Implementation of the A-star Algorithm

Observations & Analysis

Sahil Jindal	Abhishek Gupta	Rohan Gyani	Mridul Ravi Jain
110020043	110040067	110040001	110040083

Assignment Specifications

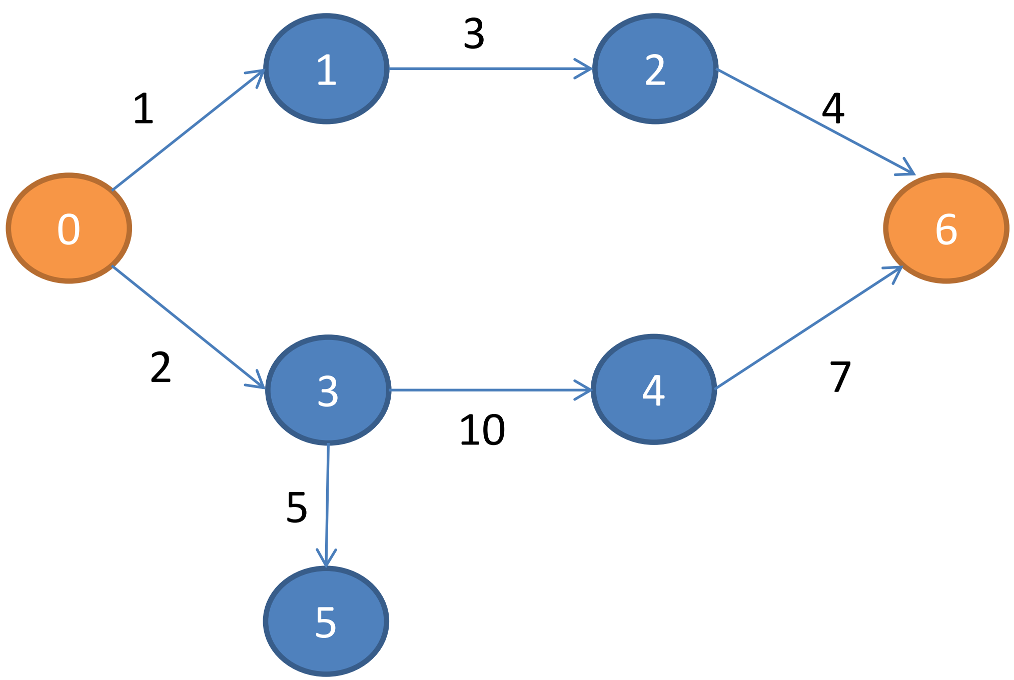
- Code A*; keep it general enough to be able to adapt to any search problem.
- Write modules for open and closed list management. Similarly for parent pointer redirection.
- Verify experimentally the intuition, "better heuristic performs better".
- Verify that "if $h(n)>h^*(n)$, for all n, A* may find the goal faster, but may discover a suboptimal path".
- Verify that monotone restriction is satisfied, parent pointer redirection for nodes on closed list is not needed.
- Come up with new heuristics for 8-puzzle and missionaries and cannibals; establish their admissibility and monotonicity or otherwise and measure performance.
- Carry out bidirectional A* search S-->G and G-->S.

A* Pseudo-Code

```
create the open list of nodes, initially containing only our starting node
create the closed list of nodes, initially empty

while (we have not reached our goal) {
        consider the best node in the open list (the node with the lowest f value)

        if (this node is the goal) {
                then we're done
        }
        else {
                move the current node to the closed list and consider all of its neighbors

                for (each neighbor) {
                        if (this neighbor is in the closed list and our current g value is lower) {
                                update the neighbor with the new, lower, g value
                                change the neighbor's parent to our current node
                        }
                        else if (this neighbor is in the open list and our current g value is lower) {
                                update the neighbor with the new, lower, g value
                                change the neighbor's parent to our current node
                        }
                        else this neighbor is not in either the open or closed list {
                                add the neighbor to the open list and set its g value
                        }
                }
        }
}
```

A-star Algorithm On a Simple Graph

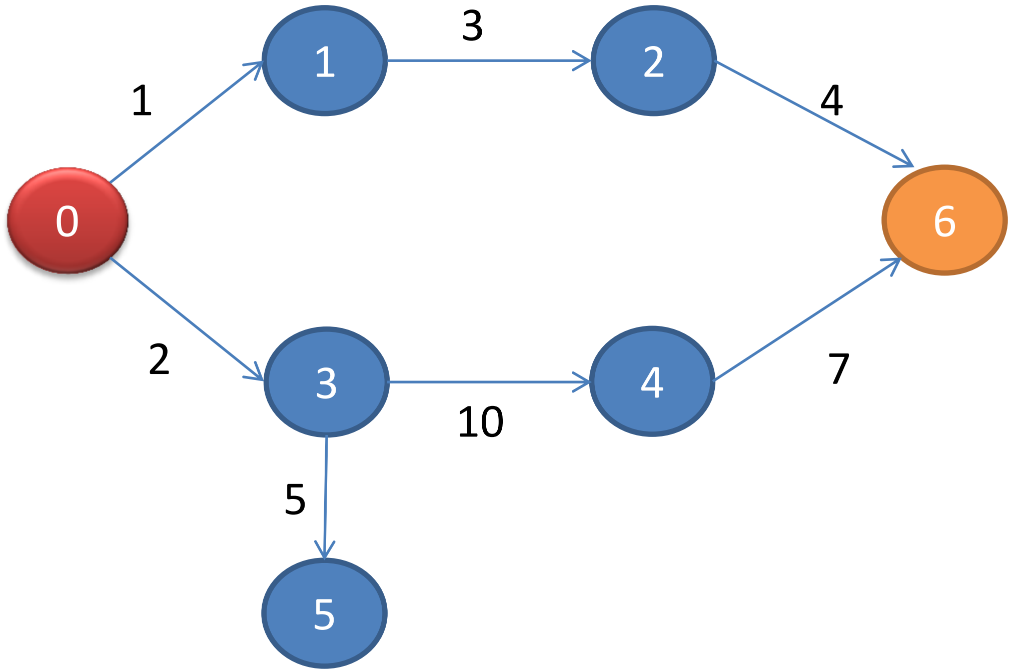
- **Without parent pointer redirection**



Start Node: 0
Goal Node: 6
h(i)= 0 for i=1,2…7

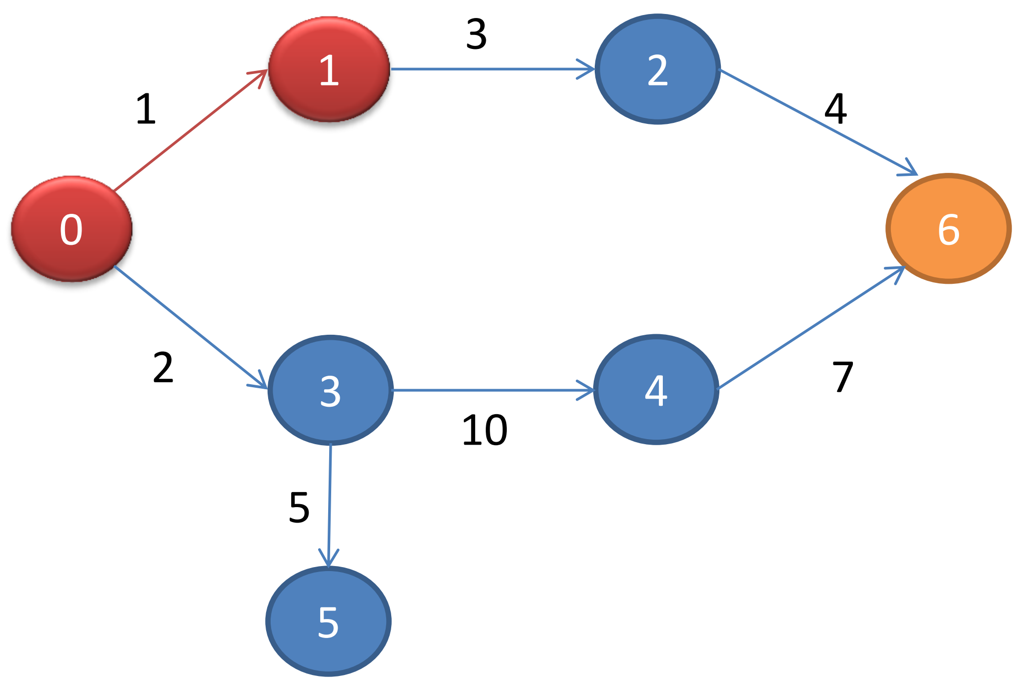
A-star Algorithm On a Simple Graph

- **Node picked by algorithm from open list:**
  - Iteration 1: Node 0



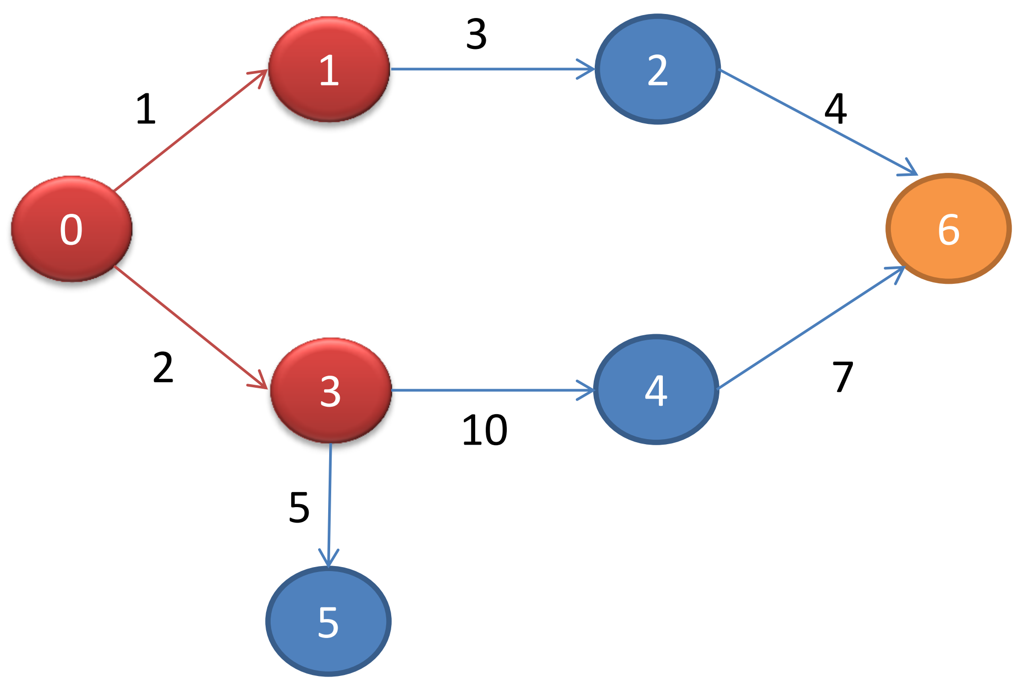
A-star Algorithm On a Simple Graph

- **Node picked by algorithm from open list:**
  - Iteration 1: Node 0
  - Iteration 2 : Node 1



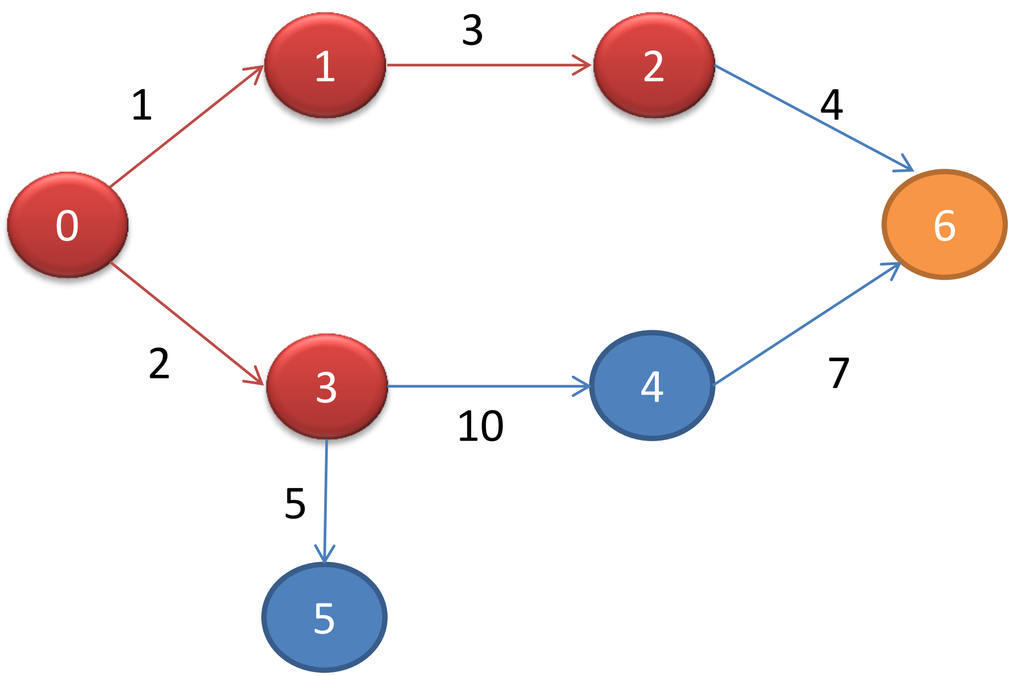
A-star Algorithm On a Simple Graph

- **Node picked by algorithm from open list:**
  - Iteration 1: Node 0
  - Iteration 2 : Node 1
  - Iteration 3 : Node 3



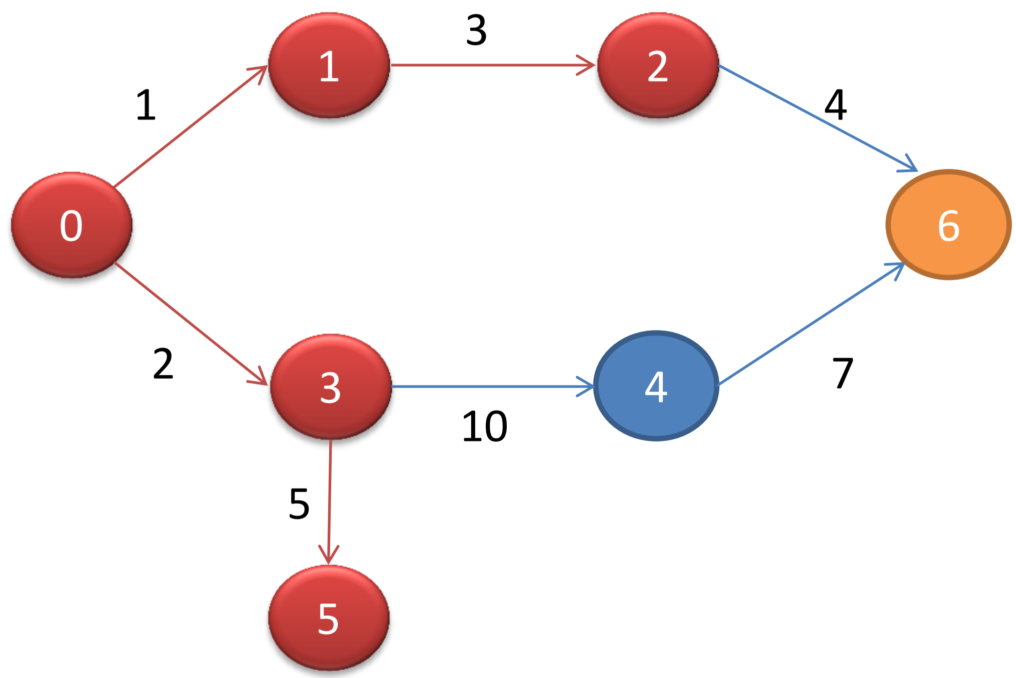
A-star Algorithm On a Simple Graph

- **Node picked by algorithm from open list:**
  - Iteration 1: Node 0
  - Iteration 2 : Node 1
  - Iteration 3 : Node 3
  - Iteration 4 : Node 2



