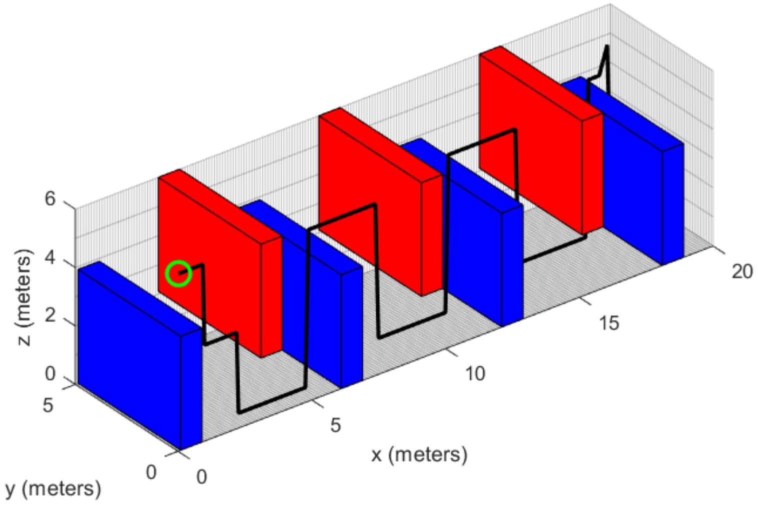
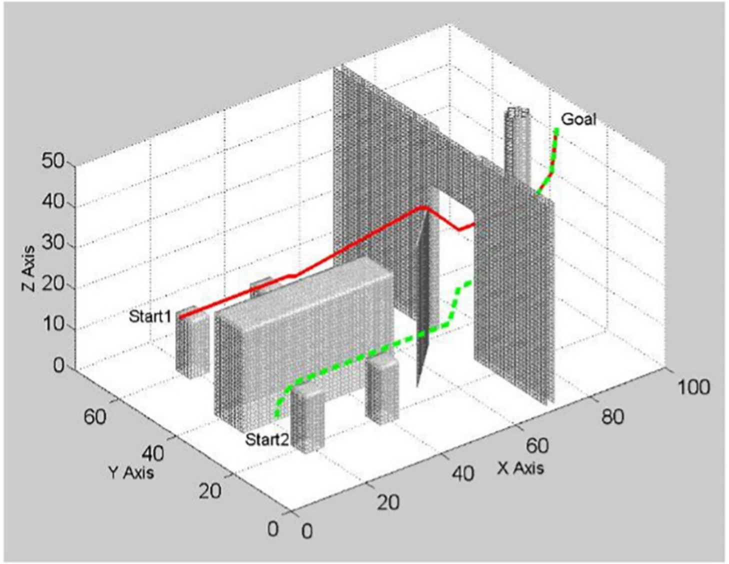
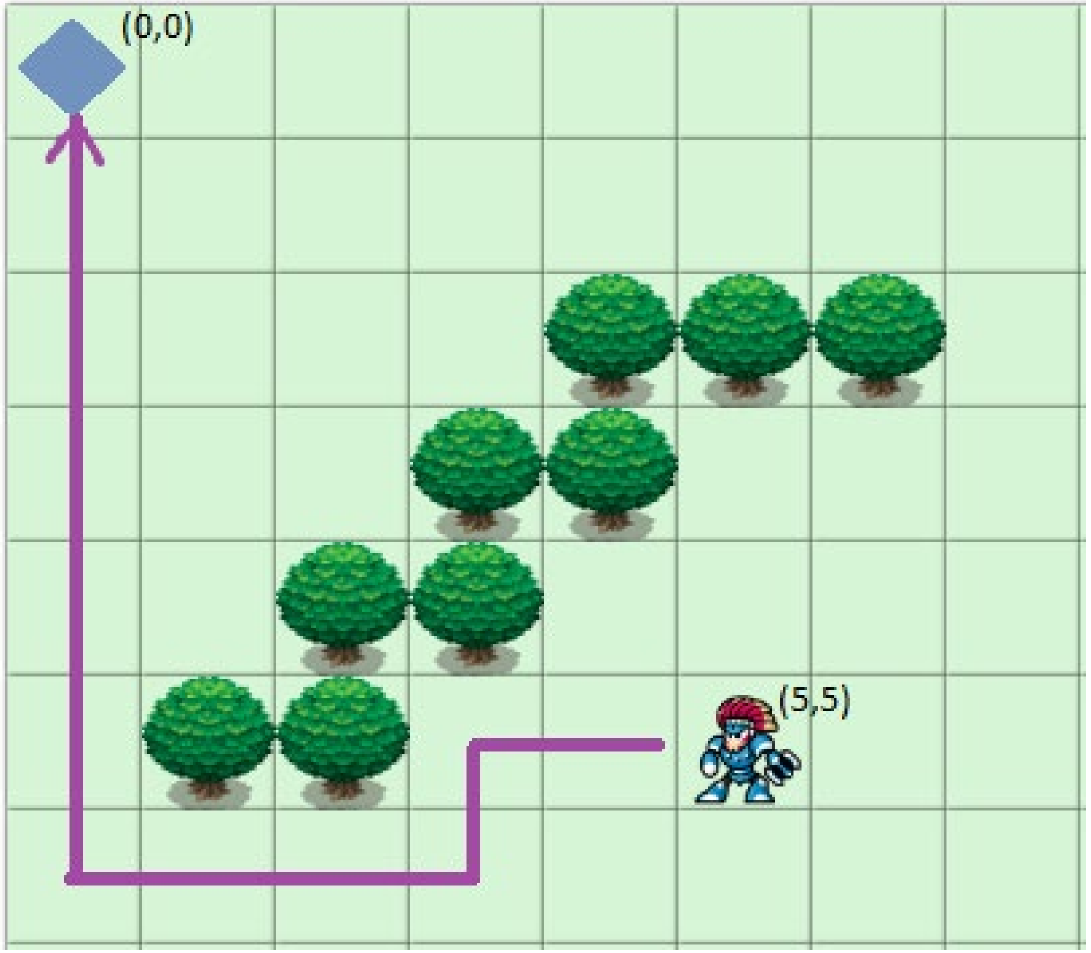
Introduction to planning algorithms

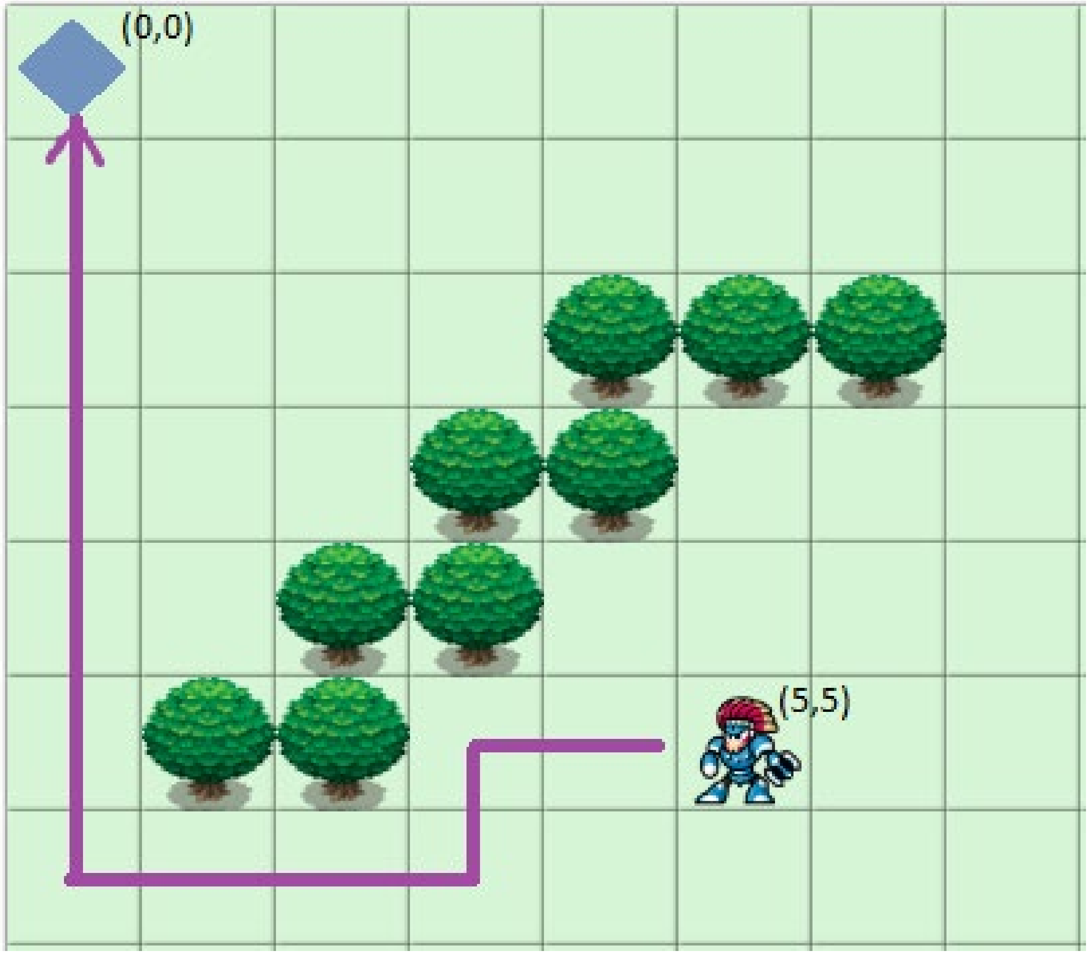




Introduction to planning algorithms



Introduction to planning algorithms



Robot configuration

State: (x,y)

