高速路网

这个题目关键在于解析图片，而关于次短路的计算跟之前的迷阵突围是一样的（有一点区别在于求的是绝对的次短路）

```
int solve_algo() {
    struct PNG *png = new PNG();
    init_PNG(png);
    load(png, "pic/test1.png");
    State *state = new State();
    init_State(state);
    parse(state, png);
    std::cout << solve1(state) << std::endl;
    std::cout << solve2(state) << std::endl;
    save(png, "pic/test1.png");
    delete_State(state);
    delete_PNG(png);
    delete state;
    delete png;
    return -1;
}
```

在 part1 的代码中我们可以看到在 parse 之前，已经对图片和州进行初始化，并且有一个 load 函数，load 函数里面的代码，可以聚焦到这三行

```
  p->image = new_pixs;
  p->width = width;
  p->height = height;
```

我们可以猜测，这个函数里面已经对图片的宽高（以像素为单位）以及像素进行了获取，因为从 PNG 这个结构体可以看出 image 是一个 PXL 结构体指针，而 PXL 中有 RGB 颜色

那么现在就到了 parse 函数，观察代码之后可以看出我们要在 parse 函数中进行一些解析工作，从而得到我们进行迪杰斯特拉的一些条件，比如说州的编号、邻接矩阵等等，然后存在 state 当中

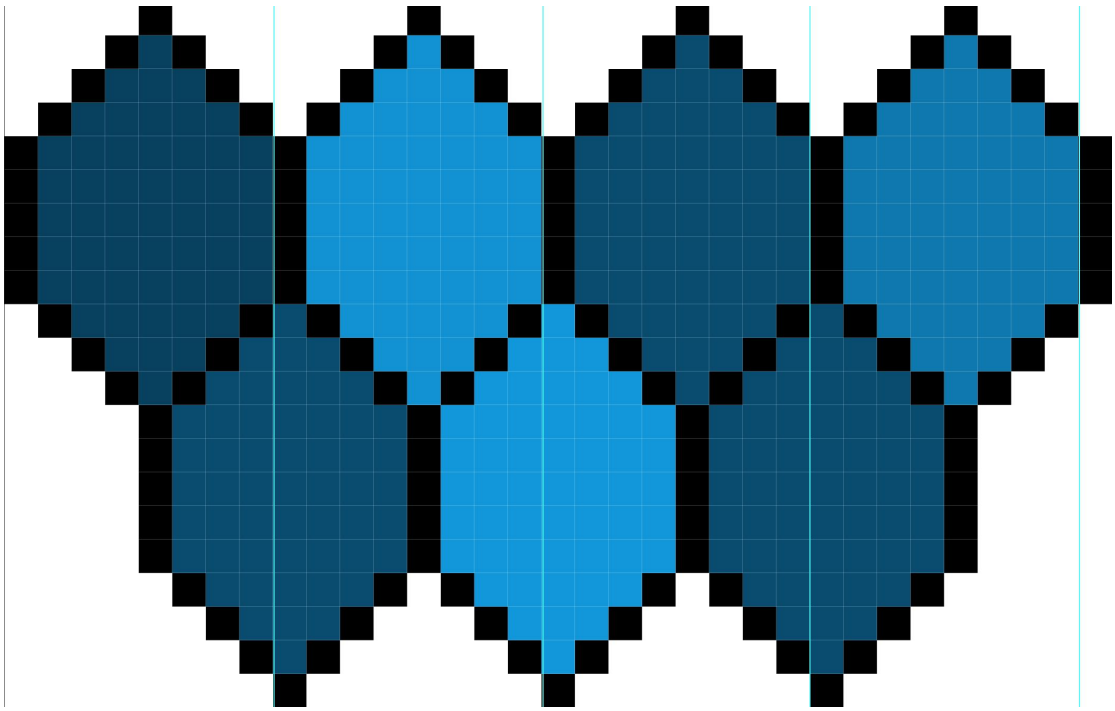
那么，首先我们来进行州的编号，从我们之前得到的 PNG 结构来看，我们进行编号就只能用到图片的宽高

