结构体作为map的键时的重载

struct local {

int x, y;

bool operator==(const local& other) const {

return x == other.x && y == other.y;

}

};

struct localHash {

size\_t operator()(const local& p) const {

return hash<int>()(p.x) ^ hash<int>()(p.y);

}

};

解释；

//因为map就是一个哈希表，自创的结构体在map函数内部没有默认的哈希函数，需要我们自己来写，就如同我们在主函数内调用bfs函数需要在main（）前面写好bfs的内容一样，map函数也很会调用其内置函数，我们需要自己重载一个struct的Hash函数

//具体来说，这段代码使用了C++标准库中的hash模板类来计算local对象的哈希值。在operator()函数中，首先调用hash<int>()(p.x)和hash<int>()(p.y)分别计算local对象的x和y成员变量的哈希值。然后使用按位异或（XOR）操作符^将这两个哈希值进行混合，得到最终的哈希值。

这样定义localHash结构体并重载operator()函数，可以在使用unordered\_map等标准库容器时，为自定义的local类型提供哈希函数，以确保在容器中进行查找、插入等操作时能够正确地处理local对象。

//当我们在代码中看到size\_t时，通常可以理解为表示内存大小的无符号整数类型，用于确保能够正确处理内存相关的操作。

结构体定义优先队列时的重载

priority\_ queue<node>q;

bool operator<(const node &a, const node &b){

return a. diss>b. diss;

}//重载定义，对于结构体来说是这样的，pair则不用

如果想要把map中的所有元素都导出到一个数组中

typedef pair<string, long long> PII;

unordered\_ map< string, int> mp ;

vector<PII> mpp(mp.begin(),mp.end());