Configuration space

Total number of states: 56

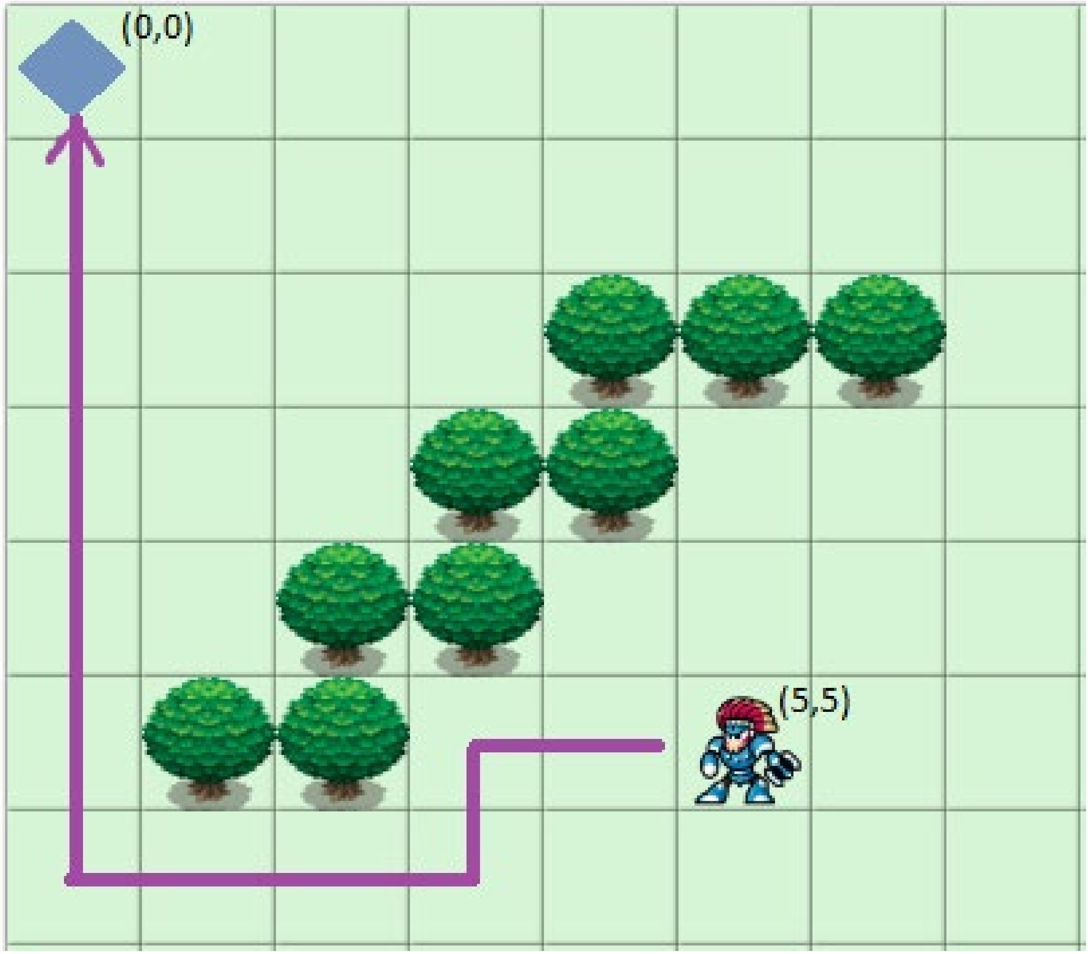
Free space

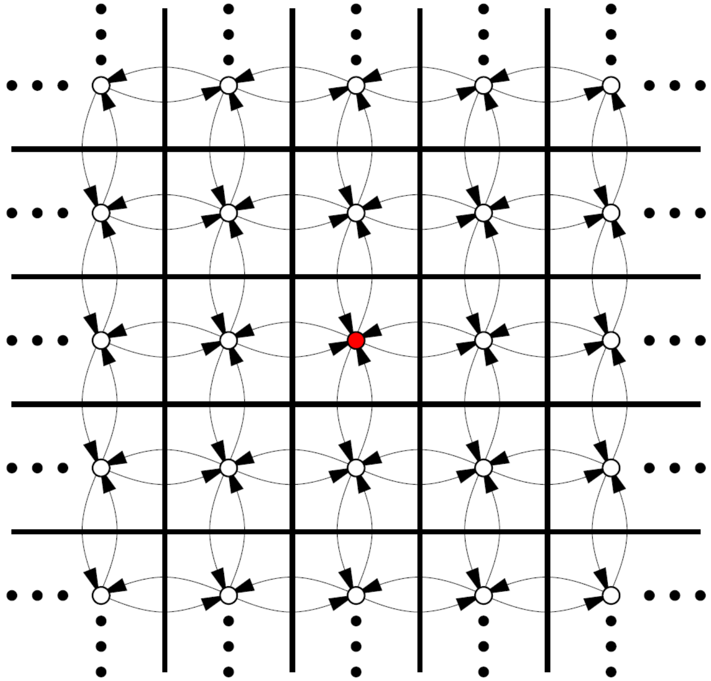
Total number of states: 47

Obstacle space

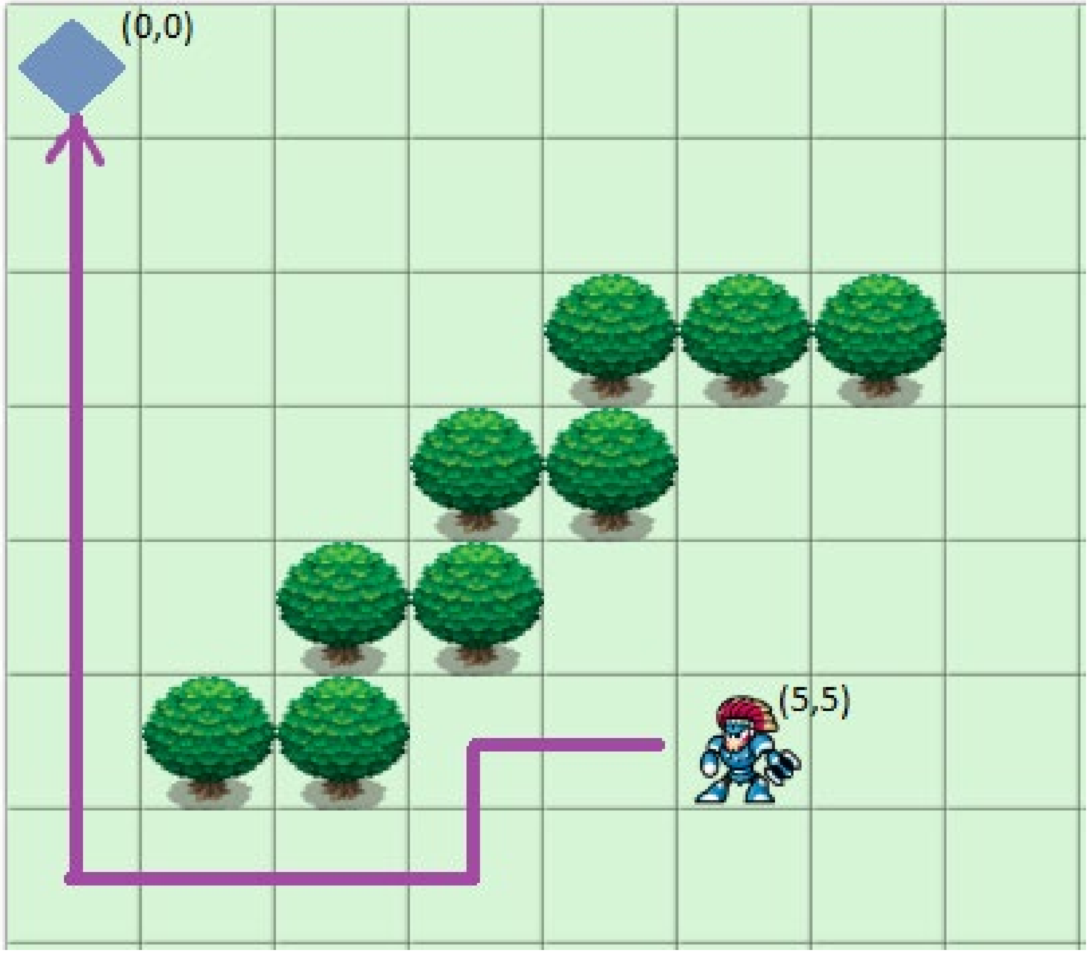
Total number of states: 9

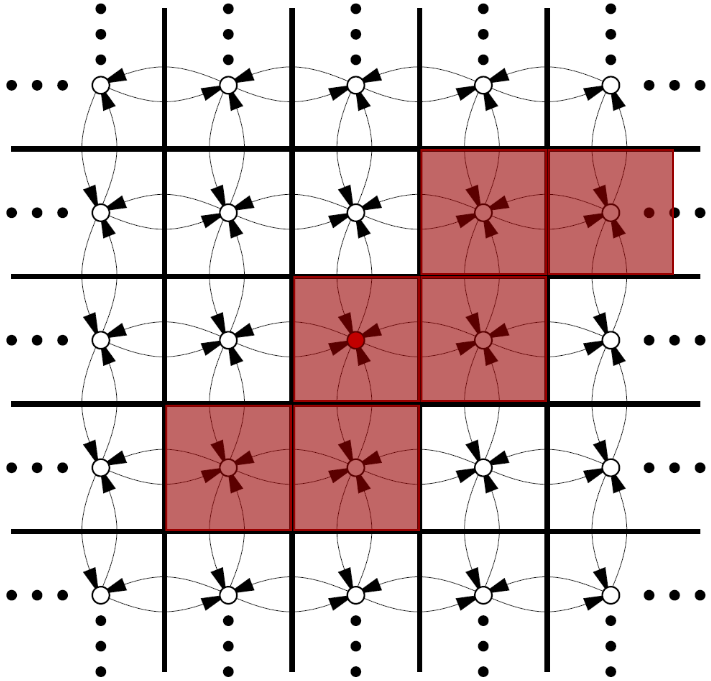
Introduction to planning algorithms



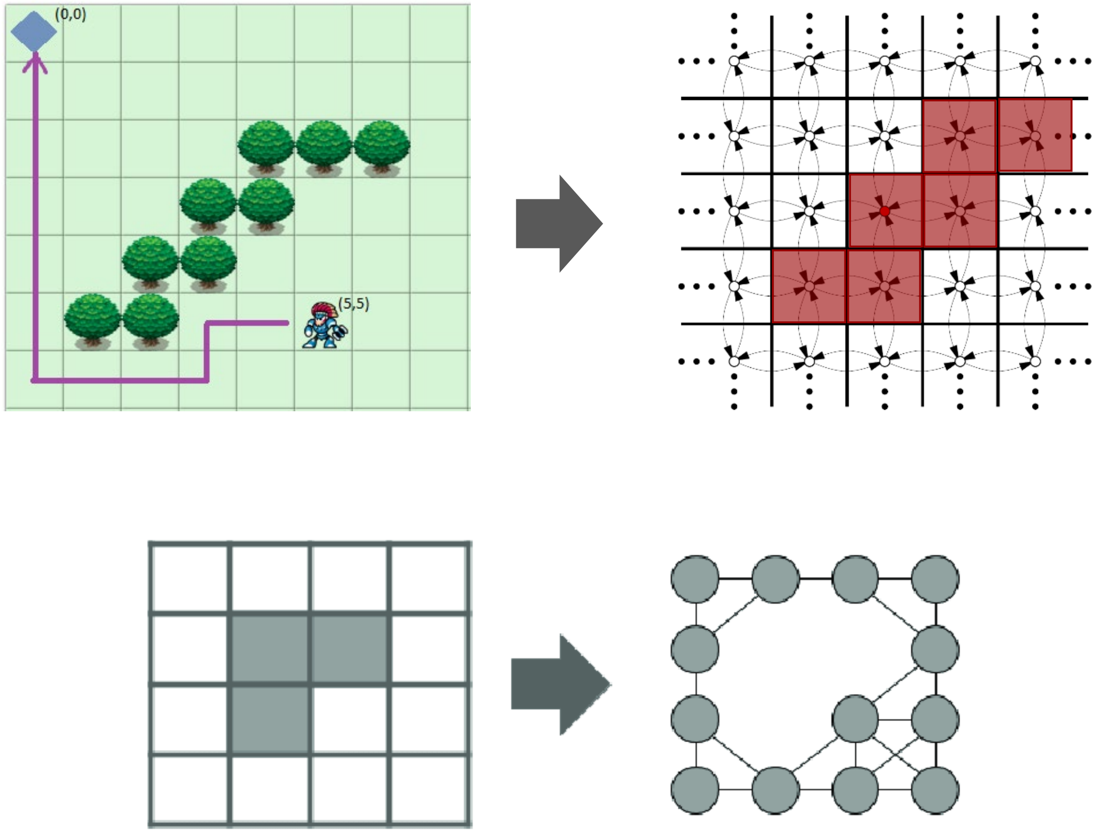


Introduction to planning algorithms

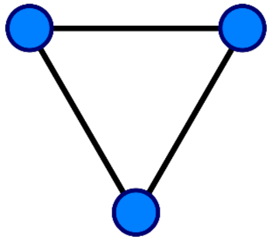




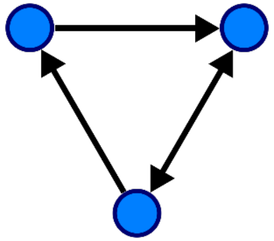
Introduction to planning algorithms



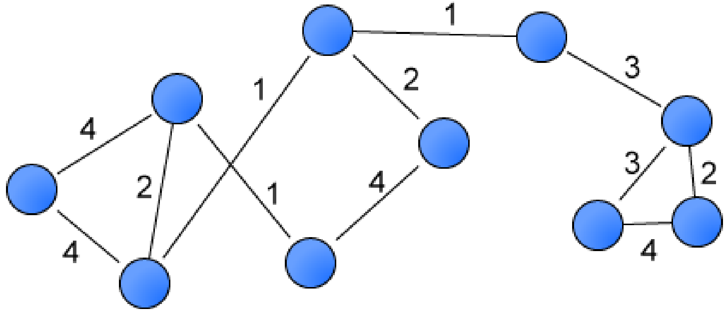
Graphs



Undirected graph



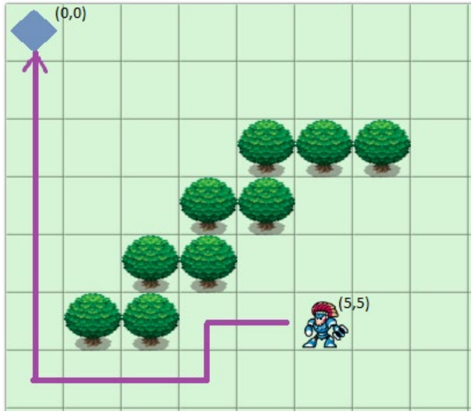
Directed graph



Weighed (undirected) graph

Basic ingredients of planning

- State
- Time
- Actions
- Initial state
- Goal state



Desired outcome of a plan

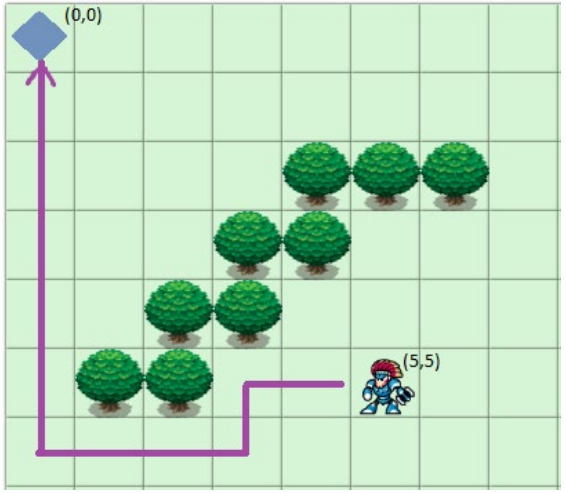
1) Feasibility: Find a plan that causes arrival at a goal state, regardless of its efficiency.
2) Optimality: Find a feasible plan that optimizes performance in some carefully specified manner, in addition to arriving in a goal state.

Formulation 2.1 (Discrete Feasible Planning)

1. A nonempty *state space* X, which is a finite or countably infinite set of *states*.
2. For each state $x \in X$, a finite *action space* $U(x)$.
3. A *state transition function* f that produces a state $f(x,u) \in X$ for every $x \in X$ and $u \in U(x)$. The *state transition equation* is derived from f as $x' = f(x,u)$.
4. An *initial state* $x_I \in X$.
5. A *goal set* $X_G \subset X$.

FORWARD_SEARCH

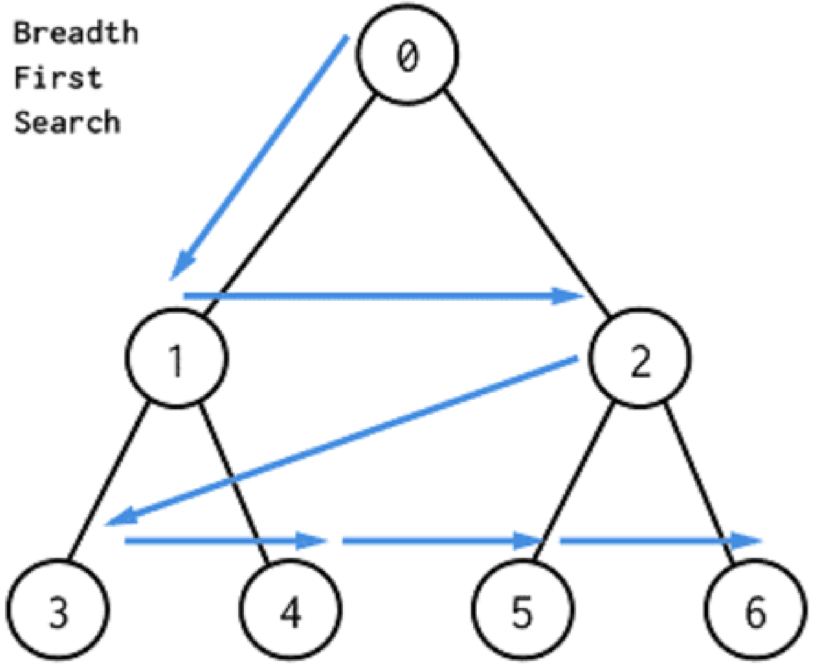
$Q.Insert(x_I)$ and mark x_I as visited
while Q not empty **do**
  $x \leftarrow Q.GetFirst()$
  if $x \in X_G$
    return SUCCESS
  forall $u \in U(x)$
    $x' \leftarrow f(x,u)$
    if x' not visited
      Mark x' as visited
      $Q.Insert(x')$
    else
      Resolve duplicate x'
return FAILURE

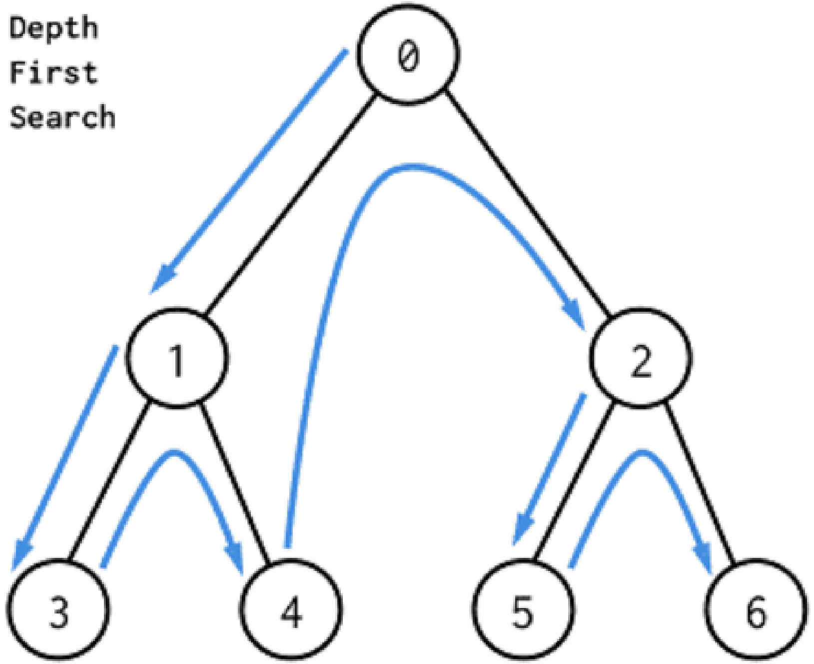


Sorting the queue

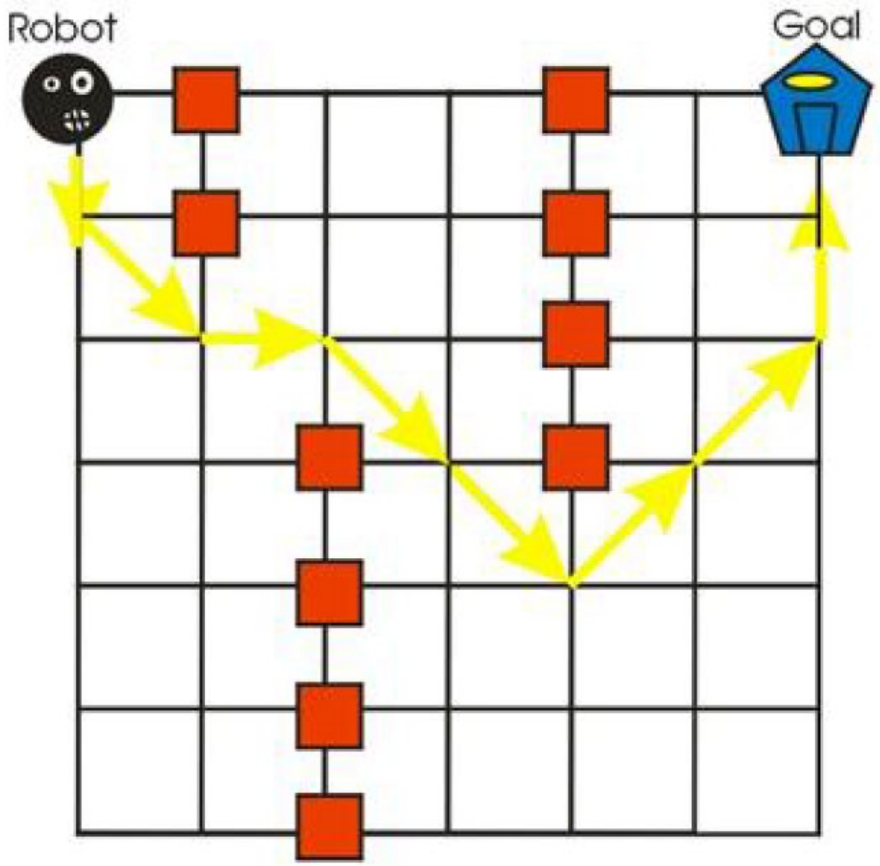
	BFS	DFS	Dijkstra	A*	(Greedy) Best-first
Sorting mechanism	FIFO (fist in, first out)	LIFO (last in, first out)	Priority queue using calculated cost to come	Priority queue using calculated cost to come plus heuristic	Priority queue using just a heuristic
Feasibility guaranteed	Always	For finite state spaces	Always	Always	Always
Optimality guaranteed	No	No	Yes	Yes	No

