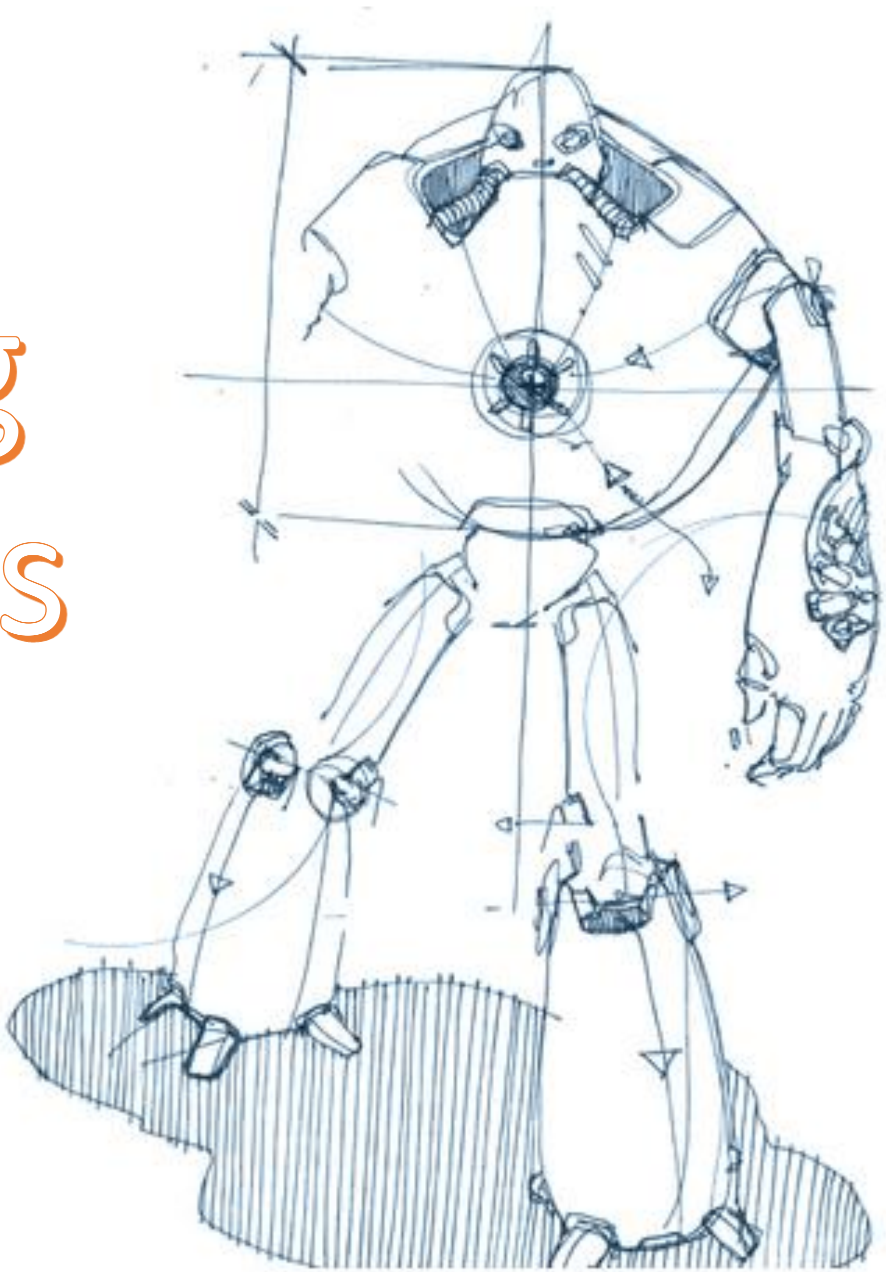
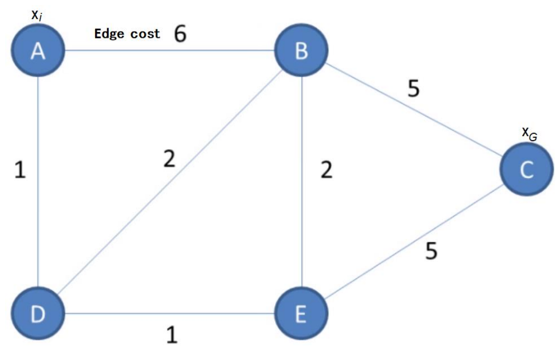
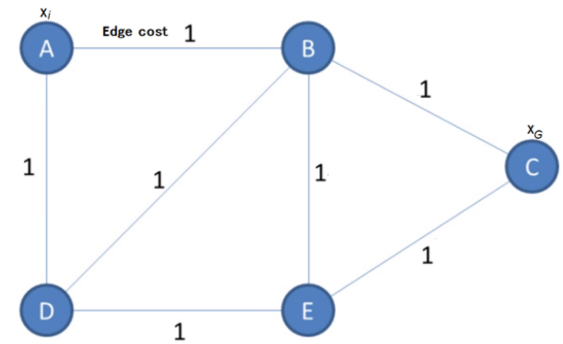
Searching Algorithms



Dr. Mustafa Shiple

Planning





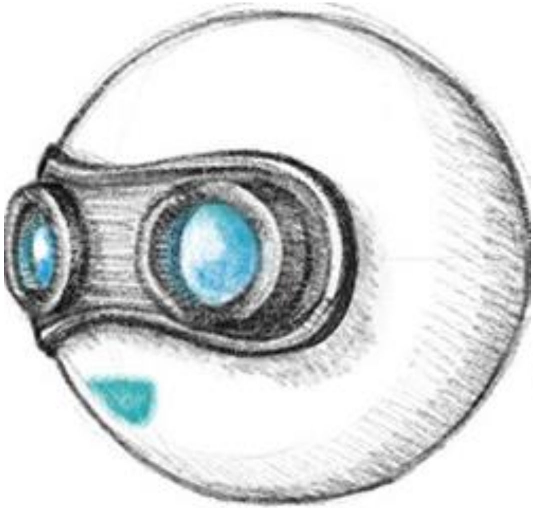
Basis of comparison	Informed search	Uninformed search
Basic knowledge	Uses knowledge to find the steps to the solution.	No use of knowledge
Efficiency	Highly efficient as consumes less time and cost.	Efficiency is mediatory
Cost	Low	Comparatively high
Performance	Finds the solution more quickly.	Speed is slower than the informed search.
Algorithms	Heuristic depth-first and A* search	Depth-first search, breadth-first search, and lowest cost first search

Planning

- Definition: is the process of determining a sequence of actions and motions, by looking ahead.

Search techniques :

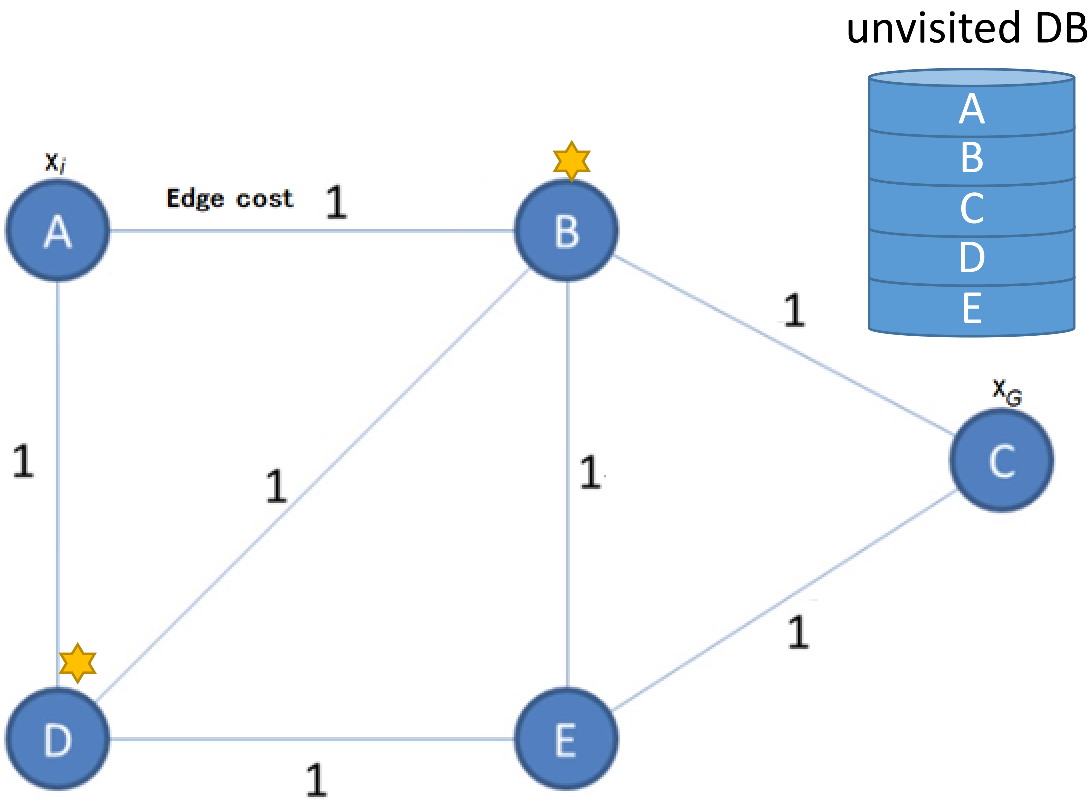
Breadth First Search (BFS)

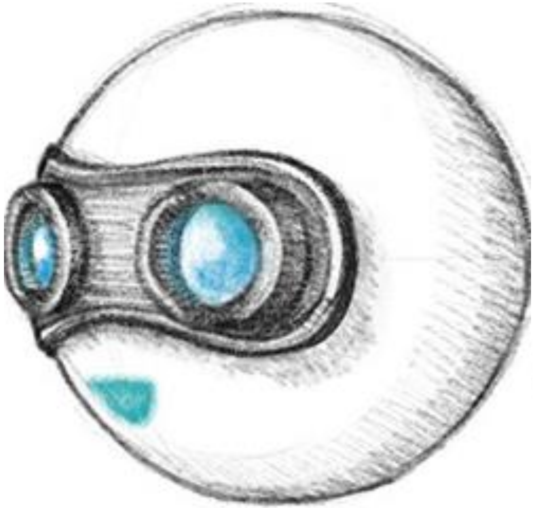


Breadth-first search (BFS)

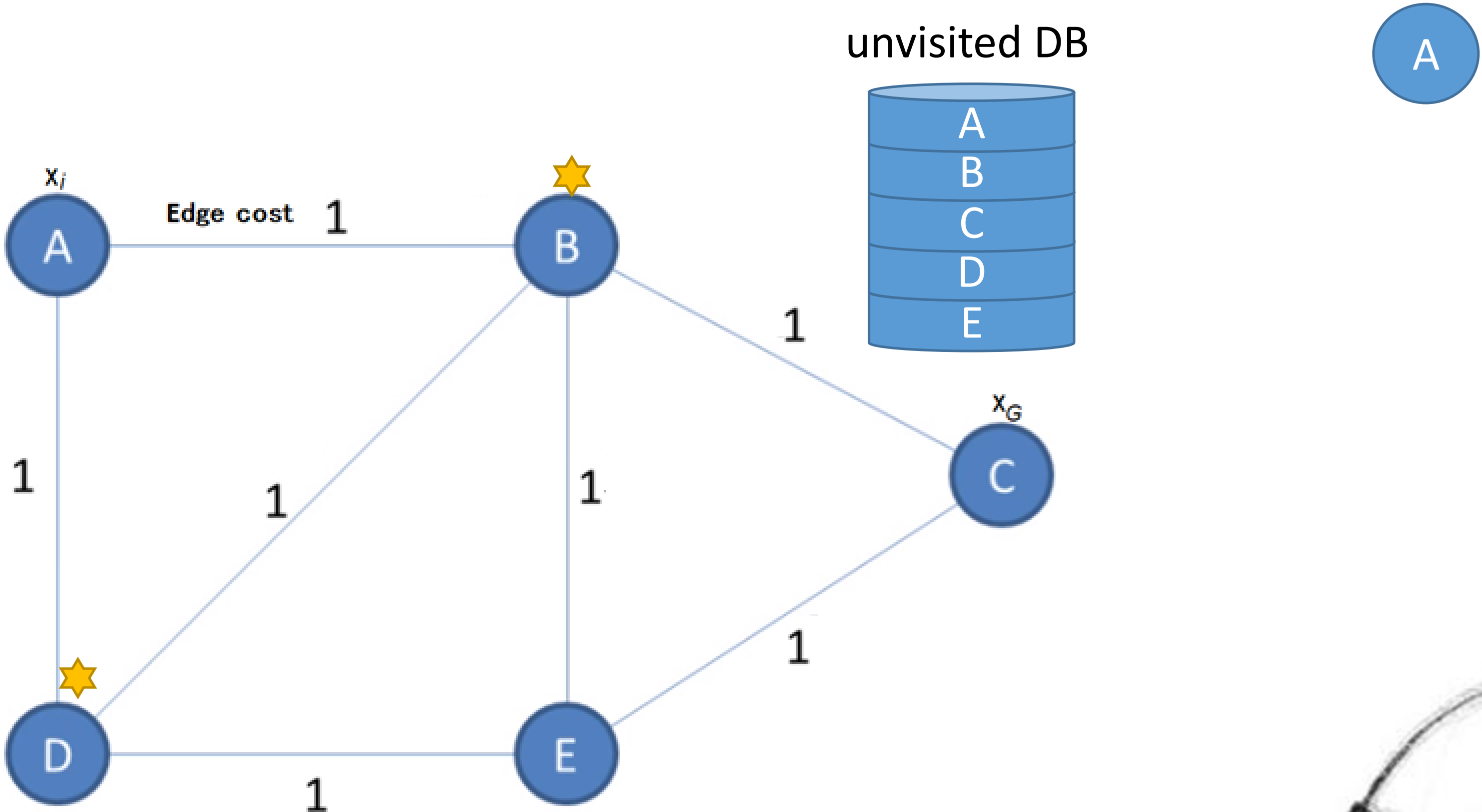
- Civil engineer, pioneering computer scientist, inventor and businessman.
- The functional program-controlled Turing-complete Z3 (First digital computer in the world).
- He designed Plankalkül, the first high-level programming language.
- BFS invented in 1945, in his (rejected for 400 Mark) Ph.D. thesis on the Plankalkül programming language

