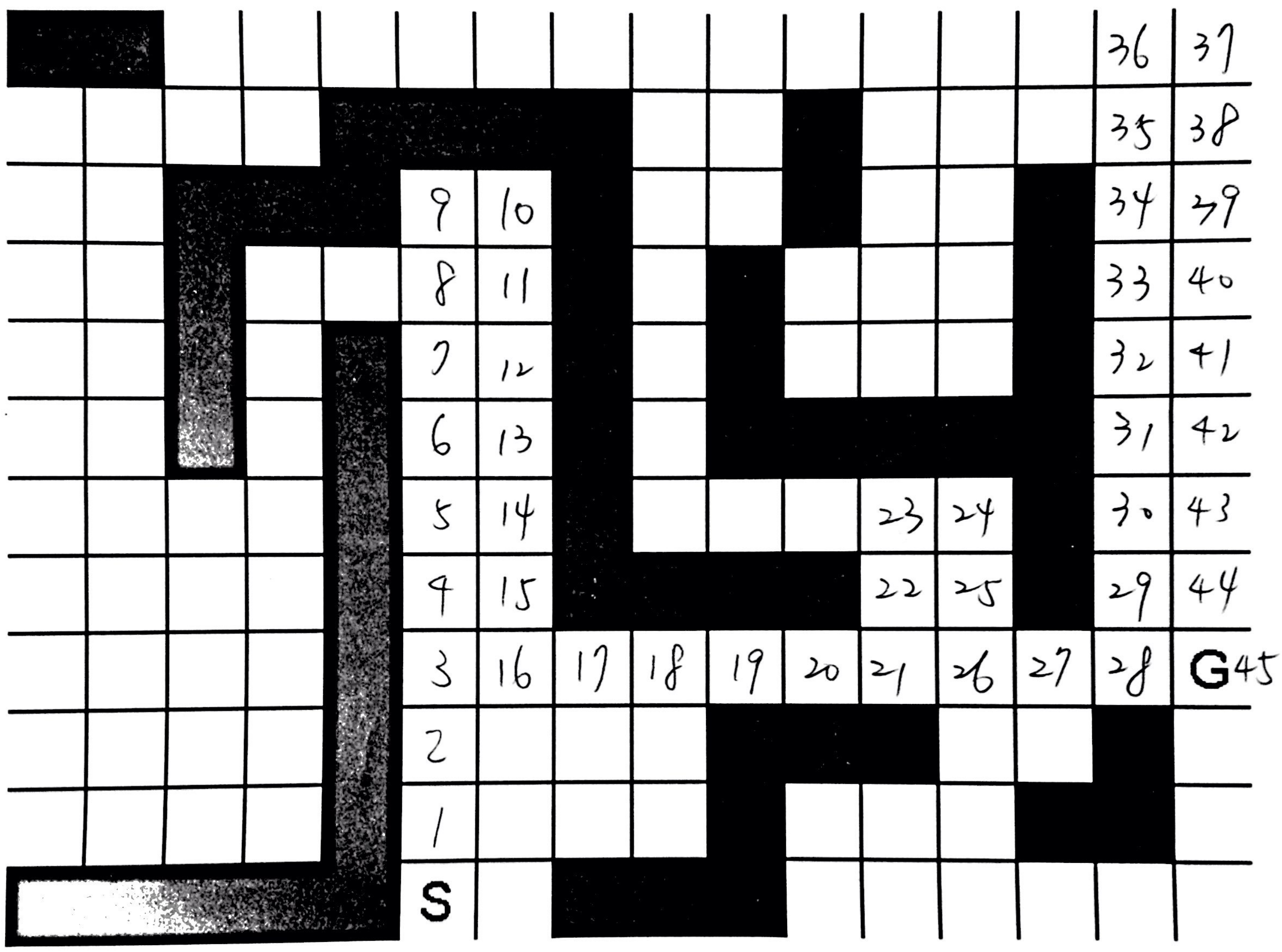
