IART - Artificial Intelligence

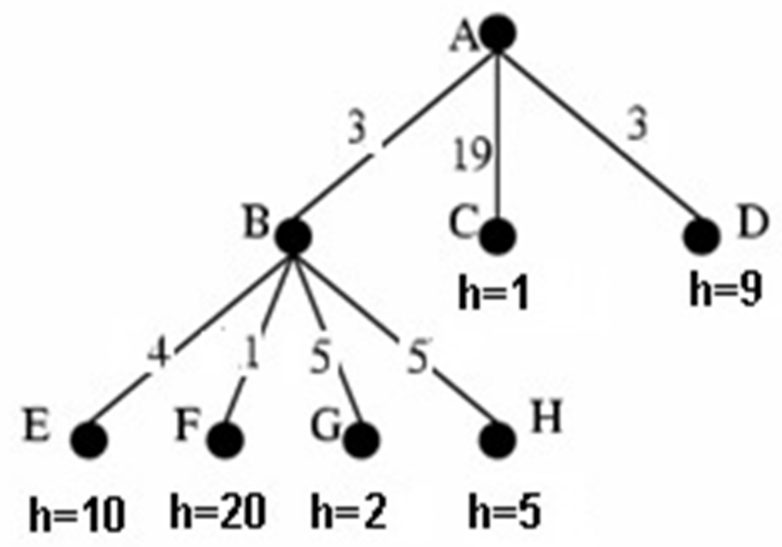
Exercise 2: Solving Search Problems

Luís Paulo Reis

LIACC – Artificial Intelligence and Computer Science Lab.
DEI/FEUP – Informatics Engineering Department, Faculty of Engineering of the University of Porto, Portugal
APPIA – Portuguese Association for Artificial Intelligence

Exercise 2.1: Search Strategies

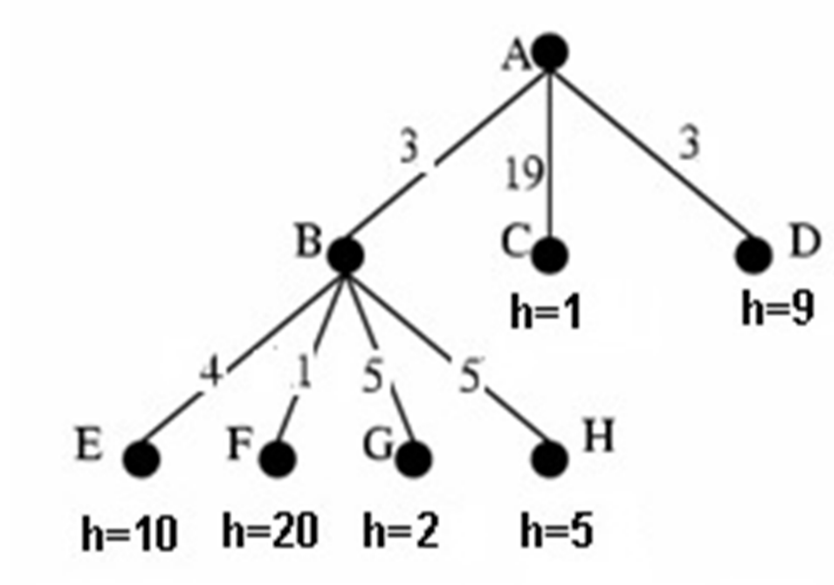
Assuming the following search tree in which each arc displays the cost of the corresponding operator g(n), and the nodes contain the value of the heuristic function h(n), indicate justifying, which node is expanded next using each of the following methods:



a) Breadth-First Search ("Pesquisa Primeiro em Largura")
b) Depth-First Search ("Pesquisa Primeiro em Profundidade")
c) Uniform Cost Search ("Pesquisa de Custo Uniforme")
d) Greedy Search ("Pesquisa Gulosa")
e) A* Algorithm Search ("Pesquisa com Algoritmo A*")

Exercise 2.1: Search Strategies

Assuming the following search tree in which each arc displays the cost of the corresponding operator g(n), and the nodes contain the value of the heuristic function h(n), indicate justifying, which node is expanded next using each of the following methods:

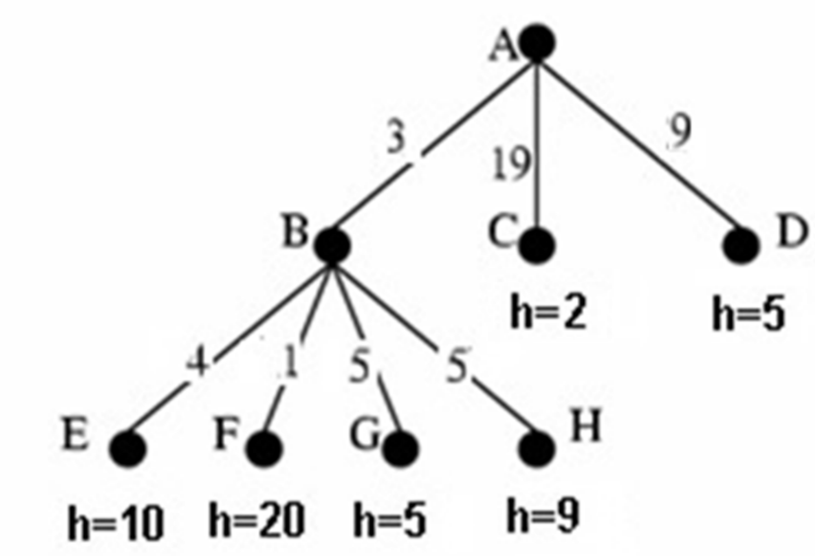


a) Breadth-First Search ("Pesquisa Primeiro em Largura"): **C**
b) Depth-First Search ("Pesquisa Primeiro em Profundidade"): **E**
c) Uniform Cost Search ("Pesquisa de Custo Uniforme"): **D**
d) Greedy Search ("Pesquisa Gulosa"): **C**
e) A* Algorithm Search ("Pesquisa com Algoritmo A*"): **G**

Exercise 2.1b: Search Strategies

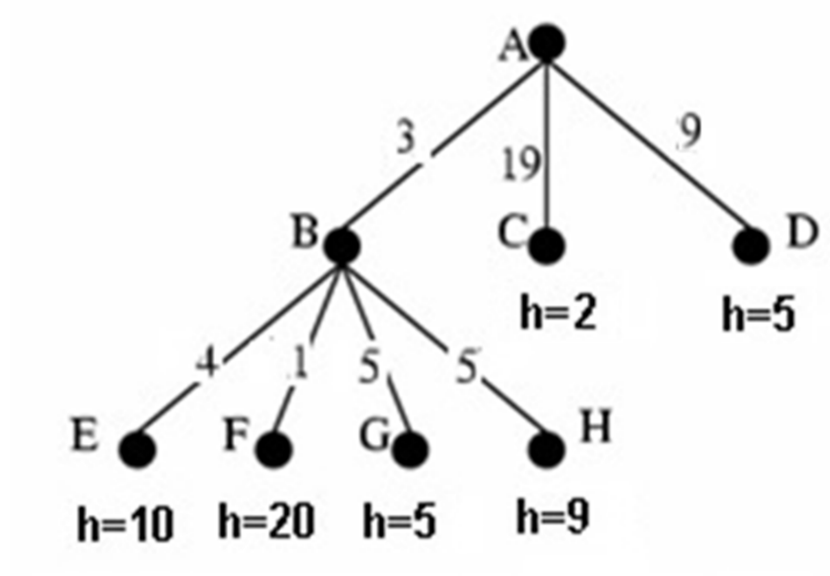
Assuming the following search tree in which each arc displays the cost of the corresponding operator g(n), and the nodes contain the value of the heuristic function h(n), indicate justifying, which node is expanded next using each of the following methods:

a) Breadth-First Search ("Pesquisa Primeiro em Largura")
b) Depth-First Search ("Pesquisa Primeiro em Profundidade")
c) Uniform Cost Search ("Pesquisa de Custo Uniforme")
d) Greedy Search ("Pesquisa Gulosa")
e) A* Algorithm Search ("Pesquisa com Algoritmo A*")



Exercise 2.1b: Search Strategies

Assuming the following search tree in which each arc displays the cost of the corresponding operator g(n), and the nodes contain the value of the heuristic function h(n), indicate justifying, which node is expanded next using each of the following methods:



a) Breadth-First Search ("Pesquisa Primeiro em Largura"): **C**
b) Depth-First Search ("Pesquisa Primeiro em Profundidade"): **E**
c) Uniform Cost Search ("Pesquisa de Custo Uniforme"): **F**
d) Greedy Search ("Pesquisa Gulosa"): **C**
e) A* Algorithm Search ("Pesquisa com Algoritmo A*"): **G**

Exercise 2.2: Solving the N Puzzle Problem

The objective of this exercise is the application of search methods, with emphasis on informed search methods and the A* algorithm, to solve the well-known N-Puzzle problem. The desired objective state for the puzzle is as follows (0 represents the empty space):

9Puzzle

1	2	3
4	5	6
7	8	0

16Puzzle

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	0

Starting from a given initial state, the goal is to determine which operations to perform to solve the puzzle, reaching the desired objective state.

a) Formulate the problem as a search problem indicating the state representation, operators (their names, preconditions, effects, and cost), initial state, and objective test.

