

线性穷举

蛮力迭代

YouTube频道: [hwdong](#)

博客: hwdong-net.github.io

穷举法

- 穷举法：穷举所有可能性。
- 根据穷举方式，可以分为**线性穷举**和**树形穷举**。

线性穷举

- 线性穷举：以线性迭代的方式穷举所有情况/元素。

for 每个情况(元素):

处理该情况(元素)

- 查找某个元素、求最大(小)值、插入排序、....

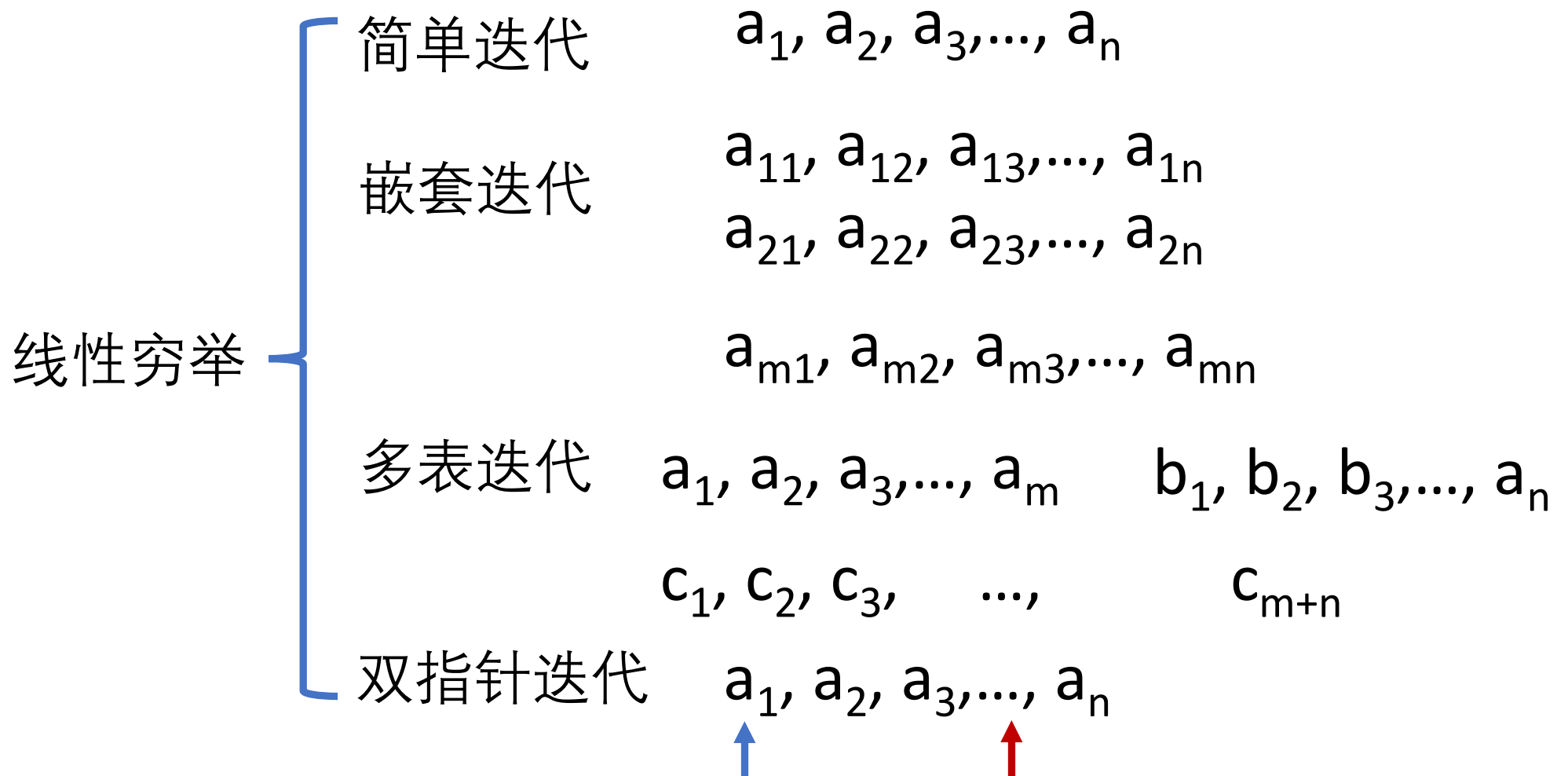
$(a_1, a_2, \dots, a_i, \dots, a_n)$

x



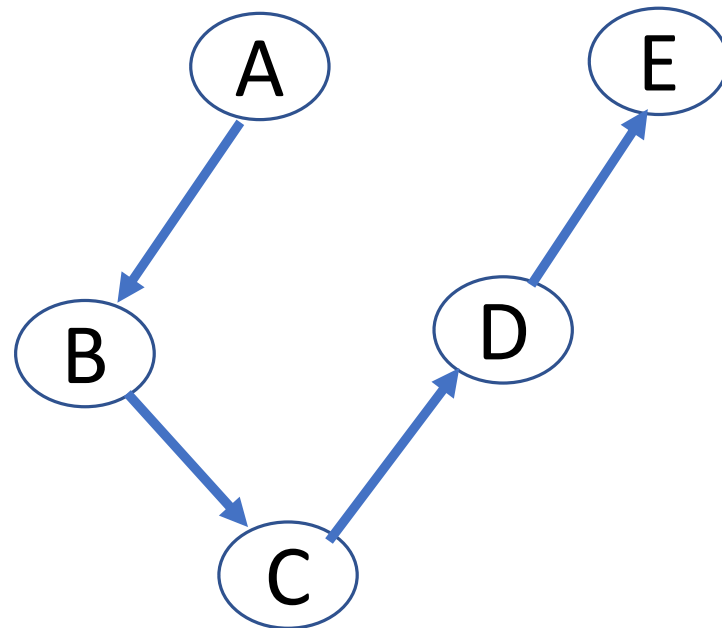
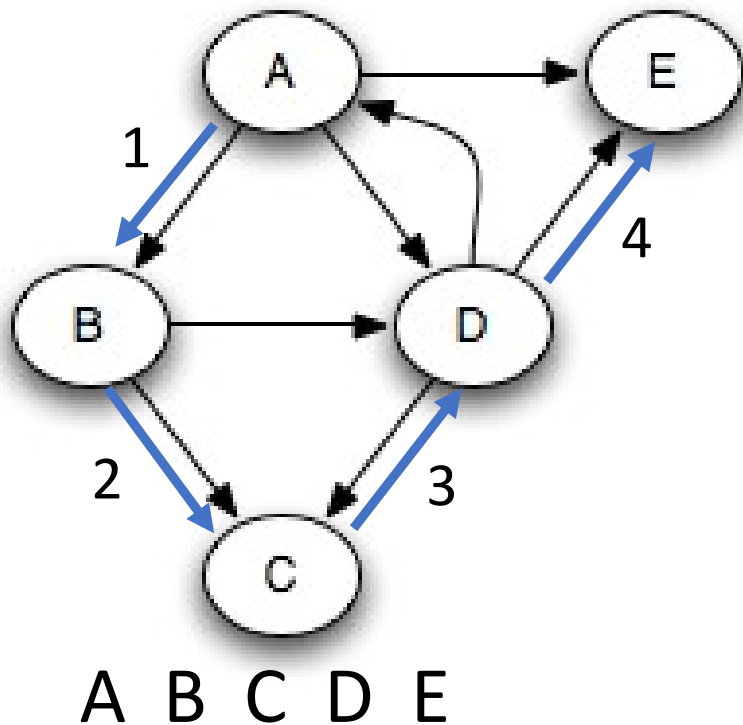
i=2

线性穷举的分类



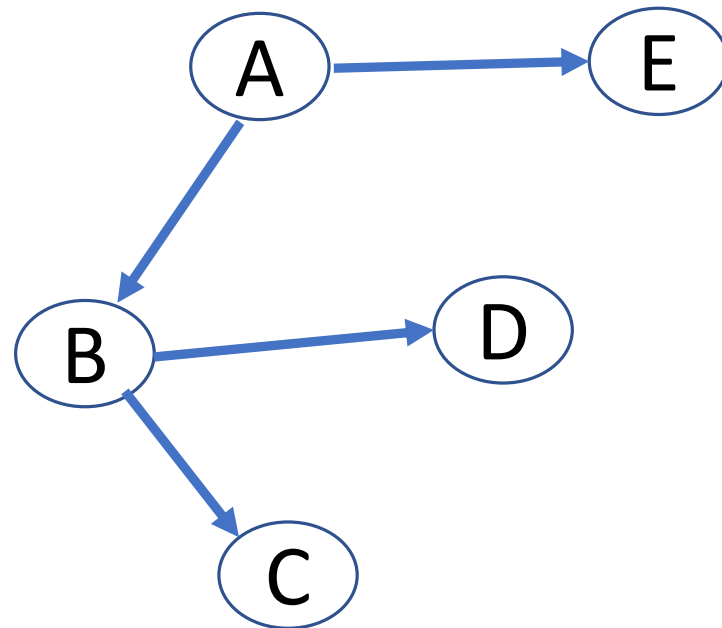
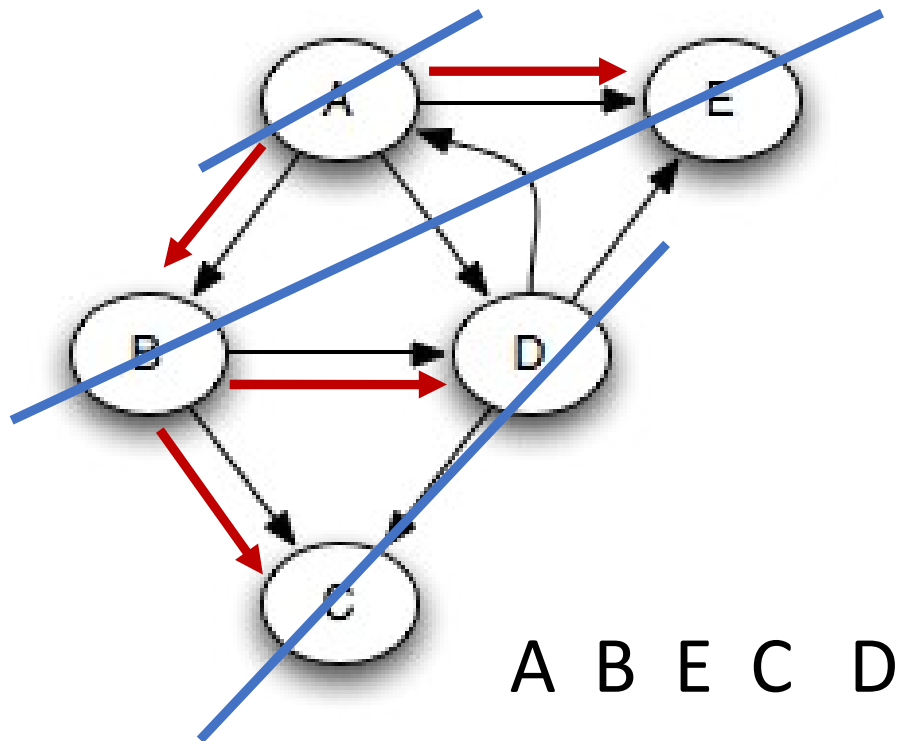
树形穷举

- 树形穷举：树形搜索所有情况（元素）
 - 树或图型结构的遍历



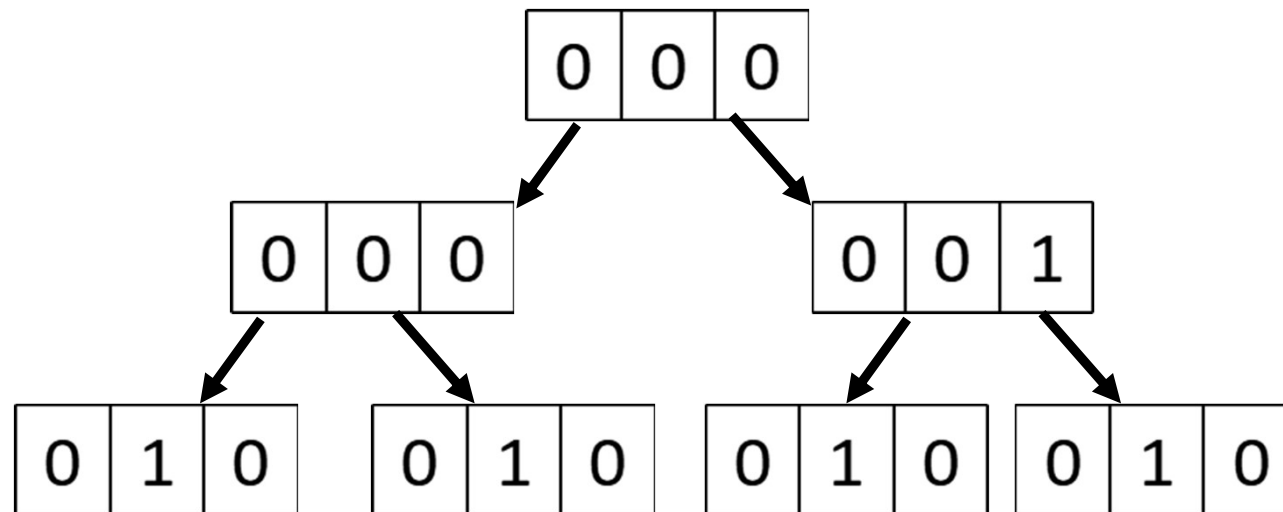
树形穷举

- 树形穷举：树形搜索所有情况（元素）
 - 树或图型结构的遍历



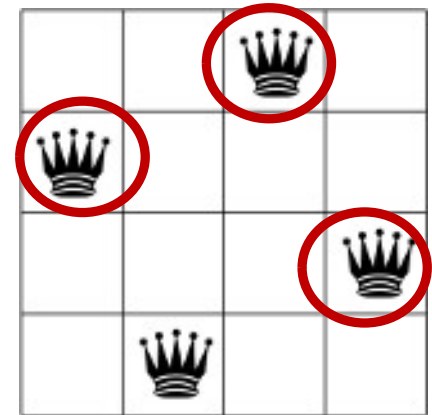
树形穷举

- 树形穷举：
 - 树或图型结构的遍历
 - 解决问题往往是一个多步决策过程，每一步多个选择，决策过程构成一个决策树或搜索树。

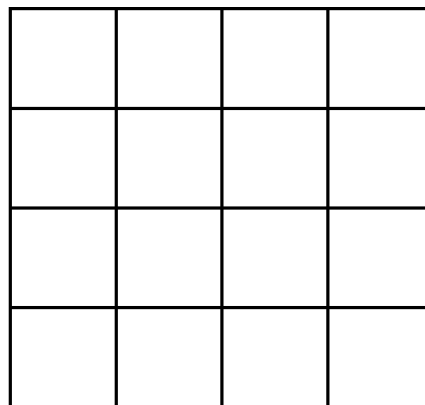


树形穷举

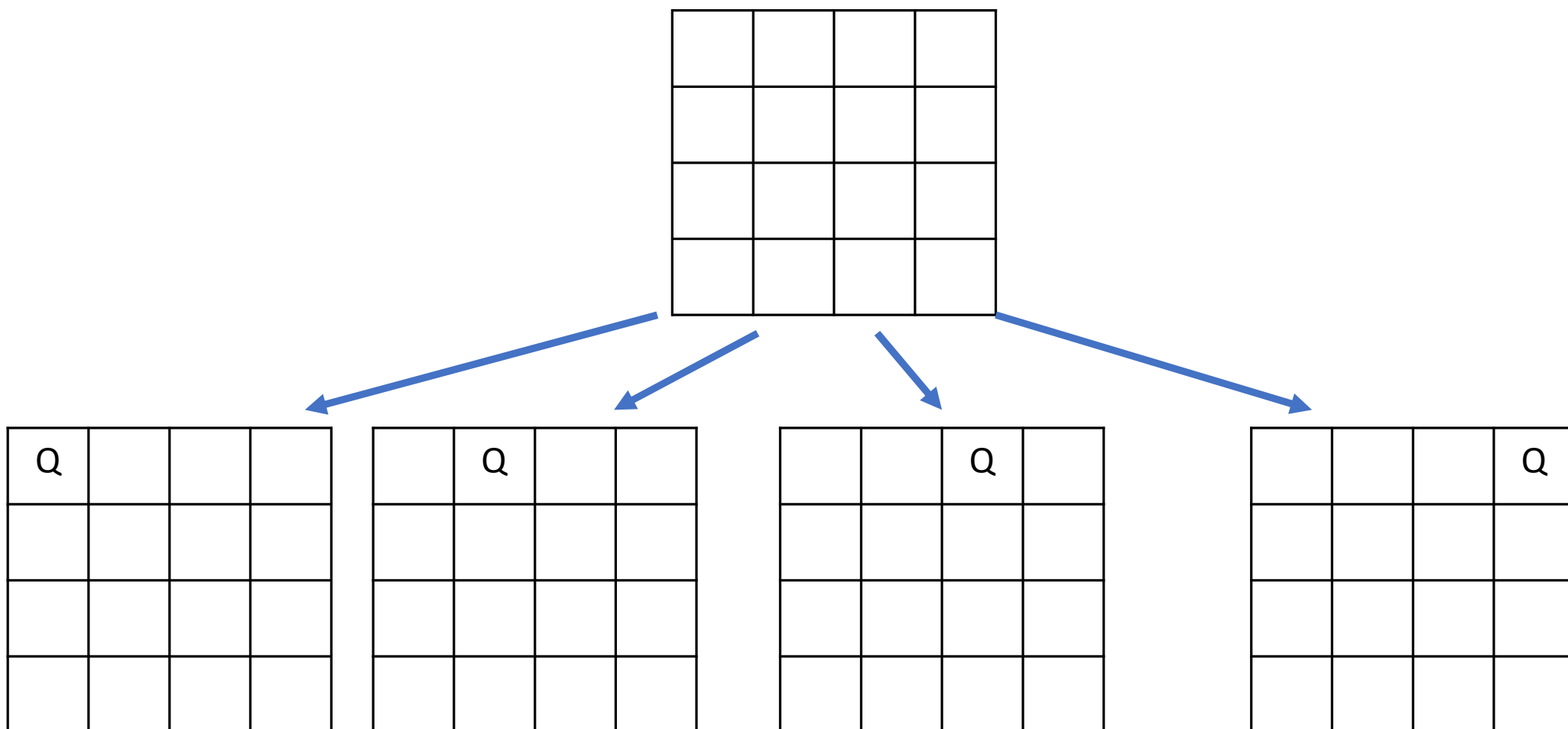
- N皇后问题：将 n 个皇后放置在 $n*n$ 的棋盘上，皇后彼此之间不能相互攻击(任意两个皇后不能位于同一行，同一列，同一斜线)。
- 多步决策：
 - 第 1 步：第1行皇后的位置决策
 - 第 2 步：第2行皇后的位置决策
 - 第 3 步：第3行皇后的位置决策

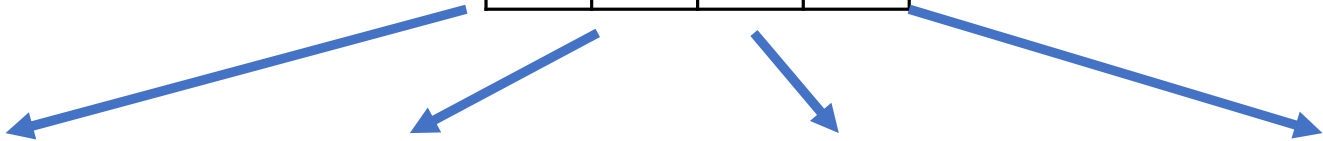


从初始状态出发



每一步决策：在某个状态，有多个决策选项





Q			

	Q		

		Q	

			Q



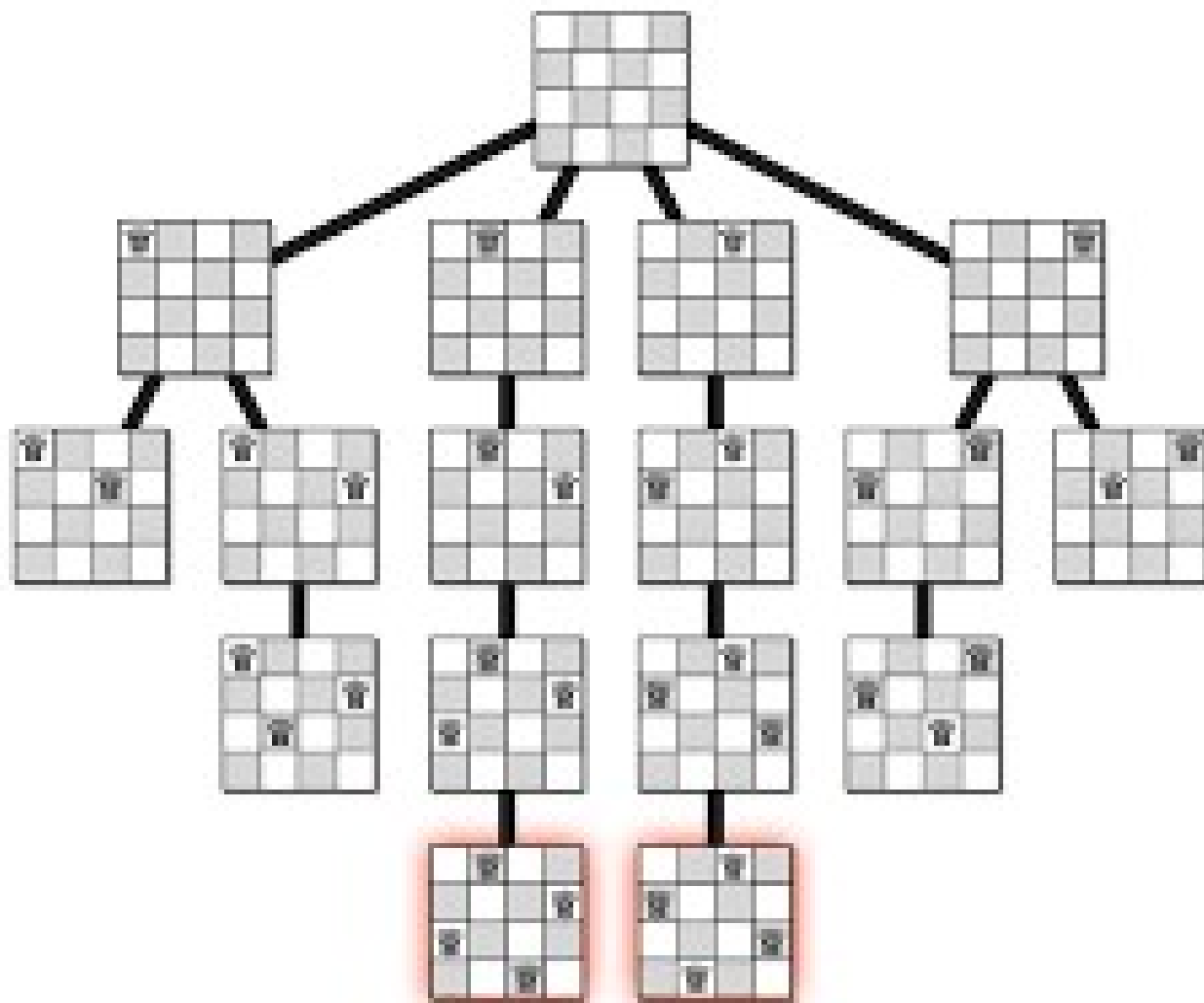
		Q	
Q			

		Q	
	Q		

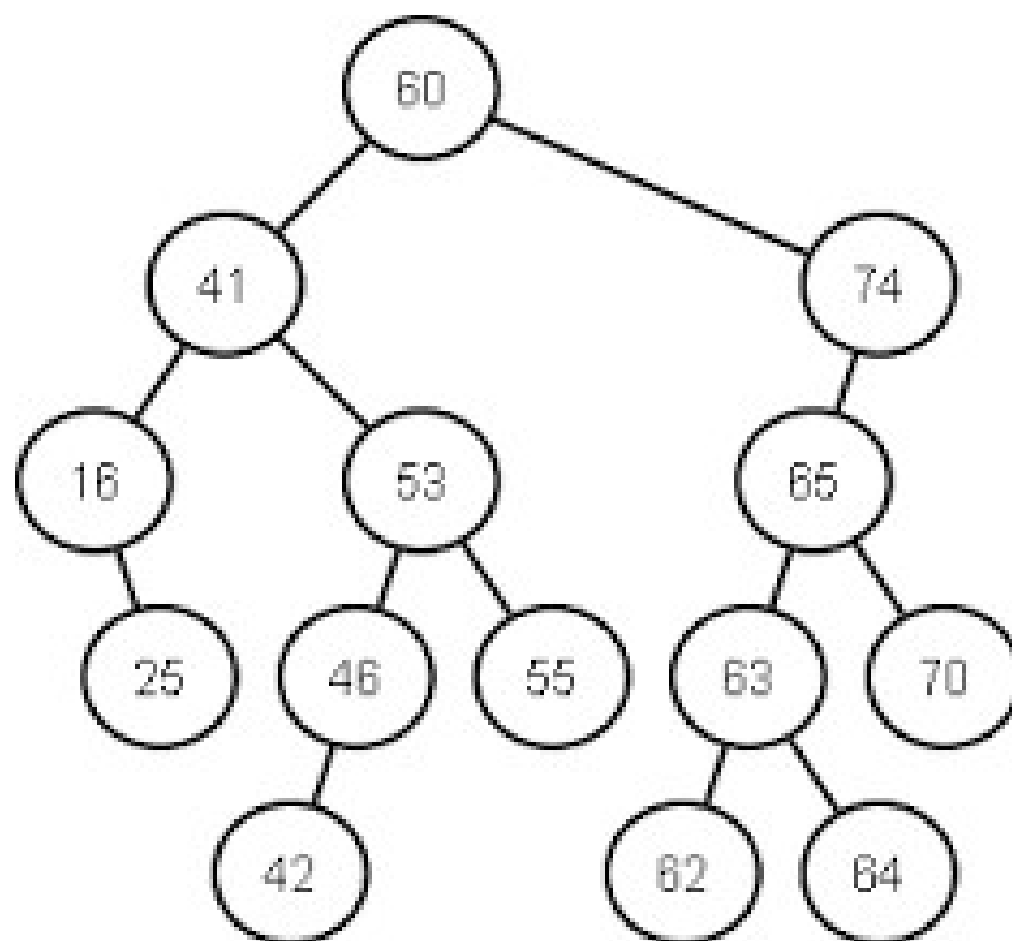
		Q	
		Q	

		Q	
			Q

- 树形穷举从根到叶子节点的路径，构成一个解。
- 剪枝：某个中间节点不可能产生解，不需要探索该节点代表的子树。



- 树形穷举的中间节点也可能是一个解，如查找 16。



穷举法

- 穷举法是很多算法设计方法的基础。线性表、树、图的遍历，分治递归、回溯法、分支限界、动态规划、贪婪法，本质上都是穷举或对穷举的改进。

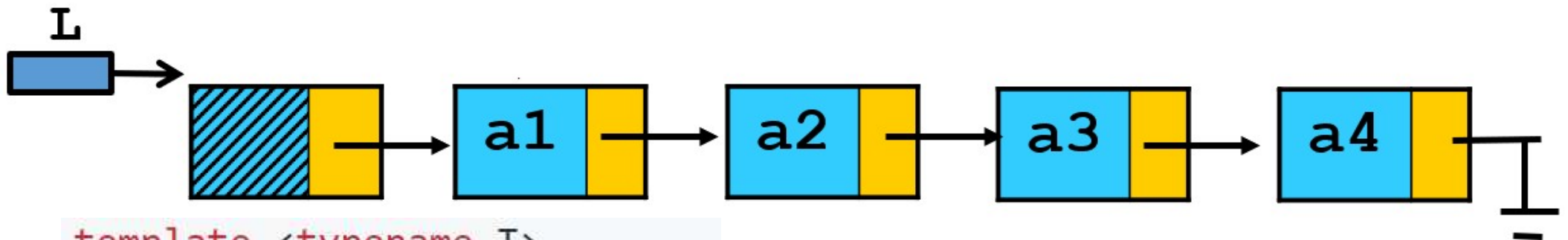
一、简单迭代

博客: hwdong-net.github.io

Youtube频道:hwdong

简单迭代

- 线性序列的2种存储表示：数组和链表

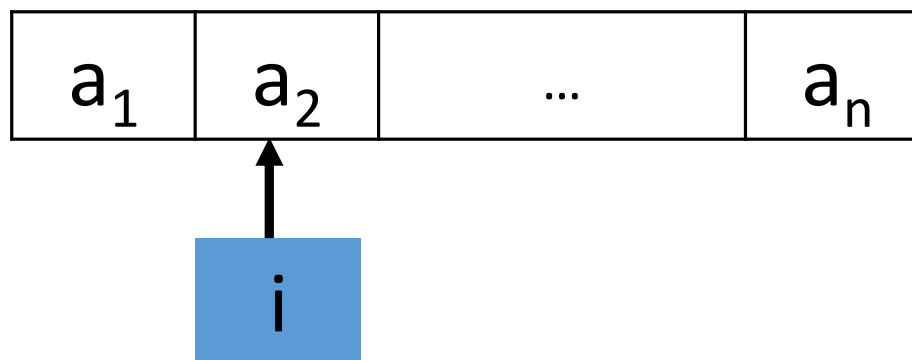


```
template <typename T>
struct LNode{
    T data;
    LNode *next;
    LNode(){next = nullptr;}
};
```