A-star Algorithm On a Simple Graph

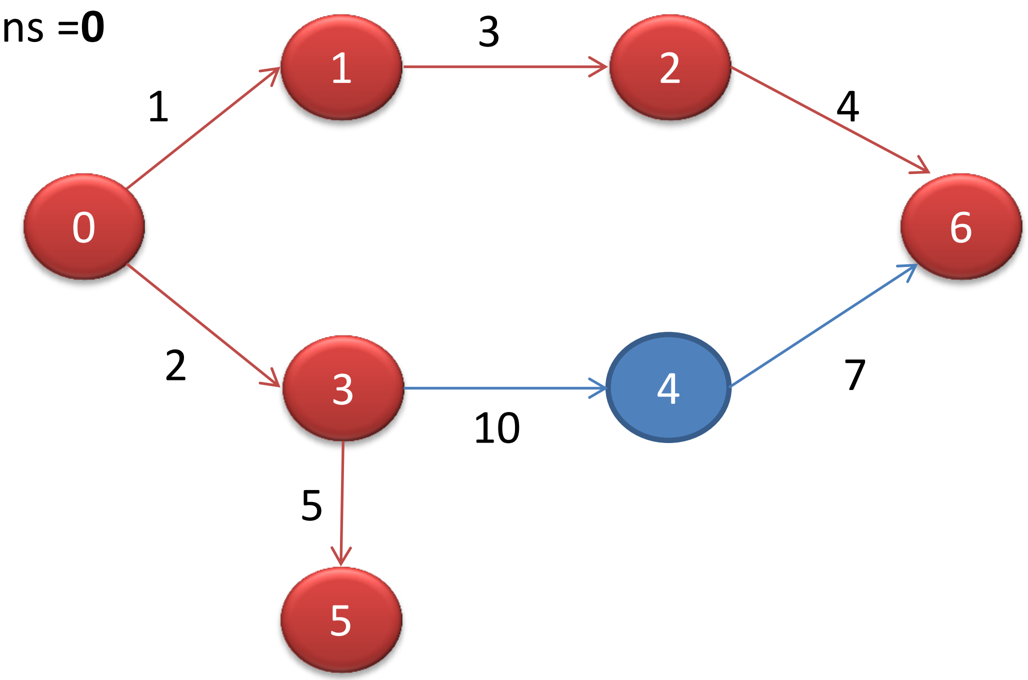
- **Node picked by algorithm from open list:**
  - Iteration 1: Node 0
  - Iteration 2 : Node 1
  - Iteration 3 : Node 3
  - Iteration 4 : Node 2
  - Iteration 5 : Node 5



A-star Algorithm On a Simple Graph

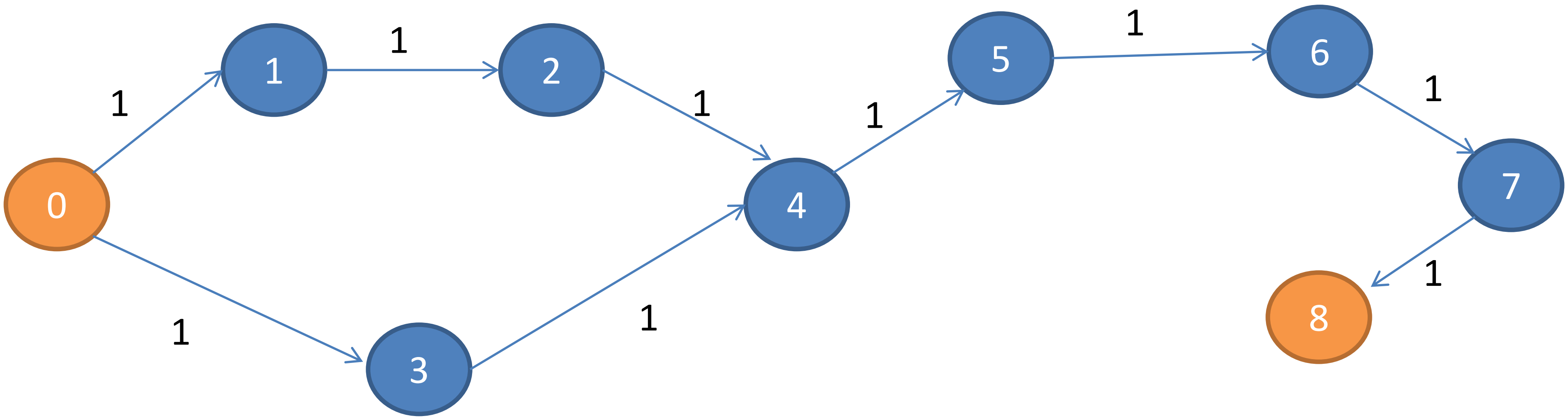
Final Output

- The optimal path is: **0 1 2 6**
- Optimal path cost is **8**
- Number of iterations taken by the A* Algo =**5**
- Number of Parent Pointer Redirections =**0**



Parent Pointer Redirection

- **We shall use the following graph & heuristic, to see a case where parent pointer redirection takes place.**