Solving the N Puzzle Problem

- **State Representation:**

  Matrix with Board: *B[3,3], B[4,4]* or in the general case *B[N,N]* filled with values 0..8 or in the general case 0..NxN-1 *// 0 represents the empty square*

  Good idea to add to the state the pair *(Xs, Ys),* i.e. the position of the empty square, for efficiency…

- **Initial State:**

  Matrix *B* filled with the desired initial state, *(Xs,Ys)= position of empty sq.*

- **Objective State:**

  Matrix *B* filled with values shown in previous slides

1	2	3
4	5	6
7	8	0

- **Operators:**

Solving the N Puzzle Problem

- **State Representation:**

  Matrix with Board: *B[3,3], B[4,4]* or in the general case *B[N,N]* filled with values 0..8 or in the general case 0..NxN-1 *// 0 represents the empty square*

  Good idea to add to the state the pair *(Xs, Ys),* i.e. the position of the empty square, for efficiency

- **Initial State:**

  Matrix *B* filled with the desired initial state, *(Xs,Ys)= position of empty sq.*

- **Objective State:**

  Matrix *B* filled with values shown in previous slides

1	2	3
4	5	6
7	8	0

- **Operators:**

  up, down, left, right   //Move the empty square in the direction shown

Solving the N Puzzle Problem

- **State Representation:**

  Matrix with Board: *B[3,3], B[4,4]* or in the general case *B[N,N]* filled with values 0..8 or in the general case 0..NxN-1 *// 0 represents the empty square*

  Good idea to add to the state the pair *(Xs, Ys),* i.e. the position of the empty square, for efficiency

- **Initial State:**

  Matrix *B* filled with the desired initial state, *(Xs,Ys)= position of empty sq.*

- **Objective State:**

  Matrix *B* filled with values shown in previous slides

1	2	3
4	5	6
7	8	0

- **Operators (4 possibilities):**

up, down, left, right	//Move the empty square in the direction shown
move(Dir)	//Move the empty square in direction Dir
move(Xdir, Ydir)	//Move the empty square in direction Xdir, Ydir
move(x1,y1,x2,y2)	//Exchange pieces (x1,y1) (x2,y2) – *not a very good idea!*

Solving the N Puzzle Problem

Y \ X	1	2	3
1	1	2	3
2	4	5	6
3	7	8	0

Operators (using possibility 1):

up, down, left, right

Name	PreCond	Effects	Cost
up			
down			
left			
right			

Solving the N Puzzle Problem

Operators (using possibility 1):

up, down, left, right

Y \ X	1	2	3
1	1	2	3
2	4	5	6
3	7	8	0

Name	PreCond	Effects	Cost
up	Ys>1	B[Xs,Ys]=B[Xs,Ys-1]; B[Xs,Ys-1]=0; Ys=Ys-1	1
down			
left			
right			

Solving the N Puzzle Problem

Y \ X	1	2	3
1	1	2	3
2	4	5	6
3	7	8	0

Operators (using possibility 1):

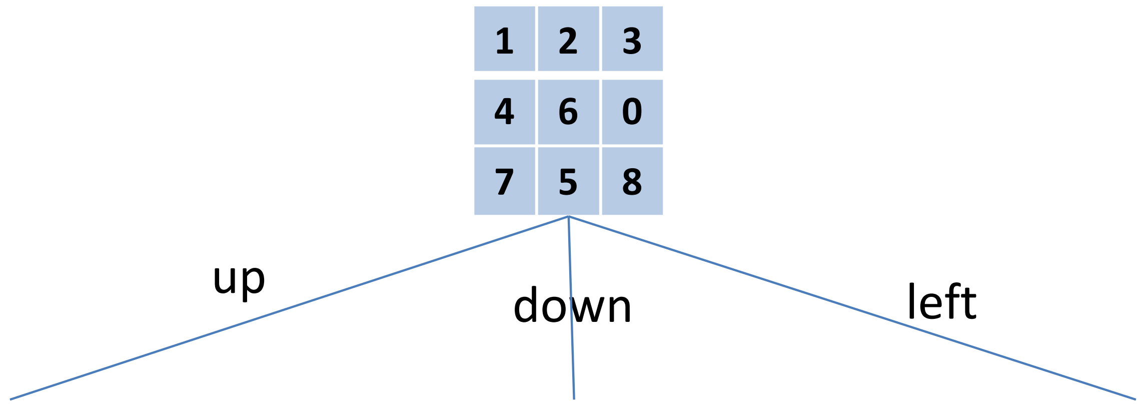
up, down, left, right

Name	PreCond	Effects	Cost
up	**Ys>1**	**B[Xs,Ys]=B[Xs,Ys-1]; B[Xs,Ys-1]=0; Ys=Ys-1**	**1**
down	**Ys<N**	**B[Xs,Ys]=B[Xs,Ys+1]; B[Xs,Ys+1]=0; Ys=Ys+1**	**1**
left	**Xs>1**	**B[Xs,Ys]=B[Xs-1,Ys]; B[Xs-1,Ys]=0; Xs=Xs-1**	**1**
right	**Xs<N**	**B[Xs,Ys]=B[Xs+1,Ys]; B[Xs+1,Ys]=0; Xs=Xs+1**	**1**

Very simple formulation using the State Representation defined!

