2021 上

阅读下列说明和代码，回答问题1和问题2，将解答写在答题纸的对应栏内。

【说明】

凸多边形是指多边形的任意两点的连线均落在多边形的边界或内部。相邻的点连线落在多边形边界上，称为边；不相邻的点连线落在多边形内部，称为弦。假设任意两点连线上均有权重，凸多边形最优三角剖分问题定义为：求将凸多边形划分为不相交的三角形集合，且各三角形权重之和最小的剖分方案。每个三角形的权重为三条边权重之和。

假设N个点的凸多边形点编号为$V_1,V_2,\ldots\ldots,V_N$,若在V_k处将原凸多边形划分为一个三角形$V_1V_kV_N$，两个子多边形$V_1,V_2,\ldots,V_k$和$V_k,V_{k+1},\ldots V_N$，得到一个最优的剖分方案，则该最优剖分方案应该包含这两个子凸边形的最优剖分方案。用m[i][j]表示带你$V_{i-1},V_i,\ldots V_j$构成的凸多边形的最优剖分方案的权重，S[i][j]记录剖分该凸多边形的k值。

则

$$m[i][j]=\begin{cases}0, i \geq j \\ \min\limits_{i\leq k<j}\{m[i][k]+m[k+1][j]+W(V_{i-1}V_kV_j)\}, i<j\end{cases}$$

其中：$W(V_{i-1}V_kV_j)=W_{i-1,k}+W_{k,j}+W_{j,i-1}$为三角形$V_{i-1}V_kV_j$的权重，$W_{i-1,k},W_{k,j}$，$W_{j,i-1}$分别为该三角形三条边的权重。求解凸多边形的最优剖分方案，即求解最小剖分的权重及对应的三角形集。

4.数据结构与算法应用真题

[C代码]

```
#include < stdio.h >
#define N 6 //凸多边形规模
int m[N + 1] [N + 1]; //m[i][j]表示多边形V(i-1)到Vj最优三角剖分的权值
int S[N + 1] [N + 1];  //S[i][j]记录多边形V(i-1)到Vj最优三角剖分的k值
int W[N + 1] [N + 1];  //凸多边形的权重矩阵，在main函数中输入
/*三角形的权重a，b，c，三角形的顶点下标*/
int get_ triangle_weight（int a，int b，int c）
{
    return W[a][b] + W[b][c] + W[c][a];
}
/*求解最优值*/
void triangle_partition(){
int i,r,k,j;
int temp;
/*初始化*/
for(i=1;i<=N;i++){
m[i][i]=0;
}
```

4.数据结构与算法应用真题

```
/*求解最优值*/
void triangle_partition(){
int i,r,k,j;
int temp;
/*初始化*/
for(i=1;i<=N;i++){
m[i][i]=0;
}
/*自底向上计算m，S*/
for(r=2;（1）;r++)
{   /*r为子问题规模*/
    for(i=1;k<=N-r+1;i++)
   {
      (2) ;
      m[i][j]= m[i][j]+m[i+1][j]+get_triangle_weight(i-1,i,j); /*k=j*/
      S[i][j]=i;
      for(k=j+1;k<j;k++)
      {   /*计算 [i][j]的最小代价*/
          temp=m[i][k]+m[k+1][j]+ge_triangle_ weight(i-1,k,j);
          if((3))
            {      /*判断是否最小值*/
                   m[i][j]=temp;
                   S[i][j]=k;
            }
      }
   }
}
}
```

4.数据结构与算法应用真题

```
/*输出剖分的三角形i，j：凸多边形的起始点下标*/
void print_triangle(int i,int j){
if(i==j) return;
print_triangle(i,S[i][j]);
print_ triangle((4));
print("V%d- -V%d- -V%d\n",i-1,S[i][j],j);
}
```

【问题1】（8分）
根据题干说明，填充C代码中的空（1）~（4）。
【问题2】（7分）
根据题干说明和C代码，该算法采用的设计策略为（5）。
算法的时间复杂度为（6），空间复杂度为（7）（用O表示）

2021 下

生物学上通常采用编辑距离来定义两个物种DNA序列的相似性,从而刻画物种之间的进化关系。具体来说，编辑距离是指将一个字符串变换为另一个字符串所需要的最小操作次数。操作有三种,分别为:插入一个字符、删除一个字符以及将一个字符修改为另一个字符。用字符数组str1和str2分别表示长度分别为len1和len2的字符串，定义二维数组d记录求解编辑距离的子问题最优解，则该二维数组可以递归定义为:

$$d[i][j]=\begin{cases} i & 若len2=0 \\ j & 若len1=0 \\ d[i-1][j-1] & 若str[i-1]=str2[j-1] \\ \min\{d[i-1][j]+1,d[i][j-1]+1,d[i-1][j-1]+1\} & 若str[i-1]\neq str2[j-1] \end{cases}$$