简单迭代

- 数组的迭代

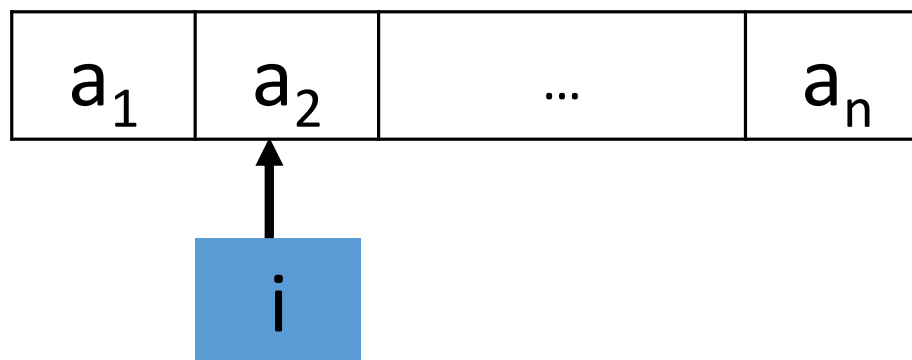
```
void traverse(vector<T> &a){  
    for (int i = 0; i < a.size(); i++) {  
        // //处理该元素: a[i]  
    }  
}
```



简单迭代

- 数组的迭代

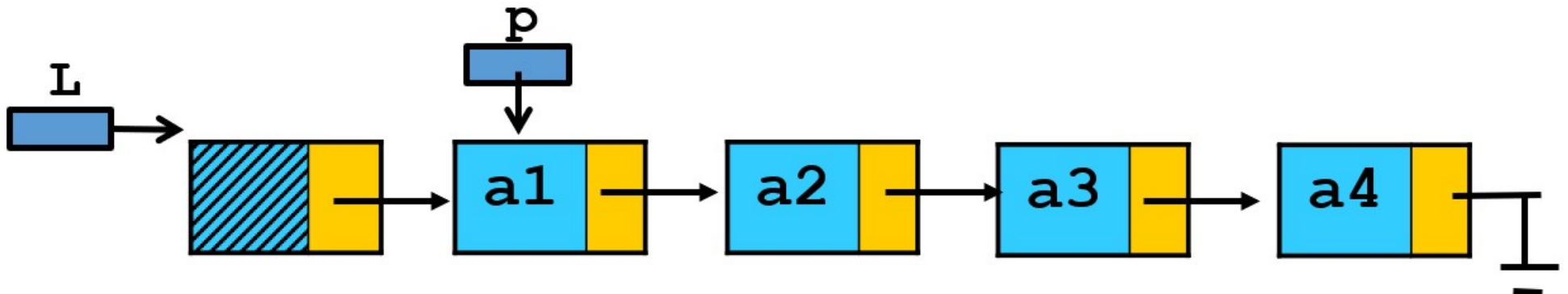
```
void traverse(vector<T> &a){  
    int i = 0;  
    while(i < a.size()) {  
        // //处理该元素: a[i]  
        i++;  
    }  
}
```



简单迭代

- 链表的迭代:

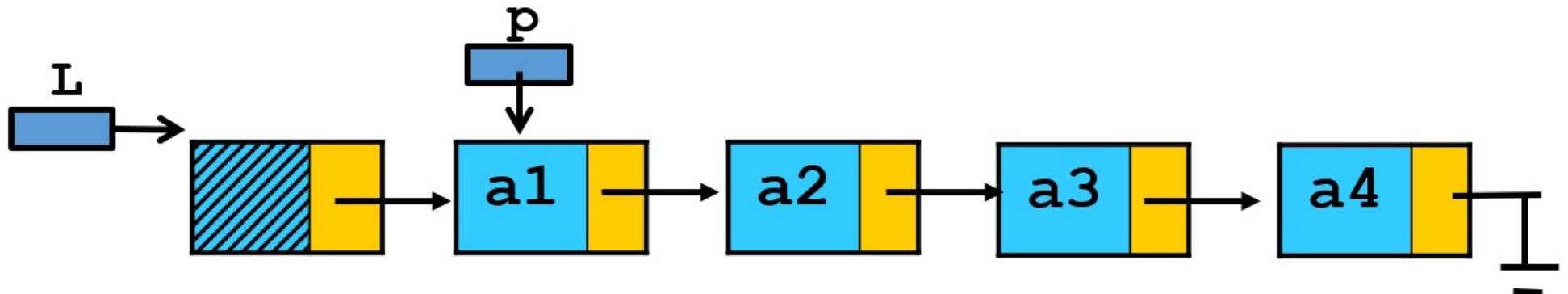
```
void traverse(LNode *L){  
    for(LNode *p = L->next;p!=null;p = p->next ){  
        //处理该结点: p->data  
    }  
}
```



简单迭代

- 链表的迭代:

```
void traverse(LNode *L){  
    LNode *p = L->next;  
    while(p!=null){  
        //处理该结点: p->data  
        p = p->next  
    }  
}
```



1. 线性查找

- 问题：在给定序列中匹配查找键的元素。

输入：[9,1,4,7,3,2,8]，查找键：3

输出：i= 5

输入：[9,1,4,7,3,2,8]，查找键：6

输出：i= -1

线性查找

- 序列中的每个元素和查找键比较，如果匹配，返回该元素位置，
- 如果都比较完，都没有匹配的，查找失败

- 如：

输入：[9,1,4,7,3,2,8]，查找键：3

输出：i= 5

输入：[9,1,4,7,3,2,8]，查找键：6

输出：i= -1

顺序查找

$(a_1, a_2, \dots, a_i, \dots, a_n)$



$i=1$

x

顺序查找

$(a_1, a_2, \dots, a_i, \dots, a_n)$



$i=2$

x

顺序查找

$(a_1, a_2, \dots, a_i, \dots, a_n)$



$i=n$

x

顺序查找

$(a_1, a_2, \dots, a_i, \dots, a_n)$

x

↑
i

```
template <typename T>
int search(vector<T> a, T x){
    for (int i = 0; i < a.size(); i++) {
        if(a[i]==x) return i;
    }
    return -1;
}
```

顺序查找

$(a_1, a_2, \dots, a_i, \dots, a_n)$

x

\uparrow
 i

```
template <typename T>
int search(vector<T> a, T x){
    for (int i = 0; i < a.size(); i++) {
        if(a[i]==x) return i;
    }
    return -1;
}
```

$x = a_1 : 1$

$x = a_2 : 2$

$x = a_i : i$

$x = a_n : n$

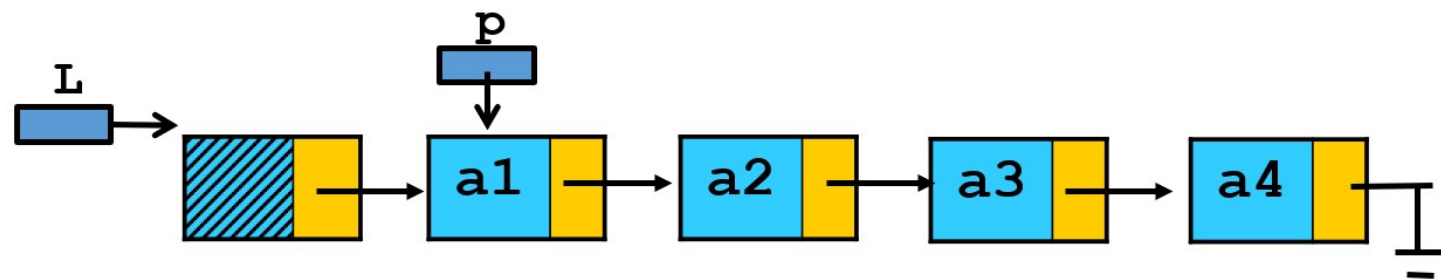
$$T(n) = \frac{1}{n}(1+2+\dots+n) = \frac{n(n+1)}{2n} = \frac{(n+1)}{2}$$

$$T(n) = \Theta(n)$$

```
template <typename T>
int search(vector<T> a, T x){
    for (int i = 0; i < a.size(); i++) {
        if(a[i]==x) return i;
    }
    return -1;
}

int main(){
    vector<int> nums{7,12,25,17,9,40};
    int ret = search(nums,25);
    cout<<"search for 25,result: "<<ret<<endl;
    cout<<"search for 19,result: "<<search(nums,19)<<endl;
}
```

链式查找



```
template <typename T>
LNode<T> * search(LNode<T> *L, T x){
    for(LNode<T> *p = L->next; p!=nullptr; p = p->next ){
        if(p->data==x)
            return p;
    }
    return nullptr;
}
```

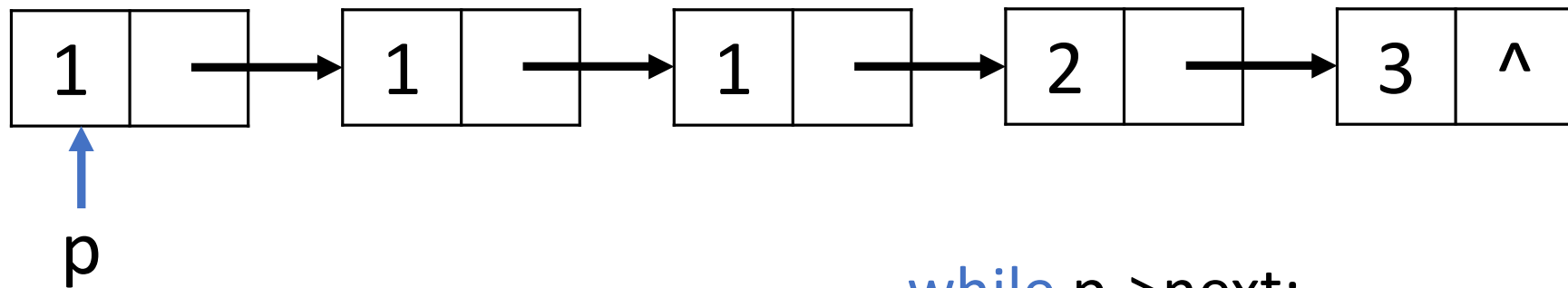
$$T(n) = \Theta(n)$$

```
int main(){
    LNode<int>* head = new LNode<int>();
    head->next = new LNode<int>();
    LNode<int>* p = head;
    int num;
    while(cin>>num){
        p->next = new LNode<int>();
        p = p->next;
        p->data = 7;
    }

    p = search(head,25);
    cout<<"search for 25,result: "<<p<<endl;
    p = search(head,19);
    cout<<"search for 19,result: "<<p<<endl;
}
```

2. 删除有序链表中的重复元素

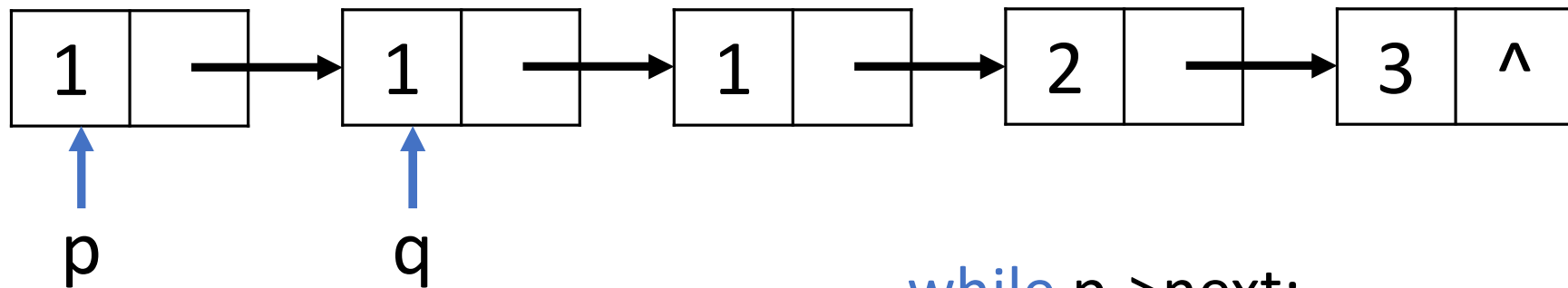
- 迭代检测每个结点是否和后一个结点值相同，如果相同，则删除后一个结点。



```
while p->next:  
    if p->val==p->next->val:
```

2. 删除有序链表中的重复元素

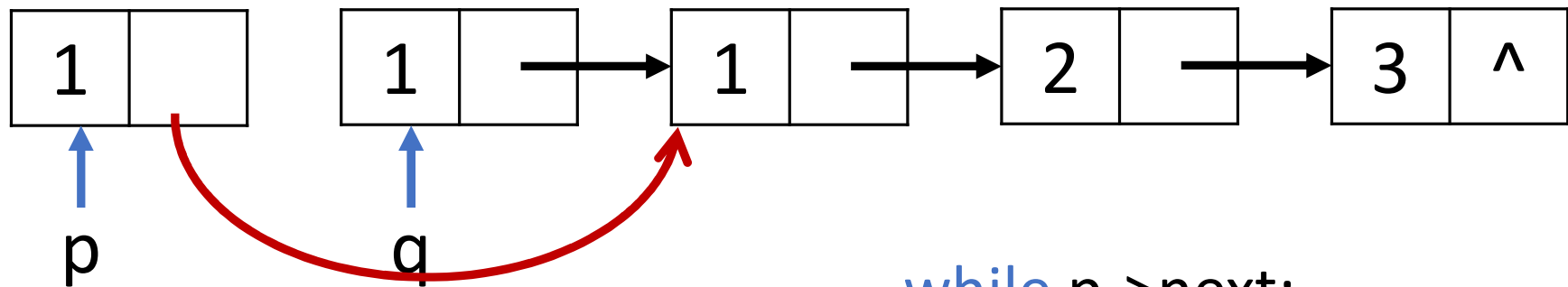
- 迭代检测每个结点是否和后一个结点值相同，如果相同，则删除后一个结点。



```
while p->next:  
    if p->val==p->next->val:  
        q= p->next
```


2. 删除有序链表中的重复元素

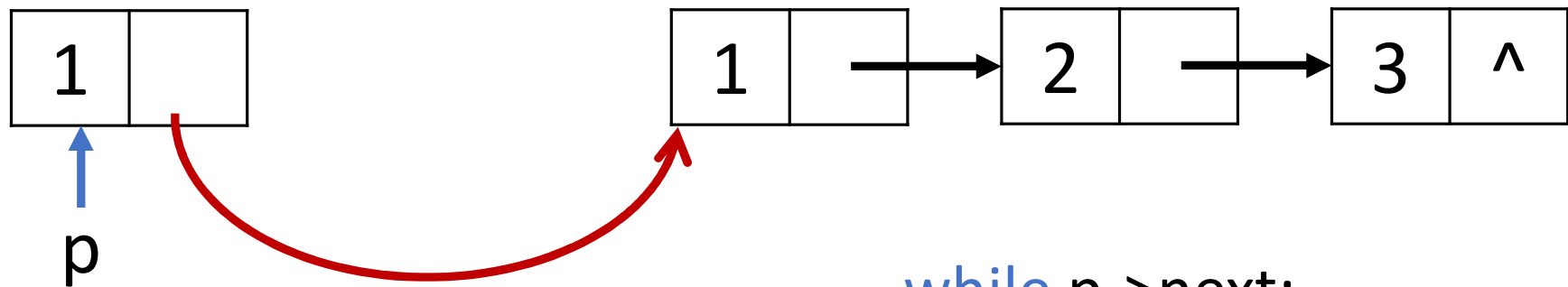
- 迭代检测每个结点是否和后一个结点值相同，如果相同，则删除后一个结点。



```
while p->next:  
    if p->val==p->next->val:  
        q= p->next  
        p->next = q->next
```

2. 删除有序链表中的重复元素

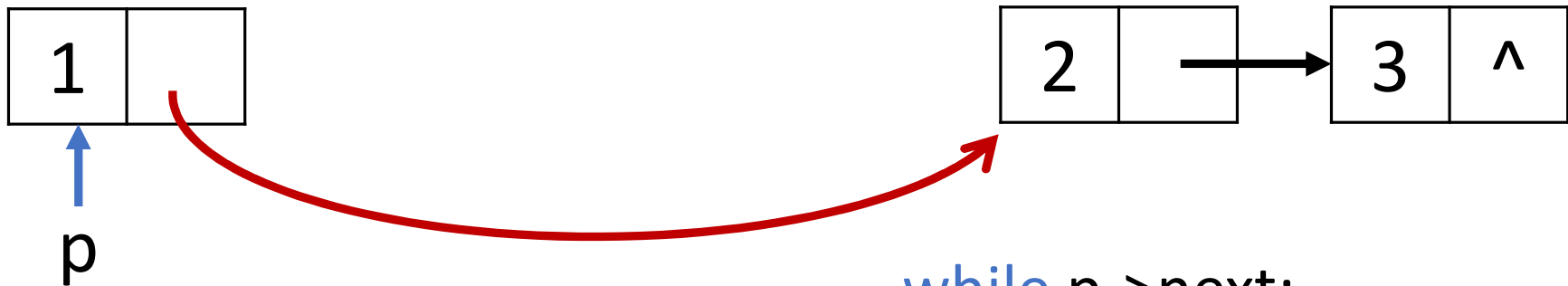
- 迭代检测每个结点是否和后一个结点值相同，如果相同，则删除后一个结点。



```
while p->next:  
    if p->val==p->next->val:  
        q= p->next  
        p->next = q->next  
        delete q
```

2. 删除有序链表中的重复元素

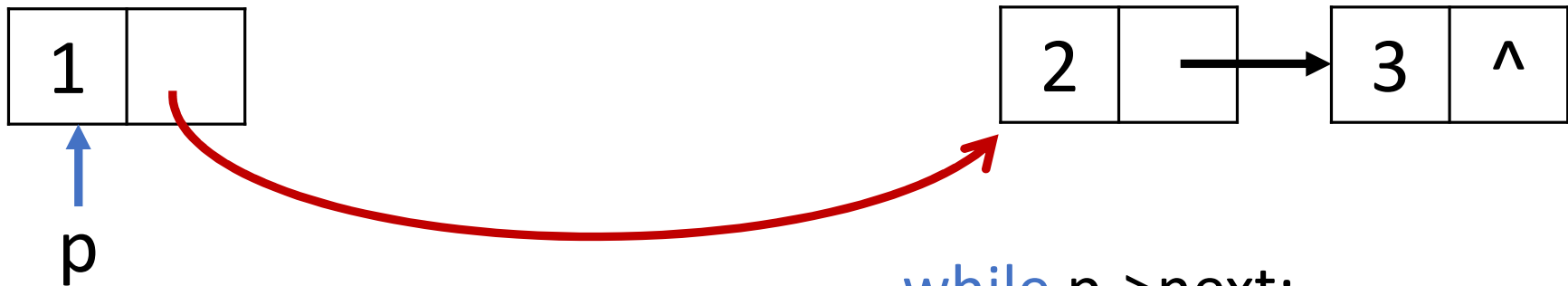
- 迭代检测每个结点是否和后一个结点值相同，如果相同，则删除后一个结点。



```
while p->next:  
    if p->val==p->next->val:  
        q= p->next  
        p->next = q->next  
        delete q
```

2. 删除有序链表中的重复元素

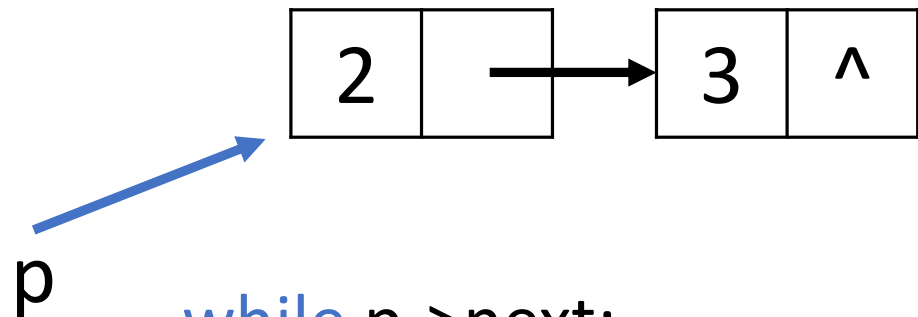
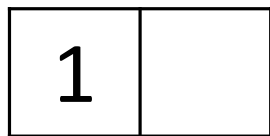
- 迭代检测每个结点是否和后一个结点值相同，如果相同，则删除后一个结点。



```
while p->next:  
    if p->val==p->next->val:  
        q= p->next  
        p->next = q->next  
        delete q  
    else p = p->next
```

2. 删除有序链表中的重复元素

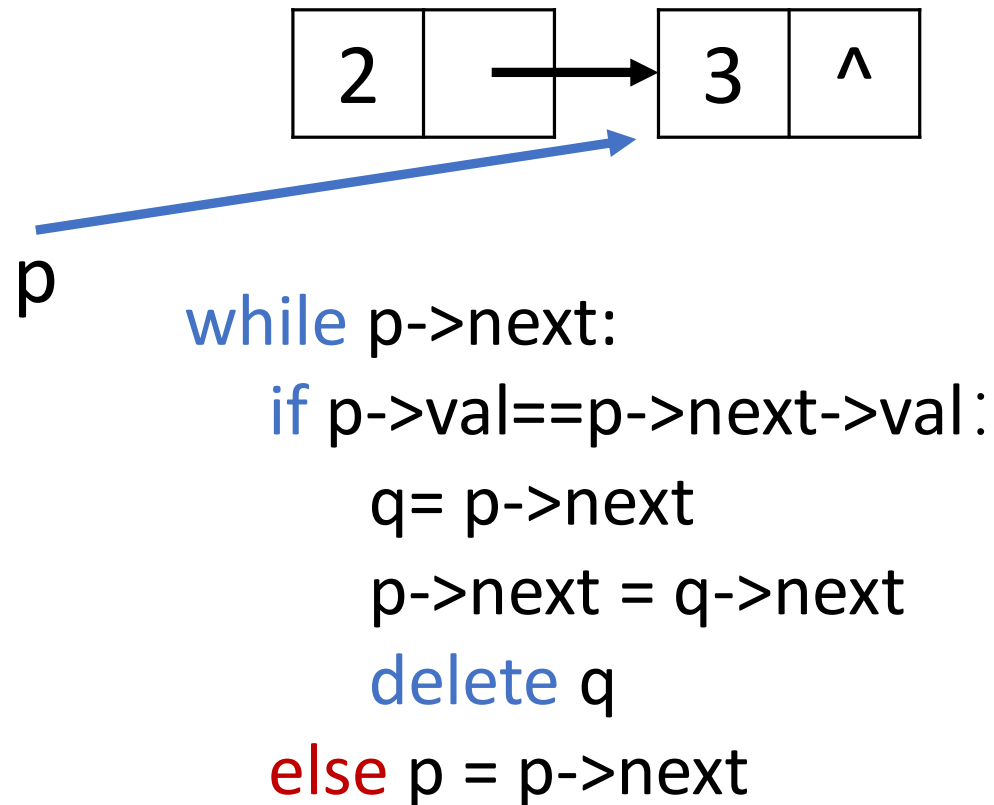
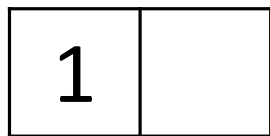
- 迭代检测每个结点是否和后一个结点值相同，如果相同，则删除后一个结点。



```
while p->next:  
    if p->val==p->next->val:  
        q= p->next  
        p->next = q->next  
        delete q  
    else p = p->next
```

2. 删除有序链表中的重复元素

- 迭代检测每个结点是否和后一个结点值相同，如果相同，则删除后一个结点。



```
ListNode* deleteDuplicates(ListNode* head) {  
    // 如果元素少于2个则直接返回  
    if (head == nullptr || head->next == nullptr) return head;  
    ListNode* p = head,*q;  
    while (p->next)  
        if (p->val==p->next->val){  
            q= p->next;  
            p->next = q->next;  
            delete q;  
        }  
        else p = p->next;  
    return head;  
}
```

```
class ListNode{
public:
    int val;
    ListNode *next;
    ListNode(int val=0){this->val = val; next = nullptr;}
};
```

```
void printList(ListNode *head){
    for(ListNode *p = head; p ;p = p->next)
        cout<<p->val<<" ";
    cout<<endl;
}
```



```
int main(){
    ListNode *head = new ListNode(1), *p = head;
    p->next = new ListNode(1); p = p->next;
    p->next = new ListNode(1); p = p->next;
    p->next = new ListNode(2); p = p->next;
    p->next = new ListNode(3);
    printList(head);
    deleteDuplicates(head);
    printList(head);
}
```

二、嵌套迭代

博客: hwdong-net.github.io

Youtube频道:hwdong

嵌套迭代

- 在迭代里嵌套迭代，称为**嵌套迭代**（多层迭代）。
- 例如，对于二维数组，可以一行一行遍历。

```
for i = 1 to n:  
    for j = 1 to n:  
        print(A[i][j]);
```

2. Two sum (leetcode 1)

- 问题：给定一个整数数组，返回两个数字的索引，使其相加等于一个特定的目标target。假设只有一对解。

Input: `nums = [2,7,11,15], target = 9`

Output: `[0,1]`

Output: Because `nums[0] + nums[1] == 9`, we return `[0, 1]`

Input: `nums = [3,2,4], target = 6`

Output: `[1,2]`

Input: `nums = [3,3], target = 6`

Output: `[0,1]`

问：

- 题目里整数数组可以改成浮点数数组吗？

嵌套的线性迭代

- 穷举所有可能的 (i, j) 对, 检查 $a_i + a_j = \text{target}$?