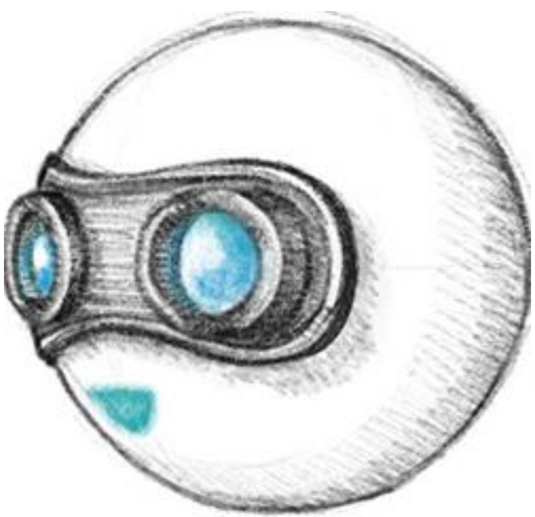
BFS :



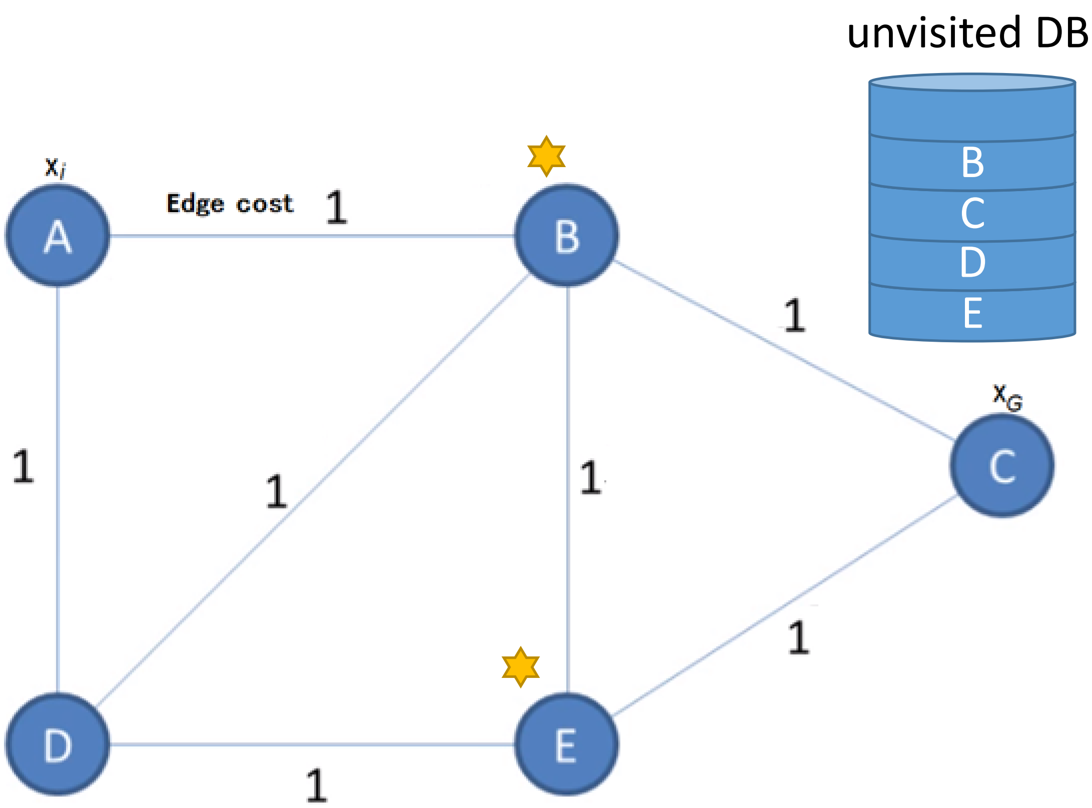


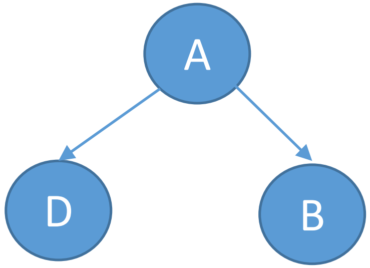
BFS :

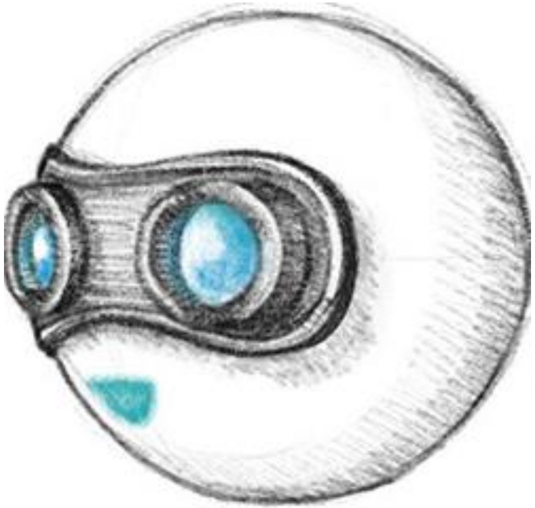




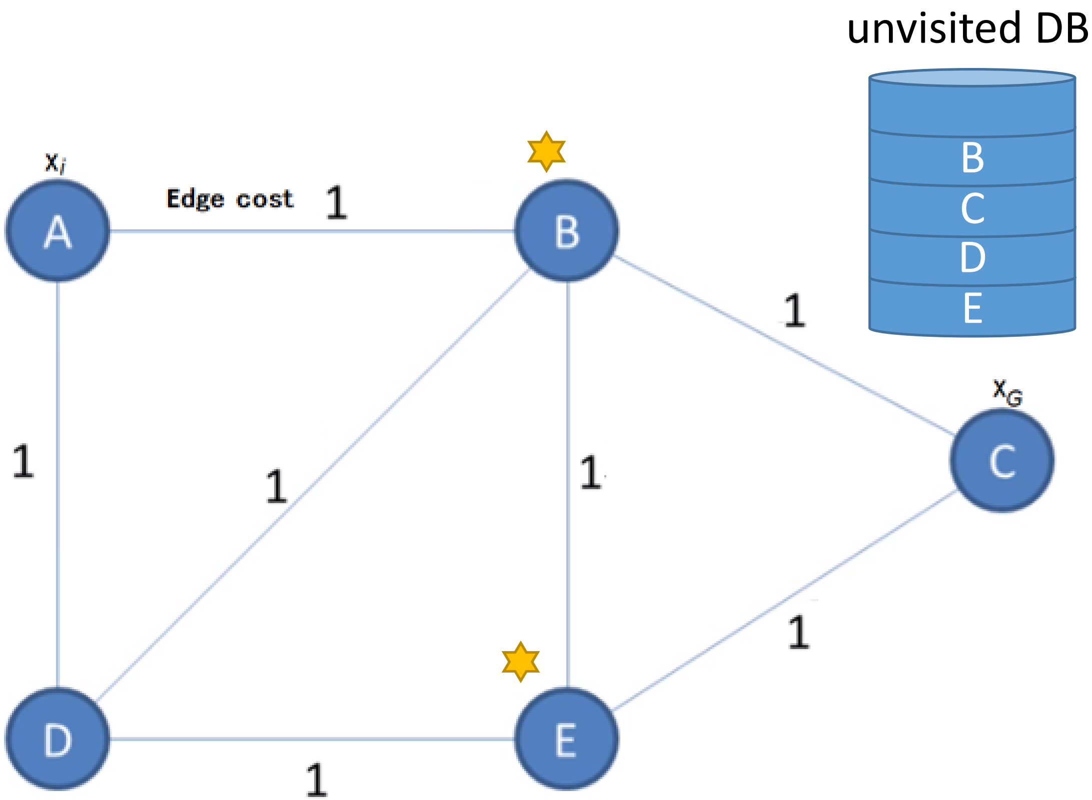
BFS :

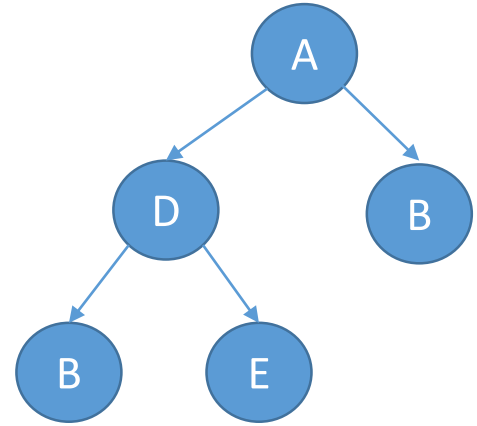


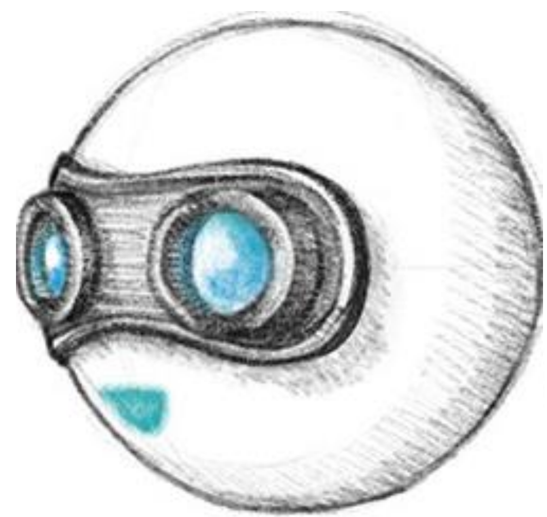




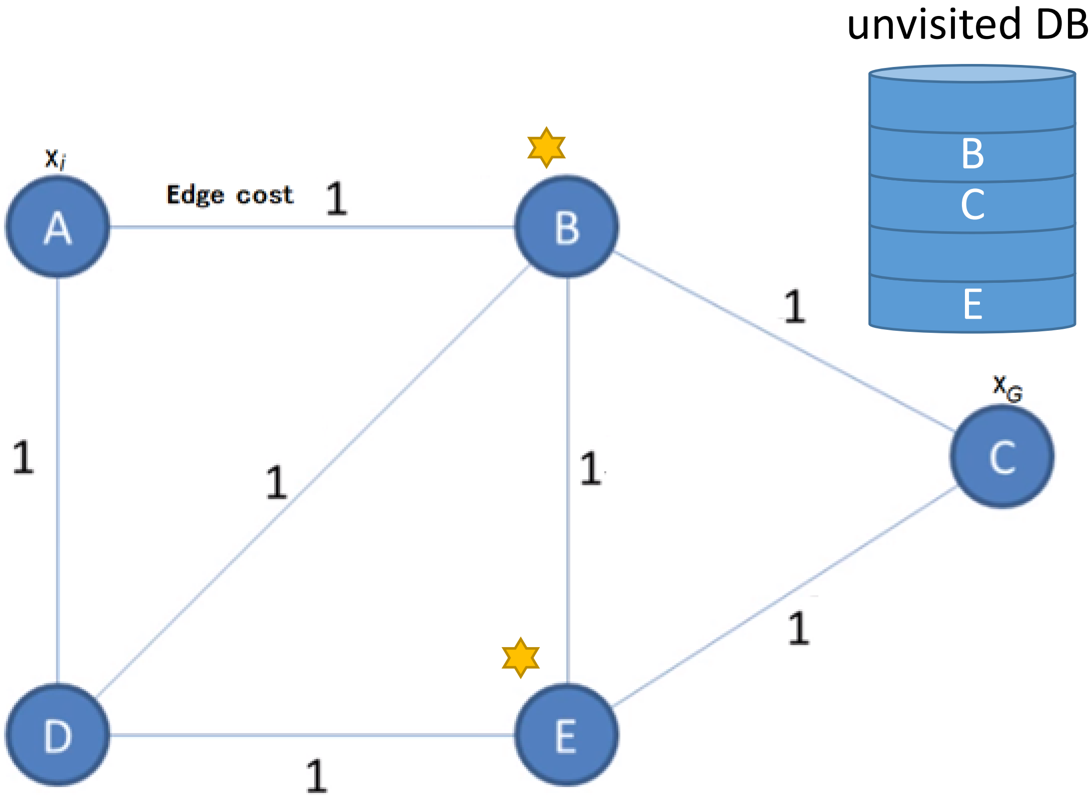
BFS :