我们先计算一列有多少个州，一行有多少个州

从下面图片可以看出，只看列的话，宽为 33 个像素，一个格子是一个像素，一共四个州，我们可以像下面这样子划分，将前 8 个格子划分为一个州，持续划分，最后一个黑色的格子忽略不计

所以一行的州的公式可以写为

(p->width)/8，后面的小数会自己约去，当然也可以直接(p->width-1)/8

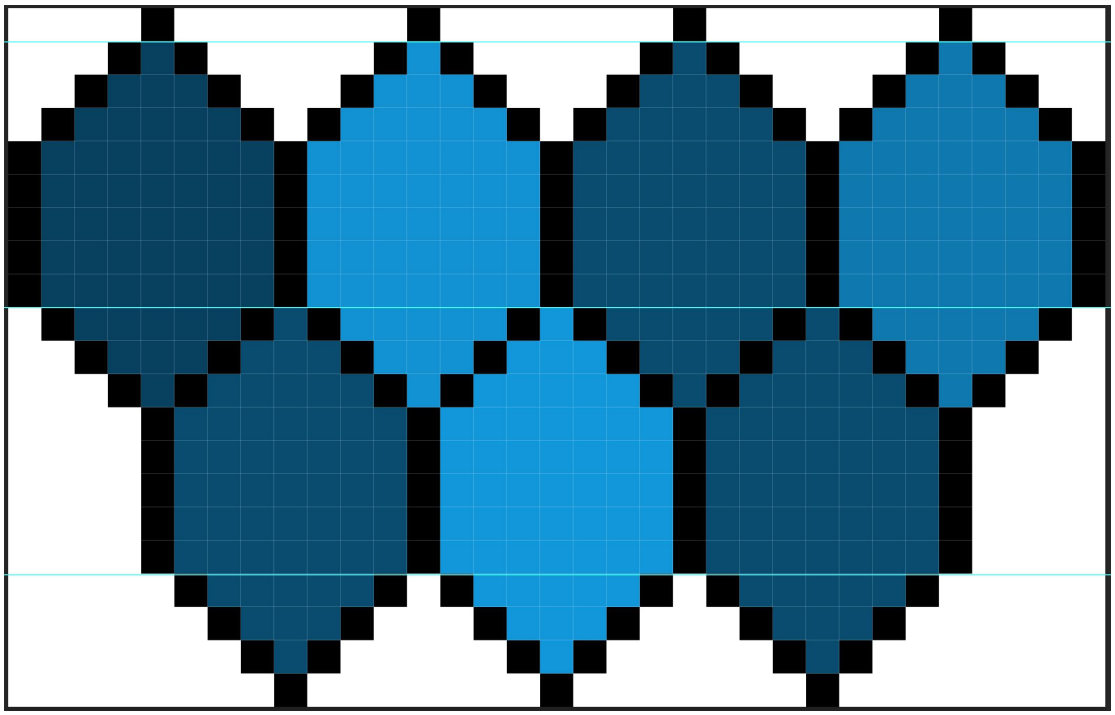


同时，我们可以看出规律，第二行比第一行少一个州。如果看其他的

图片的话，就更明显，奇数行的州都是之前算出来的每行的州数，偶数行都是少 1 一个

对于每列的州数，看到下面的参考线，除去上面的第一行的黑格子，后面每个州都占 8 个格子，然后最下面会多出 4 个格子，忽略不计，所以说一列的州数就是(p->height-5)/8

用 test2.png 也是可以用这个公式来计算出来的



再说一下每个州的产业发达程度

我们首先要定位州里面的一个像素，然后用 get_PXL 来获取 RGB 颜色，然后用下面这个公式来计算产业发达程度

$$industry_i = 255 \times 255 \times 3 - R_i \times R_i - G_i \times G_i - B_i \times B_i.$$