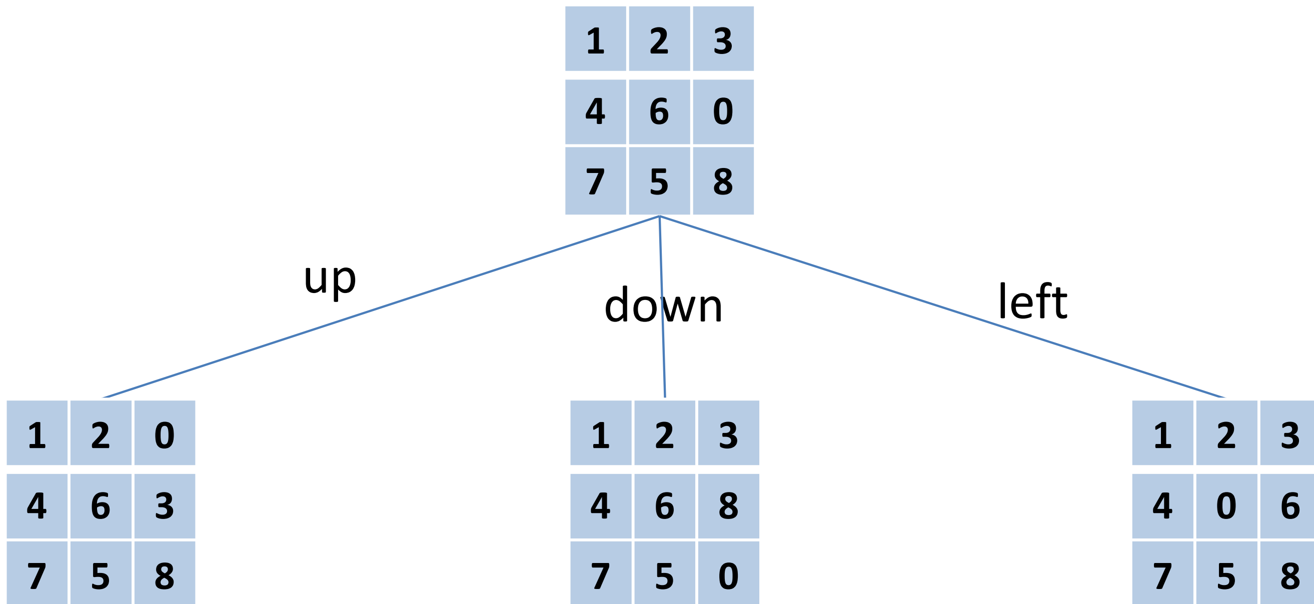
Breadth First Search - Example

up
down
left
right



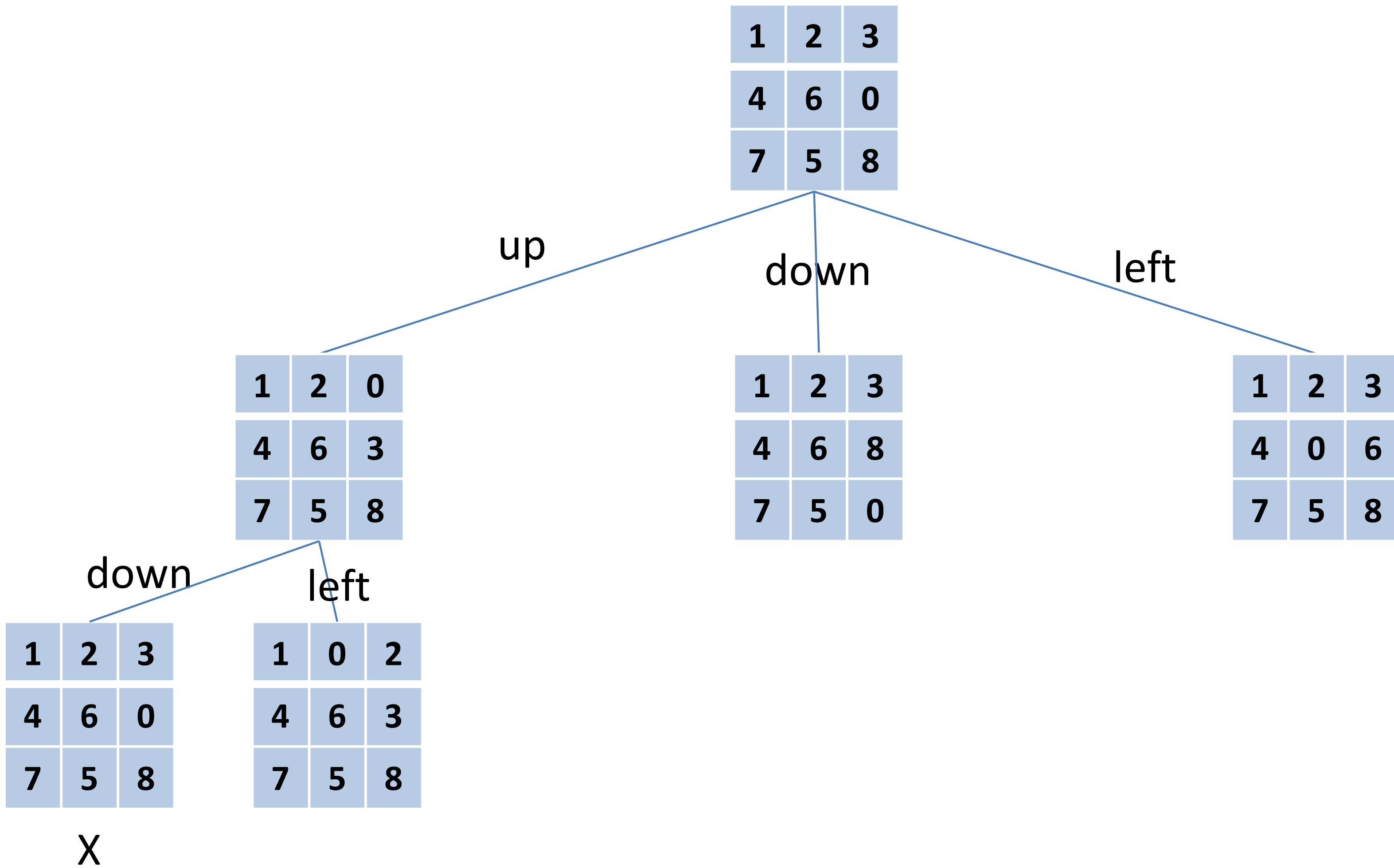
Breadth First Search

up
down
left
right



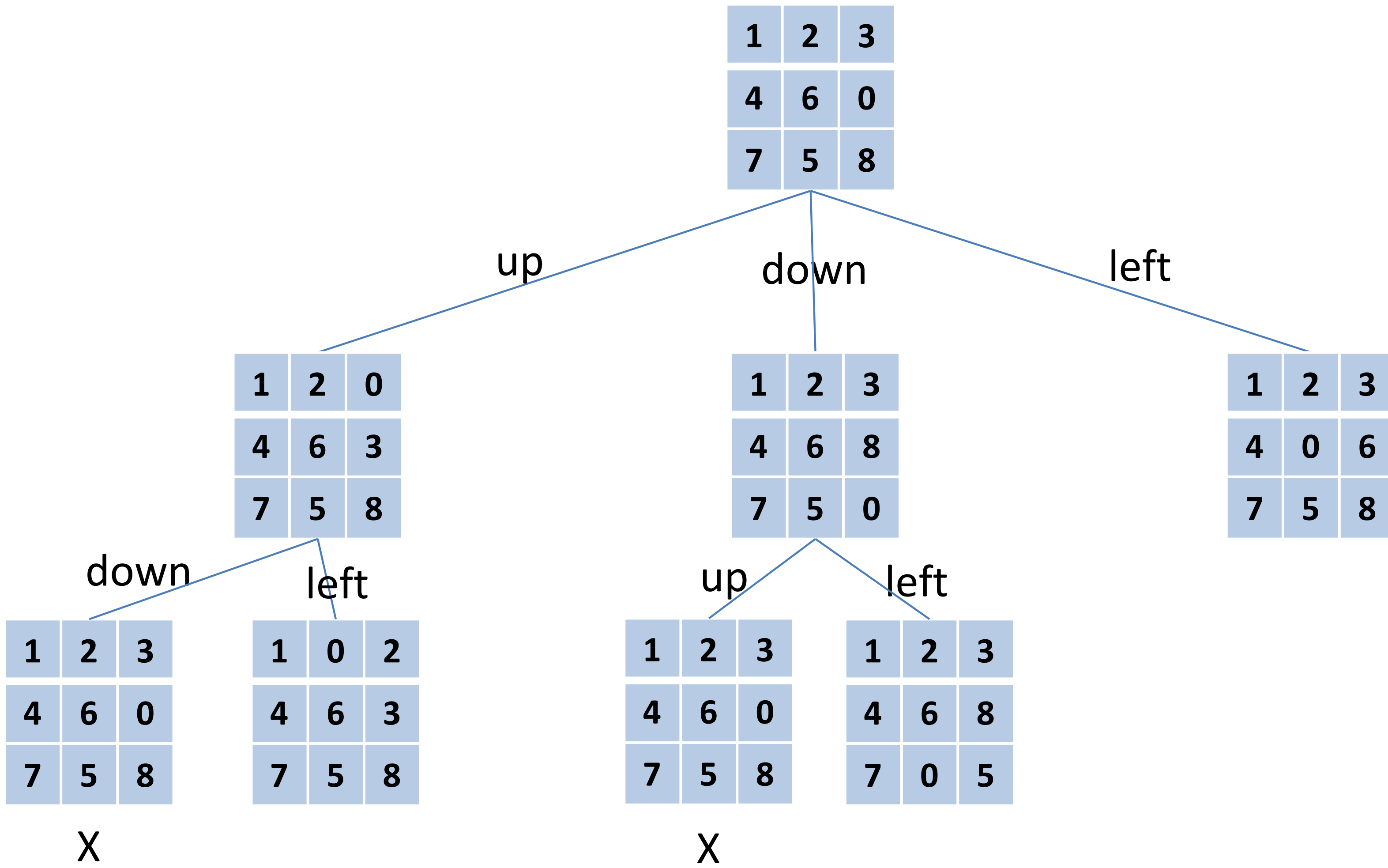
