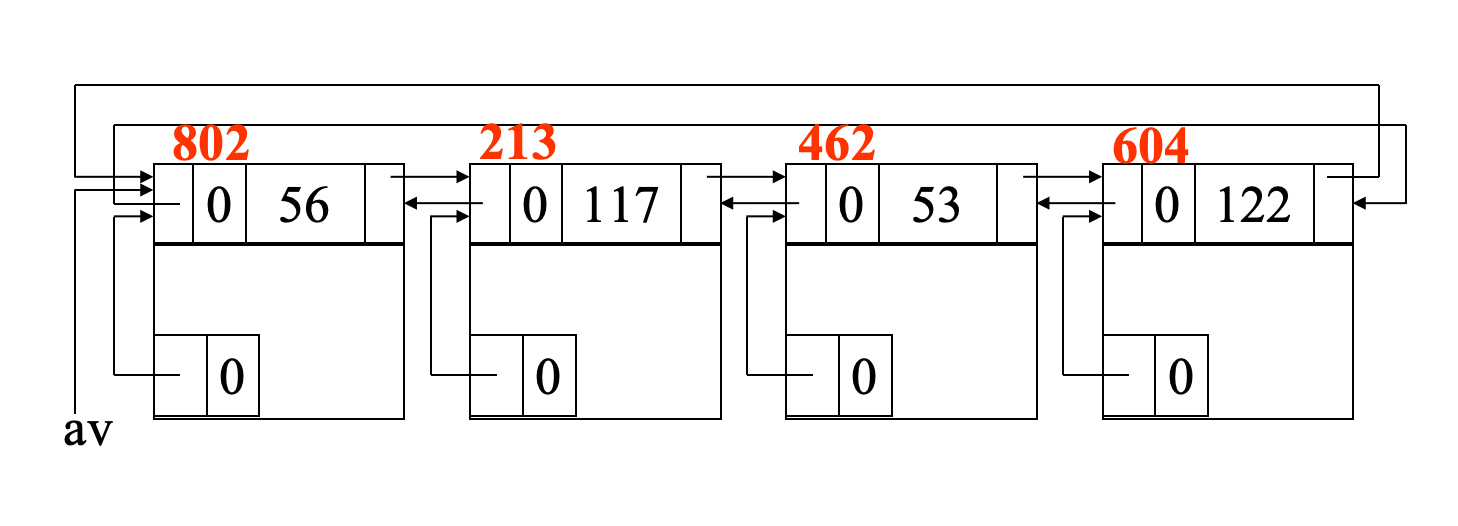
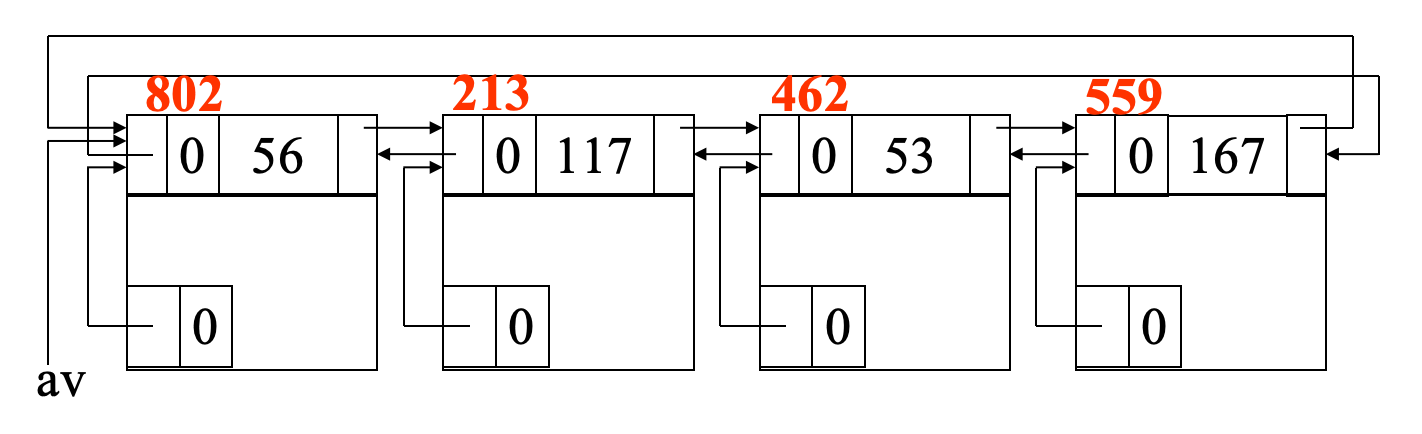
1. **假设利用边界标识法首次匹配策略分配，已知在某个时刻的可利用空间表如下：**