$(a_1, \dots, a_i, \dots, a_j, \dots, a_n)$

↑ ↑
i j

$$T(n) = \Theta(n^2)$$

```
for (int i = 0; i < a.size(); ++i)
    for (int j = i+1; j < a.size(); ++j)
        if(a[i]+a[j]==target)
            return make_pair(i,j);

return make_pair(-1,-1);
```

问题变换： $(a_1, \dots, a_i, \dots, a_j, \dots, a_n)$

- 遍历序列的每个元素 a_i 。检查序列中是否存在等于 $\text{target} - a_i$ 的元素。
- 用hash表存储所有元素值到坐标的映射。

`HashMap<int, int> m;`

- 遍历数组的每个元素 a_i 。看看m中存在 $\text{target} - a_i$ 的key

转化为hash表的查找，空间换时间 $O(n)$

```

template <typename T>
std::pair<int,int> twoSum(vector<T>& a, T target) {
    unordered_map<T, int> m;
    for (int i = 0; i < a.size(); ++i)
        m[a[i]] = i;

    for (int i = 0; i < a.size(); ++i) {
        T t = target - a[i];
        if (m.count(t) && m[t] != i)
            return make_pair(i,m[t]);
    }
    return make_pair(-1,-1);
}

```

$$T(n) = \Theta(n)$$


```
#include <iostream>
#include <vector>
using namespace std;
#include <tr1/unordered_map>
using namespace std::tr1;
```

```
int main(){
    vector<int> nums{7,12,25,17,9,40};
    std::pair<int,int> p = twoSum(nums,37);
    cout<<"result for target 37: "<<p.first<<","<<p.second<<endl;
    p = twoSum(nums,40);
    cout<<"result for target 40: "<<p.first<<","<<p.second<<endl;
}
```

3.百钱买百鸡

- 公元5世纪数学家张邱建的《张邱建算经》
- “今有鸡翁一值钱五，鸡母一值钱三，鸡雏三值钱一，凡百钱买鸡百只，问鸡翁、母、雏各几何？”
- 公鸡5文钱一只，母鸡3文钱一只，小鸡3只一文钱，用100文钱买一百只鸡，问公鸡，母鸡，小鸡要买多少只刚好凑足100文钱？

- 设公鸡为 x , 母鸡为 y , 小鸡为 z , 可以得出下列方程:
 $x + y + z = 100;$
 $5x + 3y + z/3 = 100;$

buyChicken (int money, int n):

for $x=0$ to $n/5$:

for $y=0$ to $n/3$:

for $z=0$ to n :

if $z\%3==0$ and $x+y+z=n$ and $5*x+3*y+z/3==n$:

print("公鸡、母鸡、小鸡数量为: ")

print(x, y, z)

- 设公鸡为x, 母鸡为y, 小鸡为z, 可以得出下列方程:
 $x + y + z = 100;$
 $5x + 3y + z/3 = 100;$

buyChicken (int money, int n):

for x=0 to n/5:

for y=0 to n/3:

if $5*x+3*y > \text{money}$: break;

$z = n - x - y;$

if $z < 0$: break;

if $z \% 3 == 0$ and $5*x+3*y+z/3 == n$:

print("公鸡、母鸡、小鸡数量为: ")

print(x, y, z)

```

#include <iostream>
using namespace std;
void buyChicken(int money, int n) {
    int n_5 = n/5, n_3 = n/3;
    for (int x = 0; x <= n_5; x++) {
        for (int y = 0; y <= n_3; y++) {
            if (5 * x + 3 * y > money) break;
            auto z = n - x - y;
            if (z < 0) break;
            if (z % 3 == 0 and 5 * x + 3 * y + z / 3 == money) {
                cout << "公鸡、母鸡、小鸡数量为: " << endl;
                cout << x << "\t" << y << "\t" << z << "\n";
            }
        }
    }
}

int main() {
    buyChicken(100, 100); //(4,18,78), (8,11,81), (12,4,84)
}

```

$$T(n) = \Theta(n^3)$$

公鸡、母鸡、小鸡数量为:

4 18 78

公鸡、母鸡、小鸡数量为:

8 11 81

公鸡、母鸡、小鸡数量为:

12 4 84

4. 暴力破解密码

有一个三位数的密码锁，如何打开它？不能用锤子等暴力方法砸开。

```
for i=0 to 9:  
  for j=0 to 9:  
    for k=0 to 9:  
      if ijk==password:  
        print("破解了密码！")
```

```
void brute_force_attck(int* password) {  
    for (auto i = 0; i <= 9; i++)  
        for (auto j = 0; j <= 9; j++)  
            for (auto k = 0; k <= 9; k++)  
                if (i == password[0] and j == password[1]  
                    and k == password[2]) {  
                    cout << "破解了密码:" <<i<<j<<k<< endl;  
                    return ;  
                }  
}
```

$$T(n) = 9^3$$

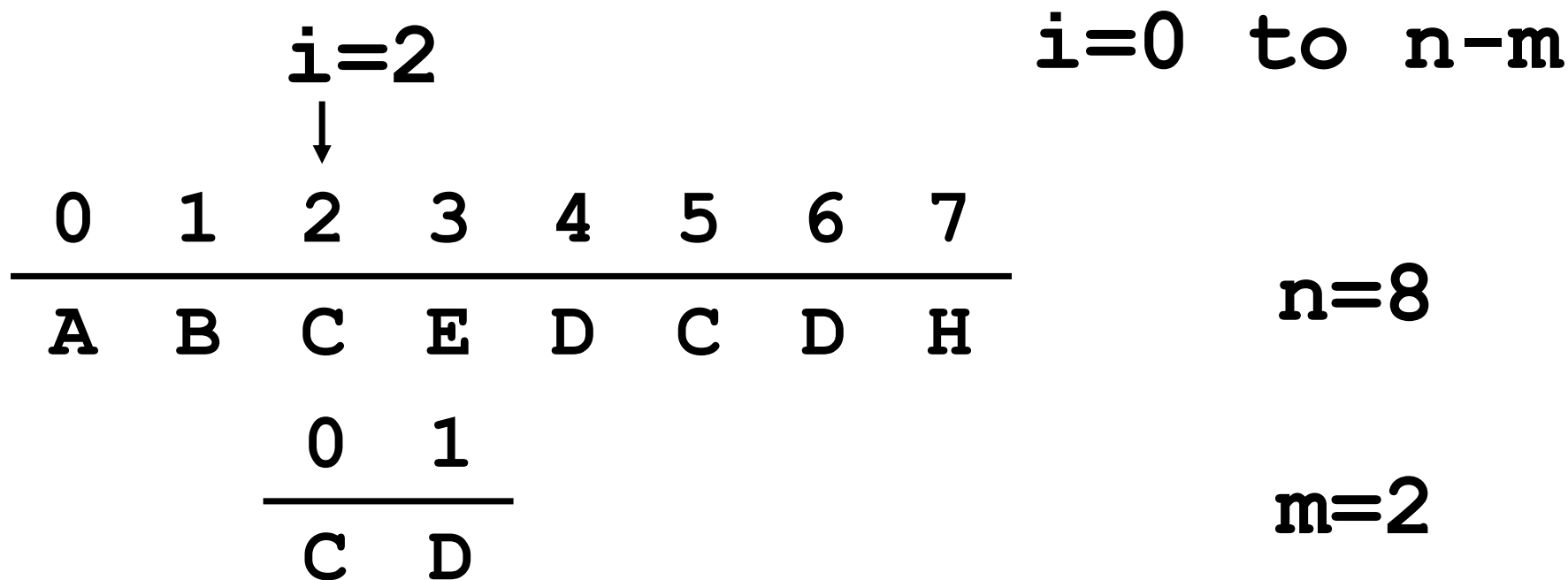

```
#include <ctime>
#include <cstdlib>
int main() {
    int password[3];
    srand(time(NULL));
    for(auto i = 0 ; i<3;i++)
        password[i] = rand() % 10;
    cout <<"生成随机密码: "<<
        password[0] << password[1] << password[2] << endl;
    brute_force_attck(password);
}
```

5. 简单的字符串匹配

- 问题：在一个字符串中查找是否有等于另外一个字符串的子串。
- 被查找的串称为**主串**，待查找的串称为**模式串**。
- 如：
- 主串：ABCEDCDH
- 模式串:CD

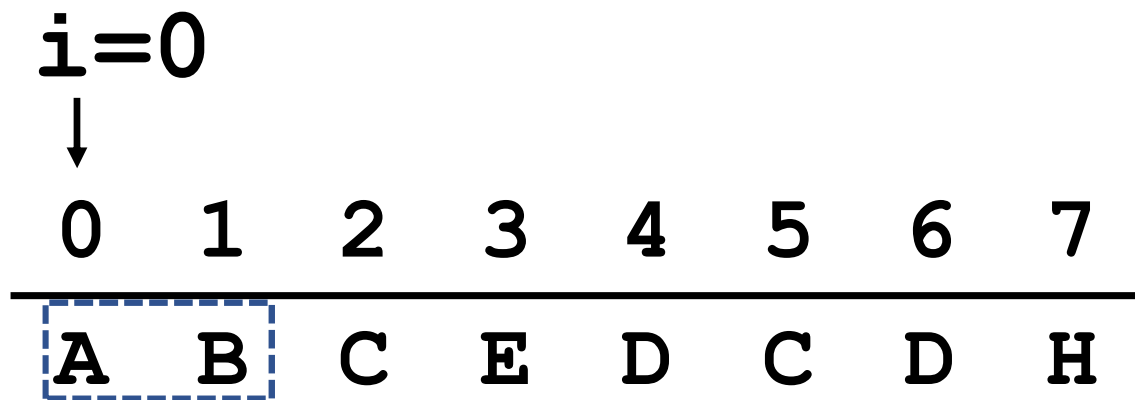
简单模式匹配

- 从主串开始位置，对每个和模式串同样长度的子串，简单该子串和模式串是否匹配



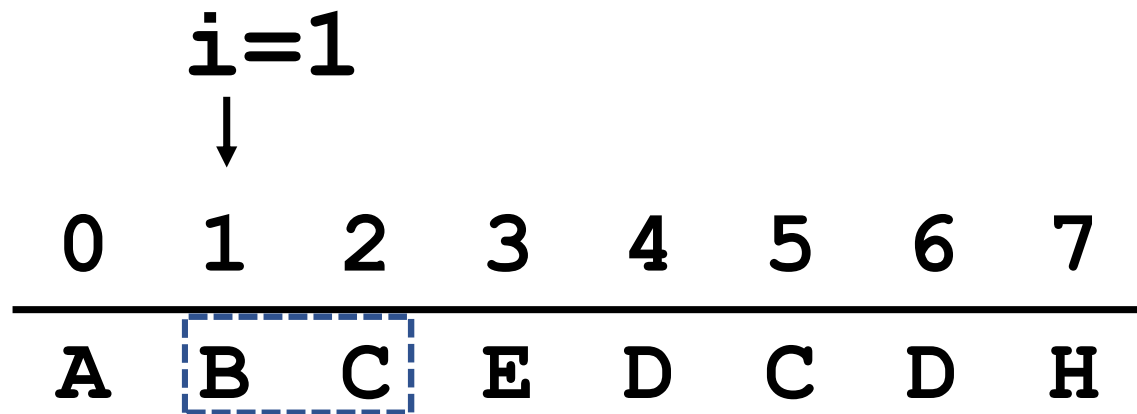
bruteForceStringMatch($T[0..n-1]$, $P[0..m-1]$)

for $i \leftarrow 0$ to $n-m$:



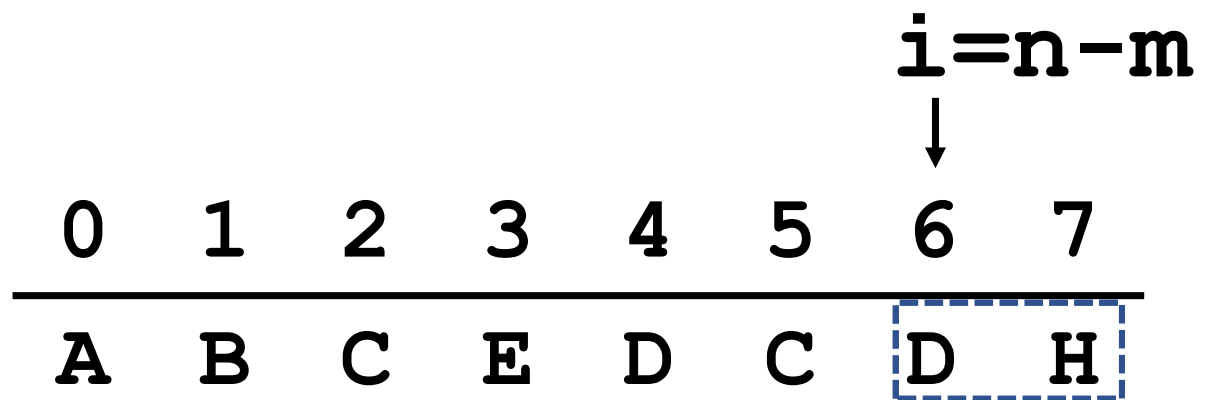
`bruteForceStringMatch(T[0..n-1], P[0..m-1])`

for $i \leftarrow 0$ to $n-m$:



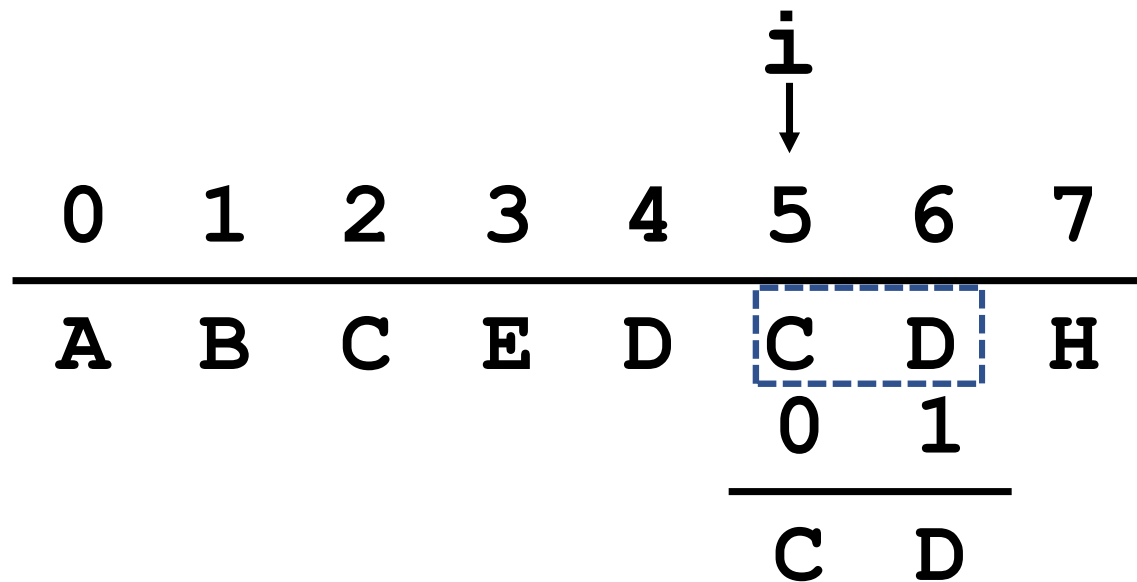
bruteForceStringMatch($T[0..n-1]$, $P[0..m-1]$)

for $i \leftarrow 0$ to $n-m$:



bruteForceStringMatch(T[0..n-1], P[0..m-1])

for $i \leftarrow 0$ to $n-m$:



bruteForceStringMatch($T[0..n-1]$, $P[0..m-1]$)

for $i \leftarrow 0$ to $n-m$ do

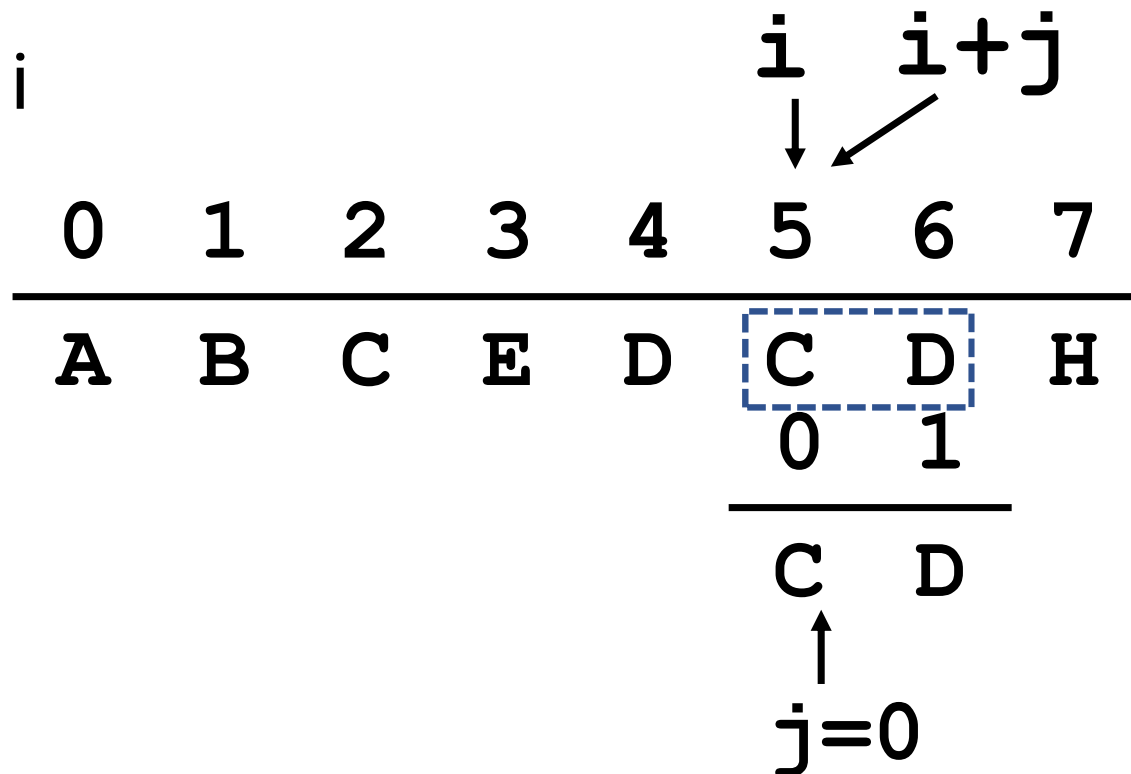
$j \leftarrow 0$

 while $j < m$ and $P[j] = T[i+j]$

$j \leftarrow j+1$

 if $j = m$ return i

return -1



bruteForceStringMatch($T[0..n-1]$, $P[0..m-1]$)

for $i \leftarrow 0$ to $n-m$ do

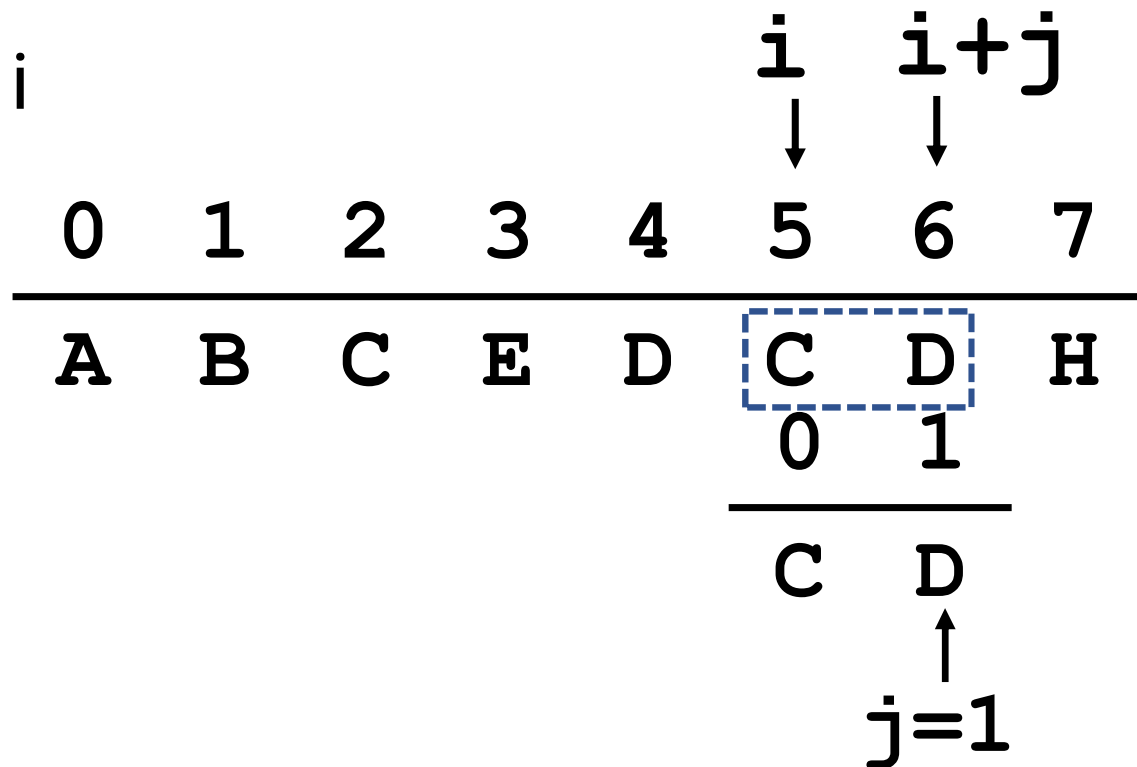
$j \leftarrow 0$

 while $j < m$ and $P[j] = T[i+j]$

$j \leftarrow j+1$

 if $j = m$ return i

return -1



bruteForceStringMatch($T[0..n-1]$, $P[0..m-1]$)

for $i \leftarrow 0$ to $n-m$ do

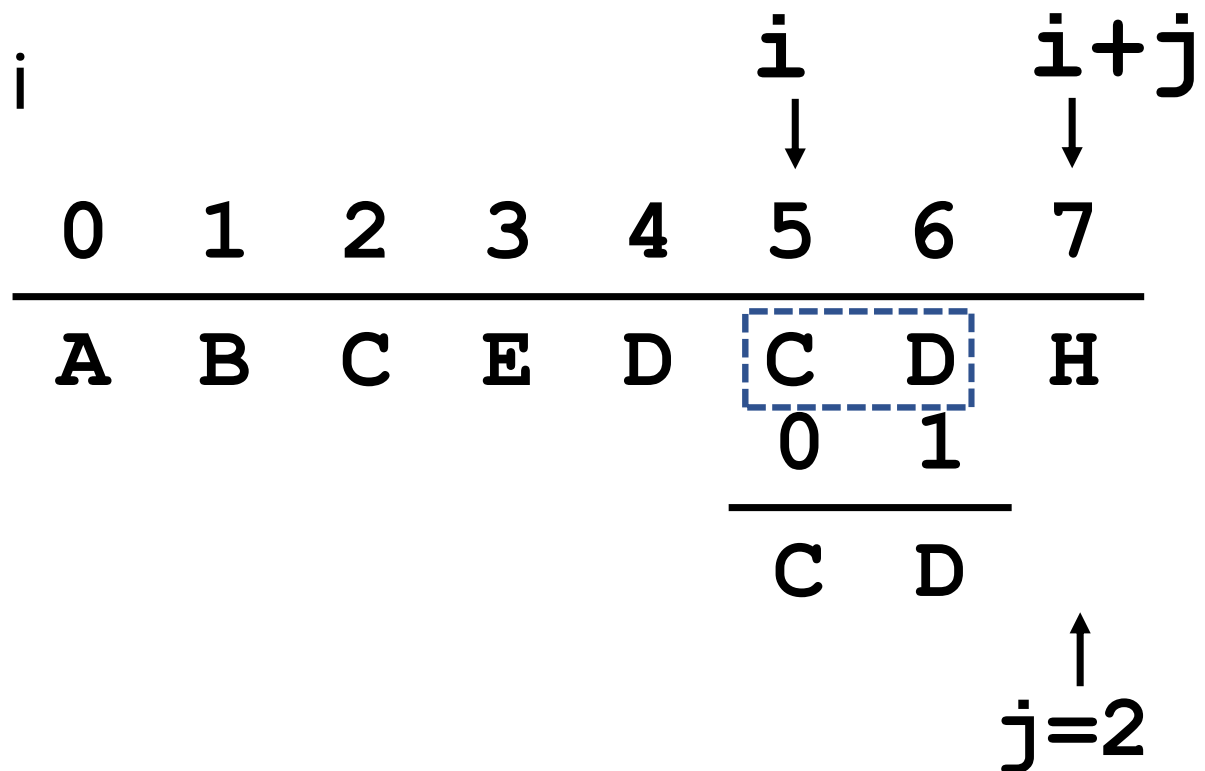
$j \leftarrow 0$

 while $j < m$ and $P[j] = T[i+j]$

$j \leftarrow j+1$

 if $j = m$ return i

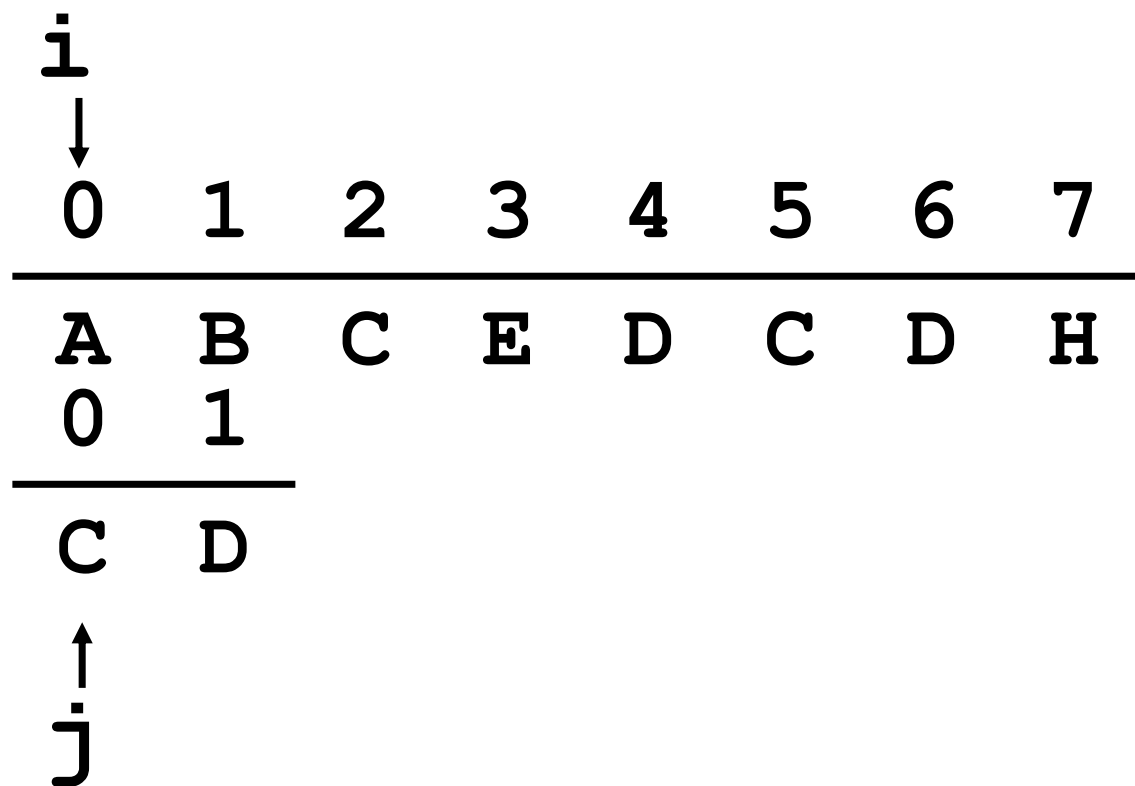
return -1



```
int bruteforce_stringmatch(const string &T,  
    const string &P) {  
    int n = T.size(), m = P.size(), n_m = n - m;  
    for (auto i = 0; i < n_m; i++) {  
        auto j{ 0 };  
        while (j < m && T[i + j] == P[j])  
            j++;  
        if (j == m) return i;  
    }  
    return -1;  
}
```