Breadth First Search

up
down
left
right



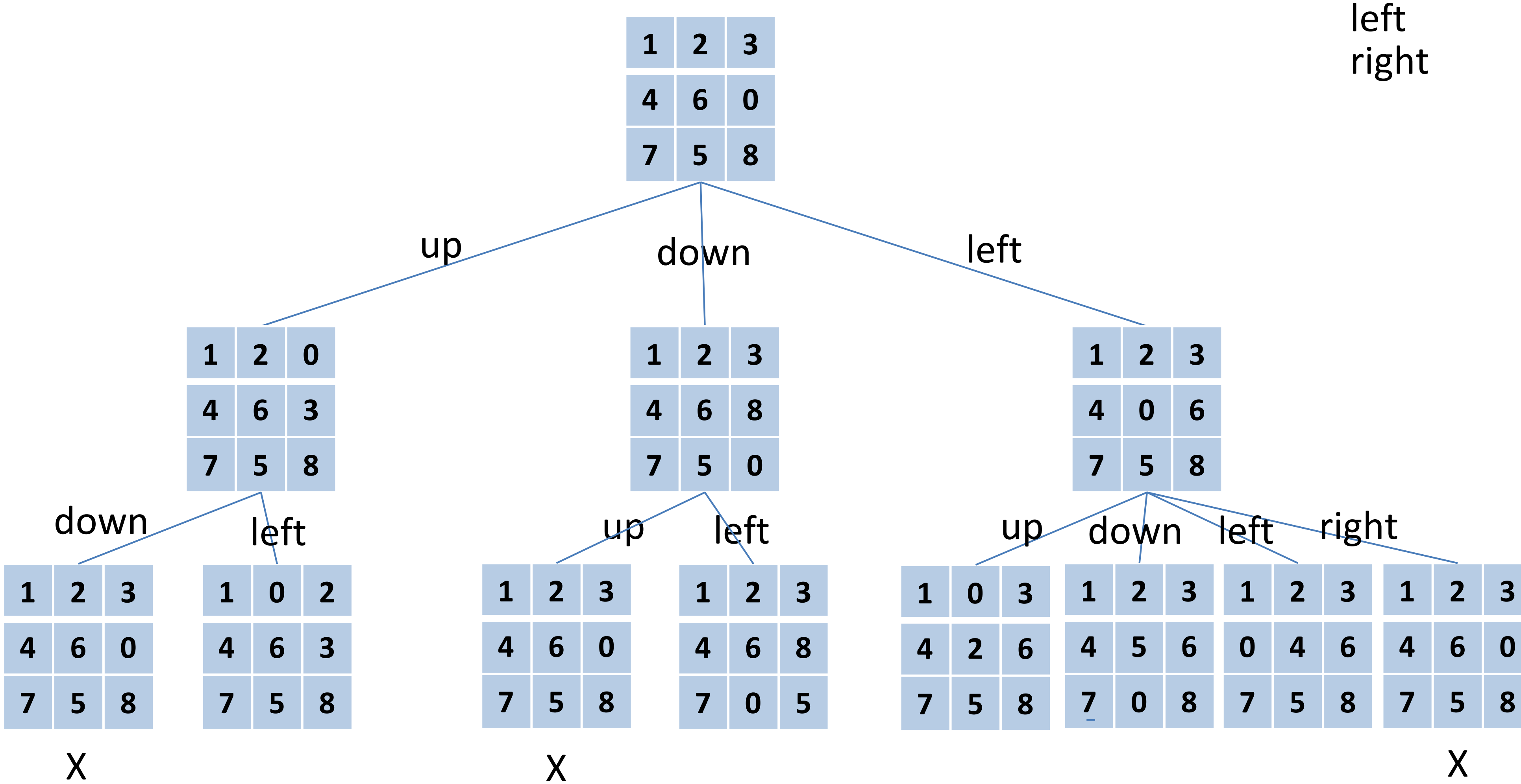
Breadth First Search

up
down
left
right



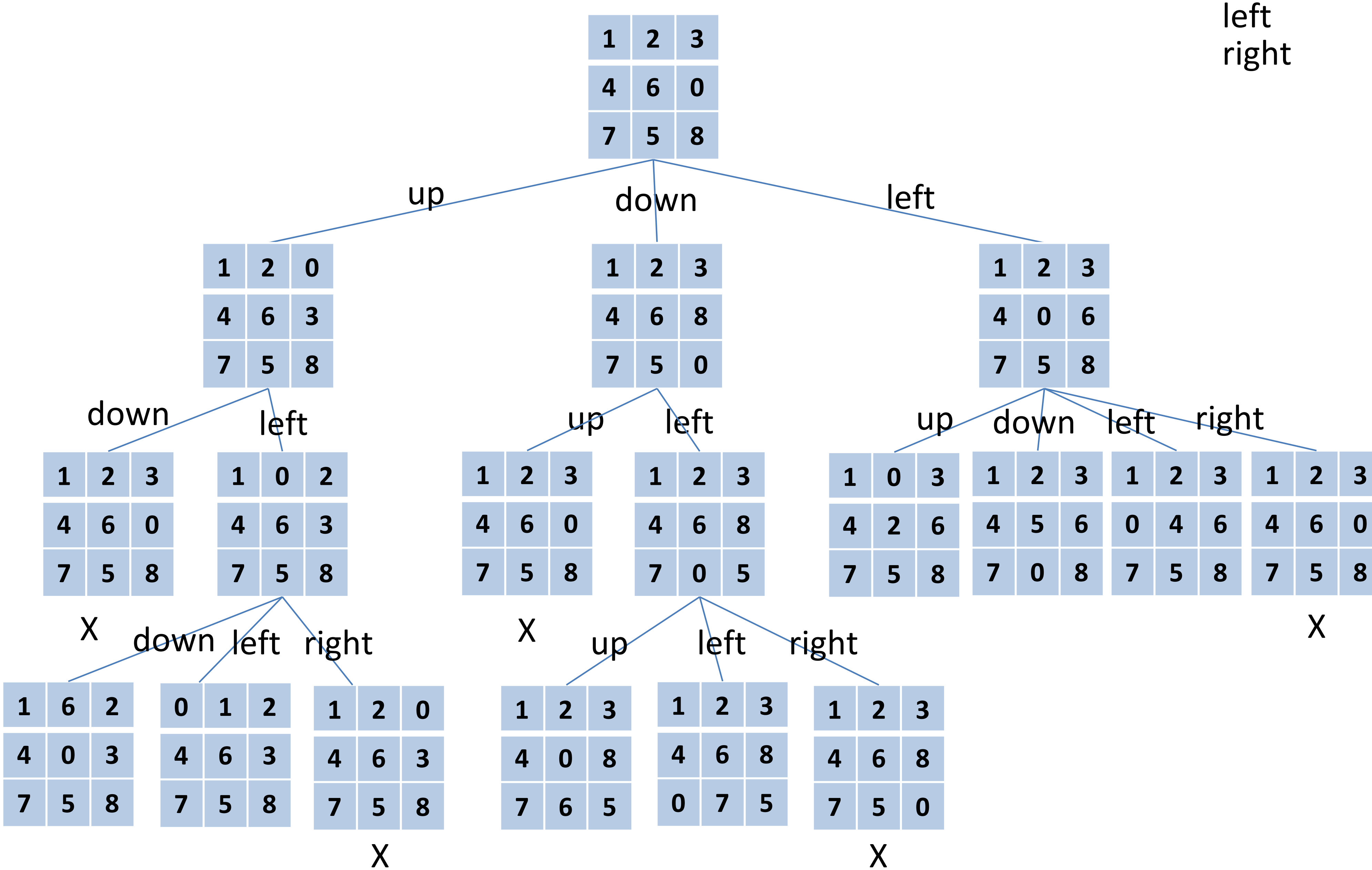
Breadth First Search

up
down
left
right