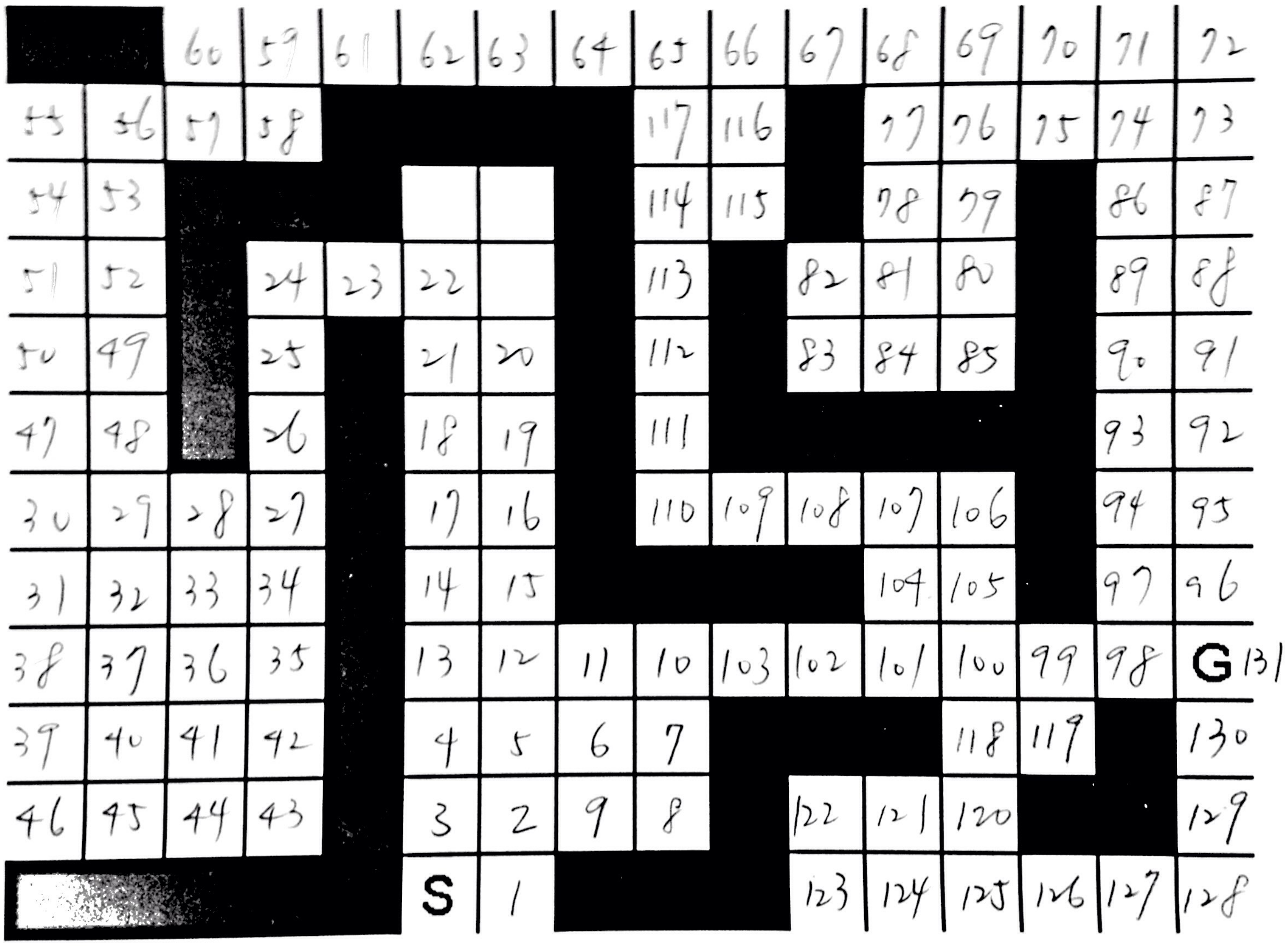
NESW