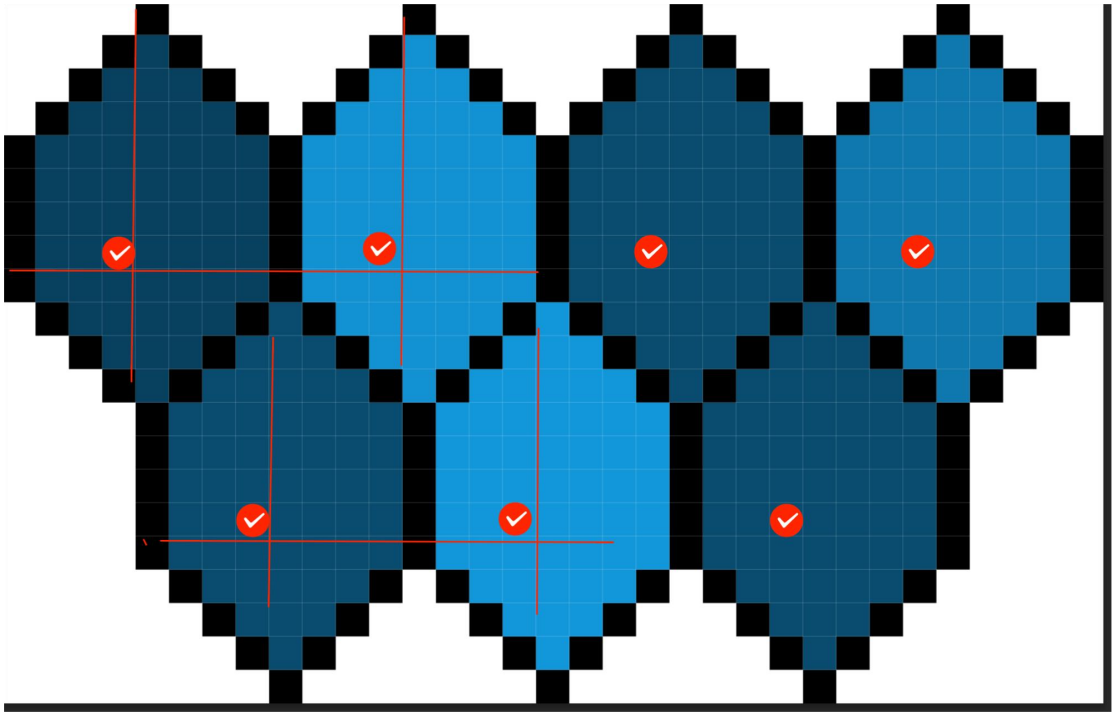
对于像素的定位，只要找到一个合适的点，确定规律即可

对于像素定位，我找的是下面有红勾的点

对于奇数行的州，首先定位到第一个州有一个 4 像素的偏移，之后每个州中的这个点就有一个固定的 8 像素的偏移

对于偶数行的州，首先定位到第一个州有一个 8 像素的偏移，之后的每个州中的这个点就有一个固定的 8 像素的偏移

对于上下的两个州，这两个点也有一个 8 像素的偏移



定位的时候从第一行（用 i 来记录）开始，区分奇数行和偶数行，对于每行州的个数（用 j 来记录），奇数行为之前计算出来的每行州的个数，偶数行则减一个。那么就是从第一行开始，一列一列依次定位