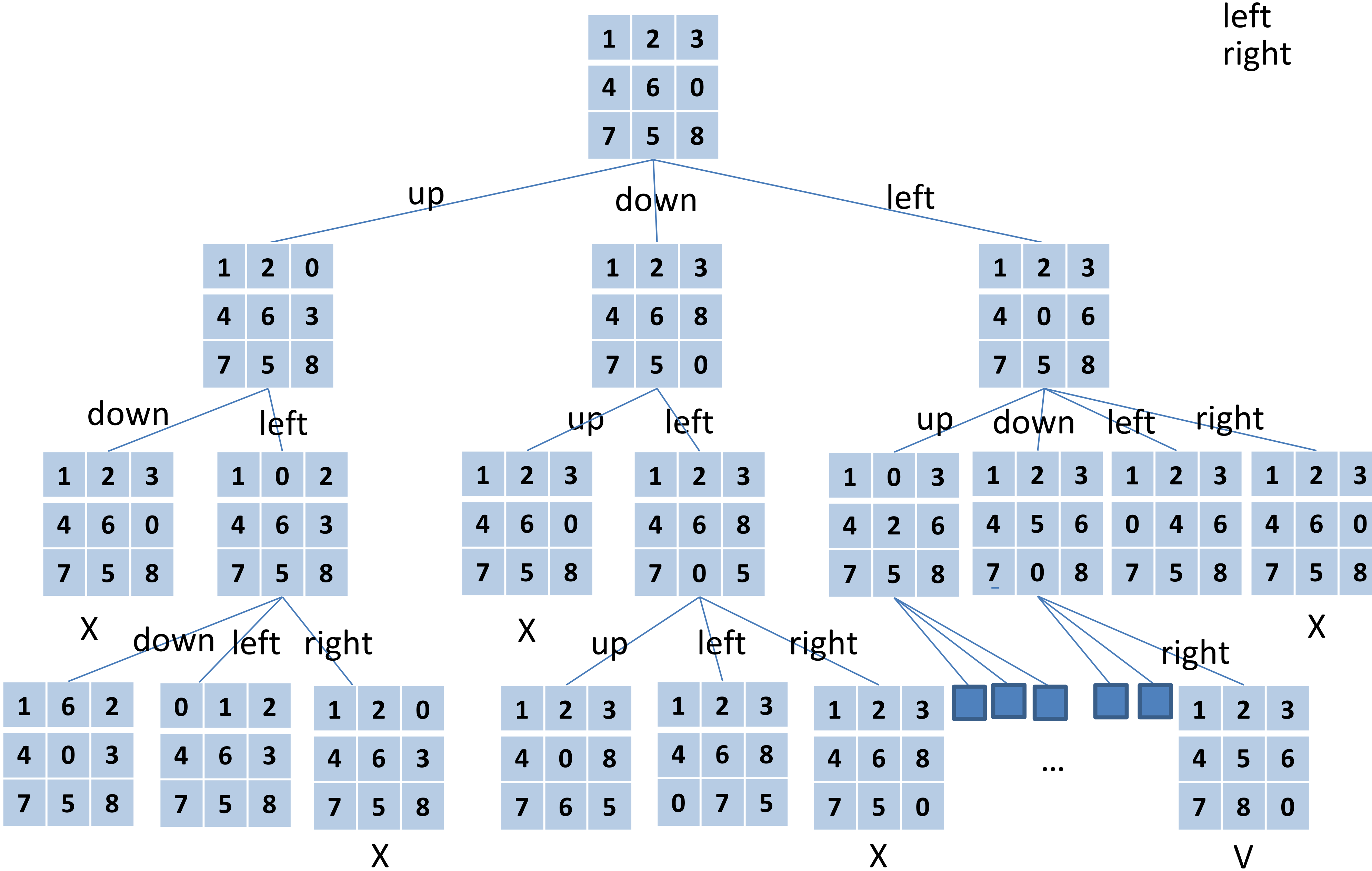
Breadth First Search

up
down
left
right



Breadth First Search

up
down
left
right



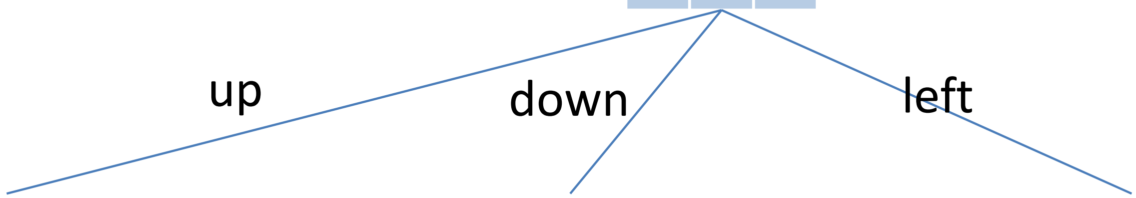
A* Algorithm (H1 - Number of incorrected placed pieces)