简单匹配算法

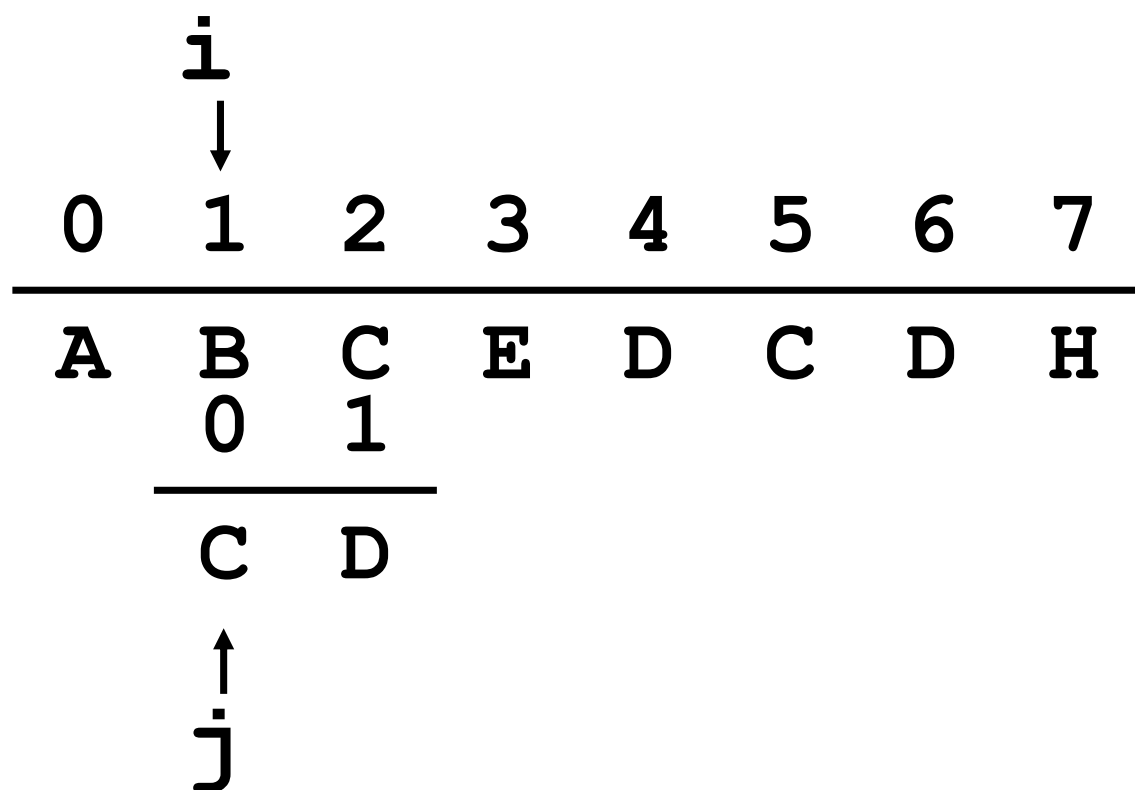
- 避免 $i+j$, 可移动 i : $i++$



$j=0$
 $i=i-j+1$

简单匹配算法

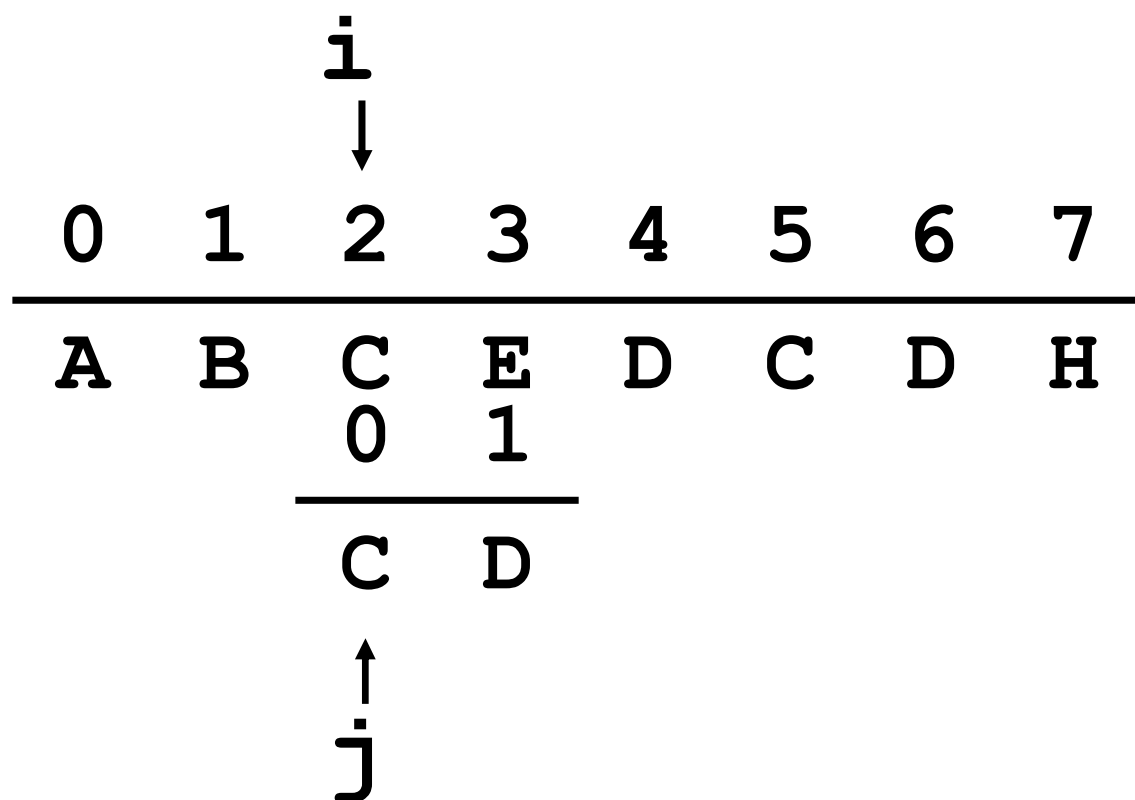
- 模式串对准主串的开始位置，依次比较模式串每个字符和主串的字符是否不匹配。
- 如果匹配，则返回匹配位置，如果不匹配，移动模式串对准主串的下一个位置



$$j=0$$
$$i=i-j+1$$

简单匹配算法

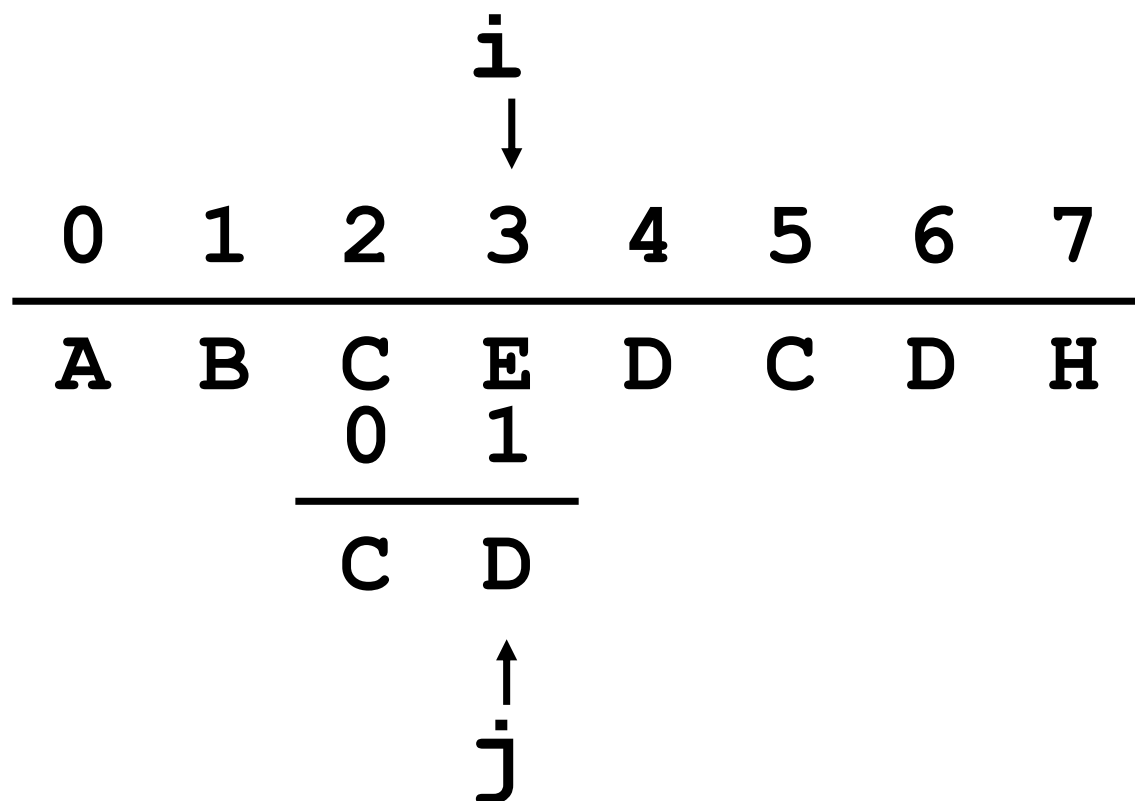
- 模式串对准主串的开始位置，依次比较模式串每个字符和主串的字符是否不匹配。
- 如果匹配，则返回匹配位置，如果不匹配，移动模式串对准主串的下一个位置



$j=j+1$
 $i=i+1$

简单匹配算法

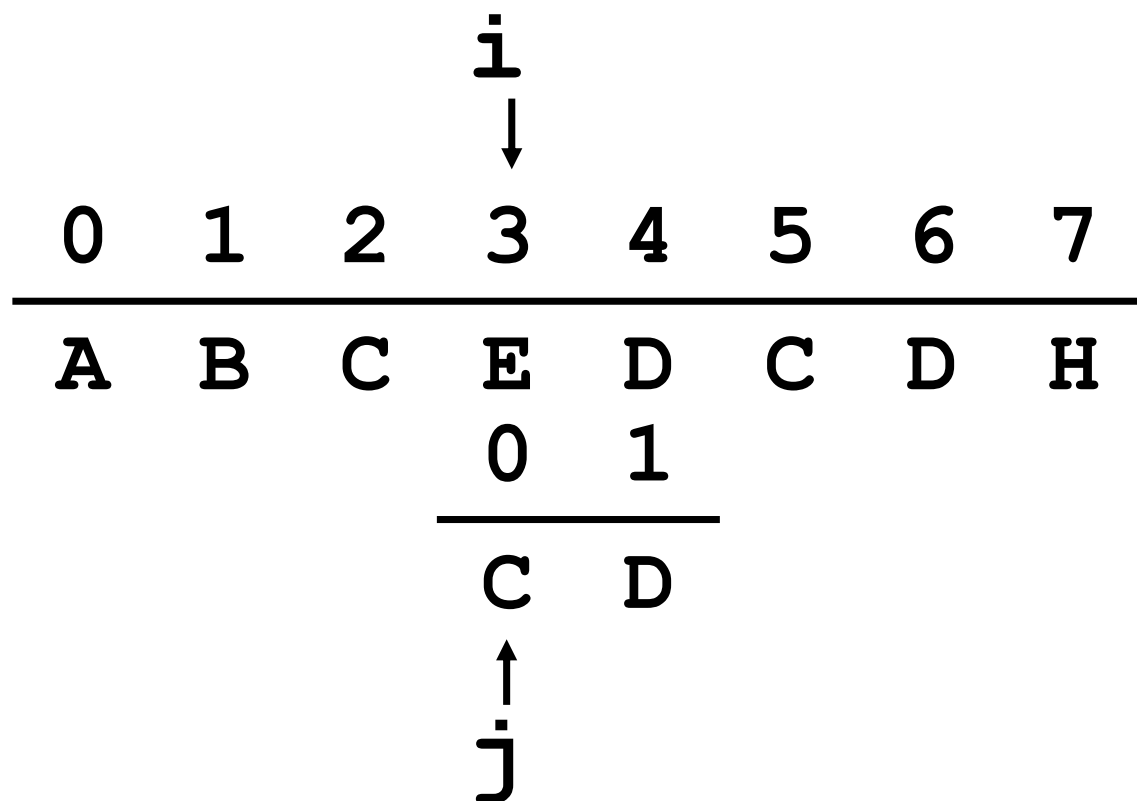
- 模式串对准主串的开始位置，依次比较模式串每个字符和主串的字符是否不匹配。
- 如果匹配，则返回匹配位置，如果不匹配，移动模式串对准主串的下一个位置



$$j=0$$
$$i=i-j+1$$

简单匹配算法

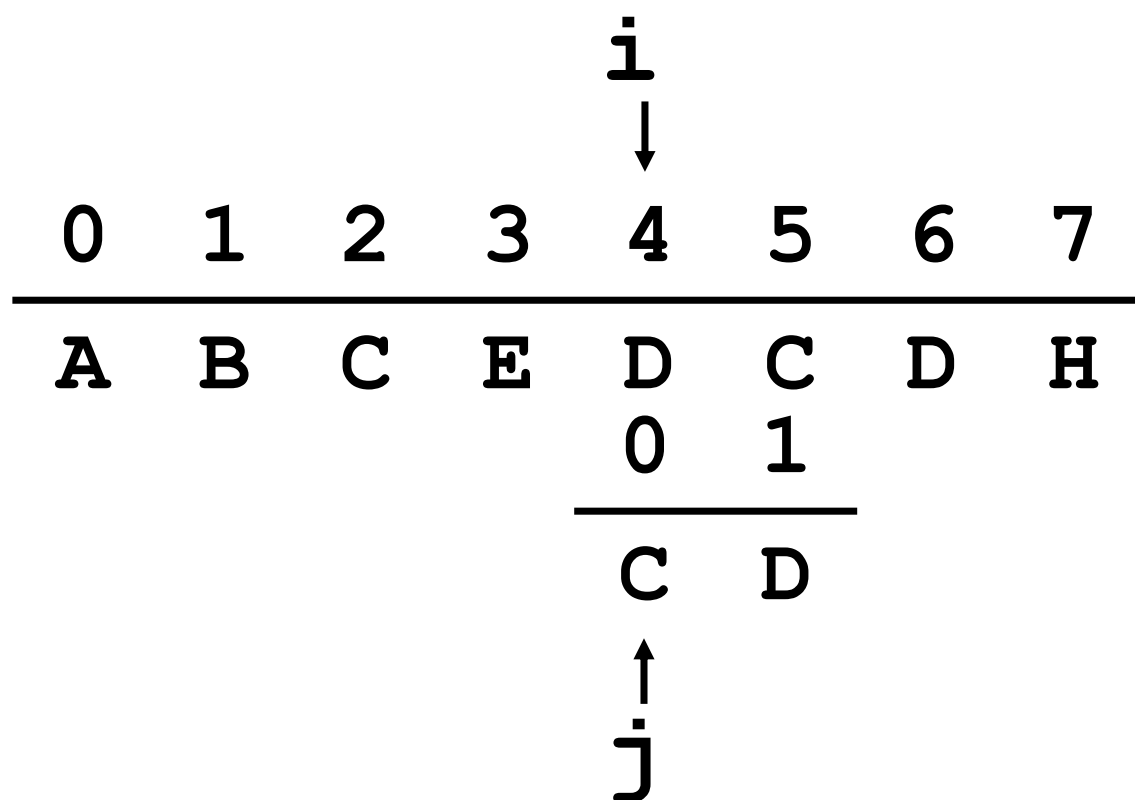
- 模式串对准主串的开始位置，依次比较模式串每个字符和主串的字符是否不匹配。
- 如果匹配，则返回匹配位置，如果不匹配，移动模式串对准主串的下一个位置



$$j=0$$
$$i=i-j+1$$

简单匹配算法

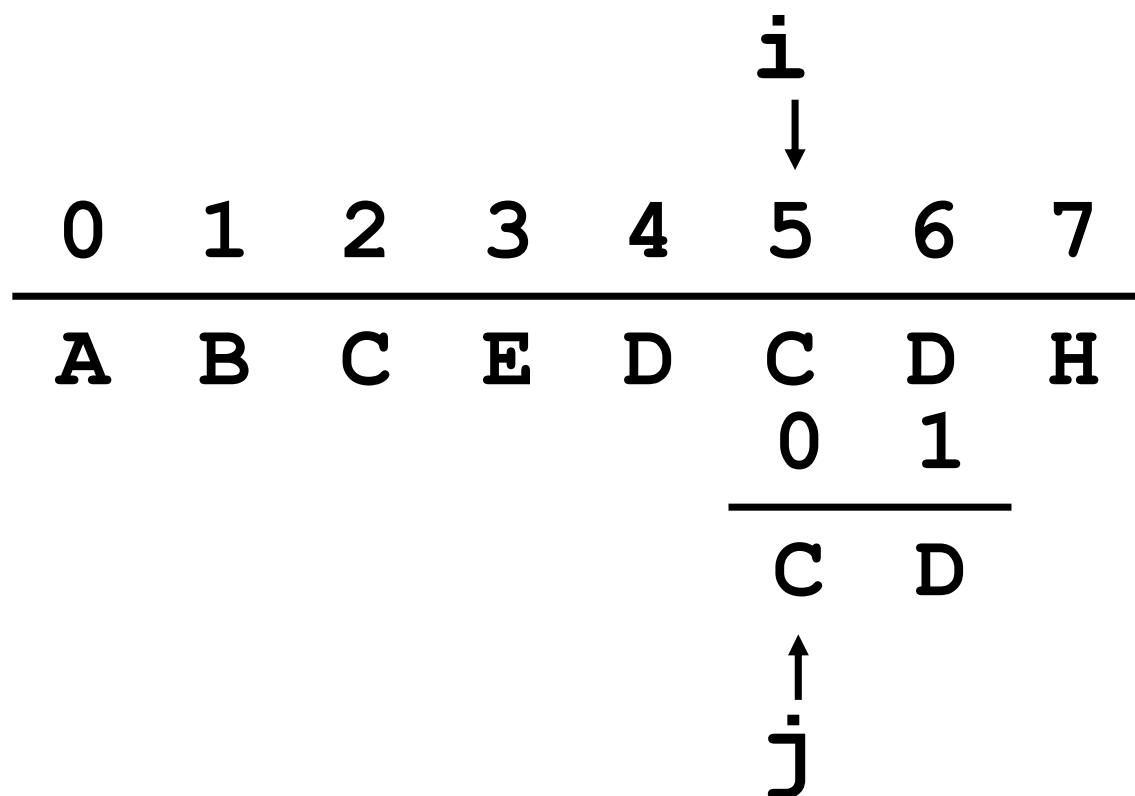
- 模式串对准主串的开始位置，依次比较模式串每个字符和主串的字符是否不匹配。
- 如果匹配，则返回匹配位置，如果不匹配，移动模式串对准主串的下一个位置



$$j=0$$
$$i=i-j+1$$

简单匹配算法

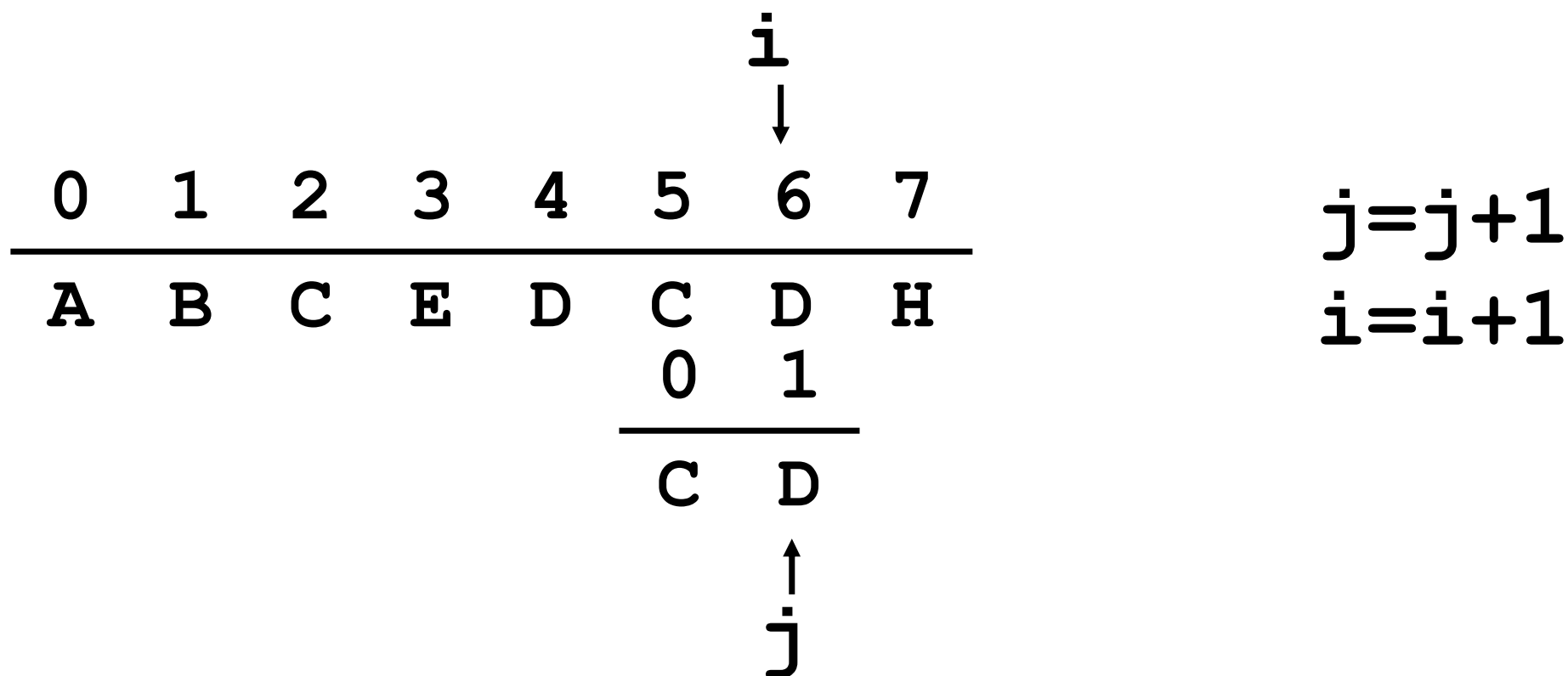
- 模式串对准主串的开始位置，依次比较模式串每个字符和主串的字符是否不匹配。
- 如果匹配，则返回匹配位置，如果不匹配，移动模式串对准主串的下一个位置



j=j+1
i=i+1

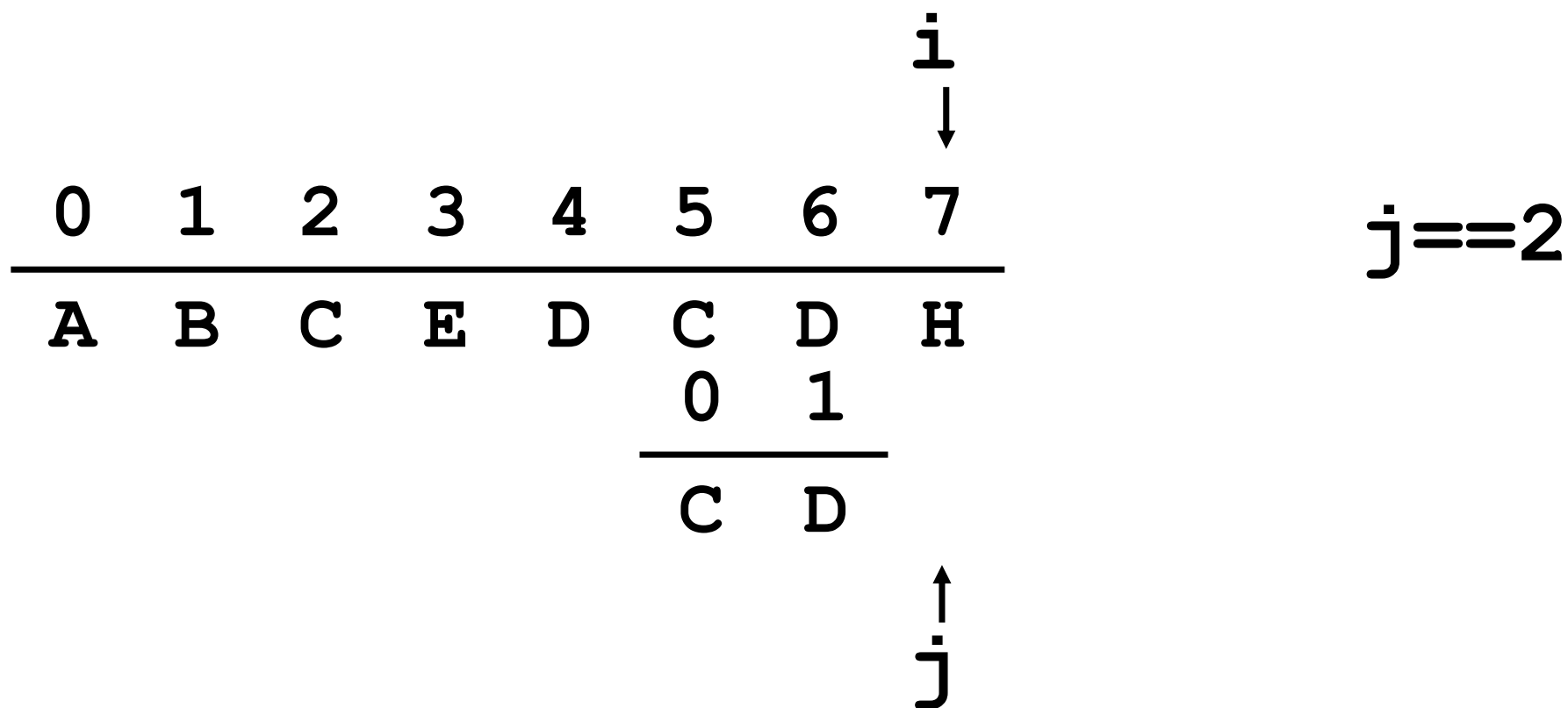
简单匹配算法

- 模式串对准主串的开始位置，依次比较模式串每个字符和主串的字符是否不匹配。
- 如果匹配，则返回匹配位置，如果不匹配，移动模式串对准主串的下一个位置



简单匹配算法

- 模式串对准主串的开始位置，依次比较模式串每个字符和主串的字符是否不匹配。
- 如果匹配，则返回匹配位置，如果不匹配，移动模式串对准主串的下一个位置



bruteForceStringMatch(T[0..n-1],P[0..m-1])

i = 0, j = 0;

while i < n and j < m

if T[i] == P[j] : //当前字符匹配, ij递增

i=i+1 j=j+1;

else: //否则i回退, j返回模式串首, 重新开始

i = i - j + 1; j = 0;

if j >= m: return i-j; //匹配成功

else return -1; //失败

0	1	2	3	4	5	6	7
A	B	C	E	D	C	D	H
					0	1	
					C	D	

j=1

j=j+1
i=i+1

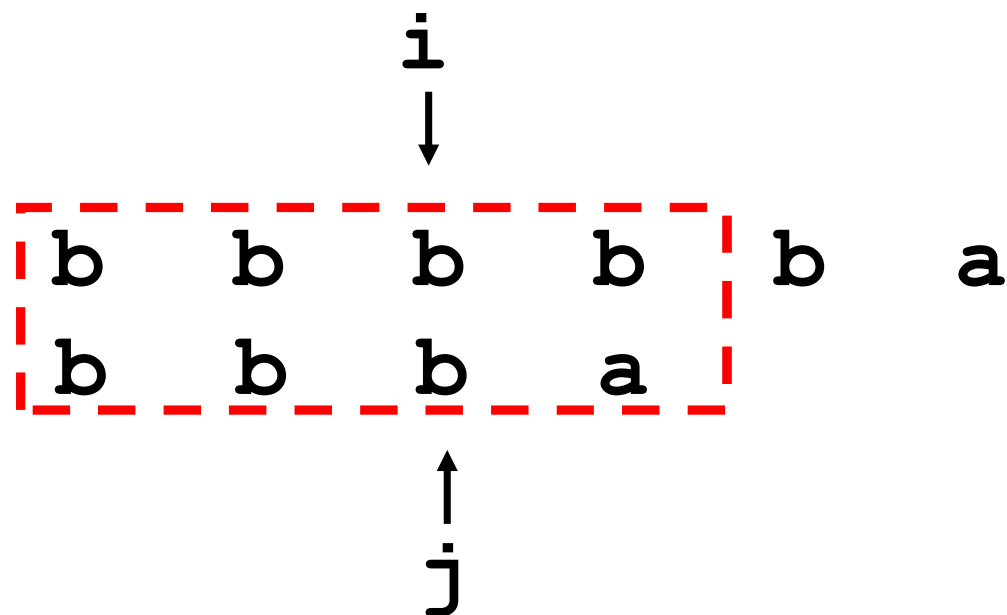
```
int bruteforce_stringmatch2(const string& T,  
    const string& P) {  
    int n = T.size(), m = P.size(), n_m = n - m;  
    int i = 0, j = 0;  
    while(i < n && j < m) {  
        if (T[i] == P[j]) {  
            i++; j++;  
        }  
        else {  
            i = i - j + 1;  
            j = 0;  
        }  
    }  
    if (j == m) return i - j;  
    return -1;  
}
```

时间复杂度

- 最差的情况

- 主串T中的每个字符，分别和模式P中的每个字符匹配一次

- 复杂度 = $O(n * m)$

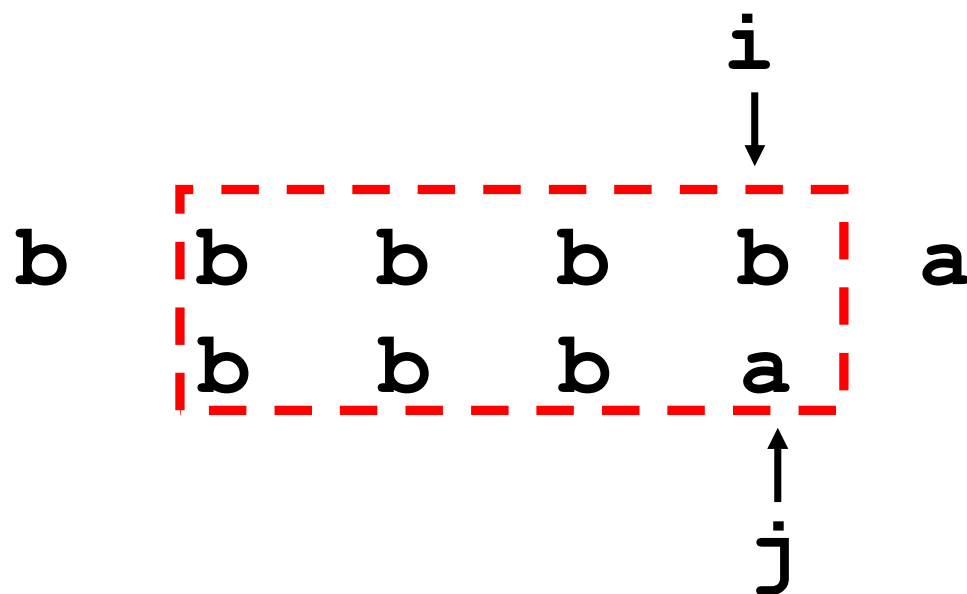


时间复杂度

- 最差的情况

- 主串T中的每个字符，分别和模式P中的每个字符匹配一次

- 复杂度 = $O(n * m)$



6. 背包问题

- n 个物品的重量 w_1 、 w_2 、...、 w_n ，价值为 v_1 、 v_2 、...、 v_n ，背包限重为 W 。问：如何装使价值最大？

item	weight	value
1	1	1
2	2	5
3	5	18
4	6	22
5	7	28

$W = 11$

按性价比 w_2 / v_2 排序

- 每个物品有装和不装2种可能行，所有可能的组合一共有 2^n .