Breadth-First search vs. Depth-First search



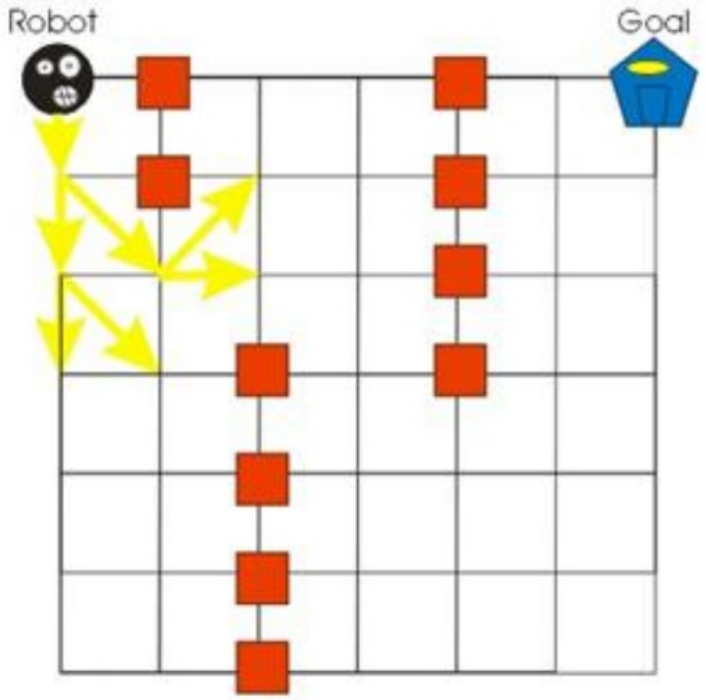


Breadth-First search vs. Depth-First search

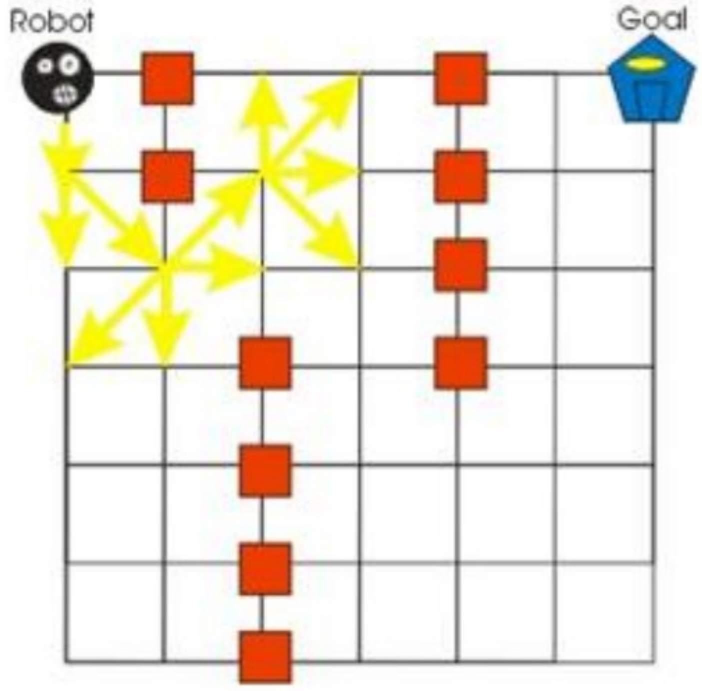


Breadth-First search vs. Depth-First search

BFS



DFS



Breadth-First search

FORWARD_SEARCH

$Q.Insert(x_I)$ and mark x_I as visited

while Q not empty **do**   First in – First out (queue)

 $x \leftarrow Q.GetFirst()$

 if $x \in X_G$

  return SUCCESS

 forall $u \in U(x)$

  $x' \leftarrow f(x, u)$

  if x' not visited

   Mark x' as visited

   $Q.Insert(x')$

  else   Do nothing

   Resolve duplicate x'

