Informed Search

"**Knowledge is power**"

How can we use knowledge to improve our search algorithm?

- One common type of knowledge is a scoring function that estimates how good a node is as the next node to expand.

- Note the word "estimates". If the scoring function was perfect, then we wouldn't need any search at all.

- The word "heuristic" means trial-and-error, exploratory, unguaranteed, rule of thumb

Informed Search

- **EXPAND**-applies all the operators of the problem to a node and returns all the successors of this node.

The evaluation *f(n) is made up of two parts:*

- g(n) cost from start state to node **n**
- h(n) **estimate** cost from **n** to goal

$$f(n) = g(n) + h(n)$$

Hill-climbing search

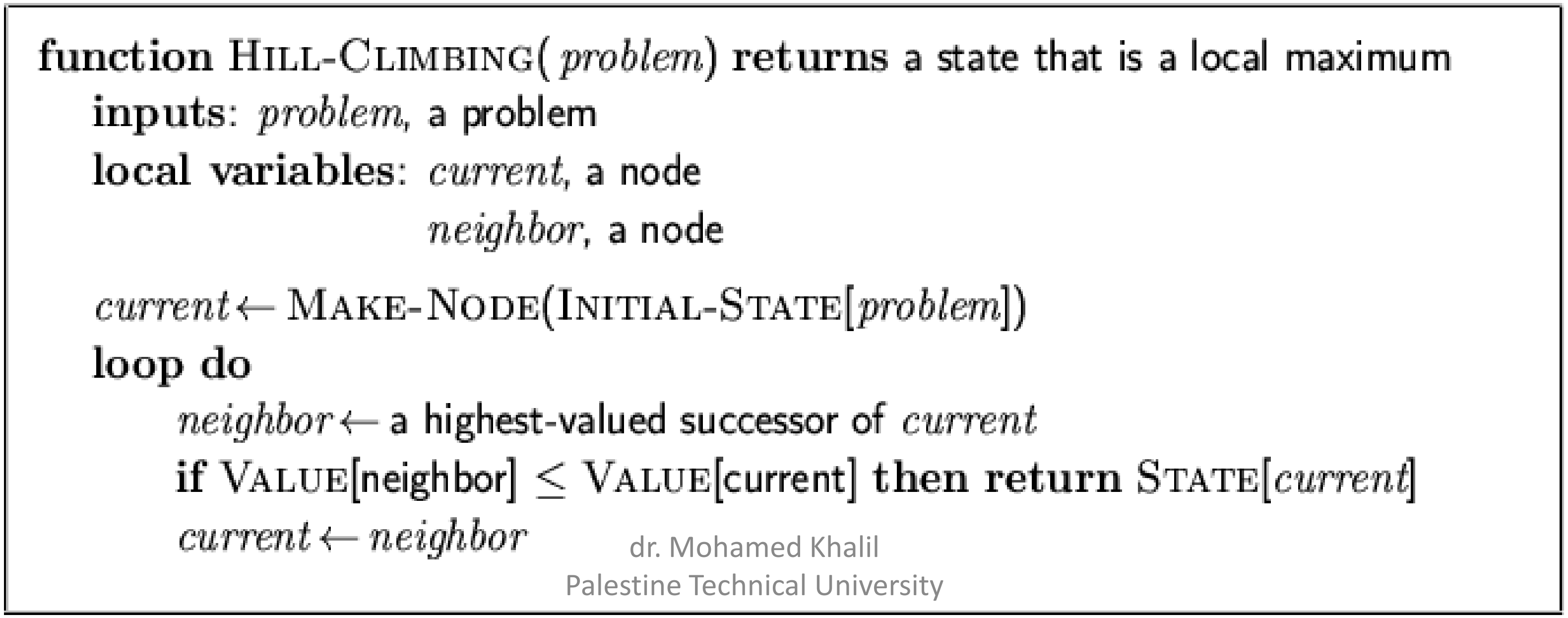
without backtracking

Steps:

1. Start State = Current State.
2. Child X= All results states when we apply all available methods on start state.
3. Apply Heuristic Function on all child.
4. Choose the state that has a maximum (better) value of Heuristic function.

Hill-climbing search

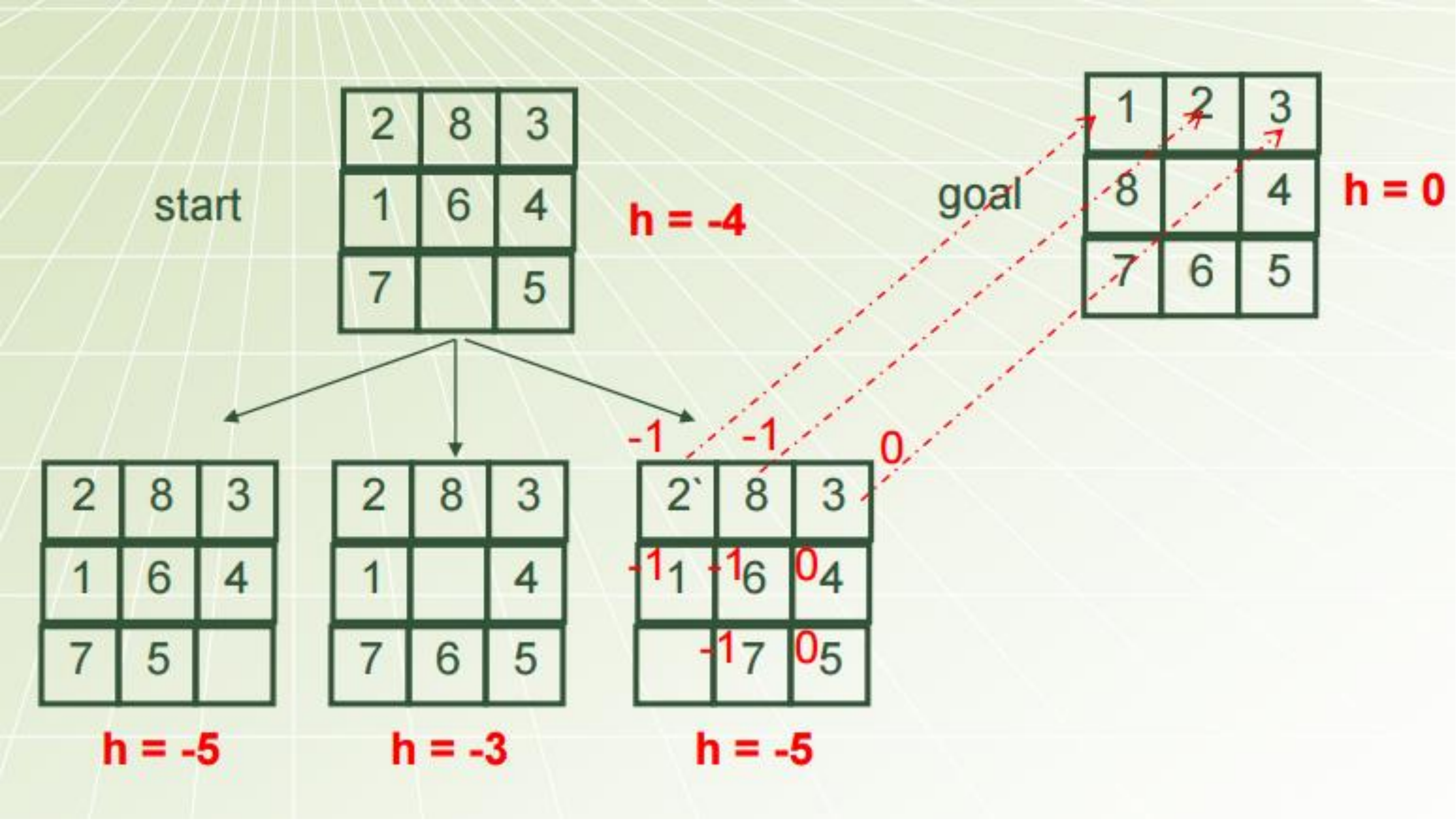
- Hill climbing is an iterative improvement algorithm that is similar to greedy best-first search, except that **backtracking is not permitted**

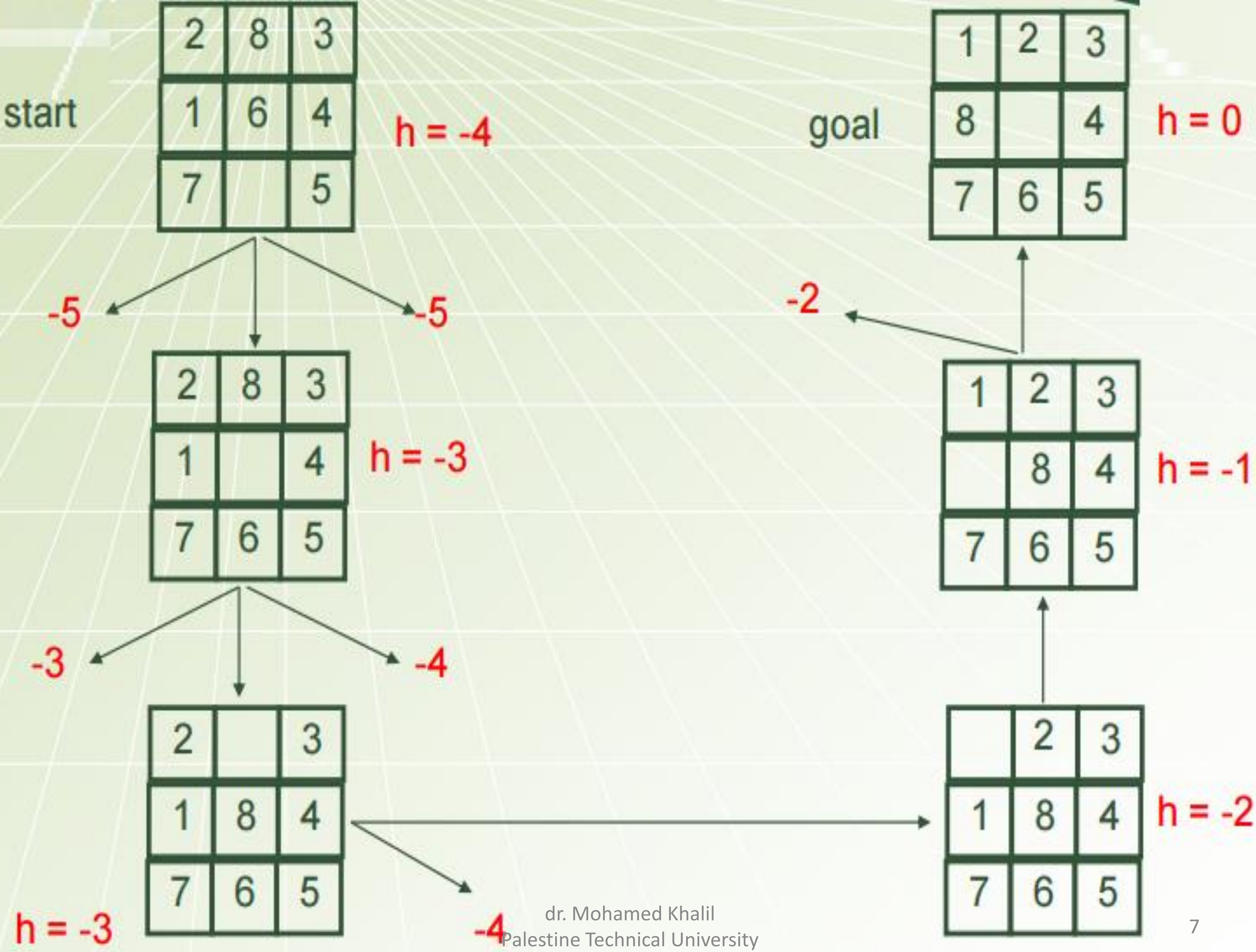


```
function HILL-CLIMBING(problem) returns a state that is a local maximum
    inputs: problem, a problem
    local variables: current, a node
                     neighbor, a node

    current $\leftarrow$ MAKE-NODE(INITIAL-STATE[problem])
    loop do
        neighbor $\leftarrow$ a highest-valued successor of current
        if VALUE[neighbor] $\leq$ VALUE[current] then return STATE[current]
        current $\leftarrow$ neighbor
```

f(n) = -(number of tiles out of place)





start

2	8	3
1	6	4
7		5

h = -4

-5

-5

2	8	3
1		4
7	6	5

h = -3

-3

-4

2		3
1	8	4
7	6	5

h = -3

-4

	2	3
1	8	4
7	6	5

h = -2

1	2	3
	8	4
7	6	5

h = -1

-2

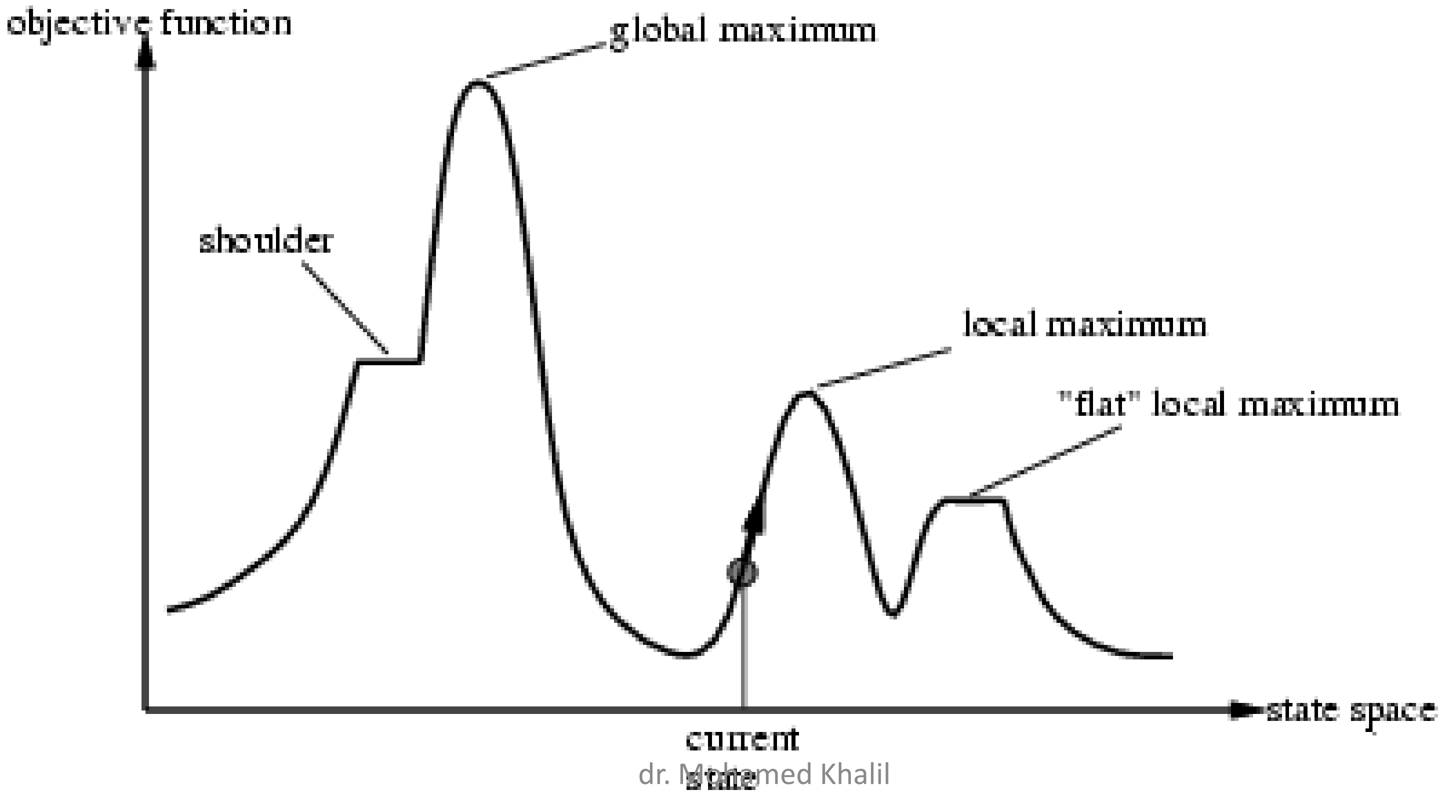
goal

1	2	3
8		4
7	6	5

h = 0

Hill-climbing search
Disadvantage

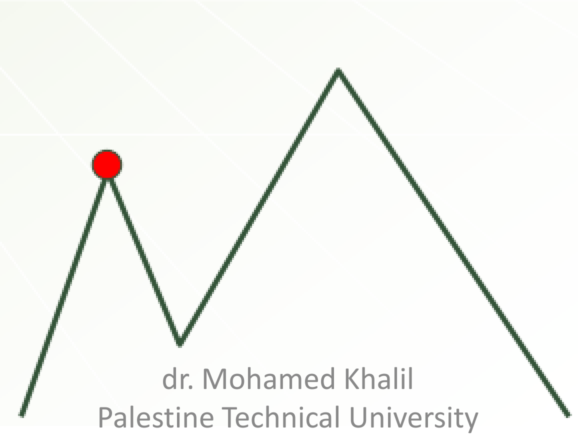
- Problem: depending on initial state, can get stuck in local maxima
-



Hill-climbing search
Disadvantage

local maxima

- Peaks that aren’t the highest point in the space.
- A state that is better than all of its neighbors, but not better than some other states far away.

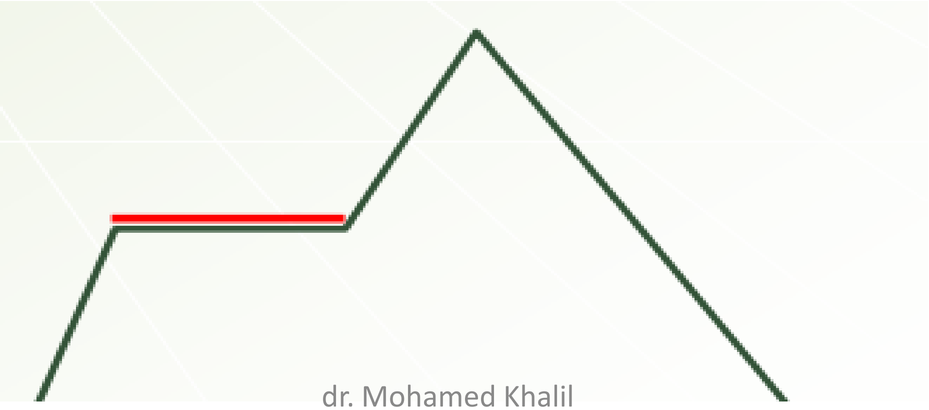


Hill-climbing search

Disadvantage

Plateaus

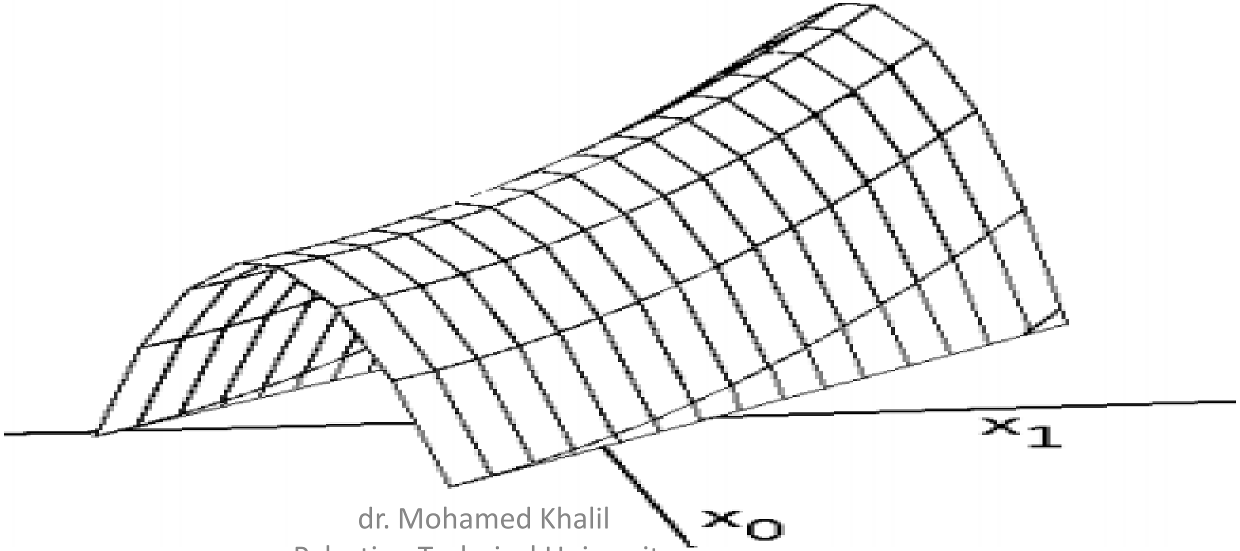
- the space has a broad flat region that gives the search algorithm no direction (**random walk**).
  - A flat area of the search space in which all neighboring states have the same value



Hill-climbing search
Disadvantage

Ridges

flat like a plateau, but with drop-offs to the sides; steps to the North, East, South and West may go down, but a step to the NW may go up.



Hill-climbing variations

- Stochastic hill-climbing(**probability)**
  - **Random** selection among the uphill moves.
- First-choice hill-climbing
  - stochastic hill climbing by generating successors randomly until a better one is found
  - Useful when there are a very large number of successors
- Random-restart hill-climbing
  - Tries to avoid getting stuck in local maxima.

Beam-search

- It uses the heuristic f(n) = h(n).
- it *keeps only a set of the best candidate* nodes for expansion.(***throws the rest way***.)
- More memory efficient.
- Optimal? and complete ? No

Nodes can be discarded which could result in the optimal path.

BEST-FIRST SEARCH (BEST-FS)

- The search space is evaluated according to a heuristic function.
- Nodes yet to be evaluated are kept on an OPEN list(represented as a priority queue)
- Nodes that have already been evaluated are stored on a CLOSED list.
- The OPEN list is then built in **order of f(n).**

This makes best-first search fundamentally ***greedy***

because it always chooses the **best local opportunity** in the search ***frontier***.

BEST-FIRST SEARCH (BEST-FS)

- Best-first search
- **Idea**: use an evaluation function f(n) for each node
- f(n) provides an estimate for the total cost.
- Expand the node n with smallest f(n).
- Implementation: Order the nodes in fringe increasing order of cost.
- Special cases:
  - greedy best-first search
  - A* search

Frontier

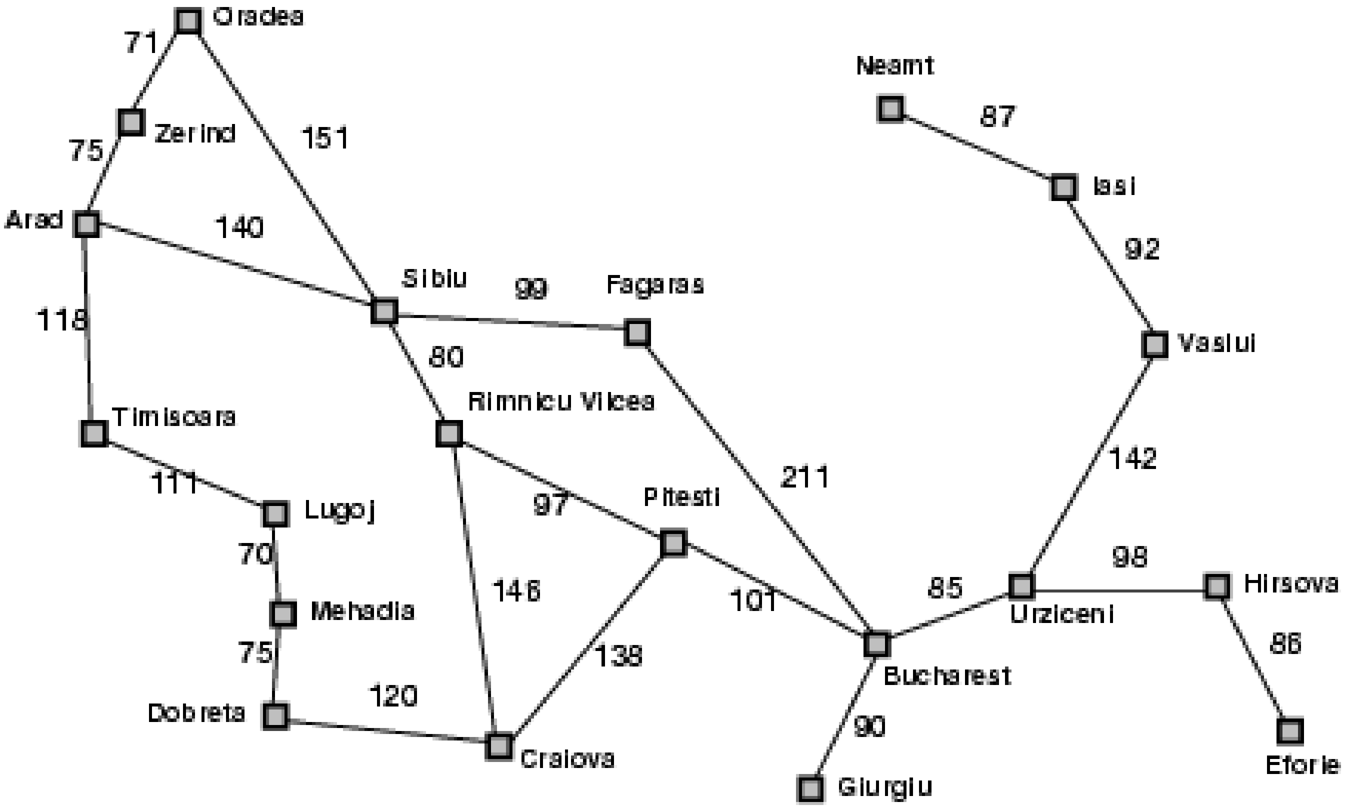
- *The search frontier is defined as the set of node opportunities that can be searched next.*
- *In Best-First search, the frontier is a priority queue sorted in f(n) order.*
- *Given the strict order of f(n), the selection of the node to evaluate from the priority queue is greedy.*

Properties of best-first search

- Complete? No – can get stuck in loops.
-
- Time? $O(b^m)$, but a good heuristic can give dramatic improvement
-
- Space? $O(b^m)$ -- keeps all nodes in memory
-
- Optimal? No

a solution can be found in a longer path (higher *h(n) with a* lower *g(n) value.*

Romania with step costs in km



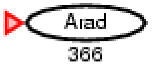
Straight–line distance
to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

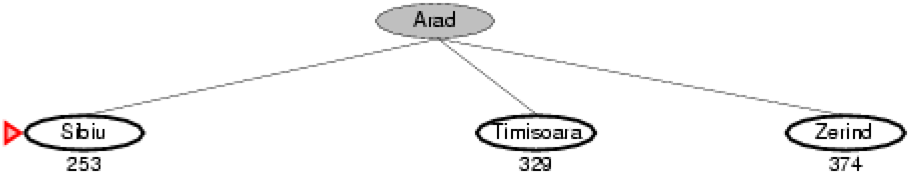
Best-first search

- Evaluation function *$f(n) = h(n)$* (heuristic)
- = estimate of cost from *n* to *goal*
-
- e.g., $h_{SLD}(n)$ = straight-line distance from *n* to Bucharest
-
- Greedy best-first search expands the node that appears to be **closest to goal**
-