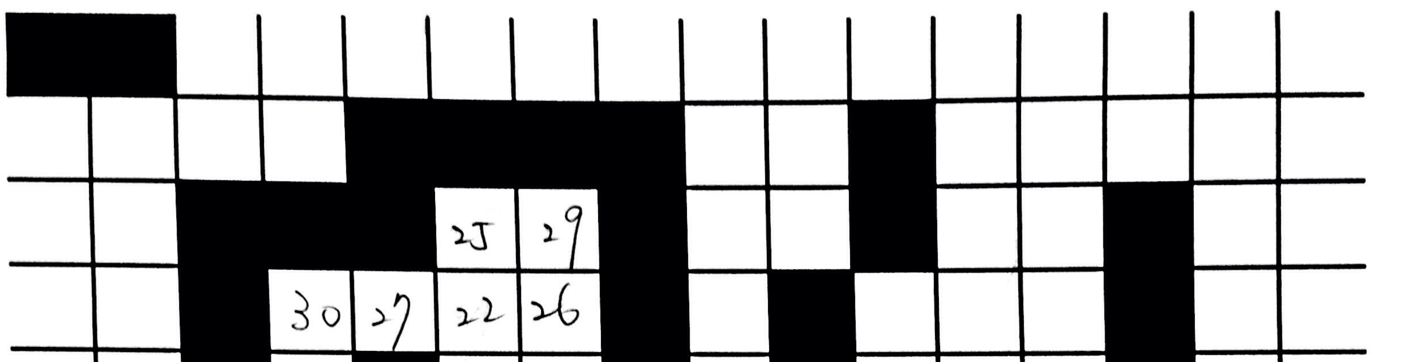

S
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
G45

W E S N



■	■	60	59	61	62	63	64	65	66	67	68	69	70	71	72
55	56	57	58	■	■	■	■	117	116	■	77	76	75	74	73
54	53	■	■	■			■	114	115	■	78	79	■	86	87
51	52	■	24	23	22		■	113	■	82	81	80	■	89	88
50	49	■	25	■	21	20	■	112	■	83	84	85	■	90	91
47	48	■	26	■	18	19	■	111	■	■	■	■	■	93	92
30	29	28	27	■	17	16	■	110	109	108	107	106	■	94	95
31	32	33	34	■	14	15	■	■	■	■	104	105	■	97	96
38	37	36	35	■	13	12	11	10	103	102	101	100	99	98	G 131
39	40	41	42	■	4	5	6	7	■	■	■	118	119	■	130
46	45	44	43	■	3	2	9	8	■	122	121	120	■	■	129
■	■	■	■	■	S	1	■	■	■	123	124	125	126	127	128

NESW


25 29
30 27 22 26
33 19 23
38 16 20
44 12 17 45 40 34 39
8 13 31 35 46
5 9 14 18 21 24 28 32 36 41 G47
3 6 10 15 37 42
1 4 7 11 43
S 2

2.

DFS and BFS both have its pros and cons depending on different cases. Normally DFS has much lower memory requirement because it doesn’t have to go through all the nodes (or steps) until finding the goal. However we should consider different cases. On one hand, if the goal is far from the initial node, then BFS would take lots of steps to reach the goal, while DFS might reach that within less steps. On the other hands, if the goal is closed to the initial node, then BFS might be better solution, while the DFS with bad direction might take more than enough steps to get there. In conclusion, which is better should be depending on different situation.

3.

{ } | S0

{S0} | A1, B2, C8

{A1, B2, C8} | C4, D5

{B2, C4, D5} | C4, E3

{E3, C4, D5} | G10

{C4, D5, G10} | E6, D6, G14

{D5, G10} G7

{G7} | none

⇒ G7

⇒ Path:

S → A → D → G #.

Cost to node			Backpointers:
S	:	0	S → Null
A	:	1	A → S
B	:	2	B → S
C	:	8 → 4	C → S → A
D	:	4	D → A
E	:	3	E → B
G	:	7	G → C → D

4.

(a)

Path= [1 3 1 3]

Number of moves: 4

(b)

Path= [1 3 3 2 4 1 3 1]

Number of moves= 8

(c)

Path=[3 2 4 1 4 2 3 1 1 3 2 4 1 3]

Number of moves: 14

Bonus:

I wrote an extra function to calculate heuristic value at the bottom of my code. If you want to test it, please remove the comment mark.

(a)

Path= [1 3 1 3]

Number of moves: 4

(b)

Path= [1 3 3 2 4 1 3 1]

Number of moves= 8

(c)

Path=[3 2 4 1 4 2 3 1 1 3 2 4 1 3]

Number of moves: 14

The results are exactly the same, and it took much less time to find the path!!