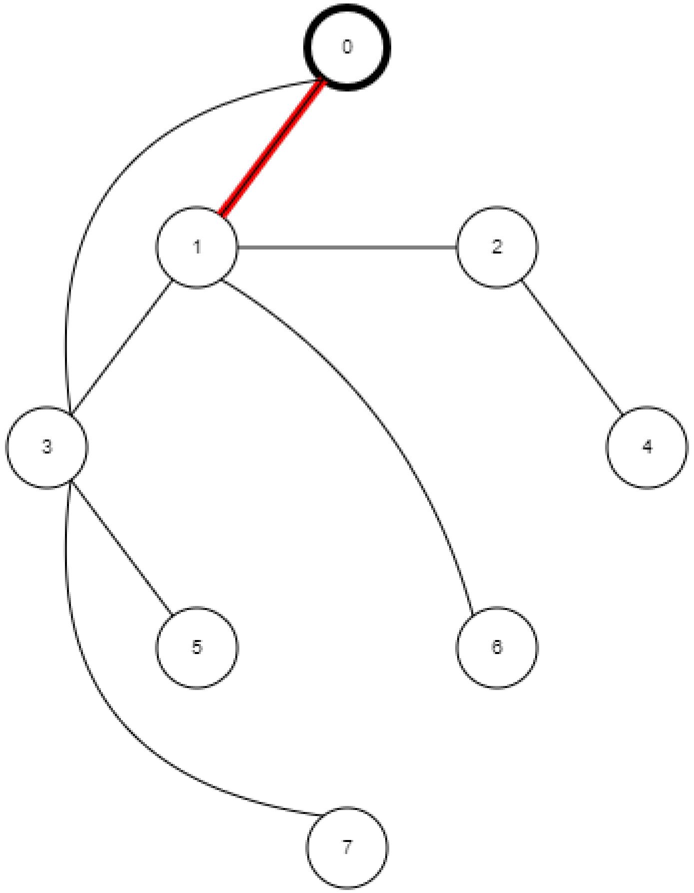
return FAILURE

Breadth-First search

BFS Queue

0

	Parent	Visited
0		f
1		$\boxed{f}$
2		f
3		f
4		f
5		f
6		f
7		f

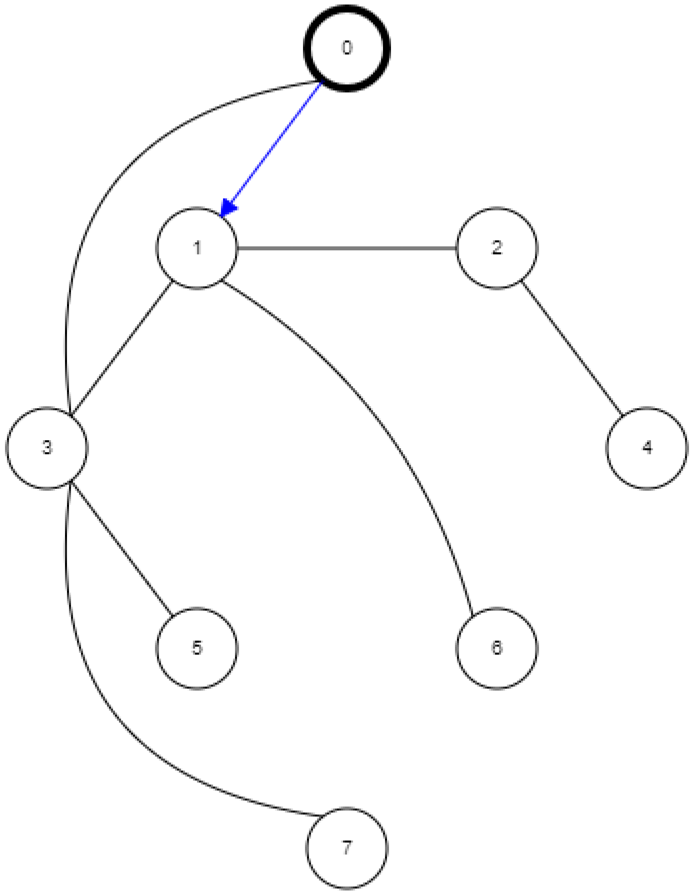


Breadth-First search

BFS Queue

0
1

	Parent	Visited
0		f
1	0	T
2		f
3		f
4		f
5		f
6		f
7		f

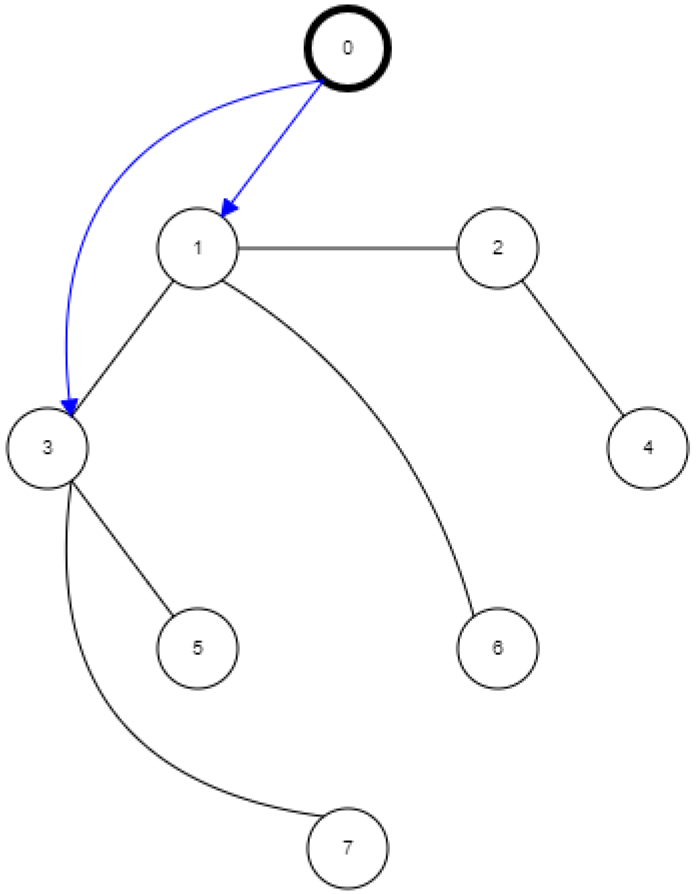


Breadth-First search

BFS Queue

0 1 3

	Parent		Visited
0		0	f
1	0	1	T
2		2	f
3	0	3	T
4		4	f
5		5	f
6		6	f
7		7	f

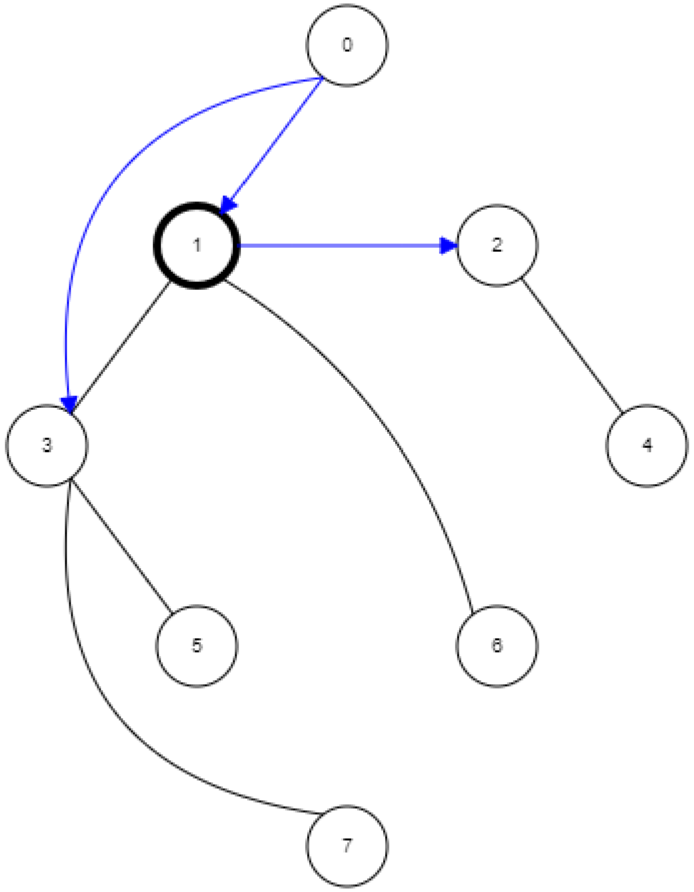


Breadth-First search

BFS Queue

1 3 2

	Parent	Visited
0		f
1	0	T
2	1	T
3	0	T
4		f
5		f
6		f
7		f

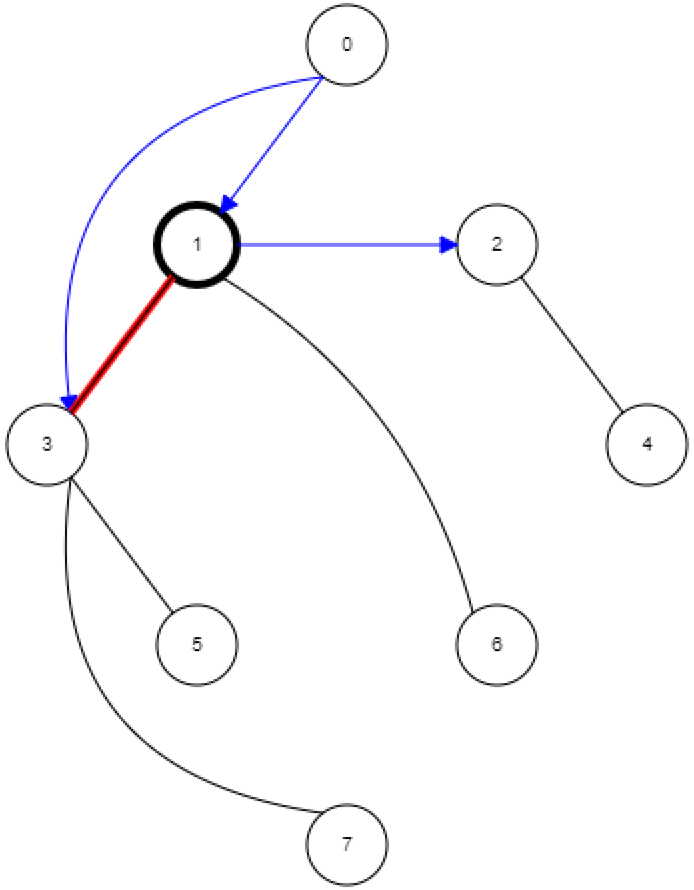


Breadth-First search

BFS Queue

1 3 2

	Parent	Visited
0		f
1	0	T
2	1	T
3	0	T
4		f
5		f
6		f
7		f

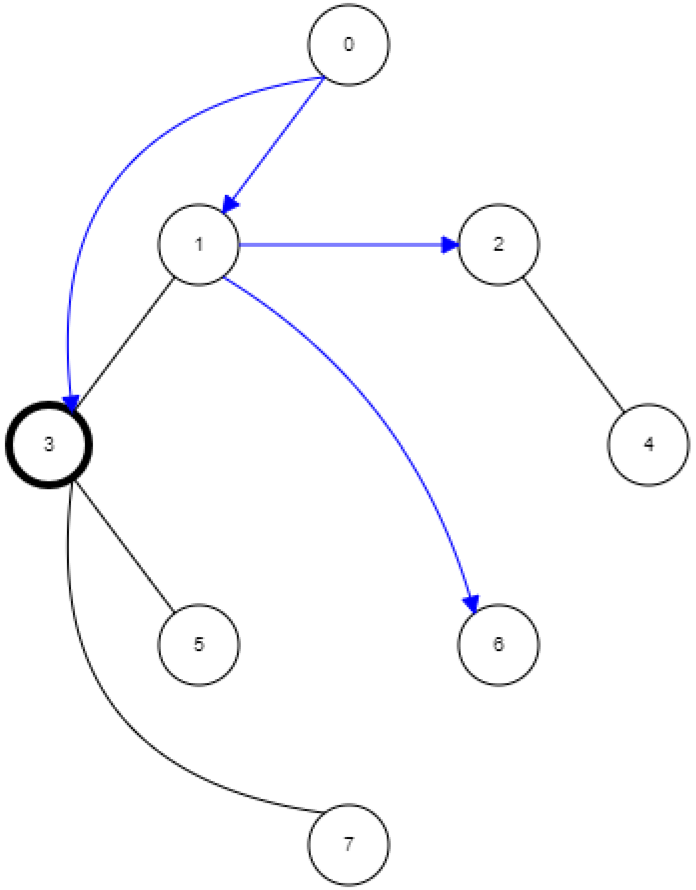


Breadth-First search

BFS Queue

3 2 6

	Parent	Visited
0		f
1	0	T
2	1	T
3	0	T
4		f
5		f
6	1	T
7		f



Breadth-First search

BFS Queue

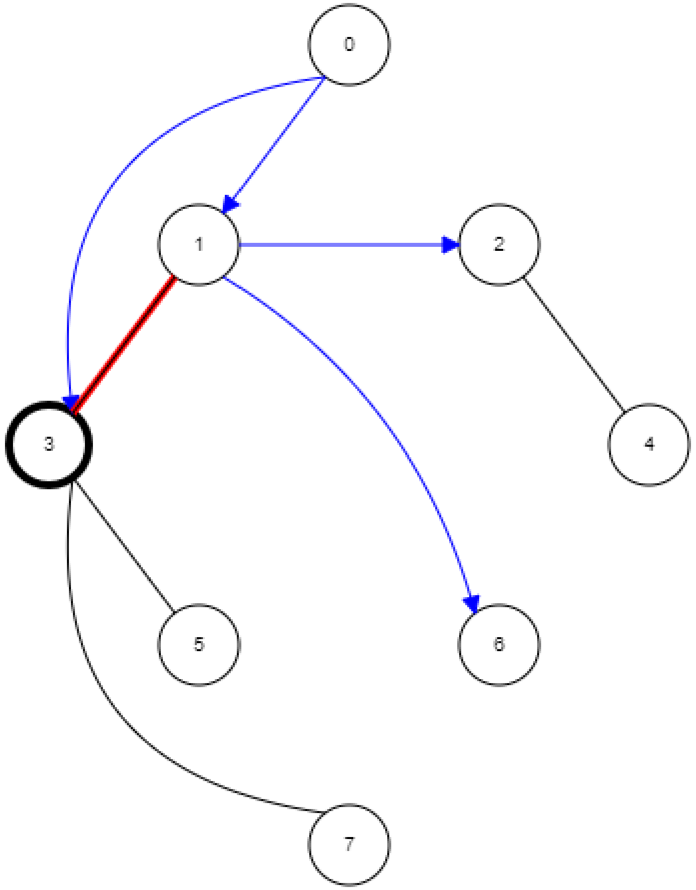
3 2 6

Parent

0	
1	0
2	1
3	0
4	
5	
6	1
7	

Visited

0	f
1	T
2	T
3	T
4	f
5	f
6	T
7	f

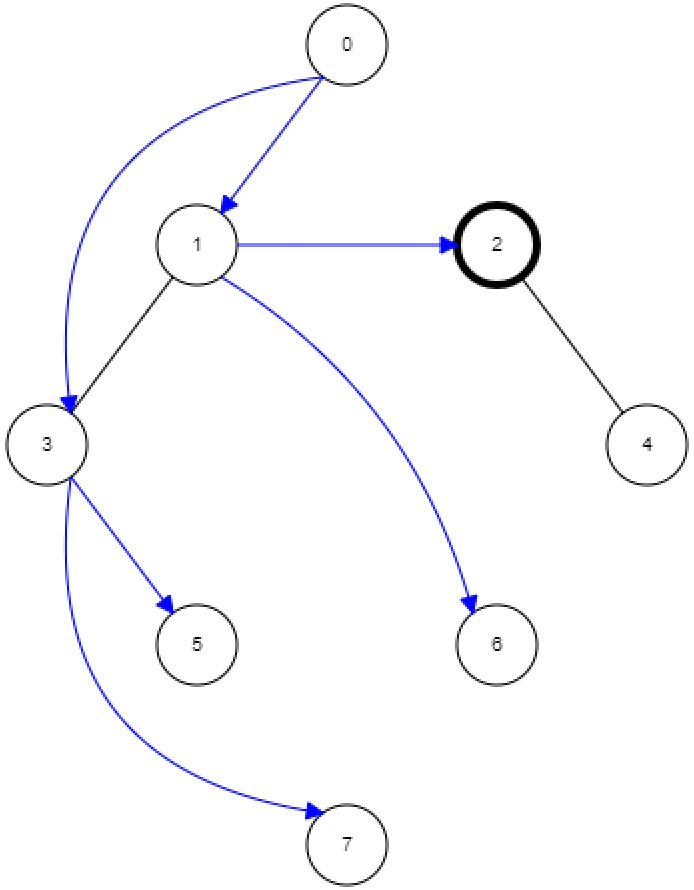


Breadth-First search

BFS Queue

2 6 5 7

	Parent	Visited
0		f
1	0	T
2	1	T
3	0	T
4		f
5	3	T
6	1	T
7	3	T



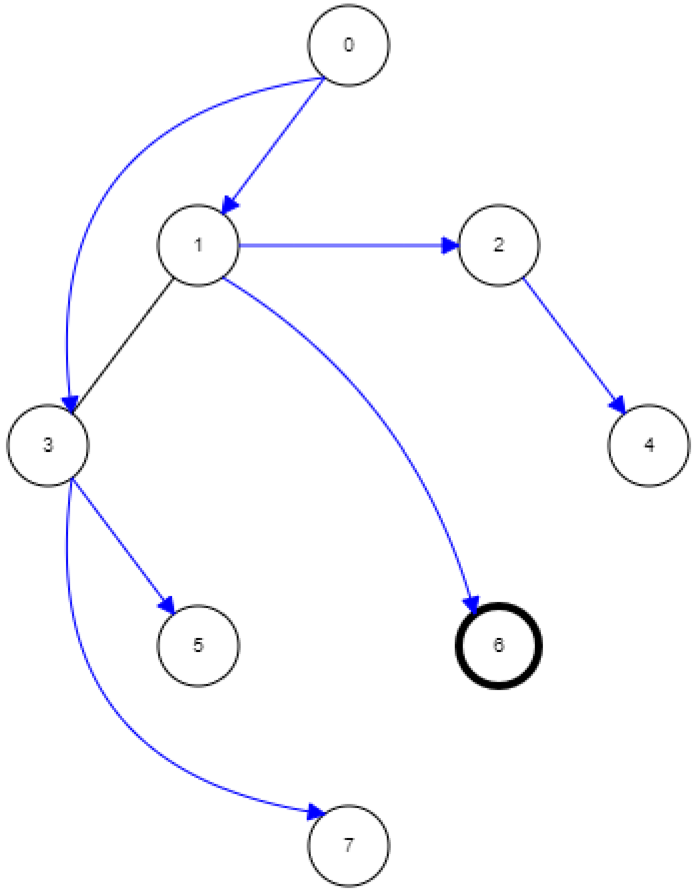
Breadth-First search

BFS Queue

6 5 7 4

	Parent
0	
1	0
2	1
3	0
4	2
5	3
6	1
7	3

	Visited
0	f
1	T
2	T
3	T
4	T
5	T
6	T
7	T

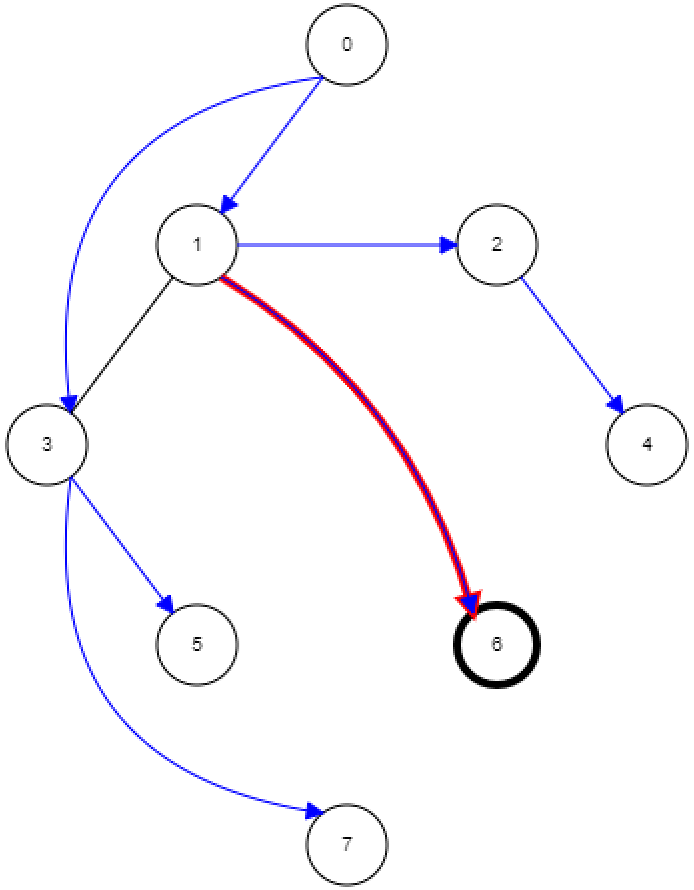


Breadth-First search

BFS Queue

6 5 7 4

	Parent	Visited
0		f
1	0	T
2	1	T
3	0	T
4	2	T
5	3	T
6	1	T
7	3	T

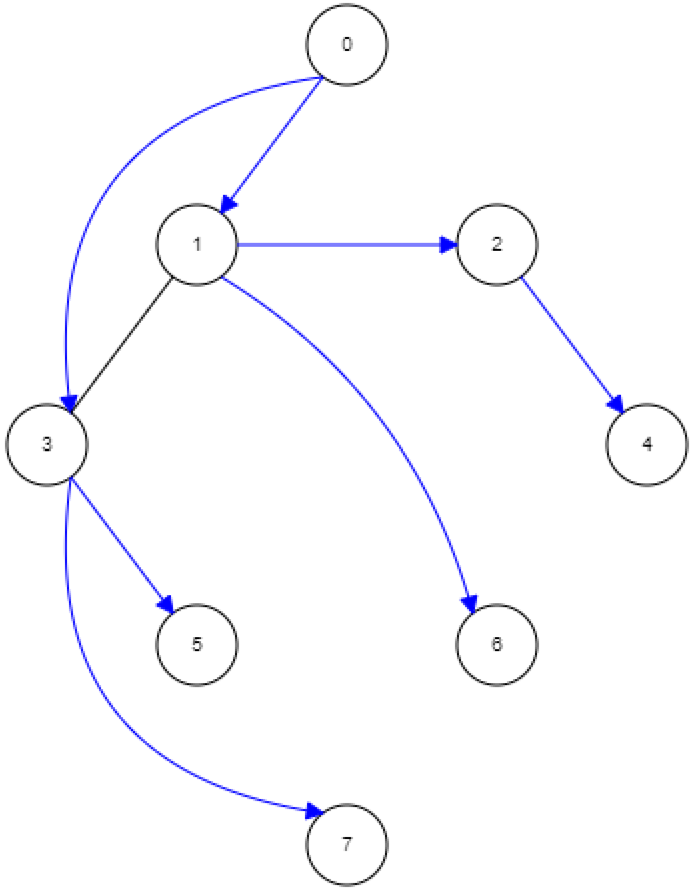


Breadth-First search

BFS Queue

	Parent
0	
1	0
2	1
3	0
4	2
5	3
6	1
7	3

	Visited
0	f
1	T
2	T
3	T
4	T
5	T
6	T
7	T



Depth-First search

FORWARD_SEARCH

$Q.Insert(x_I)$ and mark x_I as visited
while Q not empty **do** — Last in – First out (stack)
    $x \leftarrow Q.GetFirst()$
    if $x \in X_G$
        return SUCCESS
    forall $u \in U(x)$
        $x' \leftarrow f(x, u)$
        if x' not visited
            Mark x' as visited
            $Q.Insert(x')$
        else — Do nothing
            Resolve duplicate x'
return FAILURE

Depth-First search

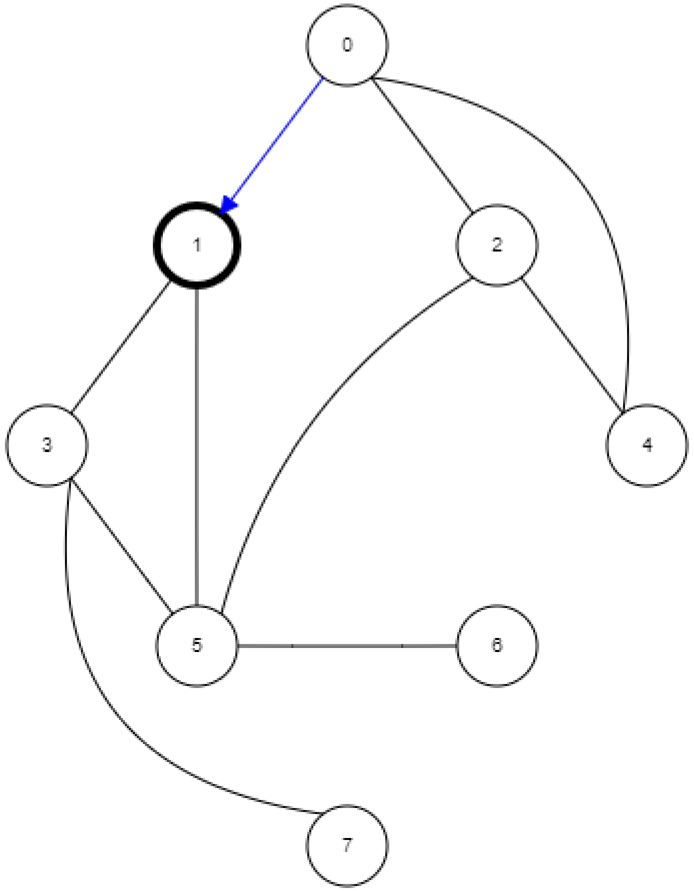
DFS(0)

Parent

0	
1	0
2	
3	
4	
5	
6	
7	

Visited

0	T
1	f
2	f
3	f
4	f
5	f
6	f
7	f



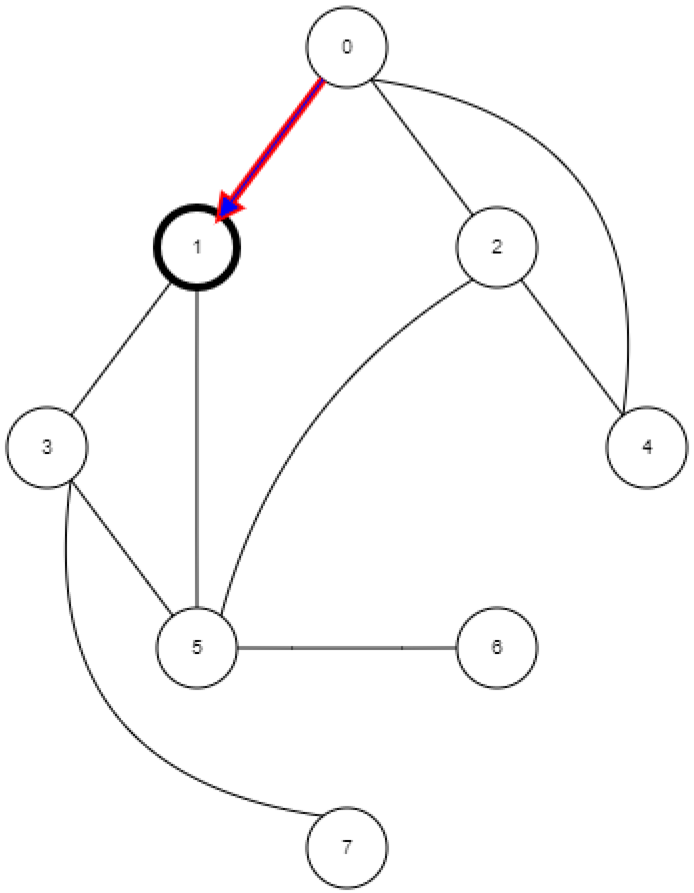
Depth-First search

DFS(0)

DFS(1)

Vertex 0 already visited.

