某地区开展扶贫攻坚工作中，深入调查城镇交通状况，发现城镇之间路网损坏严重，不少村镇之间无法通车。为实现“畅通工程”，使整个地区任何两个城镇间都可以实现快速交通（但不一定有直接的快速道路相连，只要互相间接通过快速路可达即可），需要对个城镇之间的快速道路进行修建或修缮。现得到城镇道路统计表如下，表中列出了任意两城镇间修建或修缮快速路的费用。

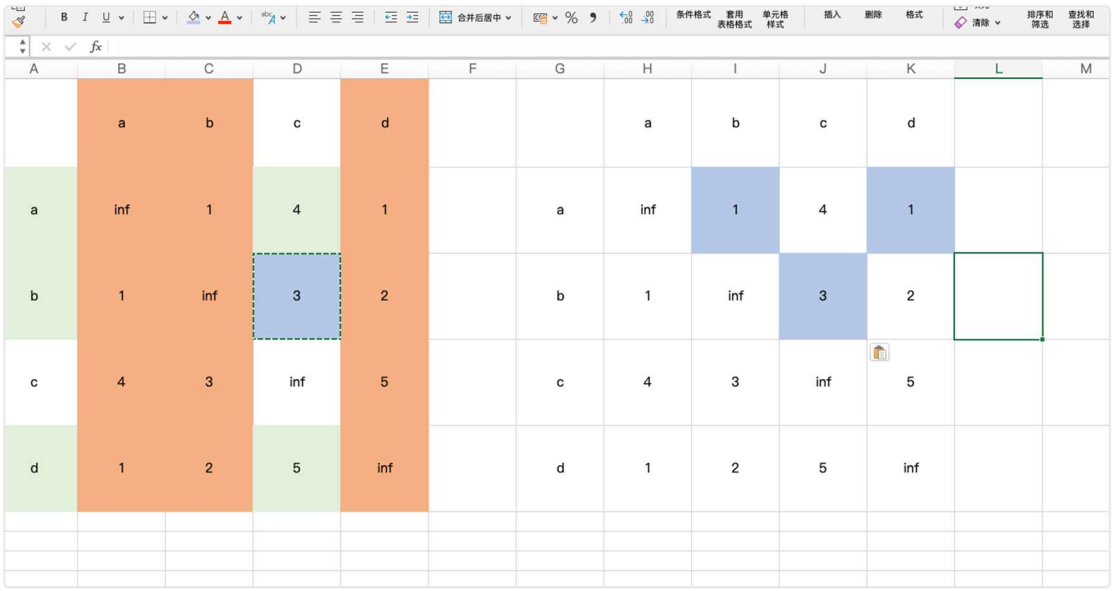
城镇编号	城镇编号	建造代价
1	2	1
1	3	4
1	4	1
2	3	3
2	4	2
3	4	5