0	0	0	0	0
---	---	---	---	---

0

0	0	0	0	1
---	---	---	---	---

1

0	0	0	1	0
---	---	---	---	---

2

0	0	0	1	1
---	---	---	---	---

3

0	0	1	0	0
---	---	---	---	---

4

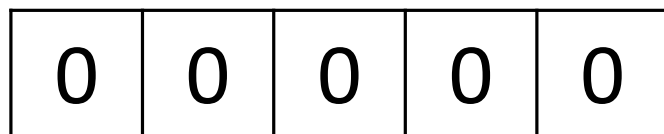
0	0	1	0	1
---	---	---	---	---

5

0	0	1	1	1
---	---	---	---	---

6

- 依次产生1到 2^n 的二进制表示.



0



j=n

- 依次产生1到 2^n 的二进制表示.

0	0	0	0	1
---	---	---	---	---

1



j=n

- 依次产生1到 2^n 的二进制表示.

0	0	0	0	0
---	---	---	---	---

1



$j = j-1$

- 依次产生1到 2^n 的二进制表示.

0	0	0	1	0
---	---	---	---	---

2

j

- 依次产生1到 2^n 的二进制表示.

0	0	0	1	0
---	---	---	---	---

2



$j=n$

- 依次产生1到 2^n 的二进制表示.

0	0	0	1	1
---	---	---	---	---

3



$j=n$

- 依次产生1到 2^n 的二进制表示.

0	0	0	1	0
---	---	---	---	---

3



$j=n$

- 依次产生1到 2^n 的二进制表示.

0	0	0	1	0
---	---	---	---	---

3



$j=j-1$

- 依次产生1到 2^n 的二进制表示.

0	0	0	0	0
---	---	---	---	---

3



$j=j-1$

- 依次产生1到 2^n 的二进制表示.

0	0	1	0	0
---	---	---	---	---

4


j

0	0	0	1	1
---	---	---	---	---


3


 $j=n$

0	0	0	1	0
---	---	---	---	---


 $j=j-1$

0	0	0	0	0
---	---	---	---	---


 $j=j-1$

0	0	1	0	0
---	---	---	---	---

4


 j

for $i=1$ to 2^n :

$j \leftarrow n$

 while $A[j] \neq 0$ and $j > 0$:

$A[j] \leftarrow 0 ; j \leftarrow j-1$

$A[j] \leftarrow 1$

Knapsack_BruteForce(W, w[], v[], n):

$A[] \leftarrow 0$ //n位0

for i=1 to 2^n :

weight $\leftarrow 0$; value $\leftarrow 0$

j \leftarrow n

while $A[j] \neq 0$ and $j > 0$:

$A[j] \leftarrow 0$; j \leftarrow j-1

$A[j] \leftarrow 1$

for k=1 to n:

if $A[k] == 1$:

weight = weight + w[k]

value = value + v[k]

Knapsack_BruteForce(W, w[], v[], n):

$A[] \leftarrow 0$ //n位0

for i=1 to 2^n :

weight $\leftarrow 0$; value $\leftarrow 0$

...

for k=1 to n:

if $A[k] == 1$:

weight = weight + w[k]

value = value + v[k]

if value > bestValue and weight \leq W:

bestValue = value

bestWeight = weight

bestChoice $\leftarrow A$

return bestChoice

蛮力法Brute-force (穷举法Exhaustive Search)

- 穷举所有可能的解(情况)。

优点:

- 适应力强，所有问题都可以用它解决
- 容易实现
- 问题规模不大的都可以用它立刻得到解
- 有些问题只能用穷举法。（求一组数的最大值）

缺点:

- 性能往往不好，时间复杂度达到指数级

7. 排序问题

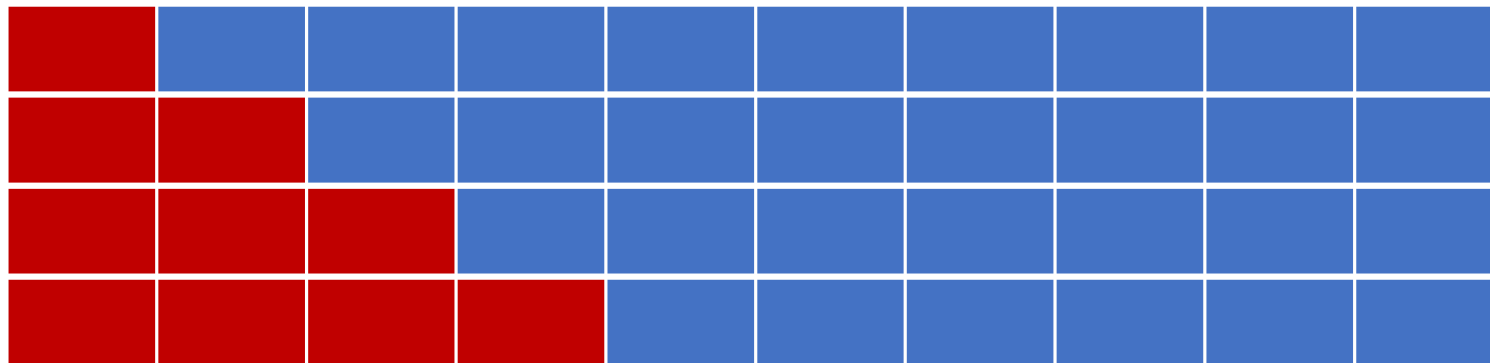
- 问题：给定一个序列，要求按照非降次序排序。

- 如：

输入：[9,1,4,7,3,2,8],

输出：[1,2,3,4,7,8,9]

- 很多简单排序算法迭代地使有序序列递增



选择排序

从所有元素中选择最小的，放在第1个位置，

从剩下元素中选择最小的，放在第2个位置，

...


从最后2个元素中选择最小的，放在第n-1个位置.

$a_1 \quad a_2 \quad a_3 \quad \dots \quad a_n$

$b_1 \quad a_2' \quad a_3' \quad \dots \quad a_n'$

$b_1 \quad b_2 \quad a_3'' \quad \dots \quad a_n''$


$(a_1, a_2, \dots, a_i, \dots, a_j, \dots, a_n)$



$(a_1, a_2, \dots, a_i, \dots, a_j, \dots, a_n)$

```

for i ← 1 to n-1 do
  min ← i
  for j = i+1 to n do
    if  $a_j < a_{\min}$ 
      min ← j
  swap  $a_i$  and  $a_{\min}$ 
  
```

 **n-i**

$$T(n) = \sum_{i=1}^{n-1} (n - i) = (n - 1 + n - 2 + \dots + 1)$$

$$T(n) = \frac{n(n-1)}{2} = \Theta(n^2)$$

冒泡排序

从第1个元素开始，每个元素和后面的比较，如果逆序，则交换。

得到第1大（小）的元素



冒泡排序

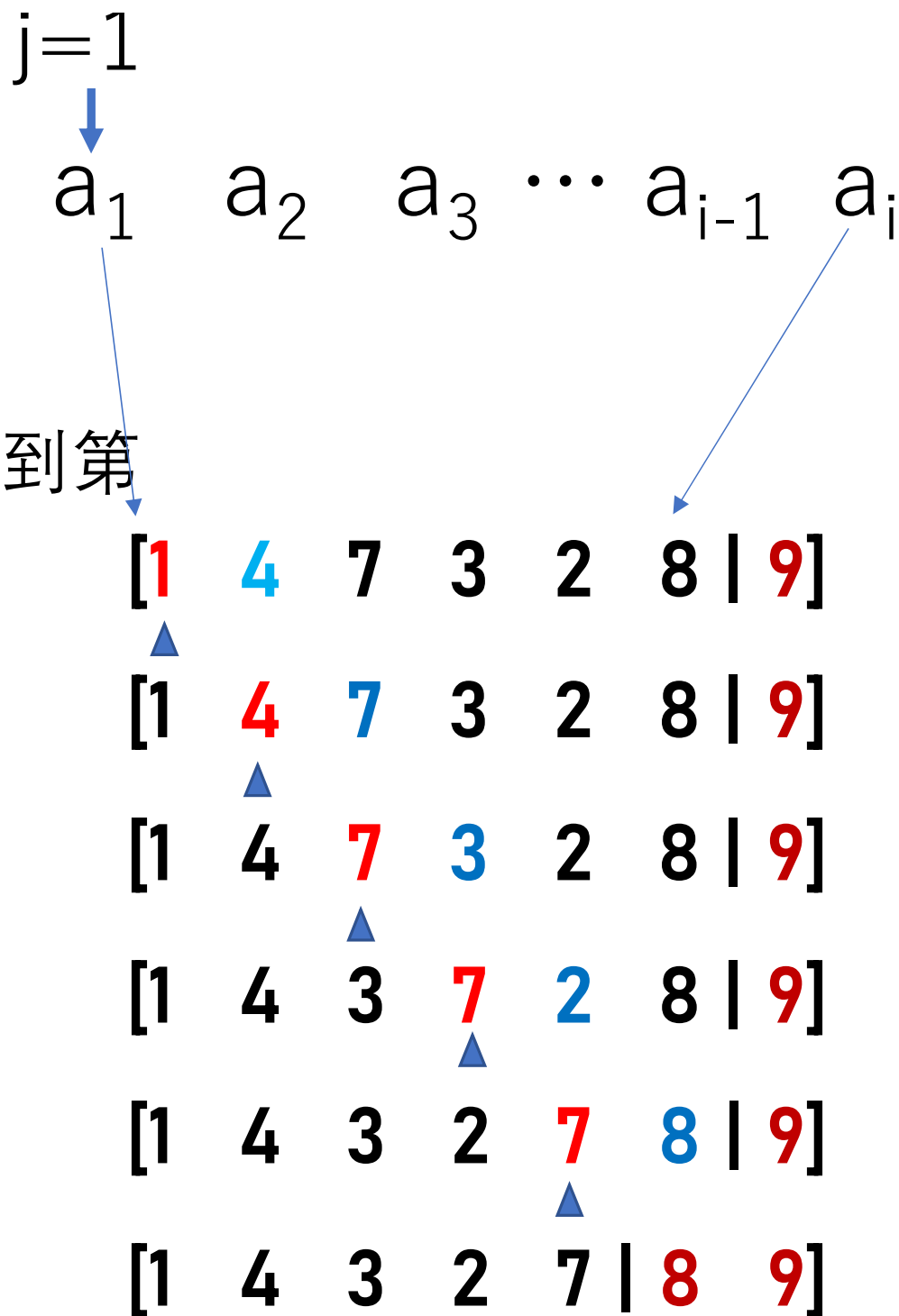
对剩下的元素重复上述过程，得到第
2大（小）的元素。

...

for $j=1$ to $i-1$ do

if $a_j > a_{j+1}$

swap a_{j+1} and a_j



冒泡排序

$j=2$



a_1 a_2 a_3 \cdots a_{i-1} a_i

对剩下的元素重复上述过程，得到第
2大（小）的元素。

...

for $j=1$ to $i-1$ do

if $a_j > a_{j+1}$

swap a_{j+1} and a_j

[1 4 7 3 2 8 | 9]

[1 4 7 3 2 8 | 9]

[1 4 7 3 2 8 | 9]

[1 4 3 7 2 8 | 9]

[1 4 3 2 7 8 | 9]

[1 4 3 2 7 | 8 9]

冒泡排序

对剩下的元素重复上述过程，得到第
2大（小）的元素。

...

for $j=1$ to $i-1$ do

if $a_j > a_{j+1}$

swap a_{j+1} and a_j



冒泡排序

对剩下的元素重复上述过程，得到第
2大（小）的元素。

...

for $j=1$ to $i-1$ do

if $a_j > a_{j+1}$


swap a_{j+1} and a_j



冒泡排序

对剩下的元素重复上述过程，得到第2大（小）的元素。 [1]

...

for **j=1** to i-1 do  i-1

if $a_j > a_{j+1}$

swap a_{j+1} and a_j



冒泡排序

对剩下的元素重复上述过程，得到第2大（小）的元素。

```
for i = n to 2
...
    for j=1 to i-1 do      ← i-1
        if aj > aj+1
            swap aj+1 and aj
```

$$T(n) = \sum_{i=n}^2 (i - 1) = (n - 1 + n - 2 + \dots + 1)$$

$$T(n) = \frac{n(n-1)}{2} = \Theta(n^2)$$

插入排序

- n 个元素的排序问题转化为： $n-1$ 次插入元素到有序序列

2	3	1	7	4
---	---	---	---	---

第1个元素是一个有序序列

2	3	1	7	4
---	---	---	---	---

第2个元素插入前面有序序列

1	2	3	7	4
---	---	---	---	---

第3个元素插入前面有序序列

1	2	3	7	4
---	---	---	---	---

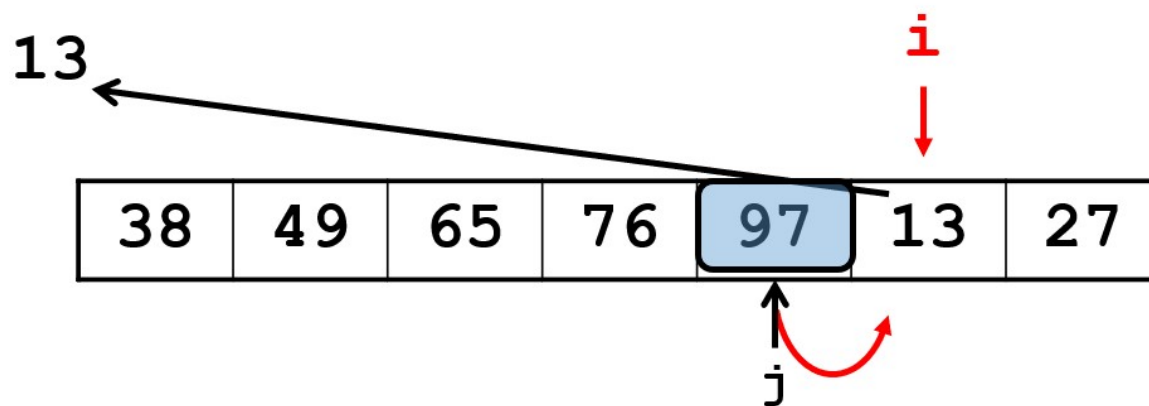
第4个元素插入前面有序序列

1	2	3	4	7
---	---	---	---	---

第5个元素插入前面有序序列

插入排序

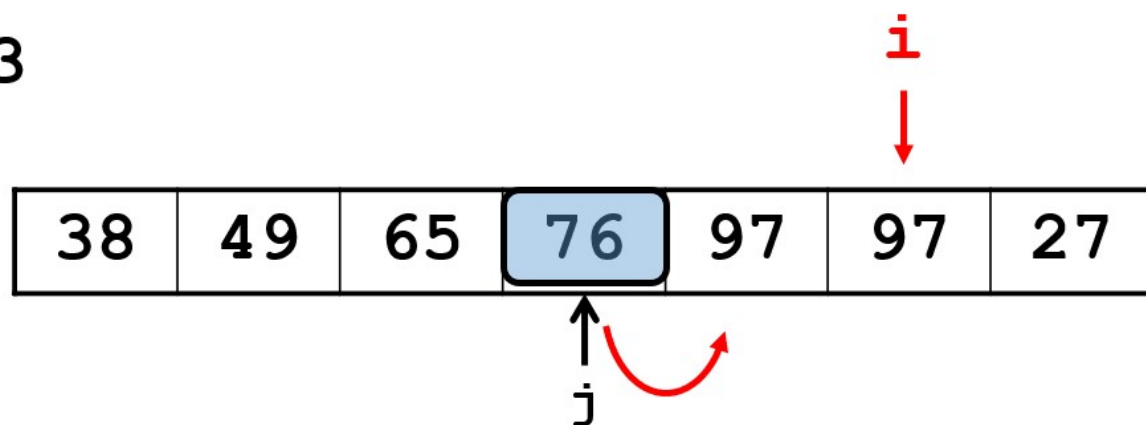
```
for i=2 to n: // 将 $a_i$ 插入到前面 $i-1$ 个元素的有序序列中  
    if  $a_i < a_{i-1}$ :  
         $t = a_i$ ;  
         $a_{i-1} \leftarrow a_i$ ;
```



插入排序

$$T(n) = \Theta(n^2)$$

```
for i=2 to n: // 将 $a_i$ 插入到前面 $i-1$ 个元素的有序序列中  
    if  $a_i < a_{i-1}$ :  
         $t = a_i$ ;  
         $a_{i-1} \leftarrow a_i$ ;  
        for( $j=i-2$ ;  $j \geq 1 \ \&\& \ t < a_j$ ;  $j--$ )  
             $a_{j+1} \leftarrow a_j$ ; //  $a_i$ 后移  
         $a_{i+1} = t$ ;
```



8. 最近点对问题

- 给定平面上N个点的坐标，找出距离最近的两个点。



两个点 $P_i=(x_i, y_i)$ 和 $P_j=(x_j, y_j)$ 间的距离是欧几里得距离：

$$d(P_i, P_j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$$

可以不求平方根

应用

- 图形学、计算机视觉、交通控制、分子建模、地理信息系统

BruteForceClosestPoints (P)

$d_{\min} \leftarrow \infty$

for $i \leftarrow 1$ to $n-1$ do

for $j \leftarrow i+1$ to n do

$d \leftarrow (x_i - x_j)^2 + (y_i - y_j)^2$

if $d < d_{\min}$

$d_{\min} \leftarrow d$

$idx1 \leftarrow i; \quad idx2 \leftarrow j$

return $idx1, idx2$

$i=1: \quad n-1$

$i=2: \quad n-2$

...

$i=n-1: \quad 1$

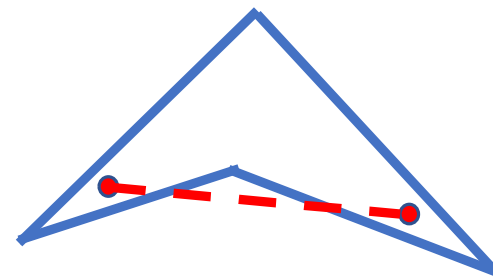
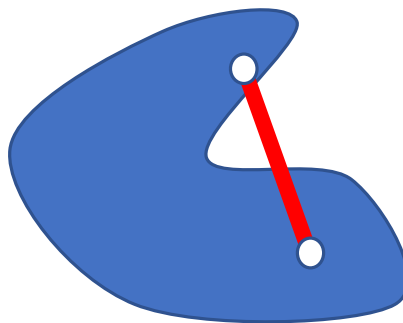
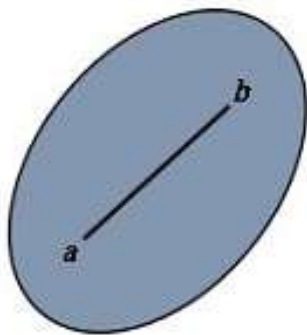
$$T(n) = n(n-1)/2 = \Theta(n^2)$$

7 凸包 (Convex Hull)

YouTube频道: [hwdong](#)

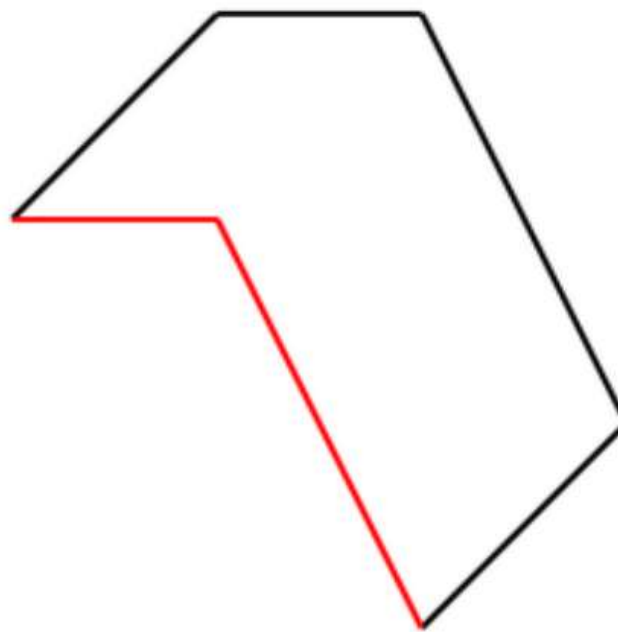
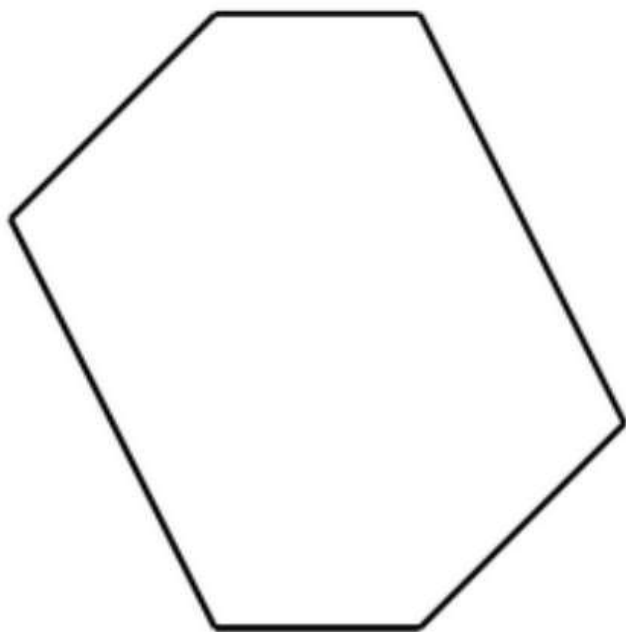
凸集

- 欧几里得空间 R^n 的集合 S 是凸集：是指对于集合内的每一对点，连接该对点的直线段上的每个点也在该集合内。



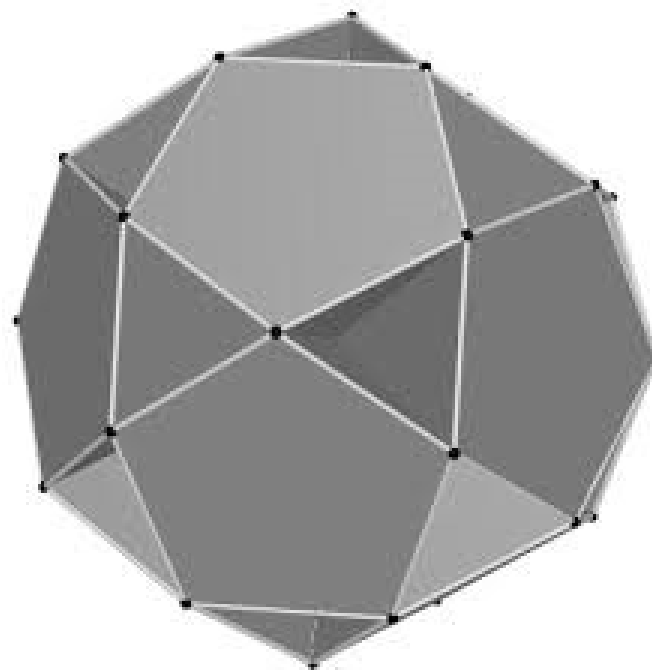
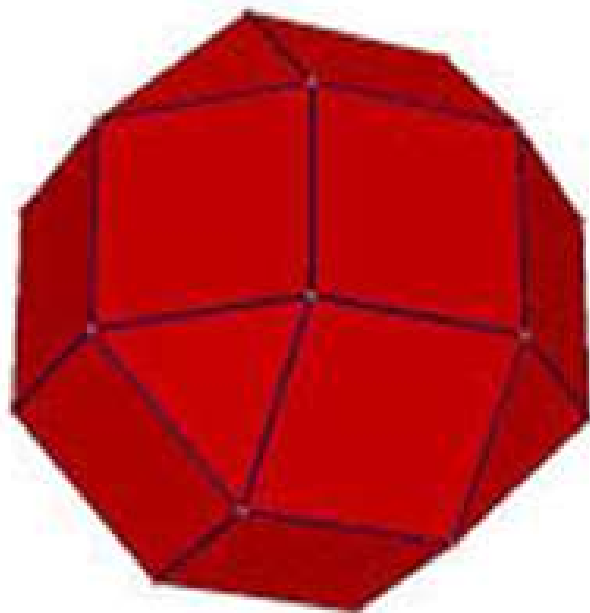
凸集

- 平面上的凸多边形、非凸多边形



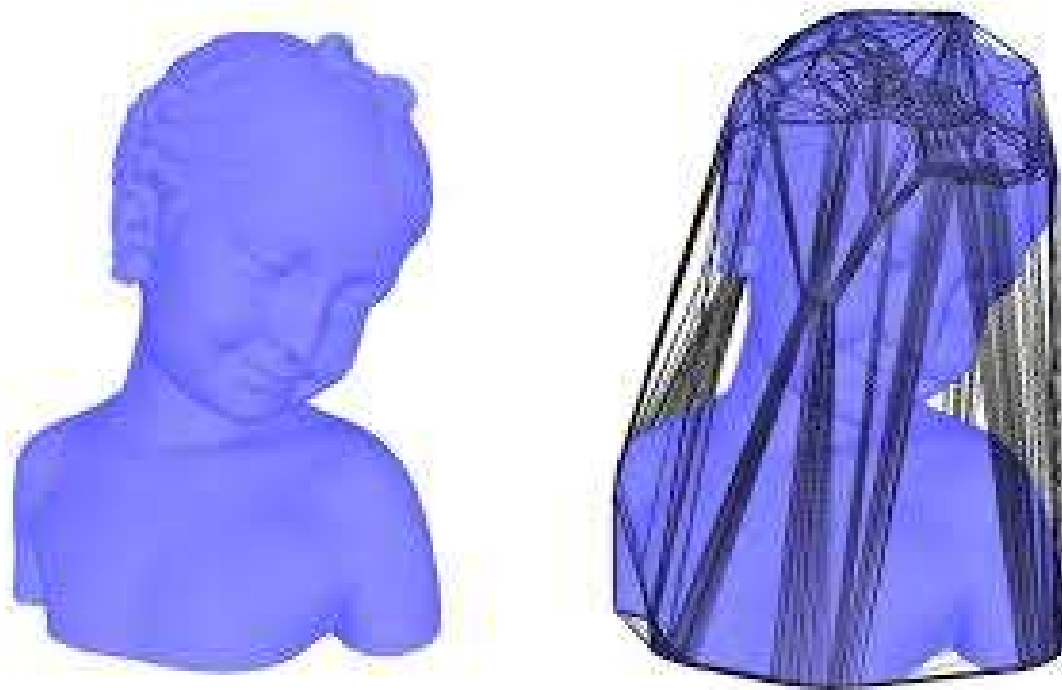
凸集

- 三维空间的凸多面体、非凸多面体



凸包 (Convex Hull)

- 集合 S 欧几里得空间 R^n 的子集, 包含 S 的所有凸集的交集, 称为 S 的凸包, 记为 $\text{conv}(S)$ 。

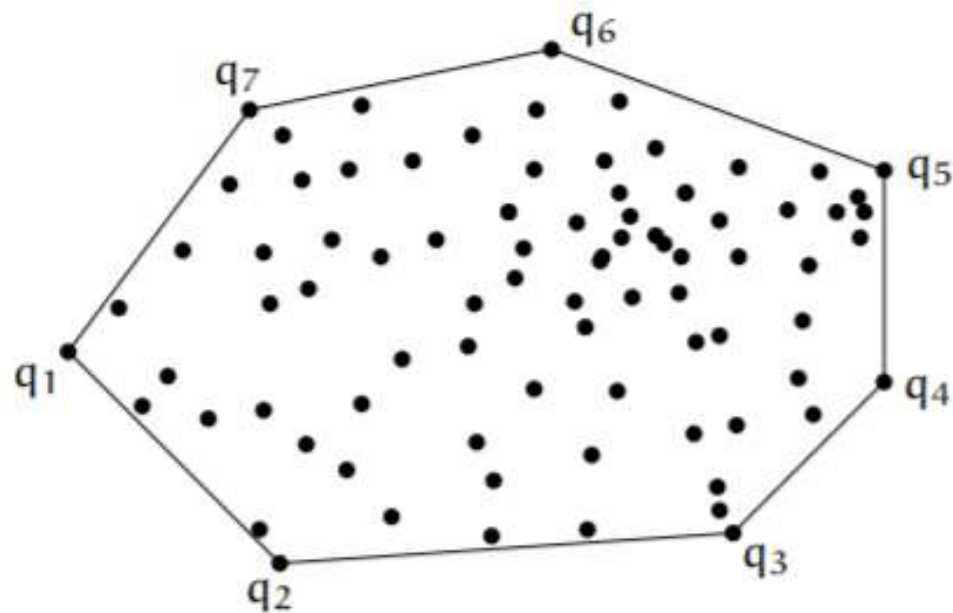


凸包 (Convex Hull)

- 集合 S 是欧几里得空间 R^n 的子集，包含 S 的所有凸集交集，称为 S 的凸包，记为 $\text{conv}(S)$ 。
- 平面点集的凸包是包围点集中所有点的最小凸多边形。



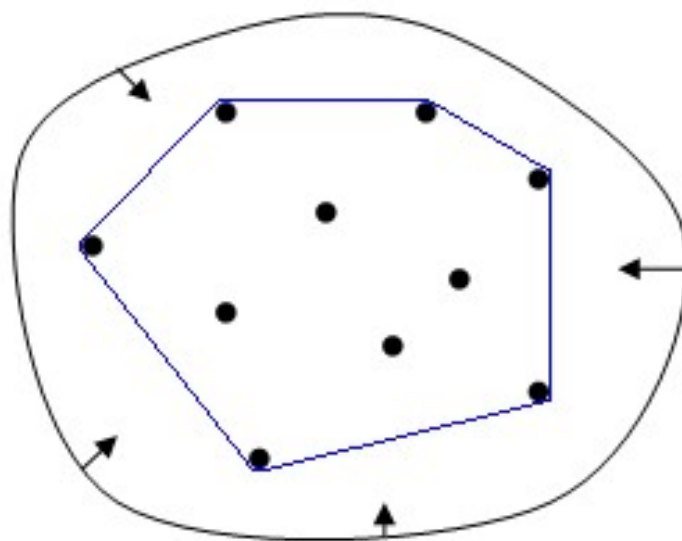
(a) Input.