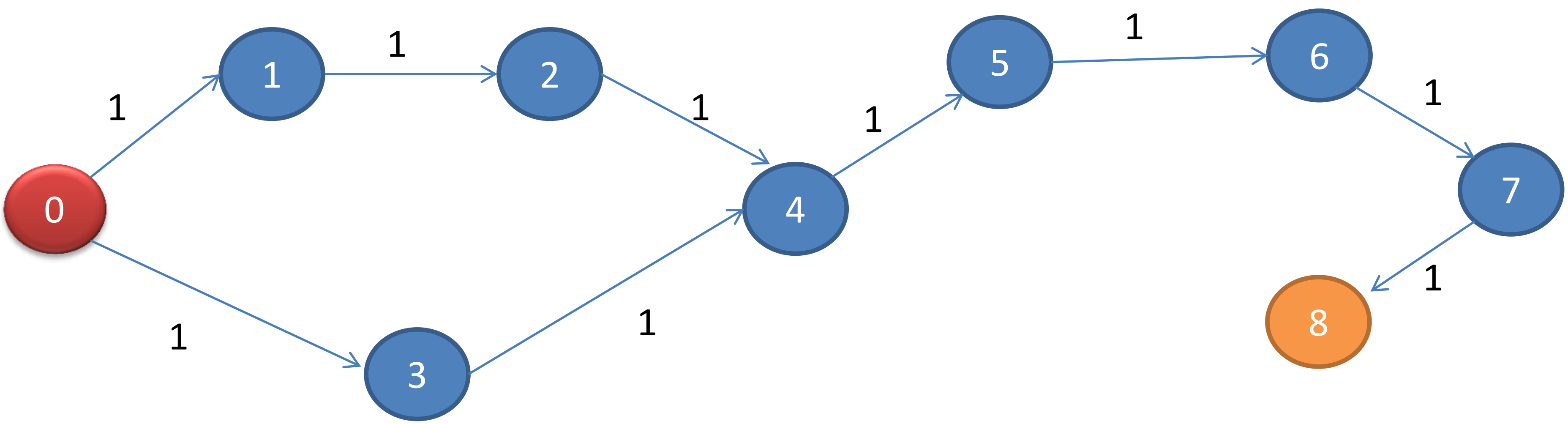


Start Node: 0 Goal Node: 8

$$h(i) = \begin{cases} 5 & \text{for } i=3 \\ 0 & \text{otherwise} \end{cases}$$

Parent Pointer Redirection

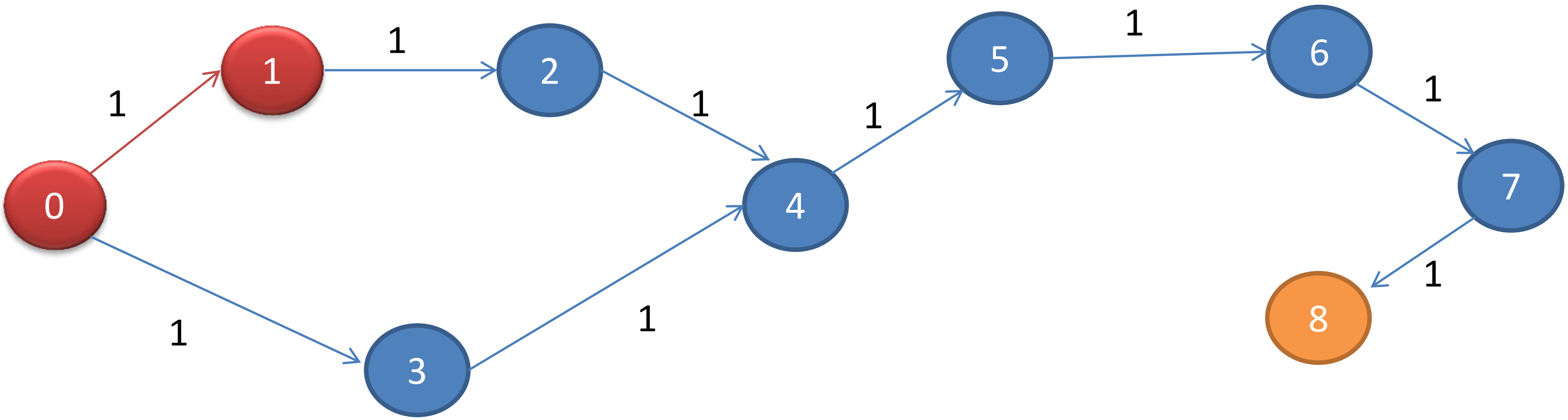
- **Iteration 1:**



The node chosen to be expanded from the open list: 0

Parent Pointer Redirection

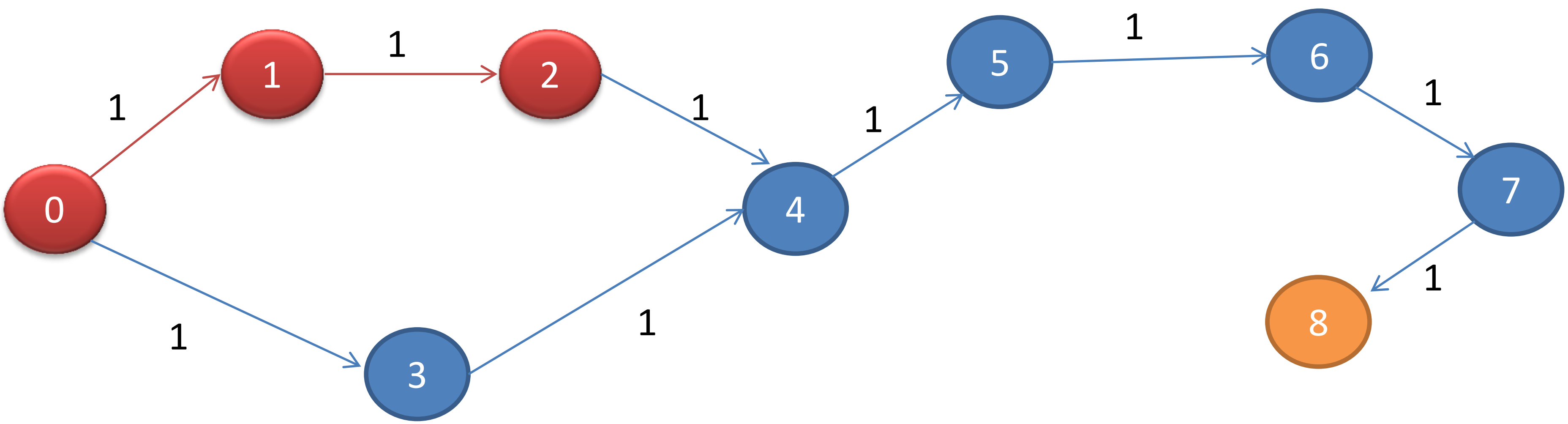
- **Iteration 2:**



The node chosen to be expanded from the open list: 1

Parent Pointer Redirection

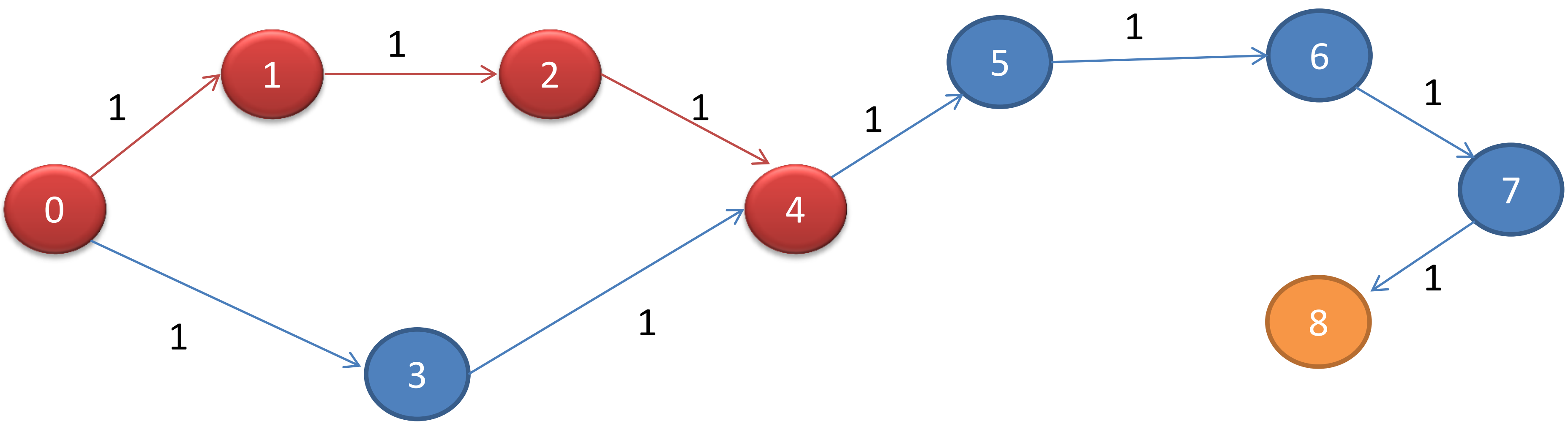
- **Iteration 3:**



The node chosen to be expanded from the open list: 2

Parent Pointer Redirection

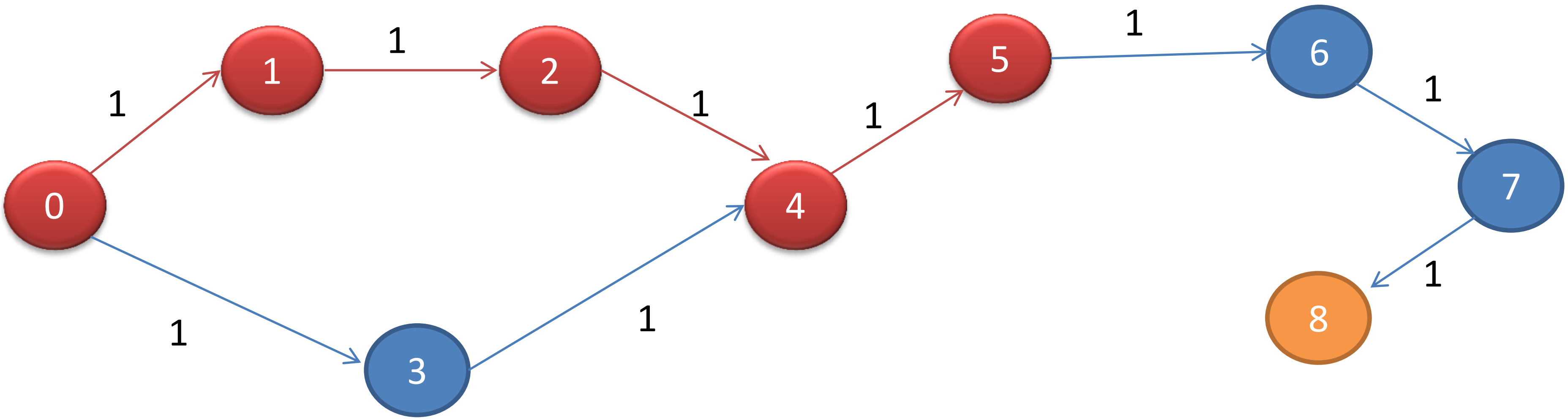
- **Iteration 4:**



The node chosen to be expanded from the open list: 4

Parent Pointer Redirection

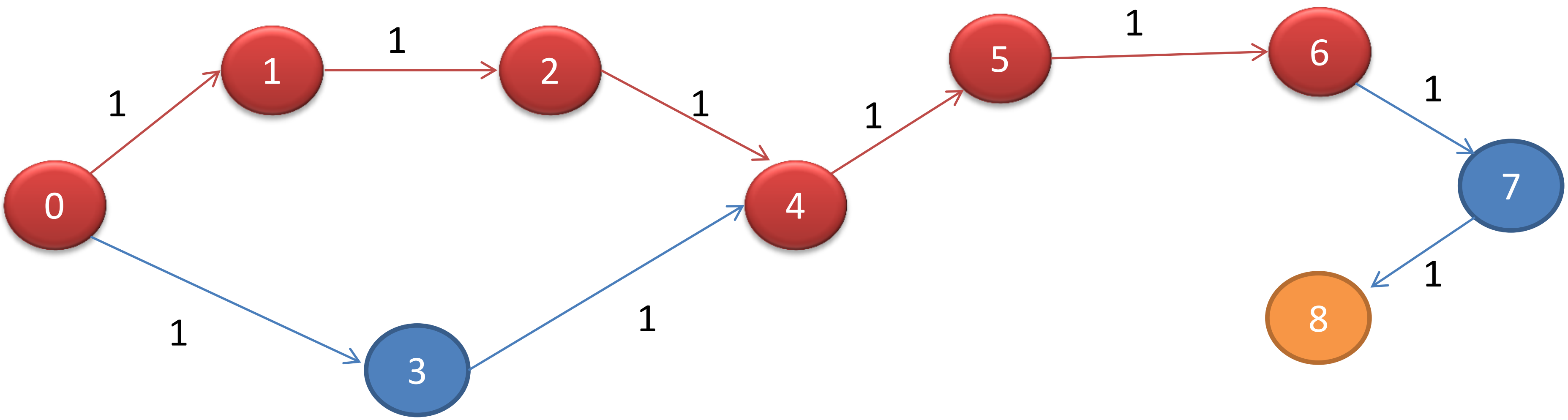
- **Iteration 5:**



The node chosen to be expanded from the open list: 5

Parent Pointer Redirection

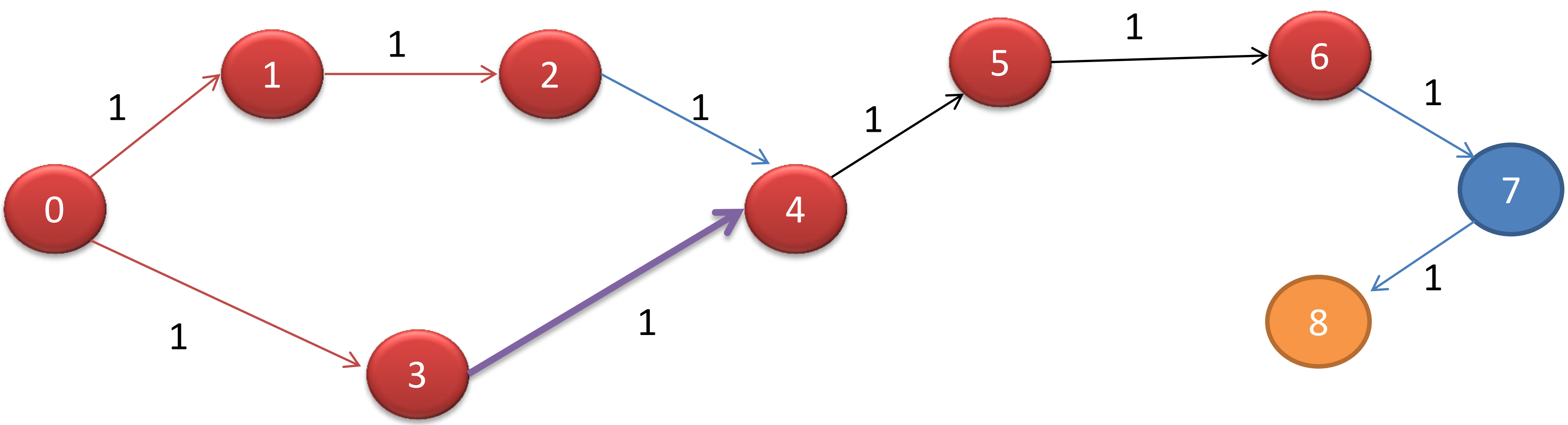
- **Iteration 6:**



The node chosen to be expanded from the open list: 6

Parent Pointer Redirection

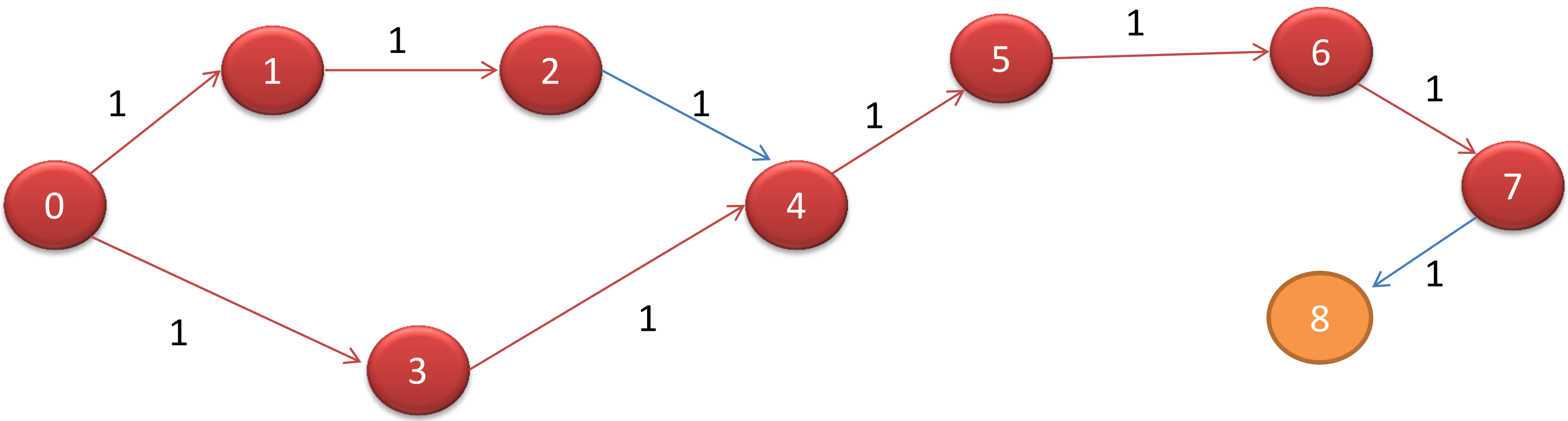
- **Iteration 7:**



The node chosen to be expanded from the open list: 3
1 Parent Pointers Redirected: 3 → 4
g values changed for 3 nodes: 4 5 6

Parent Pointer Redirection

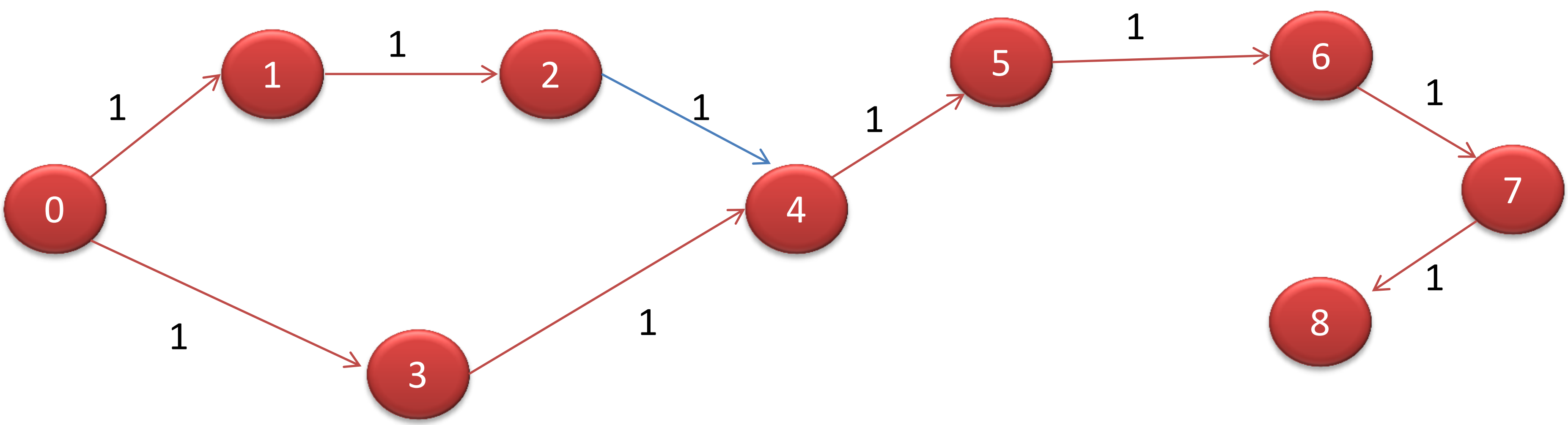
- **Iteration 8:**



The node chosen to be expanded from the open list: 7

Parent Pointer Redirection

- **Final Output:**



- The optimal path is:0 3 4 5 6 7 8
- Optimal path cost is 6
- Number of iterations taken by the A* Algo =8
- Number of Parent Pointer Redirections =3

Better Heuristic Performs Better

- We now change the heuristic of previous case as follows:
  - $h(0) = 6$
  - $h(1) = 6$
  - $h(2) = 5$
  - $h(3) = 5$
  - $h(4) = 4$

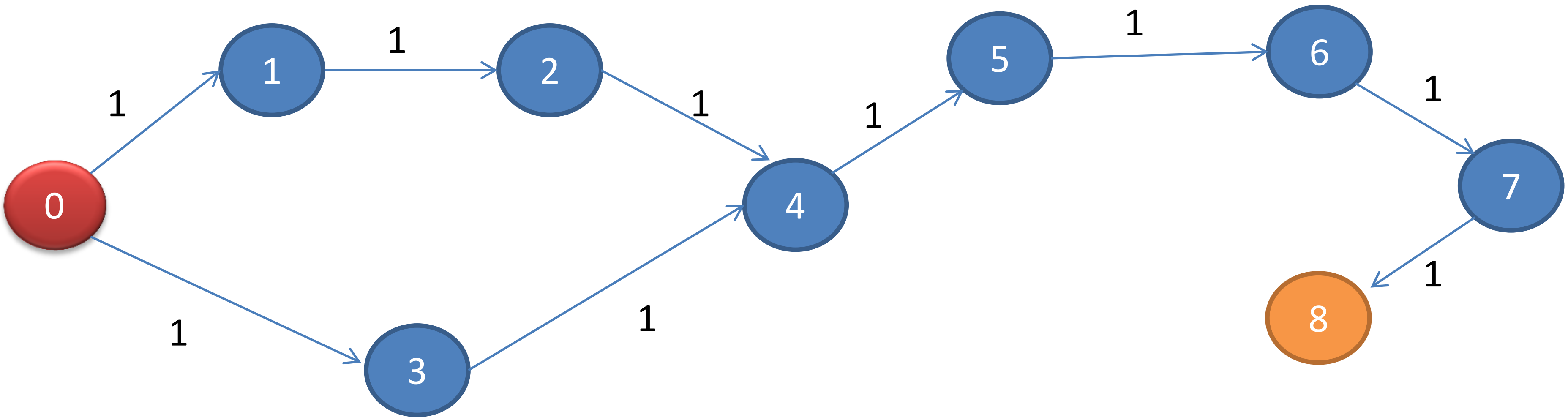
  $h(5) = 3$
  $h(6) = 2$
  $h(7) = 1$
  $h(8) = 0$