城镇编号	城镇编号	建造代价
A	B	1
A	C	4
A	D	1
B	C	3
B	D	2
C	D	5

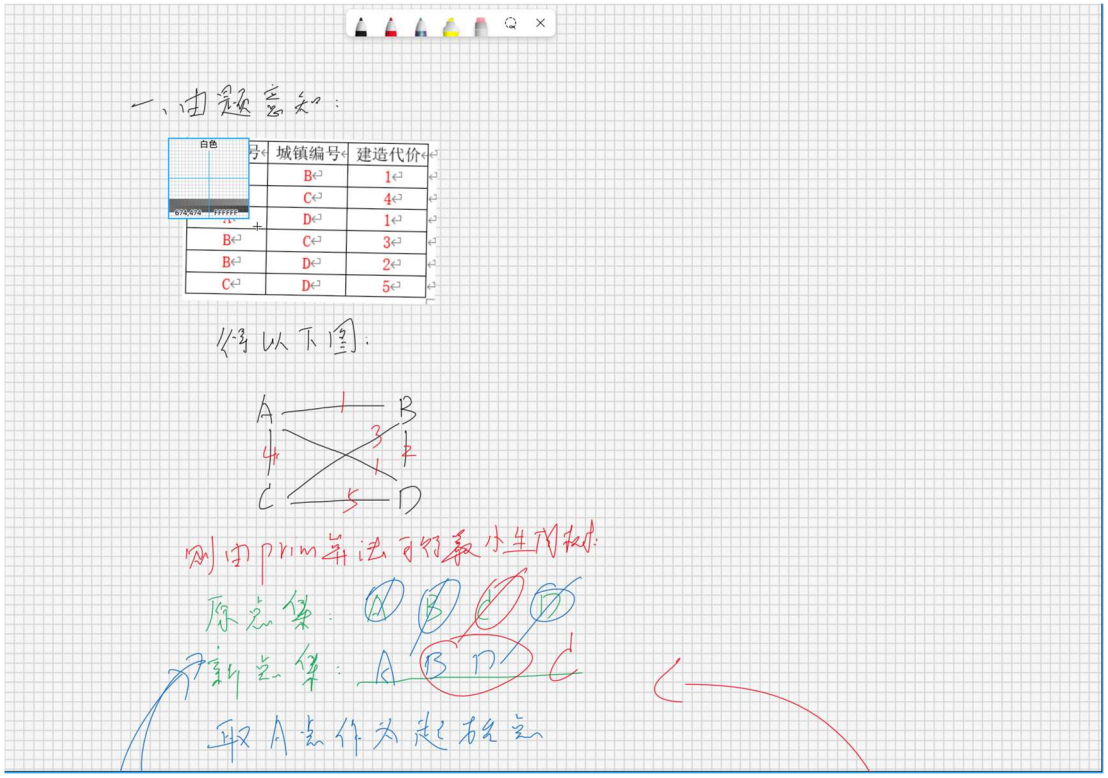
请基于数据结构与算法的基本原理，完成以下任务：

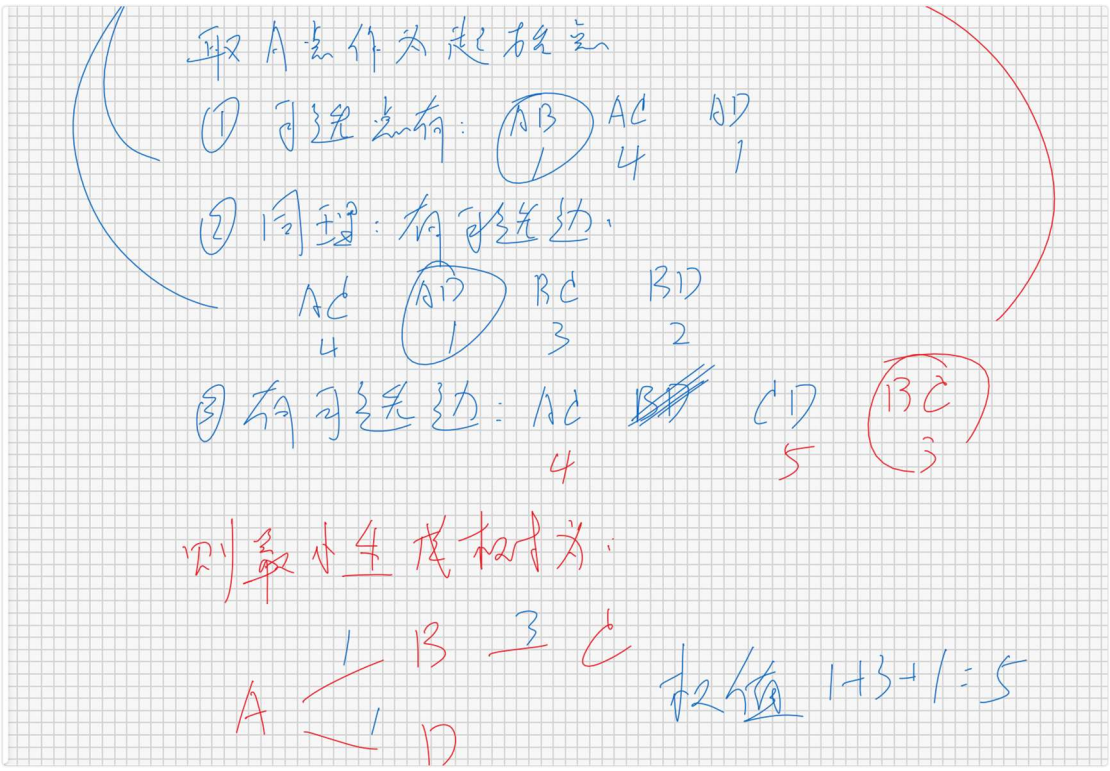
（1） 建立表示该城镇交通路网的逻辑结构模型，并给出其数学描述和可视化表示。（3 分）