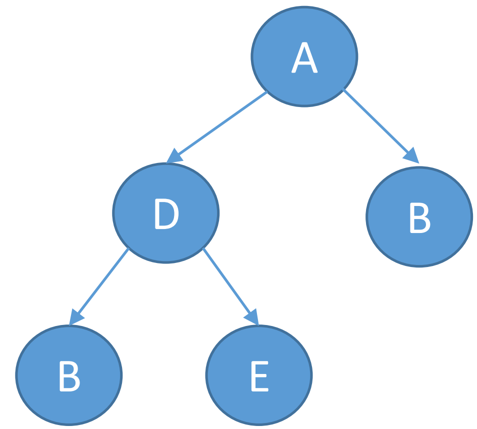


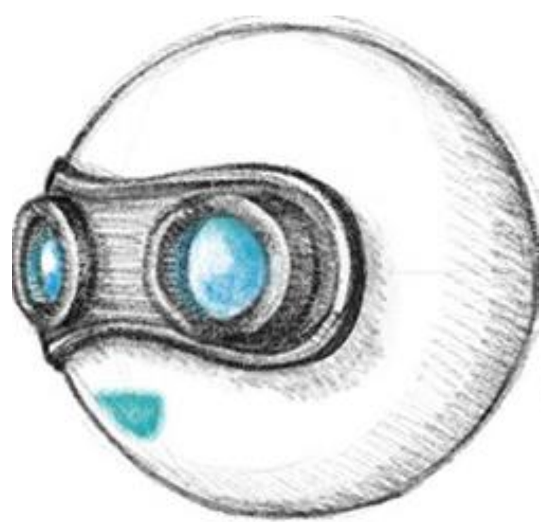




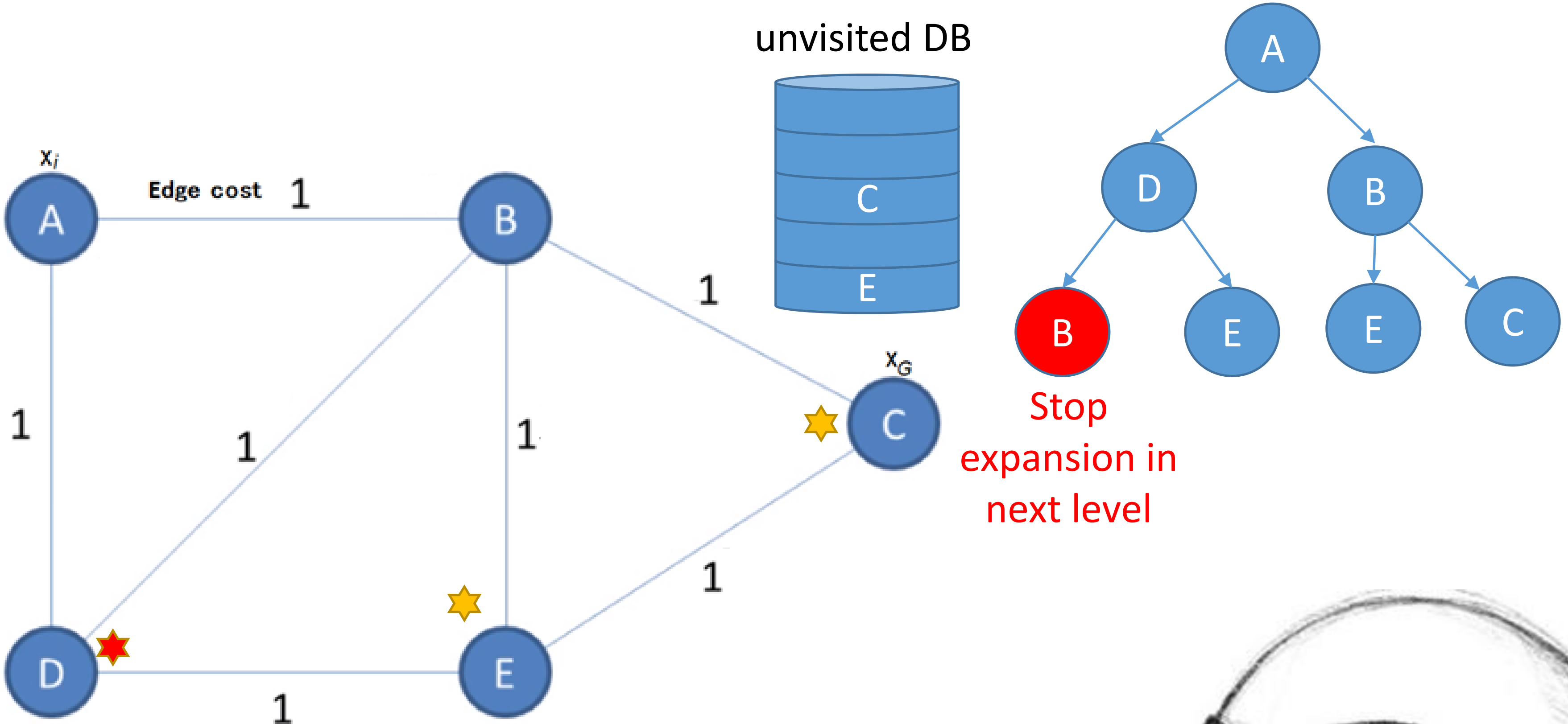
BFS :

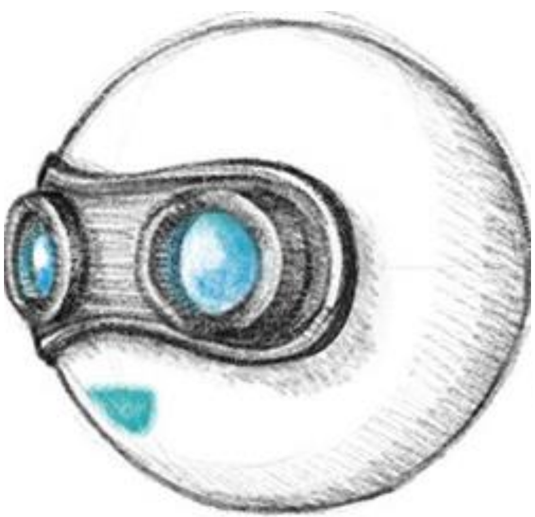




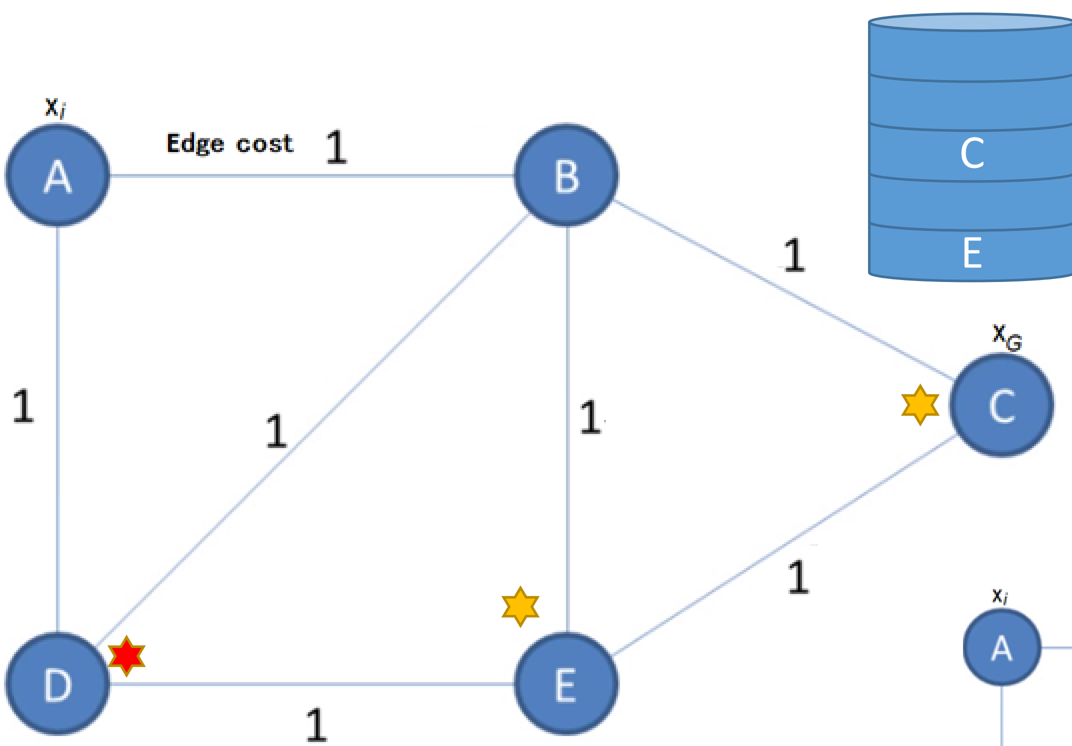


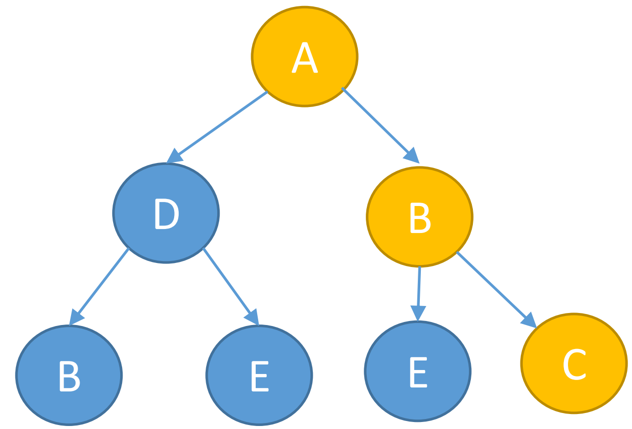
BFS :


unvisited DB
C
E
x_i
A
Edge cost 1
B
1
1
1
1
x_G
C
D
E
1
1
A
D
B
B
E
E
C
Stop expansion in next level

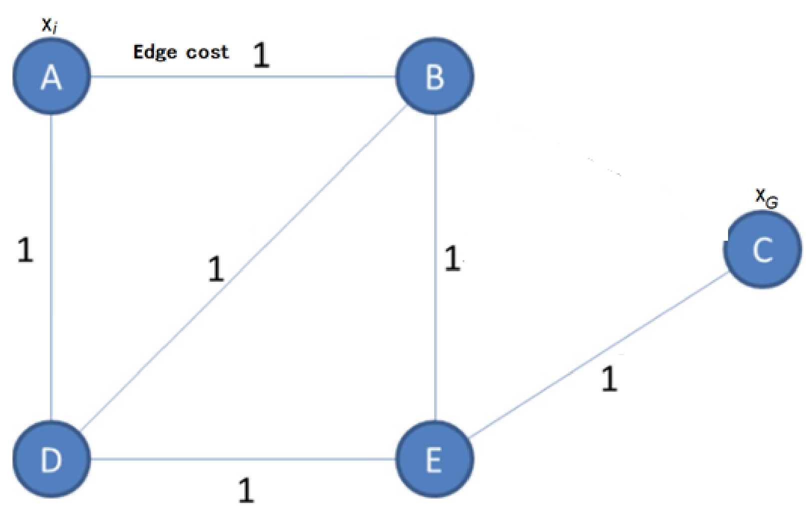


BFS :