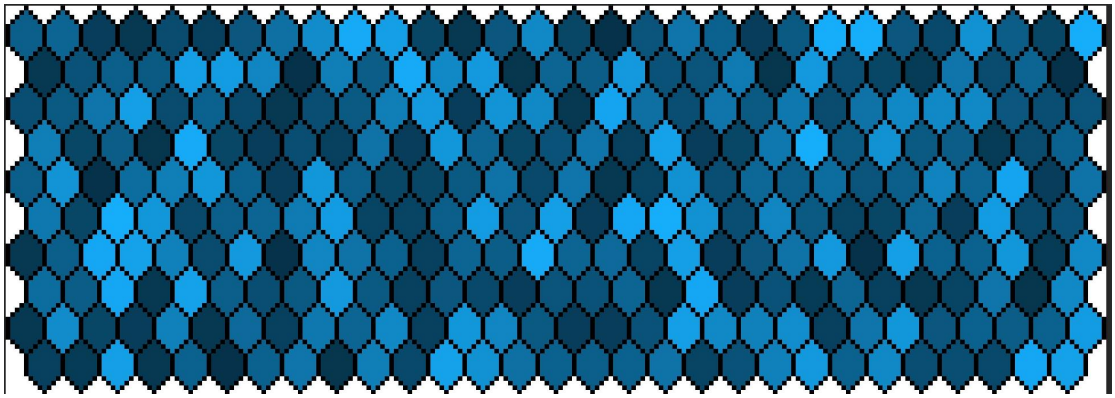
则用 get_PXL 的时候就是以下的公式

get_PXL(p,x+(j-1)*8,8+(i-1)*8),其中 x 是之前说的每行开始偏移的像素个数，为 4 或 8。注意这里 get_PXL 中的 x、y 坐标，x 坐标是

横向，y 坐标是纵向

在获取 RGB 之后，我们就可以计算州的产业发达程度。注意的是，我们在一行一列遍历州的时候同时对州进行编号，从而正好可以存储产业发达程度

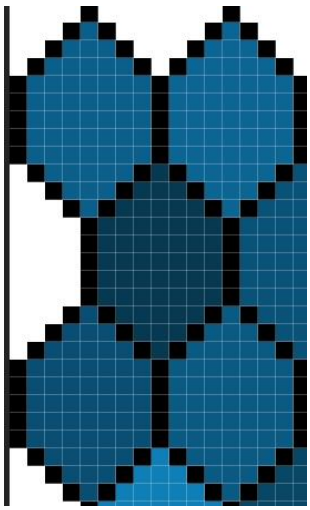
在得到每个州的编号以及产业发达程度之后，我们需要获得邻接矩阵



令州的编号为 k，一行一行来遍历，对于每一行的第一个，它与第二个有相连关系，这条边的权值为第二个州的产业发达程度（可以看成有向边），对于每行的最后一个州，它与倒数第二个州相连，这条边的权值为倒数第二个州的产业发达程度

对于其余每行中的其他列，记录它与相邻两个州的关系

对于偶数行，这里的每个州与左上左下、右上右下的州都有关系（除了最后一行，如果也是偶数行，那么只有左上、右上的关系）



这里需要找到这个州与左上左下、右上右下的州的编号关系

k-k（左上）=每列州个数

k-k（右上）=每列州个数-1

k（左下）-k=每列州个数-1

k（右下）-k=每列州个数

得到邻接矩阵之后就可以进行次短路的计算了

这里注意，如果说用了内存分配，需要回收，因为这里只用了数组，所以说不用回收

贪吃蛇

与之前的直接 bfs 不同的是，这里的移动个体变成了一条蛇，而不是单独的一个格子

首先我们需要存储蛇的身体，读取地图的时候先存储下是蛇身的点，不追求顺序,之后再进行处理,按照蛇身的顺序存储坐标于 snake[][2] 中

在 bfs 的时候，队列的结构体如下，需要存储坐标、步数和这个状态下的蛇身

```
typedef struct
{
    int x;
    int y;
    int step;
    int snake[20][2];
}Node;
```

bfs 中区别在于，判断是否可以进队的时候需要多判断下面的内容：

蛇的长度为 2 的时候不能走蛇尾

蛇的长度大于 2 的时候，不能访问蛇身，除了蛇尾

（我的做法是在出队的时候，将现在的蛇身（除了蛇尾）坐标的 vis 先设为 1），之后四个方向遍历完之后再恢复蛇身的 vis

要注意，每次进队的时候更新蛇身