- Clearly above heuristic is better than the previous heuristic which was

$$h(i) = \begin{cases} 5 & \text{for } i=3 \\ 0 & \text{otherwise} \end{cases}$$

Better Heuristic Performs Better

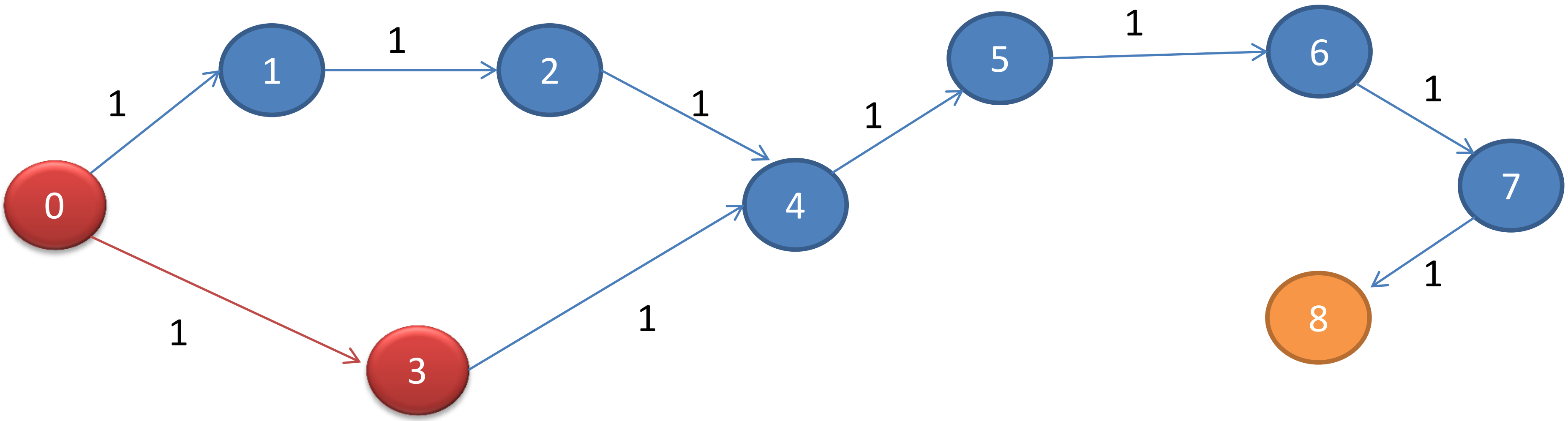
- **Iteration 1:**



The node chosen to be expanded from the open list: 0

Better Heuristic Performs Better

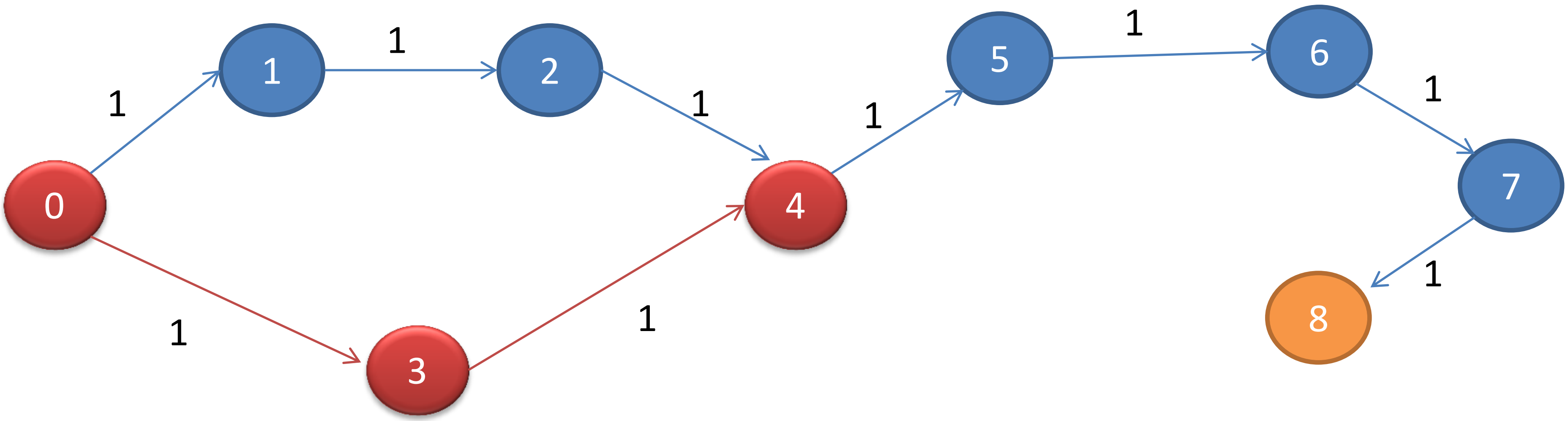
- **Iteration 2:**



The node chosen to be expanded from the open list: 3

Better Heuristic Performs Better

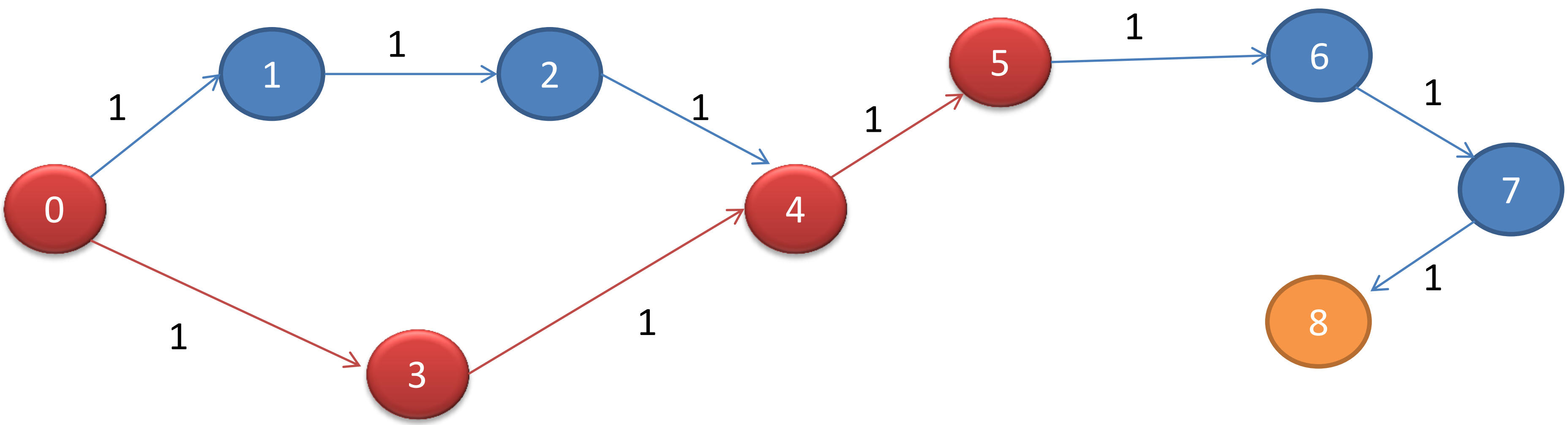
- **Iteration 3:**



The node chosen to be expanded from the open list: 4

Better Heuristic Performs Better

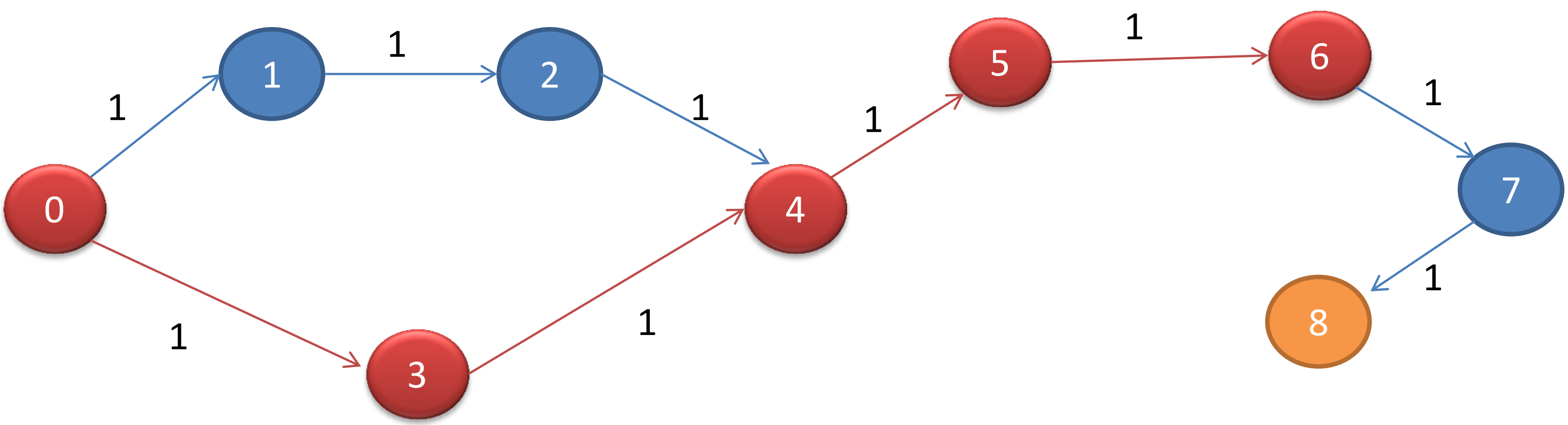
- **Iteration 4:**



The node chosen to be expanded from the open list: 5

Better Heuristic Performs Better

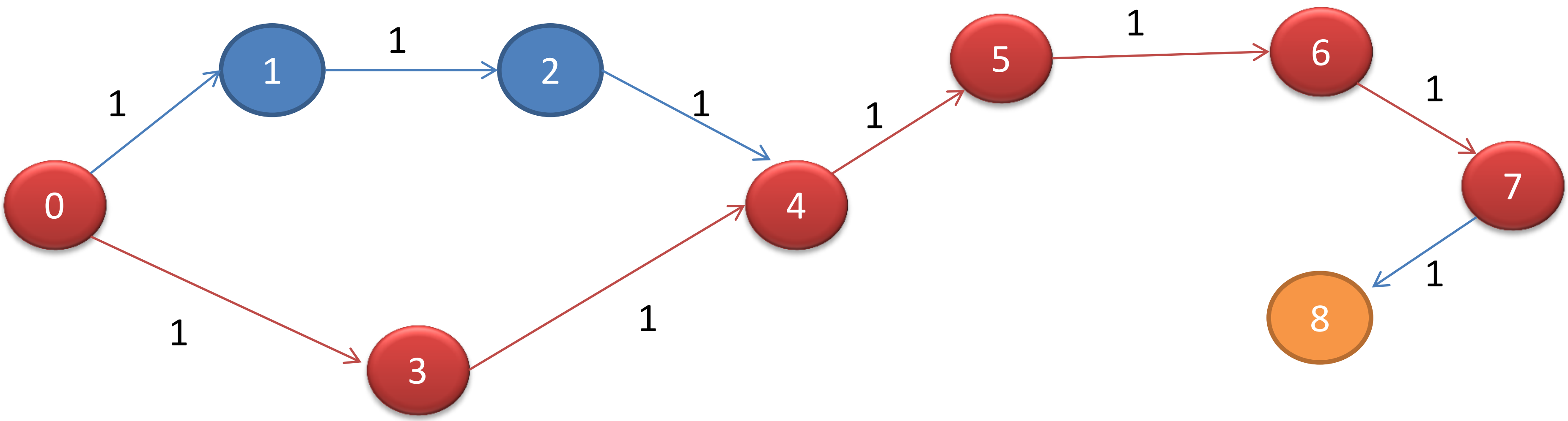
- **Iteration 5:**



The node chosen to be expanded from the open list: 6

Better Heuristic Performs Better

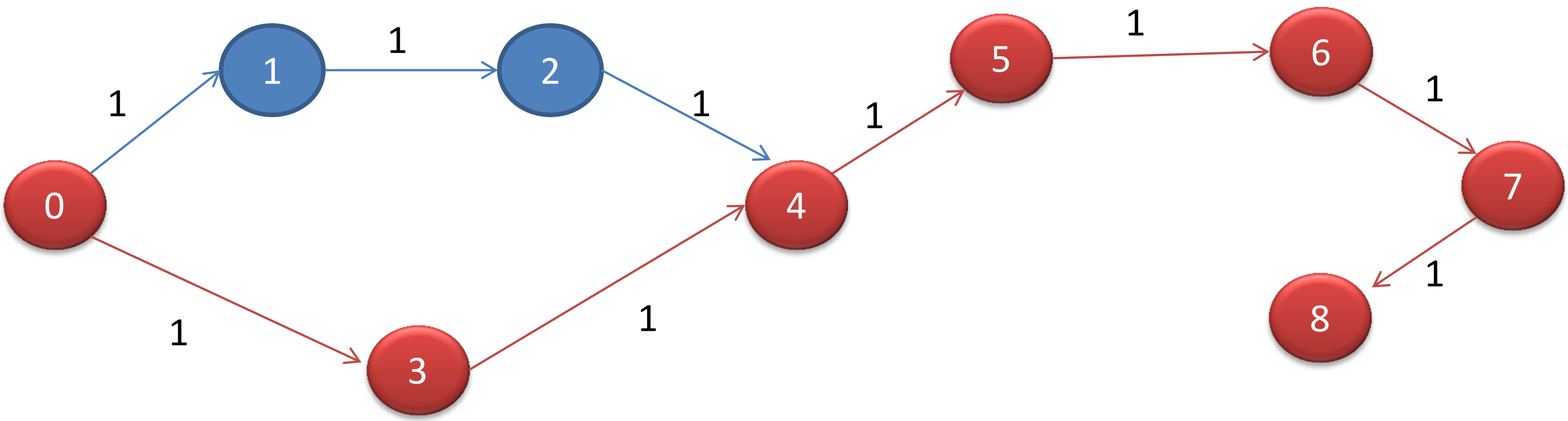
- **Iteration 6:**



The node chosen to be expanded from the open list: 7

Better Heuristic Performs Better

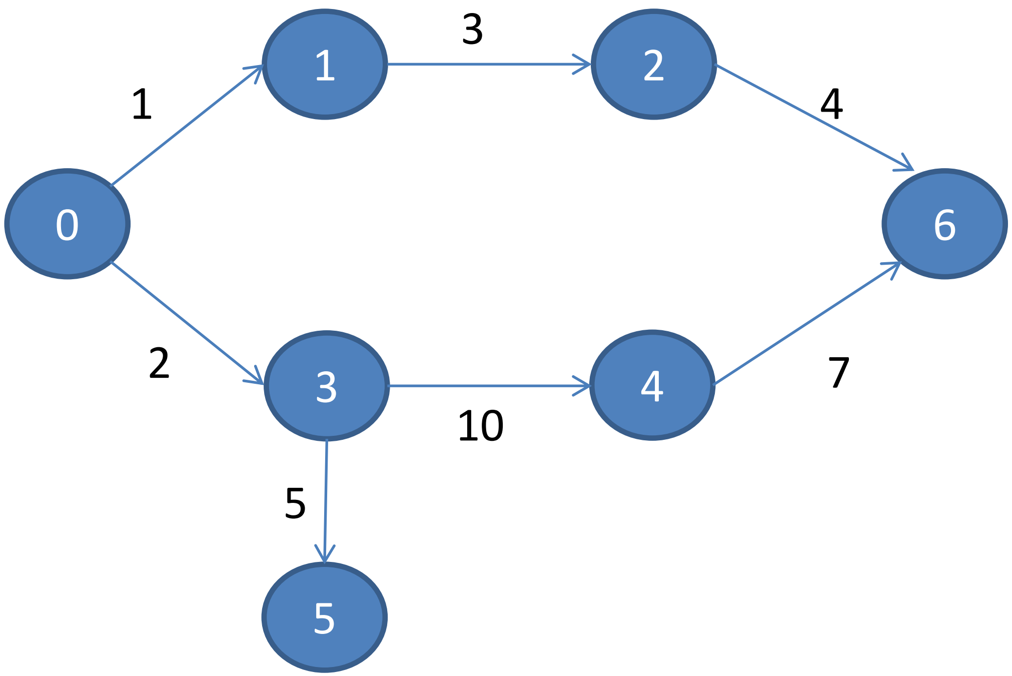
- **Final Output:**



- The optimal path **is:0 3 4 5 6 7 8**
- Optimal path cost is **6**
- Number of iterations taken by the A* Algo =**6(as opposed to 8 previously)**
- Number of Parent Pointer Redirections =**0**
- **So we see that with a better heuristic A* algorithms converges faster**

h greater than h*

- We now verify that "if h(n)>h*(n)", for all n, A* may find the goal faster, but may discover a suboptimal path.
- We consider the graph used initially and run the A* algorithm on it using a heuristic such that h > h*(n) for all n