(b) Output.

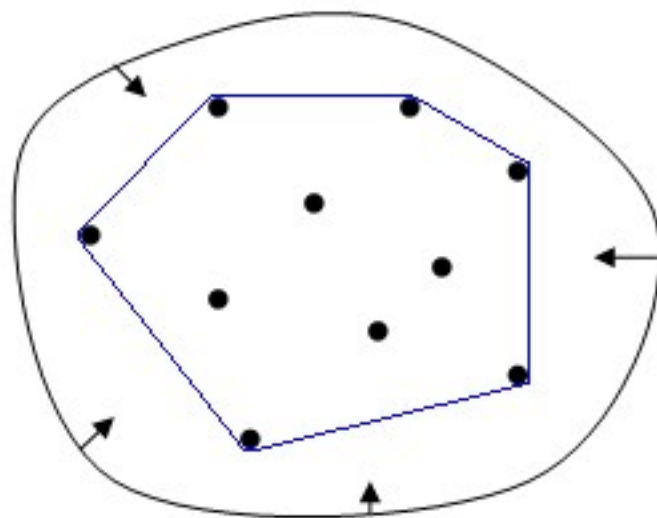
凸包 (Convex Hull)

- 集合 S 欧几里得空间 R^n 的子集，包含 S 的所有凸集的交集，称为 S 的凸包，记为 $\text{conv}(S)$ 。
- 平面点集的凸包是包围点集中所有点的最小凸多边形。
- 可以将这些点想象成一些树桩，用一根绳子紧紧地包围它们。



凸包的应用

- 凸包问题是计算几何的重要问题，在图形学、路径规划、优化等许多领域有重要应用。



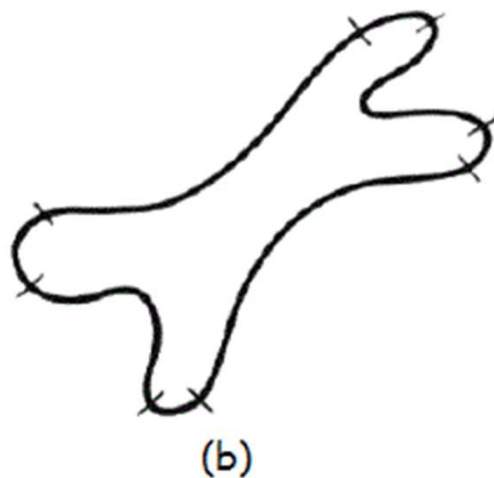
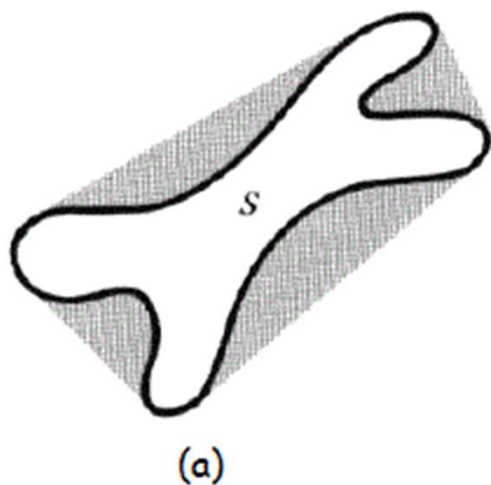
凸包的应用

- 碰撞检测(Collision avoidance)



凸包的应用

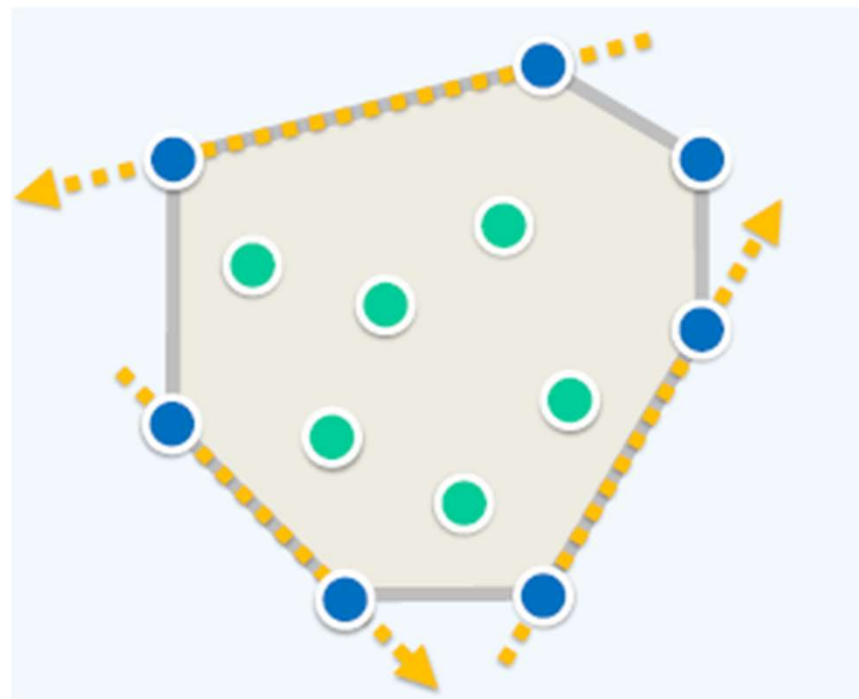
- 碰撞检测(Collision avoidance)
- 最小包围盒(Smallest box)
- 形状分析



(a) A region (S) and its convex deficiency(shaded); (b) partitioned boundary

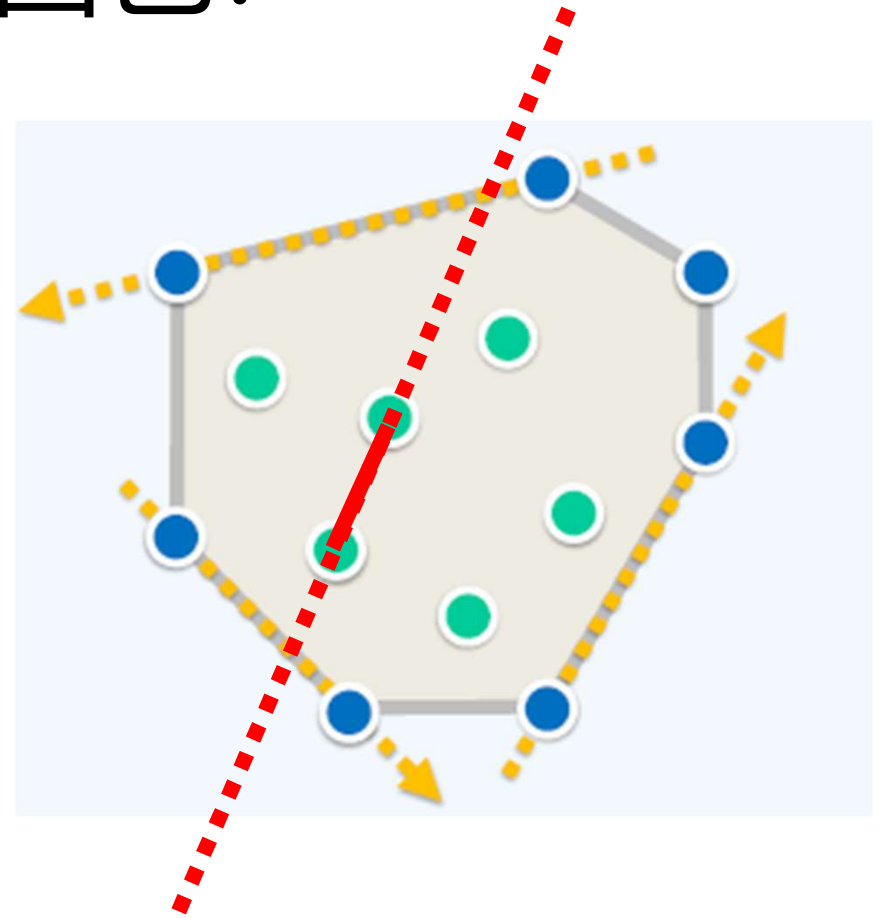
如何求一个平面点集的凸包？

- 凸包是由极边构成的。



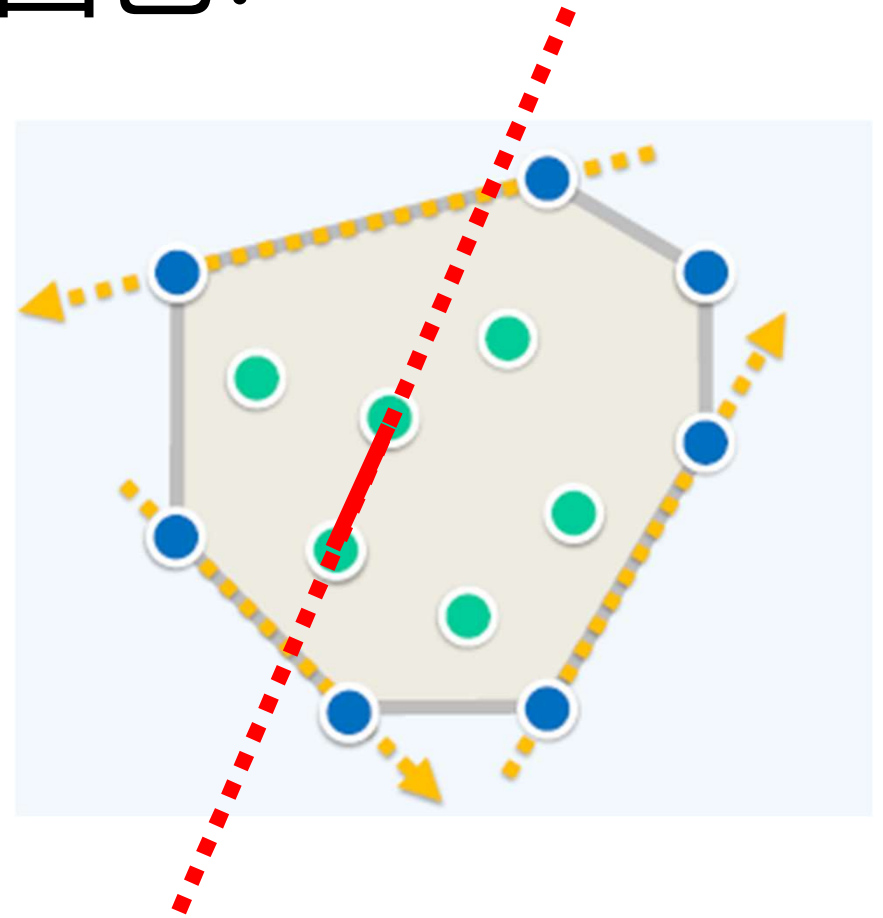
如何求一个平面点集的凸包？

- 凸包是由极边构成的。
- 极边：其他点都在该边同一侧



如何求一个平面点集的凸包？

- 凸包是由极边构成的。
- 极边：其他点都在该边同一侧
- 蛮力法：
检测每对点构成的边是否极边？



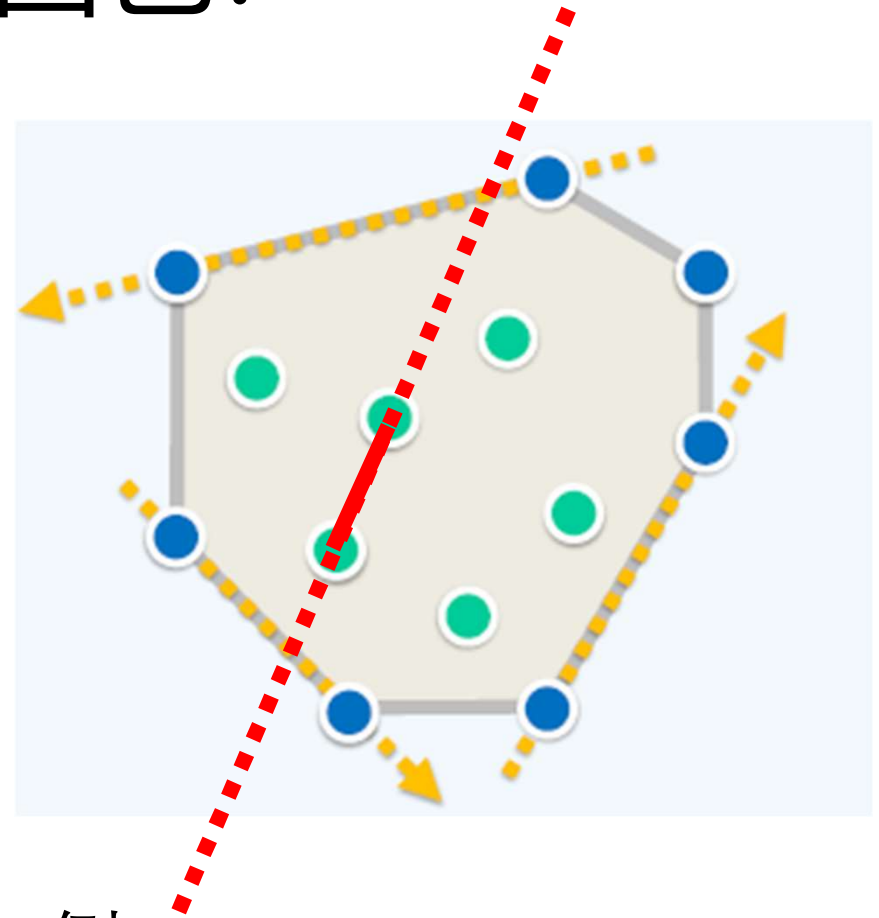
如何求一个平面点集的凸包?

- 凸包是由极边构成的。
- 极边：其他点都在该边同一侧
- 蛮力法：
检测每对点构成的边是否极边？

for 每个 P_i :

for 每个不同于 P_i 的点 P_j :

if 其他所有点都位于 P_iP_j 同一侧
 P_iP_j 是凸包的一条边



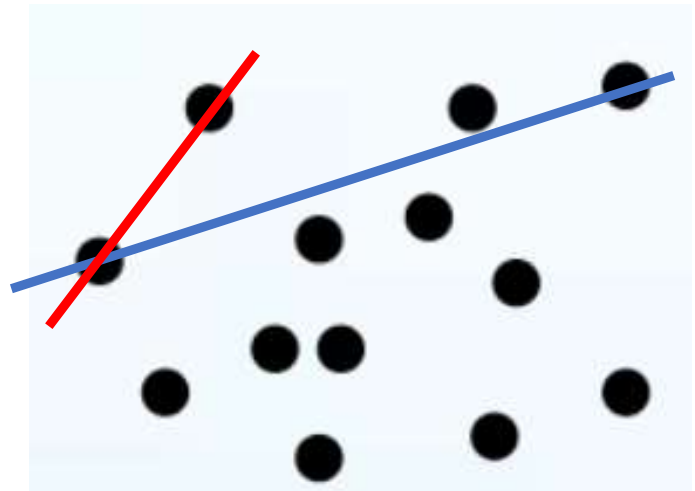
for 每个 P_i :

for 每个不同于 P_i 的点 P_j :

计算直线方程: $ax+by+c$

if 不同于 P_i 和 P_j 的其他点 P_k 都位于直线的同一侧

P_iP_j 是凸包的一条边



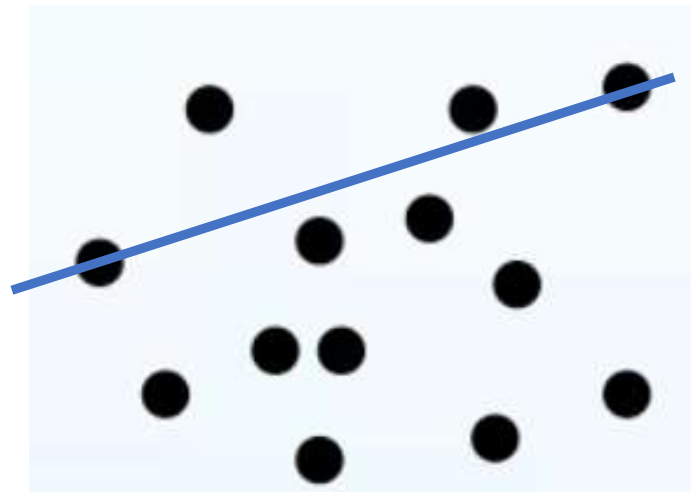
判断点 p 在直线 p1、p2 的哪一侧？

- p1、p2的坐标为：p1(x1,y1), p2(x2,y2), p1p2的直线方程为：

$$ax + by - c = 0$$

$$a = y_2 - y_1, b = x_1 - x_2, c = x_1y_2 - y_1x_2$$

- 直线将平面分为2个半平面，一个半平面上的点坐标 p(x,y) 带入方程，其值为负数，另一半平面的值为正数。



PointPairs $\leftarrow \emptyset$

for $i \leftarrow 1$ to $n-1$ **do**

for $j \leftarrow i+1$ to n **do**

$n1 \leftarrow 0; n2 \leftarrow 0; \text{sameSide} \leftarrow \text{True}$

$a \leftarrow y_j - y_i; b \leftarrow x_i - x_j; c \leftarrow x_i y_j - y_i x_j$

for $k \leftarrow 1$ to n **do**

if $k=i$ or $k=j$: **continue**

$\text{sign} = ax_k + by_k - c < 0$

if $\text{sign} < 0$: $n1 = n1 + 1$

else if $\text{sign} > 0$: $n2 = n2 + 1$

if $n1 > 0$ and $n2 > 0$:

$\text{sameSide} \leftarrow \text{False}; \text{break}$

if $\text{sameSide} = \text{True}$: Add $\langle i, j \rangle$ to PointPairs

return PointPairs

$\leftarrow n(n-1)/2$

$\leftarrow \leq (n-2)$

$T(n) = O(n^3)$

PointPairs $\leftarrow \emptyset$

for $i \leftarrow 1$ to $n-1$ **do**
 for $j \leftarrow i+1$ to n **do**

$\leftarrow n(n-1)/2$

 currSign $\leftarrow 0$; sameSide \leftarrow True

$a \leftarrow y_j - y_i$; $b \leftarrow x_i - x_j$; $c \leftarrow x_i y_j - y_i x_j$

for $k \leftarrow 1$ to n **do**

$\leftarrow \leq (n-2)$

 ...

else if sign==0 and

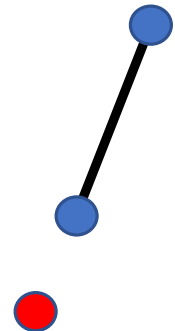
$(x_k - x_i)(x_k - x_j) > 0$ or $(y_k - y_i)(y_k - y_j) > 0$

 sameSide \leftarrow **False; break**

$T(n) = O(n^3)$

if sameSide = True: Add $\langle i, j \rangle$ to PointPairs

return PointPairs



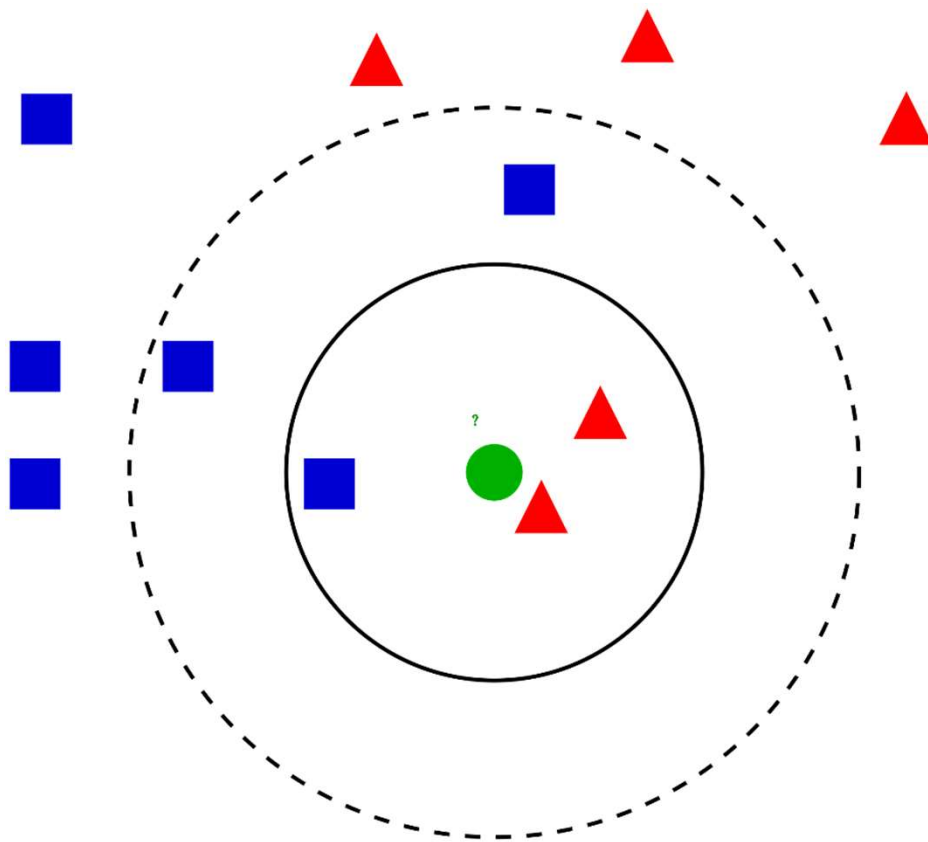
9.K近邻(KNN)

找出距离某点最近的k个点

YouTube频道: [hwdong](#)

KNN

- 是一个经典的机器学习算法。



三、多个线性表

博客：hwdong-net.github.io

Youtube频道：[hwdong](#)

多个线性表的迭代

- 有时问题设计多个线性表，就需要对多个线性表进行迭代，可同时齐头并进，也可以依次迭代。
- 如：如何合并2个（或多个）有序数组？如何合并2个（或多个）有序链表

合并2个有序数组

- 两个非递减有序序列a和b的大小分别是m、n，将它们合并为一个非递减有序序列c。
- 输入： $a = [1, 3, 7]$, $m = 3$, $b = [2, 5, 6]$, $n = 3$
- 输出： $c = [1, 2, 3, 5, 6, 7]$

合并2个有序数组

- 用2个指针(下标) i 、 j 分别指向2个数组开头，1个指针(下标) k 指向新数组开头， $a[i]$ 和 $b[j]$ 小的放入 $c[k]$ 位置上，移动相应指针。
- 将剩余元素放入 c 的后面。

a = [1, 3, 7]



i=0

b = [2, 5, 6]



j=0

c = [1, , , ,]



k=0

```
int i = 0, j = 0, k=0;
while(i<m&& j<n)
    if(a[i]<=a[j])
        c[k++] = a[i++];
    else c[k++] = b[j++];
while(i<m) //a有剩余元素，都追加到c的后面
    c[k++] = a[i++];
while(j<n) //b有剩余元素，都追加到c的后面
    c[k++] = b[j++];
```

a = [1, 3, 7]



i=1

b = [2, 5, 6]



j=0

c = [1, , , ,]



k=1

```
int i = 0, j = 0, k=0;
while(i<m&& j<n)
    if(a[i]<=a[j])
        c[k++] = a[i++];
    else c[k++] = b[j++];
while(i<m) //a有剩余元素，都追加到c的后面
    c[k++] = a[i++];
while(j<n) //b有剩余元素，都追加到c的后面
    c[k++] = b[j++];
```

a = [1, 3, 7]



i=1

b = [2, 5, 6]



j=0

c = [1, 2, , , ,]



k=1

```
int i = 0, j = 0, k=0;
while(i<m&& j<n)
    if(a[i]<=a[j])
        c[k++] = a[i++];
    else c[k++] = b[j++];
while(i<m) //a有剩余元素，都追加到c的后面
    c[k++] = a[i++];
while(j<n) //b有剩余元素，都追加到c的后面
    c[k++] = b[j++];
```

a = [1, 3, 7]



i=1

b = [2, 5, 6]



j=1

c = [1, 2, , , ,]



k=1

```
int i = 0, j = 0, k=0;
while(i<m&& j<n)
    if(a[i]<=a[j])
        c[k++] = a[i++];
    else c[k++] = b[j++];
while(i<m) //a有剩余元素，都追加到c的后面
    c[k++] = a[i++];
while(j<n) //b有剩余元素，都追加到c的后面
    c[k++] = b[j++];
```

a = [1, 3, 7]



i=0

b = [2, 5, 6]



j=0

c = [, , , , ,]



k=0

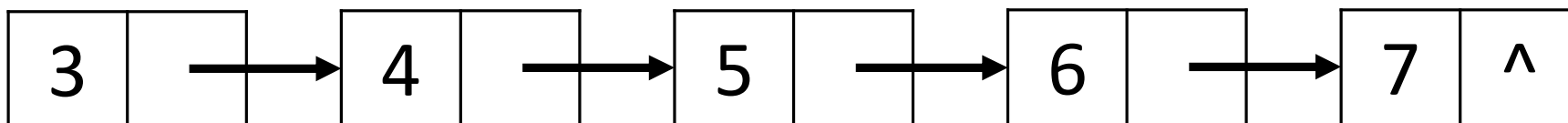
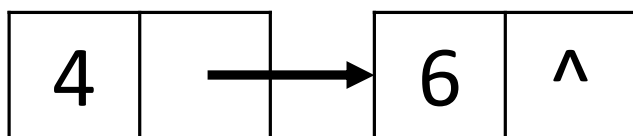
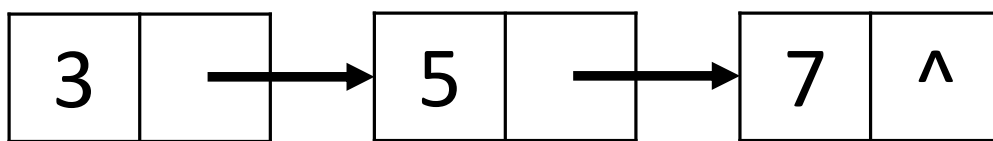
```
int i = 0, j = 0, k=0;
while(i<m&& j<n)
    if(a[i]<=a[j])
        c[k++] = a[i++];
    else c[k++] = b[j++];
while(i<m) //a有剩余元素，都追加到c的后面
    c[k++] = a[i++];
while(j<n) //b有剩余元素，都追加到c的后面
    c[k++] = b[j++];
```

```
template <typename T>
vector<T> mergeLists(const vector<T> &a , const vector<T> &b) {
    int m = a.size(), n = b.size();
    int i = 0, j = 0, k=0;
    vector<T> c(m+n);
    while(i<m&& j<n)
        if(a[i]<=a[j])
            c[k++] = a[i++];
        else c[k++] = b[j++];
    while(i<m) //a有剩余元素，都追加到c的后面
        c[k++] = a[i++];
    while(j<n) //b有剩余元素，都追加到c的后面
        c[k++] = b[j++];

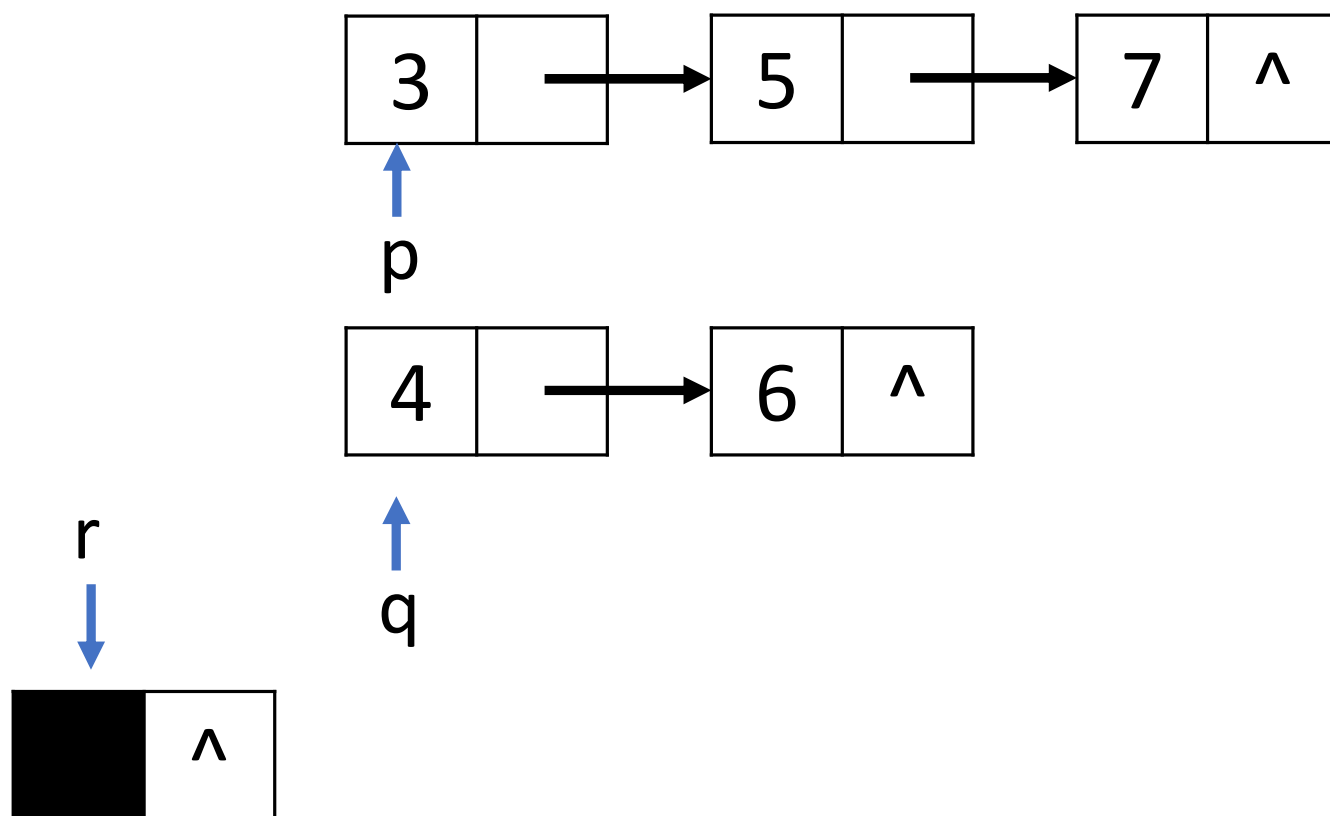
    return c;
}
```