【C代码】

下面是算法的C语言实现。

（1）常量和变量说明

A，B：两个字符数组

d：二维数组

i，j：循环变量

temp：临时变量

4.数据结构与算法应用真题

（2）C程序

```
#include <stdio.h>
#define N 100
char A[N]="CTGA";
char B[N]="ACGCTA";
int d[N][N];
int min(int a, int b){
return a <b ? a: b;
}
int editdistance(char *str1, int len1, char *str2, int len2){
int i, j;
int diff;
int temp;
for(i=0; i<=len1; i++){
d[i][0]=i;
}
for(j=0; j<=len2; j++){
__(1)__;
}
for(i=1;i<=len1;i++){
for(j=1; j<=len2; j++){
if(__(2)__){
d[i][j]=d[i-1][j-1];
}else{
temp=min(d[i-1][j]+1, d[i][j-1]+1);
d[i][j]=min(temp,__(3)__);
}
}
}
return __(4)__;
}
```

【问题1】(8分)
根据说明和C代码，填充C代码中的空(1)~(4)。
【问题2】(4分)
根据说明和C代码，算法采用了(5)设计策略，时间复杂度为(6)(用O符号表示，两个字符串的长度分别用m和n表示)。
【问题3】(3分)
已知两个字符串A="CTGA"和B="ACGCTA"，根据说明和C代码，可得出这两个字符串的编辑距离为(7)。

2020

【说明】

希尔排序算法又称最小增量排序算法，其基本思想是：

步骤 1：构造一个步长序列 delta1、delta2…、deltak，其中 delta1=n/2，后面的每个 delta 是前一个的 1/2 ， deltak=1；

步骤 2：根据步长序列、进行 k 趟排序；

步骤 3：对第 i 趟排序，根据对应的步长 delta，将等步长位置元素分组，对同一组内元素在原位置上进行直接插入排序。

【C 代码】

下面是算法的 C 语言实现。

（1）常量和变量说明

data：待排序数组 data，长度为 n，待排序数据记录在 data[0]、data[1]、…、data[n-1]中。

n：数组 a 中的元素个数。

delta：步长数组。

（2）C 程序

4.数据结构与算法应用真题

```
#include <stdio.h>
void shellsort(int data[ ], int n){
   int *delta,k,i,t,dk,j;
   k=n;
   delta=(int *)nalloc(sizeof(int)*(n/2));
   if(i=0)
      do{
         ( 1 ) ;
         delta[i++]=k;
      }while ( 2 ) ;
   i=0;
   while((dk=delta[i])>0){
      for(k=delta[i];k<n;++k)
      if( ( 3 ) ) {
         t=data[k];
         for(j=k-dk;j>=0&&t<data[j];j-=dk){
            data[j+dk]=data[j];
         }/*for*/
      ( 4 ) ; //data[j+dk]=t;
      }/*if*/
      ++i;
   }/*while*/
}
```

【问题 1】（8 分）

根据说明和 c 代码，填充 c 代码中的空（1）~ （4）。

【问题 2】（4 分）

根据说明和 c 代码，该算法的时间复杂度（5）O(n) （小于、等于或大于）。该算法是否稳定（6）（是或否）。

【问题 3】（3 分）

对数组（15、9、7、8、20、-1、 4）用希尔排序方法进行排序，经过第一趟排序后得到的数组为（7）。

2017 上

阅读下列说明和C代码，回答问题 1 至问题 3，将解答写在答题纸的对应栏内。

【说明】

假币问题：有n枚硬币，其中有一枚是假币，已知假币的重量较轻。现只有一个天平，要求用尽量少的比较次数找出这枚假币。

【分析问题】

将n枚硬币分成相等的两部分：

（1）当n为偶数时，将前后两部分，即 1…n/2和n/2+1…n，放在天平的两端，较轻的一端里有假币，继续在较轻的这部分硬币中用同样的方法找出假币；

（2）当n为奇数时，将前后两部分，即1..(n -1)/2和(n+1)/2+1…n，放在天平的两端，较轻的一端里有假币，继续在较轻的这部分硬币中用同样的方法找出假币；若两端重量相等，则中间的硬币，即第 (n+1)/2枚硬币是假币。

【C代码】

下面是算法的C语言实现，其中：

coins[]： 硬币数组

first，last：当前考虑的硬币数组中的第一个和最后一个下标

4.数据结构与算法应用真题

```
#include <stdio.h>

int getCounterfeitCoin(int coins[],  int first，int last)
{
    int firstSum = 0，lastSum = 0;
    int i;
    if(first==last-1){       /*只剩两枚硬币*/
        if(coins[first] < coins[last])
            return first;
        return last;
    }

if((last - first + 1) % 2 ==0){       /*偶数枚硬币*/
    for(i = first;i <(   1   );i++){
        firstSum+= coins[i];
    }
    for(i=first + (last-first) / 2 + 1;i < last +1;i++){
        lastSum += coins[i];
    }
    if(   2   ){
        return getCounterfeitCoin(coins,first,first+(last-first)/2;)
    }else{
        return getCounterfeitCoin(coins,first+(last-first)/2+1,last;)
    }
}
else{      /*奇数枚硬币*/
    for(i=first;i<first+(last-first)/2;i++){
        firstSum+=coins[i];
    }
    for(i=first+(last-first)/2+1;i<last+1;i++){
        lastSum+=coins[i];
    }
    if(firstSum<lastSum){
        return getCounterfeitCoin(coins,first,first+(last-first)/2-1);
    }else if(firstSum>lastSum){
        return getCounterfeitCoin(coins,first+(last-first)/2+1,last);
    }else{
        return(   3   )
    }
  }
}
```