	Parent	Visited
0		T
1	0	T
2		f
3		f
4		f
5		f
6		f
7		f

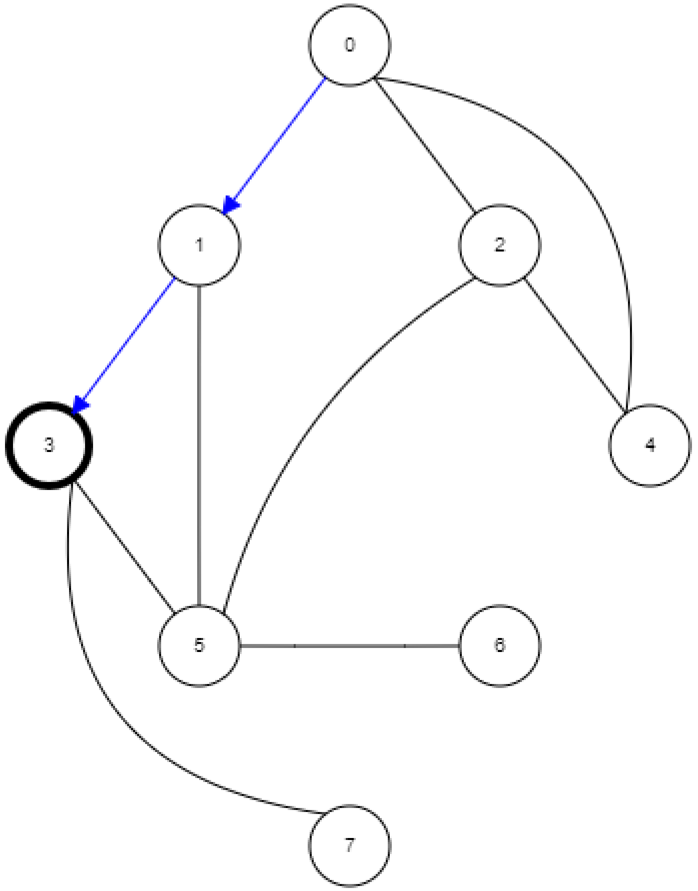


Depth-First search

DFS(0)

DFS(1)

	Parent		Visited
0		0	T
1	0	1	T
2		2	f
3	1	3	f
4		4	f
5		5	f
6		6	f
7		7	f



Depth-First search

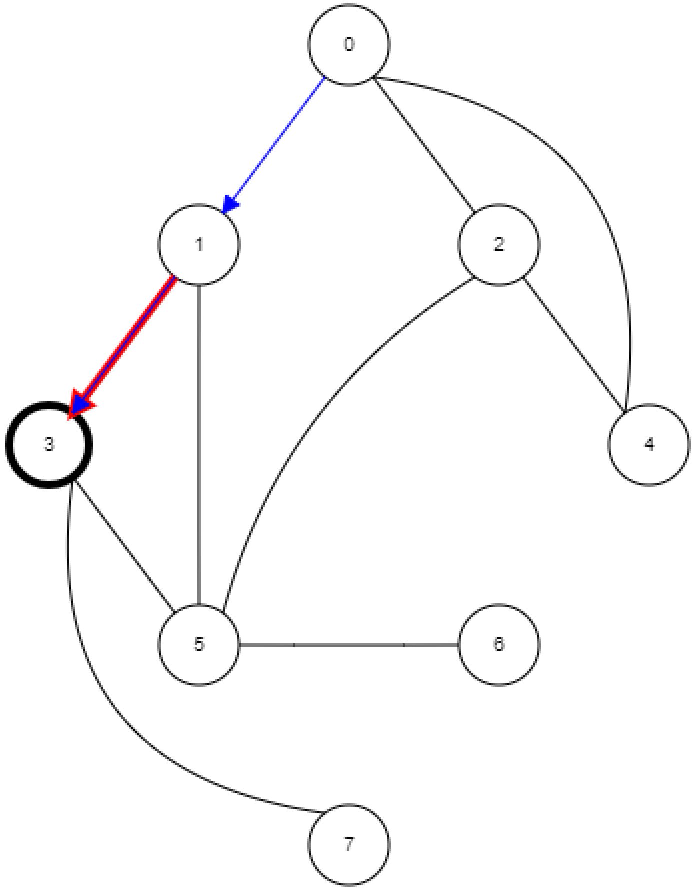
DFS(0)

DFS(1)

DFS(3)

Vertex 1 already visited.

	Parent		Visited
0		0	T
1	0	1	T
2		2	f
3	1	3	T
4		4	f
5		5	f
6		6	f
7		7	f



Depth-First search

DFS(0)

DFS(1)

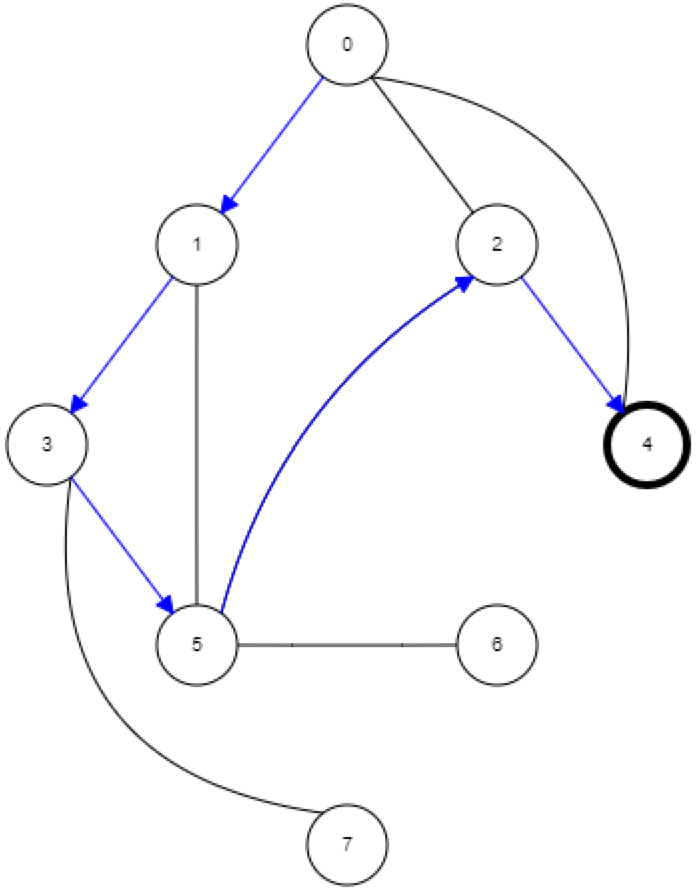
DFS(3)

DFS(5)

DFS(2)

DFS(4)

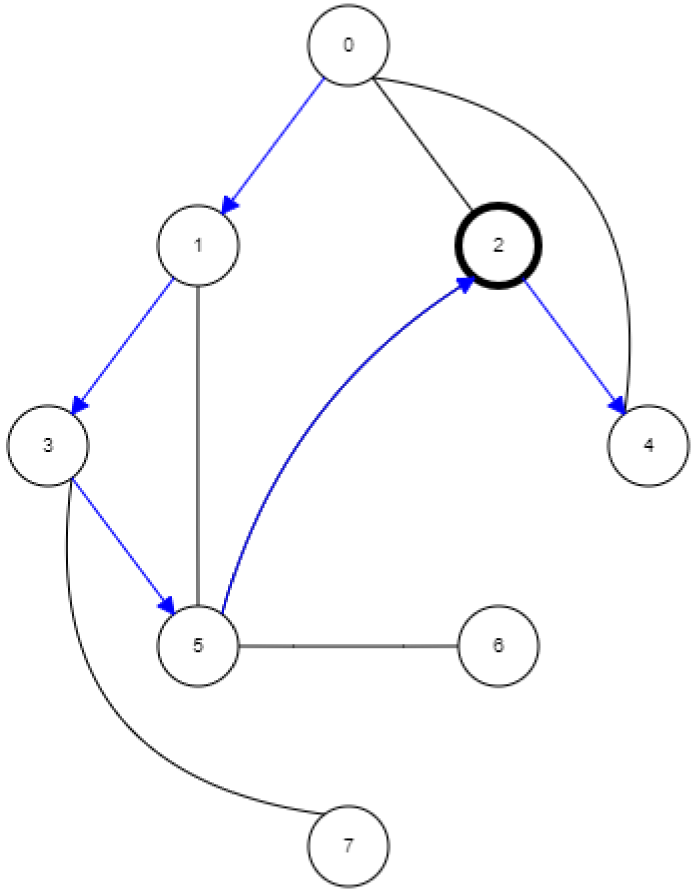
	Parent		Visited
0		0	T
1	0	1	T
2	5	2	T
3	1	3	T
4	2	4	T
5	3	5	T
6		6	f
7		7	f



Depth-First search

DFS(0)
DFS(1)
DFS(3)
DFS(5)
DFS(2)
DFS(4)

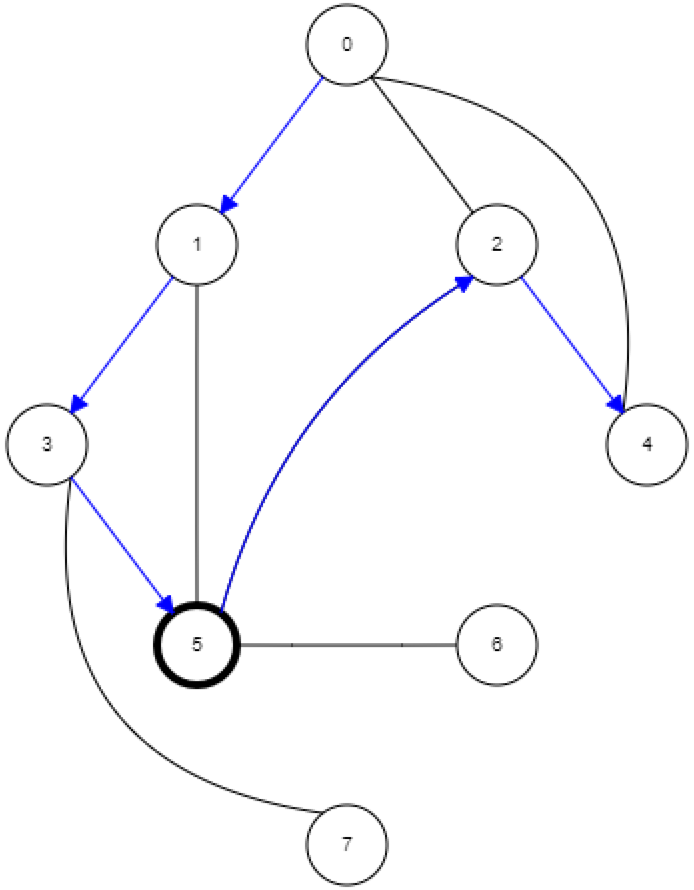
	Parent		Visited
0		0	T
1	0	1	T
2	5	2	T
3	1	3	T
4	2	4	T
5	3	5	T
6		6	f
7		7	f



Depth-First search

DFS(0)
DFS(1)
DFS(3)
DFS(5)
DFS(2)
DFS(4)

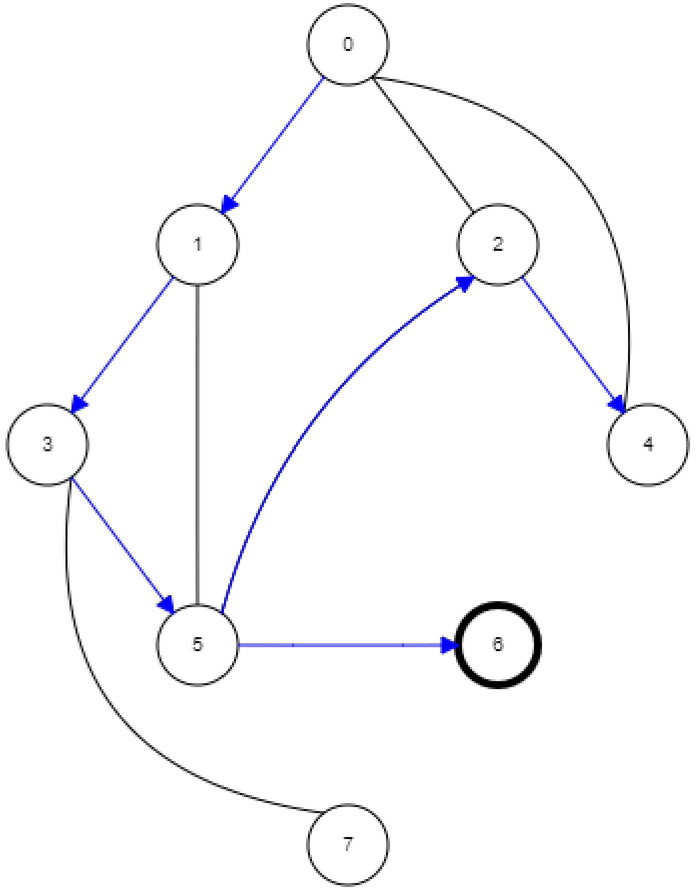
	Parent		Visited
0		0	T
1	0	1	T
2	5	2	T
3	1	3	T
4	2	4	T
5	3	5	T
6		6	f
7		7	f



Depth-First search

DFS(0)
- DFS(1)
  - DFS(3)
    - DFS(5)
      - DFS(2)
        - DFS(4)
      - DFS(6)

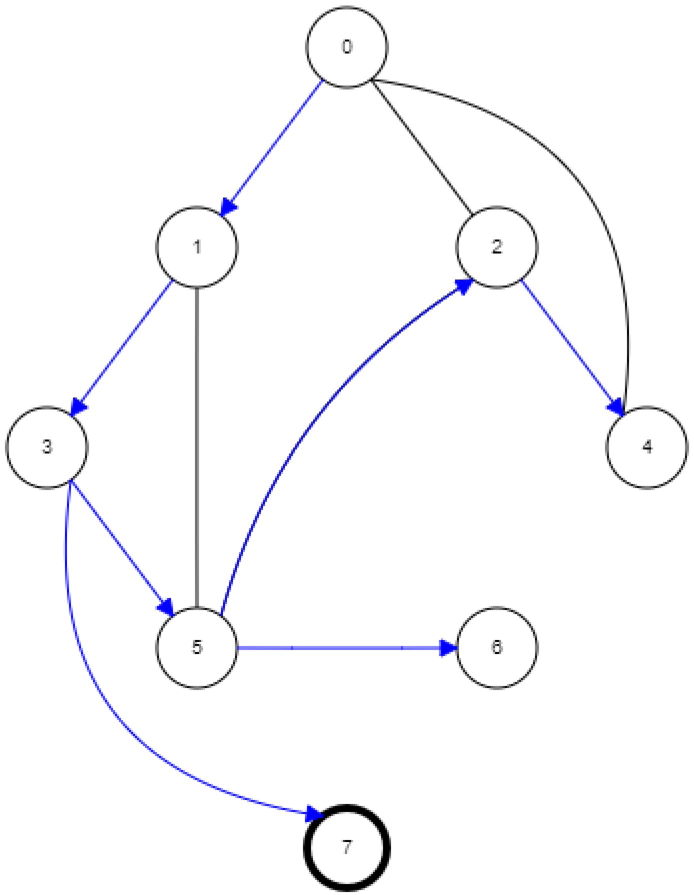
	Parent		Visited
0		0	T
1	0	1	T
2	5	2	T
3	1	3	T
4	2	4	T
5	3	5	T
6	5	6	T
7		7	f