如图所示



	a	b	c	d			a	b	c	d		
a	inf	1	4	1		a	inf	1	4	1		
b	1	inf	3	2		b	1	inf	3	2		
c	4	3	inf	5		c	4	3	inf	5		
d	1	2	5	inf		d	1	2	5	inf		





（2） 设计至少两种数据存储方案，对比分析两种存储方案的优缺点。（7 分）

1、邻接矩阵　　优点：简单、操作方便，能够更加方便的计算各个顶点的度，可以容易的看出来是否在两点之间有边的存在　　缺点：在进行增删查改的操作的时候会很麻烦，若要统计边的数目还需要构造结构

体，略显反锁

2、邻接表　　优点：可以极大的提高增删查改的效率，能够容易的在邻接表当中读出边的数目　　缺点：操作难度提高，对于边和点的关系难以体现

3、对于本题，由于数据复杂度较低且不涉及增删查改的操作，采用邻接矩阵使用数组来存放数据具有更大的优势

（3） 请选取或设计适当的算法，计算出使全地区畅通需要的最低成本。请写出选取算法的依据、算法的基本原理，计算全地区畅通所需最低成本的过程和结果。（10 分）

如代码所示

```
#include "stdio.h"
#include "stdlib.h"
#define inf 2147483647
typedef struct{
    char* arr;
    int length;
}charList;
typedef struct{
    char* border;
    int weight;
}weightGraphBorder;
typedef struct{
    weightGraphBorder* borders;
    int num;
}graphBorders;
typedef enum {false, true} boolean;
charList* appendCharArray(char, charList*);
boolean charInCharList(char, charList*);
charList* replaceCharInList(char, char, charList*);
void printMatrix(charList*, graphBorders*);
int main(){
    int numOfBorder;
    printf("输入带权值边的数量：");
    scanf("%d", &numOfBorder);
    weightGraphBorder graph[numOfBorder];
    int n = numOfBorder;
    charList points;
    points.length = 0;
```