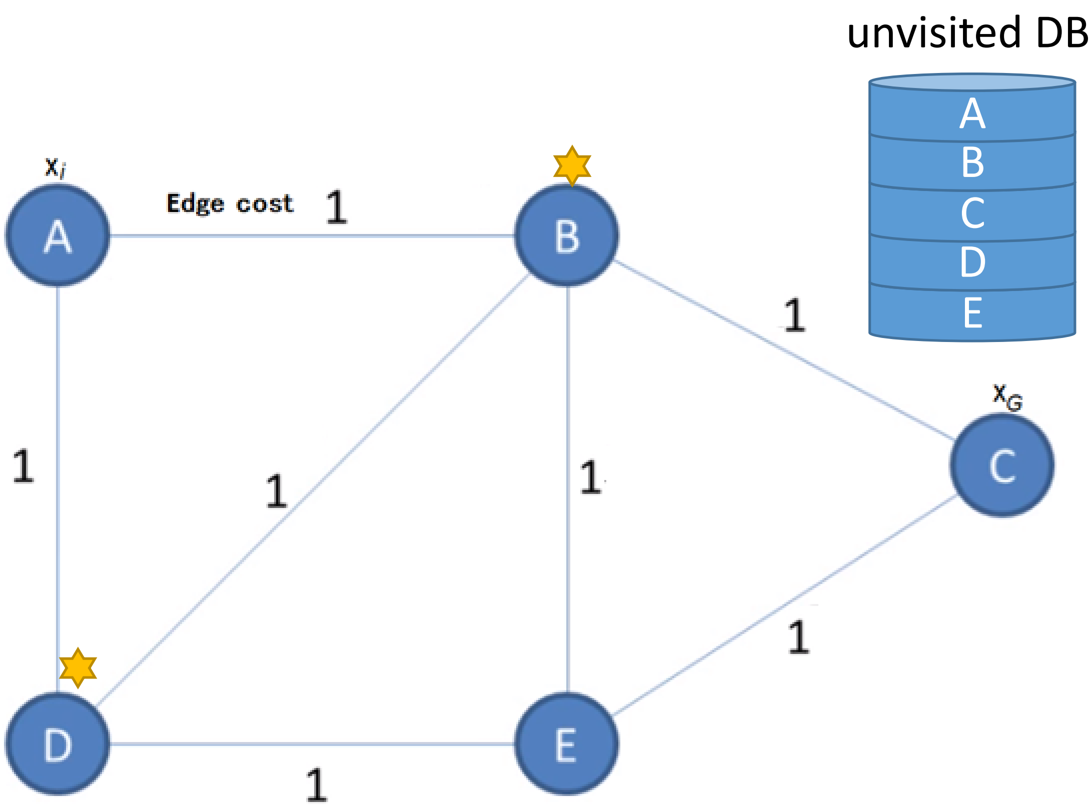


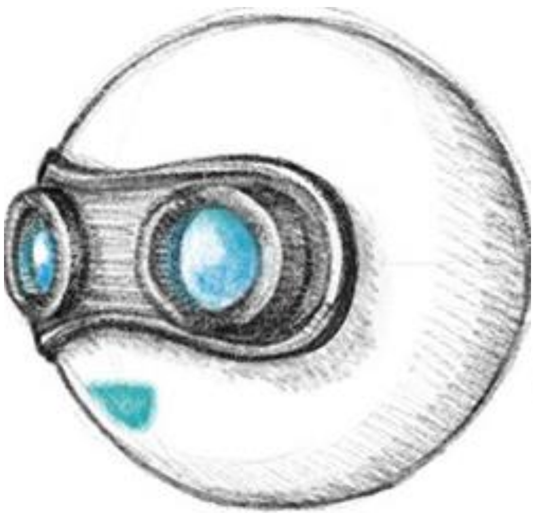


Early exit

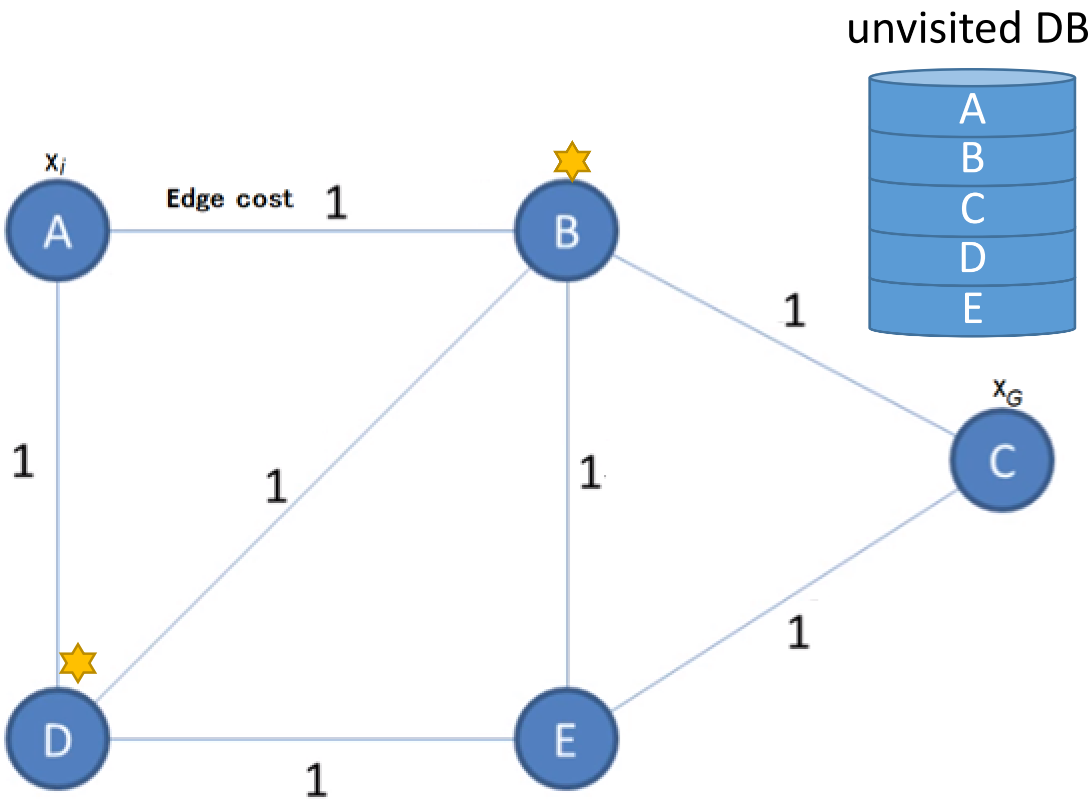


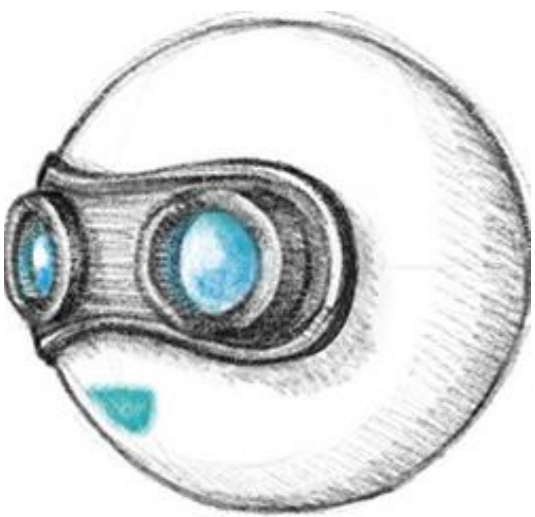
Depth-first search (DFS):



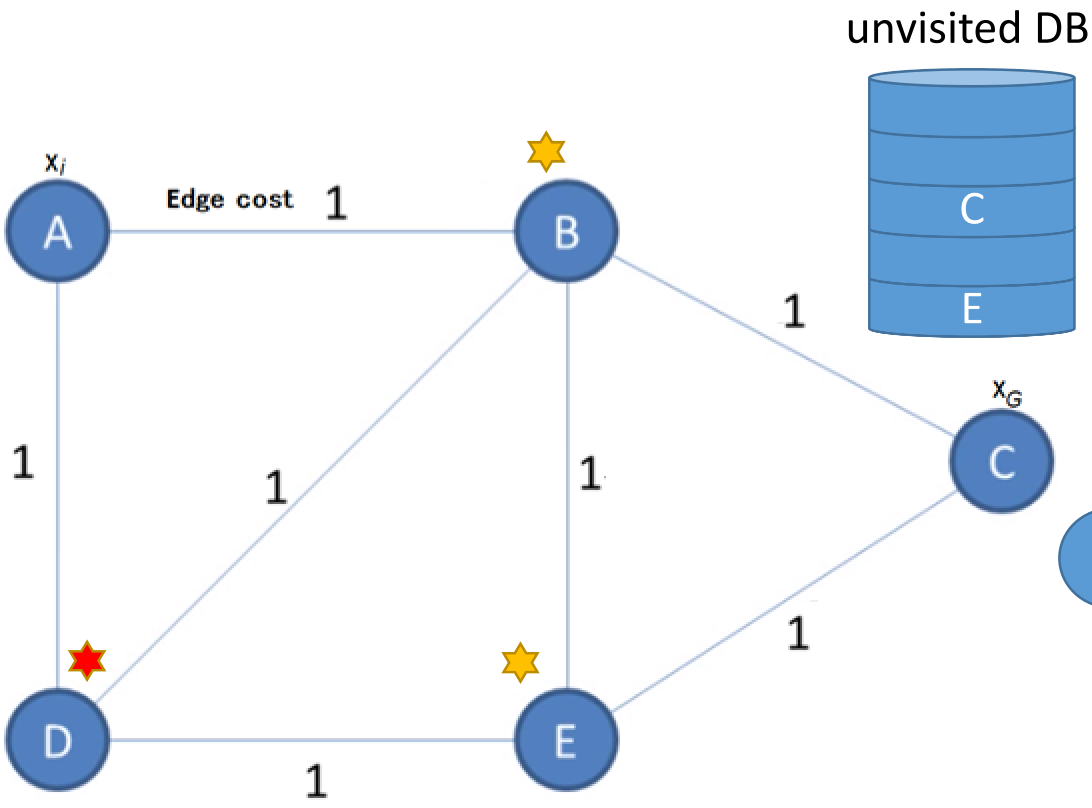


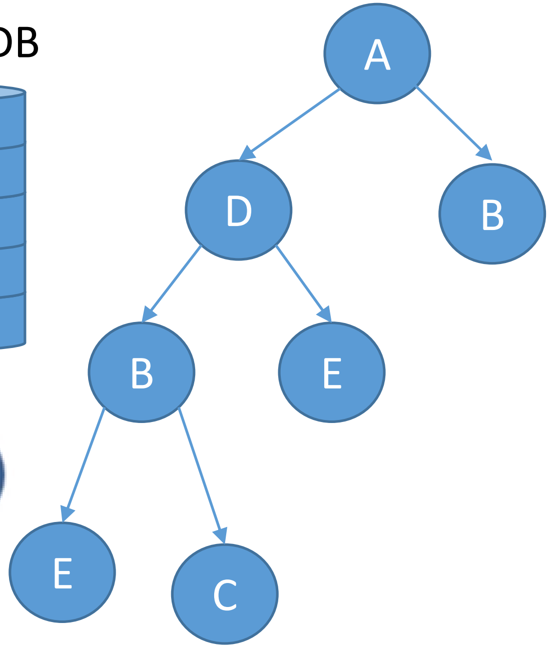
DFS :

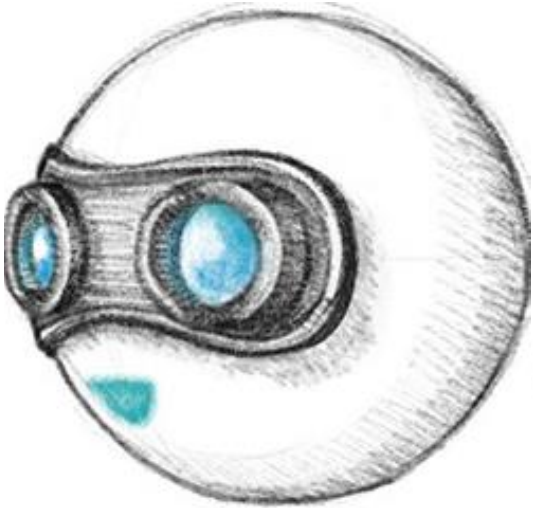




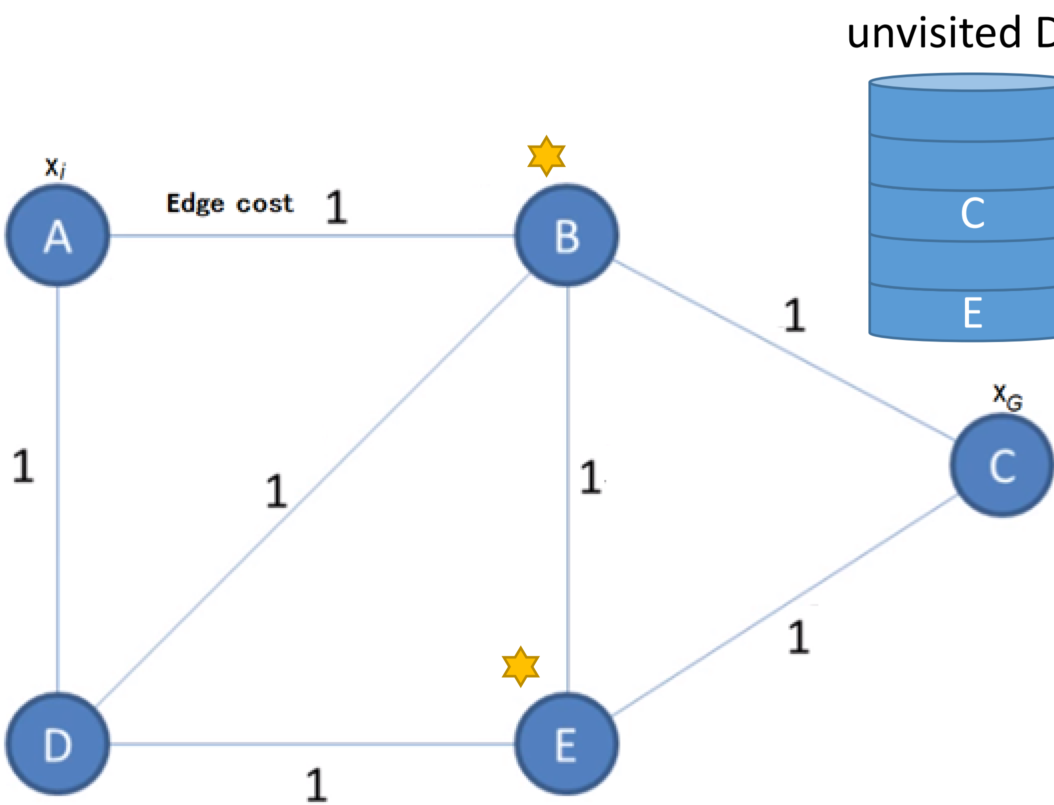
DFS :

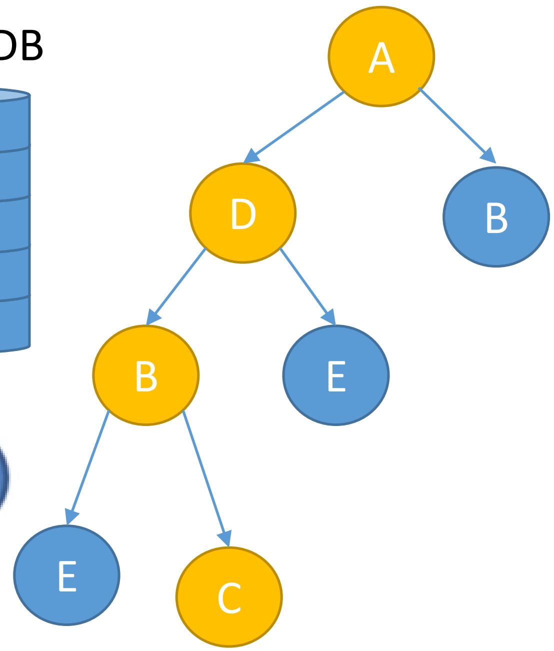


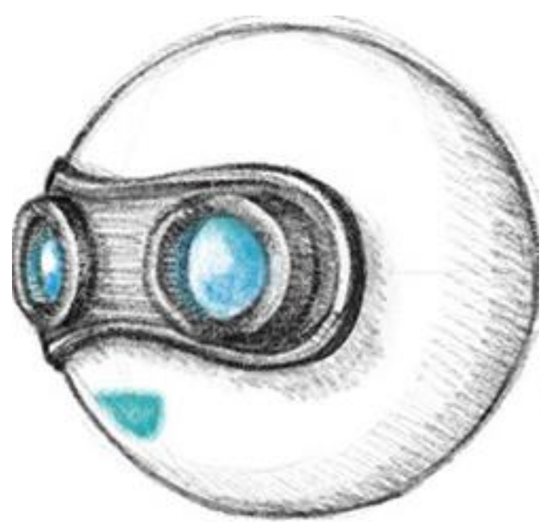




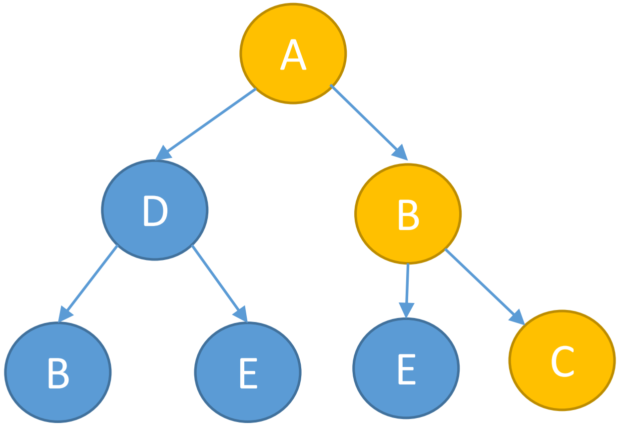
DFS :







Time and space complexity (BFS)



- b = # successors (in our case b= 2)

  maximum branching factor

- d – depth of the optimal solution

- m – maximum depth of the state space

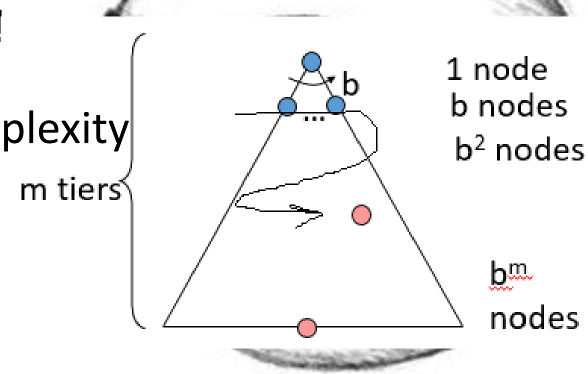
  $1+b+b^2+b^3+\ldots\ldots\ +b^d = O(b^d)$ if goal is exist= $1+2^1+2^2$

  $1+b+b^2+b^3+\ldots\ldots\ +b^m = O(b^m)$ if not exist

b = 10 and d = 10 -- > 3 hours @ processor has 1 million nodes/second !!!