合并2个有序链表

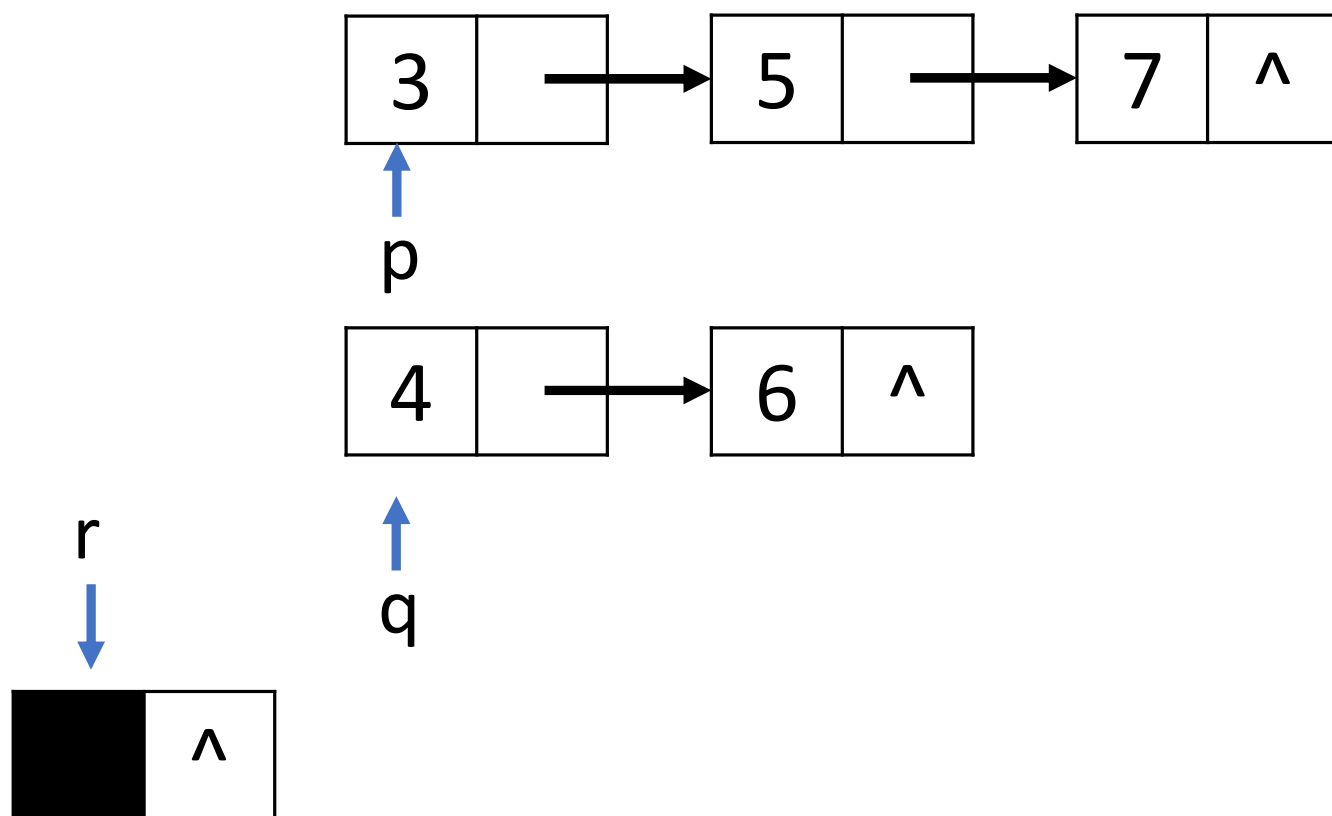
- 将两个递增链表合并为一个新的递增链表并返回。新链表是通过拼接给定的两个链表的所有节点组成的。



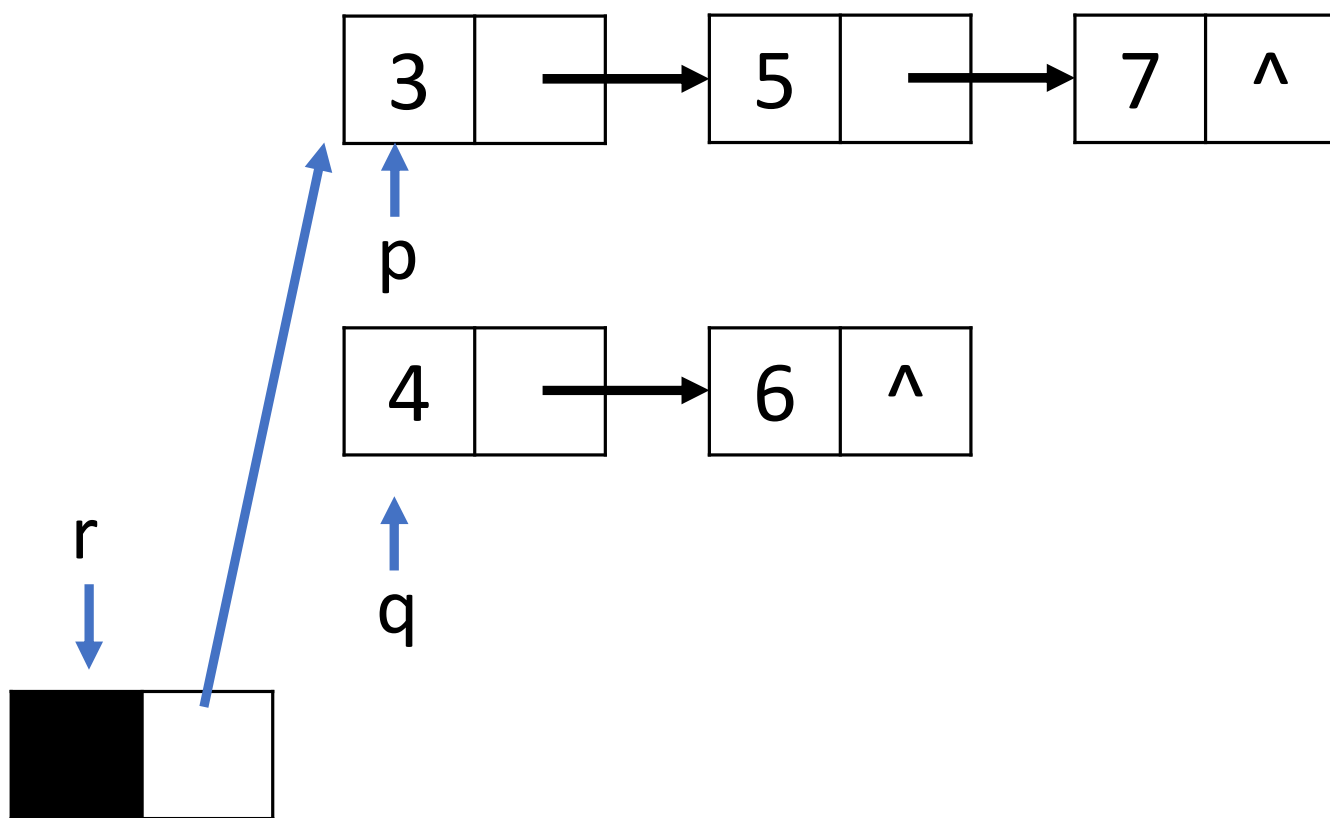
- 用2个指针p和q分别指向2个链表开头。为合并后的链表创建一个虚拟头结点head，用指针r指向新链表的尾结点



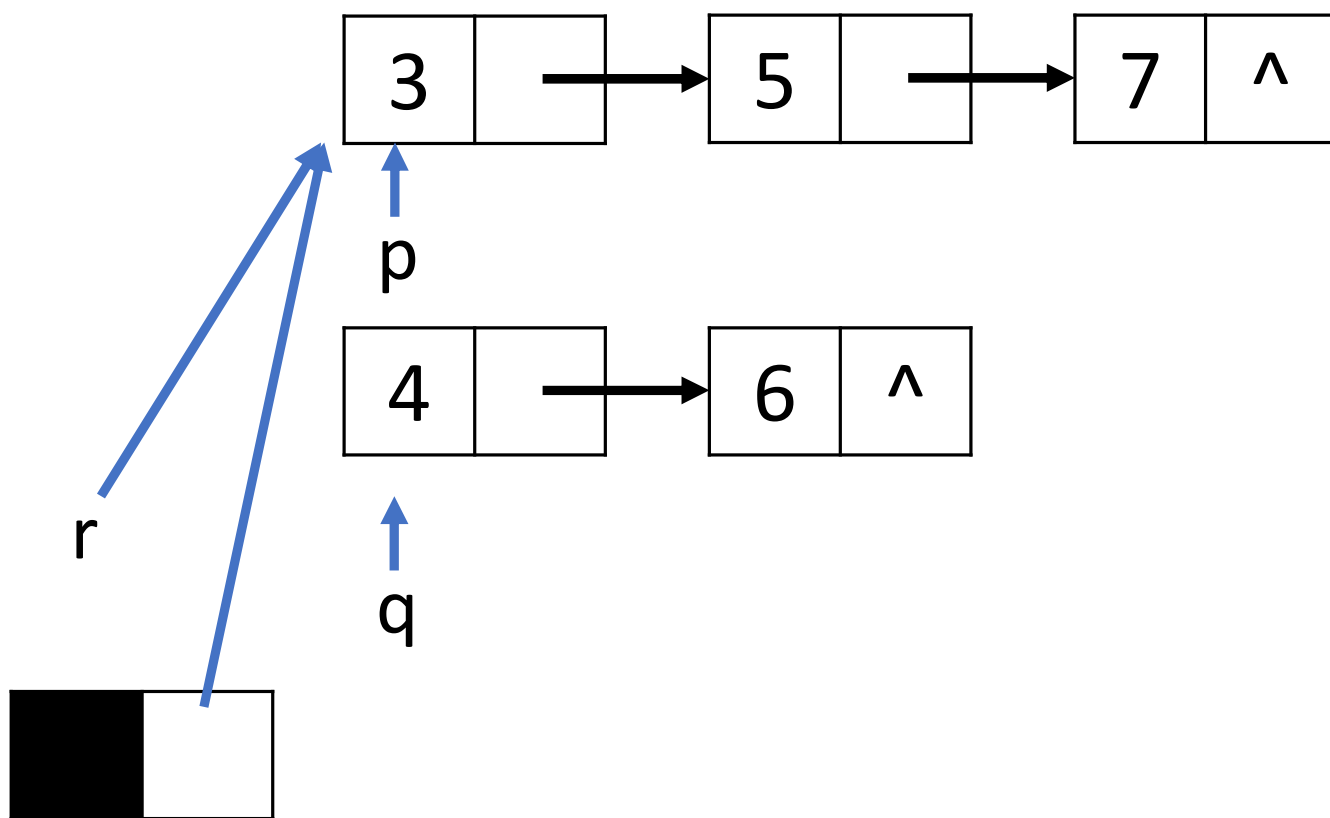
- 用2个指针p和q分别指向2个链表开头。为合并后的链表创建一个虚拟头结点head，用指针r指向新链表的尾结点
- 如果p和q都不空，将小的结点挂到新链表r的后面，更新r指向这个结点



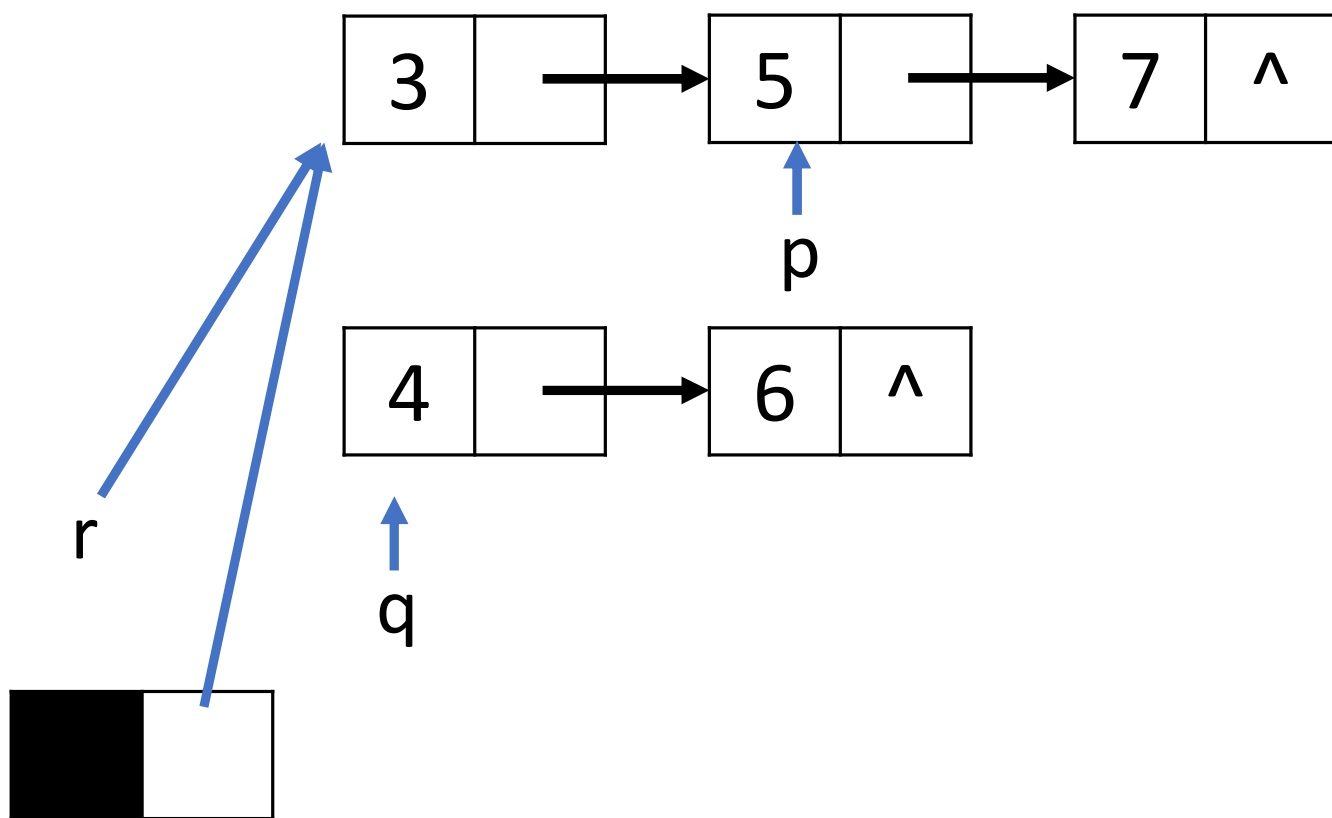
- 用2个指针p和q分别指向2个链表开头。为合并后的链表创建一个虚拟头结点head，用指针r指向新链表的尾结点
- 如果p和q都不空，将小的结点挂到新链表r的后面，更新r指向这个结点



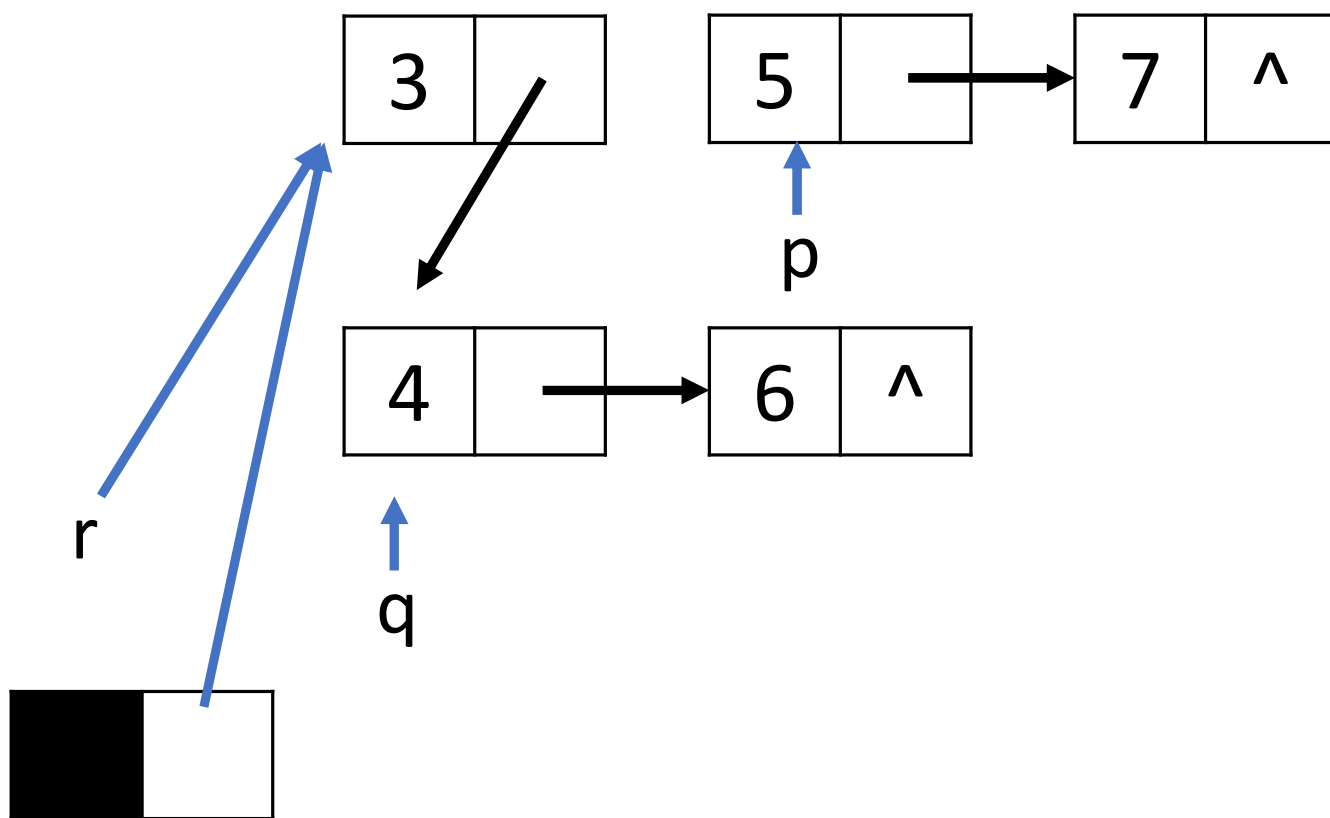
- 用2个指针p和q分别指向2个链表开头。为合并后的链表创建一个虚拟头结点head，用指针r指向新链表的尾结点
- 如果p和q都不空，将小的结点挂到新链表r的后面，更新r指向这个结点



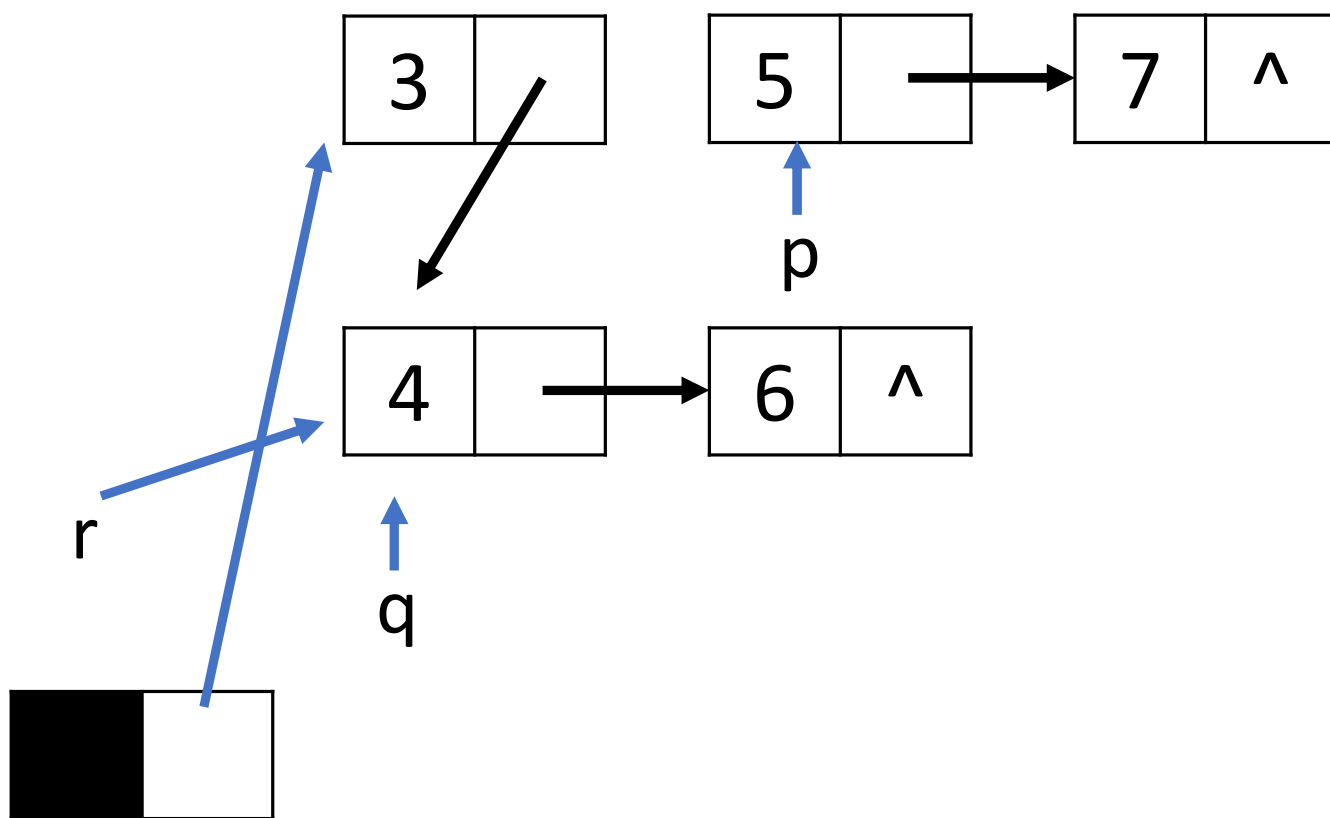
- 用2个指针p和q分别指向2个链表开头。为合并后的链表创建一个虚拟头结点head，用指针r指向新链表的尾结点
- 如果p和q都不空，将小的结点挂到新链表r的后面，更新r指向这个结点



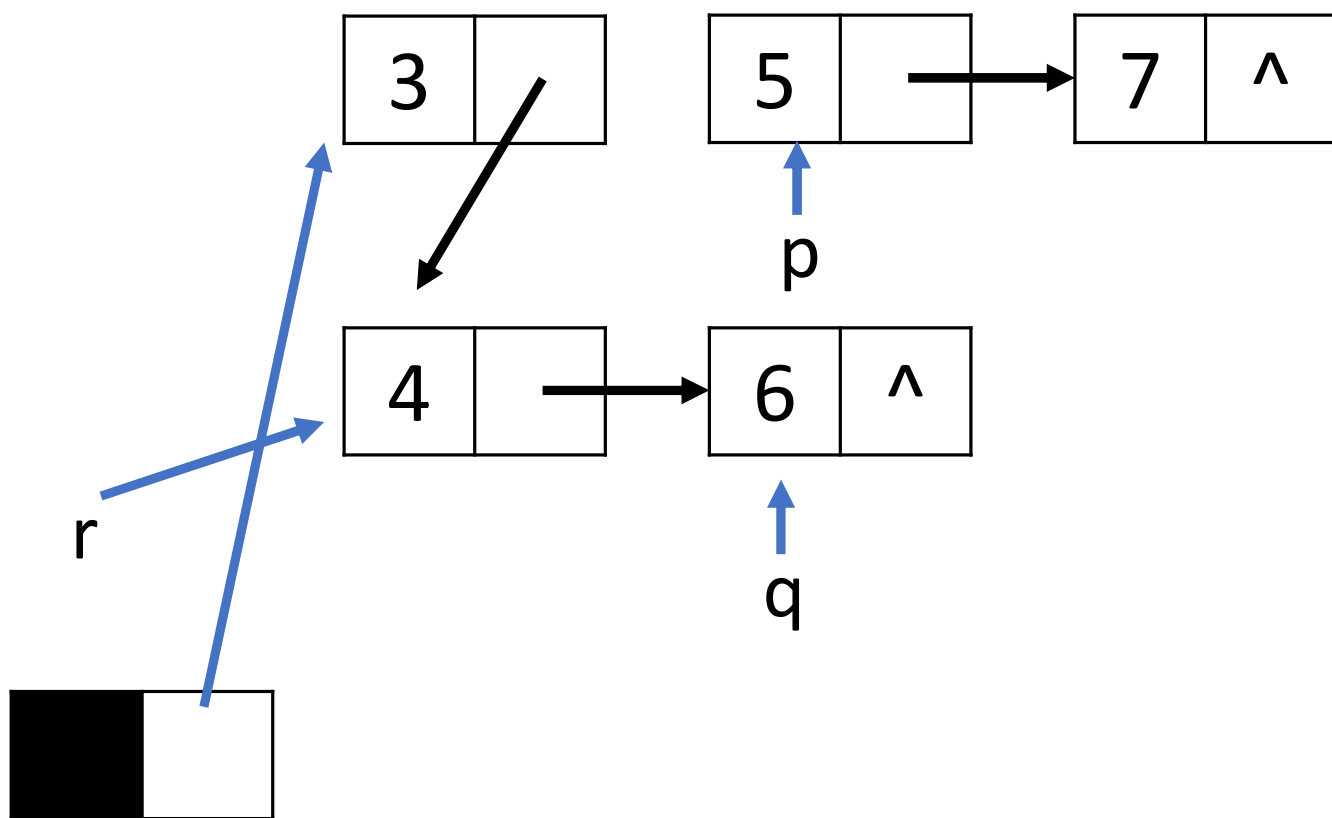
- 用2个指针p和q分别指向2个链表开头。为合并后的链表创建一个虚拟头结点head，用指针r指向新链表的尾结点
- 如果p和q都不空，将小的结点挂到新链表r的后面，更新r指向这个结点



