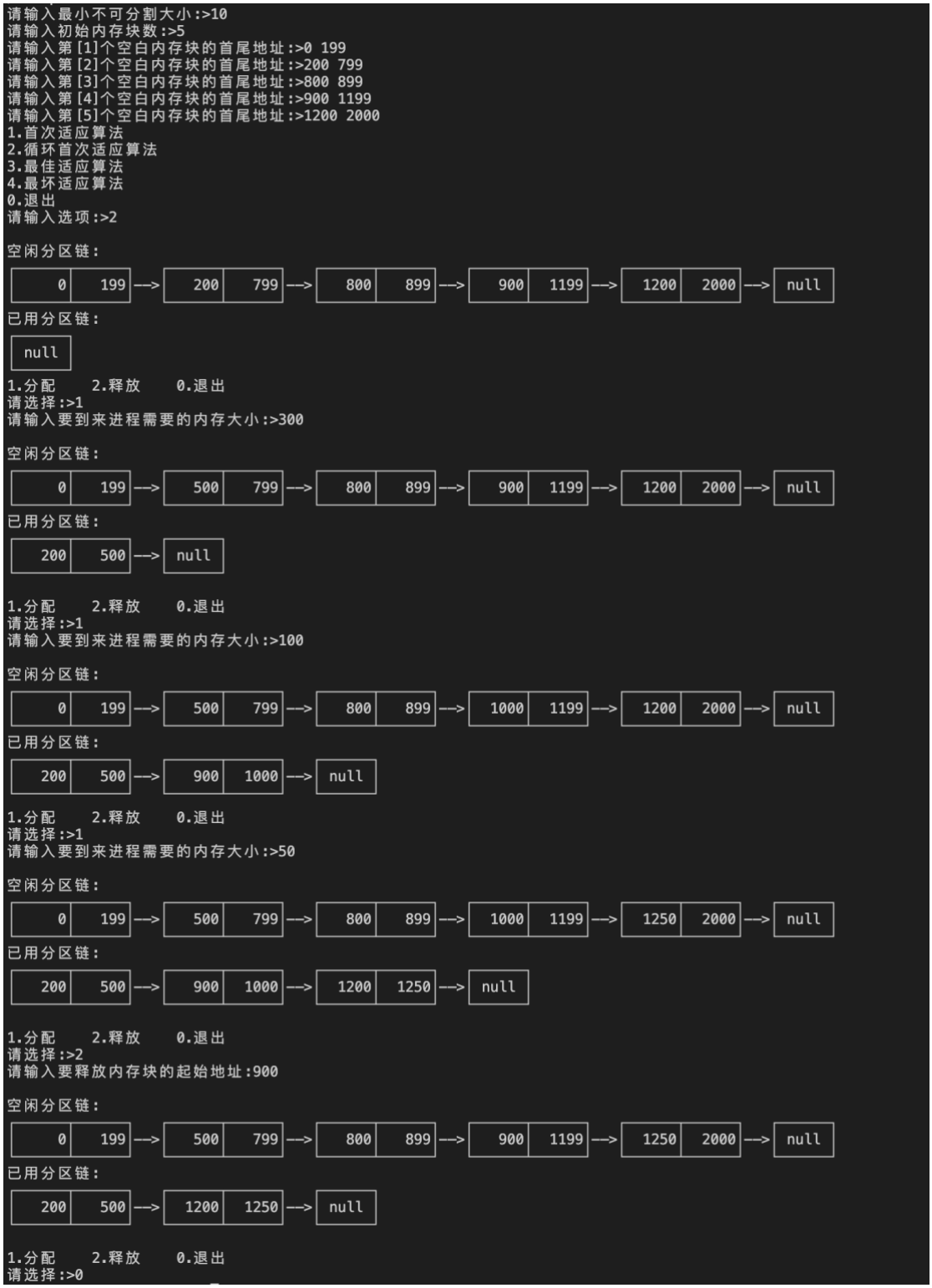
```
    #include "os.h"

    void menu()
    {
        int chose = 0;
        while(1)
        {
            os s;
            s.init();
            s.create_empty();
            cout << "1.首次适应算法" << endl;
            cout << "2.循环首次适应算法" << endl;
            cout << "3.最佳适应算法" << endl;
            cout << "4.最坏适应算法" << endl;
            cout << "0.退出" << endl;

            cout << "请输入选项:>";
            cin >> chose;
            switch(chose){
            case 0:exit(0);
            case 1:s.FF();break;
            case 2:s.NF();break;
            case 3:s.BF();break;
            case 4:s.WF();break;
            default:continue;
            }
        }
    }

    int main()
    {
        menu();
        return 0;
    }
```