这里的代码存在问题，参见另一个仓库 prim-minimum-spanning-tree

```
while (n != false){
    printf("输入%d 条边及其权值，以空格隔开：",numOfBorder-n+1);
    char* s = (char *) malloc(sizeof(char)*3);
    int l;
    scanf("%s %d",s,&l);
    graph[numOfBorder-n].border = s;
    graph[numOfBorder-n].weight = l;
    for(int i = 0; i < 2; i += 1){
        if(!charInCharList(s[i], &points)){
            points = *appendCharArray(s[i], &points);
        } else{
            continue;
        }
    }
    n -= true;
}
charList pointsAll;
pointsAll.length = points.length;
char* sss = (char*) malloc(sizeof(char)*points.length);
for (int i = 0; i < pointsAll.length; i += 1){
    sss[i] = points.arr[i];
}
pointsAll.arr = sss;
for(int i = 0; i < numOfBorder; i += 1){
    for (int j = i; j < numOfBorder; j += 1) {
        if(graph[i].weight > graph[j].weight){
            weightGraphBorder cache;
            cache = graph[i];
            graph[i] = graph[j];
            graph[j] = cache;
        }
    }
}
charList newPoints;
char* ssss = (char*) malloc(sizeof(char)*points.length);
newPoints.arr = ssss;
newPoints.length=0;
printf("选择一个开始点:");
char startPoint;
int borderLinked = 0;
getchar();
scanf("%c", &startPoint);
newPoints = *appendCharArray(startPoint, &newPoints);
points = *replaceCharInList(startPoint, '\0', &points);
```

```
    graphBorders AllBorders;
    AllBorders.borders=graph;
    AllBorders.num = numOfBorder;
    printMatrix(&pointsAll, &AllBorders);
    int weightSum = 0;
    while (borderLinked<pointsAll.length-1){
        weightGraphBorder shortest;
        shortest.weight = inf;
        for(int i = 0; i < numOfBorder; i += 1){
            if(((charInCharList(graph[i].border[0], &newPoints) &&
charInCharList(graph[i].border[1], &points))||
            (charInCharList(graph[i].border[1], &newPoints) &&
charInCharList(graph[i].border[0], &points)))
            &&(graph[i].weight < shortest.weight)){
                shortest.border=graph[i].border;
                shortest.weight=graph[i].weight;
                graph[i].weight = inf;
            } else{
                continue;
            }
        }
        if(charInCharList(shortest.border[0], &newPoints)){
            newPoints = *appendCharArray(shortest.border[1],
&newPoints);
            points = *replaceCharInList(shortest.border[1], '\0',
&points);
            printf("\n第%d条路径：%c-->%c  %d", borderLinked+1,
shortest.border[0], shortest.border[1], shortest.weight);
        } else{
            newPoints = *appendCharArray(shortest.border[0],
&newPoints);
            points = *replaceCharInList(shortest.border[0], '\0',
&points);
            printf("\n第%d条路径：%c-->%c  %d", borderLinked+1,
shortest.border[1], shortest.border[0], shortest.weight);
        }
        borderLinked += 1;
        weightSum += shortest.weight;
    }
    printf("\n最小生成树权值和：%d\nAll done!", weightSum);
    return 0;
}
charList* appendCharArray(char c, charList* arr){
    charList* p = (charList*)malloc(sizeof(charList));
```

```
    char* ss = (char*) malloc((sizeof(char)*(arr->length + 1)));
    int l = 0;
    for(int i=0; i < arr->length; i += 1){
        ss[i] = arr->arr[i];
        l += 1;
    }
    ss[arr->length] = c;
    p->arr = ss;
    p->length = arr->length + 1;
    return p;
}
boolean charInCharList(char c, charList* s){
    for (int i=0; i < s->length; i += 1){
        if(s->arr[i] == c){
            return true;
        }
    }
    return false;
}
charList* replaceCharInList(char before, char after, charList* sA){
    char* sB = (char*) malloc(sizeof(char) * (sA->length));
    charList* p = (charList*) malloc(sizeof(charList));
    for (int i = 0; i < sA->length; i += 1) {
        if(sA->arr[i]==before){
            sB[i] = after;
        } else{
            sB[i]=sA->arr[i];
        }
    }
    p->length=sA->length;
    p->arr=sB;
    return p;
}
void printMatrix(charList* points, graphBorders* borders){
    printf("\n 邻接矩阵: ");
    printf("\n\t");
    for (int i = 0; i < points->length; i += 1) {
        printf("\t%c", points->arr[i]);
    }
    for (int i = 0; i < points->length; i += 1){
        printf("\n\t%c", points->arr[i]);
        for (int j = 0; j < points->length; j += 1){
            boolean flag = true;
            for (int k = 0; k < borders->num; k += 1){
```

```
                if ((borders->borders[k].border[0] == points->arr[i]
    && borders->borders[k].border[1] == points->arr[j])||
                        borders->borders[k].border[1] == points->arr[i]
    && borders->borders[k].border[0] == points->arr[j]){
                    printf("\t%d", borders->borders[k].weight);
                    flag = false;
                    break;
                }
            }
            if (flag){
                printf("\t-");
            }
        }
    }
}
```

测试用例：

题目用例：

6

ab 1

ac 4

ad 1

bc 3

bd 2

cd 5

a

其余用例：

（1）、例 1

10

ab 7

ad 5

bc 8

bd 9

be 7

ce 5

df 6

ef 8

eg 9

fg 11

d

(2)、例 2

11

ab 18

af 19

ag 18

bc 8

bg 20

cd 20

de 9

df 16

dg 15

ef 3

fg 15

c

(3)、例 3

15

ab 2

ac 3

bd 2

ce 2

cf 4

df 2

eg 2

fg 5

fh 2

gh 2

gi 4

gj 3

hj 6

ik 5

jk 3

a