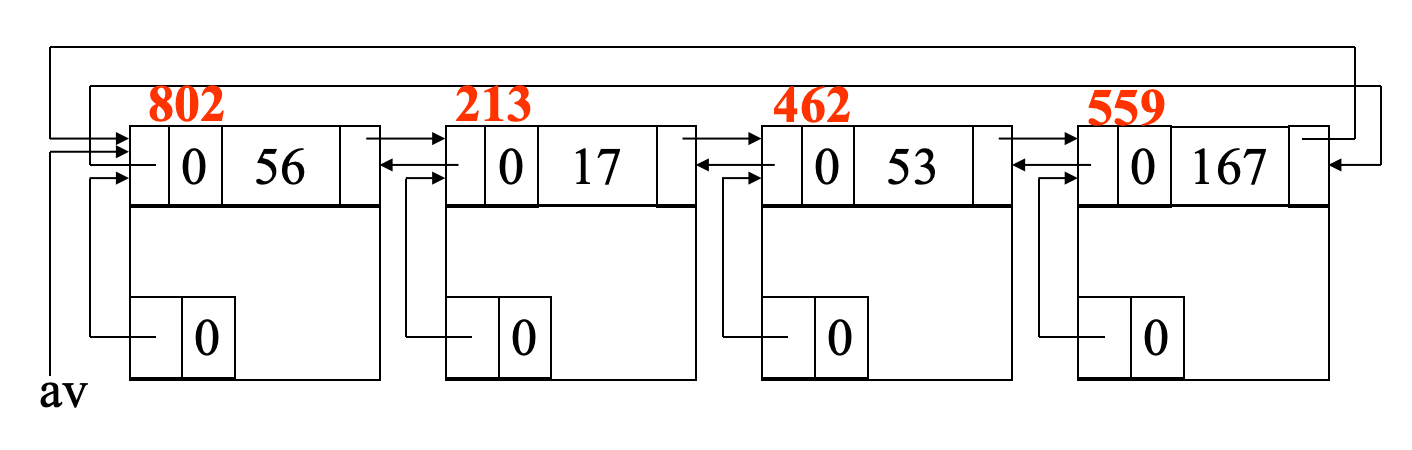
****

1. **请画出当系统回收一个起始地址为559、大小为45的空闲块之后的链表状态； （2）请画出系统在之后接受大小为 100的请求后又回收地址为515、大小为44的空闲块后的链表状态。**

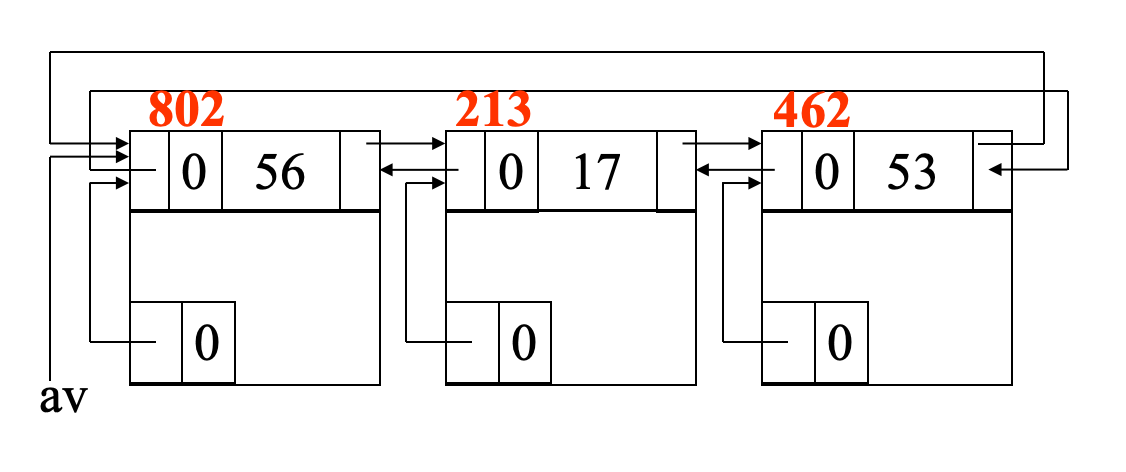
（1）与最后一个块合并



（2）接受大小为100的请求，从第二个块中分配：



回收空闲块，与最后两个块合并：



1. **查找序列以带头结点的单链表表示，各结点中设一个访问频度，初始值为 0，每次查找成功后该结点频度值增加 1。试给出算法，在每次查找后查找序列均按访问频度从大到小排列。**