运行结果：

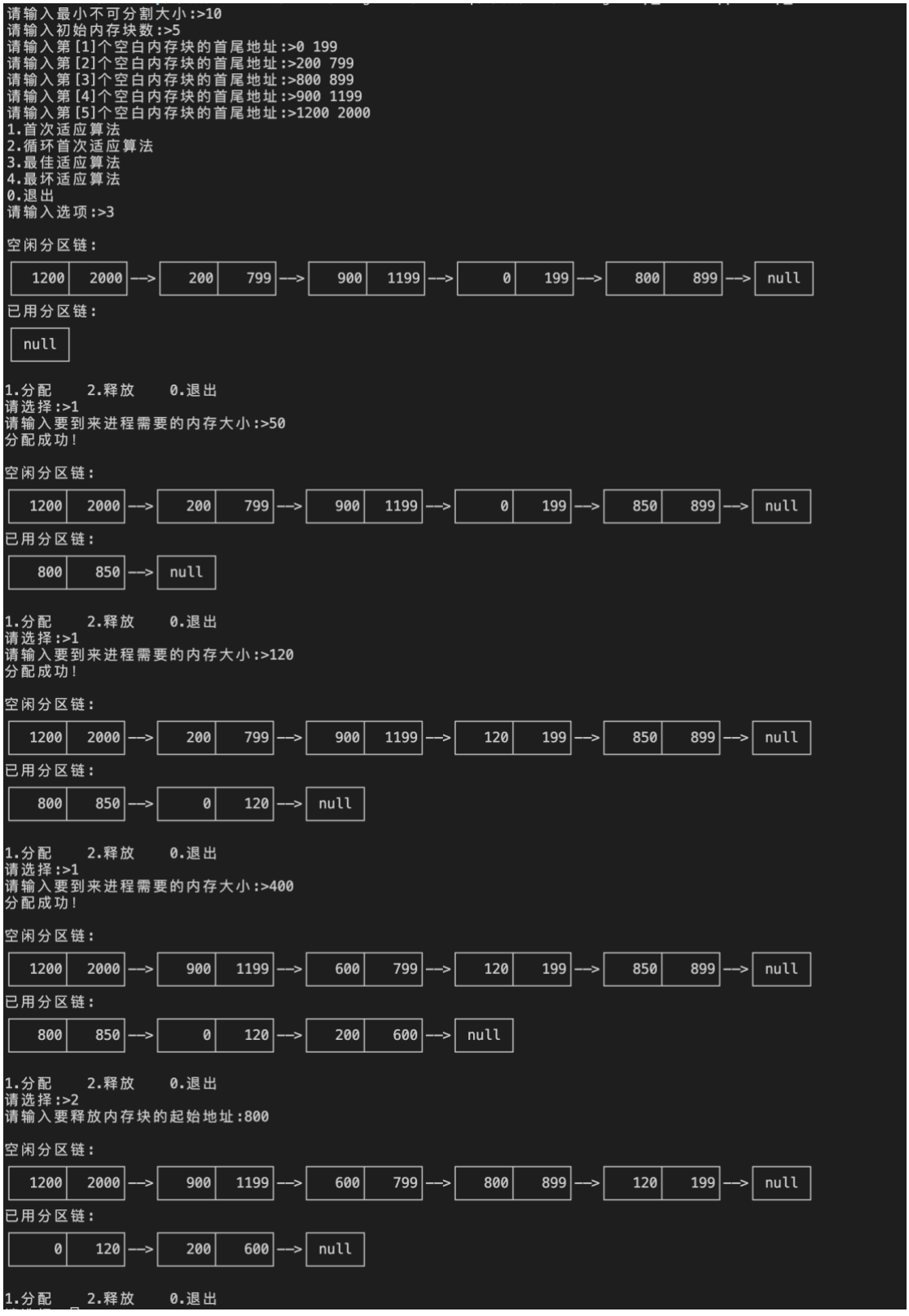
首次适应算法：



循环首次适应算法


请输入最小不可分割大小:>10
请输入初始内存块数:>5
请输入第[1]个空白内存块的首尾地址:>0 199
请输入第[2]个空白内存块的首尾地址:>200 799
请输入第[3]个空白内存块的首尾地址:>800 899
请输入第[4]个空白内存块的首尾地址:>900 1199
请输入第[5]个空白内存块的首尾地址:>1200 2000
1.首次适应算法
2.循环首次适应算法
3.最佳适应算法
4.最坏适应算法
0.退出
请输入选项:>2
空闲分区链:
0 199 --> 200 799 --> 800 899 --> 900 1199 --> 1200 2000 --> null
已用分区链:
null
1.分配 2.释放 0.退出
请选择:>1
请输入要到来进程需要的内存大小:>300
空闲分区链:
0 199 --> 500 799 --> 800 899 --> 900 1199 --> 1200 2000 --> null
已用分区链:
200 500 --> null
1.分配 2.释放 0.退出
请选择:>1
请输入要到来进程需要的内存大小:>100
空闲分区链:
0 199 --> 500 799 --> 800 899 --> 1000 1199 --> 1200 2000 --> null
已用分区链:
200 500 --> 900 1000 --> null
1.分配 2.释放 0.退出
请选择:>1
请输入要到来进程需要的内存大小:>50
空闲分区链:
0 199 --> 500 799 --> 800 899 --> 1000 1199 --> 1250 2000 --> null
已用分区链:
200 500 --> 900 1000 --> 1200 1250 --> null
1.分配 2.释放 0.退出
请选择:>2
请输入要释放内存块的起始地址:900
空闲分区链:
0 199 --> 500 799 --> 800 899 --> 900 1199 --> 1250 2000 --> null
已用分区链:
200 500 --> 1200 1250 --> null
1.分配 2.释放 0.退出
请选择:>0