1	2	3
4	6	0
7	5	8

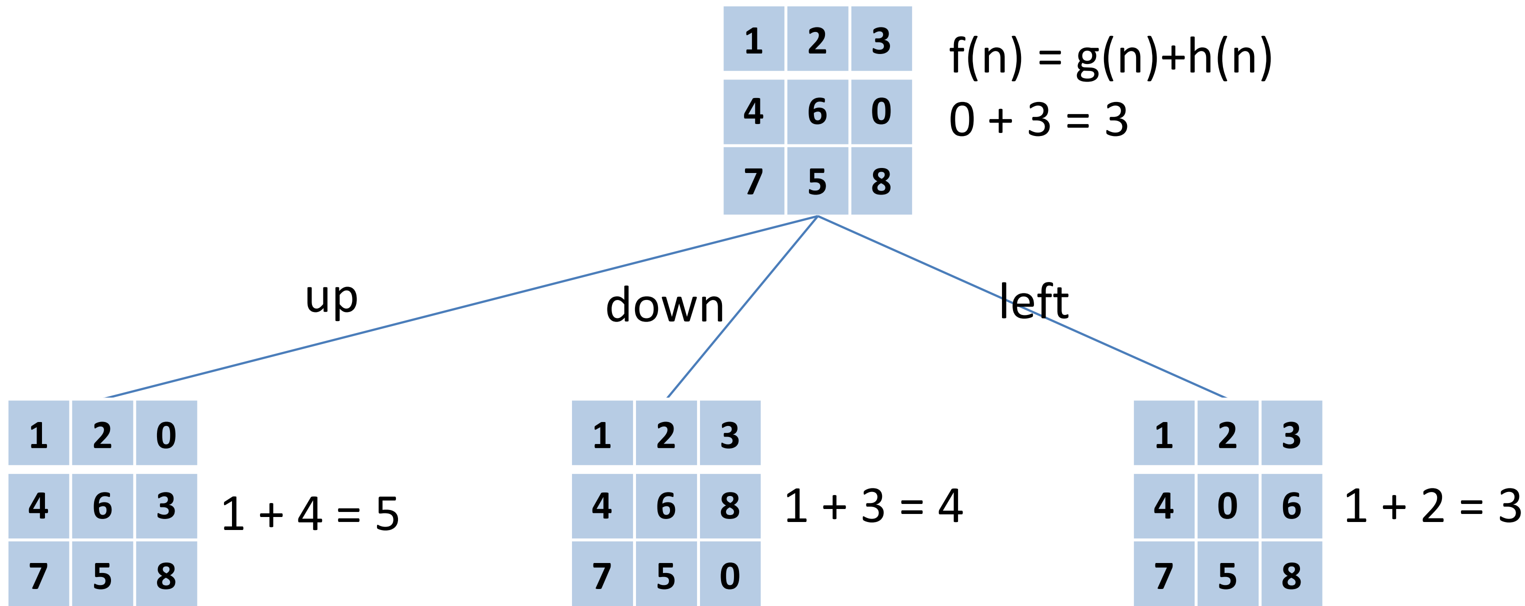
$f(n) = g(n)+h(n)$

$0 + 3 = 3$

up
down
left
right

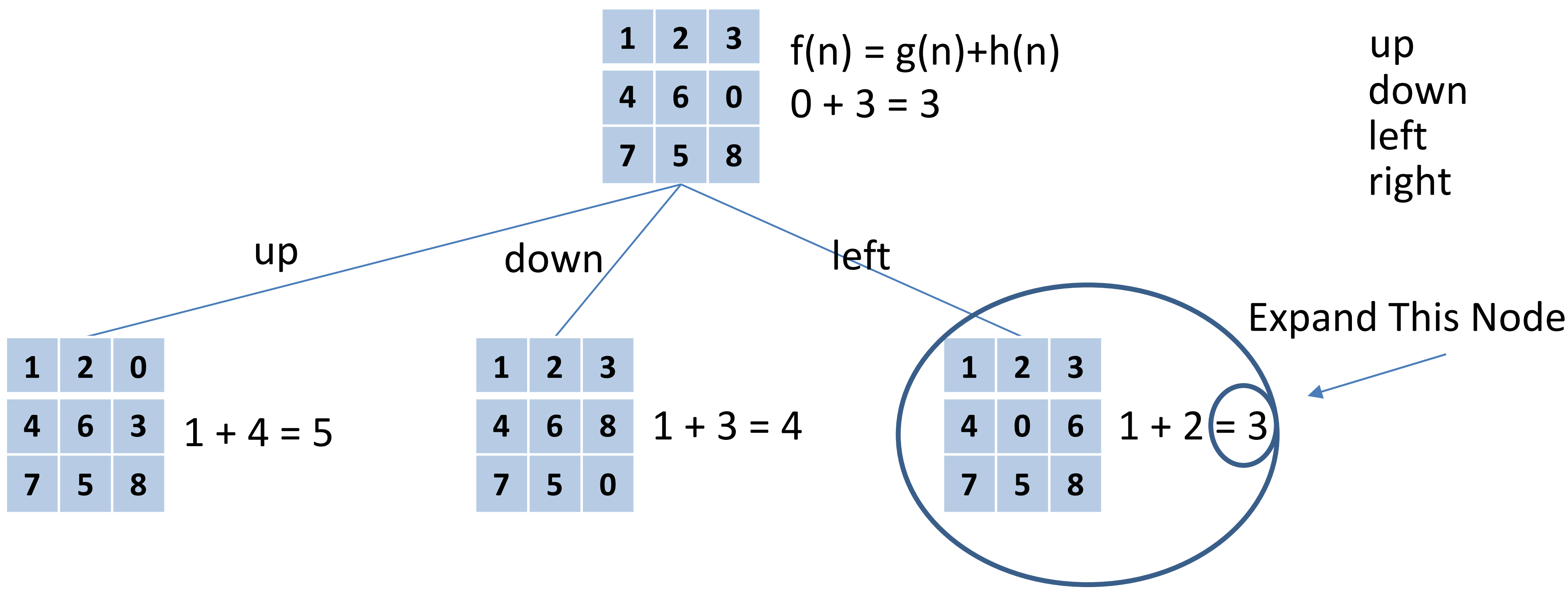


A* Algorithm (H1 - Number of incorrected placed pieces)

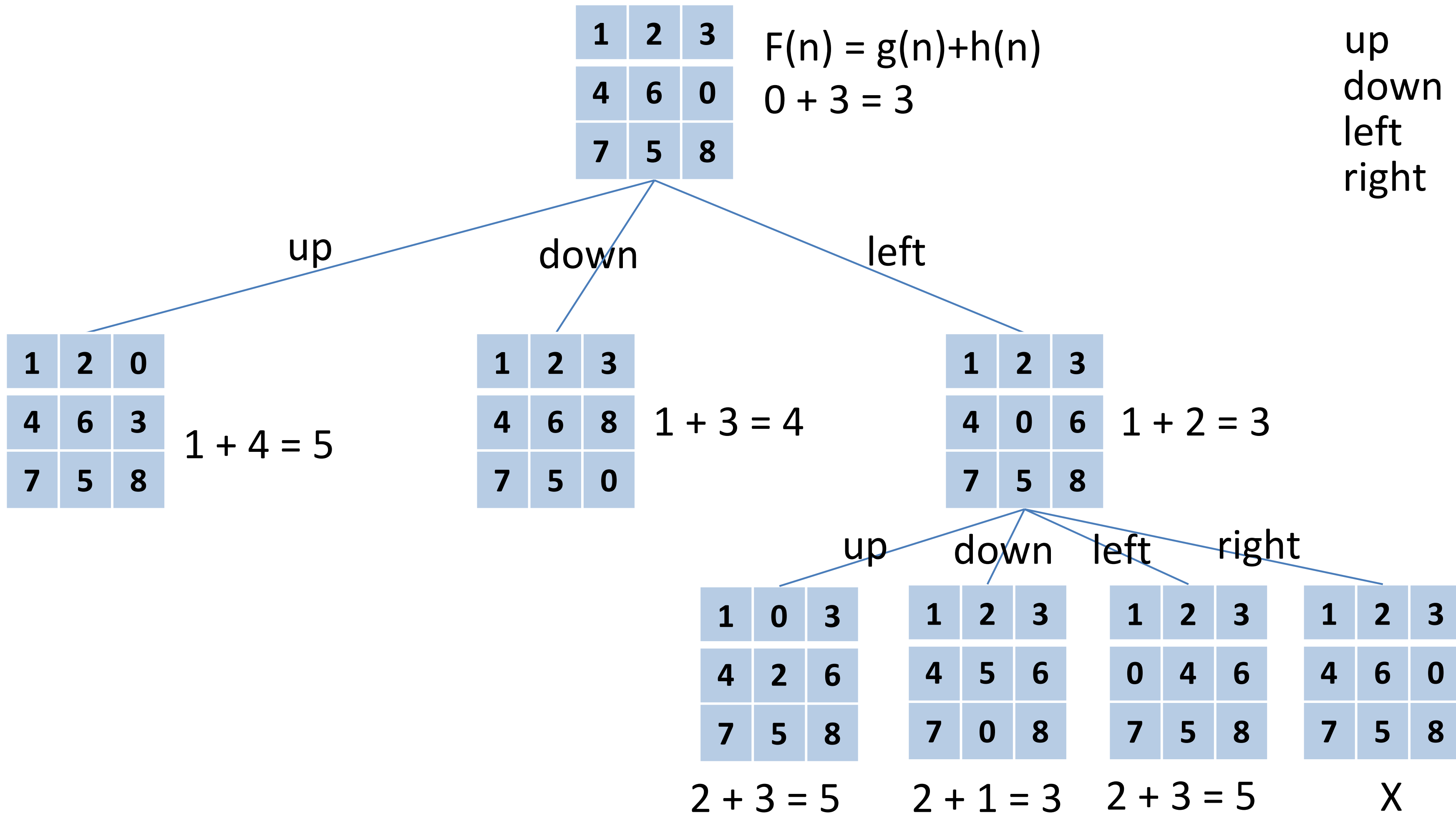


up
down
left
right

A* Algorithm (H1 - Number of incorrected placed pieces)

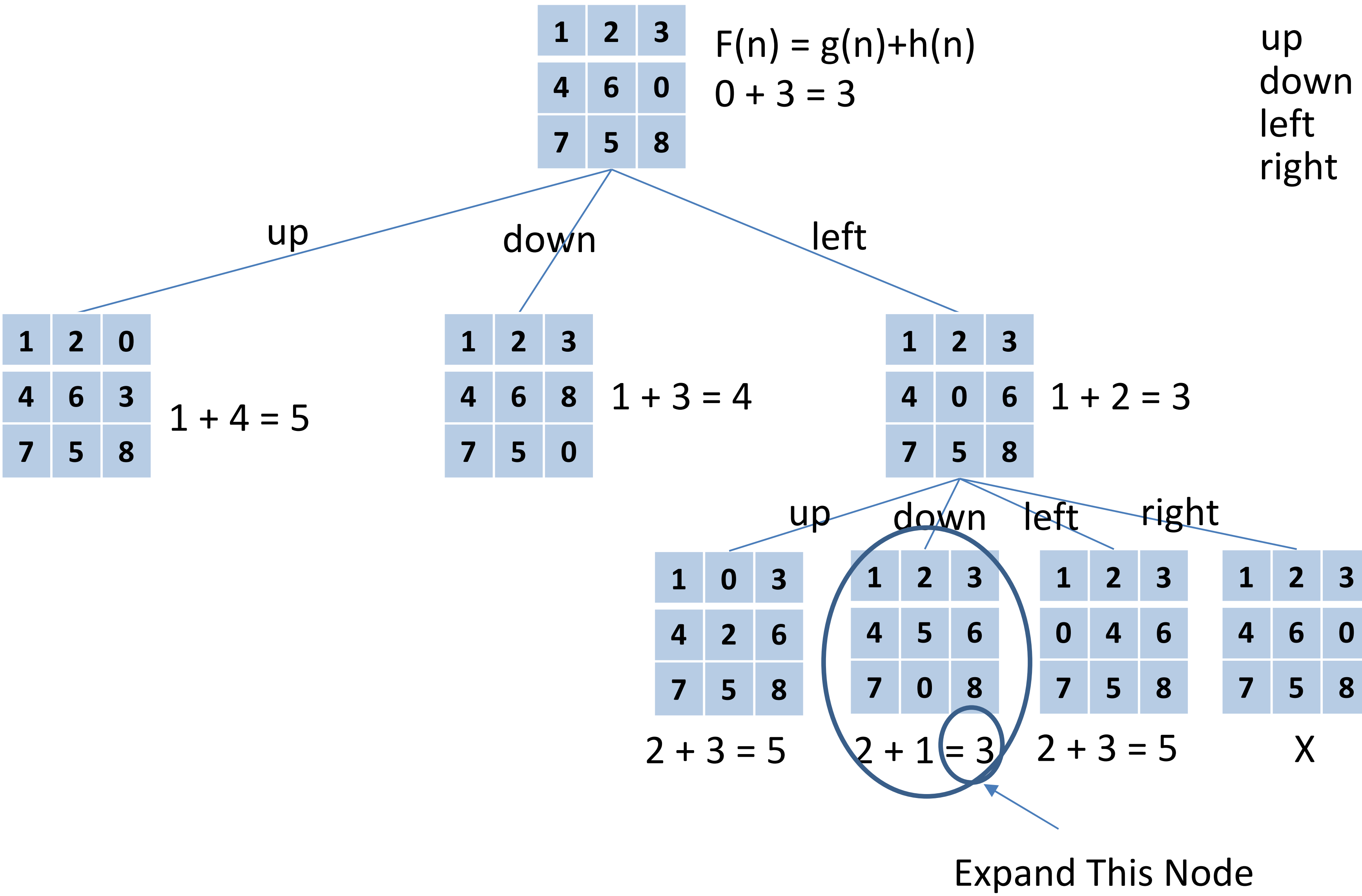


A* Algorithm (H1 - Number of incorrected placed pieces)

