1.（1）做题思路：

分析题目可知，我们需要得到大小写的个数，定义 numb 和 numc 分别表示小写字符数量和大写字符数量。先统计大小写数量，同时定义数组 b 和 c 记录下每个大小写字符的下标。然后进行比较大小写数量。将其中少的一种，根据 b 或者 c 种的下标找到字符，然后用 ASCII 码直接进行转换。

（2）代码：

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAXSIZE 1000

void count(char a[]){
    int i,numb=0,numc=0,bloc=0,cloc=0,len=strlen(a) ;
    //strlen()函数在头文件 <string.h>中，必须引入
    //也可以自己定义一个计算长度的函数

    int *b = (int *)malloc(sizeof(int)*len);
    int *c = (int *)malloc(sizeof(int)*len);
    //  如果定义一个动态数组，使用 malloc 现场开辟空间
    //  将返回的指针强制转换成 int * 型

    //  int b[MAXSIZE],c[MAXSIZE];
    //  或者这样定义一个固定长度的数组

    for(i=0,bloc=0,cloc=0;i<len;i++)
    {
        if(a[i]>='a' && a[i]<='z')
        {                       //如果 是小写字符
            numb++;             //小写字符计数器
            b[bloc++] = i;      //记录下小写字符下标
        }

        if(a[i]>='A' && a[i]<='Z')
        {                       //如果 是大写字符
            numc++;             //大写字符计数器
            c[cloc++] = i;      //记录下大写字符下标
        }
    }
```

```
    int j;
    int trans = 'a'-'A';
    if(numb>=numc)                //如果小写多余等于大写
    {
        for(j=0;j<cloc;j++)       //批量转换
            a[c[j]] = a[c[j]] + trans;
    }
    else                          //如果小写少于大写
    {
        for(j=0;j<bloc;j++)       //批量转换
            a[b[j]] = a[b[j]] - trans;
    }
    free(b);
    free(c);
    //  如果定义了动态数组，在程序结束前须要释放空间

    puts(a);
    //puts 打印字符串
    //也可以用循环+printf
}

int main( ){
    char s[MAXSIZE];
    gets(s);
    //gets 获取字符串
    //也可以用循环+scanf
    count(s);
    return 0;
}
```

2.（1）思路

学生的编号与实际位置不一一对应，所以定义一个数组 a[ ]，其下标表示学生编号，其下标对应的元素的值表示其当前位置。

进行移动操作，移动操作分为往后（数组 a 元素+1）和往前移动（-1），移动当前位置到移动范围内的所有元素，这里注意移动范围不能超过数组范围，最后更新给出的学生位置。

移动完毕打印序列，先使用一个数组根据学生下标找到并保存对应位置，然后遍历这个数组输出每个位置上的学生编号。

（2）代码

```c
#include <stdio.h>
#define MAXSIZE 1000
void print(int a[],int n){
    int b[MAXSIZE],i;
    for(i=1;i<=n;i++)
        b[a[i]] = i;

    for(i=1;i<n;i++){
        printf("%d ",b[i]);
    }
    // 输出前 n-1 个
    printf("%d",b[i]);
    // 最后一个输出后面不需要空格
}

void adjust(int a[],int n,int p,int q){
    int i;
    if(q>0)                         //如果往后移动
    {
        if(q > n-a[p])              //如果移动长度超过后面总长
            q = n-a[p];            //将移动距离设置成可移动最大距离
        for(i=1;i<=n;i++)           //循环移动
            if(a[i]>a[p] && a[i]<=a[p]+q)     //判断是否在移动范围内
                a[i] = a[i]-1;
        a[p] = a[p]+q;
    }
    else if(q<0)           //如果往前移动
    {
        if(-q > a[p]-1)       //如果移动长度超过前面总长
            q = -(a[p]-1);  //将移动距离设置成可移动最大距离
        for(i=1;i<=n;i++)     //循环移动
            if(a[i]>=a[p]+q && a[i]<=a[p]-1)  //判断是否在移动范围内
                a[i] = a[i]+1;
        a[p] = a[p]+q;

