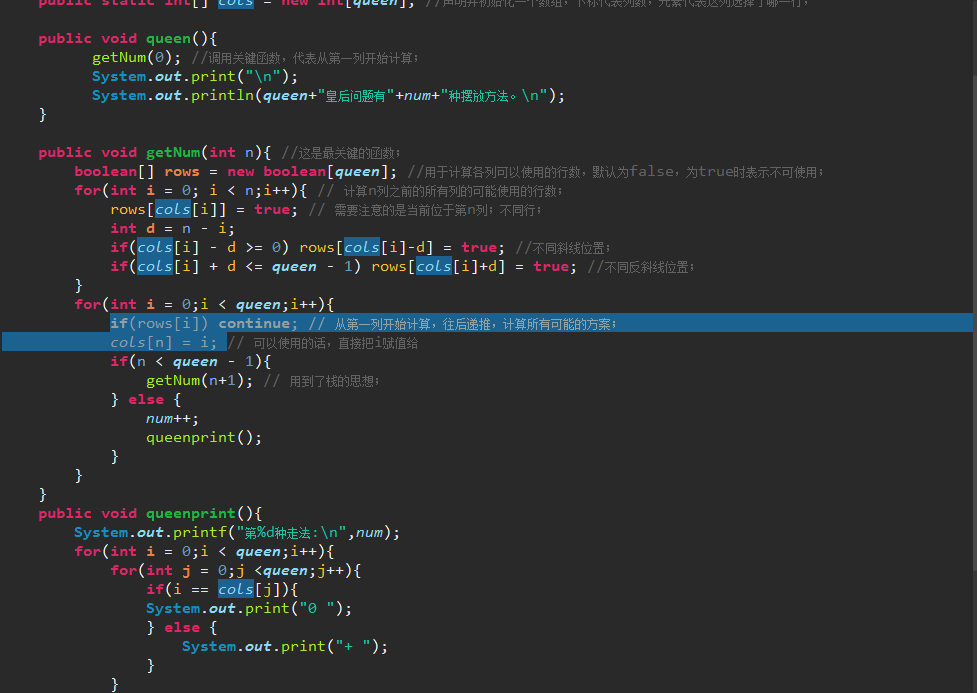
⑧皇后算法：

**核心思想：**设置两个数组，一个数组为cols[queen]表示第几列选取的是第几行(即第几个下标的元素是什么);另一个数组为rows[queen]，需要注意的是，这个rows数组会被重复创建queen次，第一次创建表示的是第一列的各行的取值变化，false表示可以取改行，true表示不可以；使用迭代，来判断rows的元素取值，判断条件即为限制条件；使用递归来判断cols的取值，当queen个元素的取值都确定后，即为一个解；这里的rows数组会在递归时被压入操作数栈中，当被递归的方法执行完毕后。会调用本地方法的rows数组；cols的元素确定是递归的，比如cols[queen-1]没有合适的取值时，（是指第queen次创建的rows的数组，元素皆为true，或者为false的元素都已经被使用）使用的方法是把为false的元素的下标赋值给cols的下标n，注意，该row第n+1次创建），则返回cols[queen-2]换下一个false，然后再对cols[queen-1]取值，如果cols[queen-2]还没有合适的值，则返回cols[queen-3]换下一个false，然后对cols[queen-2]取值，等等；下图阴影处即为核心中的核心：



public class eight\_queen {

public static int num = 0; //声明并初始化一个变量，用于计算解法方案的个数；

public static final int queen = 8; //给定的皇后行列数；

public static int[] cols = new int[queen]; //声明并初始化一个数组，下标代表列数，元素代表这列选择了哪一行；；

public void queen(){

getNum(0); //调用关键函数，代表从第一列开始计算；

System.out.print("\n");

System.out.println(queen+"皇后问题有"+num+"种摆放方法。\n");

}

public void getNum(int n){ //这是最关键的函数；

boolean[] rows = new boolean[queen]; //用于计算各列可以使用的行数，默认为false，为true时表示不可使用；

for(int i = 0; i < n;i++){ // 计算n列之前的所有列的可能使用的行数；

rows[cols[i]] = true; // 需要注意的是当前位于第n列；不同行；

int d = n - i;

if(cols[i] - d >= 0) rows[cols[i]-d] = true; //不同斜线位置；

if(cols[i] + d <= queen - 1) rows[cols[i]+d] = true; //不同反斜线位置；

}

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

if(rows[i])continue; // 从第一列开始计算，往后递推，计算所有可能的方案；

cols[n] = i; // 可以使用的话，直接把i赋值给

if(n < queen - 1){

getNum(n+1); // 用到了栈的思想；

} else {

num++;

queenprint();

}

}

}

public void queenprint(){

System.out.printf("第%d种走法:\n",num);