10 terabytes of memory @ each node needs 1 KByte!!!

space complexity (memory) = should keeps all nodes = time complexity

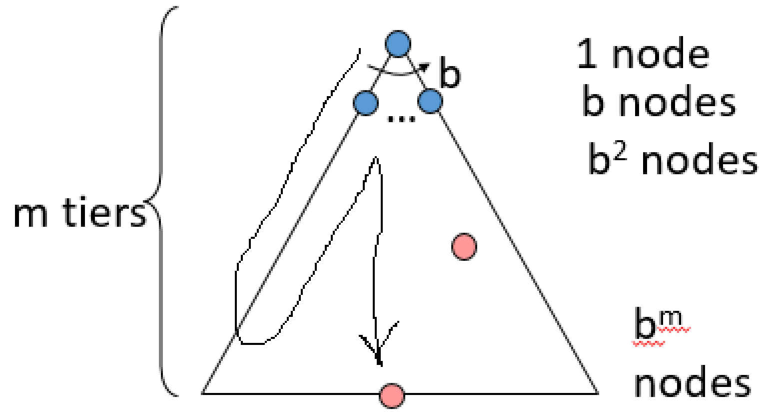


Time and space complexity (DFS)

- b = # successors (in our case b= 2)

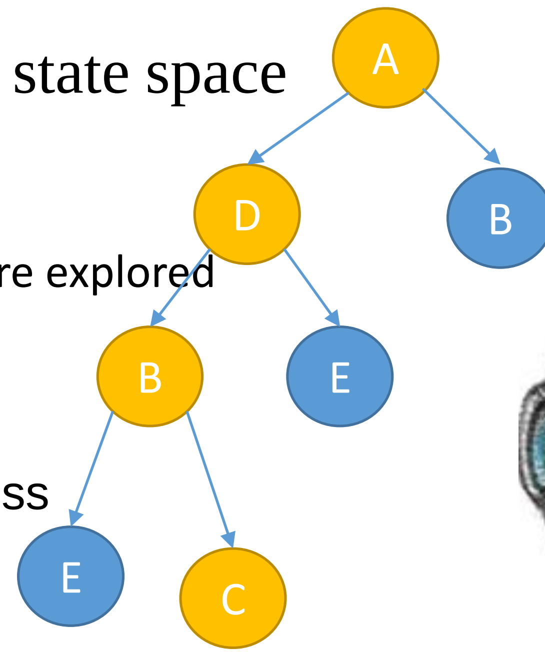
  maximum branching factor

- d – depth of the optimal solution
- m – maximum depth of the state space

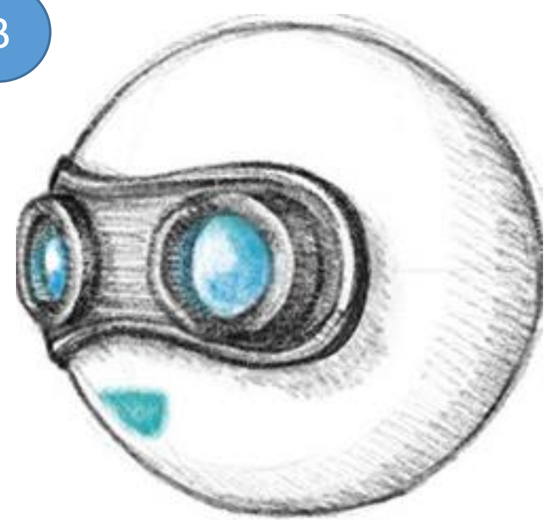


$1+b+b^2+b^3+\ldots\ldots.. +b^m = O(b^m)$

O(bm) = 2*3=6 = (A,D,B,B,E, C)nodes are explored



backtracking search uses even less memory



Time and space complexity (DFS* : backtracking search)

- b = # successors (in our case b= 2)

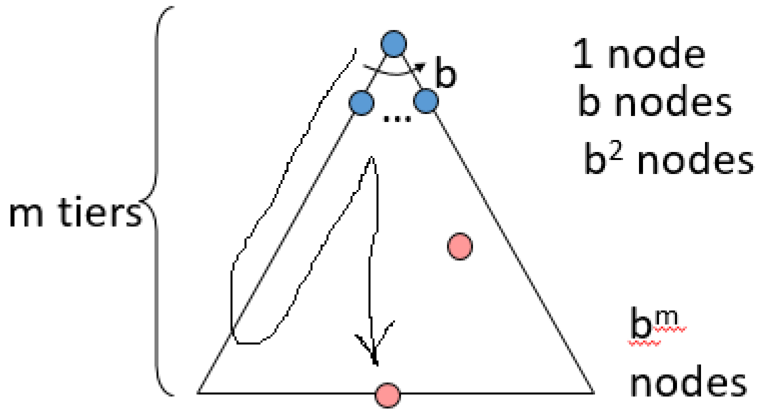
  maximum branching factor

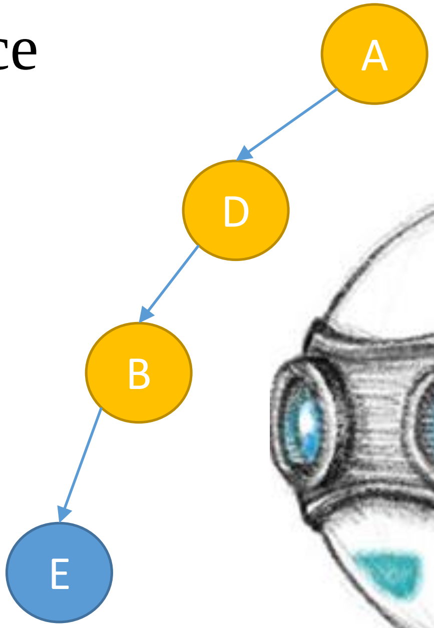
- d – depth of the optimal solution
- m – maximum depth of the state space

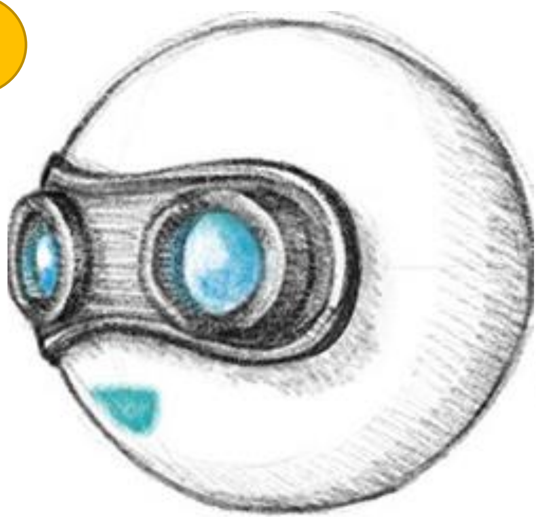
$1+b+b^2+b^3+\dots\dots \ +b^m = O(b^m)$

O(m) = 3

backtracking search uses even less memory

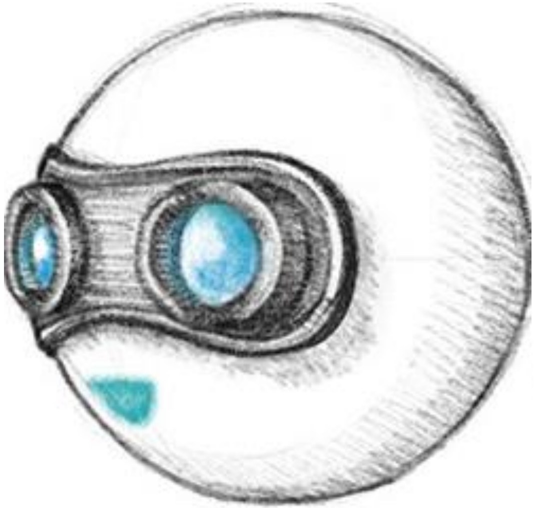






BFS vs DFS

Criterion	Breadth-First	Depth-First	Backtracking search
Complete?	Yes[1]	No	No
Optimal cost?	Yes[3]	No	No
Time	$O(b^d)$	$O(b^m)$	$O(b^m)$
Space	$O(b^d)$	$O(bm)$	$O(m)$



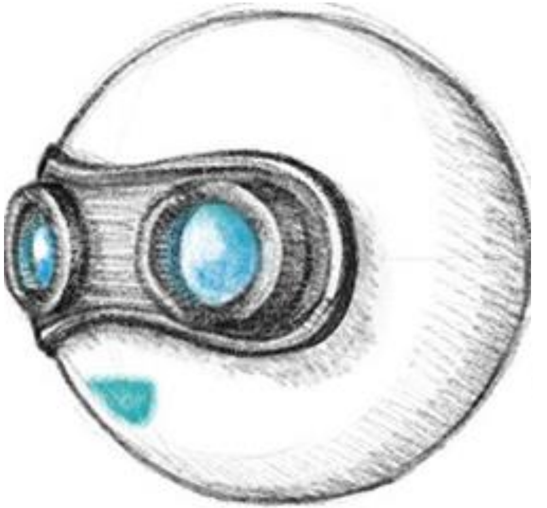
Planning

- Definition: is the process of determining a sequence of actions and motions, by looking ahead.

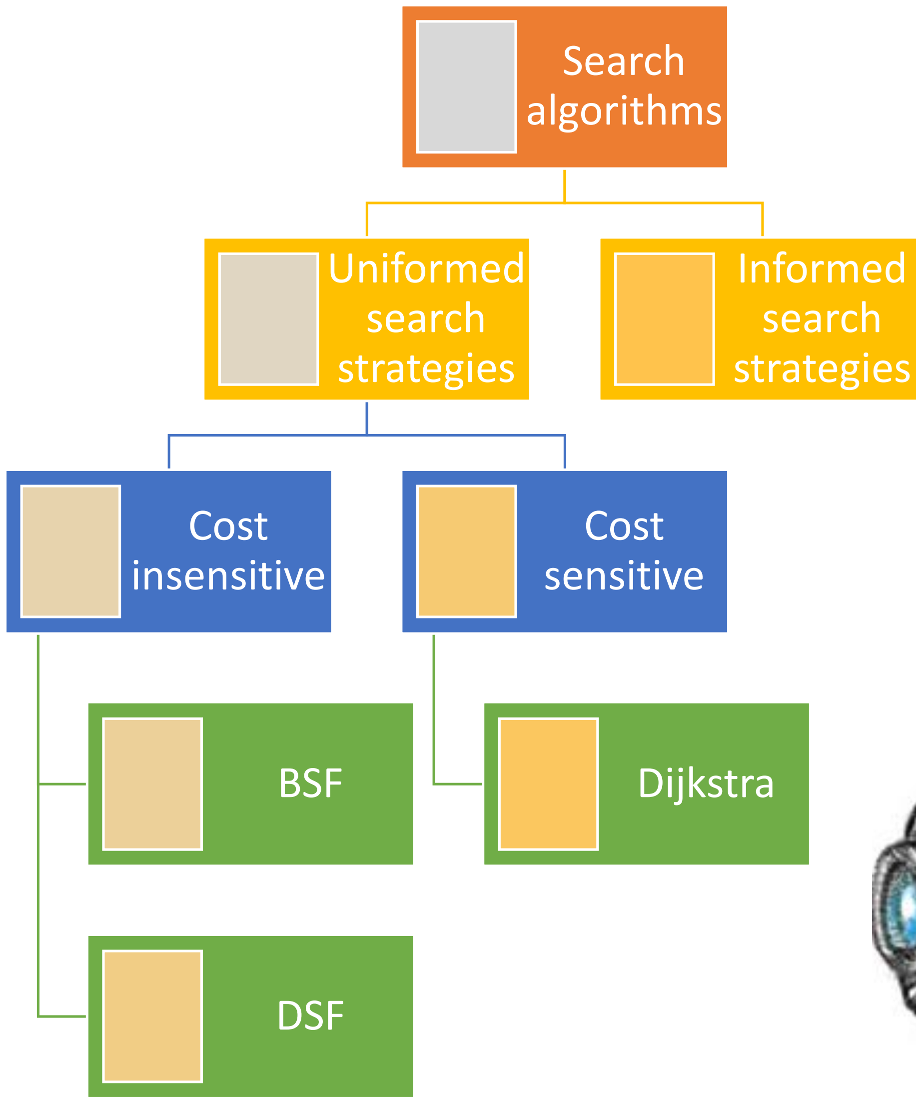
Search techniques :

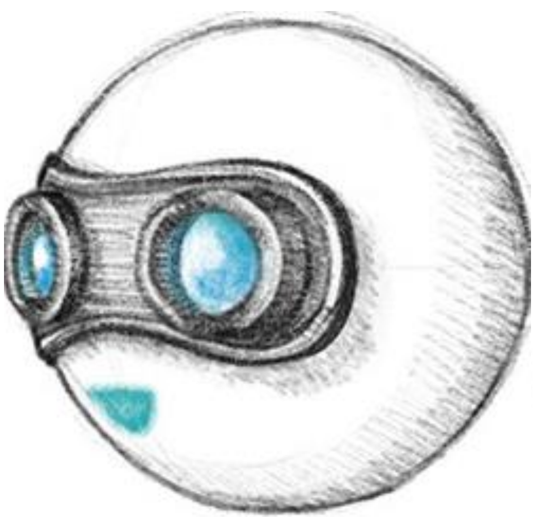
Breadth First Search (BFS) (uniform edge cost)

Dijkstra's algorithm (non-uniform edge cost) (surface condition)