Depth-First search

DFS(0)
    DFS(1)
        DFS(3)
            DFS(5)
                DFS(2)
                    DFS(4)
                DFS(6)
            DFS(7)

	Parent		Visited
0		0	T
1	0	1	T
2	5	2	T
3	1	3	T
4	2	4	T
5	3	5	T
6	5	6	T
7	3	7	T



Depth-First search

DFS(0)

DFS(1)

DFS(3)

DFS(5)

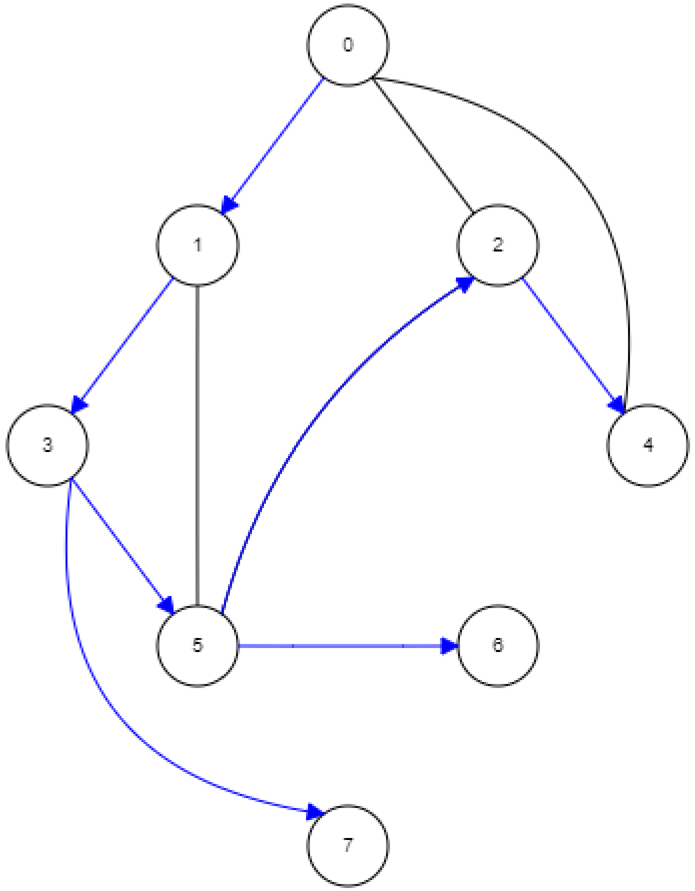
DFS(2)

DFS(4)

DFS(6)

DFS(7)

	Parent		Visited
0		0	T
1	0	1	T
2	5	2	T
3	1	3	T
4	2	4	T
5	3	5	T
6	5	6	T
7	3	7	T



Dijkstra

FORWARD_SEARCH

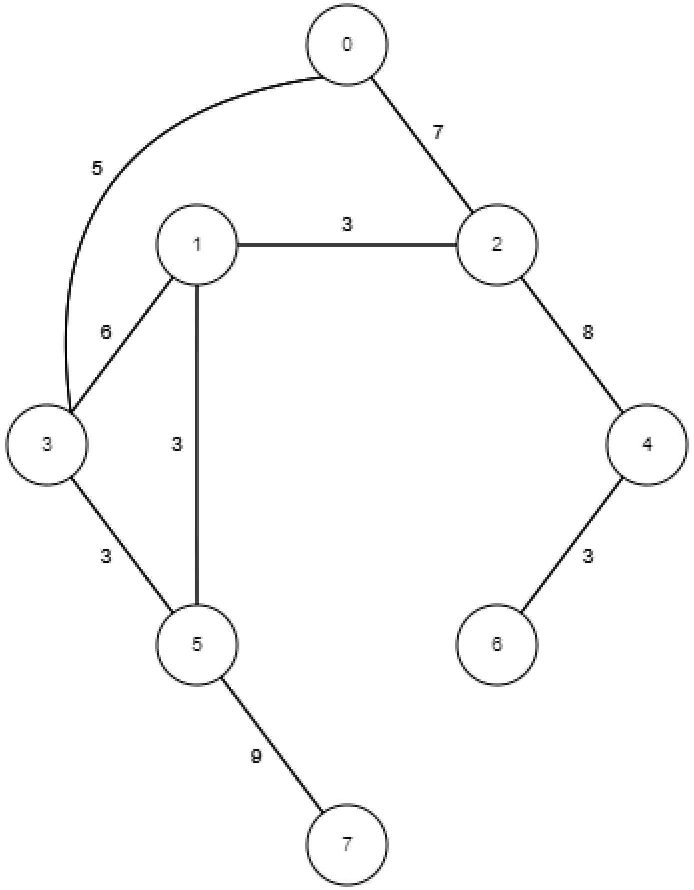
$Q.Insert(x_I)$ and mark x_I as visited
while Q not empty **do**
    $x \leftarrow Q.GetFirst()$
    if $x \in X_G$
        return SUCCESS
    forall $u \in U(x)$
        $x' \leftarrow f(x, u)$
        if x' not visited
            Mark x' as visited
            $Q.Insert(x')$
        else
            Resolve duplicate x'
return FAILURE

Priority queue –
Sorted by cost to come

Update cost to come

Dijkstra

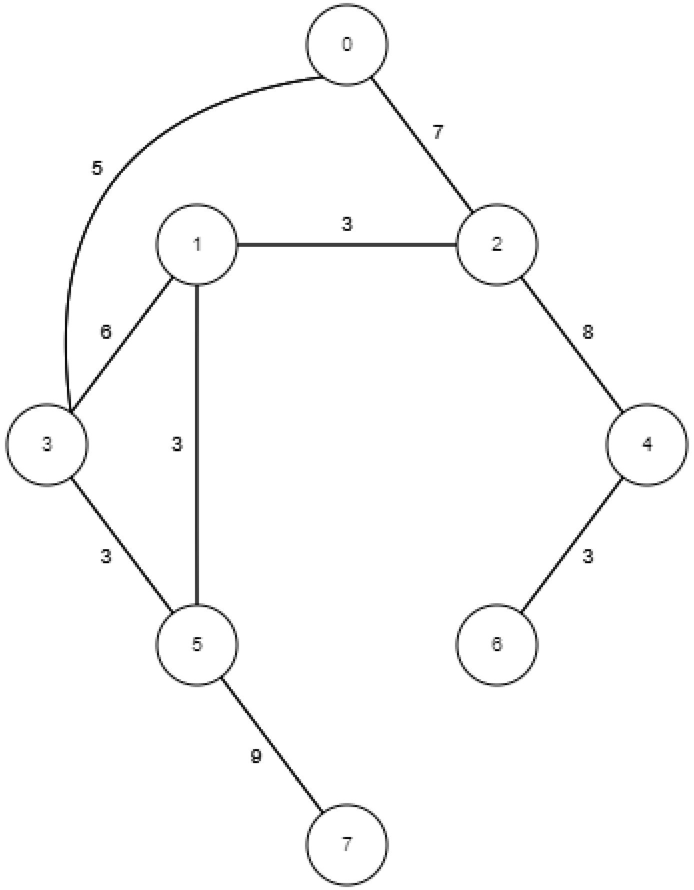
Vertex	Known	Cost	Path
0			
1			
2			
3			
4			
5			
6			
7			



Dijkstra

Finding Cheapest Uknown Vertex

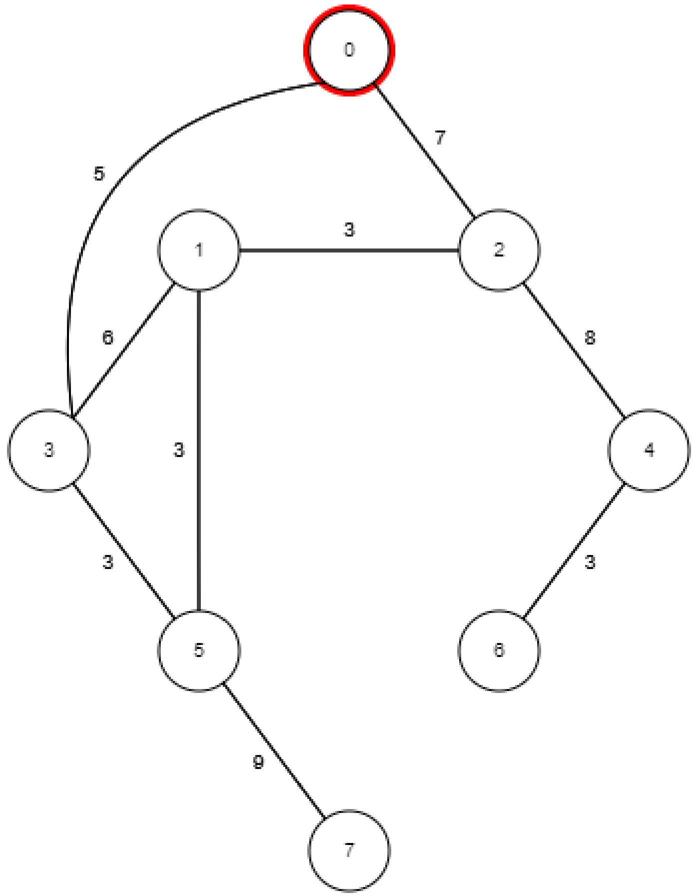
Vertex	Known	Cost	Path
0	F	0	-1
1	F	INF	-1
2	F	INF	-1
3	F	INF	-1
4	F	INF	-1
5	F	INF	-1
6	F	INF	-1
7	F	INF	-1



Dijkstra

Setting known field to True

Vertex	Known	Cost	Path
0	T	0	-1
1	F	INF	-1
2	F	INF	-1
3	F	INF	-1
4	F	INF	-1
5	F	INF	-1
6	F	INF	-1
7	F	INF	-1

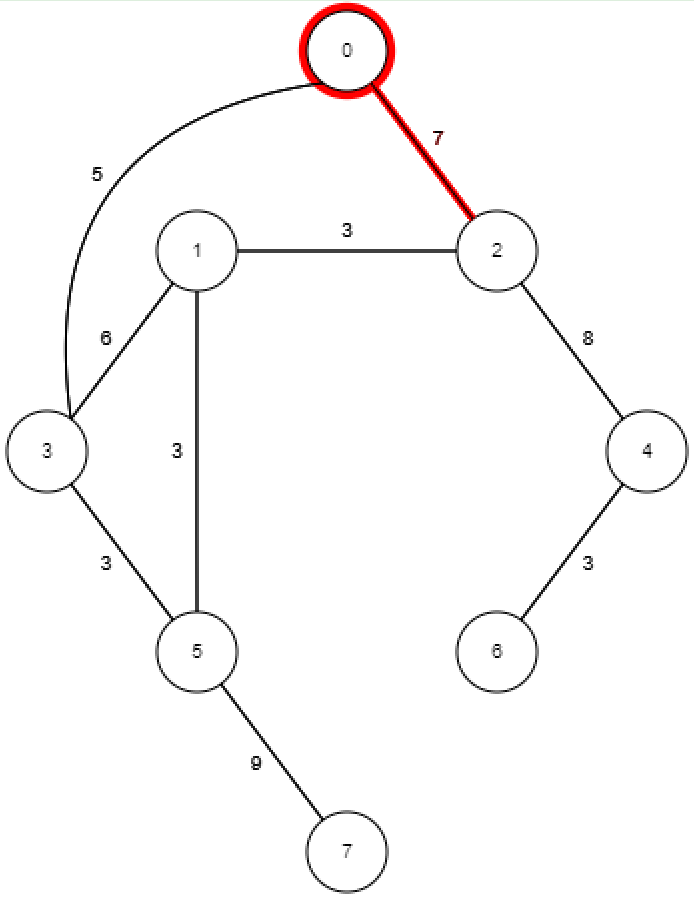


Dijkstra

Updating neighbors of vertex 0

Vertex	Known	Cost	Path
0	T	0	-1
1	F	INF	-1
2	F	INF	-1
3	F	INF	-1
4	F	INF	-1
5	F	INF	-1
6	F	INF	-1
7	F	INF	-1

INF > 0 + 7

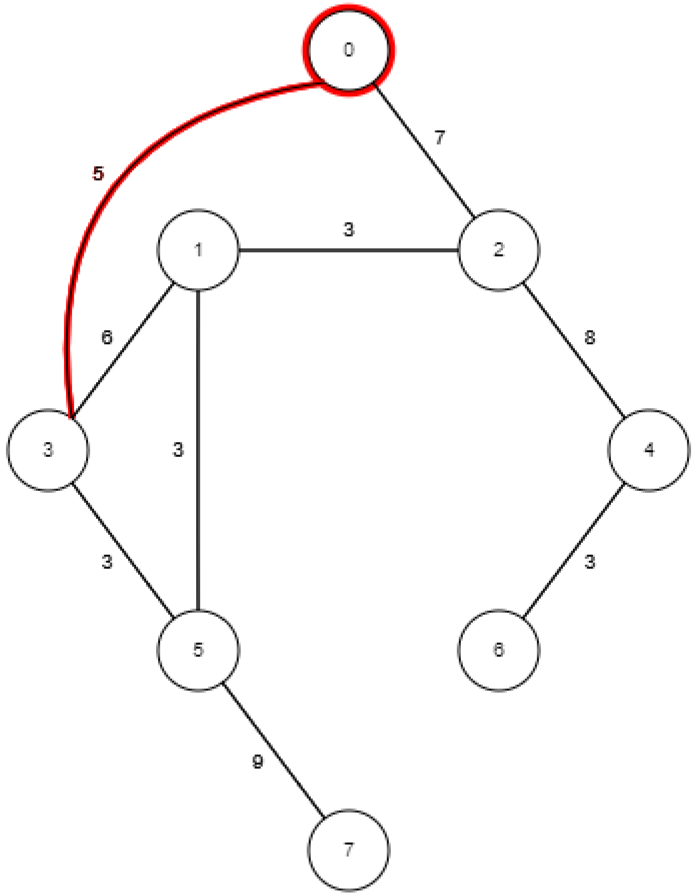


Dijkstra

Updating neighbors of vertex 0

Vertex	Known	Cost	Path
0	T	0	-1
1	F	INF	-1
2	F	7	0
3	F	INF	-1
4	F	INF	-1
5	F	INF	-1
6	F	INF	-1
7	F	INF	-1

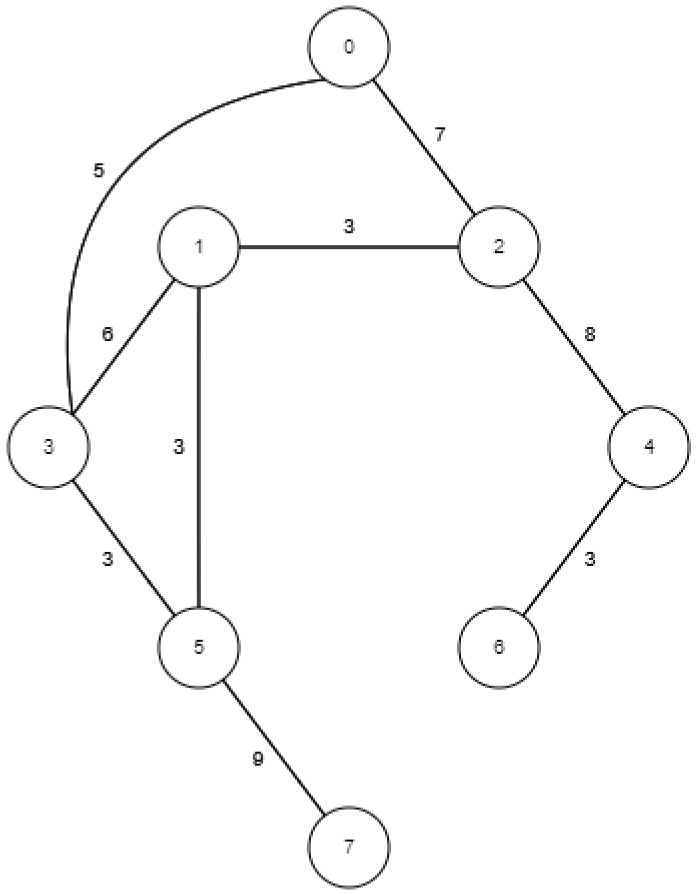
INF > 0 + 5



Dijkstra

Finding Cheapest Uknown Vertex

Vertex	Known	Cost	Path
0	T	0	-1
1	F	INF	-1
2	F	7	0
3	F	5	0
4	F	INF	-1
5	F	INF	-1
6	F	INF	-1
7	F	INF	-1