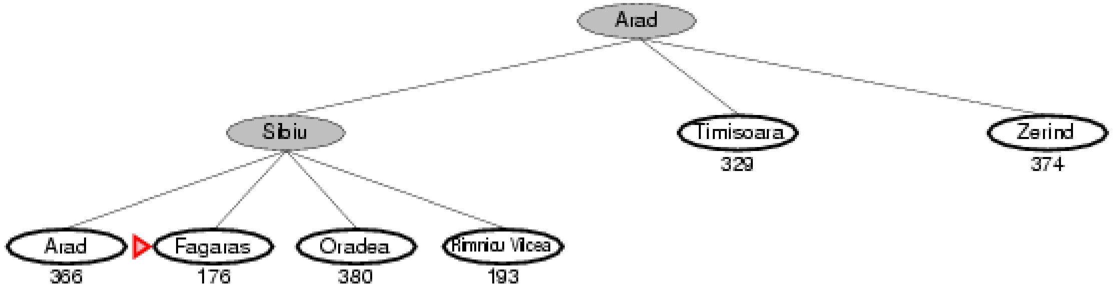
Greedy Best-first search example



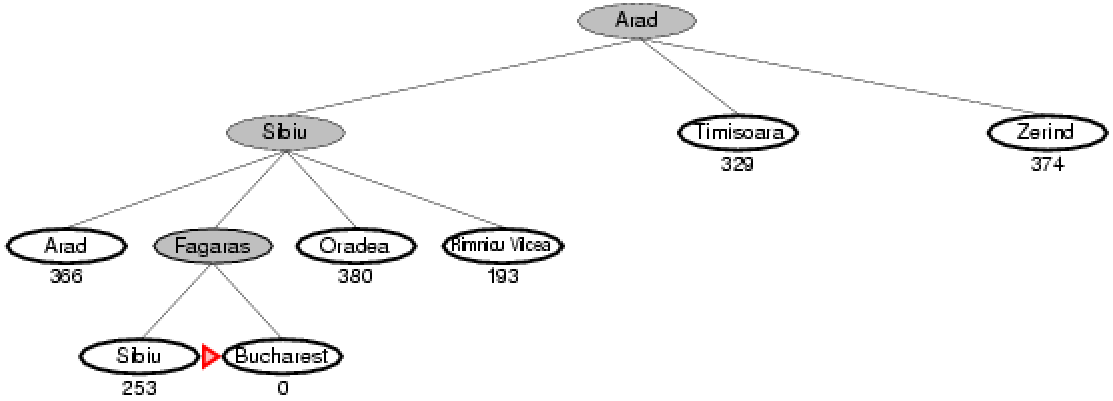
Greedy best-first search example



Greedy best-first search example

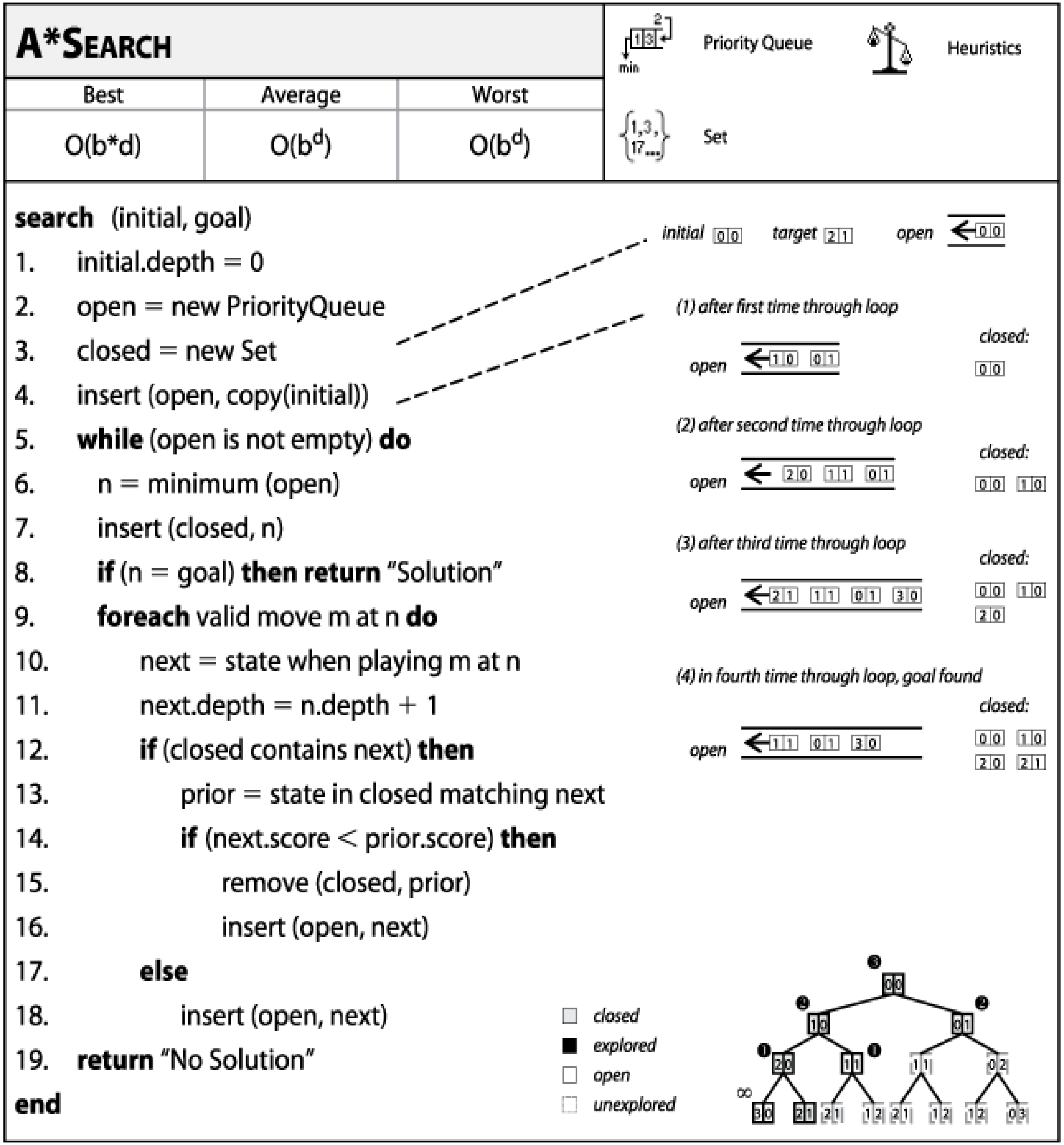


Greedy best-first search example



A^* search

- Idea: avoid expanding paths that are already expensive
-
- Evaluation function *f(n) = g(n) + h(n)*
- *g(n)* = cost so far to reach *n*
- *h(n)* = estimated cost from *n* to goal
- *f(n)* = estimated total cost of path through *n* to goal



A*SEARCH

Best	Average	Worst
O(b*d)	O(b^d)	O(b^d)

Priority Queue

Heuristics

Set

2
1 3
min
{1,3,
17...}

```
search (initial, goal)
    initial.depth = 0
    open = new PriorityQueue
    closed = new Set
    insert (open, copy(initial))
    while (open is not empty) do
      n = minimum (open)
      insert (closed, n)
      if (n = goal) then return "Solution"
      foreach valid move m at n do
          next = state when playing m at n
          next.depth = n.depth + 1
          if (closed contains next) then
              prior = state in closed matching next
              if (next.score < prior.score) then
                  remove (closed, prior)
                  insert (open, next)
          else
              insert (open, next)
    return "No Solution"
end
```

initial 0 0 *target* 2 1 *open* ← 0 0

(1) after first time through loop

open ← 1 0 0 1

closed: 0 0

(2) after second time through loop

open ← 2 0 1 1 0 1

closed: 0 0 1 0

(3) after third time through loop

open ← 2 1 1 1 0 1 3 0

closed: 0 0 1 0 2 0

(4) in fourth time through loop, goal found

open ← 1 1 0 1 3 0

closed: 0 0 1 0 2 0 2 1

□ *closed*

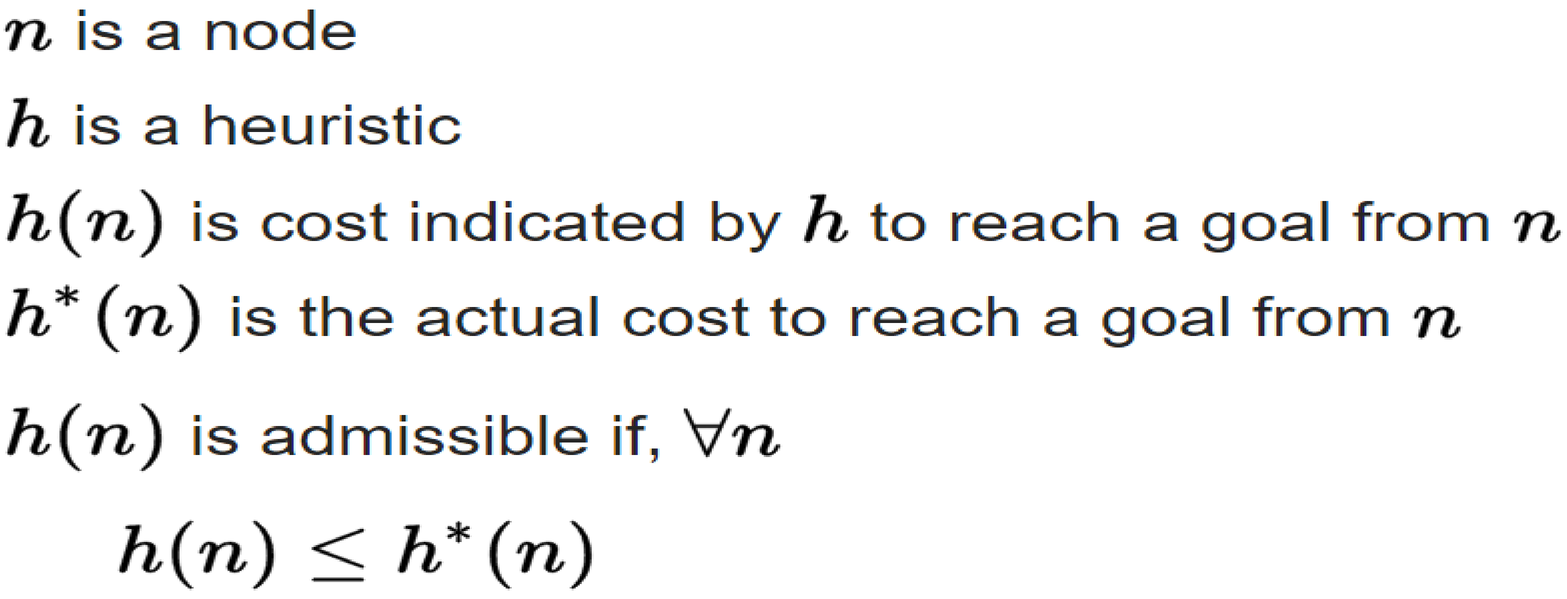
■ *explored*

□ *open*

⬚ *unexplored*

Admissible heuristic

- a heuristic function is said to be **admissible** if it never overestimates the cost of reaching the goal, i.e. the cost it estimates to reach the goal is not higher than the lowest possible cost from the current point in the path.



n is a node

h is a heuristic

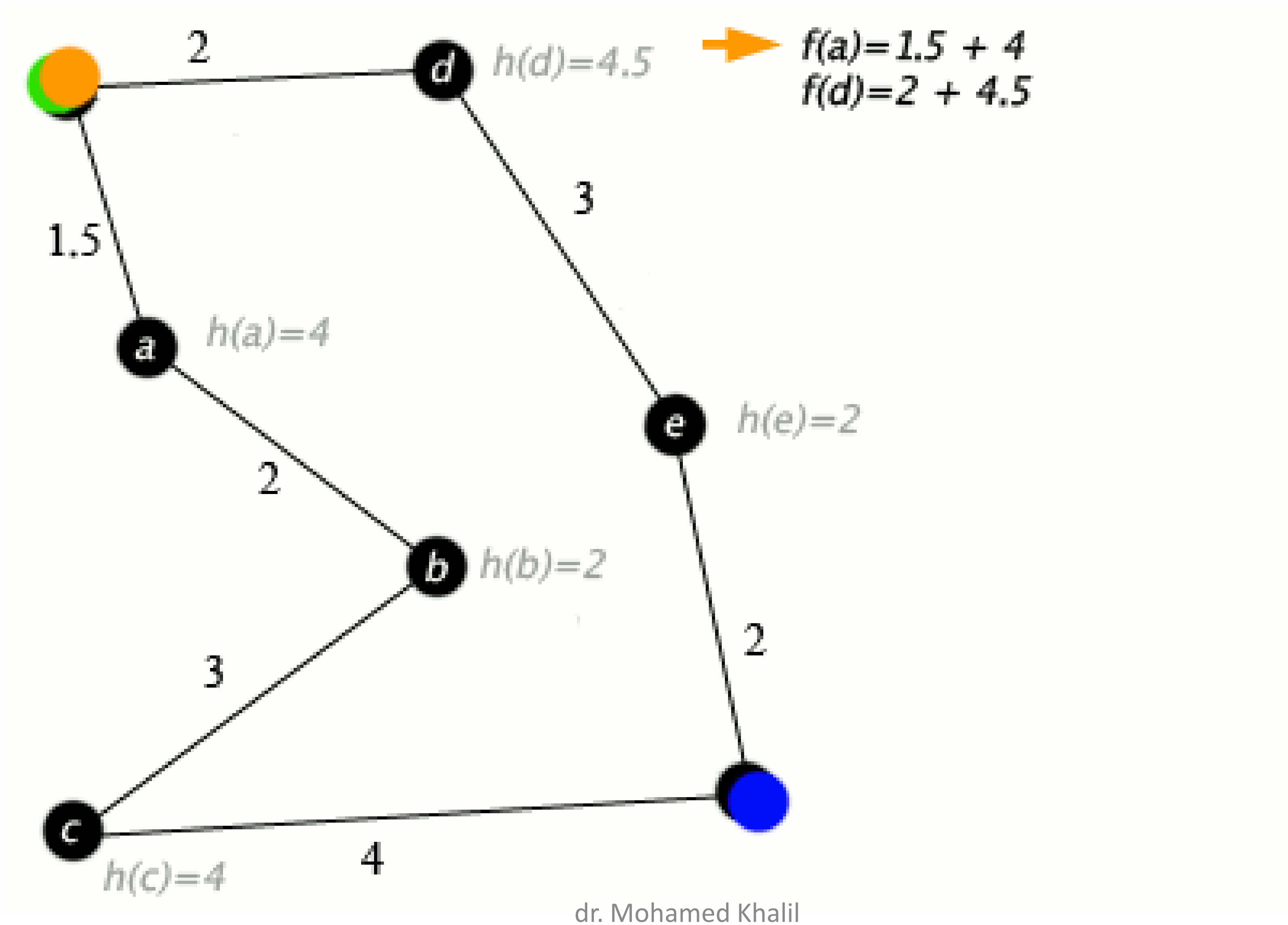
$h(n)$ is cost indicated by h to reach a goal from n

$h^*(n)$ is the actual cost to reach a goal from n

$h(n)$ is admissible if, $\forall n$

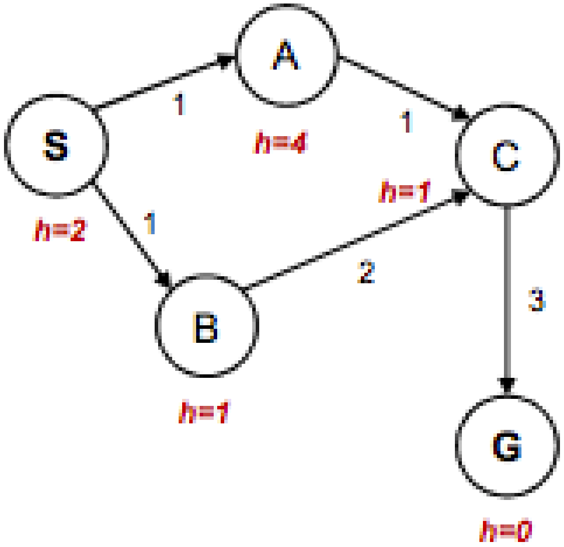
$$h(n) \leq h^*(n)$$

A* search

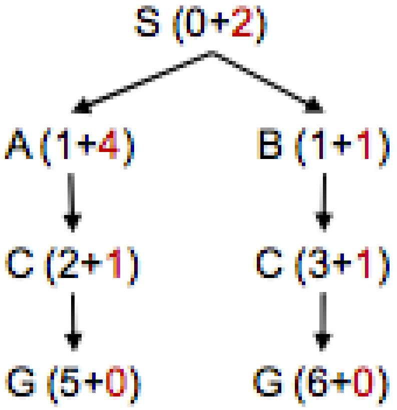


A* search

State space graph



Search tree


S (0+2)

A (1+4)

B (1+1)

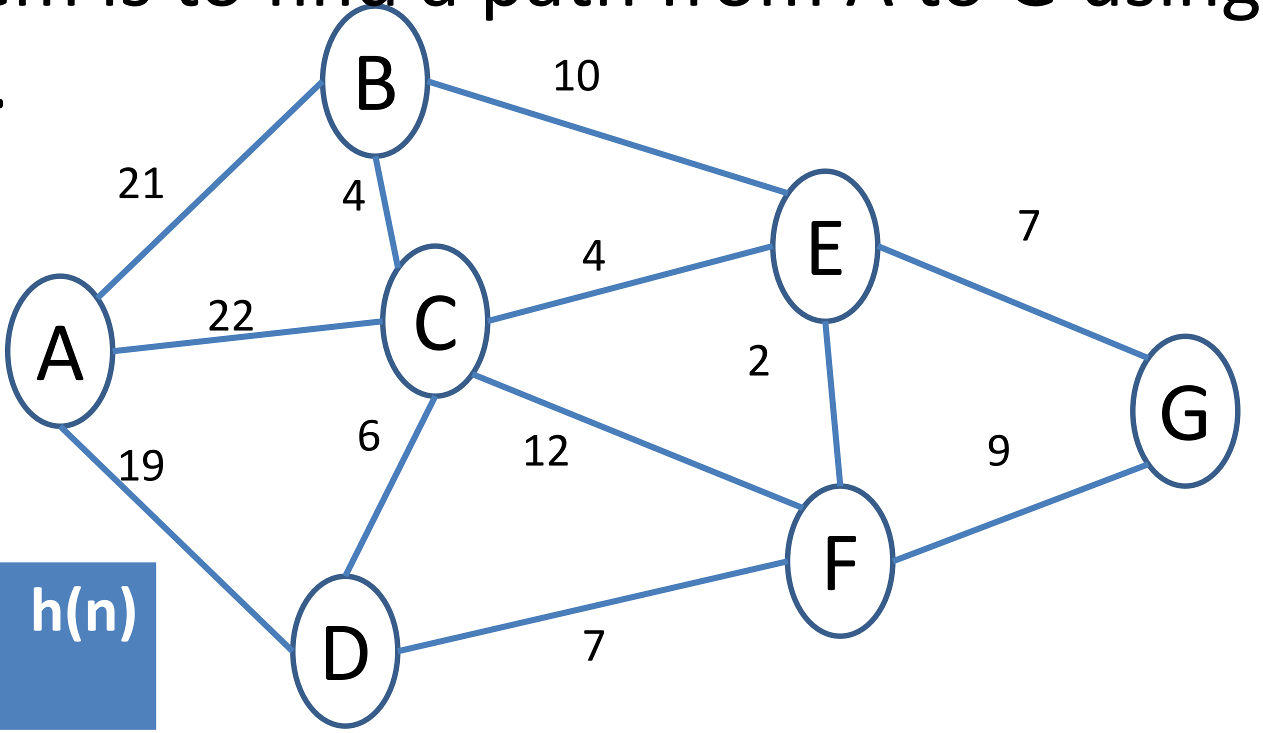
C (2+1)

C (3+1)

G (5+0)

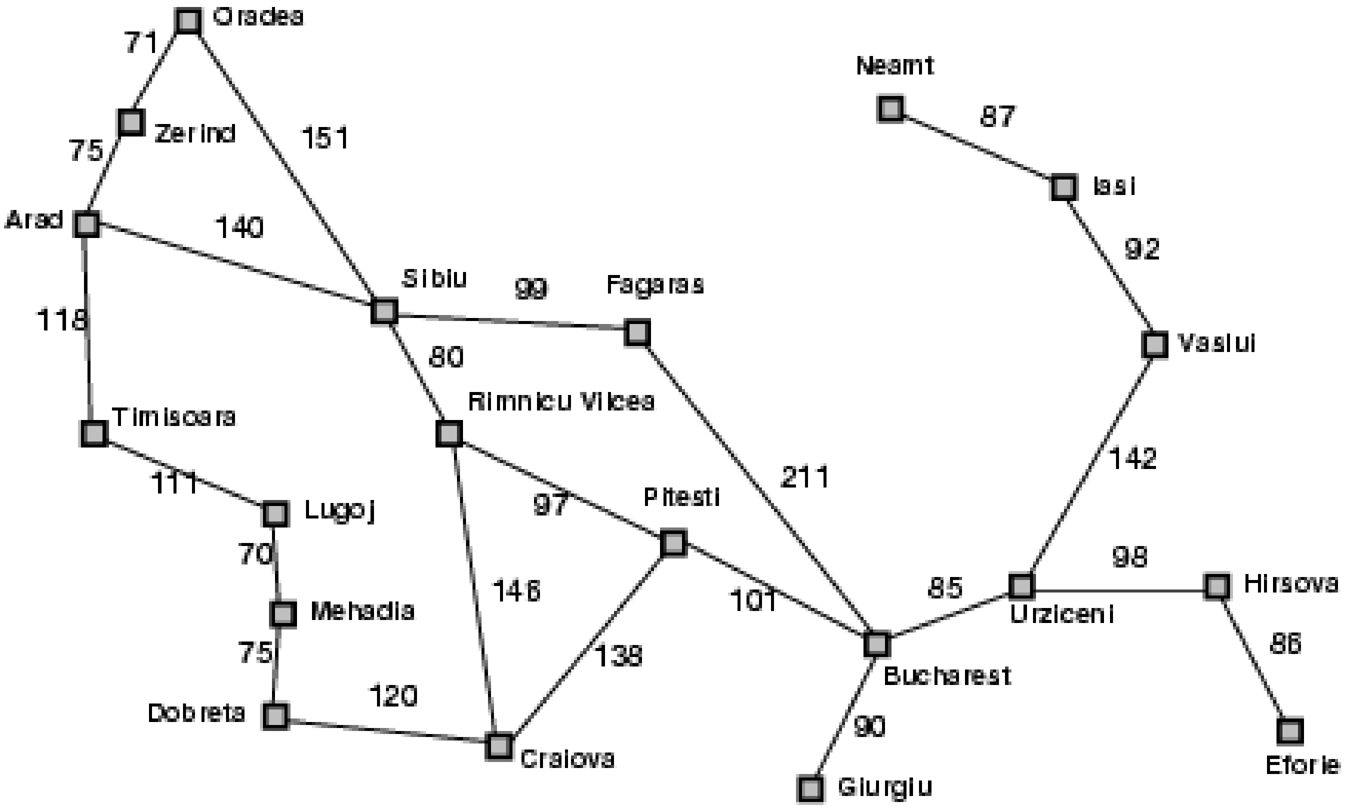
G (6+0)

the problem is to find a path from A to G using A* algorithm.



nodes	h(n)
A	34
B	19
C	14
D	18
E	7
F	9
G	0

Romania with step costs in km



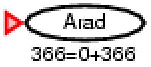
Straight–line distance
to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Open List:

Arad

A^* search example



We start with our initial state Arad. We make a node and add it to the open list. Since it's the only thing on the open list, we expand the node.

Think of the open list as a priority queue (or heap) that sorts the nodes inside of it according to their g()+h() score.

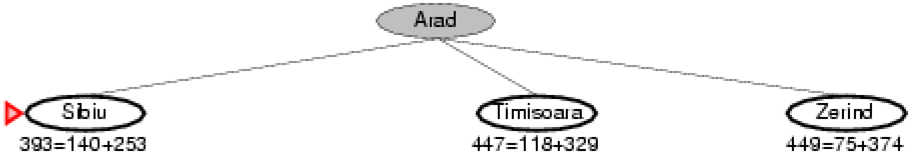
Open List:

Sibiu

Timisoara

Zerind

A* search example



We add the three nodes we found to the open list.

We sort them according to the g()+h() calculation.

Open List:

Rimricu Vicea

Fagaras

Timisoara

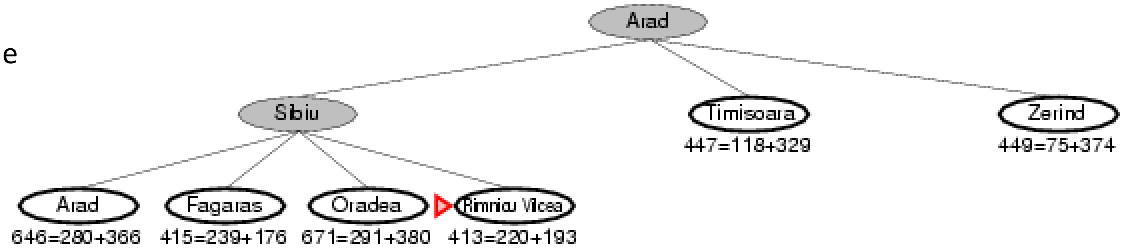
Zerind

~~Arad~~

Oradea

We've been to Arad before. Don't list it again on the open list.

A* search example



When we expand Sibiu, we run into Arad again. But we've already expanded this node once; so, we don't add it to the open list again.

Open List:

Rimricu Vicea

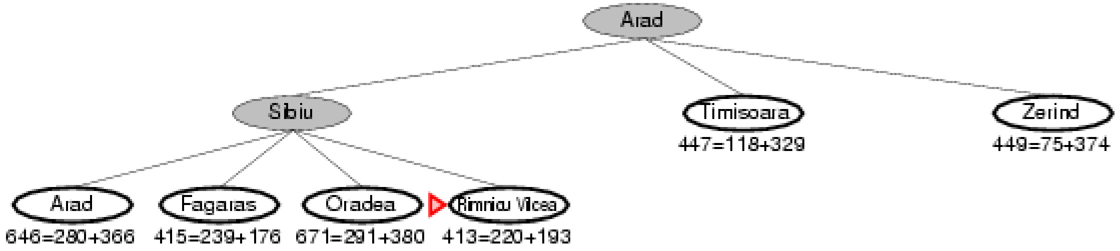
Fagaras

Timisoara

Zerind

Oradea

A* search example



We see that Rimricu Vicea is at the top of the open list; so, it’s the next node we will expand.