2014 上

阅读下列说明和C代码，回答问题1至问题3，将解答写在答题纸的对应栏内。

【说明】

采用归并排序对n个元素进行递增排序时，首先将n个元素的数组分成各含n/2个元素的两个子数组，然后用归并排序对两个子数组进行递归排序，最后合并两个已经排好序的子数组得到排序结果。

下面的C代码是对上述归并算法的实现，其中的常量和变量说明如下：

arr：待排序数组

p,q,r：一个子数组的位置从p到q，另一个子数组的位置从q+1到r

begin,end：待排序数组的起止位置

left,right：临时存放待合并的两个子数组

n1,n2：两个子数组的长度

i,j,k：循环变量

mid：临时变量

4.数据结构与算法应用真题

```
#inciude<stdio.h>
#inciude<stdlib.h>
#define MAX 65536
void merge(int arr[],int p,int q,int r) {
   int *left, *right;
   int n1,n2,i,j,k;
   n1=q-p+1;
   n2=r-q;
   if((left=(int*)malloc((n1+1)*sizeof(int)))=NULL) {
      perror("malloc error");
      exit(1);
   }
   if((right=(int*)malloc((n2+1)*sizeof(int)))=NULL) {
      perror("malloc error");
      exit(1);
   }
   for(i=0;i<n1;i++){
      left[i]=arr[p+i];
   }
   left[i]=MAX;
   for(i=0; i<n2; i++){
      right[i]=arr[q+i+1]
   }
   right[i]=MAX;
   i=0; j=0;
   for(k=p; __(1)__; k++) {
      if(left[i]> right[j]) {
         __(2)__;
         j++;
      }else {
         arr[k]=left[i];
         i++;
      }
   }
}
void mergeSort(int arr[],int begin,int end){
   int mid;
   if(__(3)__){
      mid=(begin+end)/2;
      mergeSort(arr,begin,mid);
      __(4)__;
      merge(arr,begin,mid,end);
   }
}
```

【问题1】（8分）

根据以上说明和C代码，填充1-4。

【问题2】（5分）

根据题干说明和以上C代码，算法采用了（5）算法设计策略。

分析时间复杂度时，列出其递归式位（6），解出渐进时间复杂度为（7）（用O符号表示）。空间复杂度为（8）（用O符号表示）。

【问题3】（2分）

两个长度分别为n1和n2的已经排好序的子数组进行归并，根据上述C代码，则元素之间比较次数为（9）。

2019 上

阅读下列说明和C代码，回答问题1至问题3，将解答写在答题纸的对应栏内。

【说明】

n皇后问题描述为：在一个n×n的棋盘上摆放n个皇后，要求任意两个皇后不能冲突，即任意两个皇后不在同一行、同一列或者同一斜线上。

算法的基本思想如下：

将第i个皇后摆放在第i行，i从1开始，每个皇后都从第1列开始尝试。尝试时判断在该列摆放皇后是否与前面的皇后有冲突，如果没有冲突，则在该列摆放皇后，并考虑摆放下一个皇后；如果有冲突，则考虑下一列。如果该行没有合适的位置，回溯到上一个皇后，考虑在原来位置的下一个位置上继续尝试摆放皇后，……，直到找到所有合理摆放方案。

【C代码】

下面是算法的C语言实现。

（1）常量和变量说明

n: 皇后数，棋盘规模为n×n

queen[]: 皇后的摆放位置数组， queen[i]表示第i个皇后的位置， 1≤queen[i]≤n

（2）C程序

4.数据结构与算法应用真题

（2）C程序

```
#include <stdio.h>
#define n 4
int queen[n+1];

void Show(){    /* 输出所有皇后摆放方案 */
        int i;
        printf("(");
        for(i=1;i<=n;i++){
                printf(" %d",queen[i]);
        }
        printf(")\n");
}
int Place(int j){     /* 检查当前列能否放置皇后，不能放返回0，能放返回1 */
        int i;
        for(i=1;i<j;i++){   /* 检查与已摆放的皇后是否在同一列或者同一斜线上 */
                if( ( __(1)__ ) || abs(queen[i]-queen[j]) == (j-i) ) {
                        return 0;
                }
        }
        return ___(2)___ ;
}
        void Nqueen(int j){
                int i;
                for(i=1;i<=n;i++){
                        queen[j] = i;
                        if(_____(3)____){
                                if(j == n) {     /* 如果所有皇后都摆放好，则输出当前摆放方案 */
                                        Show();
                                } else {        /* 否则继续摆放下一个皇后 */
                                                __(4)__ ;
                                }
                        }
                }
}

int main(){
        Nqueen (1);
        return 0;
}
```