最佳适应算法：


请输入最小不可分割大小:>10
请输入初始内存块数:>5
请输入第[1]个空白内存块的首尾地址:>0 199
请输入第[2]个空白内存块的首尾地址:>200 799
请输入第[3]个空白内存块的首尾地址:>800 899
请输入第[4]个空白内存块的首尾地址:>900 1199
请输入第[5]个空白内存块的首尾地址:>1200 2000
1.首次适应算法
2.循环首次适应算法
3.最佳适应算法
4.最坏适应算法
0.退出
请输入选项:>3
空闲分区链:
1200 2000 --> 200 799 --> 900 1199 --> 0 199 --> 800 899 --> null
已用分区链:
null
1.分配 2.释放 0.退出
请选择:>1
请输入要到来进程需要的内存大小:>50
分配成功!
空闲分区链:
1200 2000 --> 200 799 --> 900 1199 --> 0 199 --> 850 899 --> null
已用分区链:
800 850 --> null
1.分配 2.释放 0.退出
请选择:>1
请输入要到来进程需要的内存大小:>120
分配成功!
空闲分区链:
1200 2000 --> 200 799 --> 900 1199 --> 120 199 --> 850 899 --> null
已用分区链:
800 850 --> 0 120 --> null
1.分配 2.释放 0.退出
请选择:>1
请输入要到来进程需要的内存大小:>400
分配成功!
空闲分区链:
1200 2000 --> 900 1199 --> 600 799 --> 120 199 --> 850 899 --> null
已用分区链:
800 850 --> 0 120 --> 200 600 --> null
1.分配 2.释放 0.退出
请选择:>2
请输入要释放内存块的起始地址:800
空闲分区链:
1200 2000 --> 900 1199 --> 600 799 --> 800 899 --> 120 199 --> null
已用分区链:
0 120 --> 200 600 --> null
1.分配 2.释放 0.退出

最坏适应算法

```
请输入最小不可分割大小:>10
请输入初始内存块数:>5
请输入第[1]个空白内存块的首尾地址:>0 199
请输入第[2]个空白内存块的首尾地址:>200 799
请输入第[3]个空白内存块的首尾地址:>800 899
请输入第[4]个空白内存块的首尾地址:>900 1199
请输入第[5]个空白内存块的首尾地址:>1200 2000
1.首次适应算法
2.循环首次适应算法
3.最佳适应算法
4.最坏适应算法
0.退出
请输入选项:>4

空闲分区链:
| 1200 | 2000 | --> | 200 | 799 | --> | 900 | 1199 | --> | 0 | 199 | --> | 800 | 899 | --> | null |
已用分区链:
| null |

1.分配    2.释放    0.退出
请选择:>1
请输入要到来进程需要的内存大小:>50

空闲分区链:
| 1250 | 2000 | --> | 200 | 799 | --> | 900 | 1199 | --> | 0 | 199 | --> | 800 | 899 | --> | null |
已用分区链:
| 1200 | 1250 | --> | null |

1.分配    2.释放    0.退出
请选择:>1
请输入要到来进程需要的内存大小:>400

空闲分区链:
| 200 | 799 | --> | 1650 | 2000 | --> | 900 | 1199 | --> | 0 | 199 | --> | 800 | 899 | --> | null |
已用分区链:
| 1200 | 1250 | --> | 1250 | 1650 | --> | null |

1.分配    2.释放    0.退出
请选择:>1
请输入要到来进程需要的内存大小:>300

空闲分区链:
| 1650 | 2000 | --> | 500 | 799 | --> | 900 | 1199 | --> | 0 | 199 | --> | 800 | 899 | --> | null |
已用分区链:
| 1200 | 1250 | --> | 1250 | 1650 | --> | 200 | 500 | --> | null |

1.分配    2.释放    0.退出
请选择:>2
请输入要释放内存块的起始地址:1200

空闲分区链:
| 1650 | 2000 | --> | 500 | 799 | --> | 900 | 1199 | --> | 0 | 199 | --> | 800 | 899 | --> | 1200 | 1250 | --> |
已用分区链:
| 1250 | 1650 | --> | 200 | 500 | --> | null |

1.分配    2.释放    0.退出
```