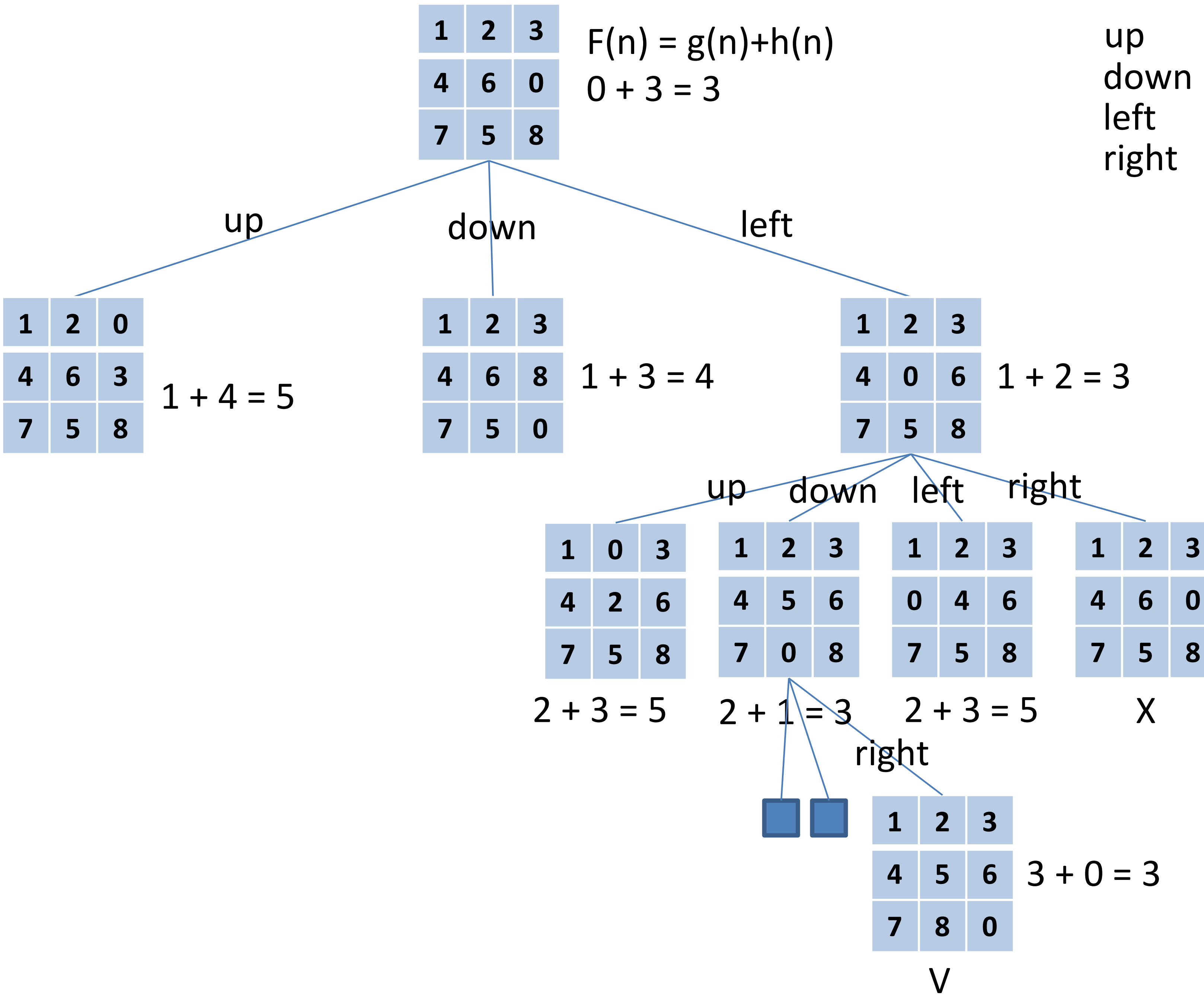


up
down
left
right

A* Algorithm (H1 - Number of incorrected placed pieces)



A* Algorithm (H1 - Number of incorrected placed pieces)



State Space

- **What is the State Space Size for the N-Puzzle:**
  - 3x3 Puzzle?
  - 4x4 Puzzle?
  - Generic Case: NxN Puzzle?

State Space

- **What is the State Space Size for the N-Puzzle:**
  - 3x3 Puzzle?

    =9*8*7*6*5*4*3*2*1 = 9!

  - 4x4 Puzzle?
  - Generic Case: NxN Puzzle?

State Space

- **What is the State Space Size for the N-Puzzle:**
  - 3x3 Puzzle?

    =9*8*7*6*5*4*3*2*1 = 9!

    or better = 9!/2

    since the state space is divided into two separate halves!

    (https://cs.stackexchange.com/questions/16515/reachable-state-space-of-an-8-puzzle)
  - 4x4 Puzzle?

    = 16!/2
  - Generic case: NxN Puzzle?

    =(N*N)!/2
  - Example: 8x8 Puzzle?

    =(8*8)!/2

State Space

- **What is the State Space Size for the N-Puzzle:**
  - 3x3 Puzzle?

    =9*8*7*6*5*4*3*2*1 = 9!

    or better = 9!/2 = 181440

    since the state space is divided into two separate halves!

    (https://cs.stackexchange.com/questions/16515/reachable-state-space-of-an-8-puzzle)
  - 4x4 Puzzle?

    = 16!/2 = $1.1*10^{13}$
  - Generic case: NxN Puzzle?

    =(N*N)!/2
  - Example: 8x8 Puzzle?

    =(8*8)!/2 = $6.3*10^{88}$

Solving the N Puzzle Problem

b) Implement code to solve this problem using the “breadth-first” strategy (in this case identical to "Uniform Cost").

c) Implement code to solve this problem using Greedy Search and using the A* Algorithm. Suppose the following heuristics for these methods:

- **H1 - Number of incorrected placed pieces;**
- **H2 - Sum of Manhattan distances from incorrected placed pieces to their correct places.**

d) Compare the results obtained concerning execution time and memory space occupied in solving the following problems using the previous methods:

Probl1	Probl2	Prob3	Prob4
1 2 3	1 3 6	1 6 2	5 1 3 4
5 0 6	5 2 0	5 7 3	2 0 7 8
4 7 8	4 7 8	0 4 8	10 6 11 12
			9 13 14 15

Information Structures

```
class SearchNode:
state: Matrix;                #matrix B and other info (Xs,Ys)
predecessor: SearchNode;      #Father of the node
operator: string;             #Operator used to generate state
numSteps: int;                #Depth
costFromStart: int;           #Cost to get to the node = depth
estimateCostToGoal: int;      #Heuristic
       . . .
```

Objective State Test

$$
\begin{array}{cc|ccc|c}
 & & & X & & \\
 & j & 1 & 2 & 3 & \\
 & i & & & & \\
 & 1 & 1 & 2 & 3 & \\
Y & 2 & 4 & 5 & 6 & N=3 \\
 & 3 & 7 & 8 & 0 & \\
\end{array}
$$

```
def objectiveTest(B: Matrix, N: int):
  for i in range(1, N)     #Para todas as linhas
    for j in range(1, N)   #Para todas as colunas
        if(B[j,i] !=0 and B[j,i]!=(i-1)*N+j)
          return False
  return True
```

(i-1)*N+j => Valor objetivo para a célula
na linha i, coluna j (B[j,i])

Operators Preconditions

X

i \ j	1	2	3
1	1	2	3
2	4	5	6
3	7	8	0

Y

N=3

```
def precond(B/(Xs,Ys): Matrix, Op: Oper):
  return (Op==up    and Ys>1 or
          Op==down  and Ys<N or
          Op==left  and Xs>1 or
          Op==right and Xs<N);
```