#include <iostream>

using namespace std;

struct Node

{

char data;

int visit;

struct Node \*next;

};

// 创建单链表

Node \*initNodeList(void)

{

Node \*head = new Node, \*tail = head; // 头结点，尾结点

head->next = NULL, head->visit = INT\_MAX;

cout << "请输入结点字符值，换行结束：";

char c;

while ((c = getchar()) != '\n')

{

Node \*newNode = new Node;

newNode->data = c, newNode->visit = 0, newNode->next = NULL;

tail->next = newNode, tail = newNode;

}

return head;

}

// 查找算法

void search(Node \*head)

{

cout << "请输入要查找的值，换行结束：";

char key = getchar();

while (key != '\n')

{

Node \*cur = head->next, \*last = head;

while (cur)

{

if (cur->data == key) // 找到了

{

cur->visit++;

Node \*tmp = head;

bool insertFlag = true;

while (tmp->next->visit >= cur->visit)

{

if (tmp->next == cur)

{

insertFlag = false;

break;

}

tmp = tmp->next;

}

if (insertFlag) // 要插入

last->next = cur->next, cur->next = tmp->next, tmp->next = cur;

break;

}

cur = cur->next, last = last->next;

}

key = getchar();

}

return;

}

// 输出单链表

void printNodeList(Node \*head)

{

Node \*cur = head->next;

while (cur)

{

cout << cur->data;

cur = cur->next;

}

return;

}

int main()

{

Node \*l = initNodeList();

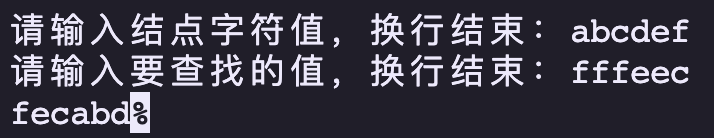
search(l);

printNodeList(l);

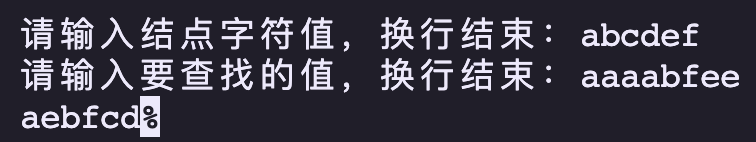
return 0;

}

测试1：

****

测试2：

****

**26. 试给出判别一棵给定的二叉树是否是排序二叉树的算法，设二叉树以二叉链表表示。**

#include <iostream>

using namespace std;

struct Node

{

int data;

Node \*lc, \*rc;

};

typedef Node \*BiTree;

// 读取字符，先序创建二叉树

void crtBt(BiTree &t)

{

int data;

cin >> data;

if (data == -1)

t = NULL;

else

{

t = new Node;

t->data = data, t->lc = NULL, t->rc = NULL;

crtBt(t->lc), crtBt(t->rc);

}

return;

}

// 排序二叉树算法

void bst(Node \*cur, Node \*&last, bool &isBst) // ldr

{

if (isBst == false) // 递归中断条件

return;

if (cur->lc) // 左子树

bst(cur->lc, last, isBst);

if (last == NULL || last->data <= cur->data) // 当前

last = cur;

else

isBst = false;

if (cur->rc) // 右子树

bst(cur->rc, last, isBst);

return;

}

// 释放二叉树结点

void deleteBt(BiTree t)

{

if (t == NULL)

return;

deleteBt(t->lc), deleteBt(t->rc), delete t;

}

int main()

{

cout << "创建二叉树，先序遍历，-1表示孩子为空：";

BiTree t;

crtBt(t);

bool isBst = true;

Node \*last = NULL;

bst(t, last, isBst);

isBst ? cout << "该二叉树是二叉排序树" : cout << "该二叉树不是二叉排序树";

deleteBt(t);

return 0;

}

测试1：





测试2：