A* search example

Open List:

Fagaras

Pitesti

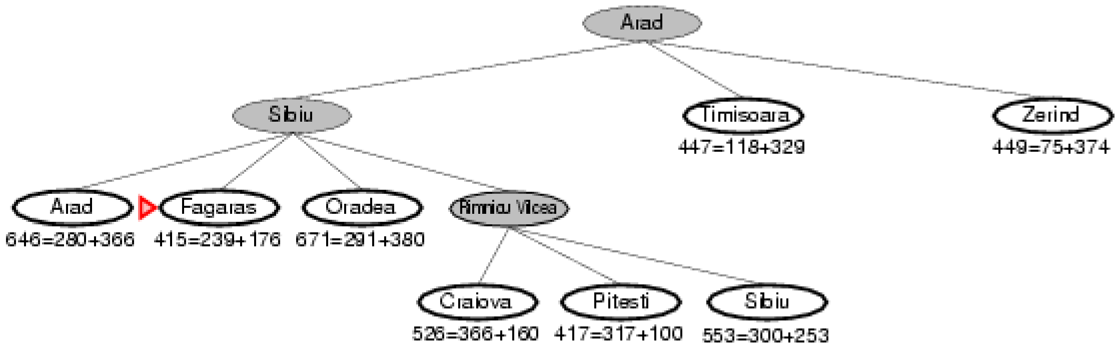
Timisoara

Zerind

Craiova

~~Sibiu~~

Oradea



When we expand Rimricu Vicea, we run into Sibiu again.

But we've already expanded this node once; so, we don't add it to the open list again.

A* search example

Open List:

Fagaras

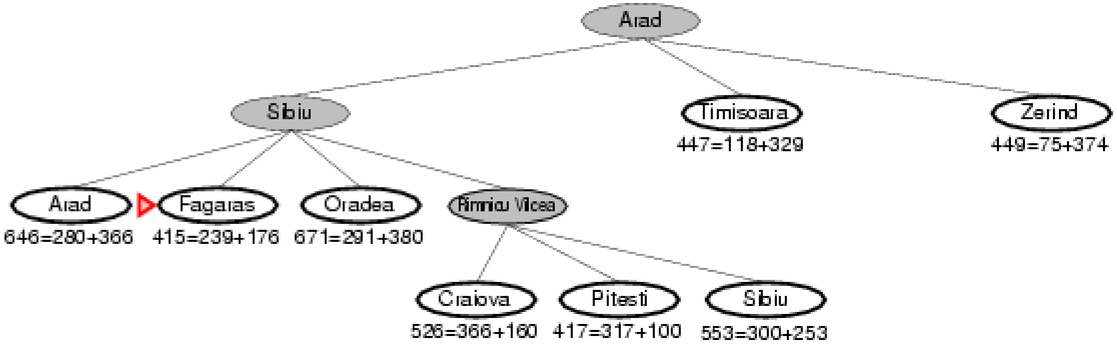
Pitesti

Timisoara

Zerind

Craiova

Oradea



Fagaras will be the next node we should expand – it’s at the top of the sorted open list.

Open List:

Pitesti

Timisoara

Zerind

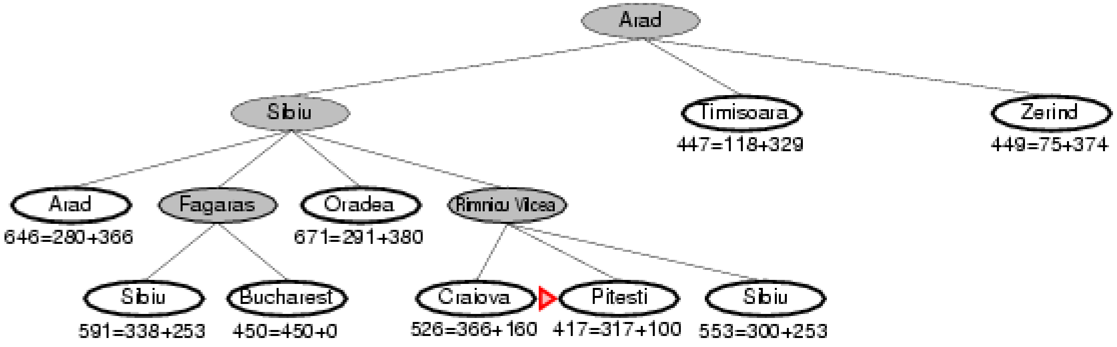
Bucharest

Craiova

~~Sibiu~~

Oradea

A* search example



When we expand Fagaras, we find Sibiu again. We don’t add it to the open list.

We also find Bucharest, but we’re not done. The algorithm doesn’t end until we “expand” the goal node – it has to be at the top of the open list.

Open List:

Pitesti

Timisoara

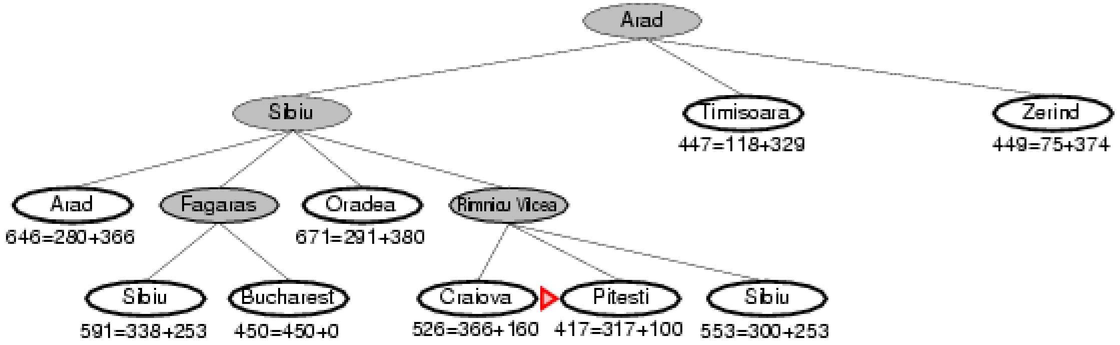
Zerind

Bucharest

Craiova

Oradea

A* search example



It looks like Pitesti is the next node we should expand.

Open List:

Bucharest

Timisoara

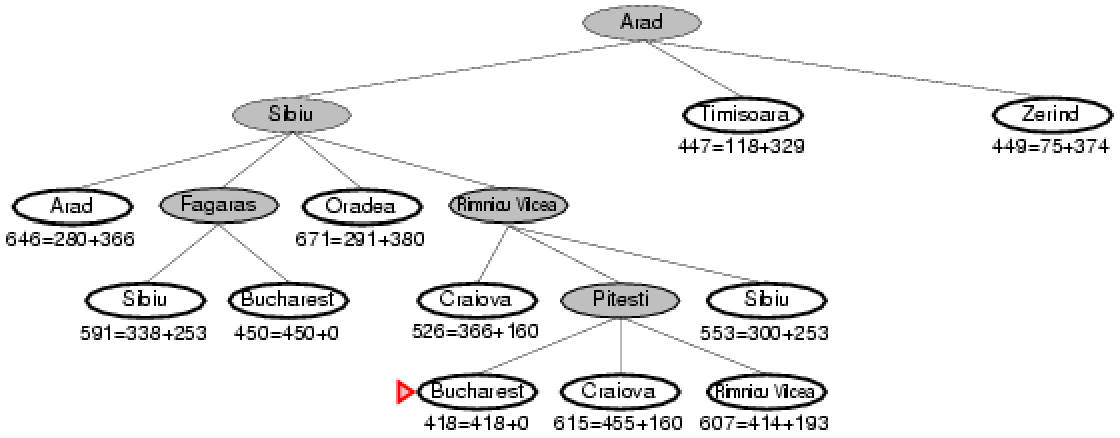
Zerind

Craiova

~~Rimricu Vicea~~

Oradea

A^* search example



We just found a better value for Bucharest; so, it got moved higher in the list. We also found a worse value for Craiova – we just ignore this.

And of course, we ran into Rimricu Vicea again. Since it's already been expanded once, we don't re-add it to the Open List.

Open List:

Bucharest

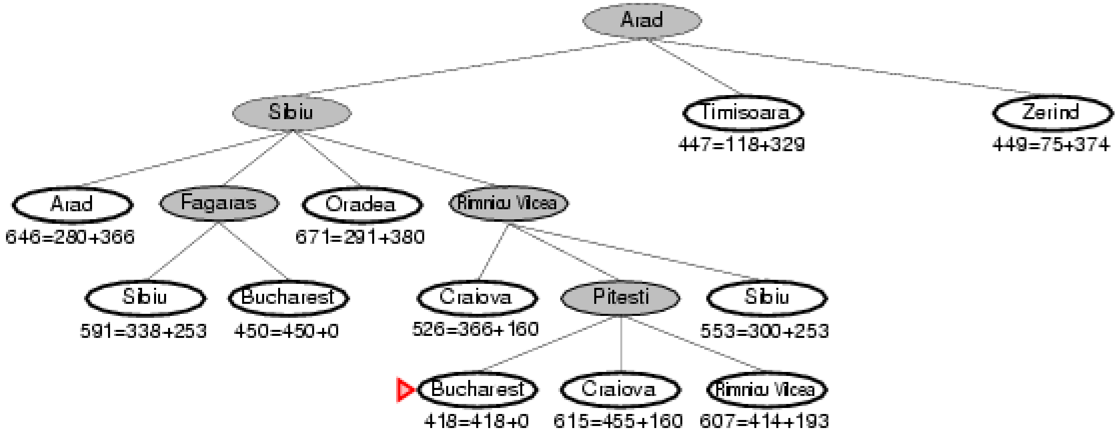
Timisoara

Zerind

Craiova

Oradea

A* search example



Now it looks like Bucharest is at the top of the open list…

Open List:

Bucharest

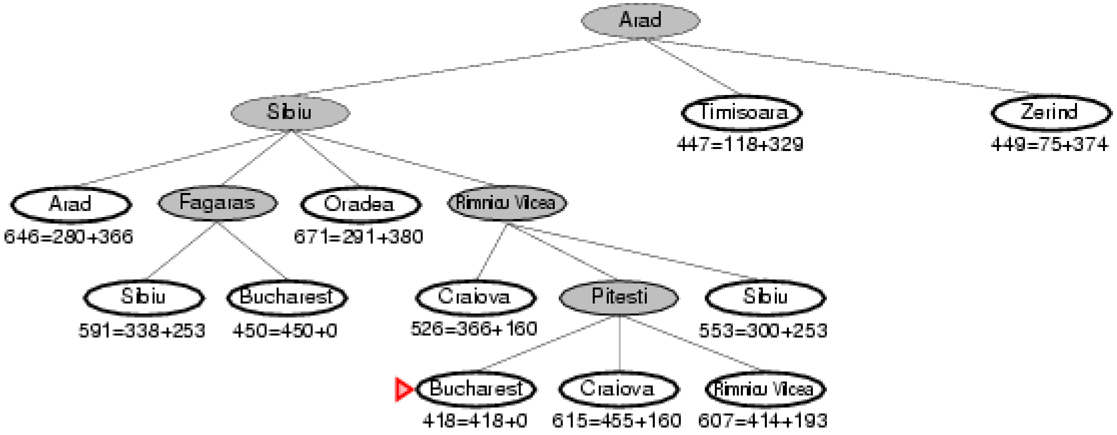
Timisoara

Zerind

Craiova

Oradea

A* search example



Now we “expand” the node for Bucharest.

We’re done!   (And we know the path that we’ve found is optimal.)

Analyzing the heuristic function

- ***f(n) = h(n)   (Greedy best first search)***
- If h(n) = *constant(or zero) g=g+1* then A* as BFS
- **If h(n) = 0 for all n then *f(n) = g(n)***

  This heuristic is admissible, A*performs exactly as UCS
- **If h(n) = h*(n) for all n**
  - Only nodes on optimal solution path are expanded
  - No unnecessary work is performed ⍰
- **The closer h is to h***, The fewer extra nodes that will be expanded.

Properties of Best First Search(A*)

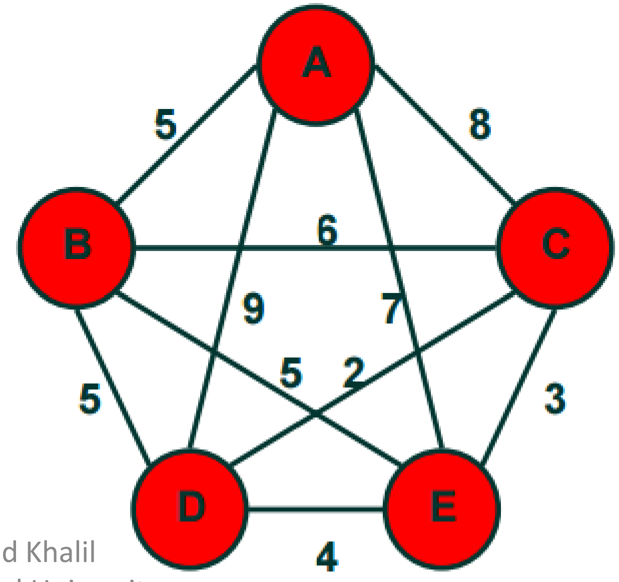
- Complete? Yes (as long as the memory supports the depth and branching factor of the tree)
- Time? $O(b^d)$
- Space? $O(b^d)$
- Optimal? Yes(If h is admissible, A* will always find a least cost path to the goal.)

Traveling Salesperson Problem (TSP)

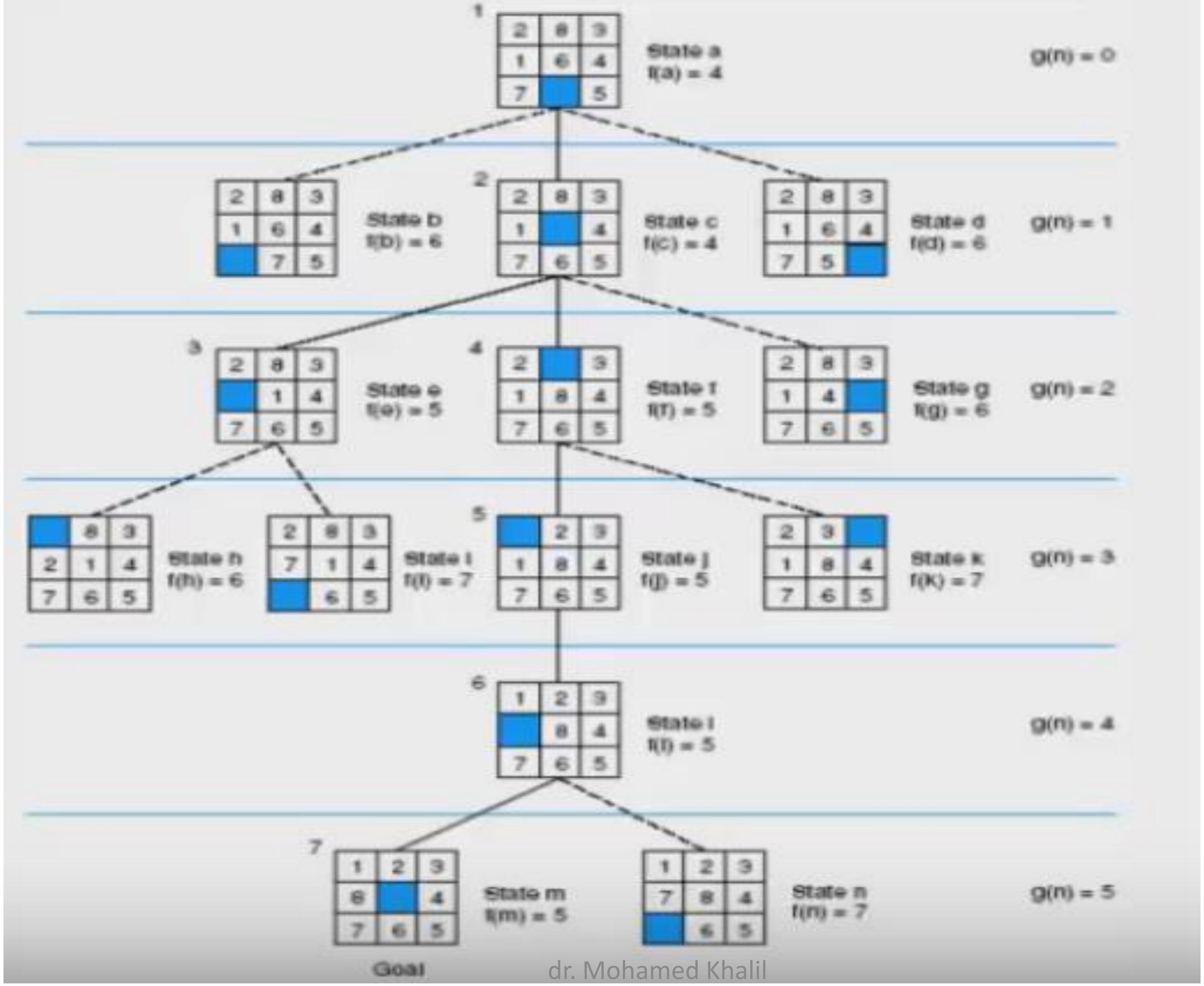
A salesman wants to visit a list of cities

- Stopping in each city only once
- Returning to the first city
- Traveling the shortest distance

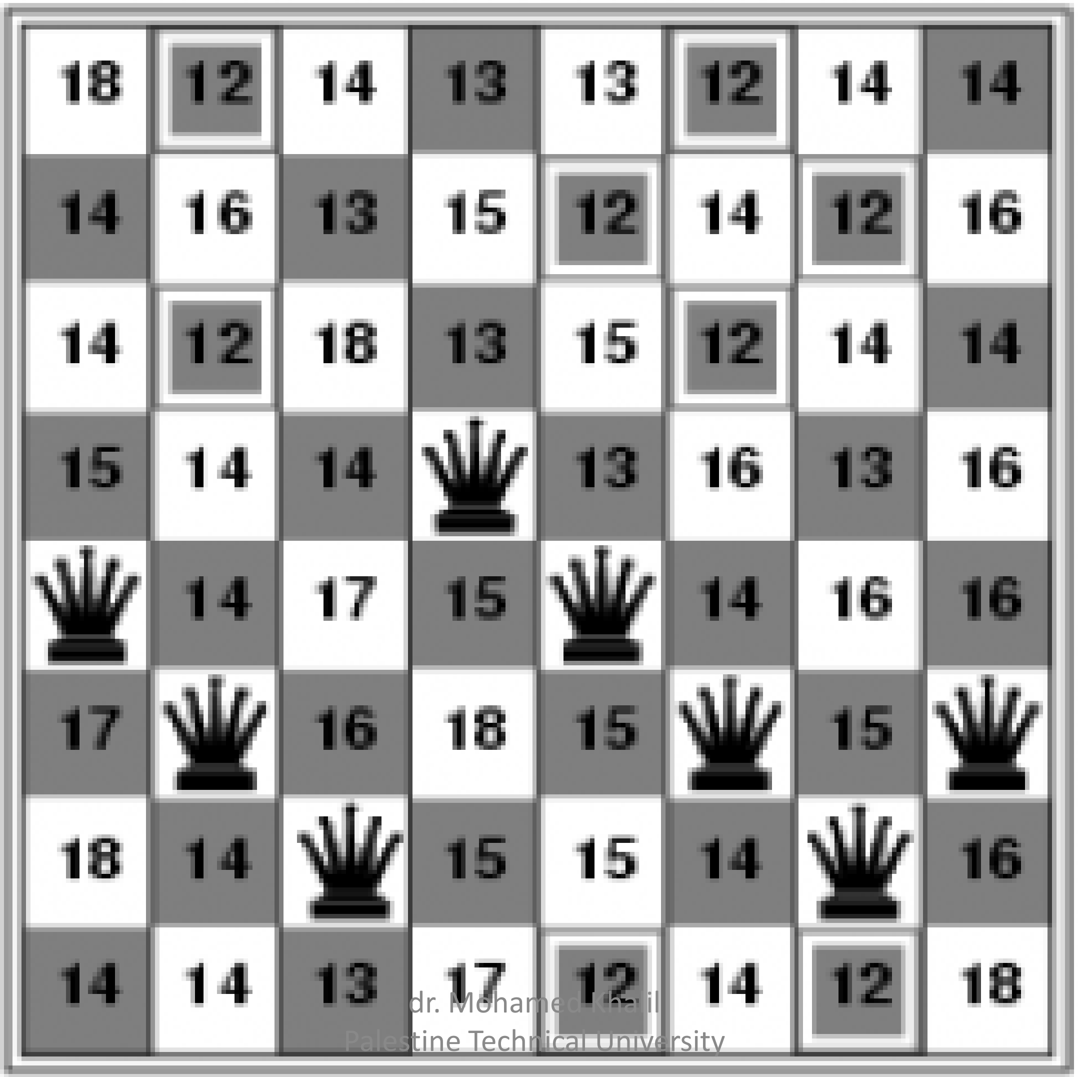
	A	B	C	D	E
A	0	5	8	9	7
B	5	0	6	5	5
C	8	6	0	2	3
D	9	5	2	0	4
E	7	5	3	4	0



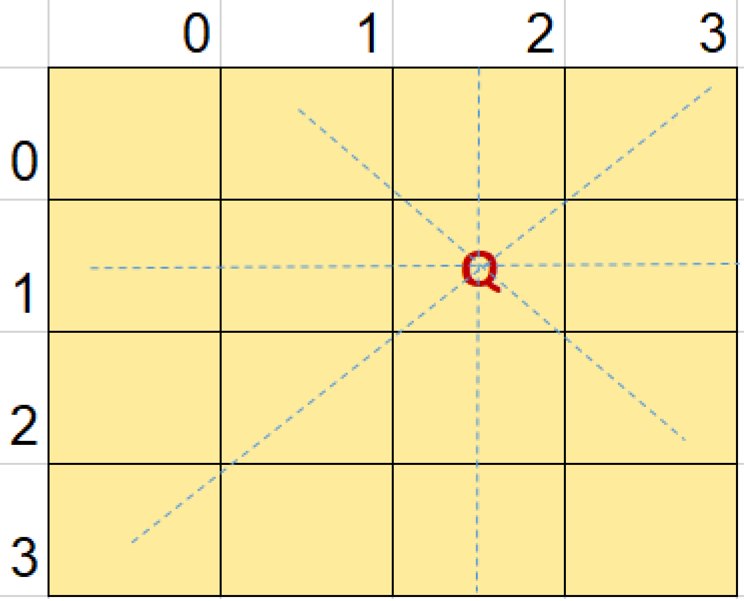
Eight-Puzzle represenation



Hill-climbing search: 8-queens problem

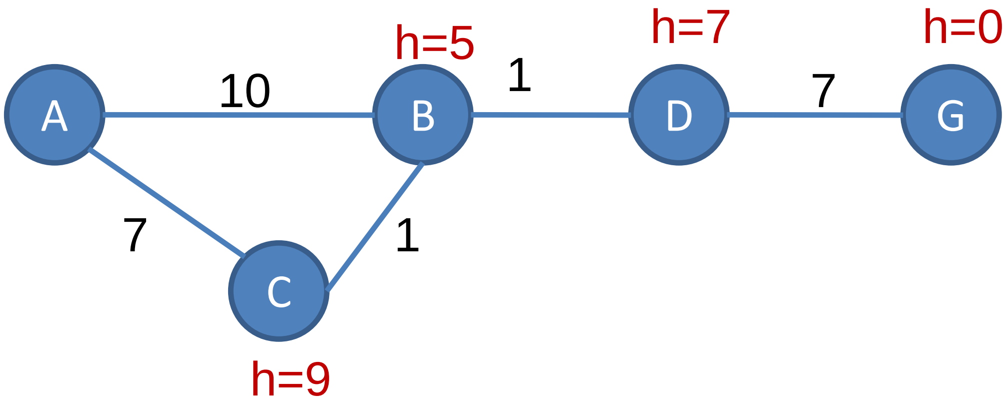


18	12	14	13	13	12	14	14
14	16	13	15	12	14	12	16
14	12	18	13	15	12	14	14
15	14	14	♛	13	16	13	16
♛	14	17	15	♛	14	16	16
17	♛	16	18	15	♛	15	♛
18	14	♛	15	15	14	♛	16
14	14	13	17	12	14	12	18

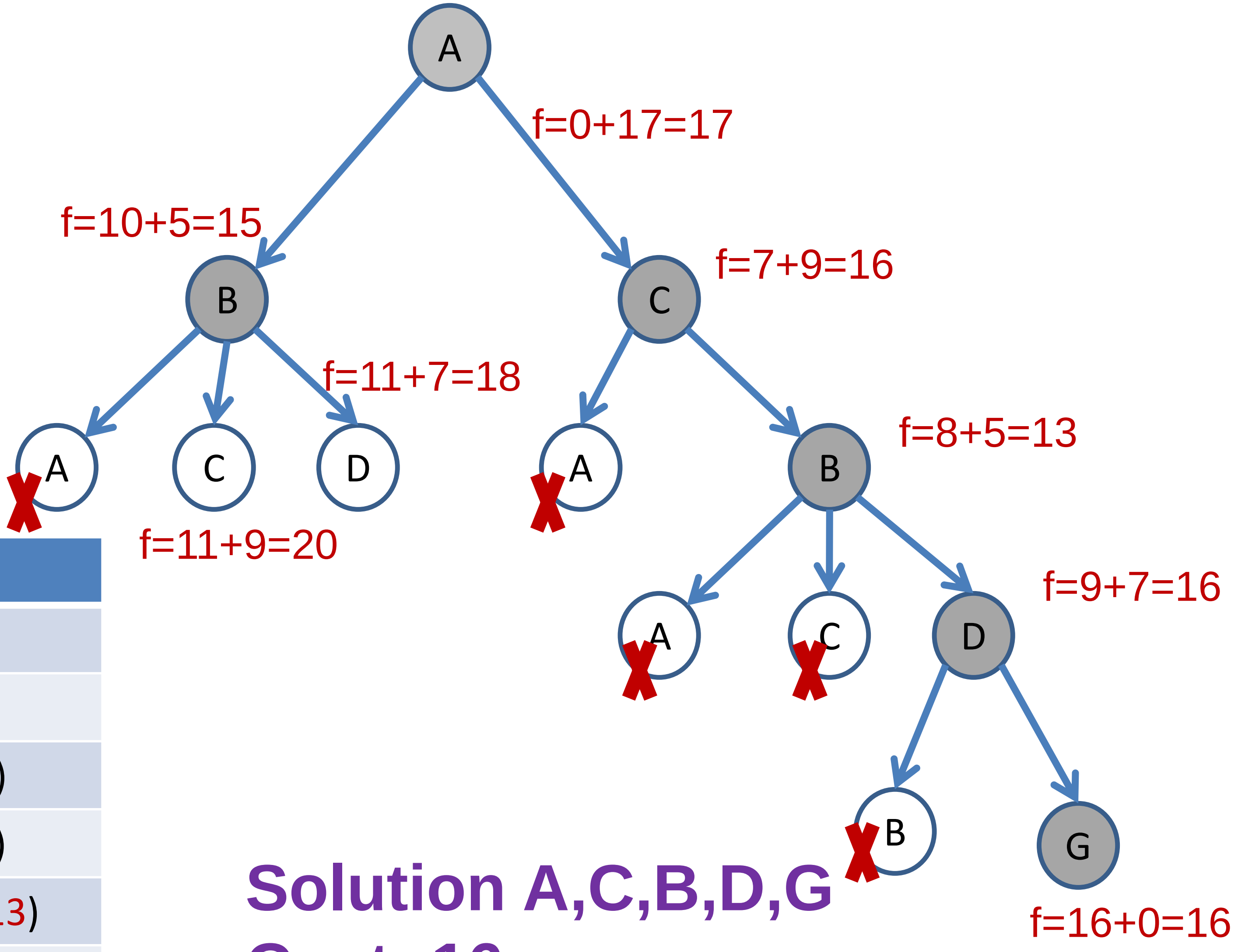


row	col	diag1	diag2
(1,0)	(0,2)	(0,1)	(3,0)
(1,1)	(2,2)	(2,3)	(2,1)
(1,3)	(3,2)		(0,3)

