2.

```
#define destinations 8
#define pathways ((int) (sizeof(road_networks) / sizeof(road_networks[0])))
```

First, declare the arrays. The variable *destinations* is the set of points from A-H while the variable *pathways* is the set of direct paths between points. Since the number of direct pathways is unknown, macro definition is used.

```
int main(){
    int i, j, location, last_location;
    //initializes road networks
    bool road_networks[destinations][destinations] = {[0][0]=true,[0][1]=true,[0][5]=true,
                                                      [1][0]=true,[1][1]=true,[1][2]=true,
                                                      [2][1]=true,[2][2]=true,[2][4]=true,
                                                      [2][5]=true,[3][3]=true,[3][4]=true,
                                                      [4][3]=true,[4][4]=true,[5][0]=true,
                                                      [5][2]=true,[5][5]=true,[6][0]=true,
                                                      [6][3]=true,[6][6]=true,
                                                      [7][5]=true,[7][7]=true,
                                                     };
```

Then, declare the data type and variable names. Since the multidimensional array *road_networks* consist of two types of data, Boolean data type is used. Designated initializers are used to shorten and for easier reading because it's a large array.

```
//displays adjacency matrix
for (i = 0; i <= pathways; i++){
    for (j = 0; j <= pathways; j++){

        //prints the blank space on the top left
        if (i == 0 && j==0){
            printf("     ");
        }

        //prints the row of points and charging stations
        else if (i == 0){
            if (j == 3 || j == 4){
                printf("[%c]   ",j+64);
            }
            else{printf(" %-5c",j+64);}
        }
        //prints the column of points and charging stations
        else if (i != 0 && j == 0){
            if (i == 3 || i == 4){
                printf("[%c]  ",i+64);
            }
            else{printf("%-5c",i+64);}
        }
```

The program will then display the adjacency matrix using nested for loops. Since the ASCII code for 'A' is 65, 64 is added to each row or column of the matrix.

```
        //prints 1 on points with direct path
        //prints 0 otherwise
        else if (road_networks[i-1][j-1]==true){
            printf(" %-5s","1");
        }
        else{printf(" %-5s","0");}
    }
    printf("\n");
}
```

Then, the program will print 1 if there is a direct path between points and 0 if there is no direct path based on the designated initialization.

```
//finds the nearest charging station
while(location != 2 || location !=3){
    //checks if user is already in the charging station
    if (road_networks[location][2]){
        location = 2;
    }
    else if (road_networks[location][3]){
        location = 3;
    }

    //checks for available routes
    /*checks if the user already crossed the same path
    to prevent the user from crossing the same route*/
    else{
        for (i = 0; i < pathways; i++){
            if (road_networks[location][i] && i !=location && i != last_location){
                last_location = location;
                location = i;
                break;
```

It was actually a little bit difficult in this part since all available routes should be checked and the user shouldn't cross the same route.

```
    //prints the location of the user
    if (location == 2 || location == 3){
        printf("Point %c: Arrived at the charging station.", location+65);
        break;
    }
    else{printf("Now at point: %c\n", location+65);}
}
return 0;
```

Finally, the program will print the paths the user took to reach the nearest charging station.