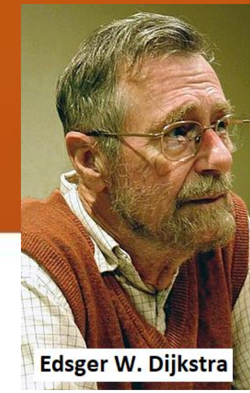


Search Tree

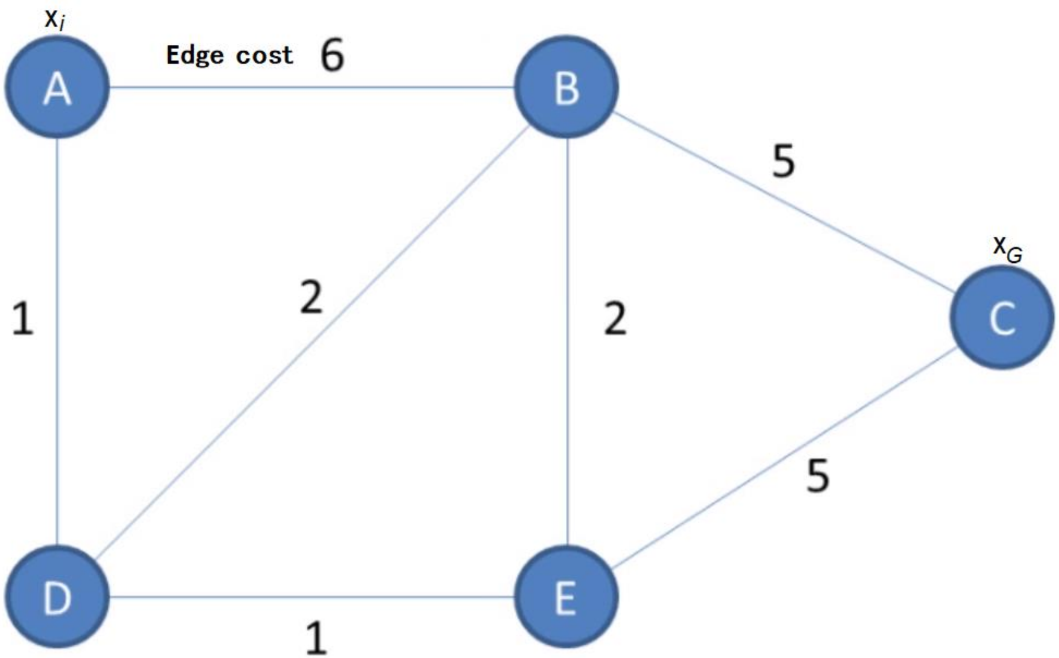

Search algorithms
Uniformed search strategies
Informed search strategies
Cost insensitive
Cost sensitive
BSF
DSF
Dijkstra



Searching Algorithms: Dijkstra's algorithm



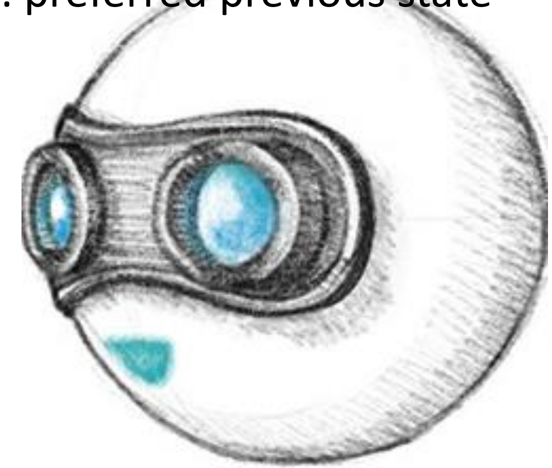
- Initialization: (start from initial State)



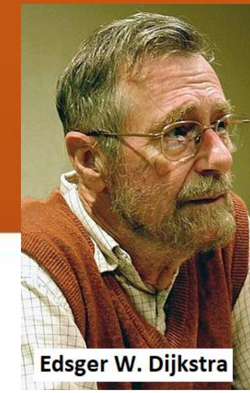
State	C(x)	PPR
A	0	-
B	∞	NULL
C	∞	NULL
D	∞	NULL
E	∞	NULL

*C(x): Optimal cost-to-come
*PPR: preferred previous state

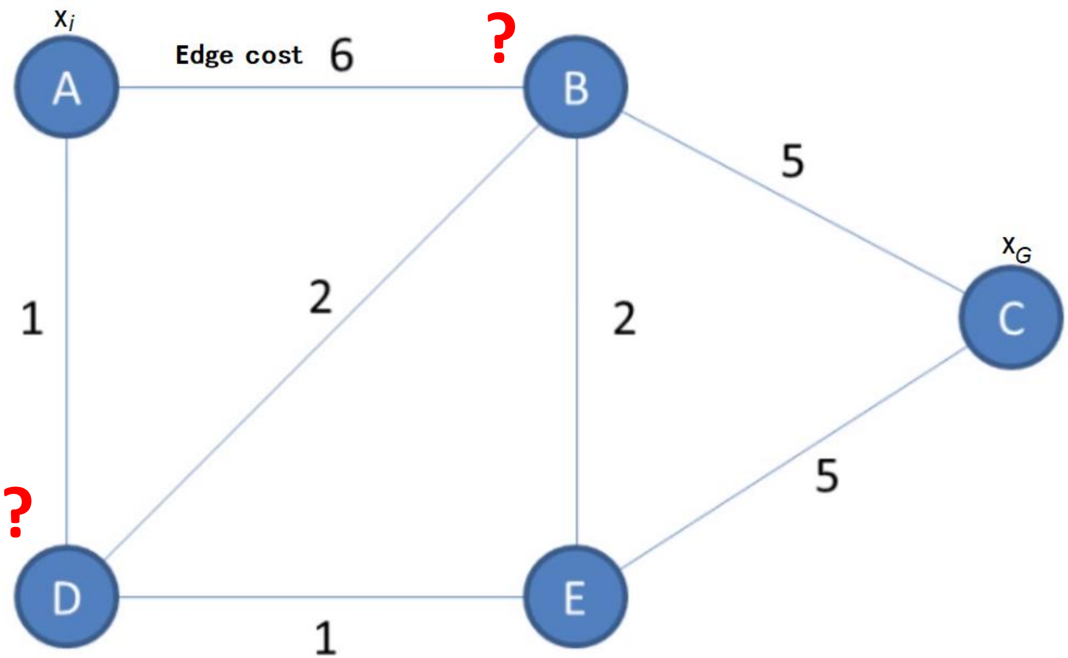
unvisited DB	Visited DB
A	NULL
B	NULL
C	NULL
D	NULL
E	NULL



Searching Algorithms: Dijkstra’s algorithm



- Step 1: (start from least C(x) which is A again)

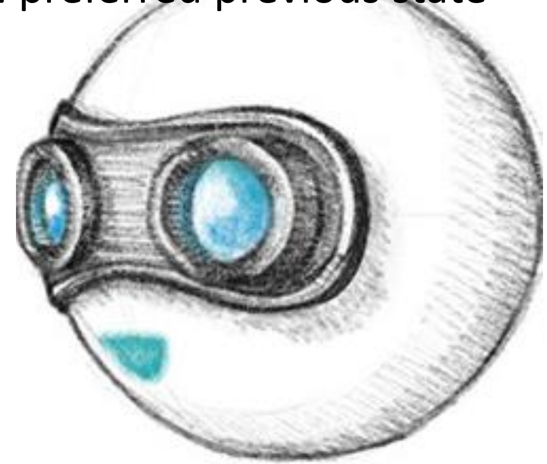


State	C(x)	PPR
A	0	-
B	6	A
C	∞	NULL
D	1	A
E	∞	NULL

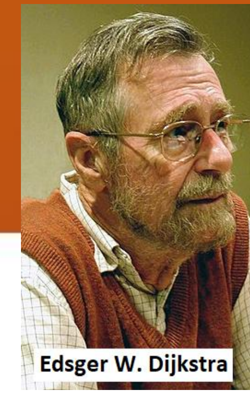
*C(x): Optimal cost-to-come
*PPR: preferred previous state

unvisited DB
B
C
D
E

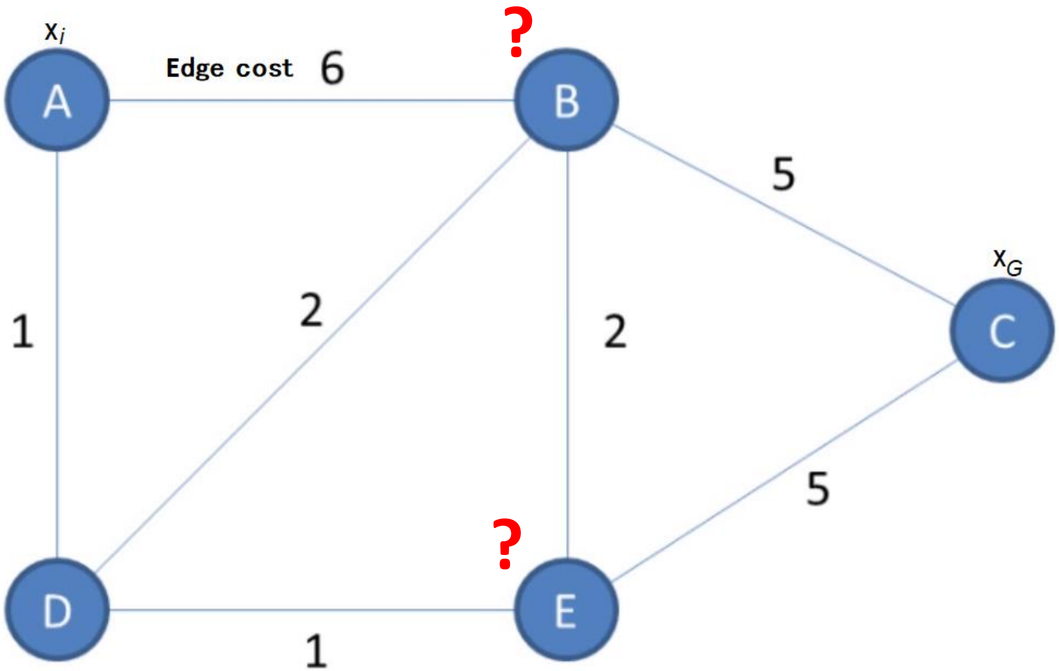
Visited DB
A
NULL
NULL
NULL
NULL



Searching Algorithms: Dijkstra’s algorithm



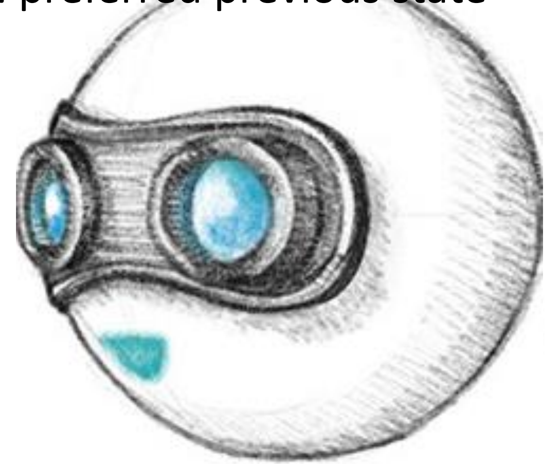
- Step 2: (start from least C(x) which is D)



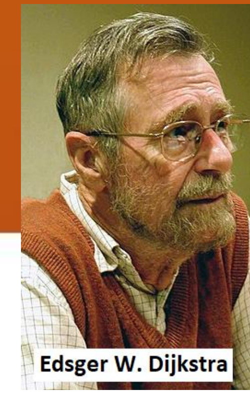
State	C(x)	PPR
A	0	-
B	3	D
C	∞	NULL
D	1	A
E	2	D

*C(x): Optimal cost-to-come
*PPR: preferred previous state

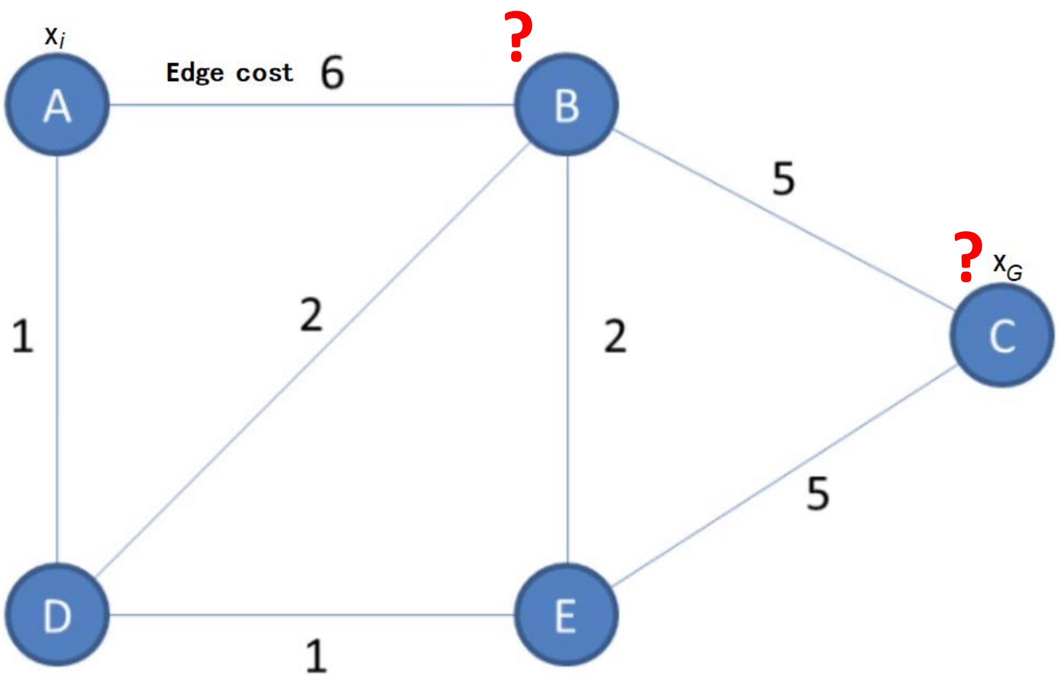
unvisited DB	Visited DB
	A
B	NULL
C	NULL
	D
E	NULL



Searching Algorithms: Dijkstra's algorithm



- Step 3: (start from least C(x) which is E)