Using A* algorithm, Find a solution (path) and the cost from A to G



Using A* algorithm, Find a solution (path) and the cost from A to G

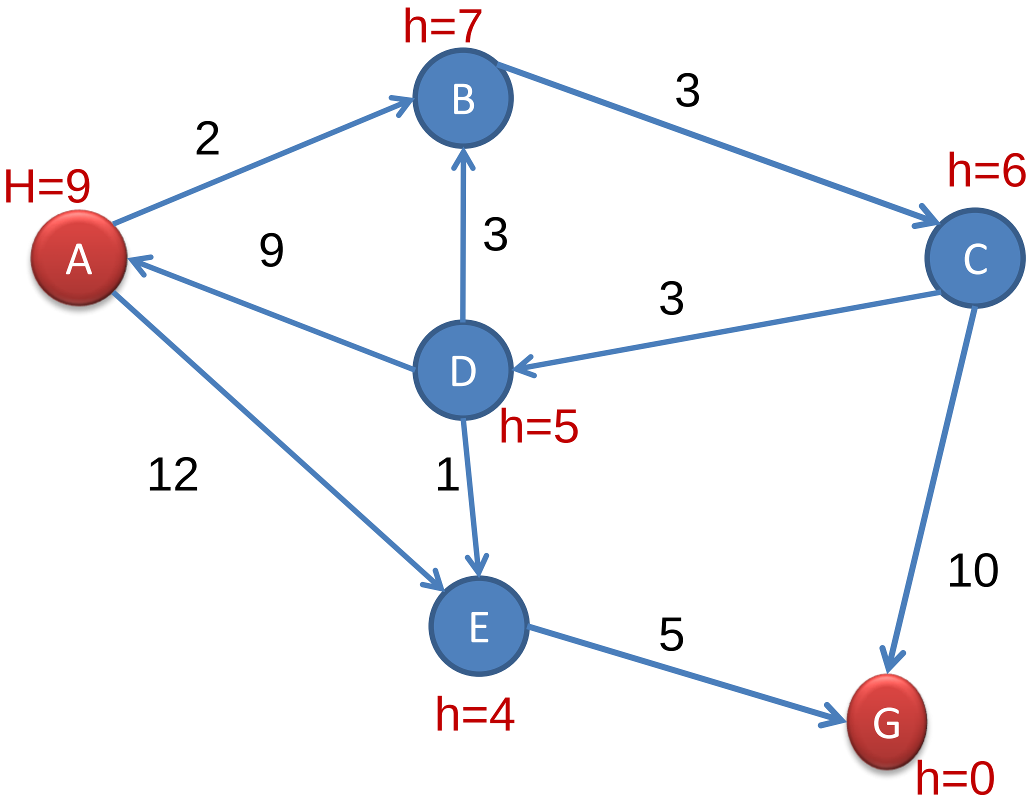


open	close
A	-
B(15),C(16)	A
C(16),D(18)	A,B(15)
B(13),D(18)	A,C(16)
D(16)	A,C,B(13)
G	A,C,B,D(16)

Solution A,C,B,D,G

Cost: 16

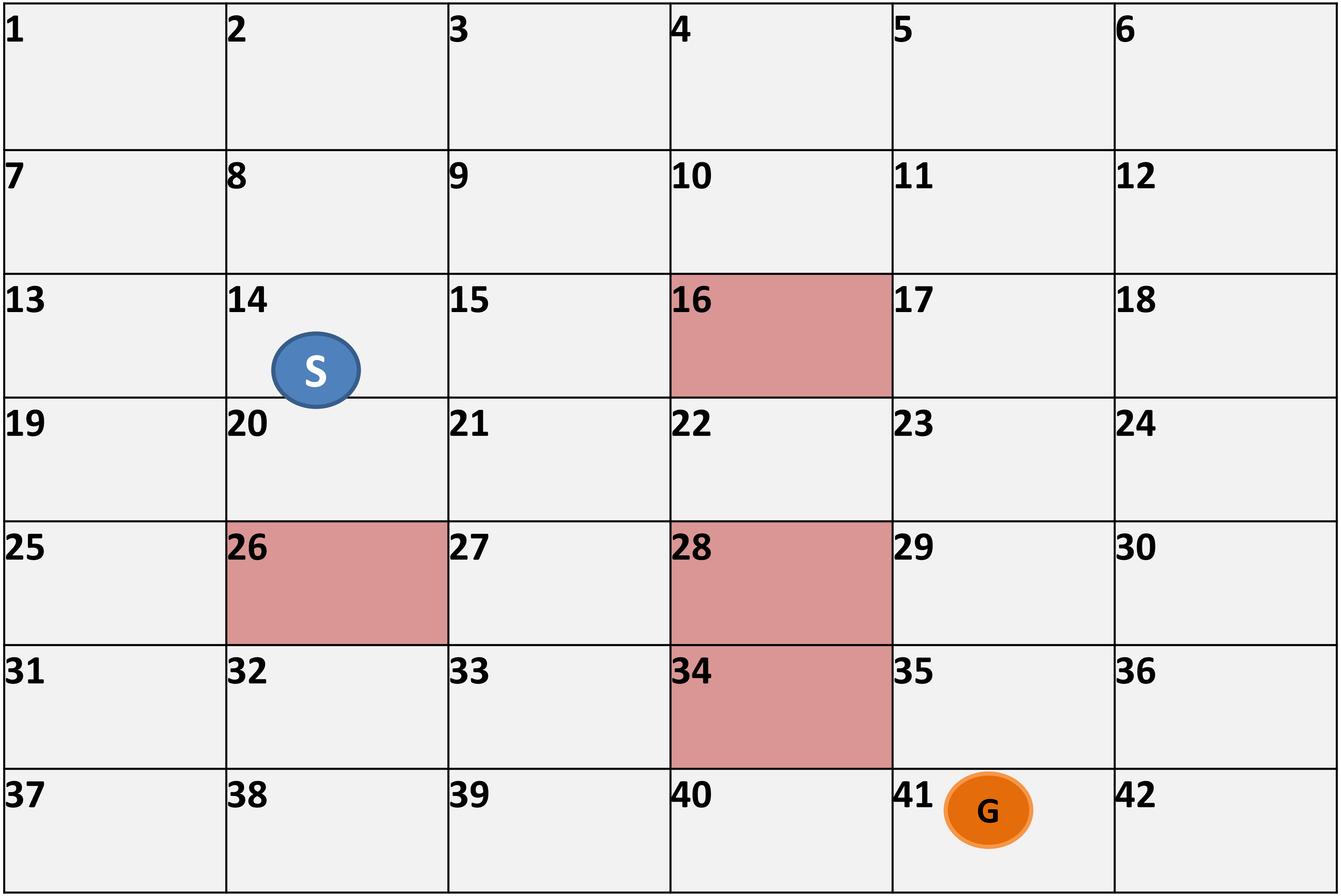
Using A* algorithm, Find a solution (path) and the cost from A to G



Maze Traversal

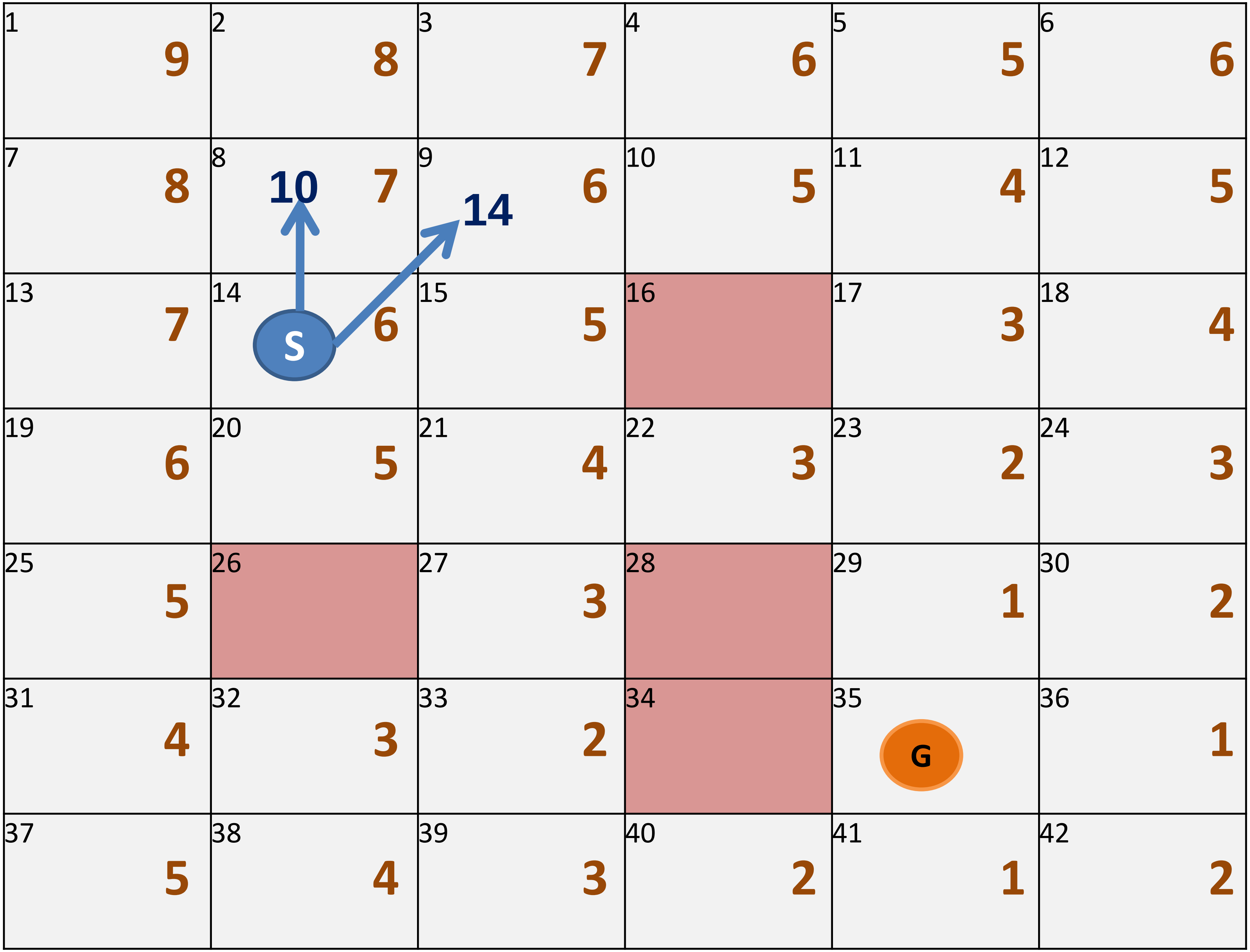
Node Data:

1. **H-value(heuristic)**
2. **G-value(movement cost)**
3. **F-value**(H+G)
4. **Parent**(A node to reach this node)
5. **Open set**-List of node that need to be checked
6. **Closed set-**List of node have been checked

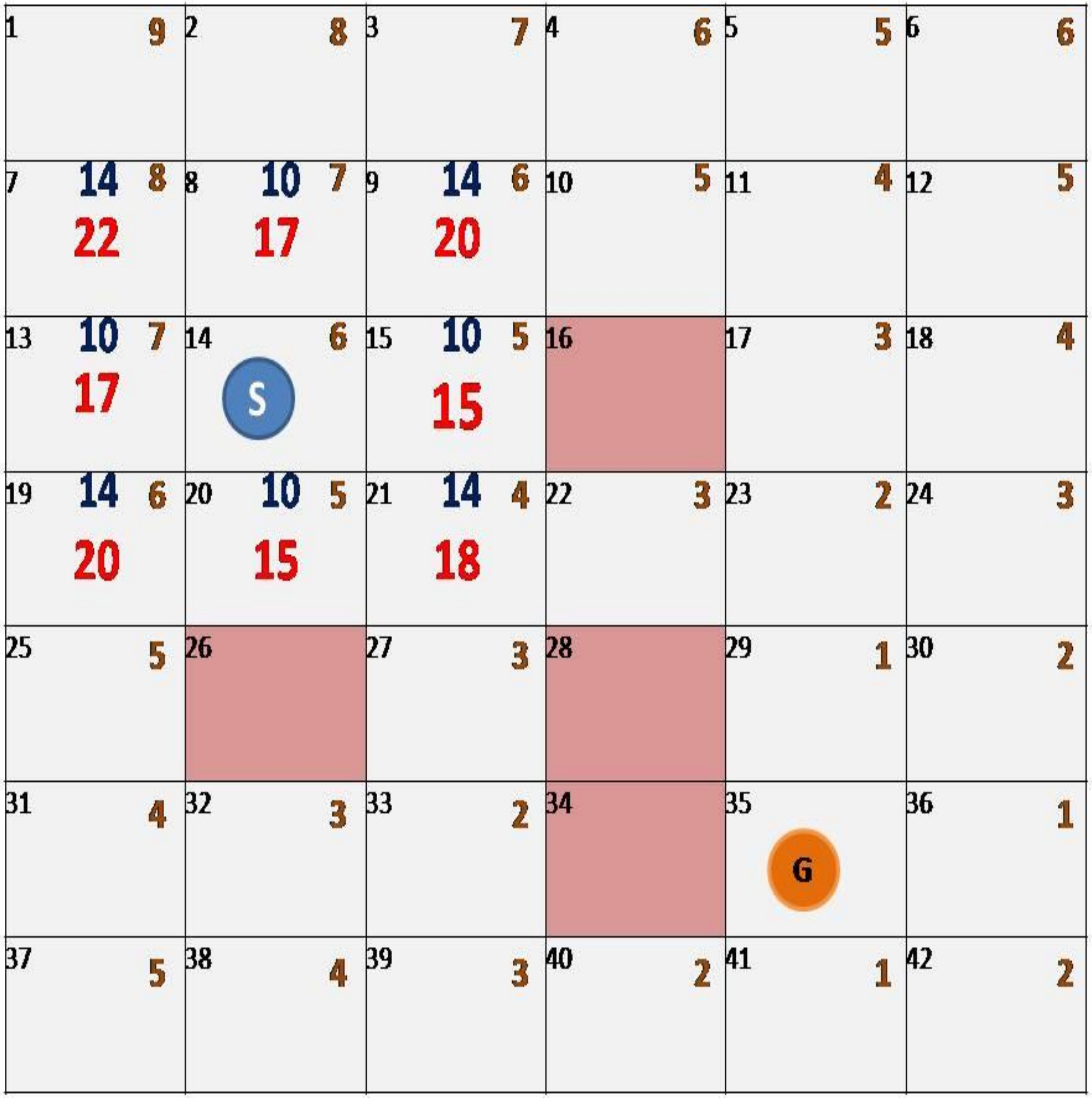


1	2	3	4	5	6
7	8	9	10	11	12
13	14 S	15	16 🟥	17	18
19	20	21	22	23	24
25	26 🟥	27	28 🟥	29	30
31	32	33	34 🟥	35	36
37	38	39	40	41 G	42

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



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

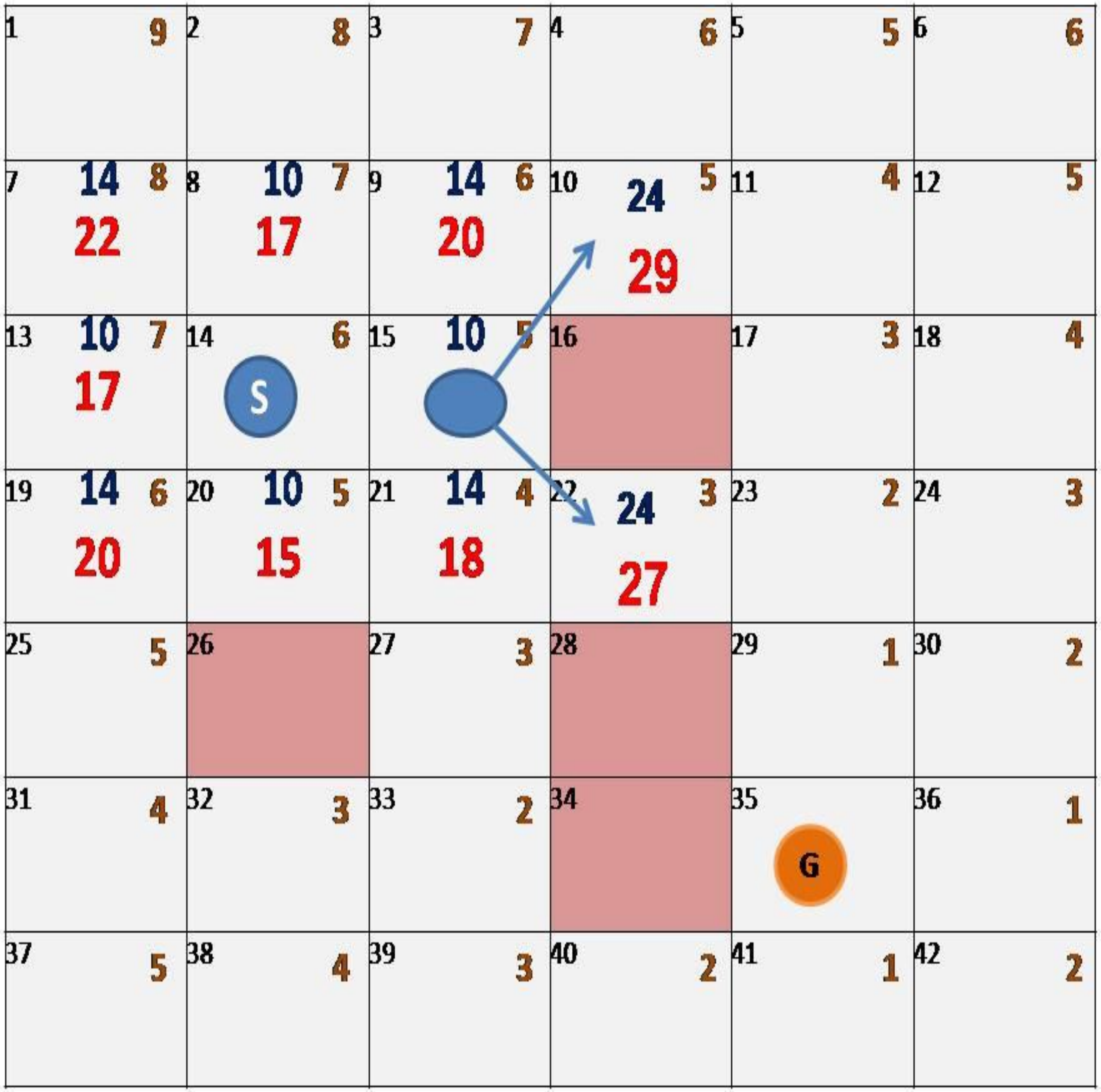


OPEN LIST

7,8,9,13,15,19,20,21

CLOSE LIST

14



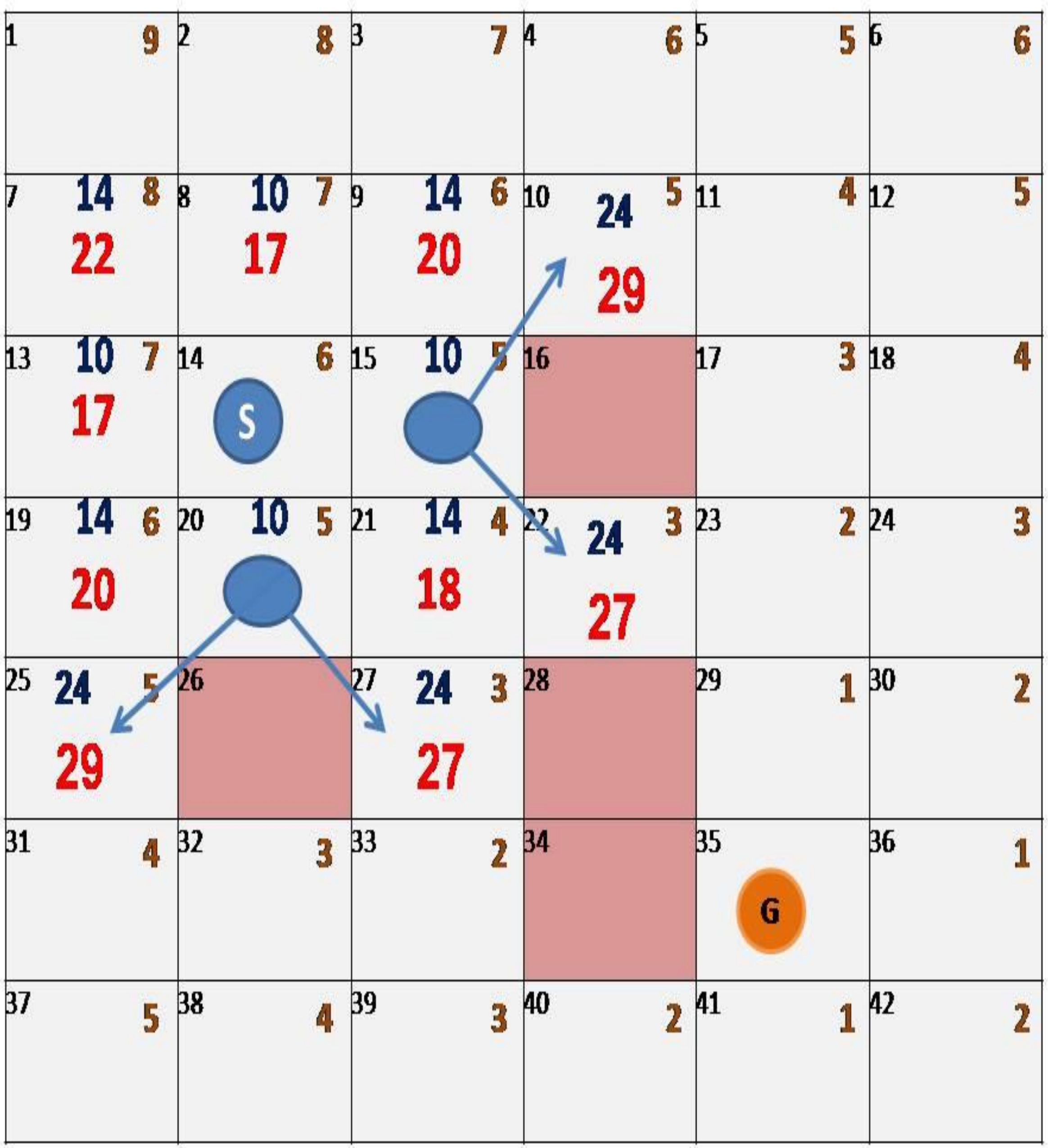
1 9	2 8	3 7	4 6	5 5	6 6
7 14 8 22	8 10 7 17	9 14 6 20	10 24 5 29	11 4	12 5
13 10 7 17	14 6 S	15 10 5	16 ▨	17 3	18 4
19 14 6 20	20 10 5 15	21 14 4 18	22 24 3 27	23 2	24 3
25 5	26 ▨	27 3	28 ▨	29 1	30 2
31 4	32 3	33 2	34 ▨	35 G	36 1
37 5	38 4	39 3	40 2	41 1	42 2