Start Node: 0

Goal Node: 6

h(1)= 20

h(1)= 30

h(1)= 40

h(1)= 10

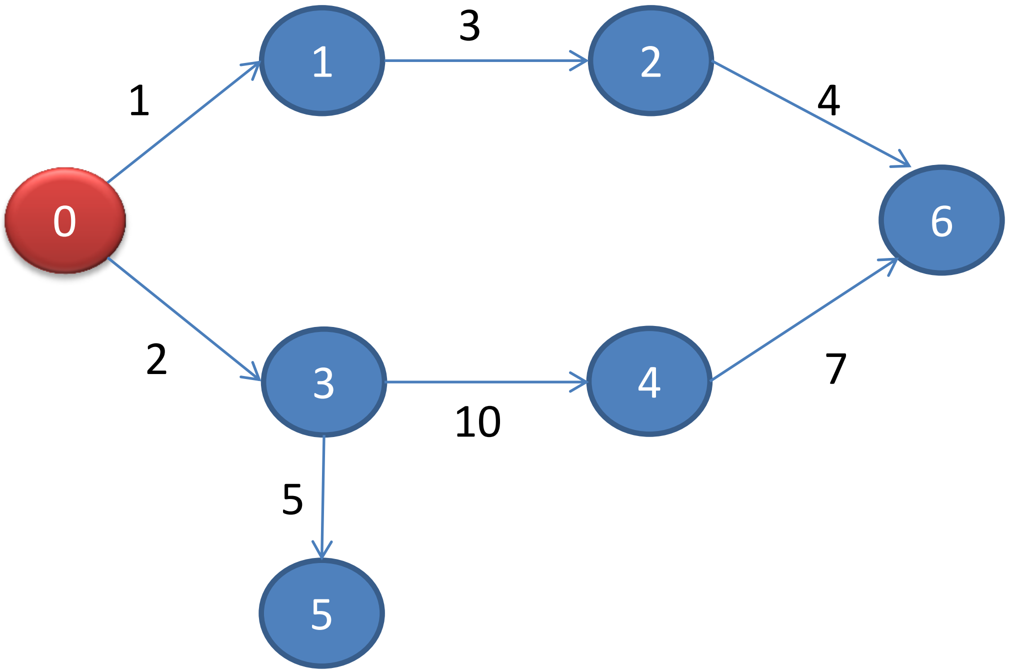
h(1)= 10

h(1)= 10

h(1)= 0

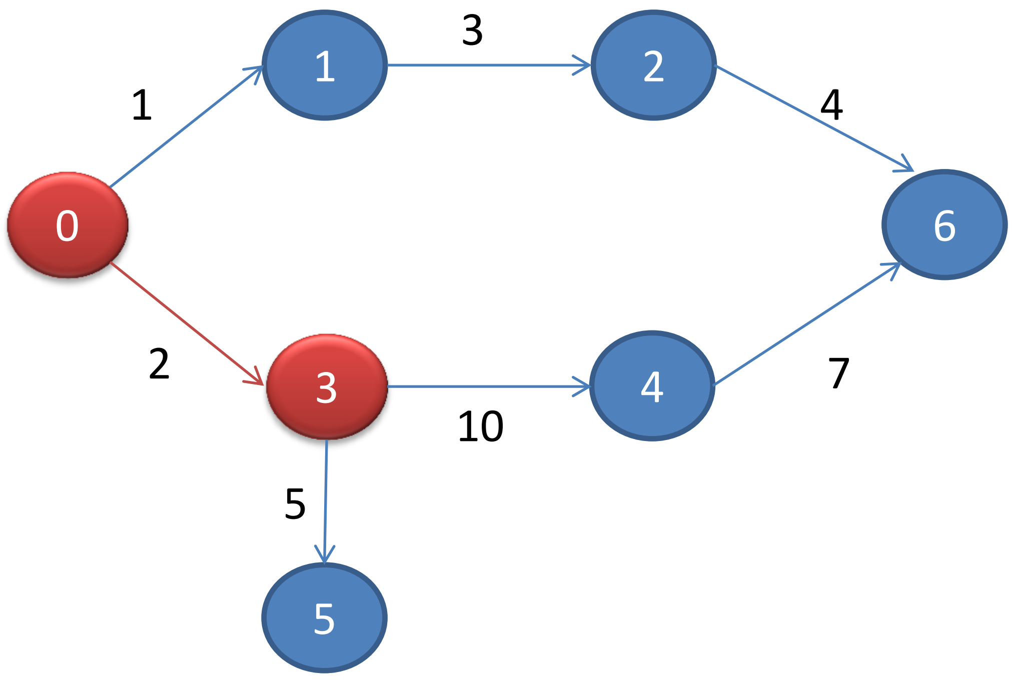
h greater than h*

- **Node picked by algorithm from open list:**
  - Iteration 1: Node 0



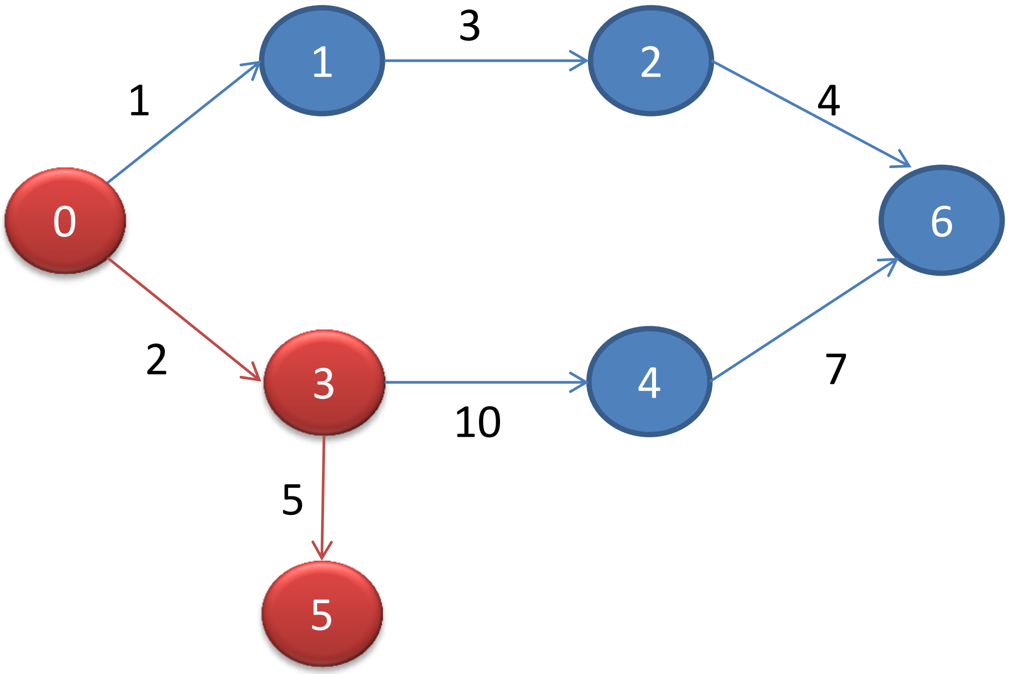
h greater than h*

- **Node picked by algorithm from open list:**
  - Iteration 1: Node 0
  - Iteration 2: Node 3



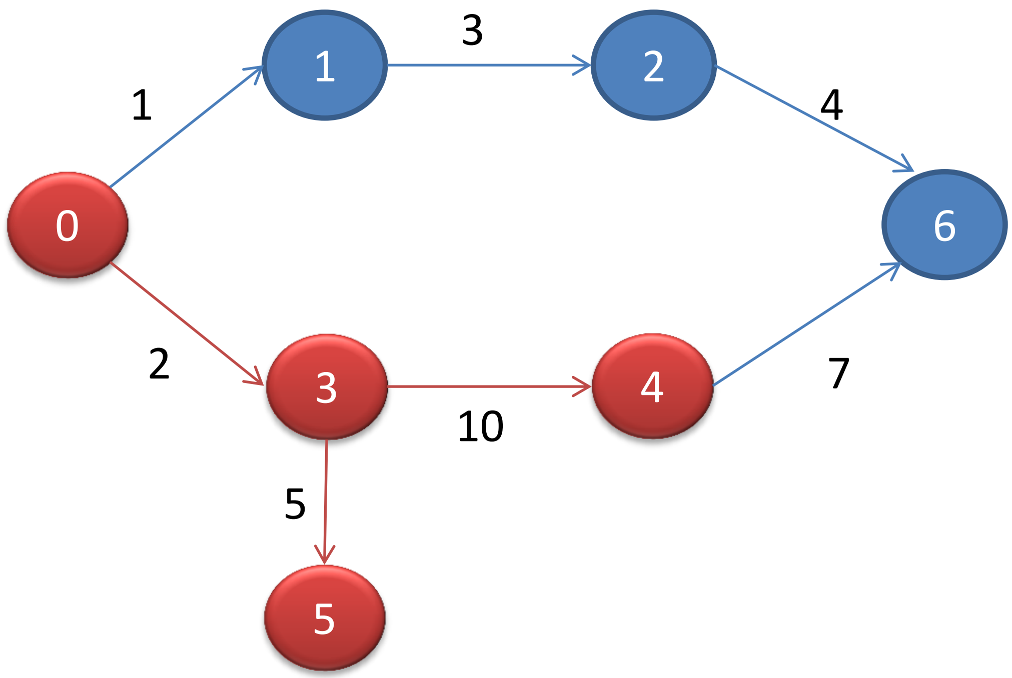
h greater than h*

- **Node picked by algorithm from open list:**
  - Iteration 1: Node 0
  - Iteration 2: Node 3
  - Iteration 3: Node 5



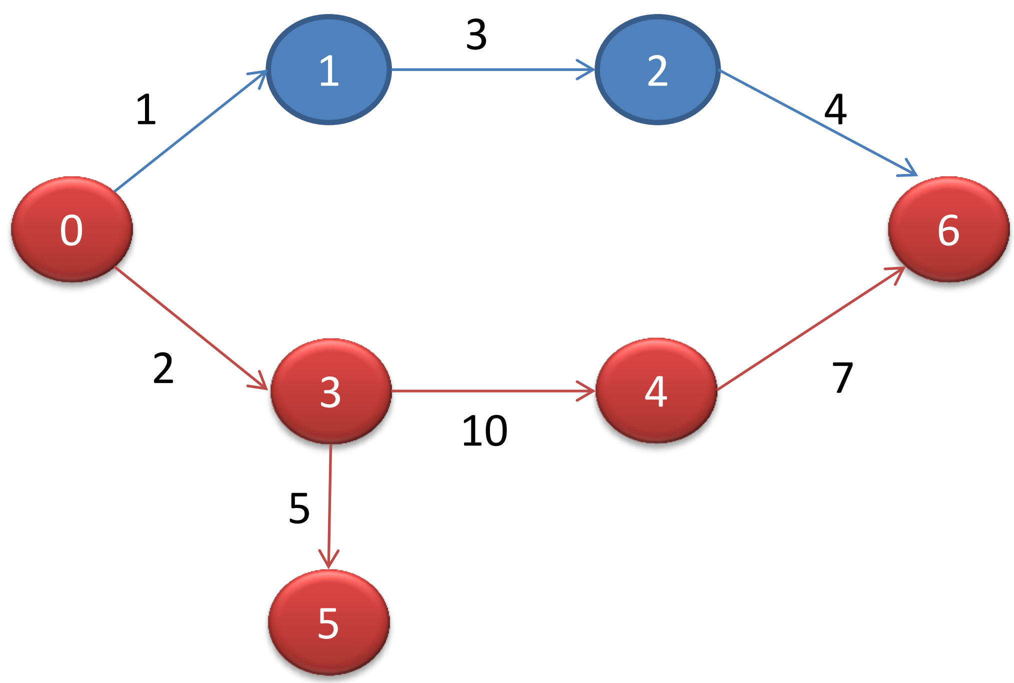
h greater than h*

- **Node picked by algorithm from open list:**
  - Iteration 1: Node 0
  - Iteration 2: Node 3
  - Iteration 3: Node 5
  - Iteration 4: Node 4



h greater than h*

- **Final Output:**
  - Number of iterations taken by the A* Algo =4
  - Number of Parent Pointer Redirections =0
  - Found path is:0 3 4 6
  - Found path cost is 19
  - The discovered path is a suboptimal path.