【问题1】（8分）

根据题干说明，填充C代码中的空（1）～（4）。

【问题2】（3分）

根据题干说明和C代码，算法采用的设计策略为__（5）__。

【问题3】（4分）

当n=4时，有__（6）__种摆放方式，分别为__（7）__。

2017 下

阅读下列说明和C代码，回答问题 1 至问题 2，将解答写在答题纸的对应栏内。

【说明】

一个无向连通图G点上的哈密尔顿（Hamilton）回路是指从图G上的某个顶点出发，经过图上所有其他顶点一次且仅一次，最后回到该顶点的路径。一种求解无向图上哈密尔顿回路算法的基本思想如下：

假设图G存在一个从顶点V0出发的哈密尔顿回路 V0 —— V1——V2——V3——...——Vn-1——V0。算法从顶点V0出发，访问该顶点的一个未被访问的邻接顶点V1，接着从顶点V1出发，访问V1一个未被访问的邻接顶点V2，...；对顶点Vi，重复进行以下操作：访问Vi的一个未被访问的邻接接点Vi+1；若Vi的所有邻接顶点均已被访问，则返回到顶点Vi-1，考虑Vi-1的下一个未被访问的邻接顶点，仍记为Vi；直到找到一条哈密尔顿回路或者找不到哈密尔顿回路，算法结束。

【C代码】

下面是算法的C语言实现。

（1）常量和变量说明

n :图G中的顶点数

c[ ] [ ]:图G的邻接矩阵

k:统计变量，当期已经访问的定点数为k+1

x［k］:第k个访问的顶点编号，从0开始

visited［x［k］］：第k个顶点的访问标志，0表示未访问，1表示已访问

4.数据结构与算法应用真题

```
#include <stido.h>
#include <stidb.h>
#define MAX 100

void Hamilton (int n,int x [MAX] , int c[MAX][MAX]) {
int i;
int visited[MAX];
int k;
 /*初始化x数组和visited数组*/
for (i=0:i<n;i++) {
    x[i]=0;
    visited [i]=0;
}
/*访问起始顶点*/
k=0
 (1) ;
x[0]=0 ;
k=k+1 ;
/*访问其他顶点*/
while (k>=0) {
   x[k]=x[k]+1;
   while (x[k]<n) {
      if ( (2) && c [x[k-1]] [x[k]] ==1) {/*邻接顶点x[k]未被访问过*/
        break;
      } else {
          x[k] = x[k] +1
      }
   }
    if (x[k] <n&&k==n-1&& (3) ) { /*找到一条哈密尔顿回路*/
       for (k=0;k<n;k++) {
            printf ( "%d-- ",x[k] ) ; /*输出哈密尔顿回路*/
       }
       printf ( "%d\n ",x[0] ) ;
       return;
} else if (x[k]<n&&k<n-1) {/*设置当前顶点的访问标志，继续下一个顶点*/
       (4)
       k=k+1;
} else {/*没有未被访问过的邻接顶点，回退到上一个顶点*/
      x[k]=0;
      visited [x[k]]=0;
       (5) ;
   }
}
}
```

【问题1】（10分）

根据题干说明。填充C代码中的空（1）~（5）。

【问题2】（5分）

根据题干说明和C代码，算法采用的设计策略为（6），该方法在遍历图的顶点时，采用的是（7）方法（深度优先或广度优先）。

2015 上

阅读下列说明和C代码，回答问题1至问题3，将解答写在答题纸的对应栏内。

【说明】

n-皇后问题是在n行n列的棋盘上放置n个皇后，使得皇后彼此之间不受攻击，其规则是任意两个皇后不在同一行、同一列和相同的对角线上。

拟采用以下思路解决n-皇后问题：第i个皇后放在第i行。从第一个皇后开始，对每个皇后，从其对应行（第i个皇后对应第i行）的第一列开始尝试放置，若可以放置，确定该位置，考虑下一个皇后；若与之前的皇后冲突，则考虑下一列；若超出最后一列，则重新确定上一个皇后的位置。重复该过程，直到找到所有的放置方案。

【C代码】

下面是算法的C语言实现。

（1）常量和变量说明

pos：一维数组，pos[i]表示第i个皇后放置在第i行的具体位置

count：统计放置方案数

i，j，k：变量

N：皇后数

（2）C程序

```
#include <stdio.h>

#include <math.h>

#define N4

/*判断第k个皇后目前放置位置是否与前面的皇后冲突*/

int isplace(int pos[], int k) {

    int i;

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

          if( __(1)__ || fabs(i-k) == fabs(pos[i] - pos[k])) {

            return 0;

          }

        }

        return 1;

}
```