    }
}

int main( ){
    int n,m,i,p,q;
    scanf("%d",&n);
    scanf("%d",&m);
```

```
    // 得到 学生的数量 n,调整的次数 m

    int a[MAXSIZE];
    for(i=1;i<=n;i++){
        a[i] = i;
    }
    // 初始化操作

    for(i=0;i<m;i++){
        scanf("%d %d",&p,&q);
        adjust(a,n,p,q);
    }
    //每次循环获得操作序列并执行调整函数

    print(a,n);
    //打印正确序列

    return 0;
}
```

3.（1）将字符串存储在二维数组中，每个一维数组就是一个单词，然后进行排序操作。另外不考虑大小写，所以需要统一格式，将所有大写字母转换成小写字母进行比较。

（2）

```
#include <stdio.h>
#include <string.h>
#define MAXSIZE 1000

void sort(char save[][MAXSIZE], int num)
{
    //冒泡排序
    char temp1[MAXSIZE],temp2[MAXSIZE],temp[MAXSIZE];
    int i,j;
    int m;
    for(i=0; i<num; i++){
        for(j=0;j<num-i;j++){

            strcpy(temp1,save[j]);
            strcpy(temp2,save[j+1]);
            for(m=0;temp1[m]!='\0';m++)
                if(temp1[m]>='A' && temp1[m]<='Z')
```

```
                    temp1[m]+=32;
            for(m=0;temp2[m]!='\0';m++)
                if(temp2[m]>='A' && temp2[m]<='Z')
                    temp2[m]+=32;
            if(strcmp(temp1,temp2)>0){
                strcpy(temp,save[j]);
                strcpy(save[j],save[j+1]);
                strcpy(save[j+1],temp);
            }
        }
    }
}

void change(char s[],char save[][MAXSIZE],int *num_p)
{   //这个函数用来将含有空格的字符串分割成若干个字符串

    int i,j=0,k=0;
    while(s[i]!='\0'){          //如果 s[i]为\0 意味着整个字符串到尾了
        save[k][j++] = s[i];    //挨个赋值
        if(s[i] == ' ' ){       //如果 s[i]为空格意味着其中一个字符串已经到尾
            save[k][j] = '\0';  //在该字符串最后添一个\0 作为结束标记
            k++;                //字符串数量+1
            j=0;
        }
        i++;
    }
    save[k][j] = '\0';
    *num_p = k;
}

int main()
{
    char s[MAXSIZE];
    char save[MAXSIZE][MAXSIZE];
    int num,i;        //num 保存字符串的最大下标
    gets(s);
    change(s,save,&num);
    for(i=0;i<=num;i++)
        puts(save[i]);
    sort(save,num); //排序
    for(i=0;i<=num;i++)
        puts(save[i]);
    return 0;
}
```

4.（1）由题意得，这是一道最小生成树问题，选用 Prim 算法。

Prim 算法的基本思想是：首先置 s={1}，然后只要 S 是 V 的真子集，就做如下贪心选择：

选取满足条件的 i 属于 S，j 属于 V-S，且 c[i][j]最小的边，并将顶点 j 添加到 S 中，这个过程重复一直到 S=V 为止。在这个过程中选取的边恰好构成 G 的一颗最小生成树。

（2）

```
void Prim(int n, Type c[][])
{
    T = 空集;
    S = {1};
    while(S != V){
        (i,j)=i 属于 S 且 j 属于 V-S 的最小边权;
        T = T U {(i,j)};
        S = S U {j};
    }
}
```

5.（1）从前往后读，确保运算次序的正确性。当读到左括号时，意味着进行一次运算。用栈 S 来存储运算数。每进行一次运算，出栈两个运算数。并将新的结果入栈，最终栈顶元素即为表达式的值。

（2）

```
#include <stdio.h>
#include <string.h>
#define maxsize 100

int s[maxsize],top=-1;

int add(int m,int n)
{
    return m+n;
}

```

```
int max(int m,int n)
{
    return m>=n? m:n;
}

int min(int m,int n)
{
    return m>=n? n:m;
}

//将字符串倒置
void reverse(char a[])
{
    int i,j;
    char temp;
    for(i=0,j=strlen(a)-1;i<j;i++,j--){
        temp = a[i];
        a[i] = a[j];
        a[j] = temp;
    }
}