OPEN LIST

7,8,9,13,19,20,21,10,
22

CLOSE LIST

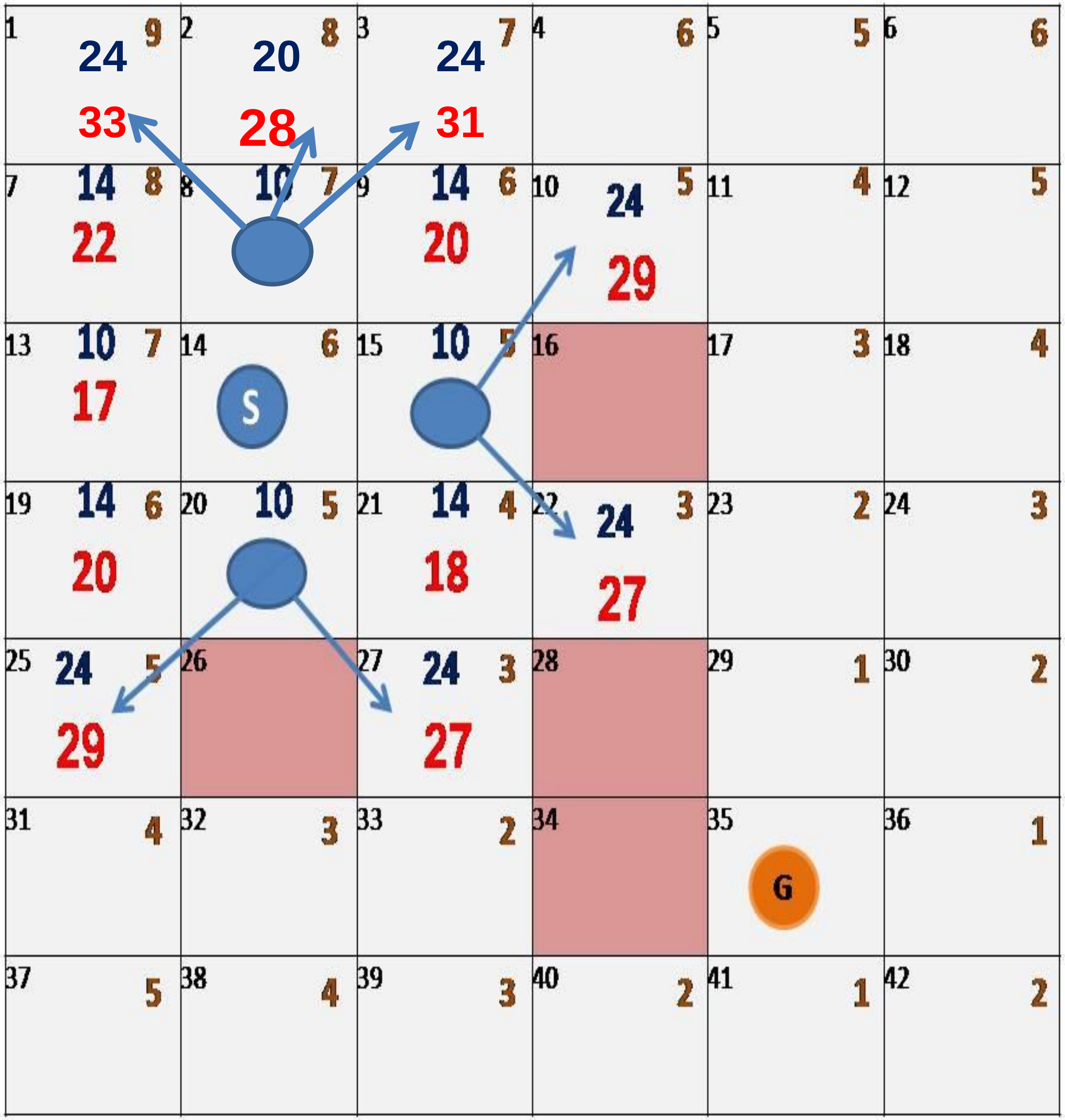
14,15



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

OPEN LIST
7,8,9,13,19,21,10,22,
25,27

CLOSE LIST
14,15,**20**



OPEN LIST

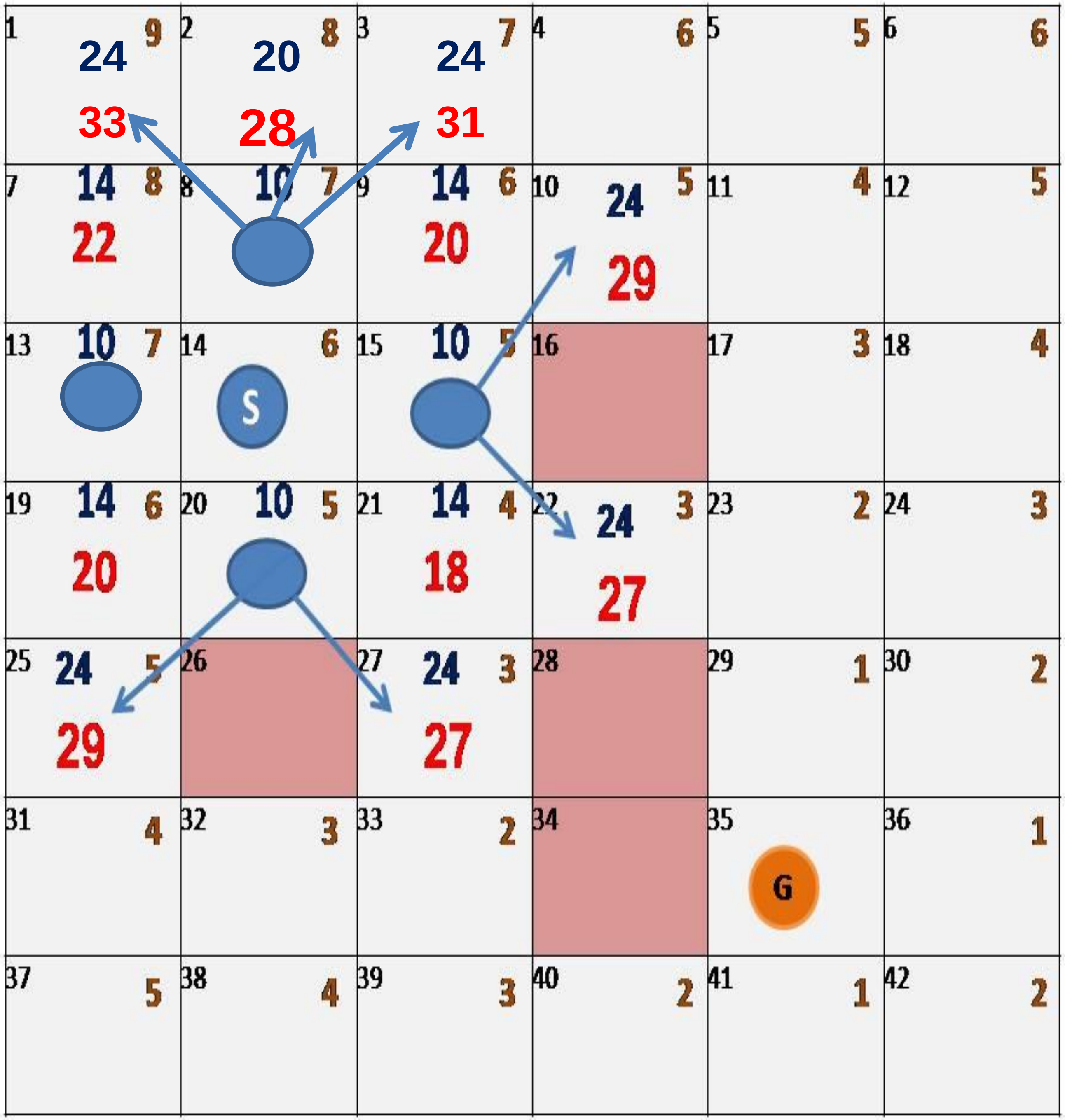
7,9,13,19,21,10,22,

25,27,

1,2,3

CLOSED LIST

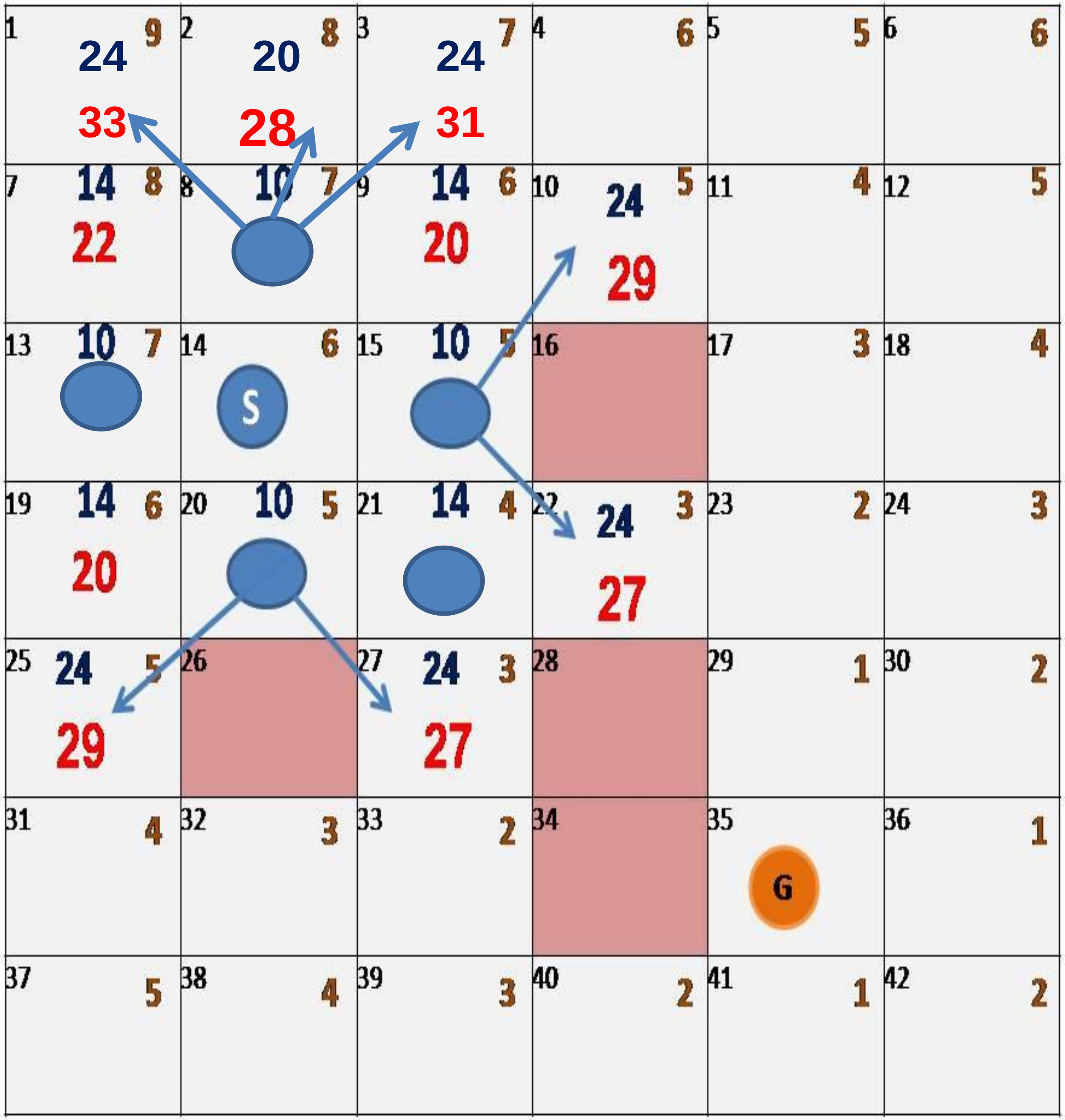
14,15,**20**,**8**



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

OPEN LIST
7,9,19,21,10,22,**25,27**
,1,2,3

CLOSED LIST
14,15,**20**,**8**,**13**



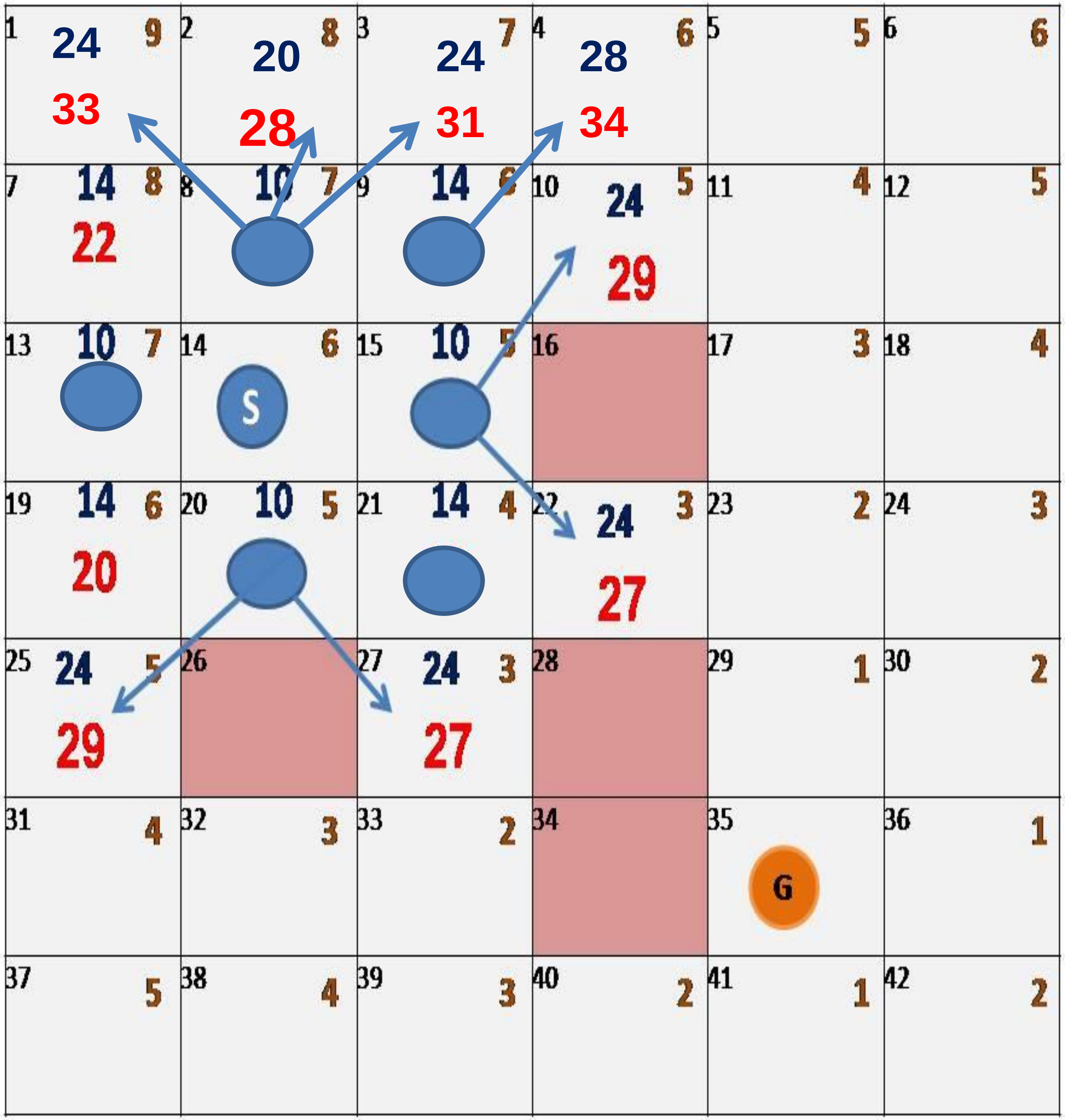
1 9 24 33	2 8 20 28	3 7 24 31	4 6	5 5	6 6
7 14 8 22	8 10 7	9 14 6 20	10 24 5 29	11 4	12 5
13 10 7	14 6 S	15 10 5	16	17 3	18 4
19 14 6 20	20 10 5	21 14 4	22 24 3 27	23 2	24 3
25 24 5 29	26	27 24 3 27	28	29 1	30 2
31 4	32 3	33 2	34	35 G	36 1
37 5	38 4	39 3	40 2	41 1	42 2