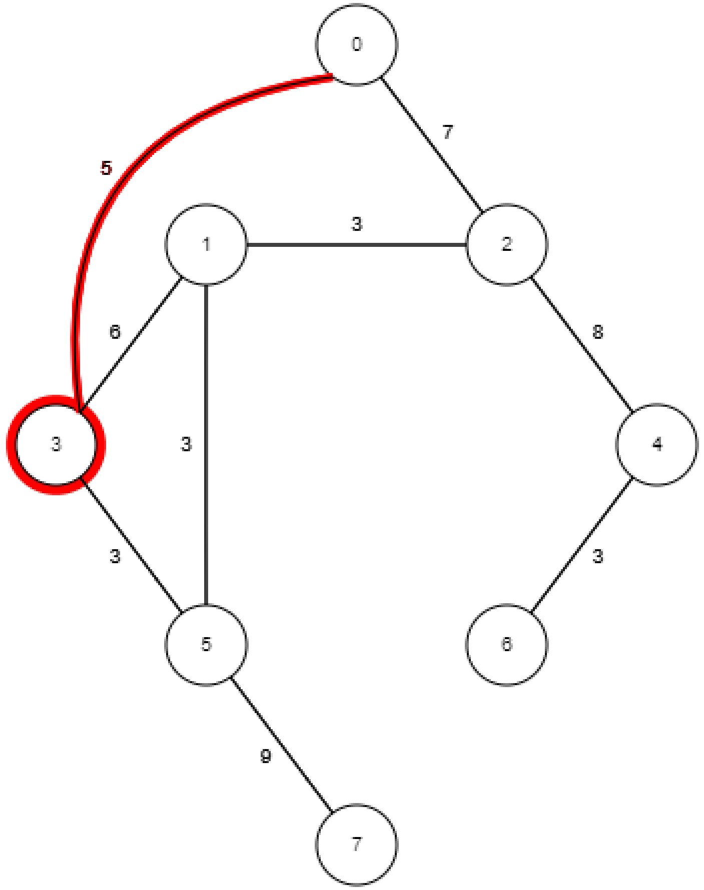


Dijkstra

Updating neighbors of vertex 3

Vertex	Known	Cost	Path
0	T	0	-1
1	F	INF	-1
2	F	7	0
3	T	5	0
4	F	INF	-1
5	F	INF	-1
6	F	INF	-1
7	F	INF	-1

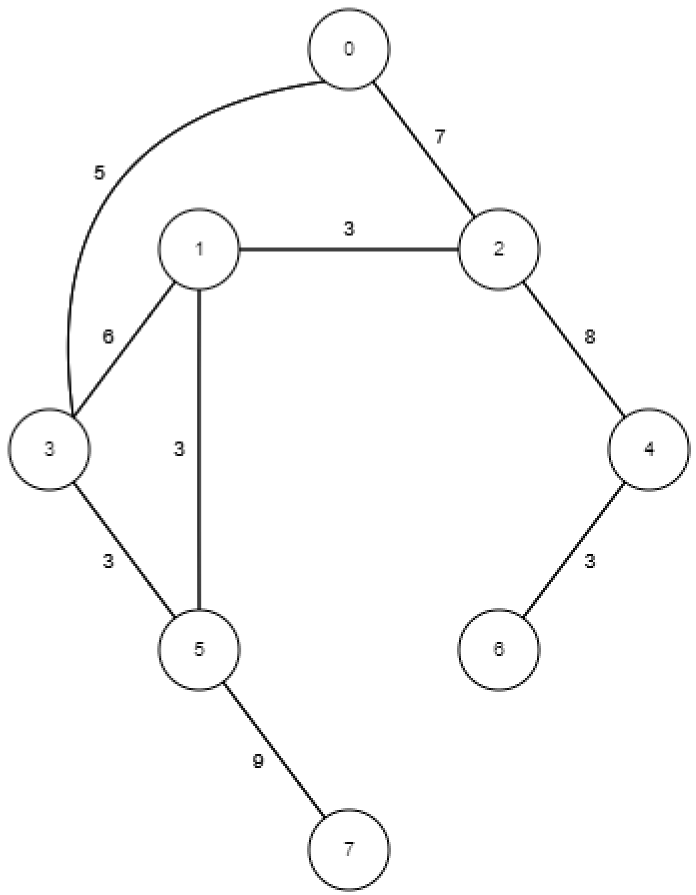
Vertex 0 known



Dijkstra

Finding Cheapest Uknown Vertex

Vertex	Known	Cost	Path
0	T	0	-1
1	F	11	3
2	F	7	0
3	T	5	0
4	F	INF	-1
5	F	8	3
6	F	INF	-1
7	F	INF	-1

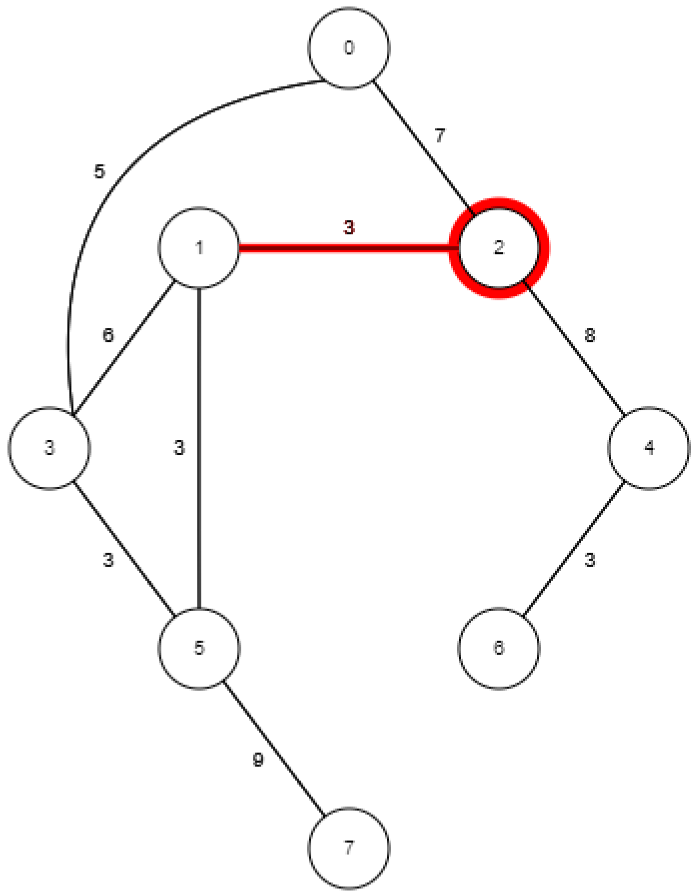


Dijkstra

Updating neighbors of vertex 2

Vertex	Known	Cost	Path
0	T	0	-1
1	F	11	3
2	T	7	0
3	T	5	0
4	F	INF	-1
5	F	8	3
6	F	INF	-1
7	F	INF	-1

11 > 7 + 3

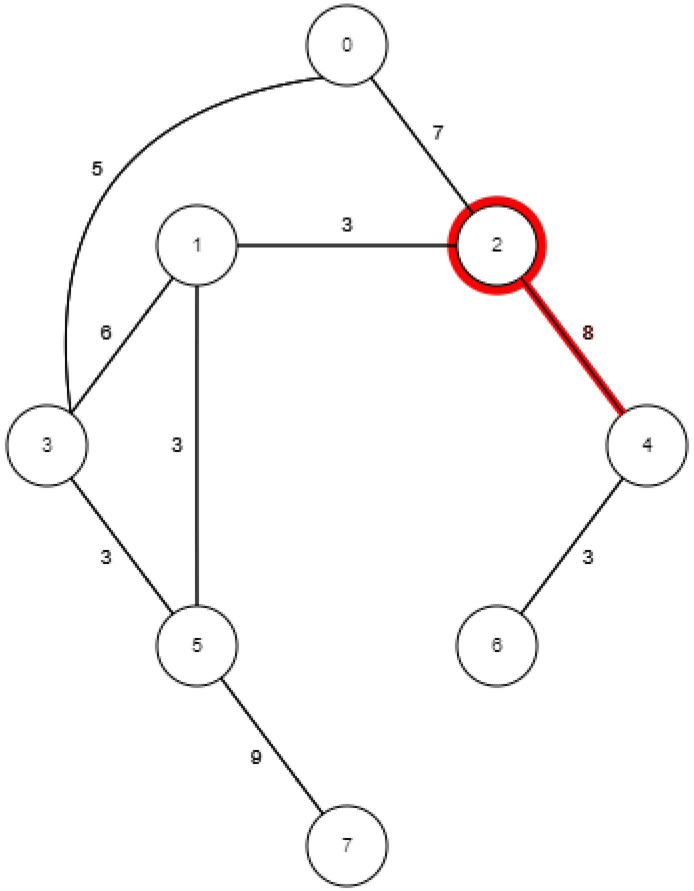


Dijkstra

Updating neighbors of vertex 2

Vertex	Known	Cost	Path
0	T	0	-1
1	F	10	2
2	T	$\boxed{7}$	0
3	T	5	0
4	F	$\boxed{INF}$	-1
5	F	8	3
6	F	INF	-1
7	F	INF	-1

INF > 7 + 8

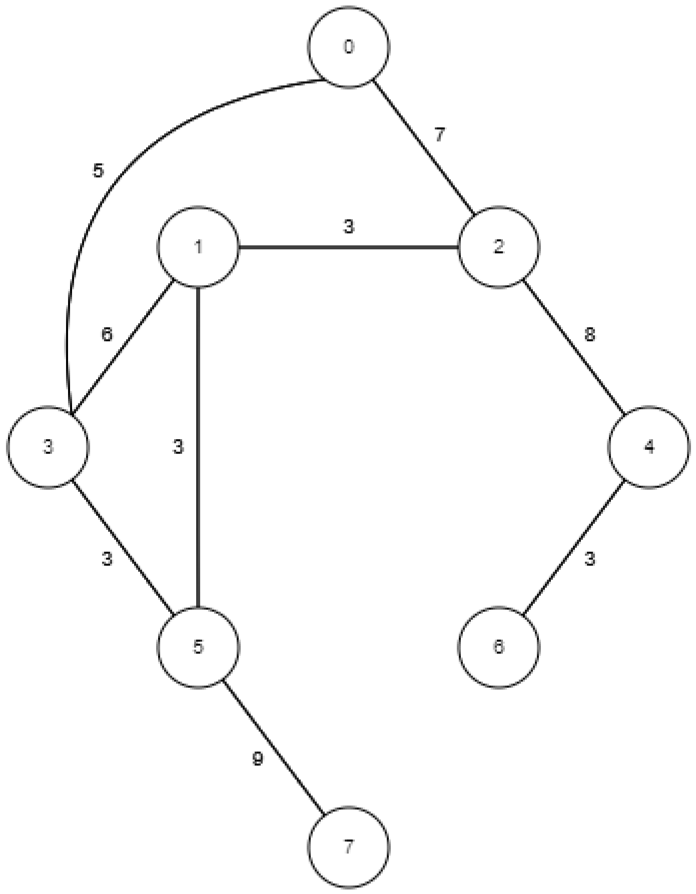


Dijkstra

Finding Paths in Table

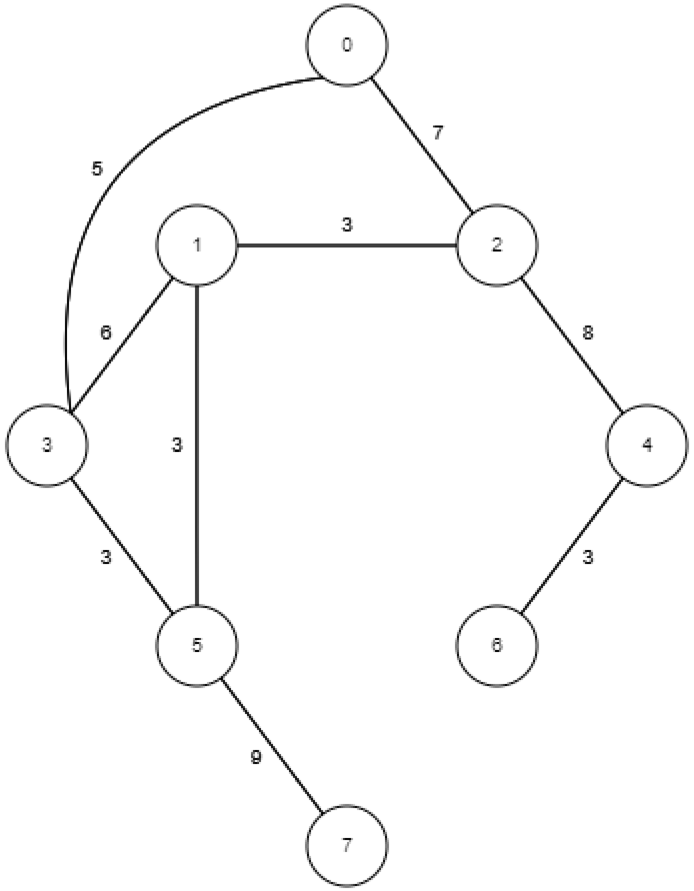
Vertex	Known	Cost	Path
0	T	0	-1
1	T	10	2
2	T	7	0
3	T	5	0
4	T	15	2
5	T	8	3
6	T	18	4
7	T	17	5

