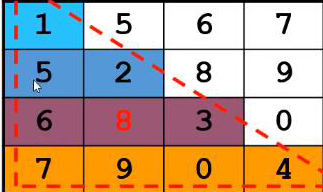
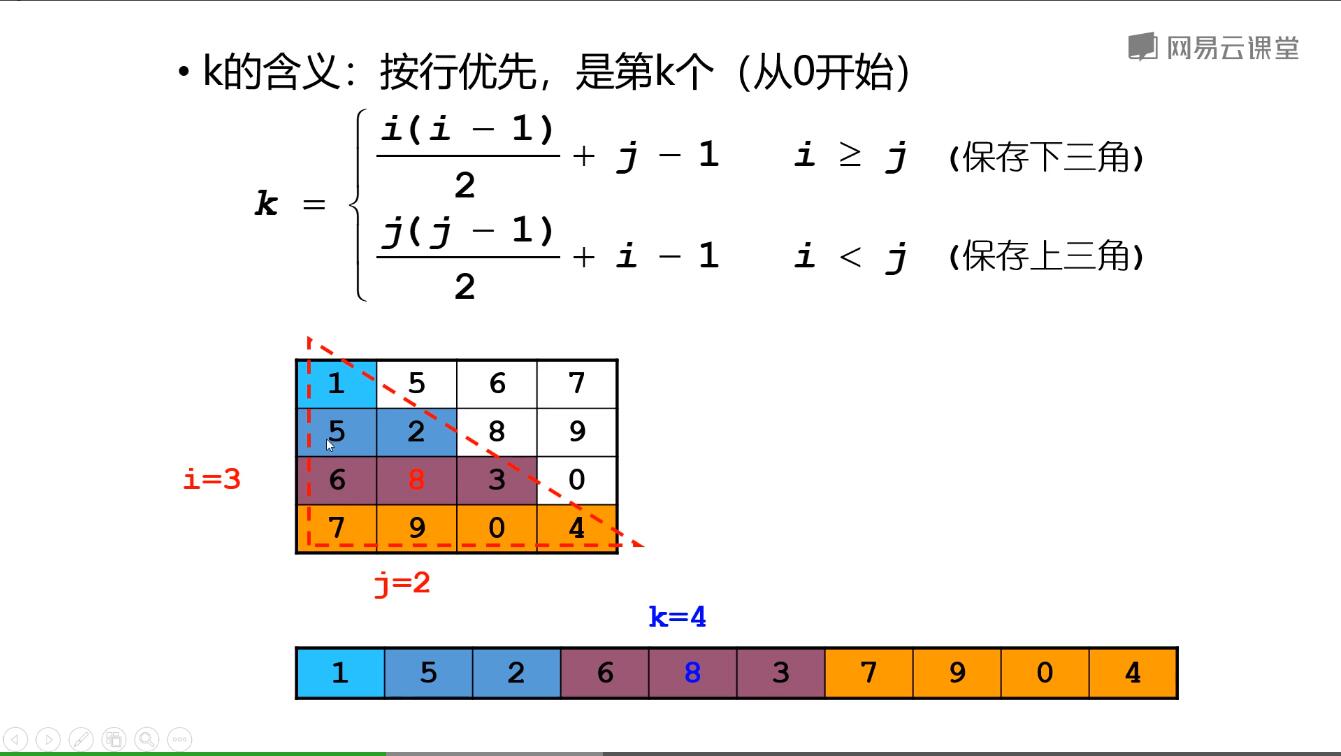
**对称矩阵的压缩**

* 1. 对称矩阵与压缩



* + 1. 我们可以想一下，如果用整个数组来放置，就会浪费挺多空间。因为有得元素是想等得呀
    2. 所以我们得做法是只存取一半相等得元素

1. 我们按行优先
2. 下三角



1. 看上面得图我们应该很明确了吧
   1. 比如5 一个是在(2, 1) 和在(1, 2)
   2. 所以我们按照下三角来存储得话，那么，我们应该怎么来存储呢
2. 按行，我们第一个元素就放置1
3. 第二行，第二个元素放5，第三个元素放2
4. …………
   1. 那我们怎么进行取值到对应的值呢，比如我要取（2，1）和 （1，2），我们知道是5
5. （3，1）我们可以得出前面应该是有1+2=3个，其实仔细一看是个等差数列，有没有，所以我们就推出了上面的公式