Monotone Restriction

- We now change the heuristic of second case as follows:
  - h(0) = 6
  - h(1) = 6
  - h(2) = 5
  - h(3) = 5
  - h(4) = 4

  h(5) = 3

  h(6) = 2

  h(7) = 1

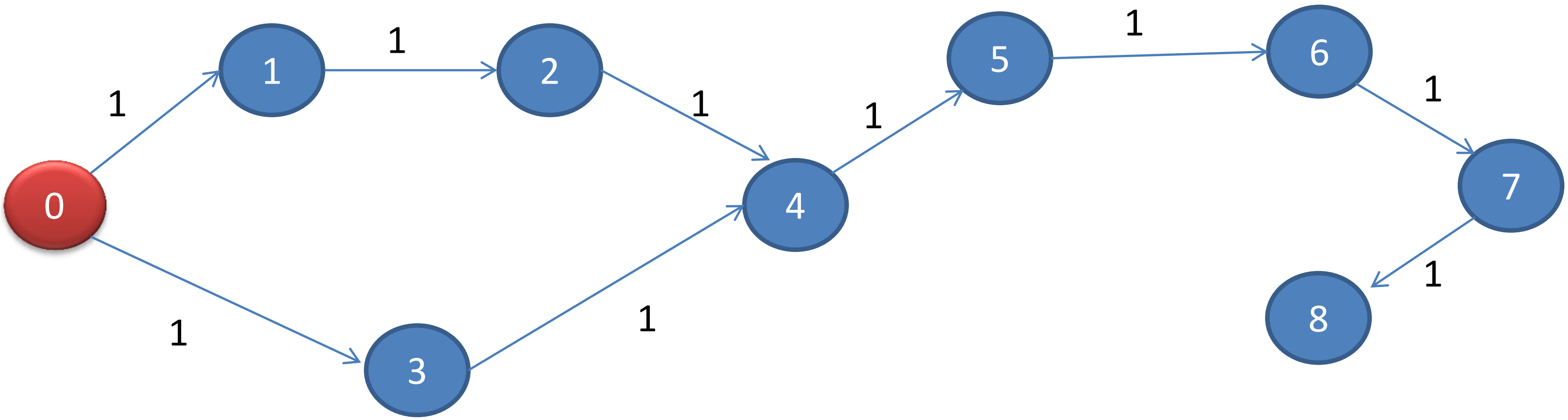
  h(8) = 0

- Note that in the above case MR is satisfied i.e.

  h(parent) <= h(child) + C(parent, child)