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

for(int j = 0;j <queen;j++){

if(i == cols[j]){

System.out.print("0 ");

} else {

System.out.print("+ ");

}

}

System.out.print("\n");

}

}

public static void main(String[] args){

eight\_queen queen = new eight\_queen();

queen.queen();

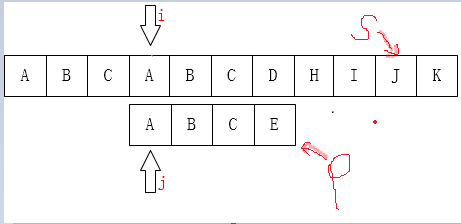
}

}

KMP字符串匹配算法：

**核心思想**：字符串匹配算法的核心是next数组的计算；在本代码中，next数组使用的是-1开头的，避免了一般next数组需要的使用字符串P-next[j]方法；本代码中可以直接使用next[j]为j赋值，本代码中的next数组的元素为下标j之前的子串最大的重合部分，比如next[6],则为P字符串前6个元素中，最大的重合部分，例：P= “asxaasdfg”，next[6]=2，P的前6个元素为asxaas，重合部分为as；如果在j=6的时候，匹配不成功，则可以直接使j=next[j]=2,即如果‘d’没有和S中的元素匹配成功，则不需要移动i，直接移动j到字符串P的第next[j]个元素即可，在本例中，是d没有匹配成功，则移到元素x位置，继续匹配；再详细一点，就是因为asx与asd中前两元素是一样的，所以，你懂得。BF算法则是你能想到的那种最直接的算法。本例中的KMP算法，计算next数组花费了大量资源，可能需要改进。S和P，以及i与j的含义如下：

S为母串；S为子串；i为S的下标；j为P的下标；



import java.util.Arrays;

import java.util.Scanner;

public class KMP {

public static void kmp(String S, String P, int[] next){

int i = 0;

int j = 0;

int Sl = S.length();

int Pl = P.length();

while(i < Sl && j < Pl){

if(j == -1 || S.charAt(i)== P.charAt(j)){

i++;

j++;

} else {

j = next[j];

}

}

if (j == Pl) {

System.out.printf("匹配成功，从字符串的第%d个开始匹配;\n", i-j+1);

return;

} else {

System.out.printf("匹配不成功\n");

return;

}

}

//next数组计算方式

public static int[] next(String P){

int[] next = new int[P.length()];

int j = 0;

int k = -1;

next[0] = -1;

while(j < P.length()-1){

if(k == -1 || P.charAt(j) == P.charAt(k)){

next[++j] = ++k;

}

else {

k = next[k];

}

}

System.out.printf("next数组为%s\n", Arrays.toString(next));

return next;

}

public static void main(String[] args){

Scanner scanner = new Scanner(System.in);

System.out.printf("请输入两个字符串，用回车键分割:\n");

String S = scanner.nextLine();

String P = scanner.nextLine();

Long time1 = System.nanoTime();

int[] next = KMP.next(P);

KMP kmp = new KMP();

kmp.kmp(S,P,next);

Long time2 = System.nanoTime();

System.out.printf("KMP匹配方法需要的时间是"+(time2-time1)+"ns;\n");

Long time3 = System.nanoTime();

KMP\_basic kmp\_basic = new KMP\_basic();

kmp\_basic.basic(S,P);

Long time4 = System.nanoTime();

System.out.printf("BF匹配方法需要的时间是"+(time4-time3)+"ns;\n");

}

}

附注：BF算法：

import java.util.Scanner;

public class KMP\_basic {

public void basic(String S,String P){

int sl = S.length();

int pl = P.length();

char[] scha = S.toCharArray();

char[] pcha = P.toCharArray();

int i,j = 0;

for(i = 0;i < sl;){

if(scha[i] == pcha[j]){

i++;

j++;

} else{

i = i-j+1;

j = 0;

}

if(j == (pl-1)){

System.out.printf("字符串%s的第%d与%d之间的字符穿与字符串%s相同\n", S,(i-j+1),(i+1),P);

// if(i == (sl-1)){

return;

// } else{

// i = i-j+1;

// j = 0;

// }

}

}

System.out.printf("两个字符串没有相同的部分\n");

return;

}

public static void main(String args[]){

Scanner scanner = new Scanner(System.in);

System.out.printf("请输入两个字符串，用回车键分割:\n");

String S = scanner.nextLine();

String P = scanner.nextLine();

Long time1 = System.nanoTime();

KMP\_basic kmp\_basic = new KMP\_basic();

kmp\_basic.basic(S,P);

Long time2 = System.nanoTime();

System.out.printf("暴力匹配方法需要的时间是"+(time2-time1)+"ns");

}

}