我们的下标可以换成自己喜欢的形式

1. 动态存储：
   1. int \*data = new int[n] 动态开辟内存
   2. 如果到内存不够时，还能进行内存的追加其实就是重新开辟一块，再把原来的元素搬过去就行了（^。^）

注：现在我们就用动态的来行操作

1. 具体操作
   * 1. 类的构建：

#include <iostream>

#include <stdlib.h>

#include <assert.h>

#include <string>

template <typename T>

class SqList{

T \*data;

int capacity, n;

public:

//初始化列表

SqList(int cap = 3){

data = new T[cap];

if(!data) throw "数据初始化失败";

capacity = cap;

n = 0;

}

//获取特定点值，我们这边的下表用人的正常思维，而不是从0快开始的，千万注意

bool get(int i, T &e){

assert(i >= 0 && i <= n);

e = data[i-1];

return true;

}

//往最后的位置插值

bool push\_back(T e){

if(n == capacity){

realloc();

}else{

data[n++] = e;

return true;

}

}

//删除最后一个元素

bool pop\_back(){

if(!data) return false;

// 因为我不需要返回回去，所以下面的被我注释掉了

// T e = data[n-1];

// delete e;

n--;

return true;

}

//在特定位置插入元素

bool insert(int pos, T e){

if(n == capacity){

realloc();

}

for(int i = n; i >= pos-1; i--){

data[i] = data[i-1];

}

data[pos-1] = e;

n++;

return true;

}

//修改某个位置的值

bool setVal(int pos, T e){

data[pos-1] = e;

return true;

}

//移除掉特点位置的值

bool remove(int pos){

for(int i = pos;i <= n; i++){

data[i-1] = data[i];

}

n--;

}

//玩一下，构造一下函数指针,那我们在创建一个函数，来比较一下把

int find(int pos,T e, bool(\*cmp)(T e1, T e2)){

for(int i = pos; i <= n; i++){

if(cmp(data[i-1], e)){

return i;

}

}

return -1;

}

//获取列表的长度

int getSize(){

return n;

}

private:

//内存不够，添加内存

bool realloc(){

T \*p = new T[capacity\*2];

if(!p) return false;

for(int i = 0; i < n; i++){

p[i] = data[i];

}

delete[] data; // 删除的是链表，也就是数组的

data = p; // 重新把p的头指针指向data

capacity \*= 2;

}

};

using std::string;

//构建student结构体，传进去template模板

struct student{

string name;

float score;

student(string n="no", float s=0.0):name(n), score(s){}

};

//打印函数

template <typename T>

void printArr(SqList<T> &L){

T e;

for(int i = 1; i <= L.getSize(); i++){

L.get(i, e);

std::cout << e << " ";

}

std::cout << std::endl;

}

//我们来对printArr进行重载一下，让他支持结构体的输出，当然我们也可以写一个专门的输出的函数

std::ostream& operator<<(std::ostream &o, student &s){

o << s.name << " ->" << s.score << std::endl;

}

//比较函数，当函数指针来用

template <typename T>

bool cmp(T e1, T e2){

if(e1 != e2)

return false;

return true;

}

int main() {

SqList<char> list;

char ch;

//在最后面添加元素，像vector数组一样

list.push\_back('A');printArr(list);

list.push\_back('B');printArr(list);

list.push\_back('C');printArr(list);

list.push\_back('D');printArr(list);

list.push\_back('E');printArr(list);

list.push\_back('E');printArr(list);

//在2位置插入元素

list.insert(2, 'X');printArr(list);

//把3位置的值重新赋值

list.setVal(3, 'Y');printArr(list);

//删除4位置的元素

list.remove(4);printArr(list);

list.pop\_back();printArr(list);

//这就是我们用到函数指针的地方

int pos = list.find(1, 'y', cmp);

if(pos >= 0){

std::cout << pos;

}else{

std::cout << "找不到该值";

}

std::cout << "--------------------------" << std::endl;

//利用结构体student当模板

SqList<student> students;

students.push\_back(student("weierlin", 98));

students.push\_back(student("xiaolong", 70.9));

/\*\*

\*我们之前值需要接收一个参数并打印出来

\*但是现在我们需要打印分数和名字

\*所以在前面要进行重载

\*\*/

printArr(students);

return 0;

}

运行结果：自行对比一下