Monotone Restriction

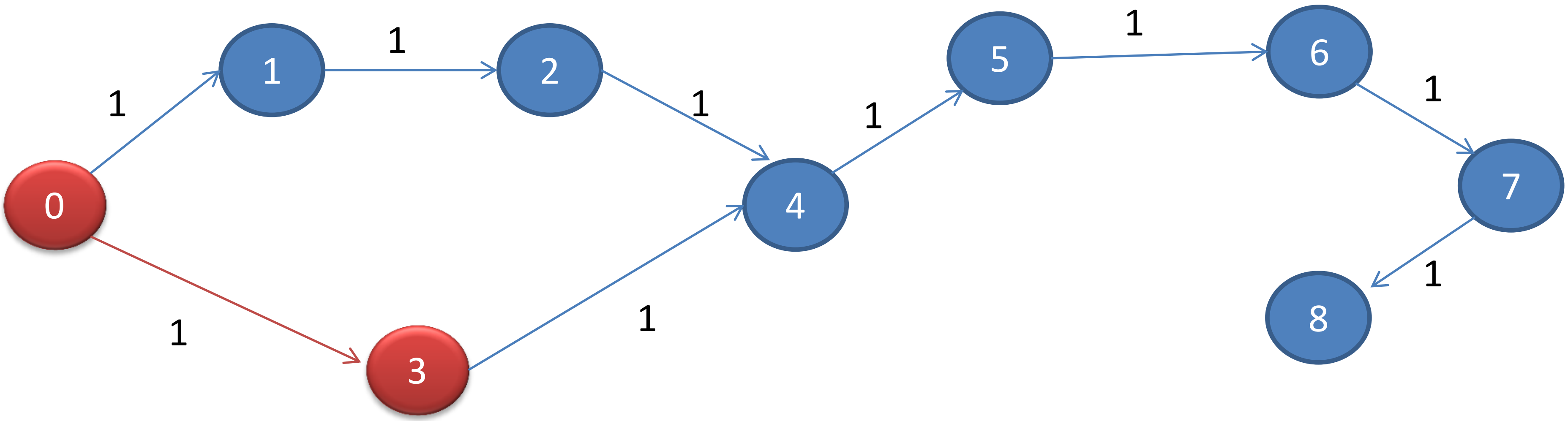
- **Iteration 1:**



The node chosen to be expanded from the open list: 0

Monotone Restriction

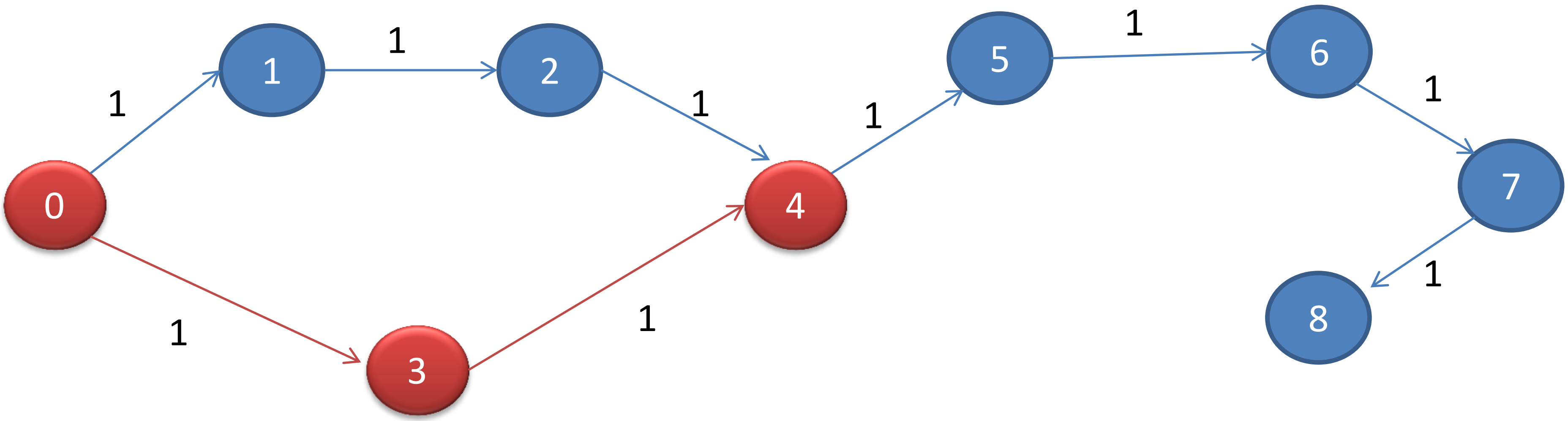
- **Iteration 2:**



The node chosen to be expanded from the open list: 3

Monotone Restriction

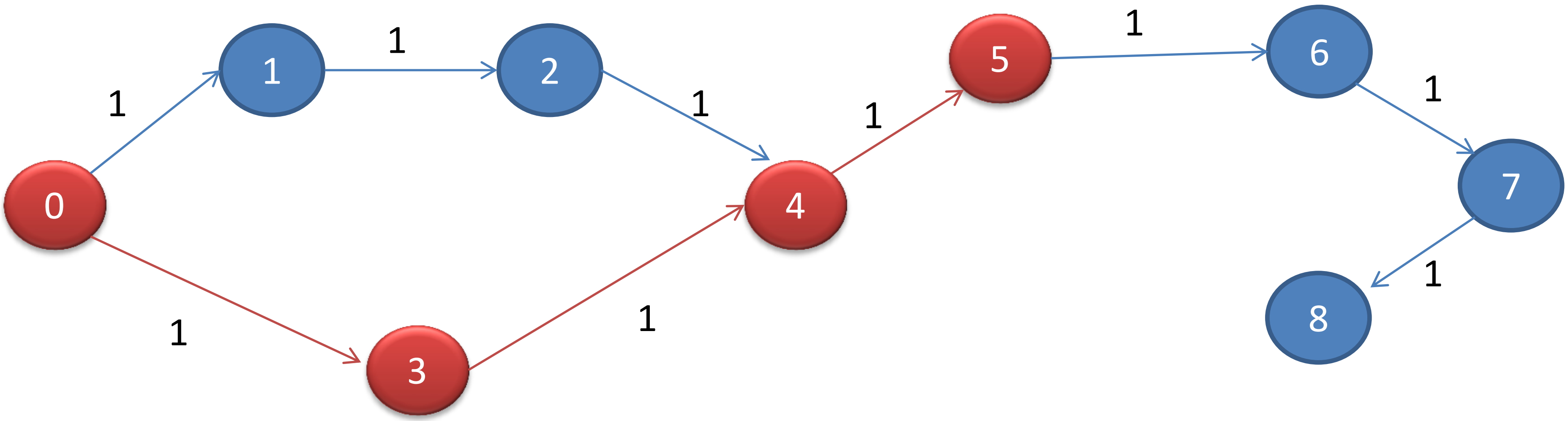
- **Iteration 3:**



The node chosen to be expanded from the open list: 4

Monotone Restriction

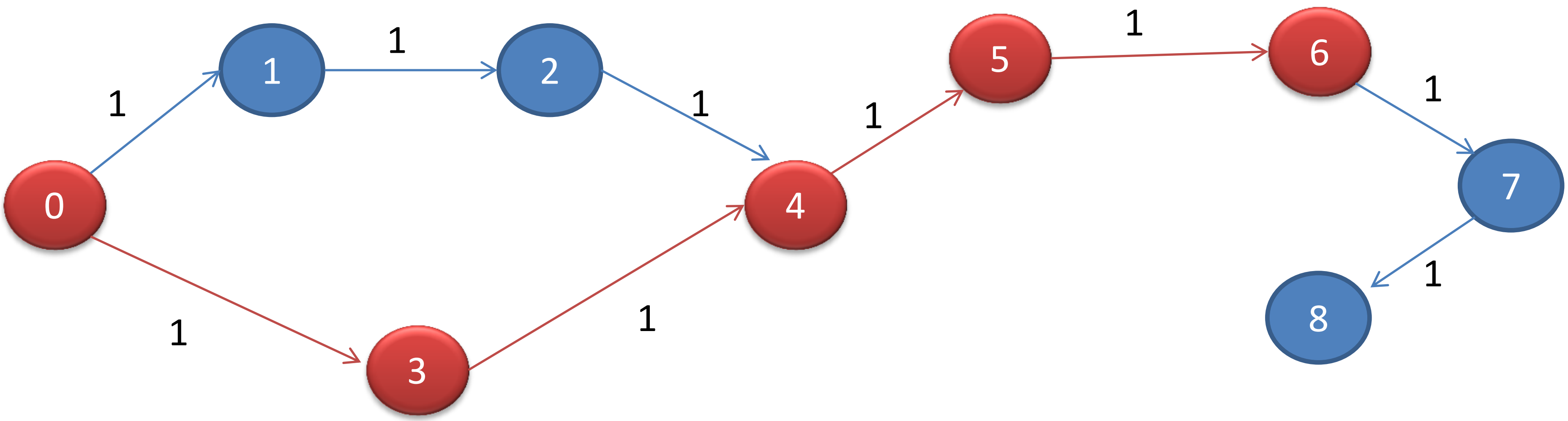
- **Iteration 4:**



The node chosen to be expanded from the open list: 5

Monotone Restriction

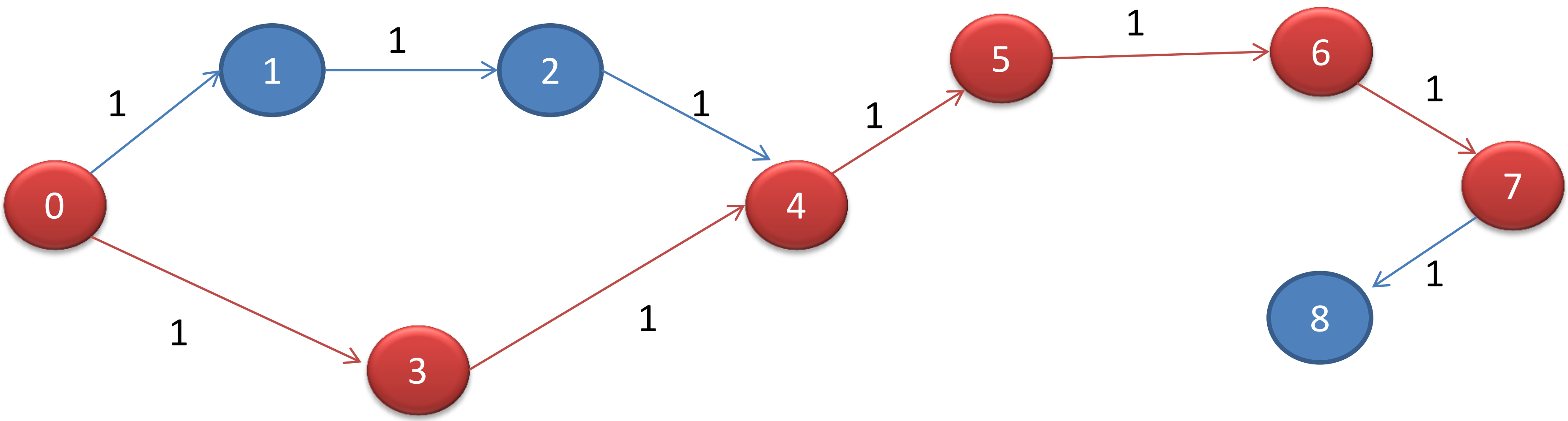
- **Iteration 5:**



The node chosen to be expanded from the open list: 6

Monotone Restriction

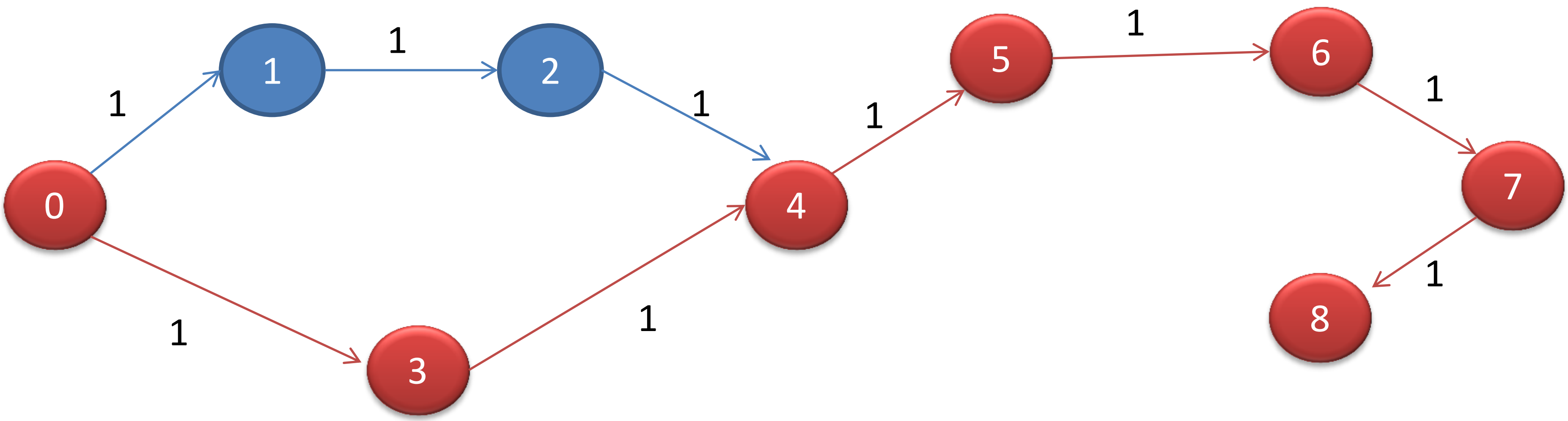
- **Iteration 6:**



The node chosen to be expanded from the open list: 7

Monotone Restriction

- **Final Output:**



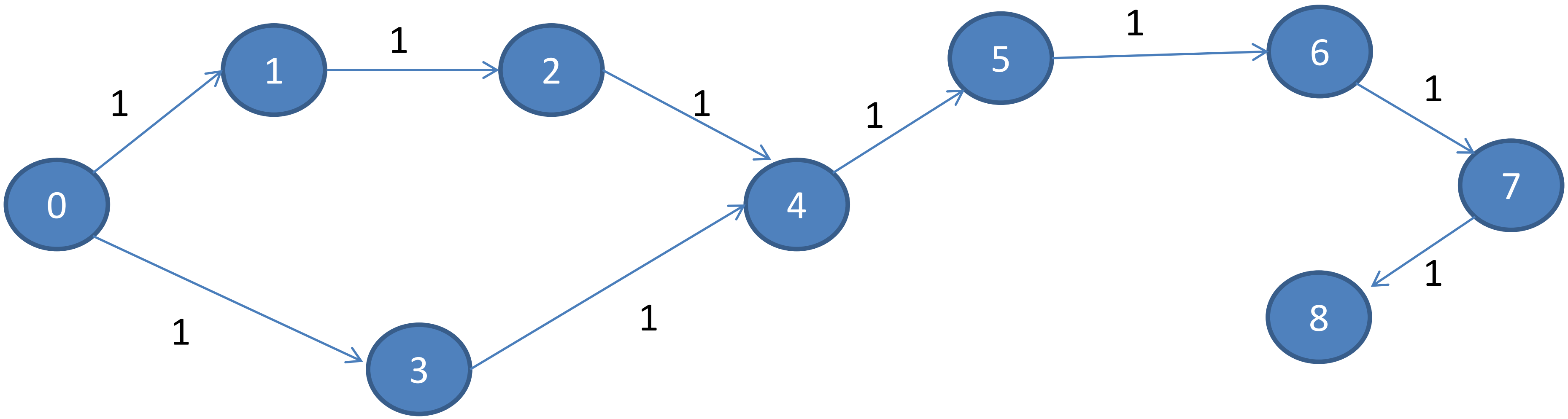
- The optimal path **is:0 3 4 5 6 7 8**
- Optimal path cost is **6**
- Number of iterations taken by the A* Algo =**6**
- Number of Parent Pointer Redirections =**0**
- **So we see that with a when Monotone Restriction is satisfied, parent pointer redirection is not needed**

Bidirectional Search Algorithm

- **Carry out the Forward and Backward search parallely.**
- **For both Forward Search & Backward Search, maintain separate Open Lists and Closed Lists.**
- **Push the Start Node in Open List of Forward Search and Goal Node in Open List of Backward Search.**
- **In each iteration of A* Algorithm,**
  - **Carry out Forward Search.**
  - **Carry out Backward Search.**
  - **Check the intersection of CL of Forward Search & CL of Backward Search.**
  - **If intersection is not NULL, Stop!**
  - **Otherwise carry out another iteration of A* Search**
- **The path discovered is given by :**
  - **Following the parent pointers from Start Node to Intersection Node in Forward Search**
  - **Reversing the direction of Parent Pointers from intersection node to Start node of Backward Search**

Bidirectional Search

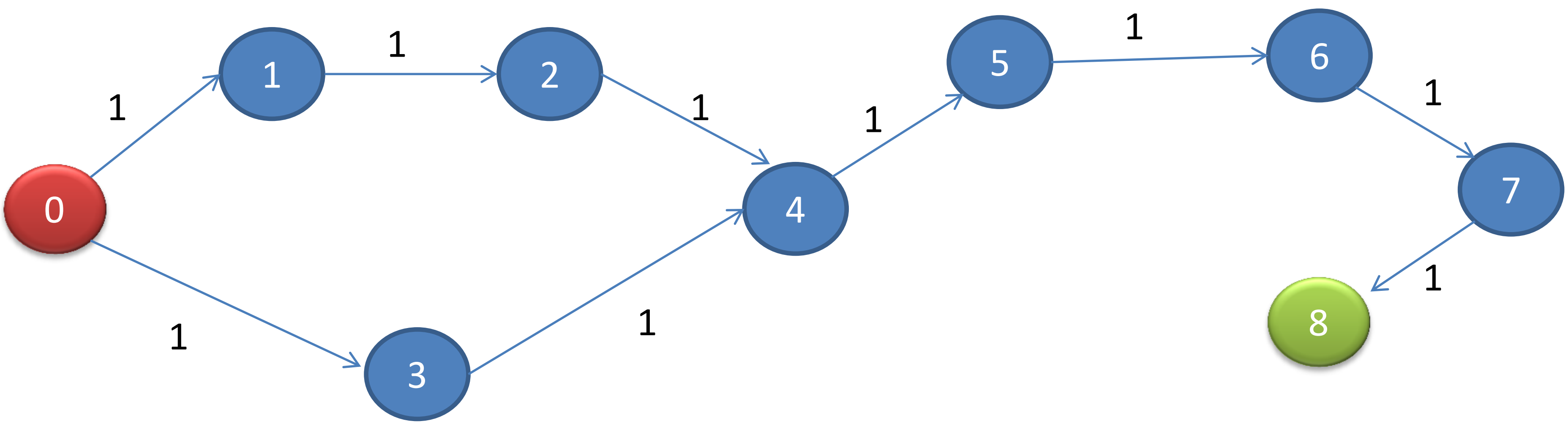
- **We shall use the following graph & heuristic, to do a bidirectional search.**



- **Heuristic** h(i) = 5 for i=3
- 0 otherwise
- **Forward Pass**: Start Node: 0 Goal Node: 8
- **Reverse Pass**: Start Node: 8 Goal Node: 0
- **We chose a node to be expanded from the Open List in forward direction and similarly in backward direction.**

Bidirectional Search

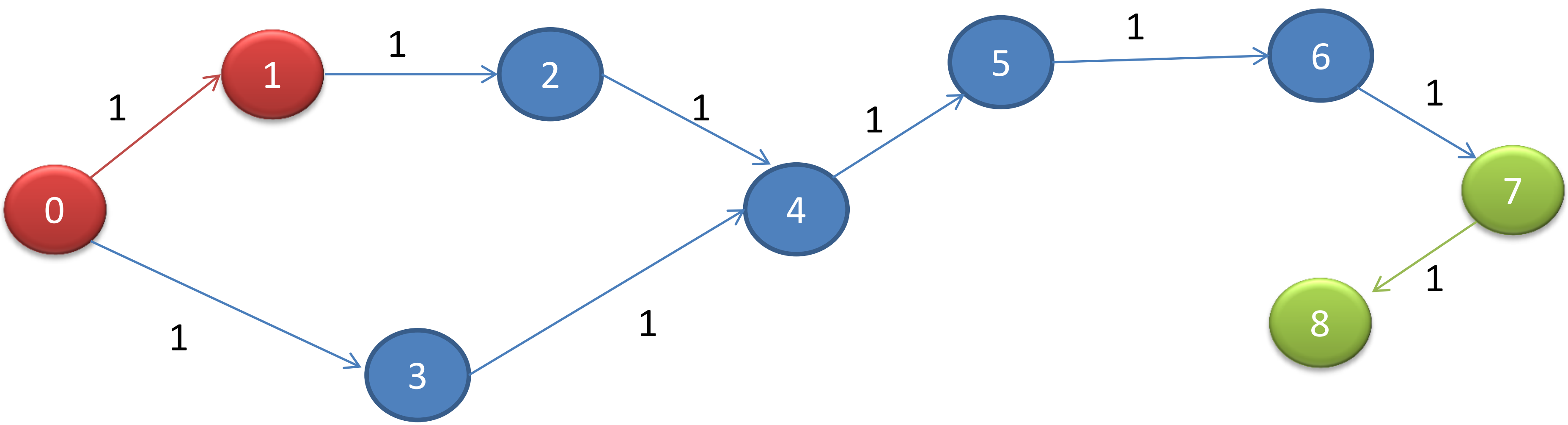
- **Iteration 1:**



The node chosen to be expanded from the open list in FORWARD direction: 0
The node chosen to be expanded from the open list in BACKWARD direction: 8

Bidirectional Search

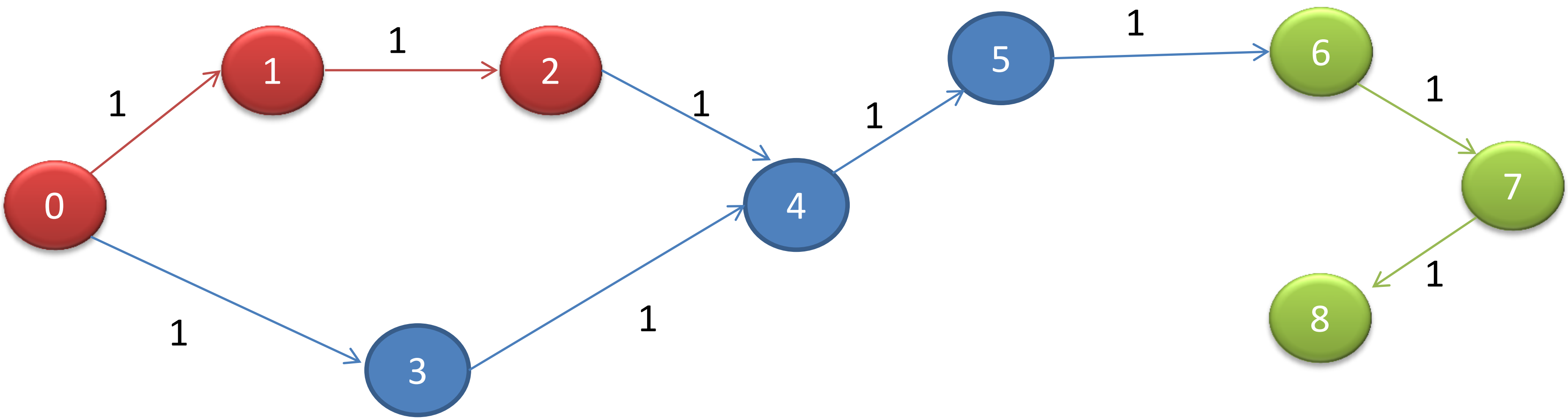
- **Iteration 2:**



The node chosen to be expanded from the open list in FORWARD direction: 1
The node chosen to be expanded from the open list in BACKWARD direction: 7

Bidirectional Search

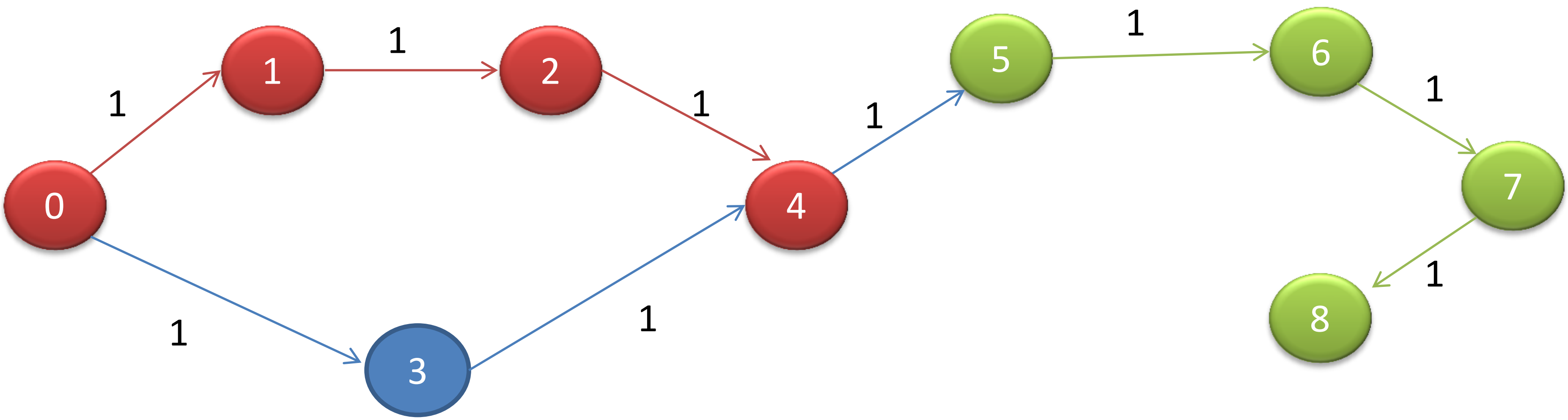
- **Iteration 3:**



The node chosen to be expanded from the open list in FORWARD direction: 2
The node chosen to be expanded from the open list in BACKWARD direction: 6

Bidirectional Search

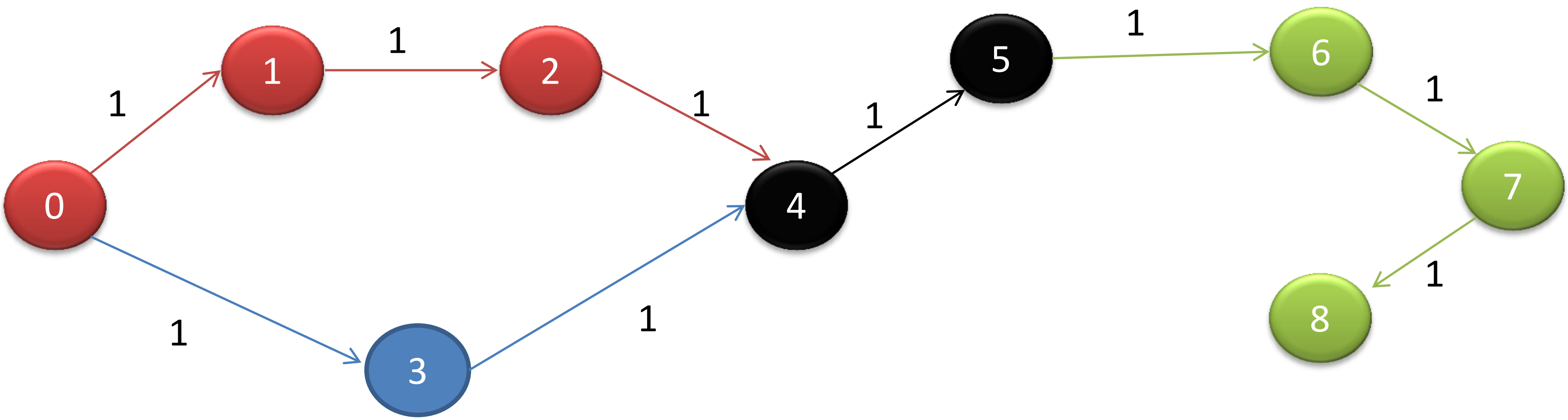
- **Iteration 4:**



The node chosen to be expanded from the open list in FORWARD direction: 4
The node chosen to be expanded from the open list in BACKWARD direction: 5