State	C(x)	PPR
A	0	-
B	3	D
C	7	E
D	1	A
E	2	D

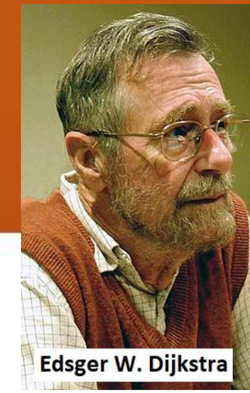
*C(x): Optimal cost-to-come

*PPR: preferred previous state

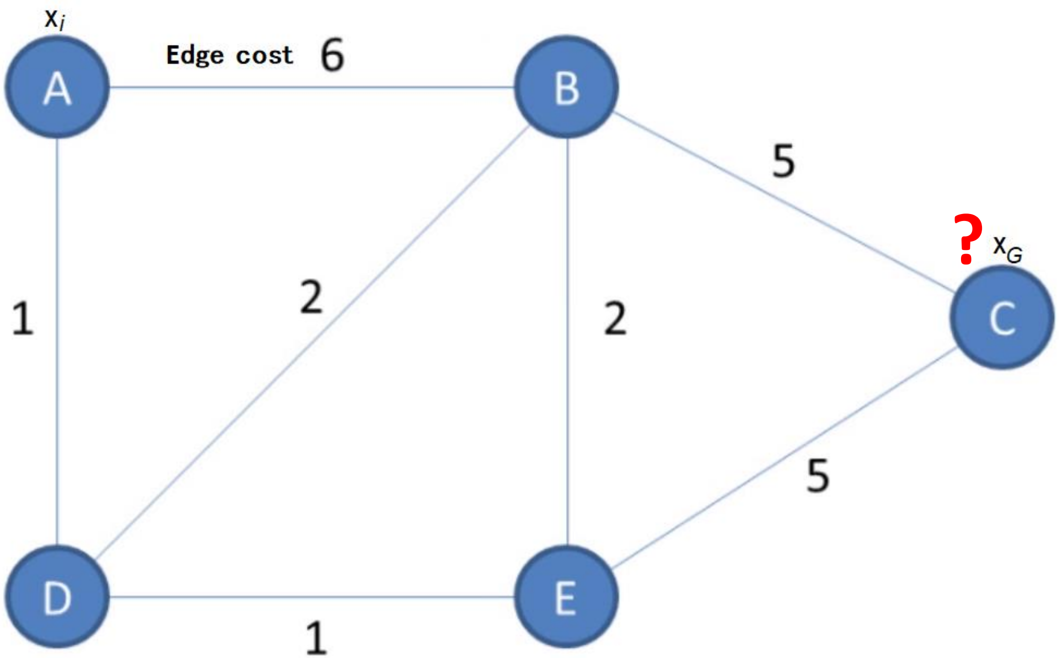
unvisited DB
B
C

Visited DB
A
NULL
NULL
D
E

Searching Algorithms: Dijkstra’s algorithm



- Step 4: (start from least C(x) which is B)



State	C(x)	PPR
A	0	-
B	3	D
C	7	E
D	1	A
E	2	D

*C(x): Optimal cost-to-come
*PPR: preferred previous state

unvisited DB

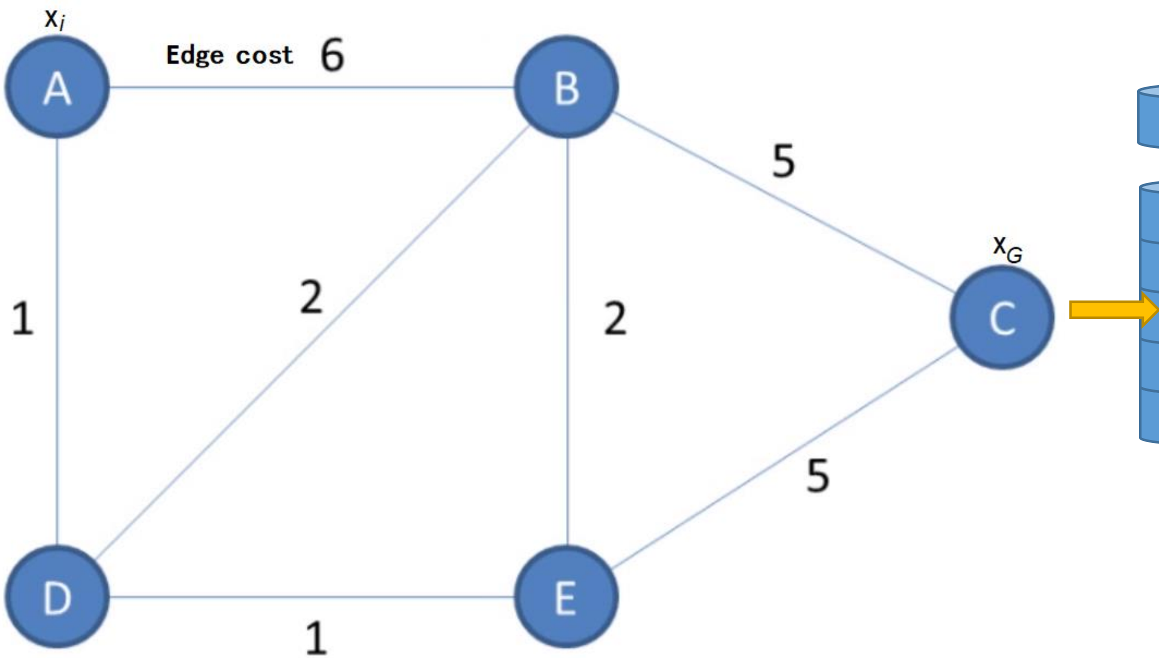
C

Visited DB

A
B
NULL
D
E

Searching Algorithms: Dijkstra's algorithm

- Step 5: (start from least C(x) which is C) no other states in unvisited

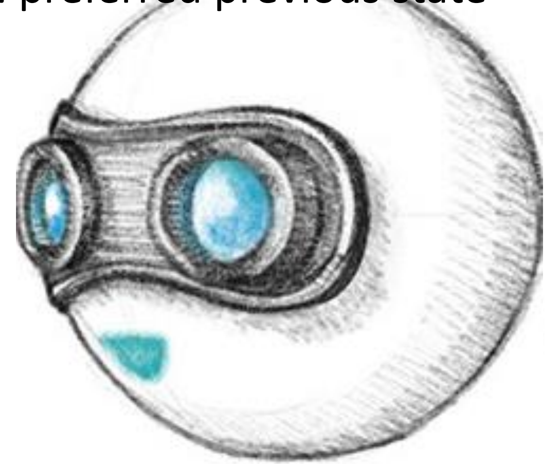


State	C(x)	PPR
A	0	-
B	3	D
C	7	E
D	1	A
E	2	D

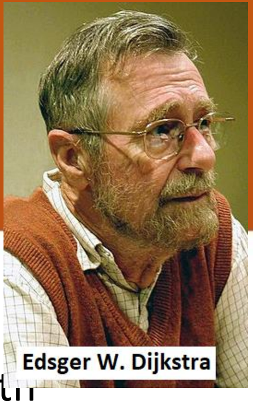
*C(x): Optimal cost-to-come

*PPR: preferred previous state

unvisited DB	Visited DB
	A
	B
	C
	D
	E

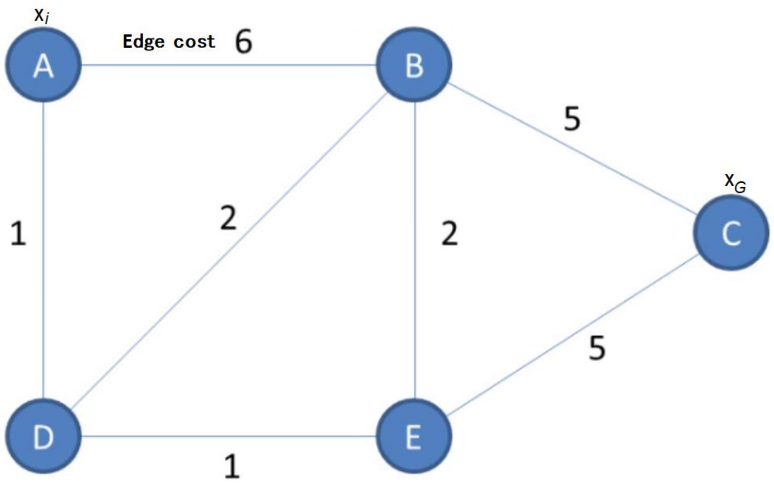


Searching Algorithms: Dijkstra's algorithm



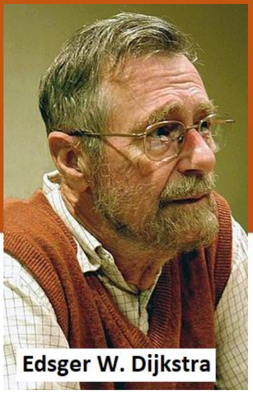
Algorithm steps:

1. Mark all states as unvisited
2. Mark the initial state with a current distance of 0 and the rest states with infinity,
3. For the current state, analyze all of its unvisited neighbors and calculate C(x).
4. Compare the recently measured C(X) with the assigned one in the database
5. Select the minimum C(X).
6. Mark the current state as visited state.
7. Choose the unvisited node that is marked with the least distance.
8. Repeat step 3.



State	C(x)	PPR
A	0	-
B	3	D
C	7	E
D	1	A
E	2	D

Searching Algorithms: Dijkstra's algorithm

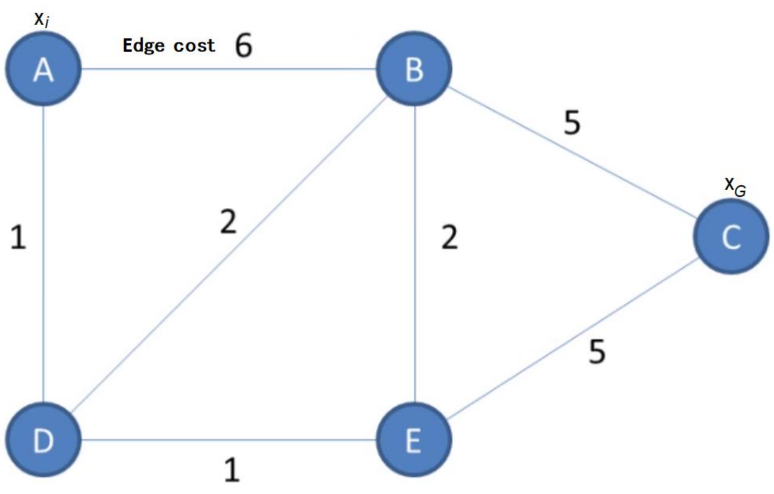


Advantages:

- little complexity which is almost linear.
- Optimal distance , avoid local minima

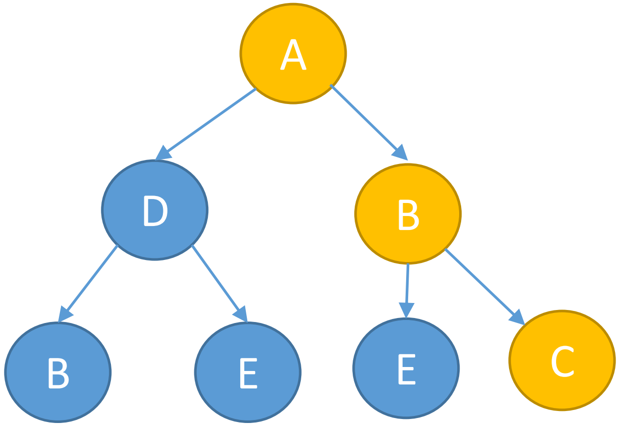
Disadvantages:

- Greedy algorithm (search in every node)
- Consume a lot of time.
- non-negative cost edges.



State	C(x)	PPR
A	0	-
B	3	D
C	7	E
D	1	A
E	2	D

Time and space complexity (BFS)



- b = # successors (in our case b= 2)

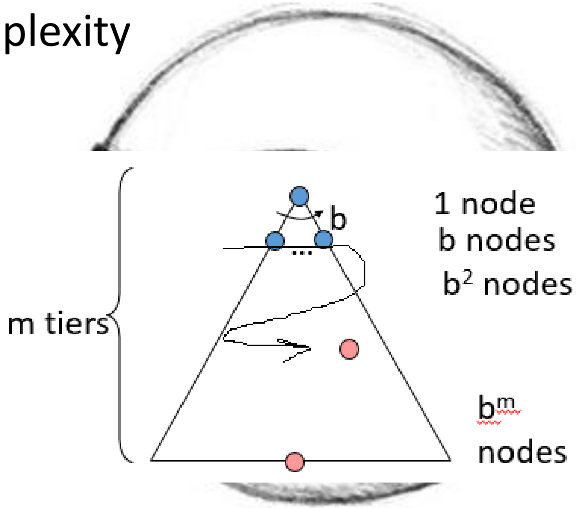
  maximum branching factor

- d – depth of the optimal solution

- m – maximum depth of the state space

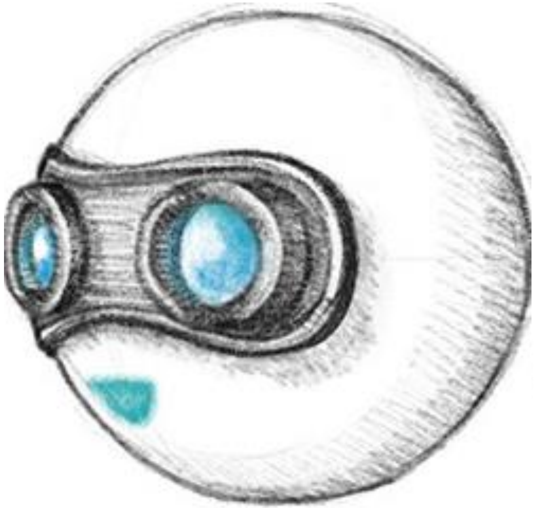
  $1+b+b^2+b^3+\ldots\ldots\ +b^m = O(b^m)$

space complexity (memory) = should keeps all nodes = time complexity



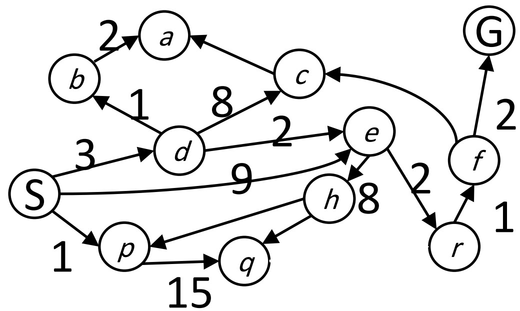
Comparison

Criterion	Breadth-First	Depth-First	Dijkstra
Complete?	Yes[1]	No	Yes
Optimal cost?	Yes[3]	No	Yes
Time	$O(b^d)$	$O(b^m)$	$O(b^m)$
Space	$O(b^d)$	$O(bm)$	$O(b^m)$