4.数据结构与算法应用真题

```
int main() {
    int i,j,count=1;
    int pos[N+1];
    //初始化位置
    for(i=1; i<=N; i++) {
        pos[i]=0;
    }
    __(2)__ ;
    while(j>=1) {
        pos[j]= pos[j]+1;
        /*尝试摆放第j个皇后*/
        while(pos[j]<=N&& __(3)__ ) {
            pos[j]= pos[j]+1;
        }
        /*得到一个摆放方案*/
        if(pos[j]<=N&&j== N) {
            printf("方案%d: ",count++);
            for(i=1; i<=N; i++){
                printf("%d  ",pos[i]);
            }
            printf("\n");
        }
        /*考虑下一个皇后*/
        if(pos[j]<=N&&__(4)__ ) {
            j=j+1;
        } else{ //返回考虑上一个皇后
            pos[j]=0;
            __(5)__ ;
        }
    }
    return 1;
}
```

【问题1】（10分）

根据以上说明和C代码，填充C代码中的空（1）～（5）。

【问题2】（2分）

根据以上说明和C代码，算法采用了___（6）___设计策略。

【问题3】（3分）

上述C代码的输出为：

___（7）___。

2019 下

阅读下列说明和C代码，回答问题1至问题3。

【说明】

0-1背包问题定义为:给定i个物品的价值v[1…i]、小重量w[1…i]和背包容量T，每个物品装到背包里或者不装到背包里。求最优的装包方案，使得所得到的价值最大。

0-1背包问题具有最优子结构性质。定义c[i][T]为最优装包方案所获得的最大价值，则可得到如下所示的递归式。

$$c[i][T]\begin{cases} 0 & \text{若 } i=0 \text{ 或 } T=0 \\ c[i-1][T] & \text{若 } T<w[i] \\ \max(c[i-1][T-w[i]]+v[i], c[i-1][T]) & \text{若 } i>0 \text{ 且 } T\geq w[i] \end{cases}$$

【c代码】

下面是算法的C语言实现。

（1）常量和变量说明

T: 背包容量

v[]:价值数组

w[]:重量数组

c[][]:c[i][j]表示前i个物品在背包容量为j的情况下最优装包方案所能获得的最大价值

4.数据结构与算法应用真题

(2) C程序

```
#include <stdio.h>
#include <math.h>
#define N 6
#define maxT 1000
int c[N][maxT]={0};

int Memoized_Knapsack(int v[N],int w[N],int T) {
   int i;
    int j;
    for(i=0;i<N;i++){
    for(j=0;j<=T;j++){
        c[i][j]= -1;
      }
    }
    return Calculate_Max_Value(v, w, N-1,T);
}
int Calculate_Max_Value(int v[N],int w[N],int  i ,int j){
    int temp =0;
    if(c[i][j]!=-1){
        return (1) ;
    }
    if(i==0||j==0){
        c[i][j]=0;
    }else{
        c[i][j]= Calculate_Max_Value(v, w,i-1,j);
        if(  (2)  ){
            temp=   (3)  ;
            if(c[i][j]<temp){
                 (4)   ;
            }
      }
    }
    return c [i][j];
}
```

【问题1】 (8分)

根据说明和C代码，填充C代码中的空 (1) - (4) 。

【问题2】 (4分)

根据说明和C代码，算法采用了 (5) 设计策略。在求解过程中，采用了 (6)

(自底向上或者自顶向下) 的方式。

【问题3】 (3分)

若5项物品的价值数组和重量数组分别为v[]= {0，1，6，18，22，28}和w[]= {0，1，2，5，6，7}背包容量为T=

11，则获得的最大价值为 (7) 。

2018 下

阅读下列说明和 C 代码，回答问题 1至问题 3，将解答写在答题纸的对应栏内。

【说明】

给定一个字符序列 B=$b_1b_2\ldots b_n$，其中$b_i \in \{A, C, G, U\}$。B 上的二级结构是一组字符对集合 S=$\{(b_i,b_j)\}$，其中$i, j \in \{1,2,\ldots,n\}$，并满足以下四个条件：

（1）S 中的每对字符是(A,U),(U,A),(C,G)和(G,C)四种组合之一；

（2）S 中的每对字符之间至少有四个字符将其隔开，即 i < j – 4；

（3）S 中每一个字符（记为 b_k）的配对存在两种情况：b_k 不参与任何配对；b_k 和字符 b_t 配对，其中 t < k– 4；

（4）（不交叉原则）若(b_i,b_j)和(b_k,b_l)是 S 中的两个字符对，且 i < k，则 i < k < j < l 不成立。

B 的具有最大可能字符对数的二级结构 S 被称为最优配对方案，求解最优配对方案中的字符对数的方法如下：

假设用 C(i,j)表示字符序列 $b_ib_{i+1}\ldots b_j$ 的最优配对方案（即二级结构 S）中的字符对数，则 C(i,j) 可以递归定义为：

$$C(i,j)=\begin{cases}\max(C(i,j-1),\max(C(i,t-1)+1+C(t+1,j-1))) & 若b_t和b_j匹配且i<j-4\\ 0 & 否则\end{cases}$$

