**算法第二次作业**

2.2

（1）

① 这题完成的是求1到n的平方和的功能

② 基本语句为：s+=i\*i

③ 执行了n次

④ 因为算法主体是一个for循环，而循环的时间复杂度等于循环体的复杂度乘以该循环运行的次数，并且循环体的基本语句是时间复杂度为O(1)的程序步骤序列，故该算法时间复杂度O（n）

（2）

① 完成的是n的平方

② 基本语句：return Q(n-1) + 2 \* n–1，

③ 执行了n次，因为n每次循环减1，一直减到n=1才结束

④ 因为执行n次基本语句，所以时间复杂度为O（n）

2.3

（1）

基本语句为2\*i<n和y = y + i \* j

语句2\*i<n执行了n次

语句y = y + i \* j执行了(n^2)/4次

（2）

基本语句m=m+1

执行(n+1)\*n=n²+n次

时间复杂性为O(n²)

2.4

(1)

int T(int n){if(n==1)return 4;else if(n>1)return 3\*T(n-1);}

T(n)=3T(n-1)=3(3T(n-2))=3(3(3T(n-3)))=3n-1

所以 T(n)=4X3n-1

(2)

int T(int n){if(n==1)return 1;else if(n>1)return 2\*T(n/3)+n;}

T(n)=2T(n/3)+n=2(2T(n/9)+n/3)+n=2(2(2T(n/3 3)+n/9)+n/3)+n

=2kT(1)+2k-1X(n/3k-1)+...+2X(n/3)+n

= 3\*n-2\*2k

= 3\*n-2\*

**3.1**

**（1）BF算法**

**伪代码：**

输入：序列S，T

输出：第一次出现的下标

1. 首先设定 S 和 T 的起始比较下标 i 和 j；  
2. 循环直到 i+m>n 或者T中的字符都比较完(j==m)

2.1 如果S[i]==T[j], 继续比较S和T的下一个字符，否则

2.2 将 i 和 j 回溯，准备下一轮比较

3. 如果T中的字符都比较完(j==m)，则返回比较的起始下标  
    否则返回-1，表示匹配失败

3. 如果T中的字符都比较完(j==m)，则返回比较的起始下标  
    否则返回-1，表示匹配失败

**实现：**

#include<iostream>

using namespace std;

int BF(char S[], char T[]){

int index = 0;

int i = 0, j = 0;

while ((S[i] != '\0') && (T[j] != '\0')){

if (S[i] == T[j]){

i++;

j++;

}

else {

++index;

i = index;

j = 0;

}

}

if (T[j] == '\0')

return index + 1;

else

return 0;

}

int main(){

char s1[19]="siemsingqyuenshehge";

char s2[8]="gqyuesss";

cout<< BF( s1, s2) <<endl;return 0;

}

**（2）KMP算法**

**伪代码：**

输入：序列S，T

输出：第一次出现的下标

1.首先初始化好NEXT数组, next[0]=0,next[1]=1

2.循环查找模式P是否在T中

1）首先比较P[i] == T[j]，如果相等，继续比较下一个，否则执行2.2）

2）令j=next[j]，继续比较（这一步避免了回溯）

3）如果j==0; 表明没有匹配，则i++, j++

3.直到找到P在T中的位置或者T已经被比较晚结束。

3.直到找到P在T中的位置或者T已经被比较晚结束。

    否则返回-1，表示匹配失败

3. 如果T中的字符都比较完(j==m)，则返回比较的起始下标  
    否则返回-1，表示匹配失败

next数组的初始化

1.定义next数组, 令next[0]=0, next[1]=1

2.从p[2]开始循环计算对应的next数组

3.循环计算next[j]的值

4.从next[j]往前找到某个p[i]=p[next[i]]，如果相等则next[j]=next[i]+1

5.否则，令i=next[i]继续向前查找，直到找到相等的为止。

6.        如果i=0，则表明模式P中没有p[j]相同的前缀，令next[j]=1

**实现：**

#include<iostream>

using namespace std;

void GetNext(char T[ ], int next[ ]){

int i, j, len;

next[0] = -1;

for (j = 1; T[j]!='\0'; j++){

for (len = j - 1; len >= 1; len--){

for (i = 0; i < len; i++)

if (T[i] != T[j-len+i])

break;

if (i == len){

next[j] = len; break;

}

}

if (len < 1)

next[j] = 0;

}

}

int KMP(char S[ ], char T[ ]) {

int i = 0, j = 0;int next[80];

GetNext(T, next);

while (S[i] != '\0' && T[j] != '\0'){

if (S[i] == T[j]){

i++; j++;}

else{j = next[j];

if (j == -1) {

i++; j++;}

}

}

if (T[j] == '\0')

return (i - strlen(T) +1);

else

return 0;

}

int main(){

char s1[]="dsahihdashdadsdadas";

char s2[]="ihdas";

cout<<KMP(s1[],s2[])<<endl;

return 0;

}

**3.6**

输入：数组r[n]，数组中元素个数n，待删除元素x

输出：删除x后的数组

1. i=0,j=n-1
2. 循环直到i==j
   1. 循环直到r[j]不等于x
      1. j--;n--;
   2. 循环直到r[i]==x
      1. r[i]=r[j];n--;j--;

**3.10**

输入：k维空间上的n个点

输出：n个点的最近距离

1. minDist=INF
2. i=0，循环直到i==n-1
   1. j=i+1，循环直到j==n
      1. 计算i与j之间的距离d，如果小于minDist, 则令minDist=d;
3. 返回minDist

**实现：**

**时间复杂度为O（n2）**

int distance (int p1[],int p2[])

{

int distance=0;

int temp=0;

int k;

printf("请输入要计算的点在几维空间：注意k<50 \n");

scanf("%d",&k);

if(k>50)

printf("您输入的范围过大\n");

printf("请输入点的坐标P1 X[50]\n");

for(i=0;i<k;i++)

scanf("%d",&x[i]);

printf("请输入点的坐标P2 y[50]\n");

for(i=0;i<k;i++)

scanf("%d",&y[i]);

for(i=0;i<k;i++){

temp=(y[i]-x[i])\*(y[i]-x[i])

distance+=temp;

}

return distance ;

}

int shortdistance(int n)

{

int i,j;

int n ;

int p1[],int p2[];

int d=1000;

for(i=1;i<n+1;i++){

distance(p1[],p2[]);

if(distance<d)

d=distance;

}

return d;

}