OPEN LIST

7,9,19,10,22,**25,27,1,2,3**

CLOSED LIST

14,15,**20,8,13,21**



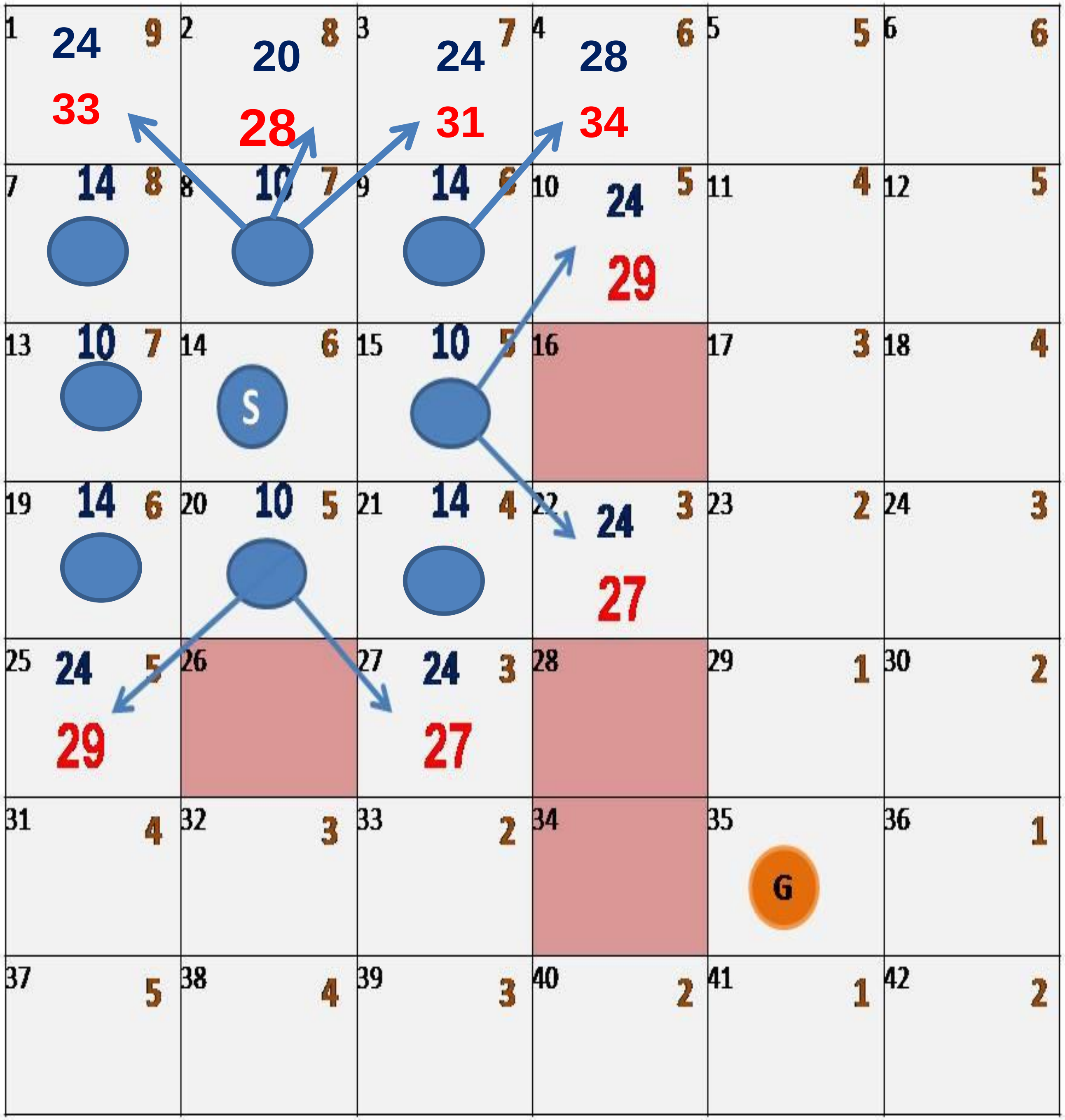
OPEN LIST
7,19,10,22,**25,27,1,2,**
3,4

CLOSED LIST
14,15,**20,8,13,21,9**

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

OPEN LIST
7,10,22,**25,27,1,2,3,4**

CLOSED LIST
14,15,**20,8,13,21,9,**
19

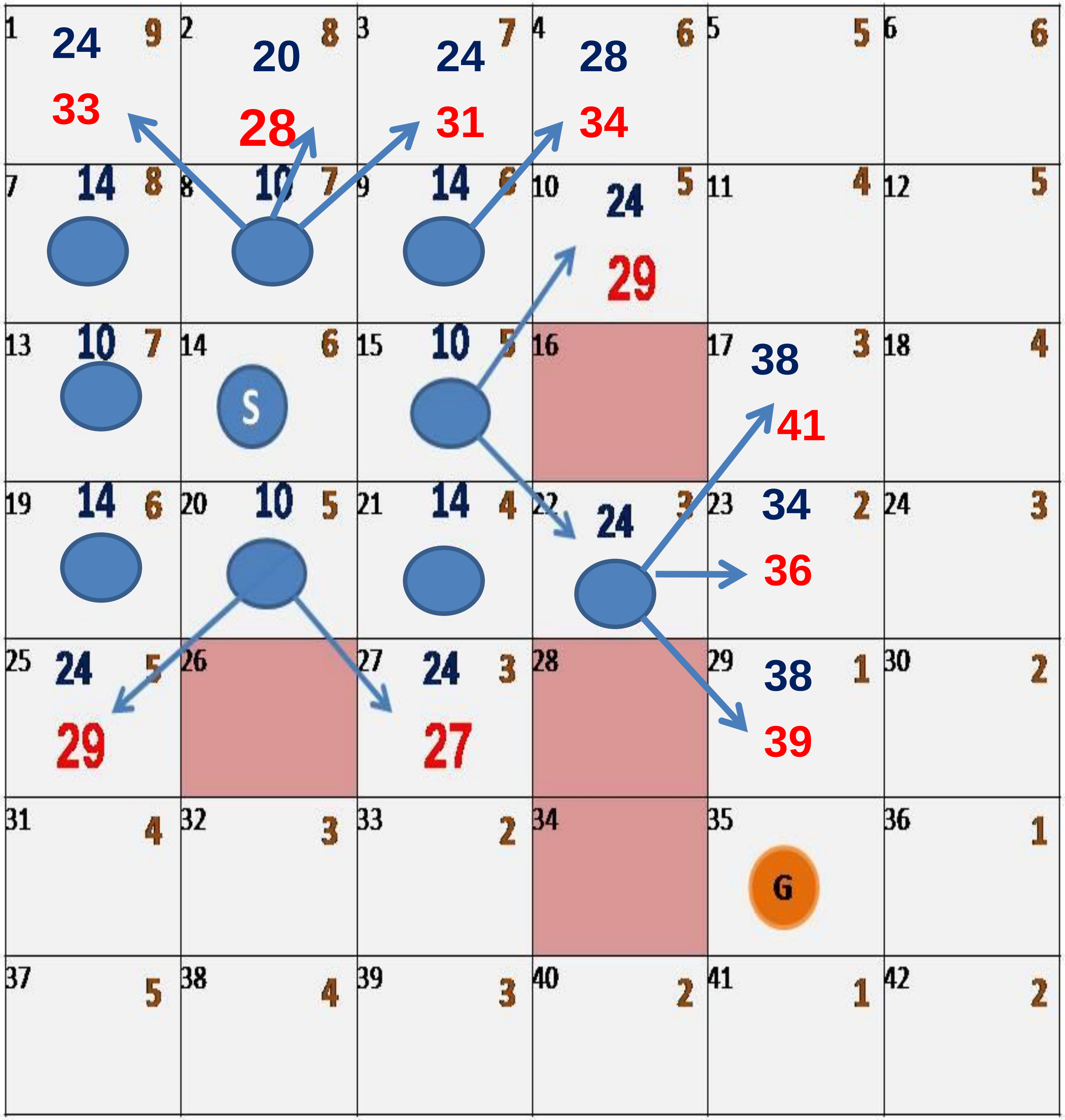


OPEN LIST

10,22,**25,27,1,2,3,4**

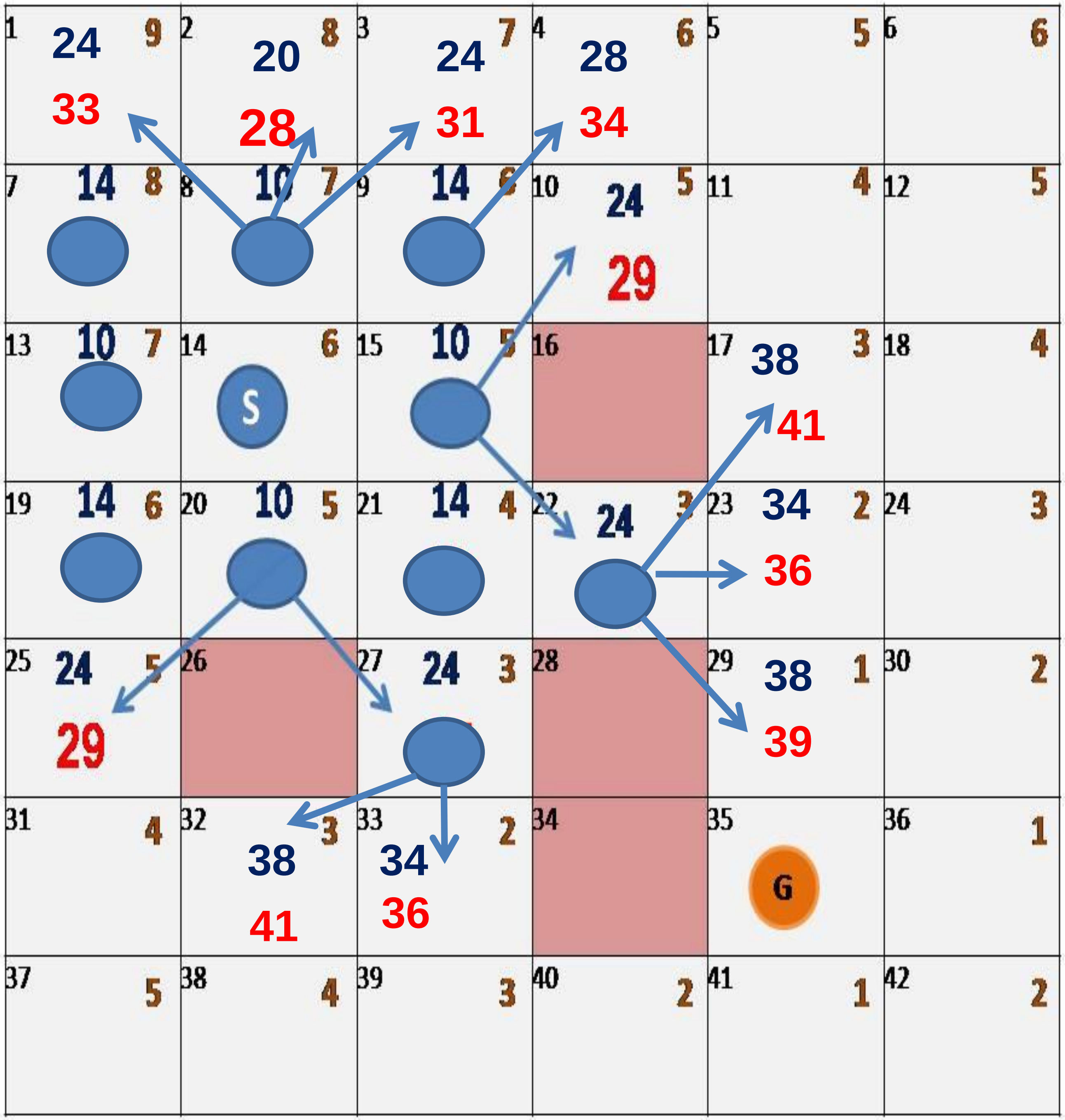
CLOSED LIST

14,15,**20,8,13,21,9,**
19,7



OPEN LIST
10,25,27,1,2,3,4,
17,23,29

CLOSED LIST
14,15,20,8,13,21,9,
19,7,22

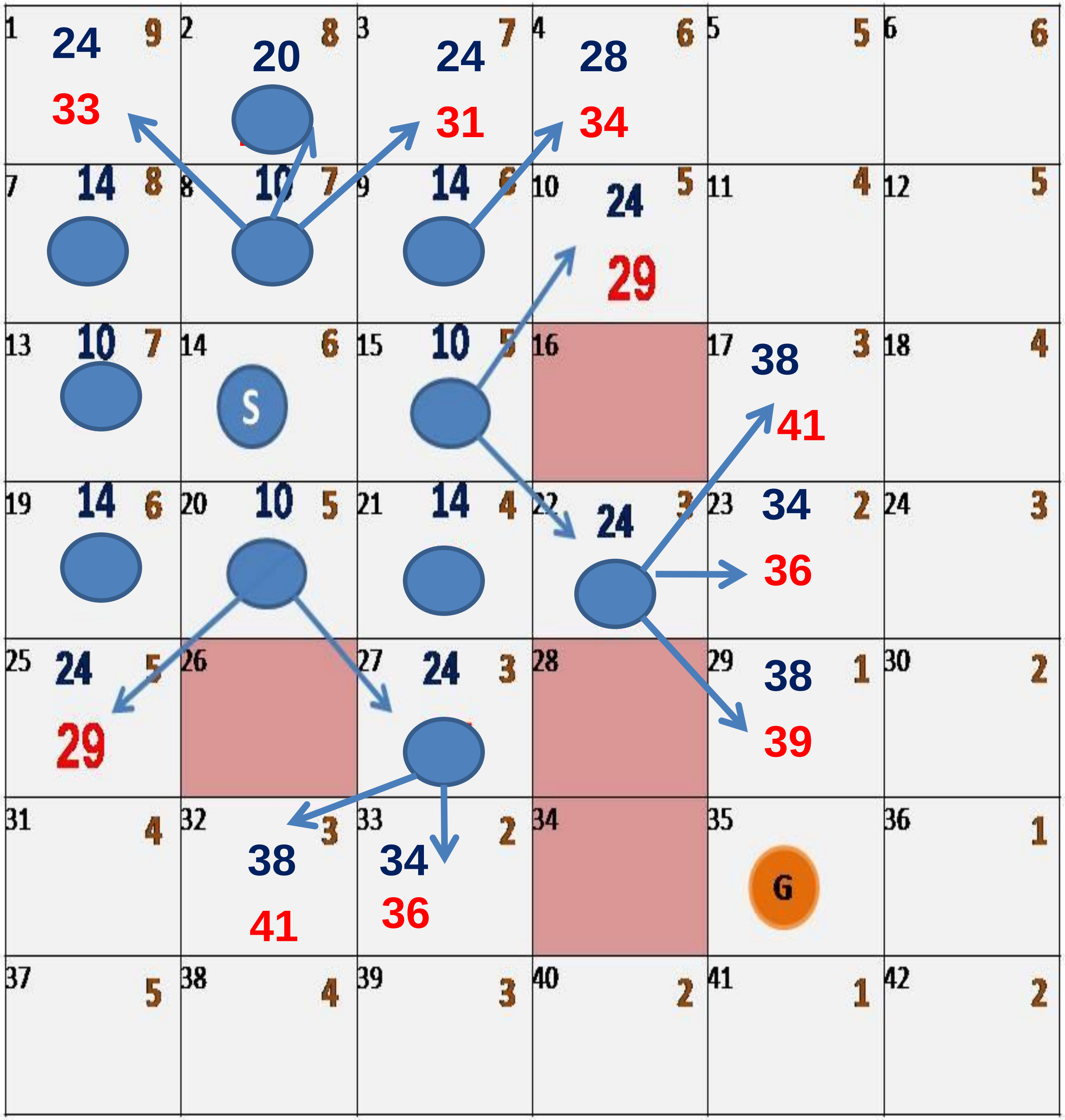


OPEN LIST

10,**25,1,2,3,4,17,**
23,29,
32,33

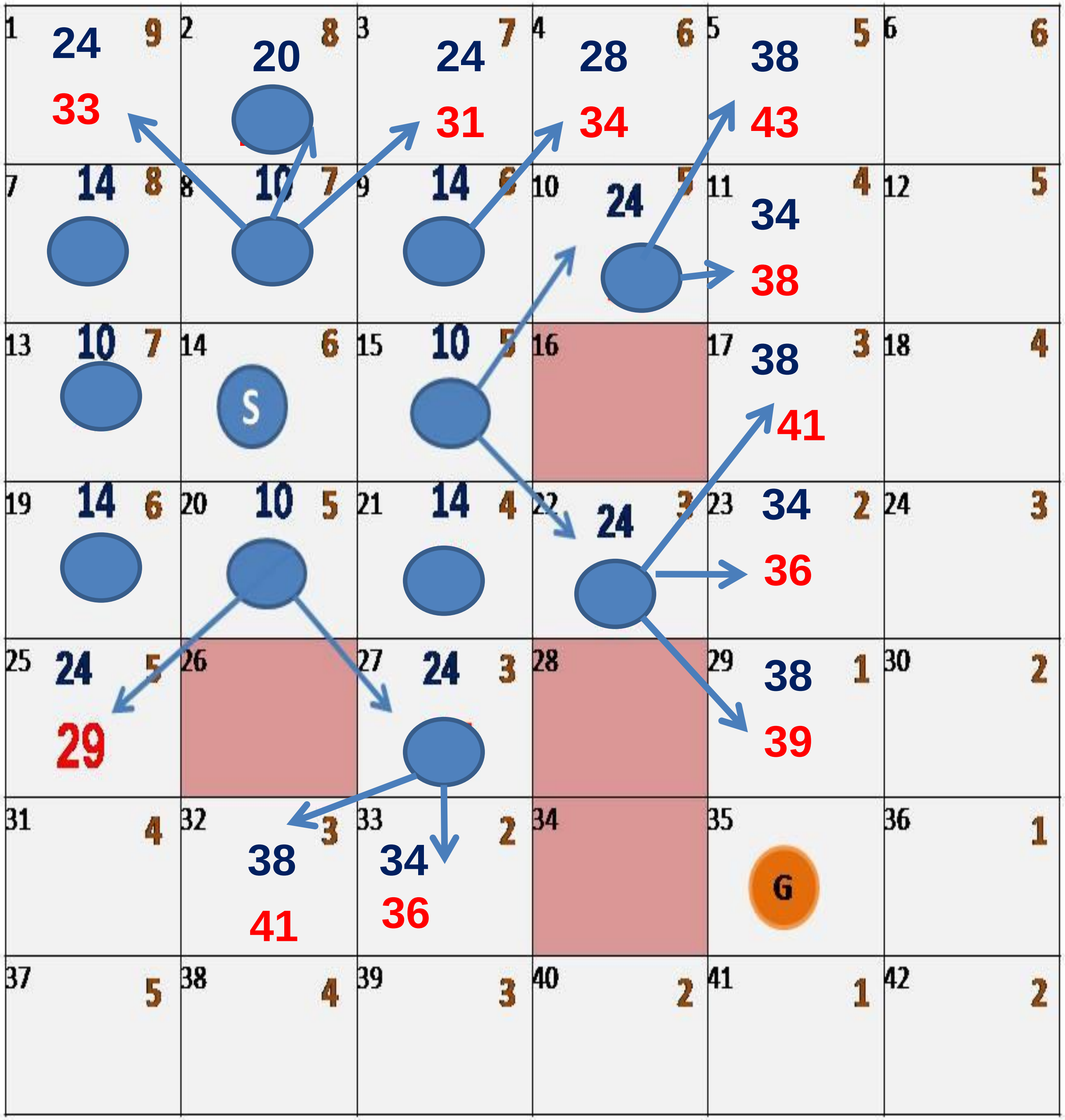
CLOSED LIST

14,15,**20,8,13,21,9,**
19,7,22,27



OPEN LIST
10,25,1,3,4,17,
23,29,
32,33

CLOSED LIST
14,15,20,8,13,21,9,
19,7,22,27,2

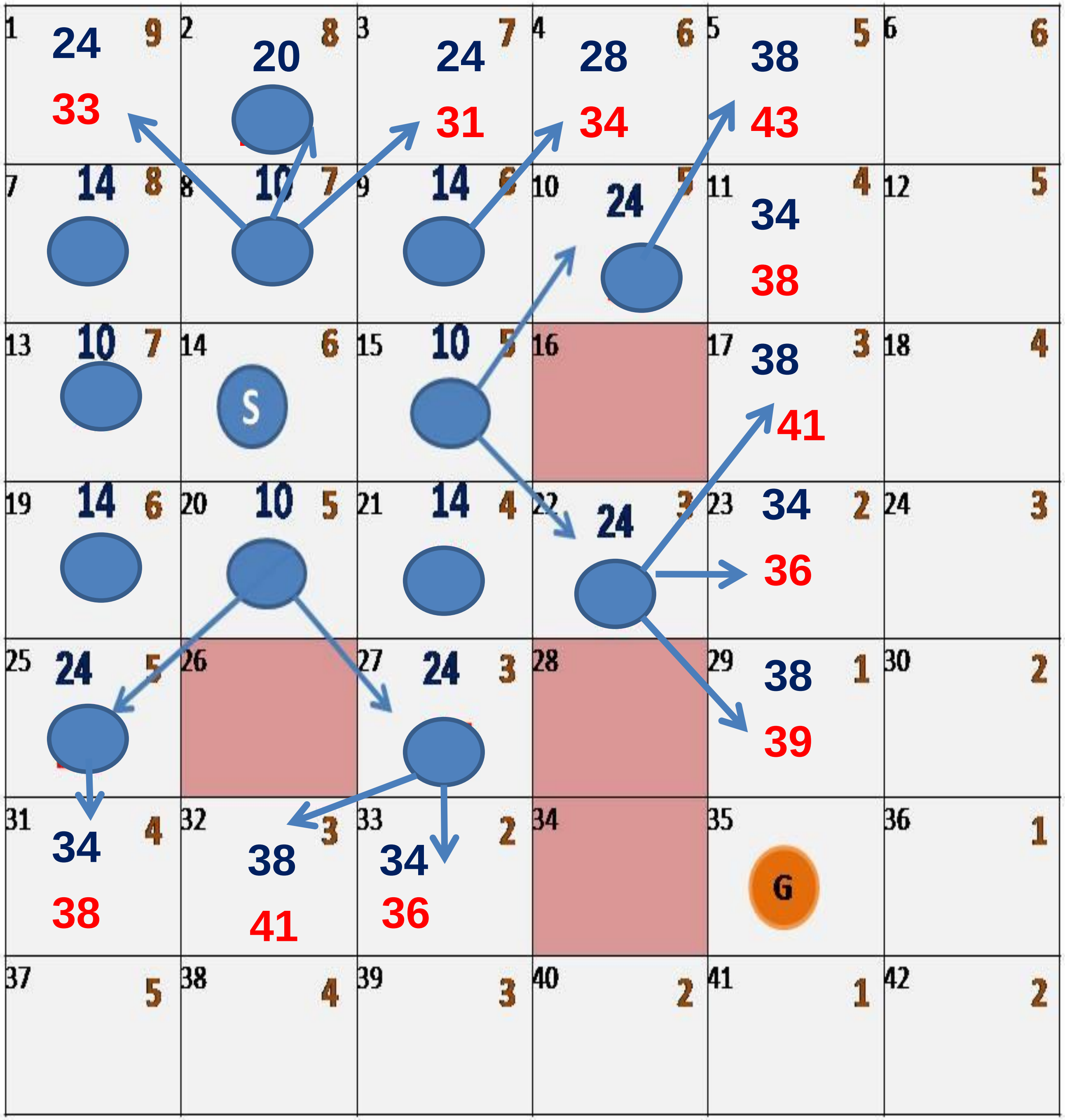


OPEN LIST

25,1,3,4,17,
23,29,
32,33,5,11

CLOSED LIST

14,15,20,8,13,21,9,
19,7,22,27,2,10

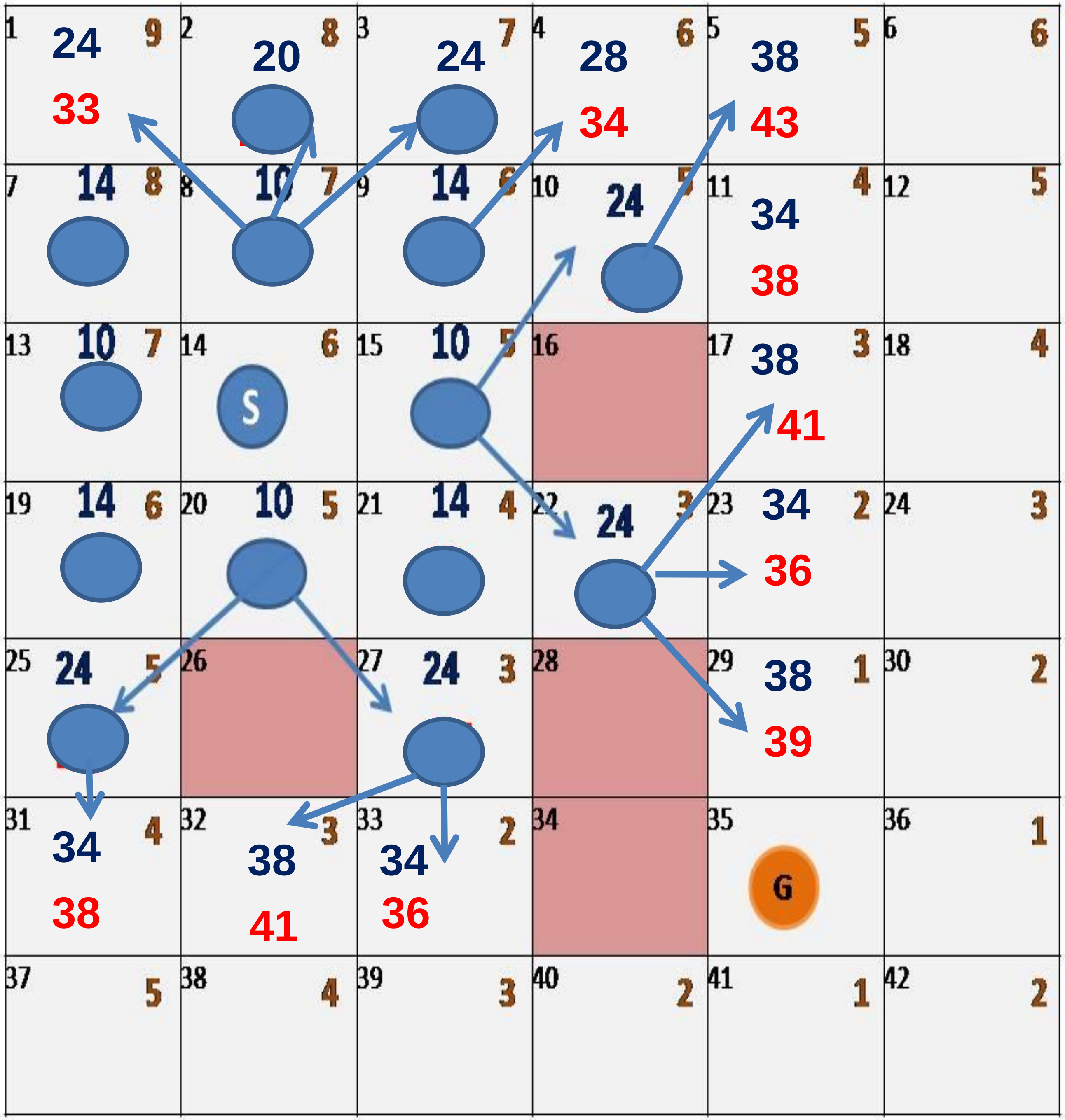


OPEN LIST

1,3,4,17,
23,29,
32,33, 5,11,31

CLOSED LIST

14,15,**20,8,13,21,9,**
19,7,22,27,2,10,25

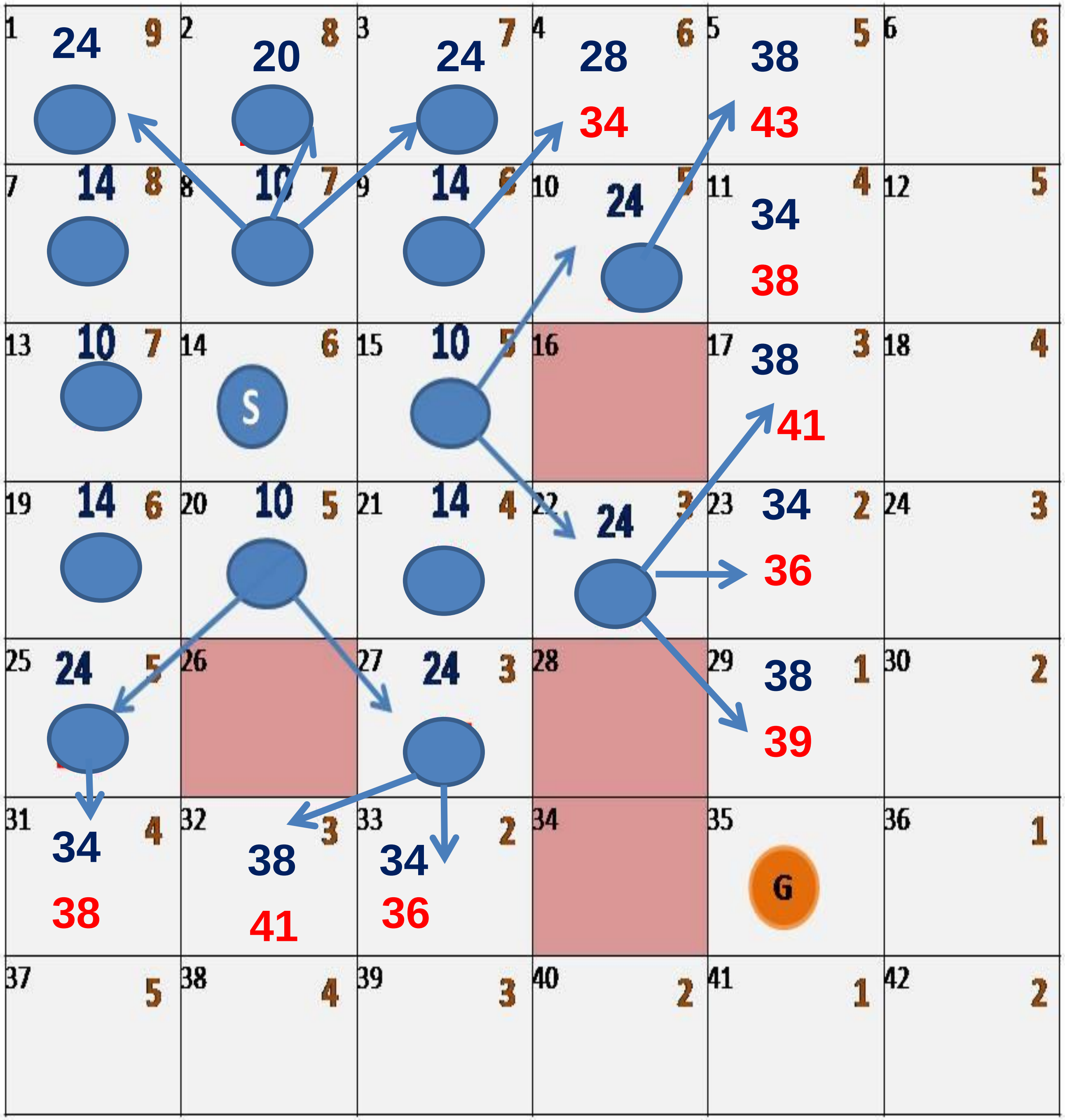


OPEN LIST

1, 4,17,
23,29,
32,33, 5,11,31

CLOSED LIST

14,15,20,8,13,21,9,
19,7,22,27,2,10,25
,3

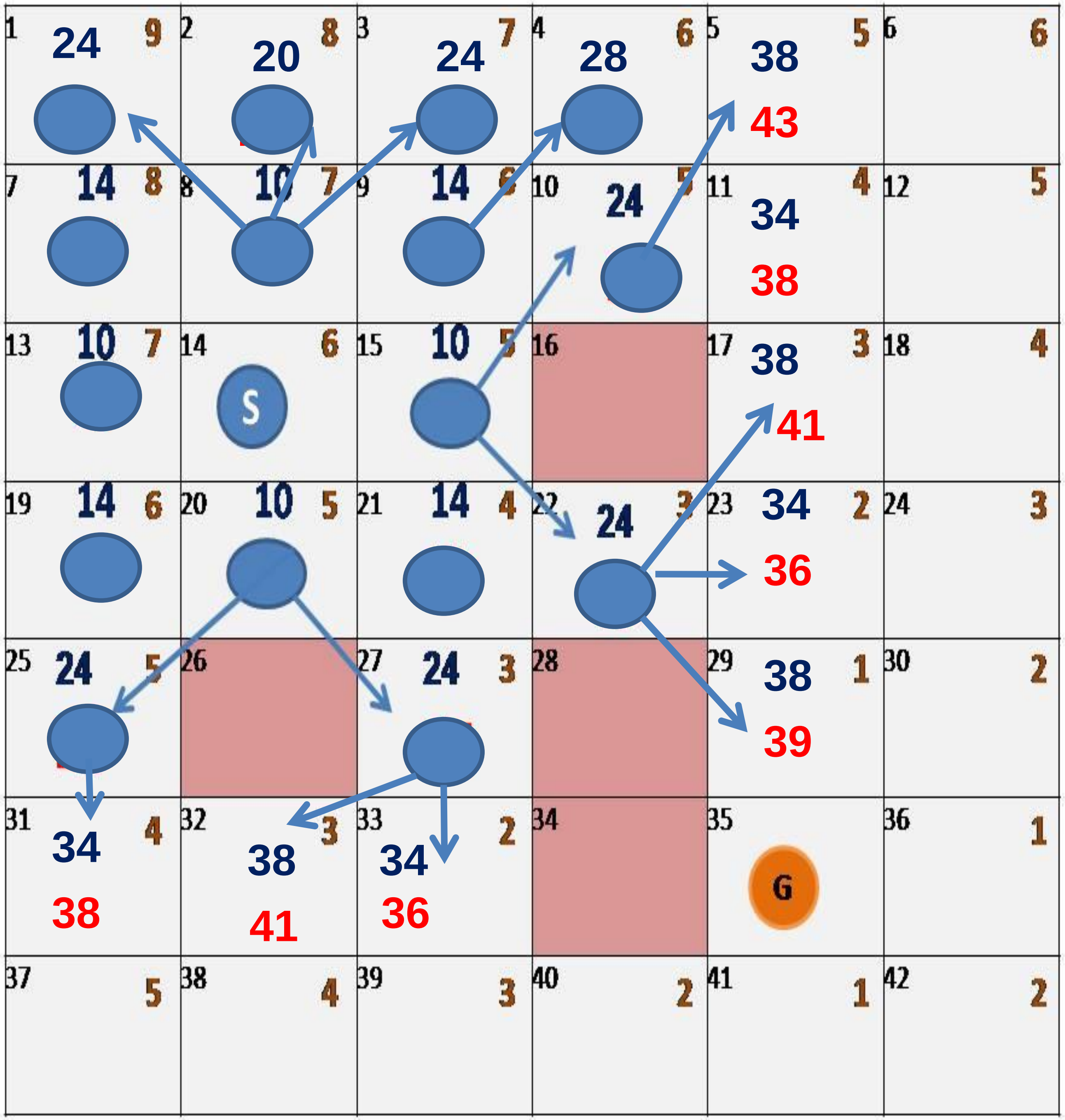


OPEN LIST

4,17,
23,29,
32,33, 5,11,31