下面代码是算法的C语言实现，其中

n：字符序列长度

B[]：字符序列

C[][]：最优配对数量数组

4.数据结构与算法应用真题

【C 代码】

```
#include <stdio.h>
#include <stdlib.h>
#define LEN 100

/*判断两个字符是否配对*/
int isMatch(char a,char b){
    if((a == 'A' && b == 'U') || (a == 'U' && b == 'A'))
        return 1;
    if((a == 'C' && b == 'G') || (a == 'G' && b == 'C'))
        return 1;
    return 0;
}

/*求最大配对数*/
int RNA_2(char B[LEN], int n){
    int i, j, k, t;
    int max;
    int C[LEN][LEN] = {0};

    for(k = 5; k <= n - 1; k++){
        for(i = 1; i <= n - k; i++){
            j = i + k;
            ____(1)____;
            for(____(2)____; t <= j - 4; t++){
                if(____(3)____ && max < C[i][t - 1] + 1 + C[t + 1][j - 1])
                    max = C[i][t - 1] + 1 + C[t + 1][j - 1];
            }
            C[i][j] = max;
            printf("c[%d][%d] = %d--", i, j, C[i][j]);
        }
    }
    return ____(4)____;
}
```

【问题 1】（8分）

根据题干说明，填充 C 代码中的空（1）-（4）。

【问题2】（4分）

根据题干说明和 C 代码，算法采用的设计策略为（5）。

算法的时间复杂度为（6），（用O表示）。

【问题 3】（3 分）

给定字符序列 ACCGGUAGU ，根据上述算法求得最大字符对数为（7）。

2018 上

阅读下列说明和C代码，回答问题1和问题2，将解答填入答题纸的对应栏内。

【说明】

某公司购买长钢条，将其切割后进行出售。切割钢条的成本可以忽略不计，钢条的长度为整英寸。已知价格表p，其中pi（i = 1，2，…，m）表示长度为i英寸的钢条的价格。现要求解使销售收益最大的切割方案。

求解此切割方案的算法基本思想如下：

假设长钢条的长度为n英寸，最佳切割方案的最左边切割段长度为i英寸，则继续求解剩余长度为n - i 英寸钢条的最佳切割方案。考虑所有可能的i，得到的最大收益rn对应的切割方案即为最佳切割方案。rn的递归定义如下：

$$r_n = \max_{1 \le i \le n}(p_i + r_{n-i})$$

对此递归式，给出自顶向下和自底向上两种实现方式。

4.数据结构与算法应用真题

【C代码】

```
/* 常量和变量说明
    n：长钢条的长度
    p[]：价格数组
*/
#define LEN 100

int Top_Down_ Cut_Rod(int p[],int n){  /*自顶向下*/
    int r=0;
    int i;
    if(n == 0){
        return 0;
    }
    for(i=1;   （1）   ;i++){
        int tmp = p[i]+Top_Down_Cut_Rod(p,n-i);
        r=(r>=tmp)?r：tmp;
    }
    return r;
}


int Bottom_Up_Cut_Rod(int p[],int n){    /*自底向上*/
    int r[LEN]={0};
    int temp=0;
    int i,j;
    for(j=1;j<=n;j++){
        temp=0;
        for(i=1;   （2）   ;i++) {
            temp=   （3）   ;
        }
        （4）   ;
    }
    return r[n];
}
```

【问题1】（8分）

根据说明，填充C代码中的空（1）～（4）。

【问题2】（7分）

根据说明和C代码，算法采用的设计策略为（5）。

求解r_n时，自顶向下方法的时间复杂度为（6）；自底向上方法的时间复杂度为（7）（用O表示）。

2016 上

阅读下列说明和C代码，回答问题1至问题3，将解答写在答题纸的对应栏内。

【说明】

在一块电路板的上下两端分别有n个接线柱。根据电路设计，用(i,π(i))表示将上端接线柱i与下端接线柱π(i)相连，称其为该电路板上的第i条连线。如图4-1所示的π(i)排列为{8,7,4,2,5,1,9,3,10,6}。对于任何1≤i<j≤n，第i条连线和第j条连线相交的充要条件是π(i)>π(j)。

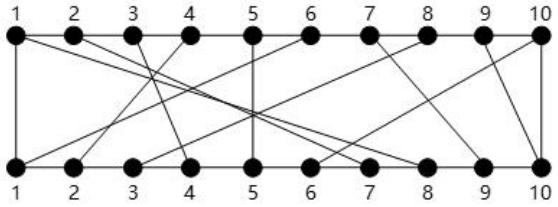

图4-1 电路布线示意

在制作电路板时，要求将这n条连线分布到若干绝缘层上，在同一层上的连线不相交。现在要确定将哪些连线安排在一层上，使得该层上有尽可能多的连线，即确定连线集Nets={(i,π(i))，1≤i≤n}的最大不相交子集。

【分析问题】

记N(i,j)={t|(t,π(t))∈Nets,t≤i,π(t)≤j}。N(i,j)的最大不相交子集为MNS(i,j)，size(i,j)=|MNS(i,j)|。