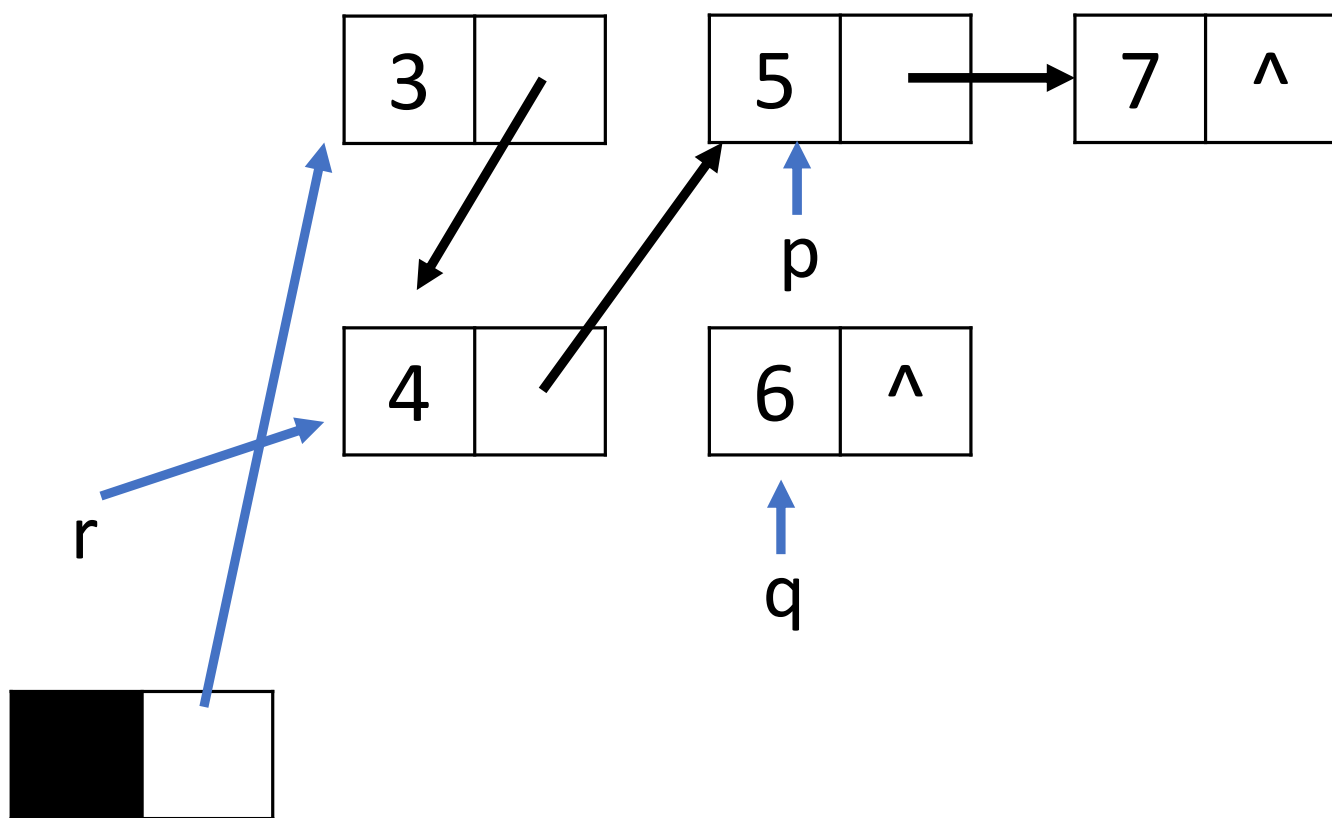
- 用2个指针p和q分别指向2个链表开头。为合并后的链表创建一个虚拟头结点head，用指针r指向新链表的尾结点
- 如果p和q都不空，将小的结点挂到新链表r的后面，更新r指向这个结点



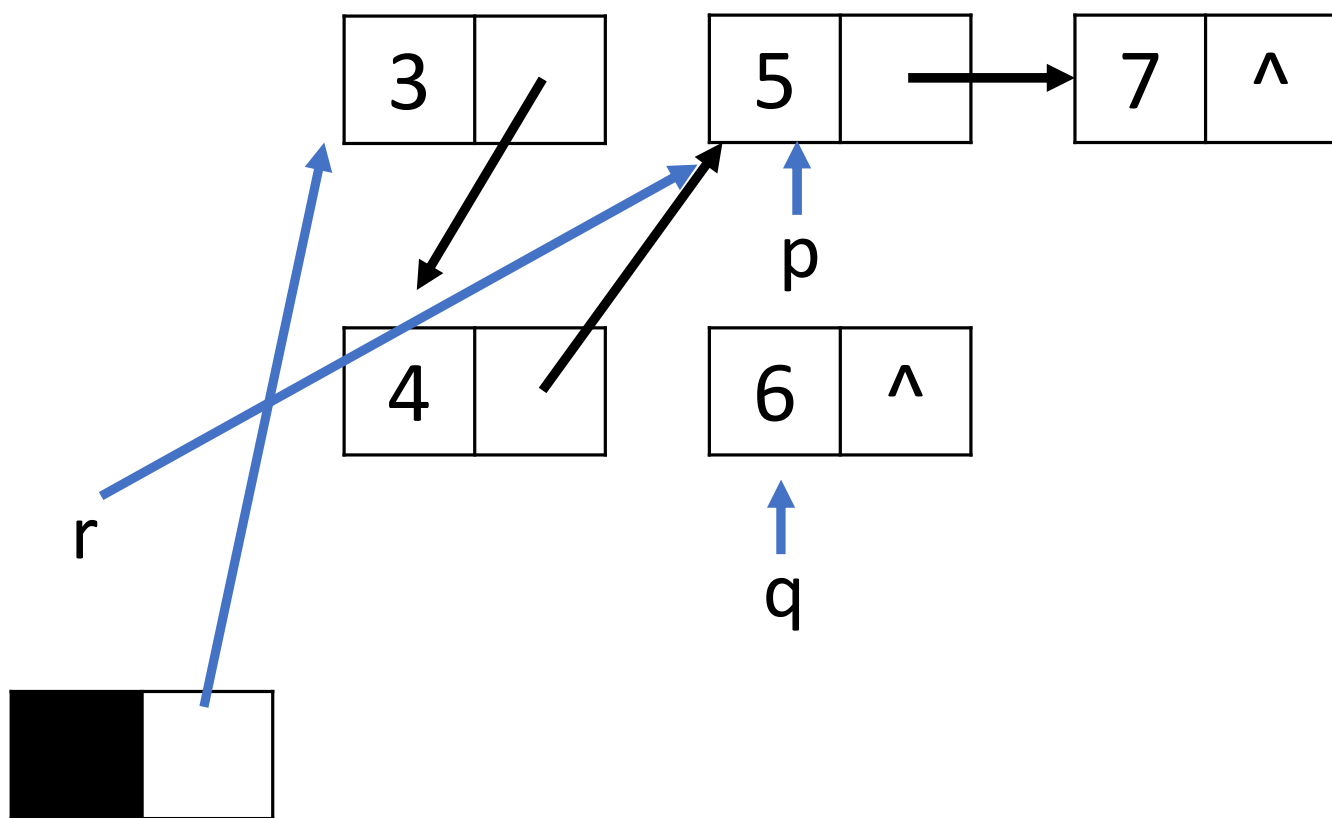
- 用2个指针p和q分别指向2个链表开头。为合并后的链表创建一个虚拟头结点head，用指针r指向新链表的尾结点
- 如果p和q都不空，将小的结点挂到新链表r的后面，更新r指向这个结点



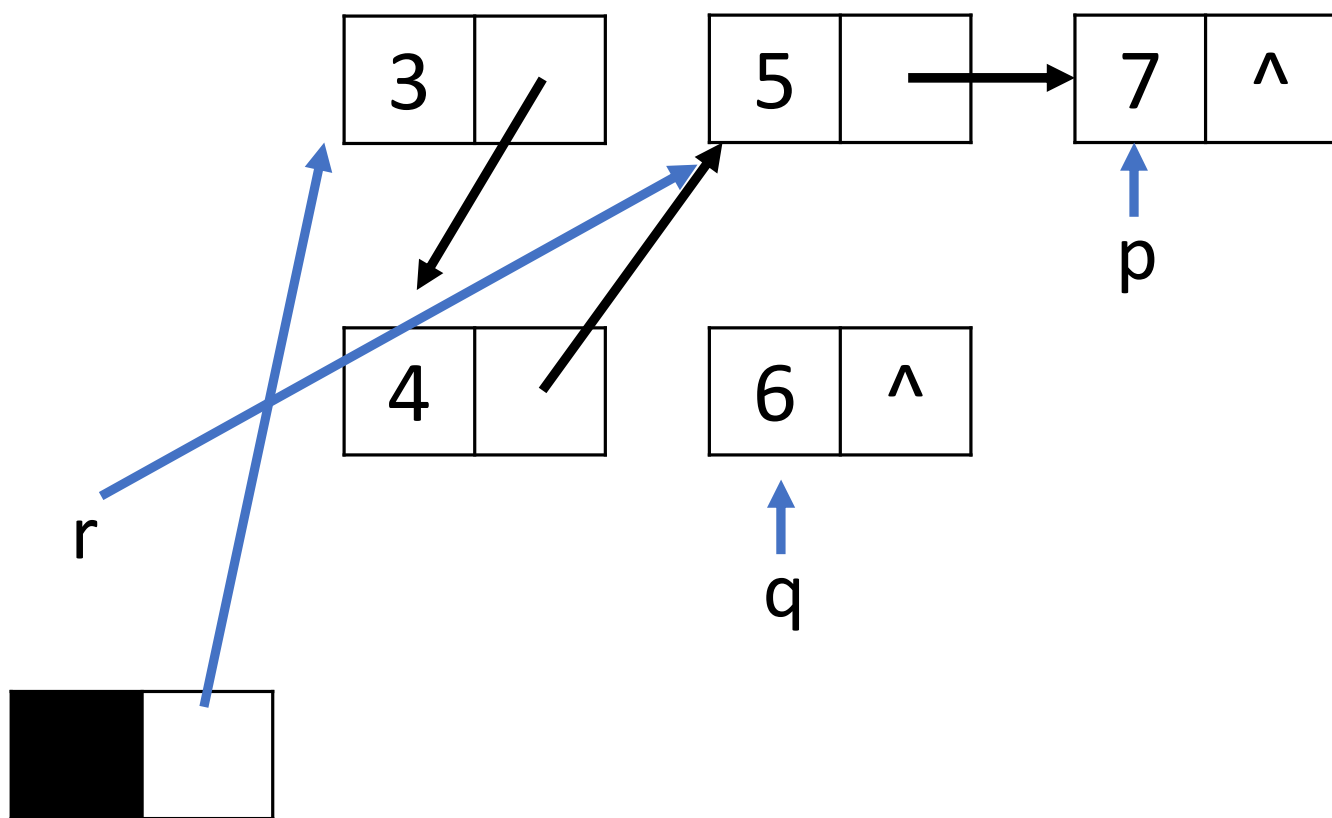
- 用2个指针p和q分别指向2个链表开头。为合并后的链表创建一个虚拟头结点head，用指针r指向新链表的尾结点
- 如果p和q都不空，将小的结点挂到新链表r的后面，更新r指向这个结点



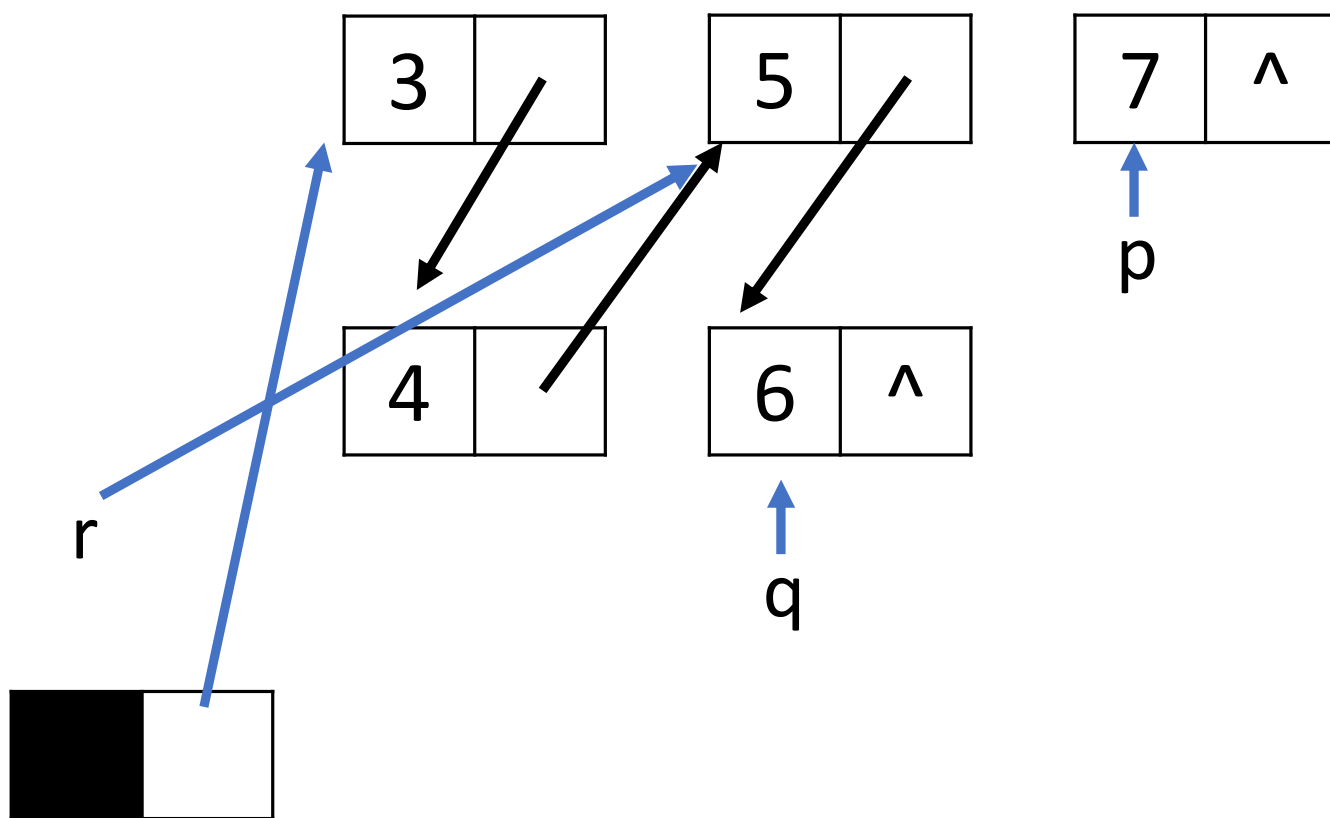
- 用2个指针p和q分别指向2个链表开头。为合并后的链表创建一个虚拟头结点head，用指针r指向新链表的尾结点
- 如果p和q都不空，将小的结点挂到新链表r的后面，更新r指向这个结点



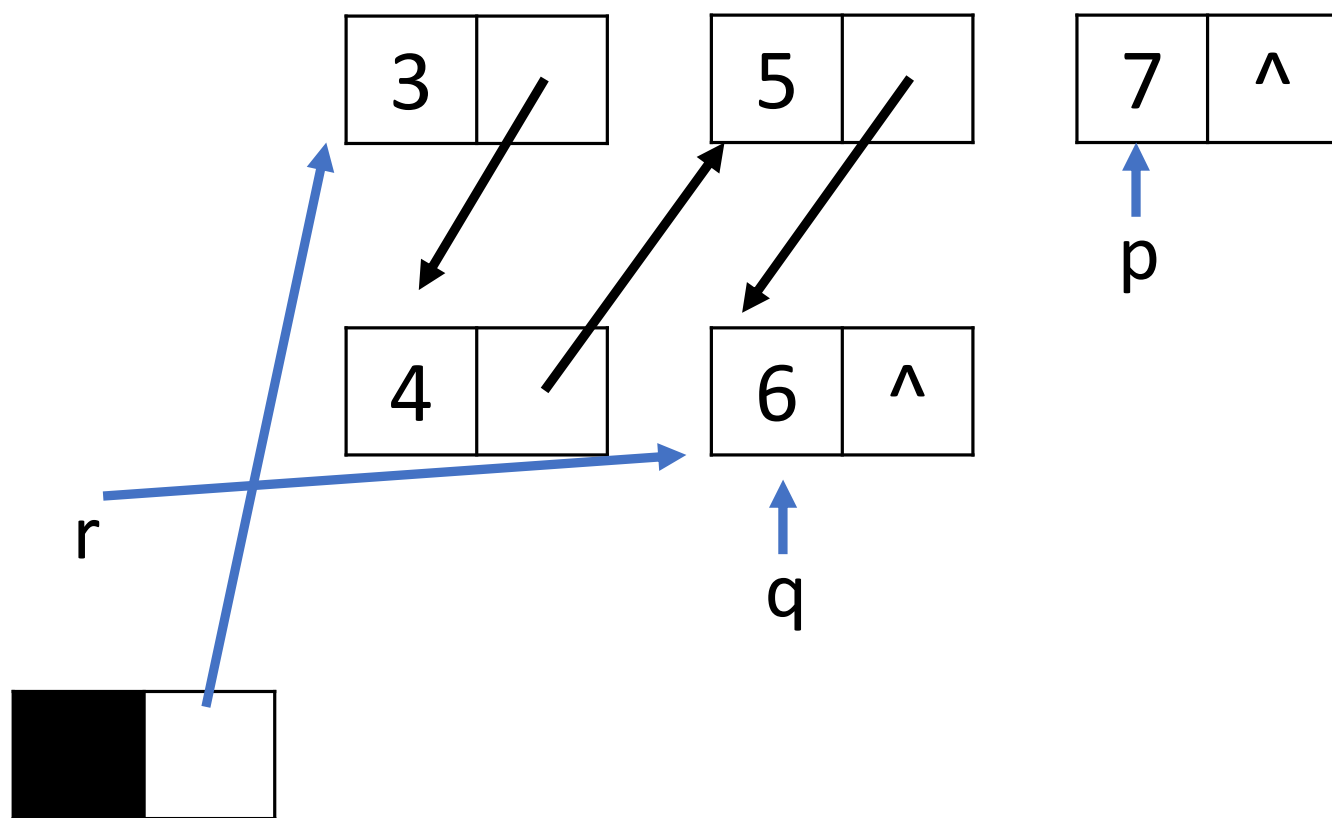
- 用2个指针p和q分别指向2个链表开头。为合并后的链表创建一个虚拟头结点head，用指针r指向新链表的尾结点
- 如果p和q都不空，将小的结点挂到新链表r的后面，更新r指向这个结点



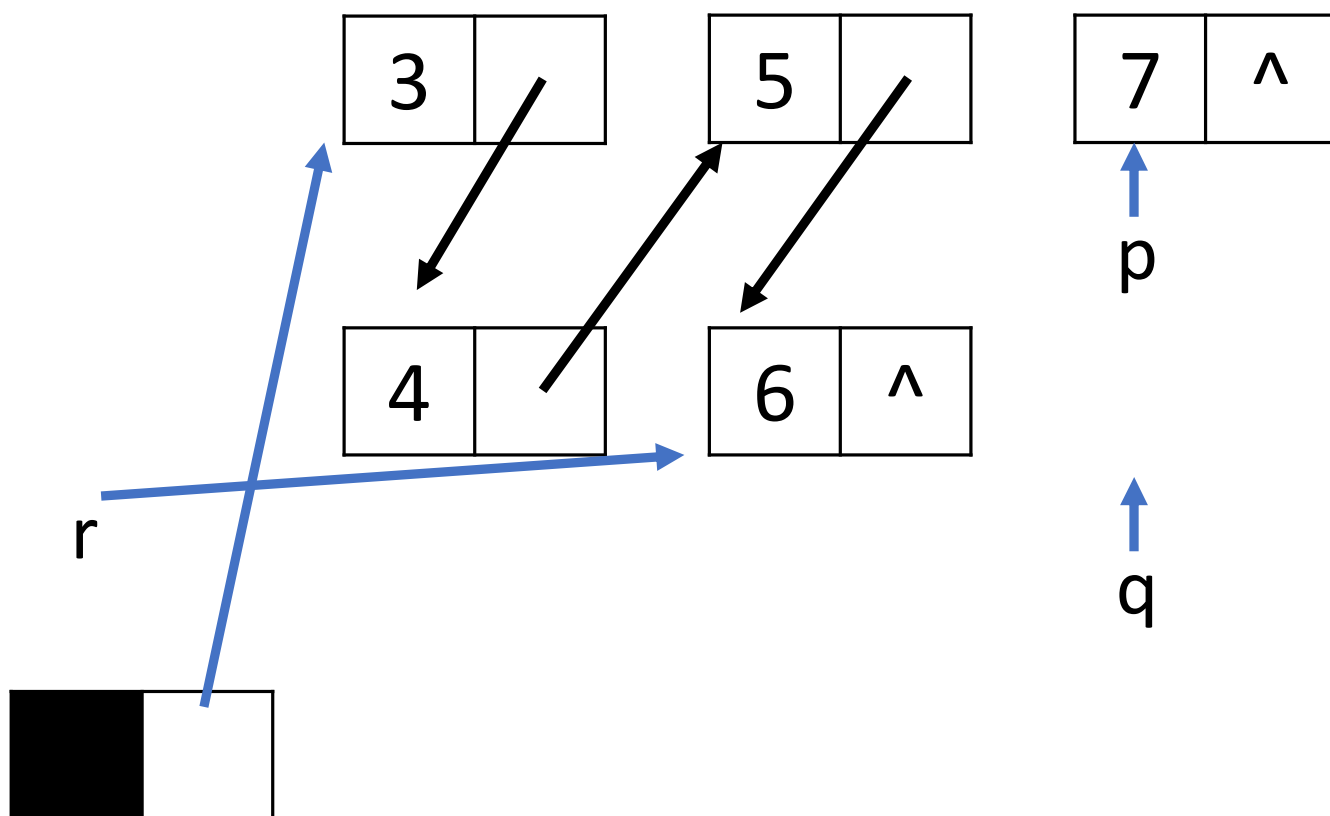
- 用2个指针p和q分别指向2个链表开头。为合并后的链表创建一个虚拟头结点head，用指针r指向新链表的尾结点
- 如果p和q都不空，将小的结点挂到新链表r的后面，更新r指向这个结点



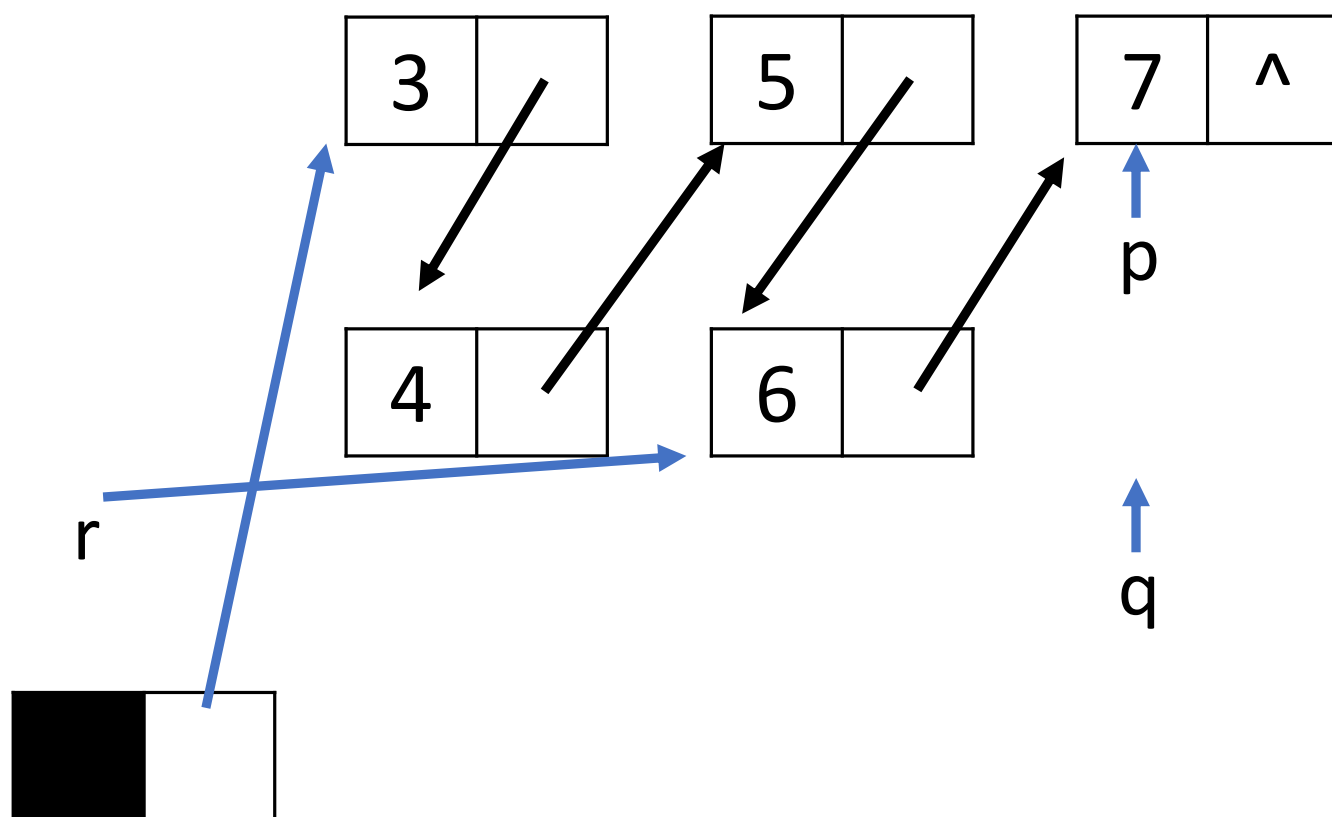
- 用2个指针p和q分别指向2个链表开头。为合并后的链表创建一个虚拟头结点head，用指针r指向新链表的尾结点
- 如果p和q都不空，将小的结点挂到新链表r的后面，更新r指向这个结点



- 用2个指针p和q分别指向2个链表开头。为合并后的链表创建一个虚拟头结点head，用指针r指向新链表的尾结点
- 如果p和q都不空，将小的结点挂到新链表r的后面，更新r指向这个结点



- 用2个指针p和q分别指向2个链表开头。为合并后的链表创建一个虚拟头结点head，用指针r指向新链表的尾结点
- 如果p和q都不空，将小的结点挂到新链表r的后面，更新r指向这个结点
- 将剩余链表挂到新链表的最后。



```
ListNode* mergeLists(ListNode *P, ListNode *Q) {  
    // 虚拟头结点  
    ListNode *head = new ListNode(), *r = head;  
    ListNode *p = P, *q = Q;  
    while (p && q) {  
        // 将p和q指向结点值较小的的结点挂到r指向结点后面  
        if (p->val < q->val) {  
            r->next = p;  
            p = p->next;  
        } else {  
            r->next = q;  
            q = q->next;  
        }  
        r = r->next; //更新新链表的尾指针  
    }  
    if (p)    r->next = p;  
    if (q)    r->next = q;  
    p = head->next; delete head;  
    return p;  
}
```



```
class ListNode{
public:
    int val;
    ListNode *next;
    ListNode(int val=0){this->val = val; next = nullptr;}
};
```

```
int main(){
    ListNode *P = new ListNode(3);
    P->next = new ListNode(5);
    P->next->next = new ListNode(7);
    ListNode *Q = new ListNode(4);
    Q->next = new ListNode(6);
    ListNode *H = mergeLists(P,Q);
    for(ListNode *p = H; p ;p = p->next)
        cout<<p->val<<" ";
    cout<<endl;
}
```

关注我

博客：hwdong-net.github.io

Youtube频道：[hwdong](#)



hwdong
2.05K subscribers

CUSTOMIZE CHANNEL

MANAGE VIDEOS

HOME

VIDEOS

PLAYLISTS

COMMUNITY

CHANNELS

ABOUT