CLOSED LIST

14,15,20,8,13,21,9,
19,7,22,27,2,10,25
,3,1

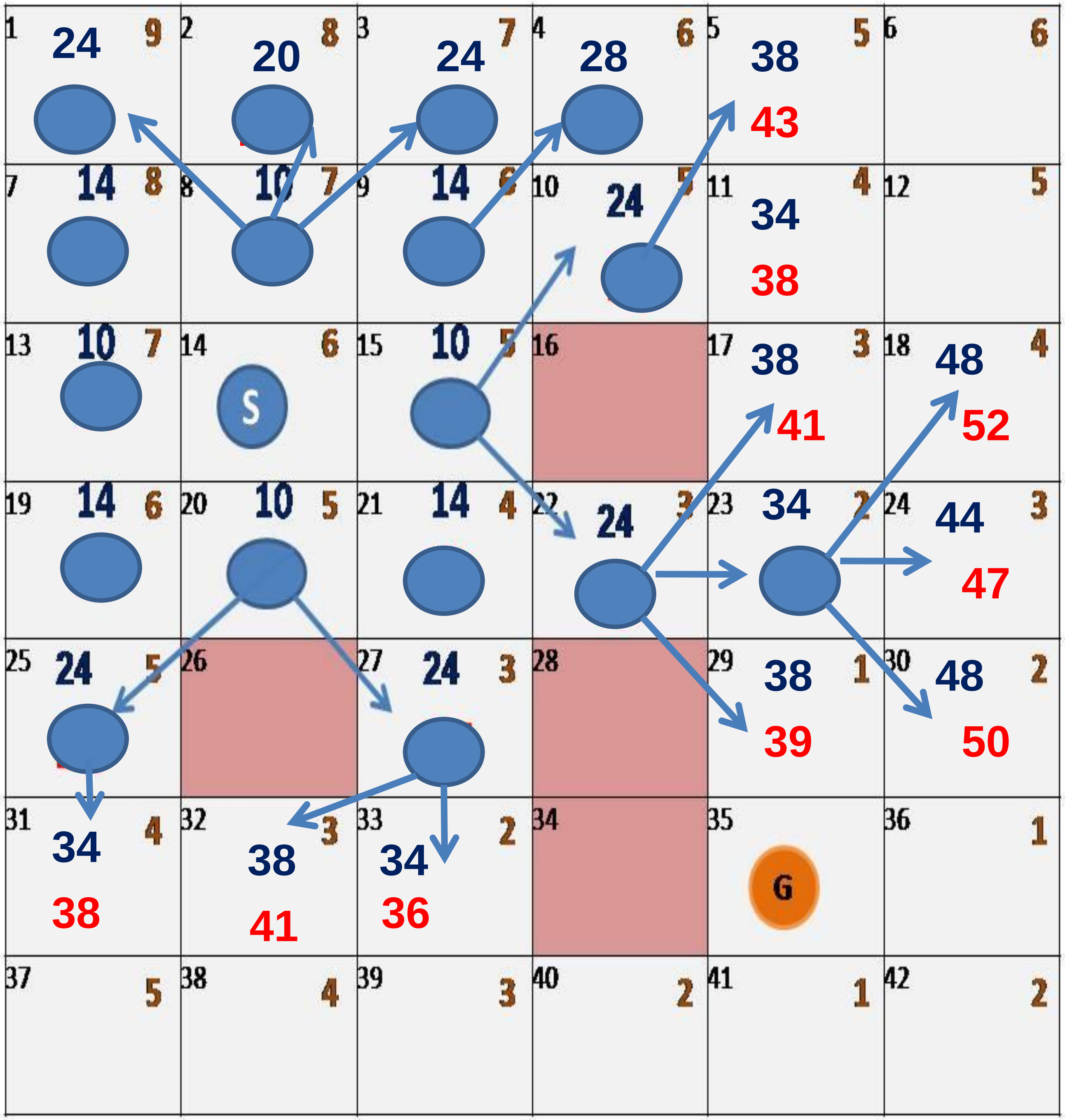


OPEN LIST

17,
23,29,
32,33, 5,11,31

CLOSED LIST

14,15,20,8,13,21,9,
19,7,22,27,2,10,25
,3,1, 4,



OPEN LIST

17,

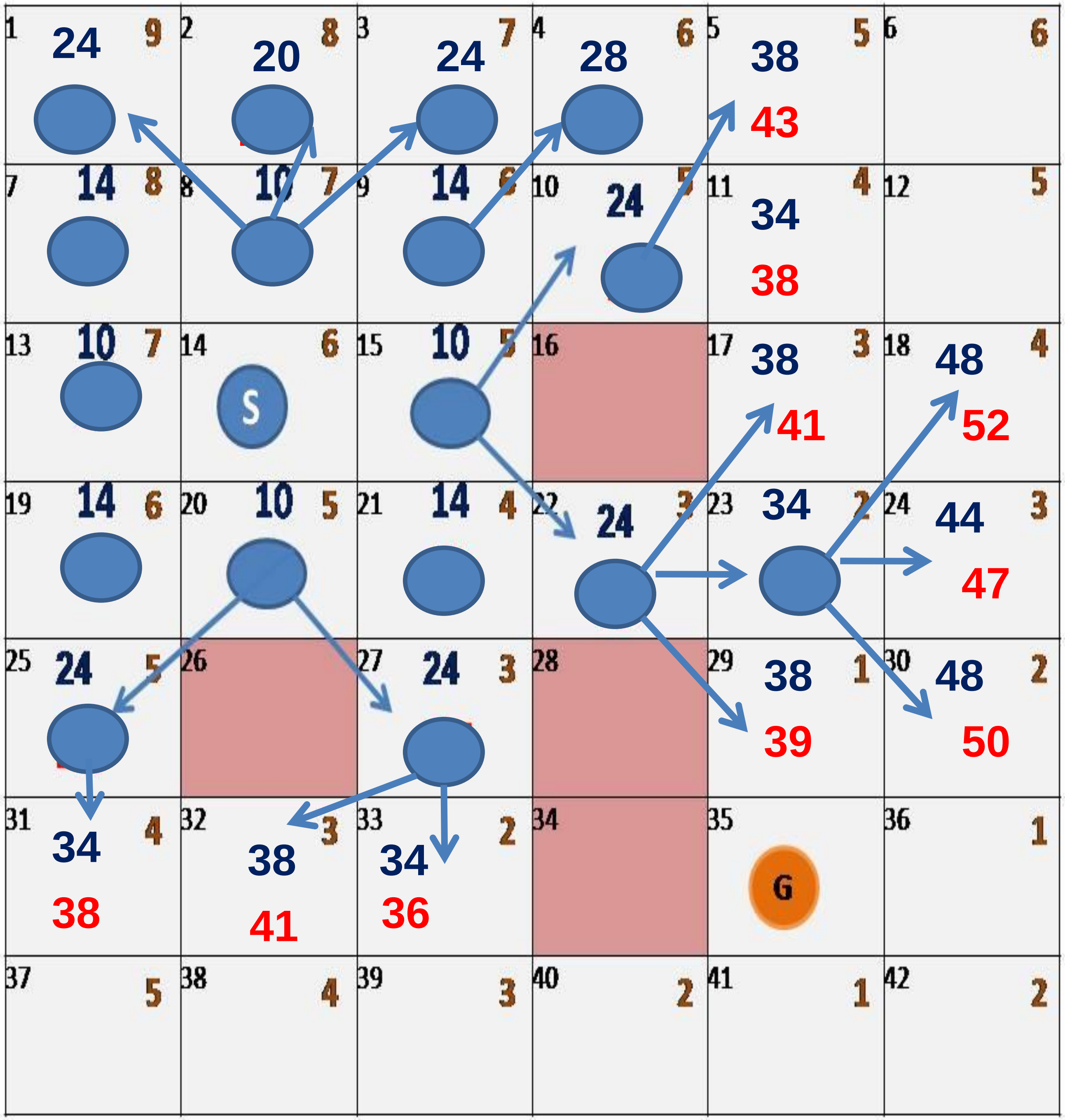
29,

32,33, 5,11,31

18,24,30

CLOSED LIST

14,15,20,8,13,21,9,
19,7,22,27,2,10,25
,3,1, 4,23



OPEN LIST

17,

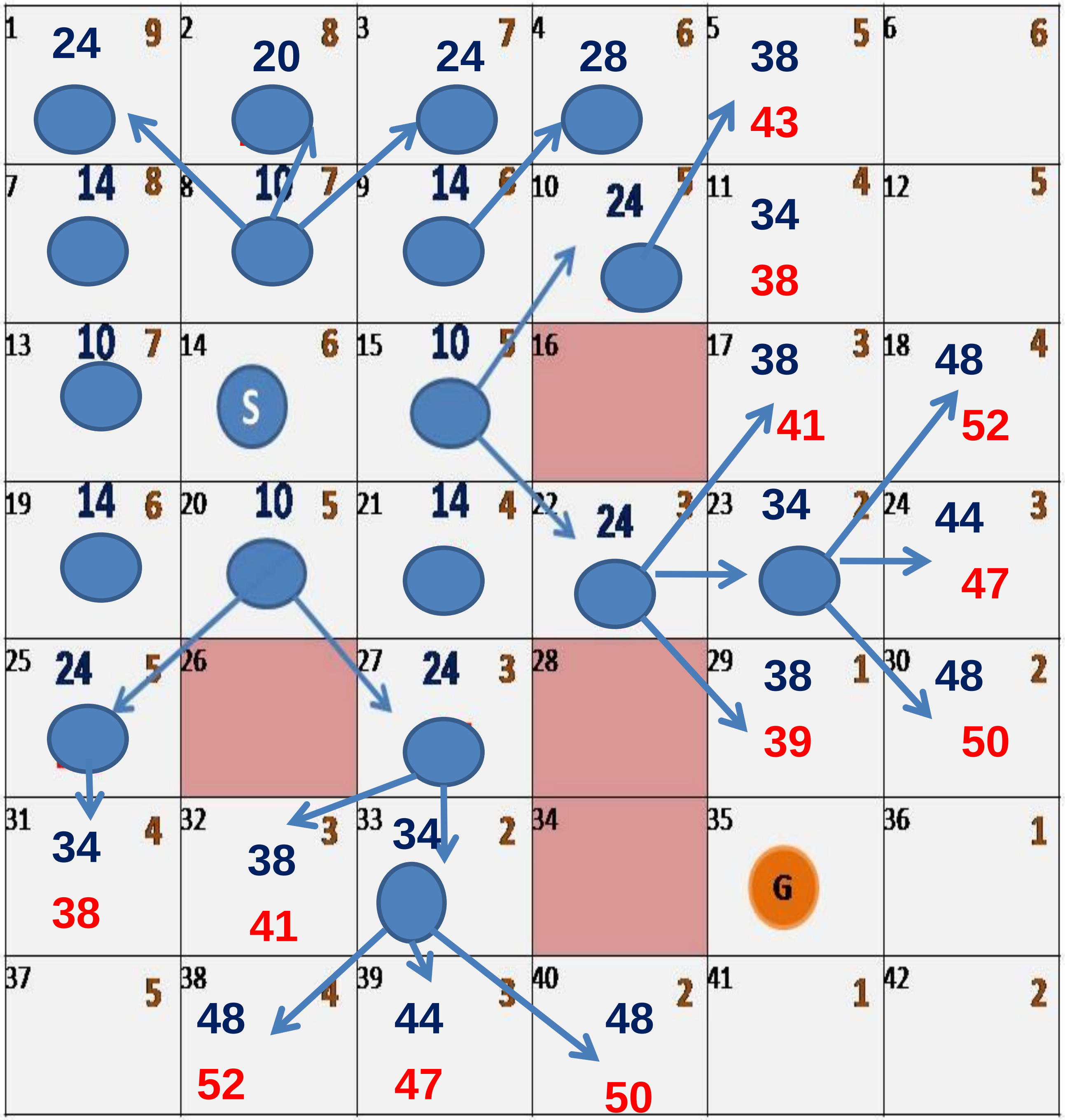
29,

32,33, 5,11,31

18,24,30

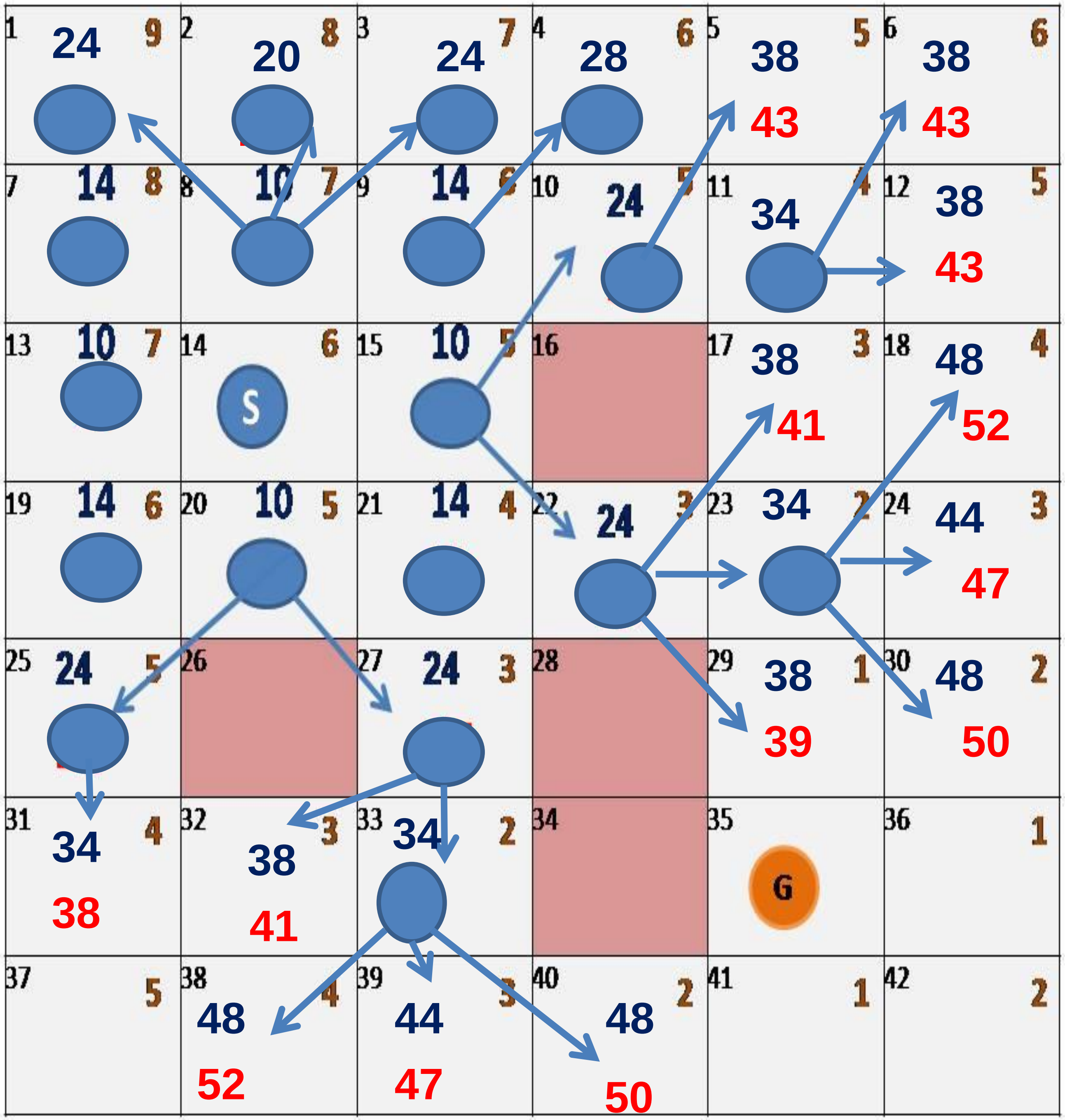
CLOSED LIST

14,15,20,8,13,21,9,
19,7,22,27,2,10,25
,3,1, 4,23



OPEN LIST
17,
29,
32,5,11,31
18,24,30
38,39,40

CLOSED LIST
14,15,20,8,13,21,9,
19,7,22,27,2,10,25
,3,1, 4,23,33

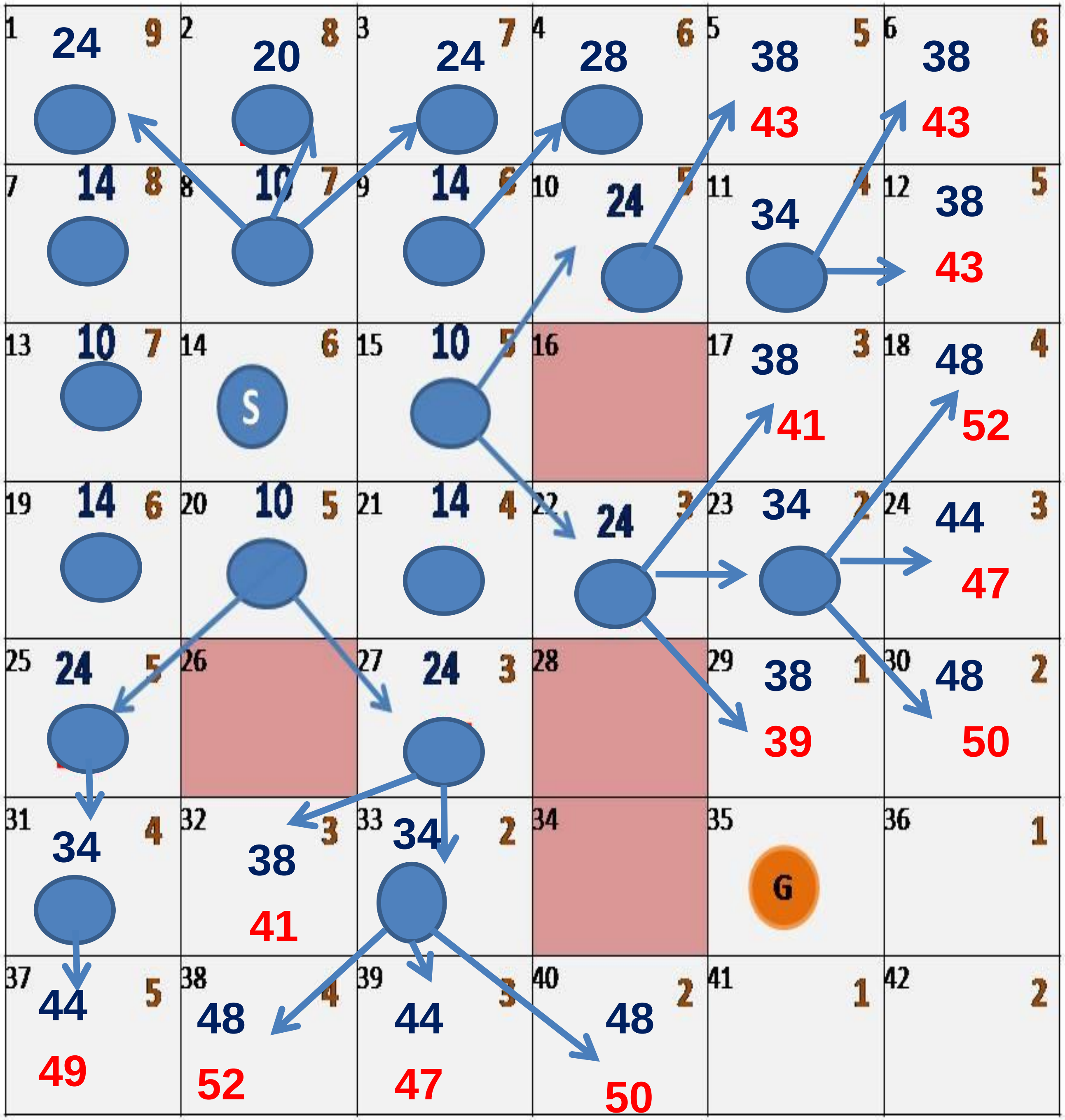


OPEN LIST

17,29,
32,5,31
18,24,30
38,39,40
6,12

CLOSED LIST

14,15,20,8,13,21,9,
19,7,22,27,2,10,25
,3,1, 4,23,33,11

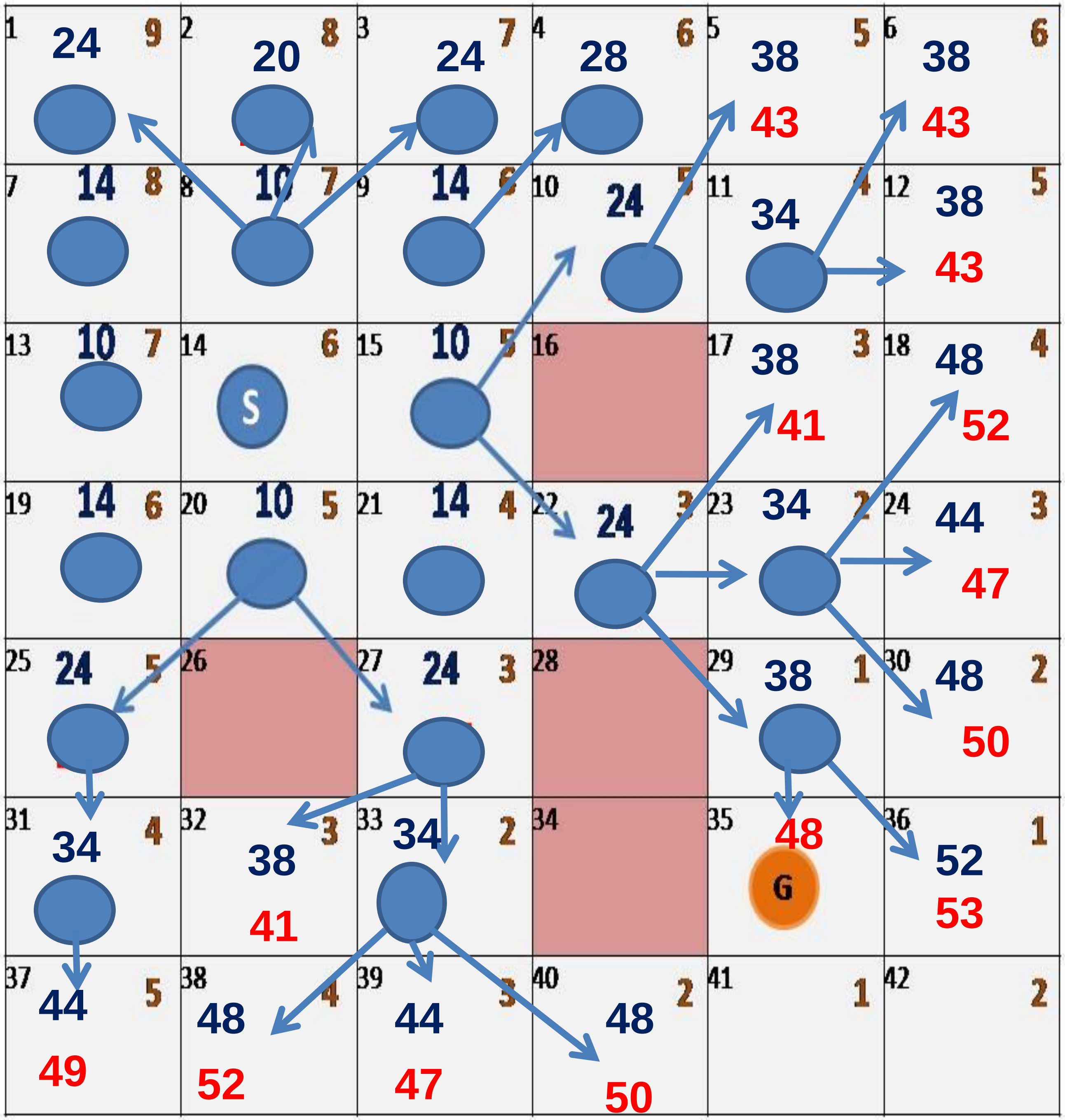


OPEN LIST

17,29,
32,5,
18,24,30
38,39,40 ,6,12
37

CLOSED LIST

14,15,20,8,13,21,9,
19,7,22,27,2,10,25
,3,1, 4,23,33,11,31

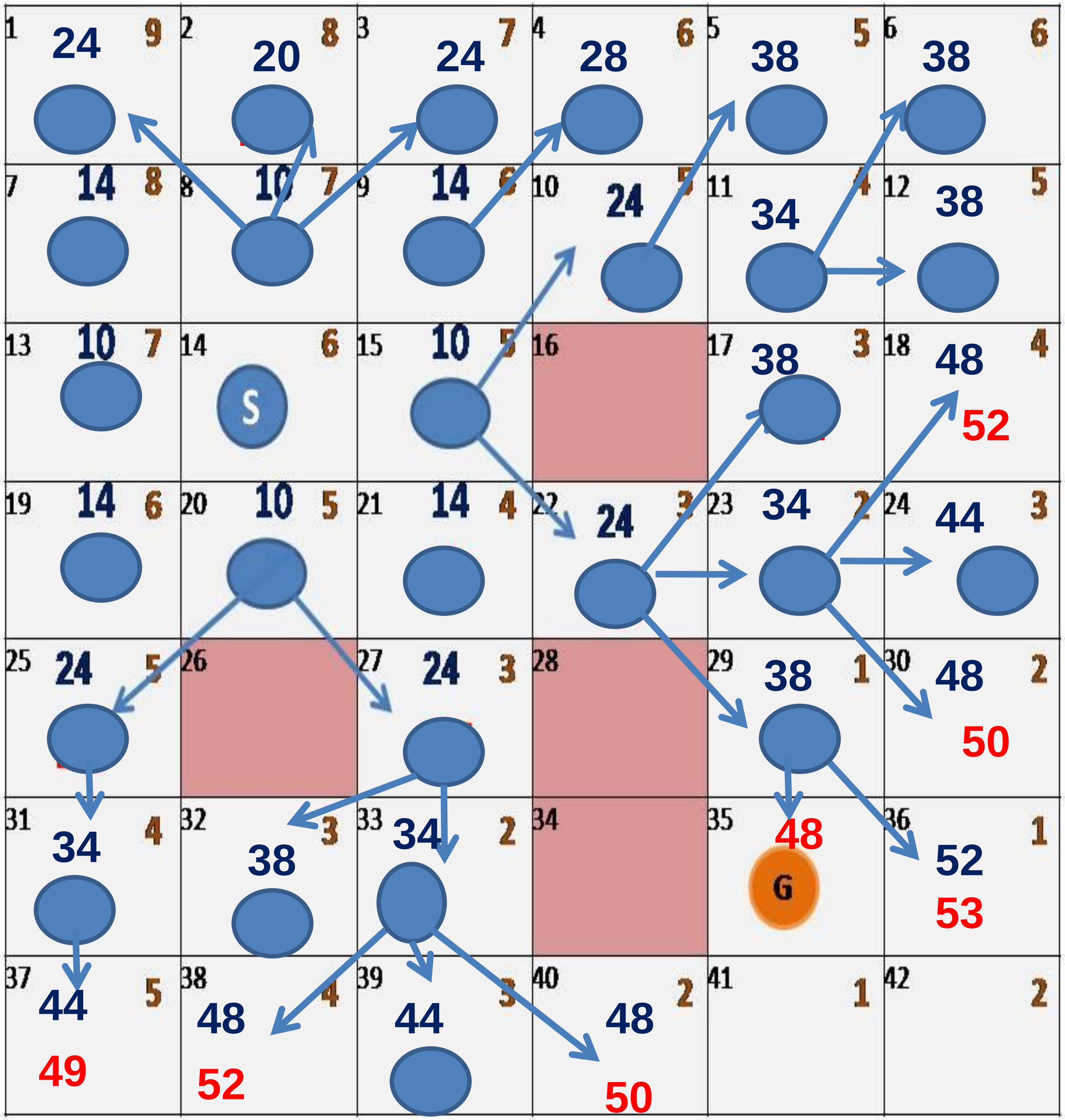


OPEN LIST

17,
32,5,
18,24,30
38,39,40 ,6,12,37
35,36

CLOSED LIST

14,15,20,8,13,21,9,
19,7,22,27,2,10,25
,3,1, 4,23,33,11,31
29



OPEN LIST

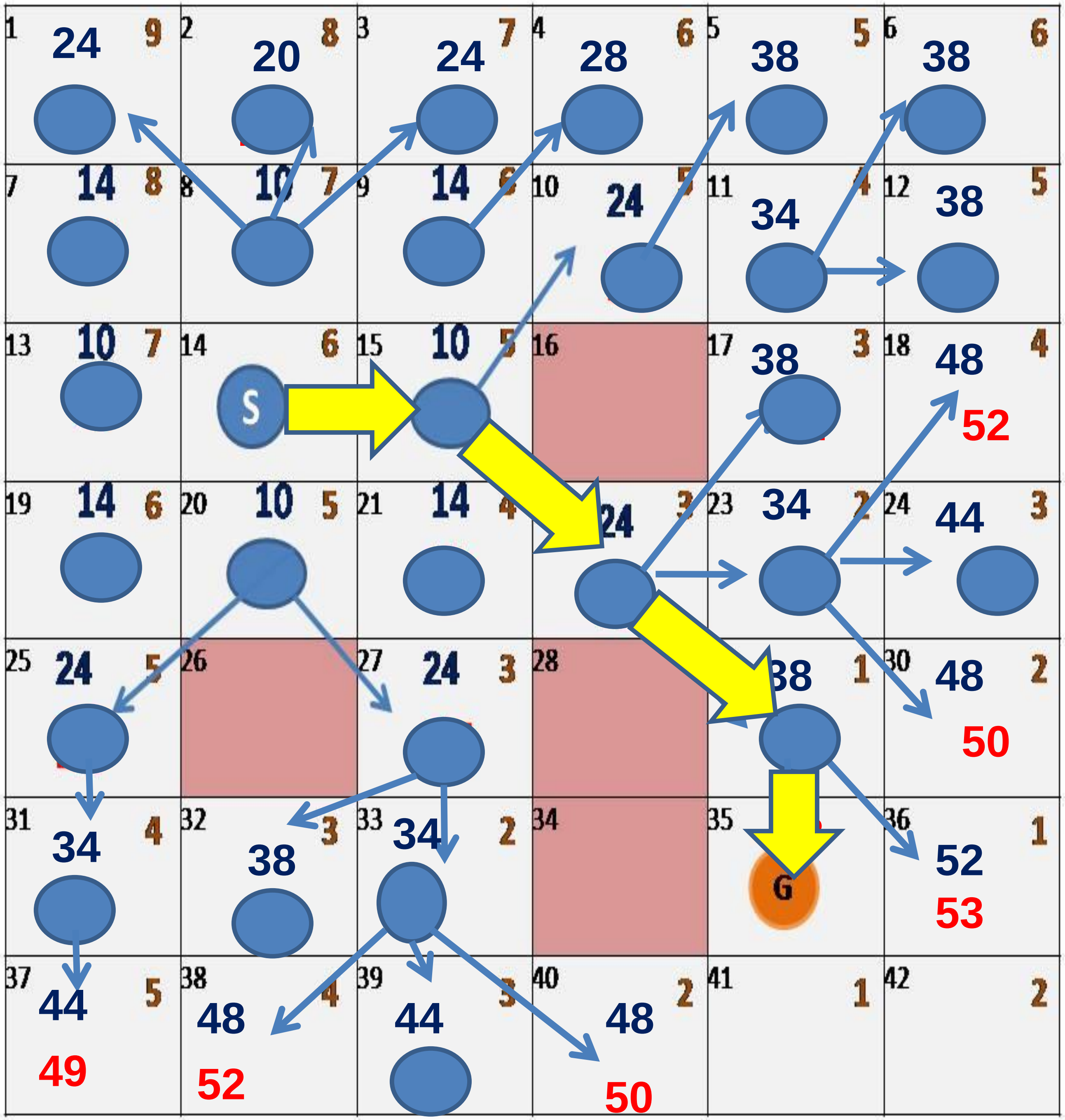
18,30

38,40 ,37

35,36

CLOSED LIST

14,15,20,8,13,21,9,
19,7,22,27,2,10,25
,3,1, 4,23,33,11,31
29
17,32,
5,6,12
39,24