0

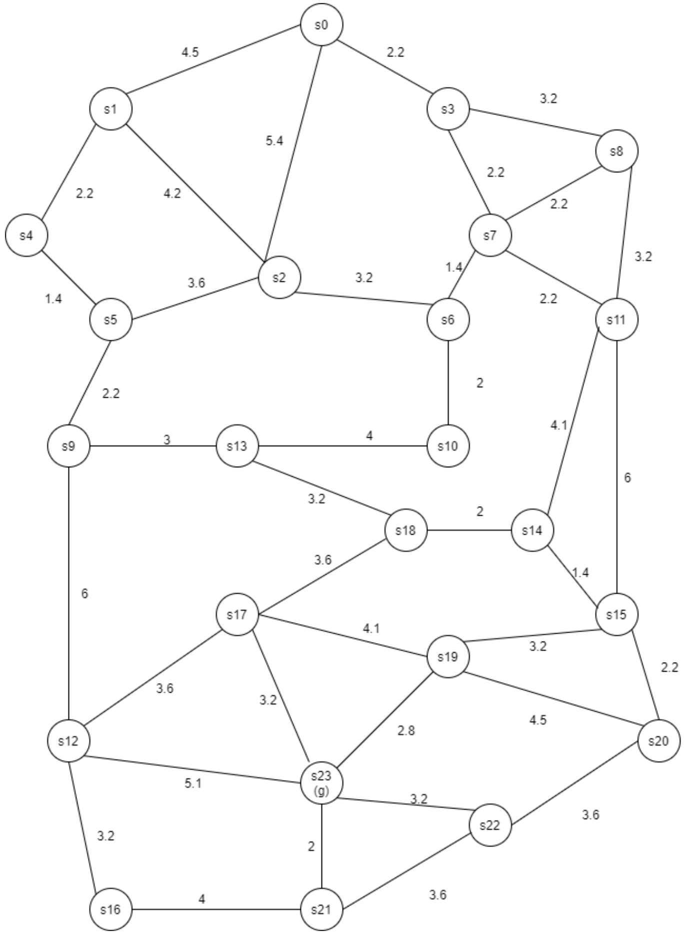


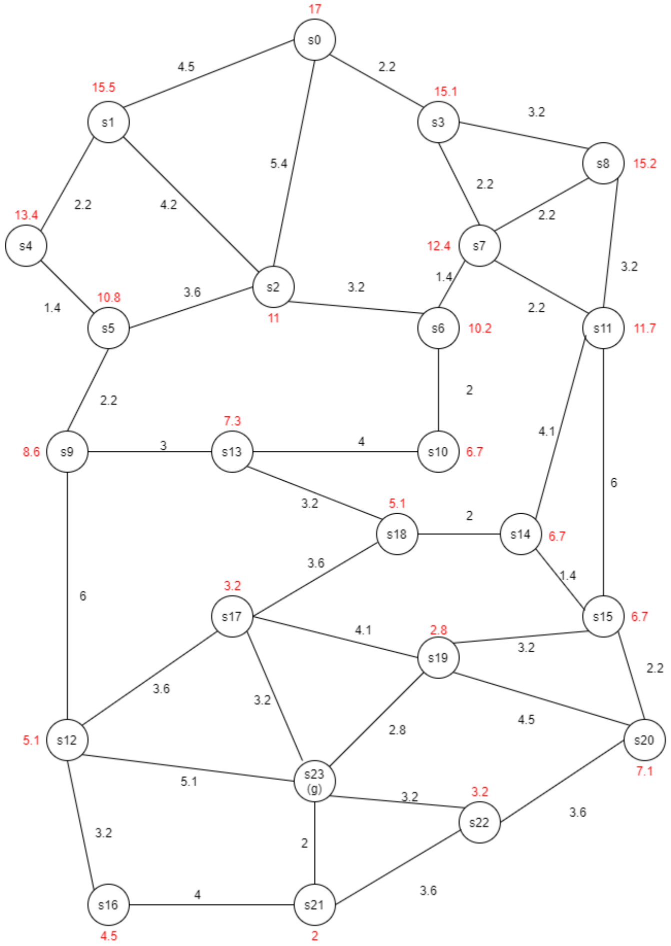
Dijkstra

Vertex	Known	Cost	Path	
0	T	0	-1	0
1	T	10	2	0 2 1
2	T	7	0	0 2
3	T	5	0	0 3
4	T	15	2	0 2 4
5	T	8	3	0 3 5
6	T	18	4	0 2 4 6
7	T	17	5	0 3 5 7

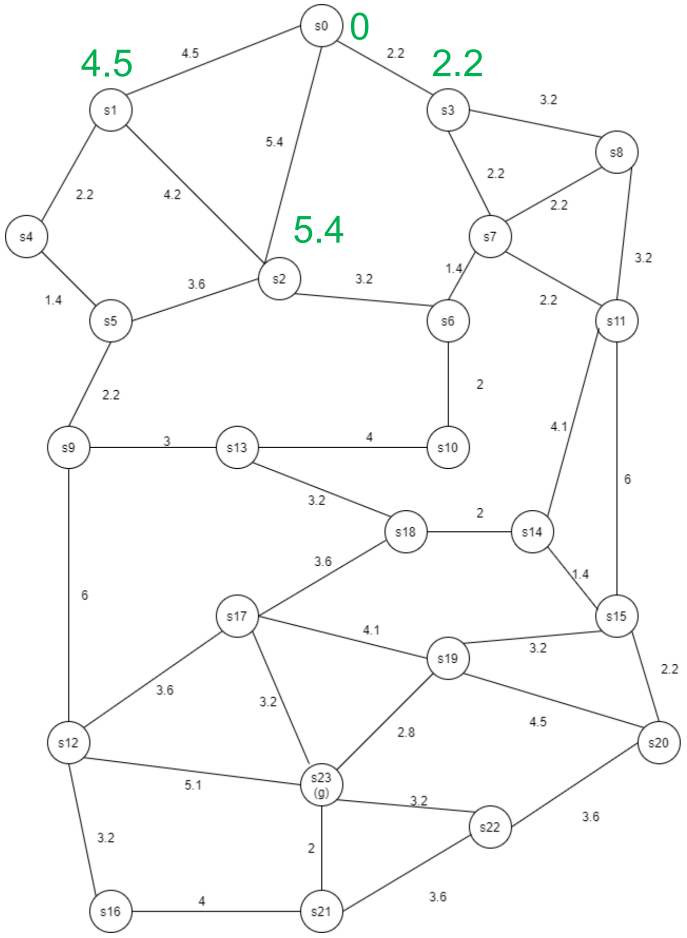


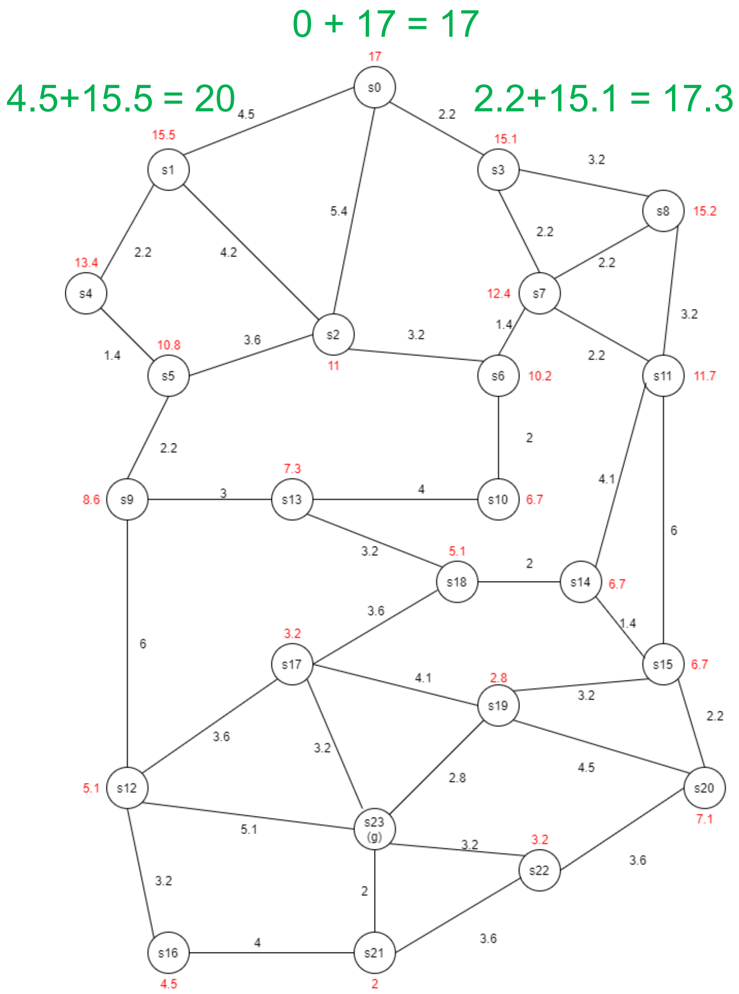
Dijkstra vs. A*



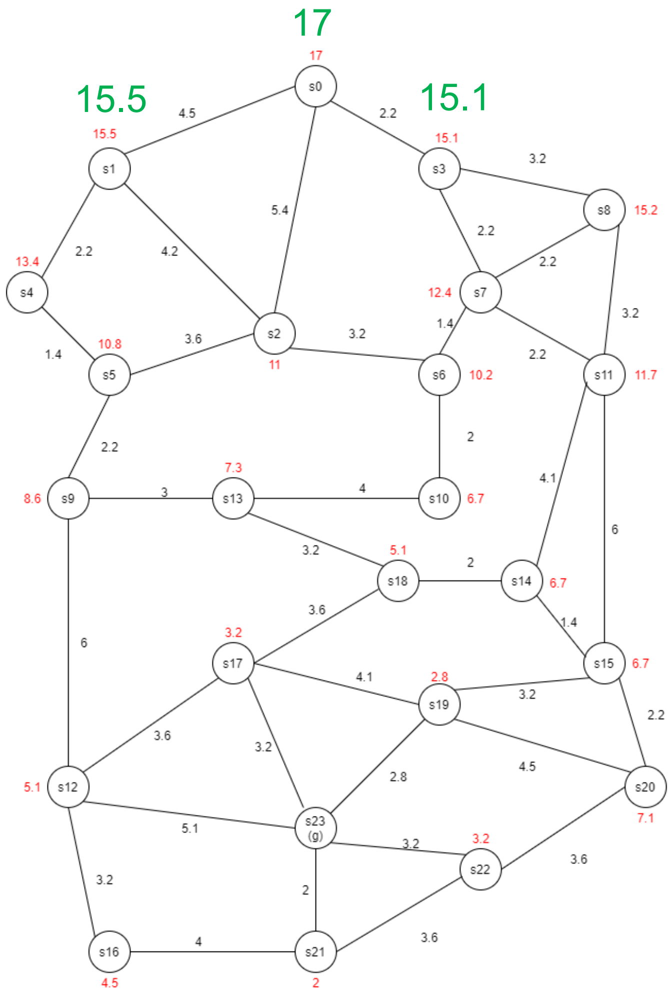


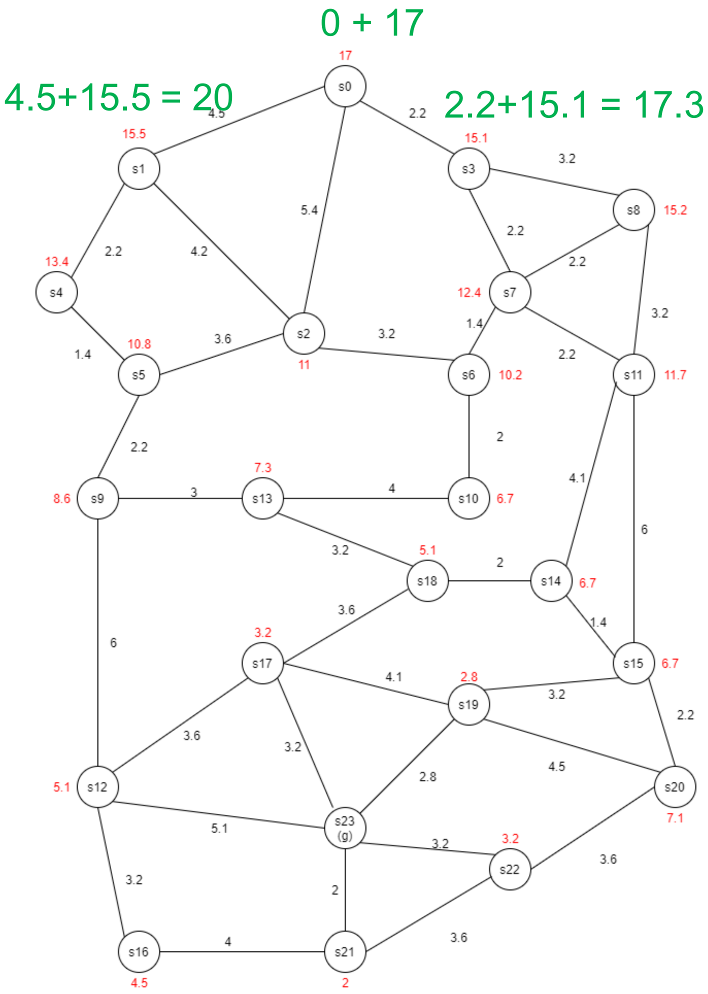
Dijkstra vs. A*





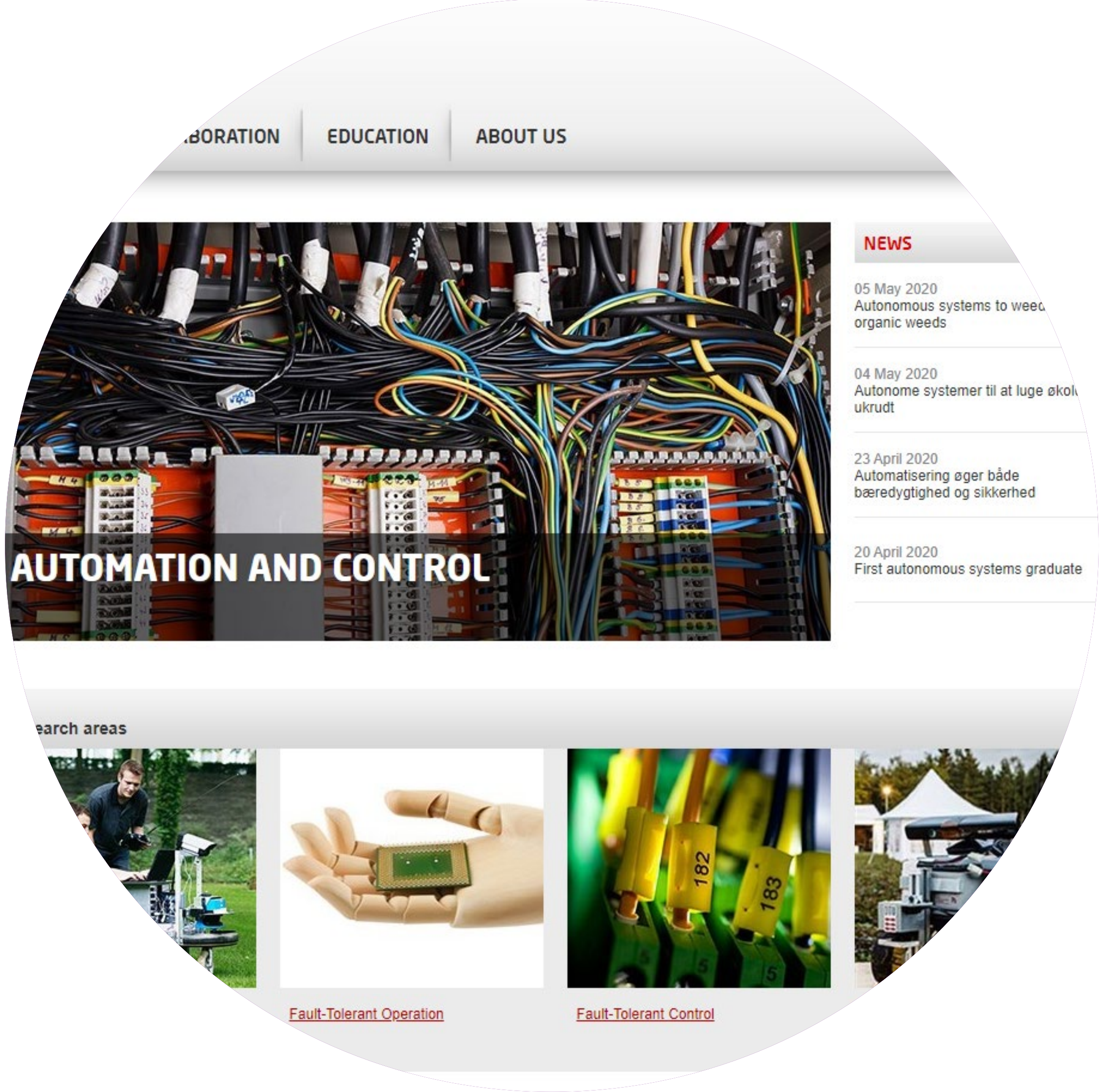
(Greedy) Best-first vs. A*





Algorithm comparison

	BFS	DFS	Dijkstra	A*	(Greedy) Best-first
Sorting mechanism	FIFO (fist in, first out)	LIFO (last in, first out)	Priority queue using calculated cost to come	Priority queue using calculated cost to come plus heuristic	Priority queue using just a heuristic
Feasibility guaranteed	Always	For finite state spaces	Always	Always	Always
Optimality guaranteed	No	No	Yes	Yes	No



CHRISTOPHER REINARTZ
PHD STUDENT
DEPARTMENT OF ELECTRICAL ENGINEERING
TECHNICAL UNIVERSITY OF DENMARK

CCREIN@ELEKTRO.DTU.DK

AUT.ELEKTRO.DTU.DK