Operators Effects

```
def effects(B/(Xs,Ys): State, Op: Oper):
  if op==up:
    B[Xs,Ys] = B[Xs,Ys-1]
    B[Xs,Ys-1] = 0
    Ys = Ys-1
  if op==down:
    B[Xs,Ys] = B[Xs,Ys+1]
    B[Xs,Ys+1] = 0
    Ys = Ys+1
  if op==left:
    B[Xs,Ys] = B[Xs-1,Ys]
    B[Xs-1,Ys] = 0
    Xs = Xs-1
  if op==right:
    B[Xs,Ys] = B[Xs+1,Ys]
    B[Xs+1,Ys] = 0
    Xs = Xs+1
  return B/(Xs,Ys)
```

X

	i \ j	1	2	3	
	1	1	2	3	
Y	2	4	5	6	N=3
	3	7	8	0	

Heuristics Calculation

H1 - Number of incorrected placed pieces

X

i \ j	1	2	3
1	1	2	3
2	4	5	6
3	7	8	0

Y

N=3

```
def heuristic1(B: State, N: int):
  h1=0
  for i in range(1, N):
    for j in range(1, N):
        if B[j,i]!=0 and B[j,i]!=(i-1)*N+j):
          h1++
  return h1
```

Heuristics Calculation

H2 - Sum of Manhattan distances from incorrected placed pieces to their correct places

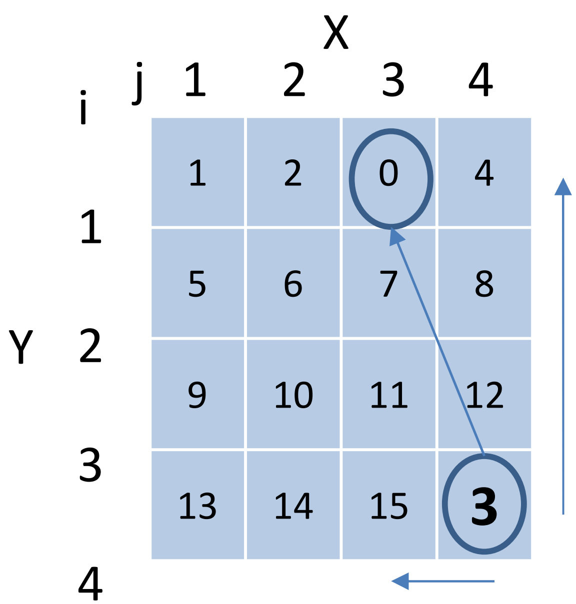
```
def heuristic2(B: State, N: int):
  h2=0
  for i in range(1, N):
    for j in range(1, N):
        if(B[j,i]!=0 and B[j,i]!=(i-1)*N+j):
          h2 += …
  return h2
}
```

X

	i \ j	1	2	3	
	1	**1**	**2**	**3**	
Y	2	**4**	**5**	**6**	N=3
	3	**7**	**8**	**0**	

Heuristics Calculation

H2 - Sum of Manhattan distances from incorrected placed pieces to their correct places



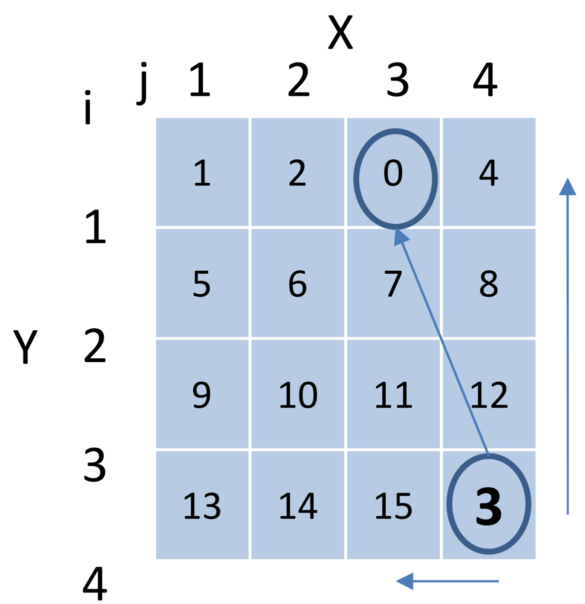
N=4

`X=j=4 ; Y=i=4 ; B[j,i]= B[4,4]=3`

Heuristics Calculation

H2 - Sum of Manhattan distances from incorrected placed pieces to their correct places

N=4

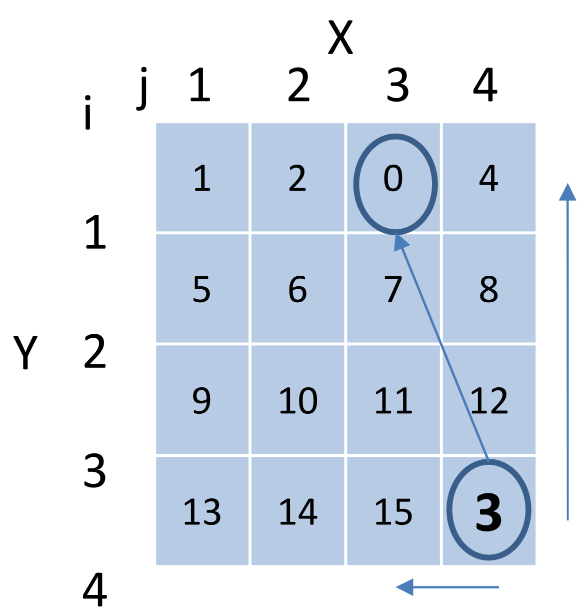


`X=j=4 ; Y=i=4 ; B[j,i]=3`

`Xcorr = (B[j,i]-1)%N+1 = (3-1)%4+1= 3`

Heuristics Calculation

H2 - Sum of Manhattan distances from incorrected placed pieces to their correct places



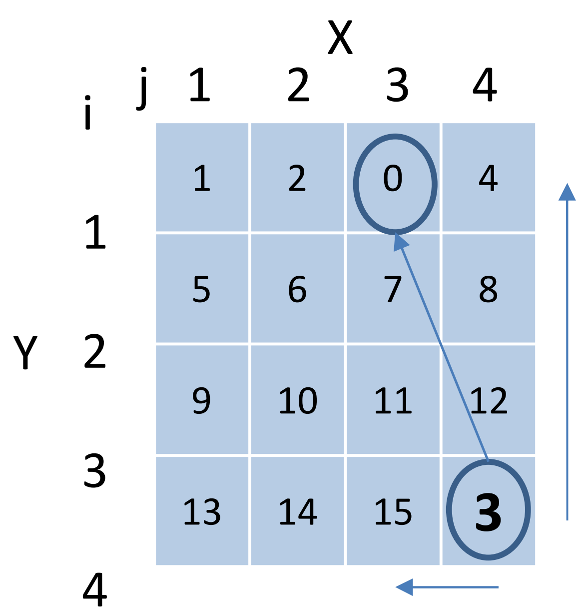
N=4

```
X=j=4 ; Y=i=4 ; B[j,i]=3
```

```
Xcorr = (B[j,i]-1)%N+1) = (3-1)%4+1)= 3
Ycorr = (B[j,i]+N-1)/N) = (3+4-1)/4 = 1
```

Heuristics Calculation

H2 - Sum of Manhattan distances from incorrected placed pieces to their correct places



N=4

```
X=j=4 ; Y=i=4  ;  B[j,i]=3

Xcorr = (B[j,i]-1)%N+1) = (3-1)%4+1)= 3
Ycorr = (B[j,i]+N-1)/N) = (3+4-1)/4 = 1

Man Distance = abs(X-Xcorr) + abs(Y-Ycorr) =
               abs(4-3) + abs(4-1) =
                1 + 3 = 4
```

Heuristics Calculation

H2 - Sum of Manhattan distances from incorrected placed pieces to their correct places

X

Y	i \ j	1	2	3
	1	1	2	3
	2	4	5	6
	3	7	8	0

N=3

```
def heuristic2(B: State, N: int):
  h2=0
  for i in range(1, N):
    for j in range(1, N):
        if(B[j,i]!=0 and B[j,i]!=(i-1)*N+j):
          h2+= abs(j-(B[j,i]-1)%N+1) +
               abs(i-(B[j,i]+N-1)/N)
  return h2
```

IART - Artificial Intelligence

Exercise 2: Solving Search Problems

Luís Paulo Reis

LIACC – Artificial Intelligence and Computer Science Lab.
DEI/FEUP – Informatics Engineering Department, Faculty of Engineering of the University of Porto, Portugal
APPIA – Portuguese Association for Artificial Intelligence