经分析，该问题具有最优子结构性质。对规模为n的电路布线问题，可以构造如下递归式：

（1）当i=1时，$size(1,j)=\begin{cases}0 & j<\pi(1)\\ 1 & \text{其他情况}\end{cases}$

（2）当i>1时，$size(i,j)=\begin{cases}size(i-1,j) & j<\pi(i)\\ \max\{size(i-1,j),\ size(i-1,\pi(i)-1)+1\} & \text{其他情况}\end{cases}$

【C代码】

下面是算法的C语言实现。

（1）变量说明

size[i][j]：上下端分别有i个和j个接线柱的电路板的第一层最大不相交连接数

pi[i]： π(i)，下标从1开始

（2）C程序

```
#include "stdlib.h"
#include <stdio.h>
#define  N  10   /*问题规模*/
int m=0;     /*记录最大连接集合中的接线柱*/
void maxNum(int pi[],int size[N+1][N+1],int n){/*求最大不相交连接数*/
   int i, j;
   for(j=0; j < pi[1]; j++)   size[1][j] = 0;   /*当j<π(1)时  */
   for(j=pi[1];j<=n;j++)   （1）   ; /*当j>=π(1)时  */
   for(i=2; i < n; i++)  {
      for(j=0; j < pi[i]; j++)   （2）   ; /*当j<pi[i]时  */
      for(j=pi[i];j<=n;  j++)  {/*当j>=c[i]时,考虑两种情况*/
         size[i][j]=size[i-1][j]>=size[i-1][pi[i]-1]+1 ?size[i-1][j]:size[i-1][pi[i]-1]+1;
      }
   }
   /*最大连接数  */
   size[n][n]=size[n-1][n]>=size[n-1][pi[n]-1]+1 ? size[n-1][n]:size[n-1][pi[n]-1]+1;
}
/*构造最大不相交连接集合，net[i]表示最大不相交子集中第i条连线的上端接线柱的序号  */
void constructSet  (int pi[],int size[N+1][N+1],int n,int net[n])  {
   int i,j=n;
   m=0;
   for(i=n ; i>1 ; i--)   {   /*从后往前*/
      if(size[i][j]!=size[i-1][j]){  /*(i,pi[i])是最大不相交子集的一条连线*/
         （3）;   /*将i记录到数组net中，连接线数自增1*/
         j= pi[i]-1;   /*更新扩展连线柱区间*/
      }
   }
   if(j>=pi[1])  net[m++]=1;  /*当i=1时*/
}
```

4.数据结构与算法应用真题

【问题1】（6分）

根据以上说明和C代码，填充C代码中的空（1）～（3）。

【问题2】（6分）

据题干说明和以上C代码，算法采用了__（4）__算法设计策略。

函数maxNum和constructSet的时间复杂度分别为__（5）__和__（6）__（用O表示）。

【问题3】（3分）

若连接排列为{8,7,4,2,5,1,9,3,10,6}，即如图4-1所示，则最大不相交连接数为__（7）__，包含的连线为__（8）__（用(i,π(i))的形式给出）。

2015 下

阅读下列说明和C代码，回答问题1至问题3，将解答写在答题纸的对应栏内。

【说明】

计算两个字符串x和y的最长公共子串（Longest Common Substring）。

假设字符串x和字符串y的长度分别为m和n，用数组c的元素c[i][j]记录x中前i个字符和y中前j个字符的最长公共子串的长度。

c[i][j]满足最优子结构，其递归定义为：

$$C[i][j]=\begin{cases} c[i-1][j-1]+1 & 若i>0且j>0且x[i-1]=y[j-1] \\ 0 & 其它 \end{cases}$$

计算所有c[i][j](0 ≤i ≤ m，0 ≤j ≤ n)的值，值最大的c[i][j]即为字符串x和y的最长公共子串的长度。根据该长度即i和j，确定一个最长公共子串。

【C代码】

（1）常量和变量说明

x，y：长度分别为m和n的字符串

c[i][j]：记录x中前i个字符和y中前j个字符的最长公共子串的长度

max：x和y的最长公共子串的长度

maxi, maXj：分别表示x和y的某个最长公共子串的最后一个字符在x和y中的位置（序号）

4.数据结构与算法应用真题

（2）C程序

```
#include <stdio.h>
#include <string.h>

int c[50][50];
int maxi;
int maxj;

int lcs(char *x, int m, char *y, int n)    {
    int i, j;
    int max= 0;
    maxi= 0;
    maxj = 0;


    for ( i=0; i<=m ; i++)          c[i][0] = 0;
    for (i =1; i<= n; i++)           c[0][i]=0;
    for (i =1; i<= m; i++)   {
       for (j=1; j<= n; j++)   {
          if (   (1)   )   {
           c[i][j] = c[i -1][j -1] + 1;
           if(max<c[i][j]) {
              (2)   ;
            maxi = i;
            maxj =j;
           }
       }
       else    (3)   ;
      }
   }
    return max;
}
```