//将字符串数字转换为数字
int change(char a[])
{
    int i;
    int sum=0;
    for(i=0;i<strlen(a);i++)
        sum = sum*10+a[i]-'0';
    return sum;
}

int main( ){
    int n,i,j=0,k;
    int p,q;
    char str[maxsize],a[maxsize],b[400];
    scanf("%d",&n);
    getchar();
    while(n--){
        gets(str);
        //初始化数组 a
        //memset()用来对一段内存空间全部设置为某个字符,一般用不着
        memset(a,'\0',sizeof(a));
        //从后往前读
```

```
        for(i=strlen(str)-1; i>=3; i--){
            if(str[i]==')') continue;
            //读到数字
            if(str[i]!=','&&str[i]!='(')
            {
                a[j++] = str[i];
                a[j] = '\0';
            }
            //一个数字读取完成，转化为数字，存到栈中
            if(str[i-1]==','||str[i-1]=='(')
            {
                reverse(a);
                //将数字字符串转化为数字，入栈
                s[++top] = change(a);
                j=0;
            }
            if(str[i]=='('){
                p = s[top--];
                q = s[top--];
                //查看'（'前的运算
                switch(str[i-1]){
                    case 'd':s[++top] = add(p,q);i=i-3;continue;
                    case 'n':s[++top] = min(p,q);i=i-3;continue;
                    case 'x':s[++top] = max(p,q);i=i-3;continue;
                }
            }
        }
    printf("%d\n",s[top]);
    }
    return 0;
}
```

6.（1）分析得知：每一个位置 map[i][j]只可能来自 map[i][j-1]向右走一个结点或者 map[i-1][j]向下走一个结点，因此只需要比较到达 map[i][j-1]和到达 map[i-1][j]的路径较小值加上 map[i][j]就是所求答案，

思路：求出到达每一个结点 map[i][j]的最小路径将其保存在数组 dp[i][j]中，求任意 dp[i][j]的值完全依赖于 dp[i-1][j]和 dp[i][j-1]，因此易知，先求出 dp[][]数组的

第 1 行和第 1 列，然后从上到下，从左到右计算出每一个位置的结果值。

② 创建一个二维数组记录每个位置的最小路径 dp[n][m];

②求出 dp[ ][ ]中第 1 行和第 1 列的结果填充到 dp[ ][ ]中；注意：在动态规划问题中第 1 行和第 1 列需要手动求出，需要根据问题的要求进行求解，一般第 1 行和第 1 列的求解很简单。

③ 从上到下，从左到右，通过二重循环求出任意 dp[i][j]的结果填充到 dp[ ][ ]中；注意：二重循环中 i, j 都是从 1 开始进行遍历，即从矩阵第 2 行第 2 列的位置开始填充。

④ 最后 dp[n - 1][m - 1]就是所求的结果。

当 j=0 且 i=0 时 dp[0][0] = map [0][0];

当 j=0 时: dp[i][0] = dp[i-1][0] + map[i][0];

当 i=0 时: dp[0][j] = dp[0][j-1] + map [0][j];

其余情况 dp[i][j] = min(dp[i-1][j] ,dp[i][j-1])+ map [i][j];

（2）

```
#include <stdio.h>
#include <stdlib.h>
#define MAXSIZE 1000

int min(int a,int b){
    return a<b?a:b;
}

//矩阵最短路径和问题：动态规划
int MinPath(int map[][MAXSIZE],int n,int m){
    //特殊输入
    if(n<=0||m<=0) return 0;

    //创建动态规划结果矩阵 dp[][]
    int dp[MAXSIZE][MAXSIZE];
    // int *dp = (int *)malloc(sizeof(int)*n*m);
```