OPEN LIST

18,30

38,40 ,37

36

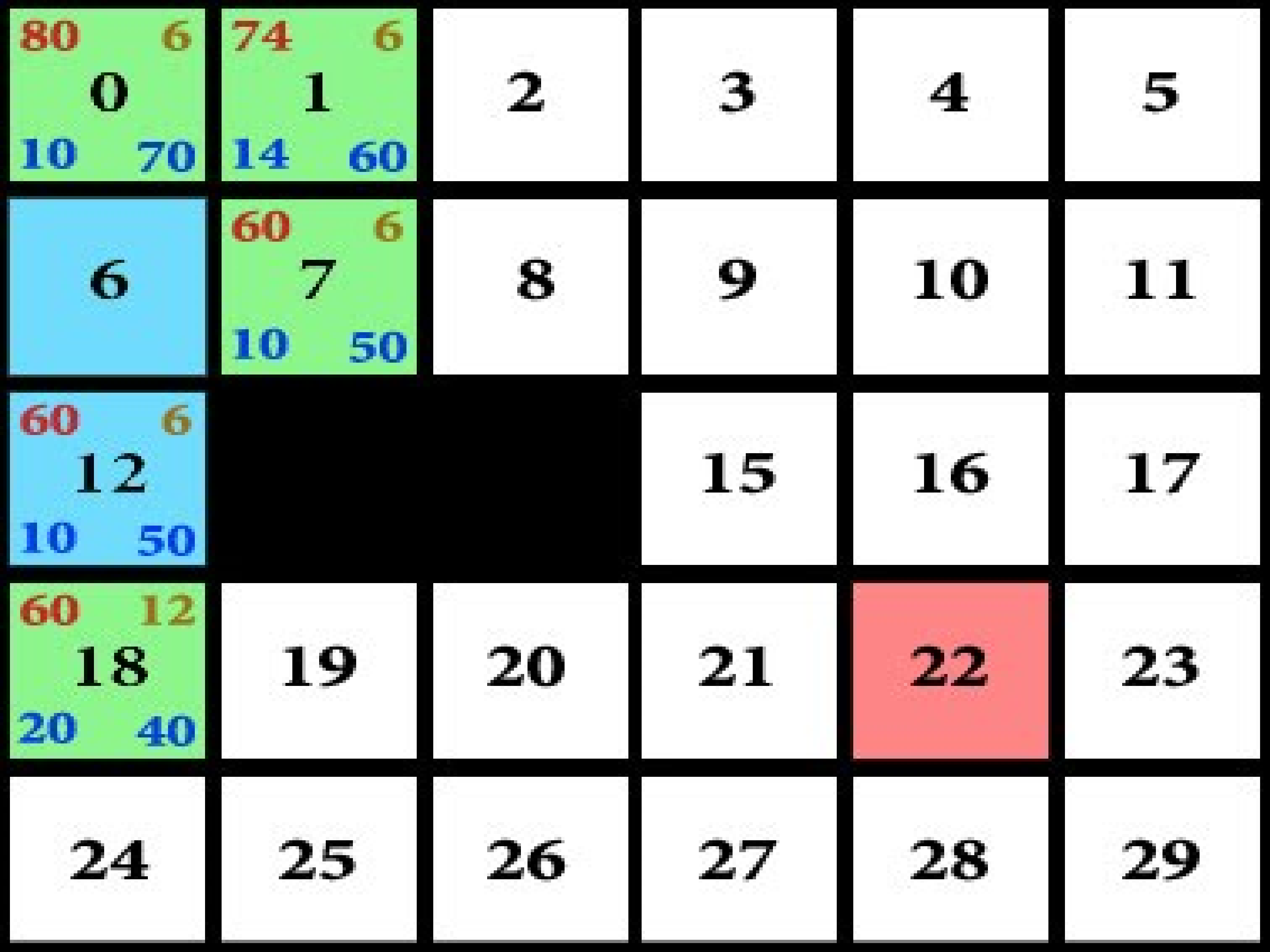
CLOSED LIST

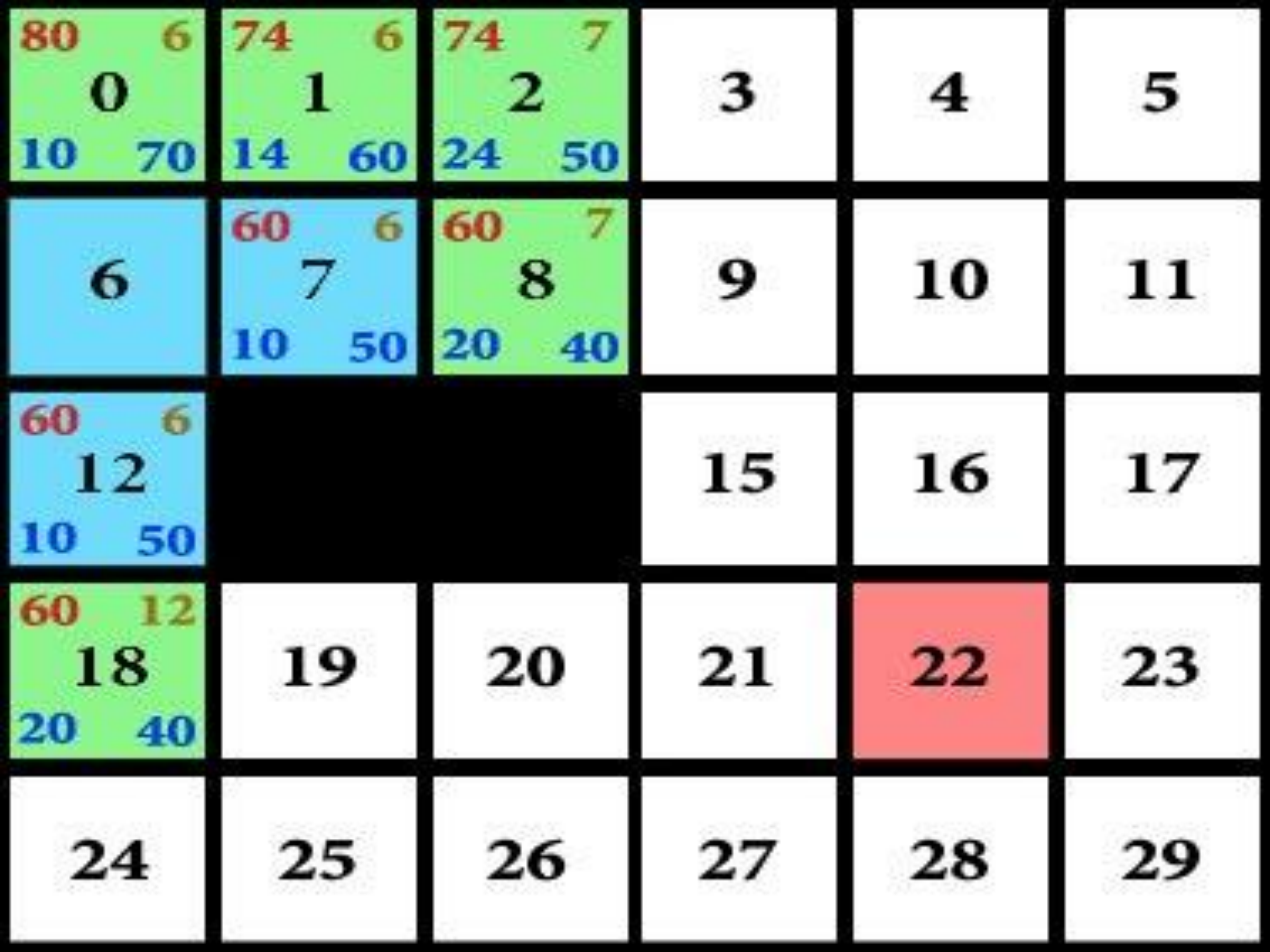
14,15,20,8,13,21,9,
19,7,22,27,2,10,25
,3,1, 4,23,33,11,31
29
17,32,
5,6,12
39,24

35

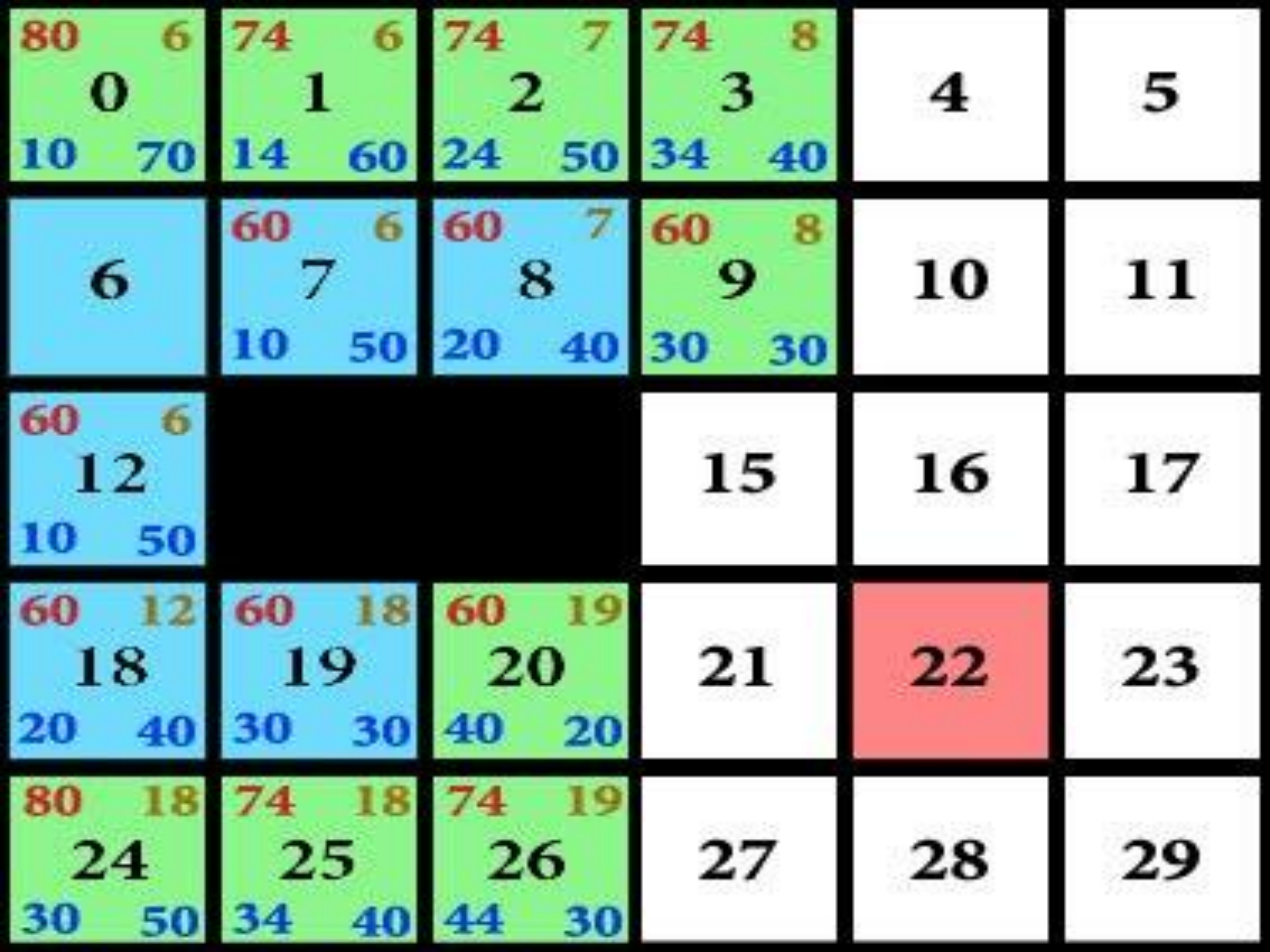
0	1	2	3	4	5
🟩 6	7	8	9	10	11
12	■	■	15	16	17
18	19	20	21	🟥 22	23
24	25	26	27	28	29

🟩 80 6 0 10 70	🟩 74 6 1 14 60	2	3	4	5
🟦 6	🟩 60 6 7 10 50	8	9	10	11
🟩 60 6 12 10 50	■	■	15	16	17
18	19	20	21	🟥 22	23
24	25	26	27	28	29

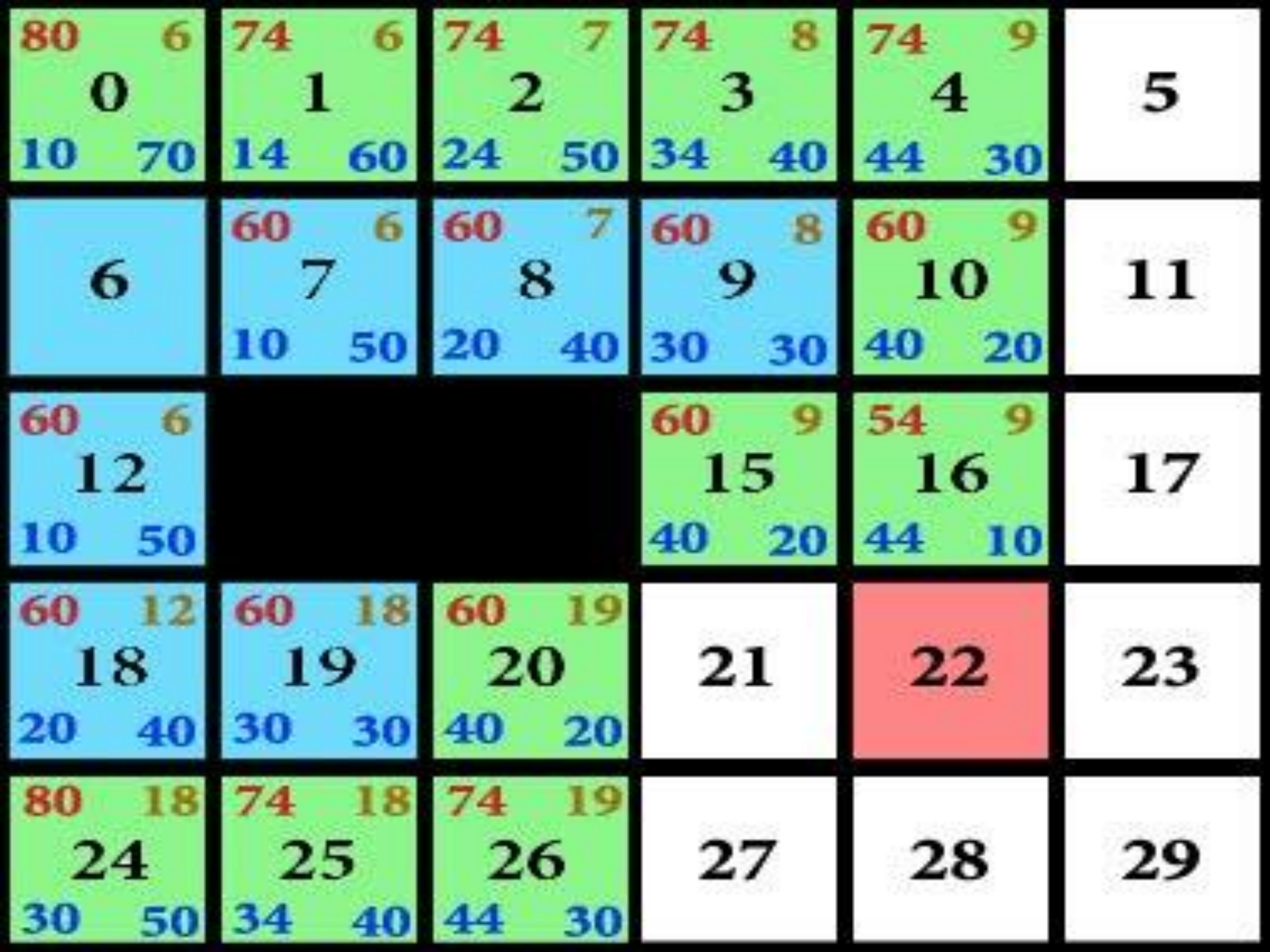

80
6
0
10
70
74
6
1
14
60
2
3
4
5
6
60
6
7
10
50
8
9
10
11
60
6
12
10
50
15
16
17
60
12
18
20
40
19
20
21
22
23
24
25
26
27
28
29



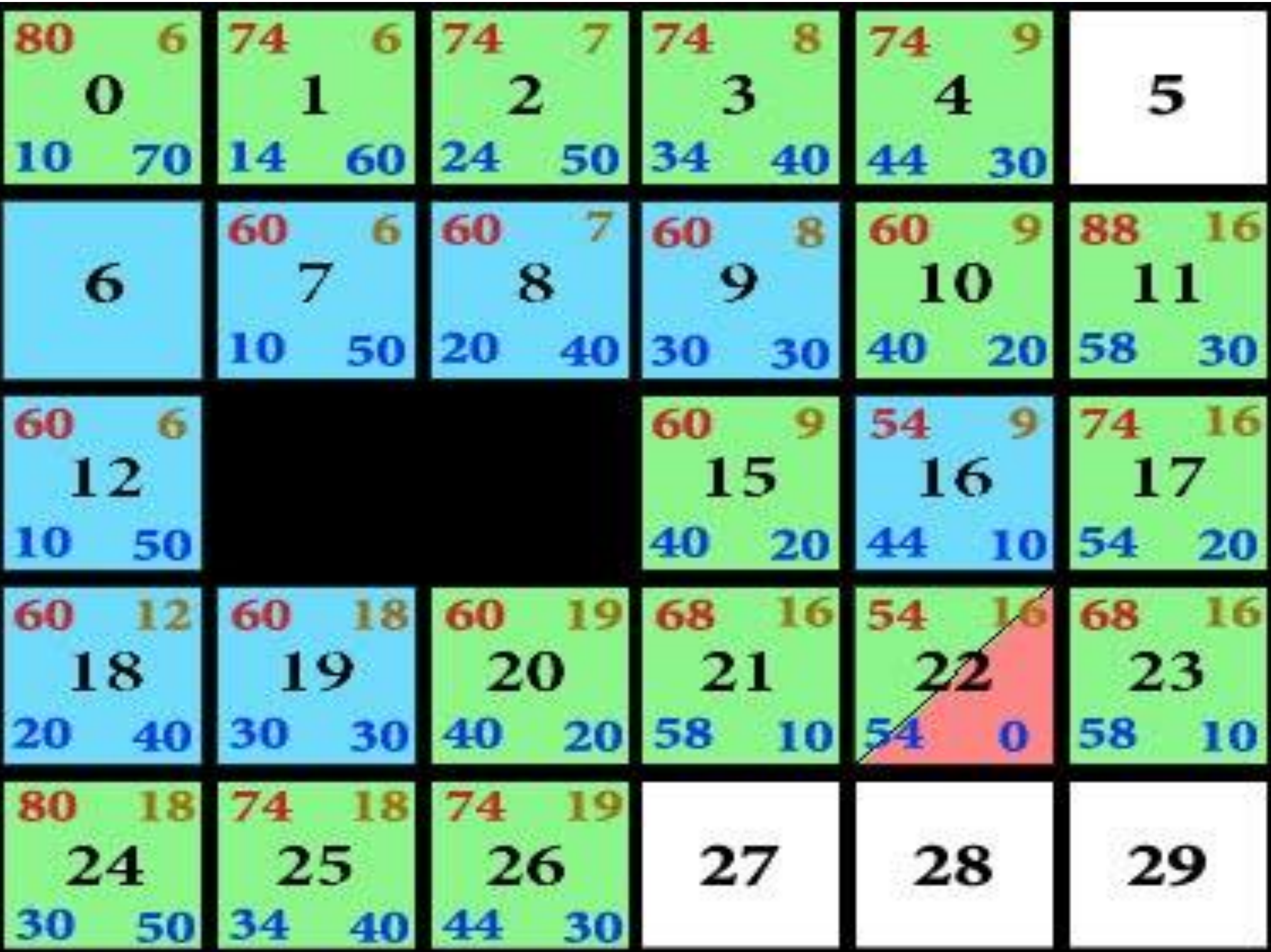
80 6 0 10 70	74 6 1 14 60	74 7 2 24 50	3	4	5
6	60 6 7 10 50	60 7 8 20 40	9	10	11
60 6 12 10 50			15	16	17
60 12 18 20 40	19	20	21	22	23
24	25	26	27	28	29



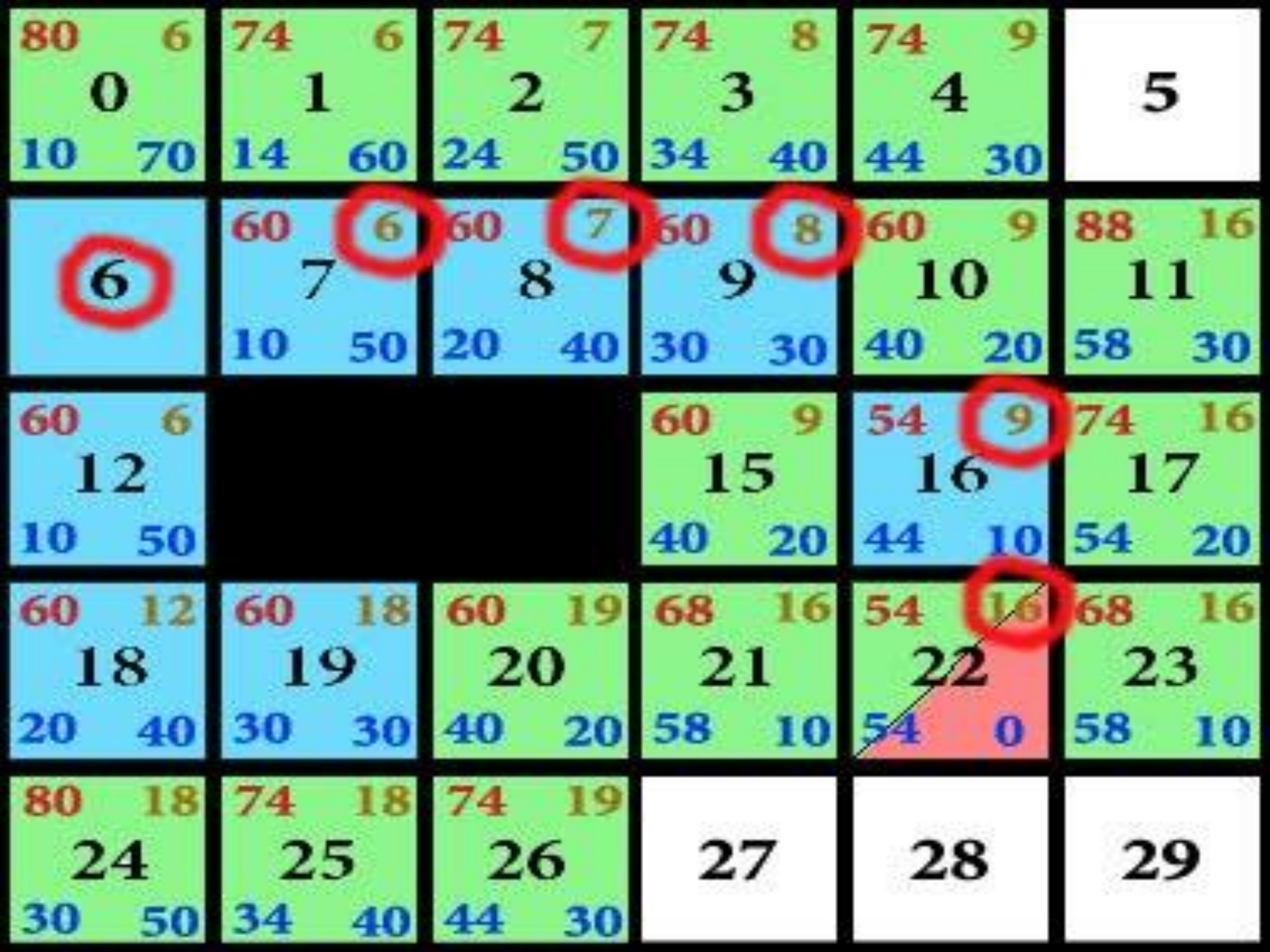
80 6 0 10 70	74 6 1 14 60	74 7 2 24 50	74 8 3 34 40	4	5
6	60 6 7 10 50	60 7 8 20 40	60 8 9 30 30	10	11
60 6 12 10 50			15	16	17
60 12 18 20 40	60 18 19 30 30	60 19 20 40 20	21	22	23
80 18 24 30 50	74 18 25 34 40	74 19 26 44 30	27	28	29



🟩 80 6 0 10 70	🟩 74 6 1 14 60	🟩 74 7 2 24 50	🟩 74 8 3 34 40	🟩 74 9 4 44 30	5
🟦 6	🟦 60 6 7 10 50	🟦 60 7 8 20 40	🟦 60 8 9 30 30	🟩 60 9 10 40 20	11
🟦 60 6 12 10 50	⬛	⬛	🟩 60 9 15 40 20	🟩 54 9 16 44 10	17
🟦 60 12 18 20 40	🟦 60 18 19 30 30	🟩 60 19 20 40 20	21	🟥 22	23
🟩 80 18 24 30 50	🟩 74 18 25 34 40	🟩 74 19 26 44 30	27	28	29



80 6 0 10 70	74 6 1 14 60	74 7 2 24 50	74 8 3 34 40	74 9 4 44 30	5
6	60 6 7 10 50	60 7 8 20 40	60 8 9 30 30	60 9 10 40 20	88 16 11 58 30
60 6 12 10 50			60 9 15 40 20	54 9 16 44 10	74 16 17 54 20
60 12 18 20 40	60 18 19 30 30	60 19 20 40 20	68 16 21 58 10	54 16 22 54 0	68 16 23 58 10
80 18 24 30 50	74 18 25 34 40	74 19 26 44 30	27	28	29



80 6 0 10 70	74 6 1 14 60	74 7 2 24 50	74 8 3 34 40	74 9 4 44 30	5
6	60 6 7 10 50	60 7 8 20 40	60 8 9 30 30	60 9 10 40 20	88 16 11 58 30
60 6 12 10 50			60 9 15 40 20	54 9 16 44 10	74 16 17 54 20
60 12 18 20 40	60 18 19 30 30	60 19 20 40 20	68 16 21 58 10	54 16 22 54 0	68 16 23 58 10
80 18 24 30 50	74 18 25 34 40	74 19 26 44 30	27	28	29

0	1	2	3	4	5
🟩 6	🟦 7	🟦 8	🟦 9	10	11
12	■	■	15	🟦 16	17
18	19	20	21	🟥 22	23
24	25	26	27	28	29