4.数据结构与算法应用真题

```
void printLCS(int max, char *x) {
        int i= 0;
      if (max == 0)       return;
    for ( (4)   ; i < maxi; i++)
          printf("%c",x[i]);
}
void main(){
char* x= "ABCADAB";
char*y= "BDCABA";
 int max= 0;
 int m = strlen(x);
 int n = strlen(y);

 max=lcs(x,m,y,n);
 printLCS(max , x);
}
```

【问题1】（8分）

根据以上说明和C代码，填充C代码中的空（1）～（4）。

【问题2】（4分）

根据题干说明和以上C代码，算法采用了__（5）__设计策略。

分析时间复杂度为__（6）__（用O符号表示）。

【问题3】（3分）

根据题干说明和以上C代码，输入字符串x= "ABCADAB'，'y="BDCABA"，则输出为__（7）__。

2014 下

阅读下列说明和C代码，回答问题1至问题3，将解答写在答题纸的对应栏内。

【说明】

计算一个整数数组a的最长递增子序列长度的方法描述如下：

假设数组a的长度为n，用数组b的元素b[i]记录以a[i](0≤i；其中b[i]满足最优子结构，可递归定义为：

$$\begin{cases} b[0]=1 \\ b[i]=\max\limits_{\substack{0\le k\le i \\ a[k]\le a[i]}}\{b[k]\}+1 \end{cases}$$

【C代码】

下面是算法的C语言实现。

（1）常量和变量说明

a：长度为n的整数数组，待求其最长递增子序列

b：长度为n的数组，b[i]记录以a[i](0≤i度，其中0≤ilen：最长递增子序列的长度

i,j：循环变量

temp：临时变量

4.数据结构与算法应用真题

（2）C程序

```
#include <stdio.h>

int maxL(int*b, int n) {
int i, temp=0;
for(i=0; i<n; i++) {
if(b[i]>temp)
temp=b[i];
}
return temp;
}

int main() {
int n, a[100], b[100], i, j, len;
scanf("%d", &n);
for(i=0; i<n; i++) {
scanf("%d", &a[i]);
}
(1) ;
for(i=1; i<n; i++) {
for(j=0, len=0; (2) ; j++) {
if( (3) && len<b[j])
len=b[j];
}
(4) ;
}
Printf("len:%d\n", maxL(b,n));
printf("\n");
}
```

【问题1】（8分）

根据说明和C代码，填充C代码中的空（1）～（4）。

【问题2】（4分）

根据说明和C代码，算法采用了__（5）__设计策略，时间复杂度为__（6）__（用O符号表示）。

【问题3】（3分）

已知数组a={3,10,5,15,6,8}，根据说明和C代码，给出数组b的元素值。

2016 下

阅读下列说明和C代码，回答问题1至问题3，将解答写在答题纸的对应栏内。

【说明】

模式匹配是指给定主串t和子串s，在主串t中寻找子串s的过程，其中s称为模式。如果匹配成功，返回s在t中的位置，否则返回-1 。

KMP算法用next数组对匹配过程进行了优化。KMP算法的伪代码描述如下：

1．在串t和串s中，分别设比较的起始下标i=j=0。

2．如果串t和串s都还有字符，则循环执行下列操作：

（1）如果j=-l或者t[i]=s[j]，则将i和j分别加1，继续比较t和s的下一个字符；

（2）否则，将j向右滑动到next[j]的位置，即j =next[j]。

3．如果s中所有字符均已比较完毕，则返回匹配的起始位置（从1开始）；否则返回-1。

其中，next数组根据子串s求解。求解next数组的代码已由get_next函数给出。

【C代码】

（1）常量和变量说明

t，s：长度为lt和ls的字符串

next:next数组，长度为ls

4.数据结构与算法应用真题

（2）C程序

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
/*求next[]的值*/
void get_next( int *next, char *s, int ls) {
   int i=0，j=-1;
   next[0]=-1;/*初始化next[0]*/
   while(i < ls){/*还有字符*/
   if(j==-1| ls[i]==s[j]){/*匹配*/
      j++;
      i++;
   if( s[i]==s[j])
     next[i] = next[j];
   else
     Next[i] = j;
  }
else
  j = next[j];
 }
}
 int kmp( int *next, char *t ,char *s, int lt, int ls )
 {
      Int i= 0,j =0 ;
      while (i < lt &&  （1）  ){
        if( j==-1 ||  （2）  ){
             i ++ ;
             j ++ ;
         } else
              （3）  ;
}
 if (j >= ls)
 return  （4）  ;
 else
    return -1;
 }
```

【问题1】（8分）

根据题干说明，填充C代码中的空（1）～（4）.

【问题2】（2分）

根据题干说明和C代码，分析出kmp算法的时间复杂度为（5）（主串和子串的长度分别为lt和ls，用O符号表示）。

【问题3】（5分）

根据C代码，字符串“BBABBCAC”的next数组元素值为（6）（直接写素值，之间用逗号隔开）。若主串为“AABBCBBABBCACCD”，子串为“BBABBCAC”，则函数Kmp的返回值是（7）。