```
17.
18.     //求解第 1 行第 1 列的结果值
19.     dp[0][0]=map[0][0];
20.     //求第 1 行的结果值
21.     for(int i=1;i<m;i++){
22.         dp[0][i]=dp[0][i-1]+map[0][i];
23.     }
24.     //求第 1 列的结果值
25.     for(int i=1;i<n;i++){
26.         dp[i][0]=dp[i-1][0]+map[i][0];
27.     }
28.     //从上到下，从左到右求任意 dp[i][j]
29.     for(int i=1;i<n;i++){
30.         for(int j=1;j<m;j++){
31.             dp[i][j]=map[i][j]+min(dp[i-1][j],dp[i][j-1]);
32.         }
33.     }
34.     //返回右下角的结果值
35.     return dp[n-1][m-1];
36. }
```

7.（1）思路是用 map 数组来存储地图，然后从@点出发，向四周计数。这里可以把 map 看成拥有最多四个孩子的图，即可使用深度优先遍历算法或者广度优先遍历算法。

若使用深度优先遍历，则访问当前节点后，访问其一个孩子，只要满足条件，就一直递归遍历。否则回溯访问其下一个孩子。（借鉴回溯法，中后序遍历二叉树）

若使用广度优先遍历，则访问当前节点后，将其所有满足条件的孩子入队。出队一个节点作为扩展节点，继续重复上述操作。（借鉴分支限界法，层次遍历二叉树）

递归，如果遇到'A'瓷砖，则返回 1 + 周围几个'A'瓷砖递归下去遇到的'A'瓷砖数。

注意当数过一个'A'瓷砖后要把那个'A'瓷砖赋值为'#'瓷砖，保证不会重复计数。

其中可能的边界条件有两种，第一种是遇到房间边缘，第二种是遇到'#'瓷砖。

（2）

```
1.  #include<stdio.h>
2.  #include<stdlib.h>
```

```
#define MAXSIZE 1000              //定义数组可能的最大尺寸 MAXSIZE

char map[MAXSIZE][MAXSIZE]; //map 是瓷砖集合
int w,h;                          //w,h 房间的宽和高
                                  //定义全局变量，方便函数调用
int cal_num(int x,int y){
    if(x<0||x>=h||y<0||y>=w)     return 0;    //达到房间的边界
    if(map[x][y]=='#')  return 0;       //遇到'#'瓷砖不能通行，不计数
    else {
        map[x][y]='#';        //将遇到的'A'瓷砖置为'#'，以后遇到不再计数
                              //下式,当前计数+1，然后递归探索周围四块瓷砖
        return 1+cal_num(x-1,y)+cal_num(x+1,y)+cal_num(x,y-
1)+cal_num(x,y+1);
    }
}

int main(){
    scanf("%d %d\n",&w,&h);      //得到 w 和 h,第一行的末尾有个'\n'
    int i,j;
    for(i=0;i<h;i++)
        gets(map[i]);        //得到接下来的瓷砖集合
    for(i=0;i<h;i++)          //找到当前位置'@'
        for(j=0;j<w;j++)
            if(map[i][j]=='@')
                printf("%d\n",cal_num(i,j));
    return 0;
}
```

8（1）类比于书本上的全排列问题，先选中一个砝码 i，将其添加栈中，剩余需要称出重量 M 减去当前考虑的砝码 i，然后递归其子问题，用后面的砝码来称出剩余重量。当 M<0 时表示选取的砝码总重量已大于 M。当 M=0 时，表示选取的砝码总重量恰好等于 M，得到可行解，这时计数栈中砝码数量并更新最小值。

（2）

```
#include<stdio.h>
#define MAXSIZE 100000
#define INF 10000000000

int stack[MAXSIZE]; //stack[]栈，存放已经选取的砝码
int top=-1;
```

```
int min_num = INF;  //min_num 表示最少砝码数，INF 是已定义的无穷大

void cal_num(int w[],int a,int b,int m){    //m 表示剩余需要称出的重量
    int i;
    if(m<0) return;           //剩余需要称出的重量<0 时 return
    if(m==0){                //剩余需要称出的重量=0 时，即找到一种可行解
        int count=0;         //计数并更新最小值
        for(i=0;i<=top;i++)  count +=1;
        if(count<min_num) min_num =count;
    }
    for(i=a;i<=b;i++){
        stack[++top] = w[i];         //将 i 砝码选中
        cal_num(w,i+1,b,m-w[i]);//考虑后面砝码。需要称出重量 M 减去 i 号砝码重量
        top--;                       //将 i 砝码剔除
    }
}

int main(){
    int w[MAXSIZE]={0}; //w 数组存放所有砝码
    int n,m,i;          //n 表示砝码的数量 N，m 表示需要称出的重量 M
    scanf("%d",&n);     //得到砝码的数量 N
    scanf("%d",&m);     //得到需要称出的重量 M
    for(i=0;i<n;i++) //得到 N 个砝码的重量
        scanf("%d",&w[i]);
    cal_num(w,0,n,m);   //计算最少砝码数
    printf("min_num: %d\r\n",min_num);  //打印最少砝码数
    return 0;
}
```