Bidirectional Search

- **Stop Condition Reached:**



- The search stops when the two searches meet at a common node.
- **The Algorithm takes only 4 iterations to converge.**
- **However the path found is NOT an optimal path.**
- **Cost of discovered path = 7**
- **Cost of optimal path = 6 (via 0-3-4-5-6-7-8)**

Missionaries and Cannibals

- Constraints
  - The boat can carry at most 2 people
  - On no bank should the cannibals outnumber the missionaries

- State : *<#M, #C, P>*
  - *#M* = Number of missionaries on bank *L*
  - *#C* = Number of cannibals on bank *L*
  - *P* = Position of the boat

- *Start State = <3, 3, L>*
- *Goal State = < 0, 0, R >*
- Operations
  - *M2* = Two missionaries take boat
  - *M1* = One missionary takes boat
  - *C2* = Two cannibals take boat
  - *C1* = One cannibal takes boat
  - MC = One missionary and one cannibal takes boat

Missionaries and Cannibals

- **Heuristic, h(n) = 2n-1**
    - **where n=number of people on left bank**
    - **if n = 0 , h = 0 as we have reached the goal state**

- Nodes chosen to be expanded from the open list: 0 1 3 2 4 5 6 7 8 9 10 11 12
- Optimal Path discovered:

  0 1 4 5 6 7 8 9 10 11 12 14
- Optimal Path cost = 11
- Number of iterations taken by A* Algorithm = 13
- Number of parent pointer redirections = 0
- Note that this heuristic is Neither Admissible Nor Monotone
- Consider state S = <2,2,L>
    - h*(S) = 5 (optimal path : <2,2,L> - <0,2,R> - <0,3,L> - <0,1,R> - <0,2,L> - <0,0,R>)
    - h(S) = 7 (2*4-1 = 7) Hence Not Admissible
- Also for P = <2,2,L> C = <0,2,L>
    - h(P) = 7 h(C) = 3
    - C(P,C) = 1 Hence Not Monotone

Missionaries and Cannibals

- **Heuristic, h(n) = 2n+1**
  - **where n=number of people on left bank**
  - **if n = 0 , h = 0 as we have reached the goal state**

- Nodes chosen to be expanded from the open list: 0 1 3 2 4 5 6 7 8 9 10 11 12
- Optimal Path discovered:

  0 1 4 5 6 7 8 9 10 11 12 14
- Optimal Path cost = 11
- Number of iterations taken by A* Algorithm = 13
- Number of parent pointer redirections = 0
- Note that this heuristic is Neither Admissible Nor Monotone
- Consider state S = <2,2,L>
  - h*(S) = 5 (optimal path : <2,2,L> - <0,2,R> - <0,3,L> - <0,1,R> - <0,2,L> - <0,0,R>)
  - h(S) = 9 (2*4+1 = 9) Hence Not Admissible
- Also for P = <2,2,L> C = <0,2,L>
  - h(P) = 9 h(C) = 5
  - C(P,C) = 1 Hence Not Monotone

Missionaries and Cannibals

- **Heuristic, h(n) = n-1**
  - **where n=number of people on left bank**
  - **if n = 0 , h = 0 as we have reached the goal state**

- Nodes chosen to be expanded from the open list: 0 1 3 2 4 5 6 7 8 9 10 11 12 13
- Optimal Path discovered:

  0 1 4 5 6 7 8 9 10 11 12 14
- Optimal Path cost = 11
- Number of iterations taken by A* Algorithm = 13
- Number of parent pointer redirections = 0
- Note that this heuristic is Admissible but Not Monotone
- Consider P = <2,2,L> C = <0,2,L>
  - h(P) = 3 h(C) = 1
  - C(P,C) = 1 Hence Not Monotone

Missionaries and Cannibals

- **Heuristic, h(n) = n/2**
  - **where n=number of people on left bank**
  - **if n = 0 , h = 0 as we have reached the goal state**
- Nodes chosen to be expanded from the open list: 0 3 4 1 2 5 6 7 8 9 10 11 12 13
- Optimal Path discovered:

  0 3 4 5 6 7 8 9 10 11 12 14
- Optimal Path cost = 11
- Number of iterations taken by A* Algorithm = 14
- Number of parent pointer redirections = 0
- This heuristic is Admissible as well as Monotone.

8-Puzzle Problem

2		3
1	8	5
4	7	6

Start State

1	2	3
4	5	6
7	8	

Goal State

- Tile movement represented as the movement of the blank space.
- Operators:
  - L : Blank moves left
  - R : Blank moves right
  - U : Blank moves up
  - D : Blank moves down
  - $C(L) = C(R) = C(U) = C(D) = 1$

8-Puzzle Problem

- Heuristic, $h(n)$ = sum of Manhattan distances of tiles from their destined position

  $h(n)$ = X-dist + Y-dist from Goal State

- The optimal path is:0-1- 4-10-22-39-69-119
- Optimal path cost is 7
- Number of nodes expanded from Open List =7
- Number of Parent Pointer Redirections =0

8-Puzzle Problem

- Heuristic, h(n) = no. of tiles displaced from their destined position.
- The optimal path is:0 1 4 10 22 39 69 119
- Optimal path cost is 7
- Number of nodes expanded from Open List = 8
- Number of Parent Pointer Redirections =0

- Here also we see that ***Manhattan Distance***, which is a better heuristic than ***No. of Displaced Tiles*** heuristic performs better(converges faster).

8-Puzzle Problem(our own heuristic)

- Heuristic, h(n) = Manhattan distances of only the blank tile
- The optimal path is:0-1- 4-10-22-39-69-119
- Optimal path cost is 7
- Number of nodes expanded from Open List =**53**
- Number of Parent Pointer Redirections =0

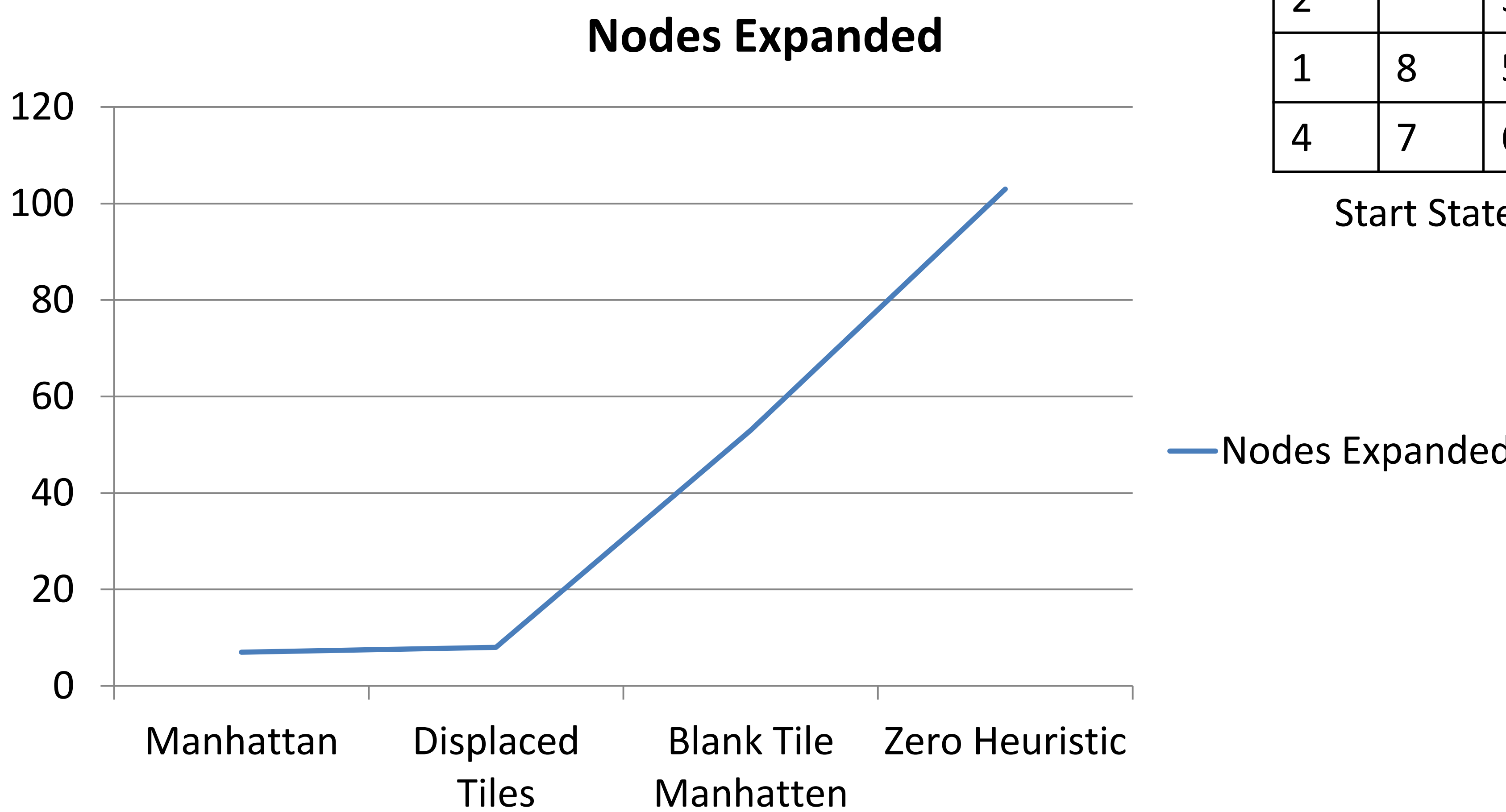
- Now we observe that this heuristic expands a lot higher number of nodes from the Open List as this is a very poor heuristic.

8-Puzzle Problem(our own heuristic)

- Heuristic, h(n) = 0 for all nodes
- The optimal path is:0-1- 4-10-22-39-69-119
- Optimal path cost is 7
- Number of nodes expanded from Open List =103
- Number of Parent Pointer Redirections =0

- Now we observe that this heuristic expands even a lot higher number of nodes from the Open List as this is a very poor heuristic.

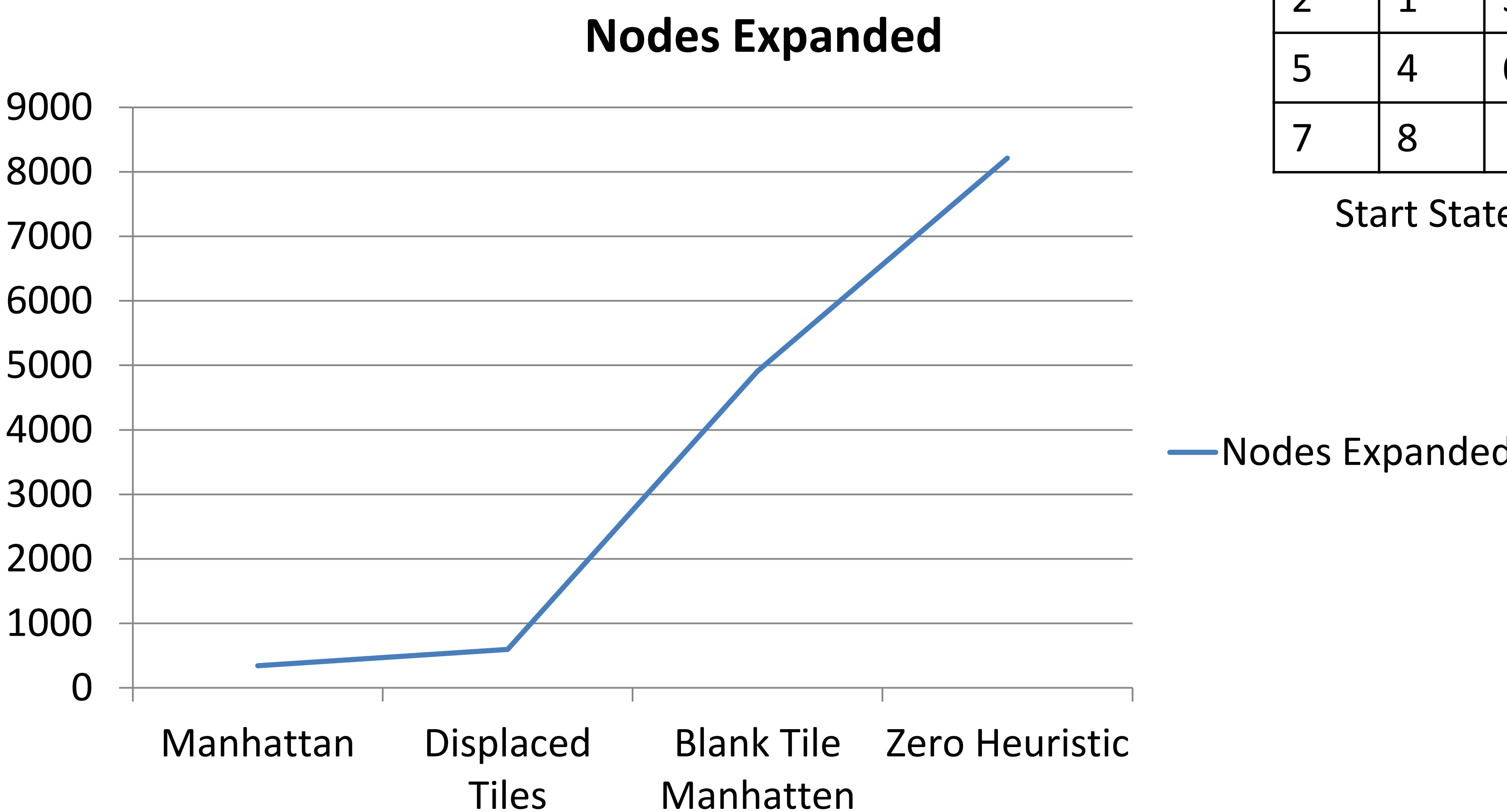
Comparing Heuristics (8-Puzzle)



2		3
1	8	5
4	7	6

Start State

Comparing Heuristics (8-Puzzle)



2	1	3
5	4	6
7	8	

Start State

8-Puzzle Non Reachability

Start State

2	1	3
5	4	7
6	8	

Goal State

1	2	3
4	5	6
7	8	

- We now have a start state from which we can not reach to the goal state.
- However, we can figure the non-reachability before running the A* Algorithm by using the following strategy:

I. Write the start state puzzle in row major form i.e. in above case start state can be written as [2 1 3 5 4 7 6 8].

II. Count the no. of inversions in this array.

III. Repeat the above steps (I) & (II) for goal state. Since Goal State is fixed, row major form is [1 2 3 4 5 6 7 8] & its no of inversions are zero.

IV. Now we Use the following rule:

"Start State with even no of inversions can reach a Goal State with even no. of inversions and

Start State with odd no of inversions can reach a Goal State with odd no. of inversions"

V. Since our goal state contains 0 (even) inversions, so **our start state must have even no. of inversions**.