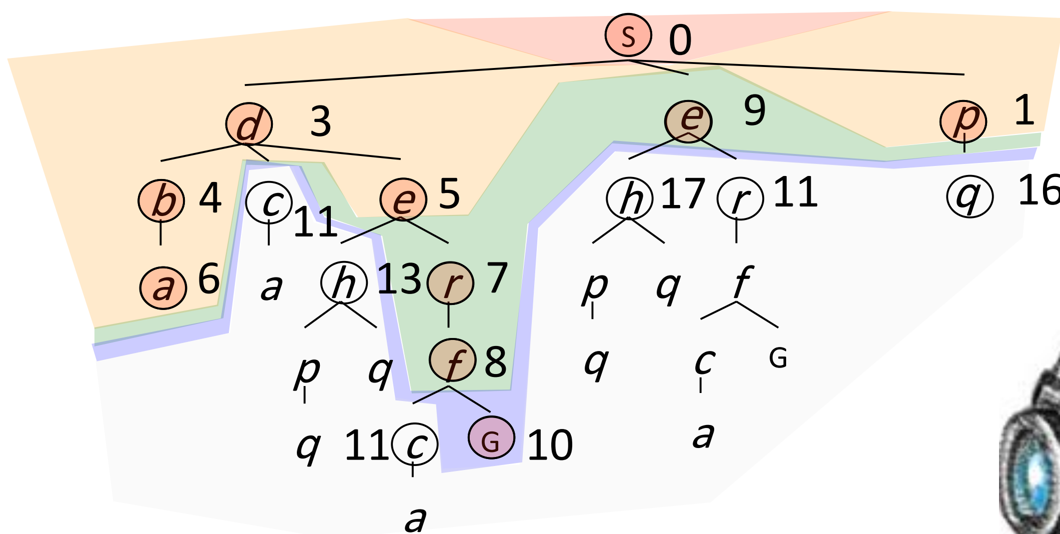


Uniform Cost Search (Dijkstra *)

Strategy: expand a cheapest node first:



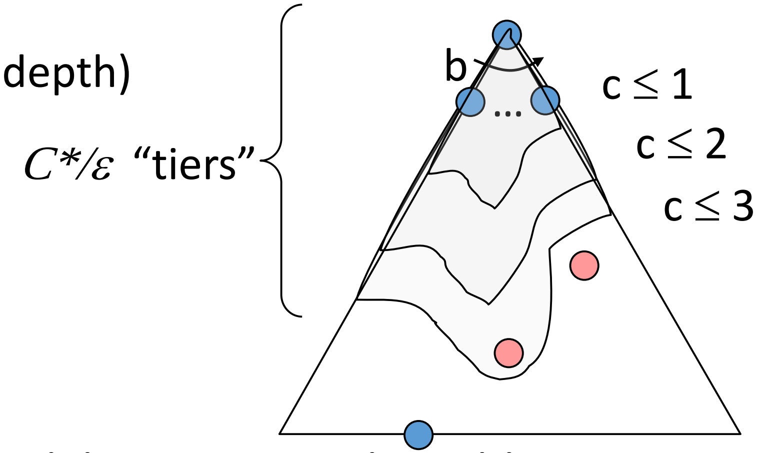
Cost contours



ai.berkeley.edu (Nikita Kitaev)

Uniform Cost Search (UCS) Properties

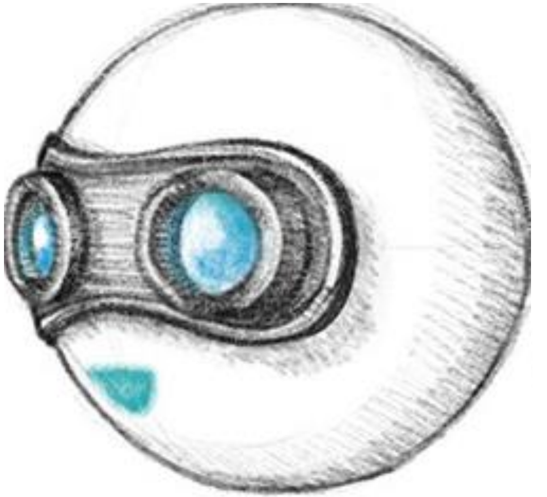
- What nodes does UCS expand?
  - Processes all nodes with cost less than cheapest solution!
  - If that solution costs C^* and arcs cost at least ε , then the "effective depth" is roughly C^*/ε
  - Takes time O($b^{C^*/\varepsilon}$) (exponential in effective depth)



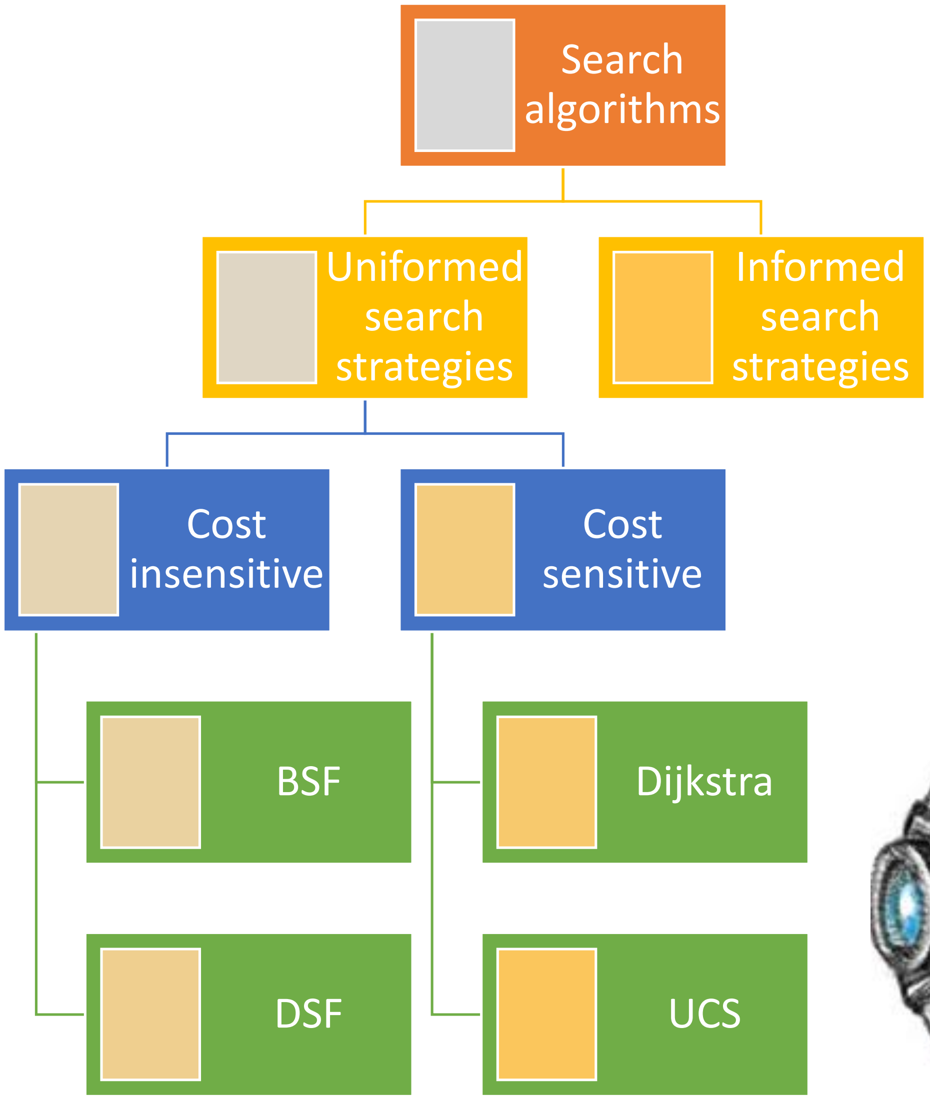
- How much space does the fringe take?
  - Has roughly the last tier, so O($b^{C^*/\varepsilon}$)
- Is it complete?
  - Assuming best solution has a finite cost and minimum arc cost is positive, yes!
- Is it optimal?
  - Yes! (Proof next lecture via A*)

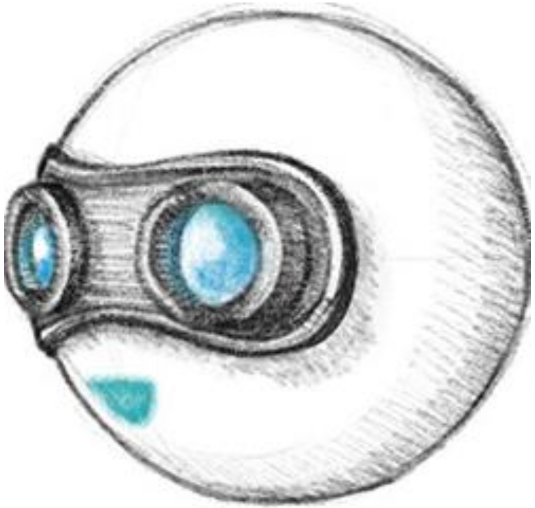
Comparison

Criterion	Breadth-First	Depth-First	Backtracking search	**Dijkstra**	Uniform-Cost
Complete?	Yes[1]	No	No	Yes	Yes[1,2]
Optimal cost?	Yes[3]	No	No	Yes	Yes
Time	$O(b^d)$	$O(b^m)$	$O(b^m)$	$O(b^m)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$
Space	$O(b^d)$	$O(bm)$	$O(m)$	$O(b^m)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$



Search Tree


Search algorithms
Uniformed search strategies
Informed search strategies
Cost insensitive
Cost sensitive
BSF
DSF
Dijkstra
UCS

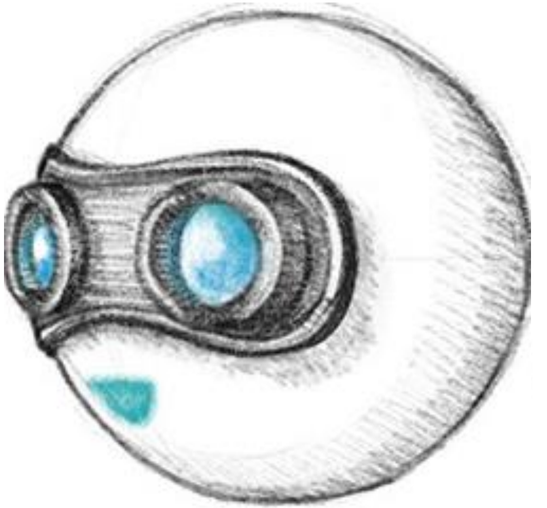


Planning

- Definition: is the process of determining a sequence of actions and motions, by looking ahead.

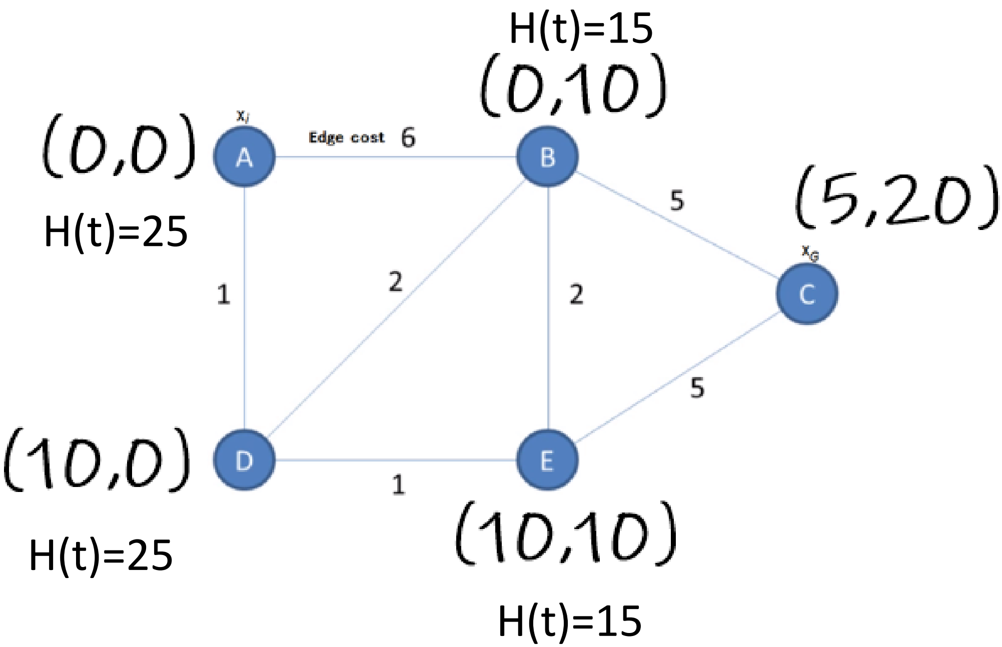
Search techniques :

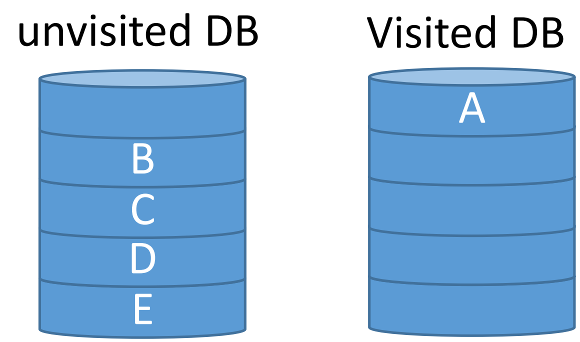
Breadth First Search (BFS) (uniform edge cost)
Dijkstra's algorithm (non-uniform edge cost) (surface condition)
Greedy Best First Search (nearest nodes)



Searching Algorithms: Greedy Best First Search

- Initialization: (start from initial State)

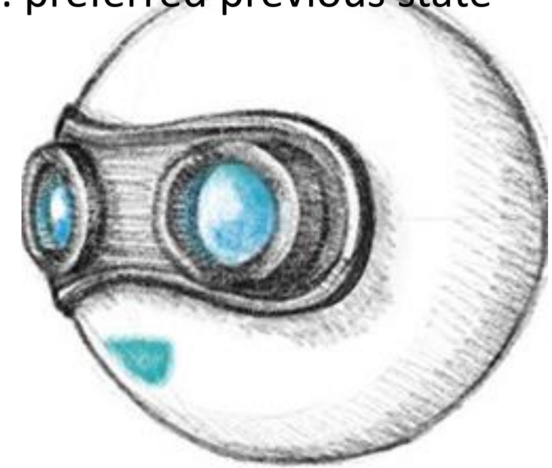




State	H(n)	f(n)	PPR
A	25	0	-
B	15	∞	NULL
C	0	∞	NULL
D	25	∞	NULL
E	15	∞	NULL

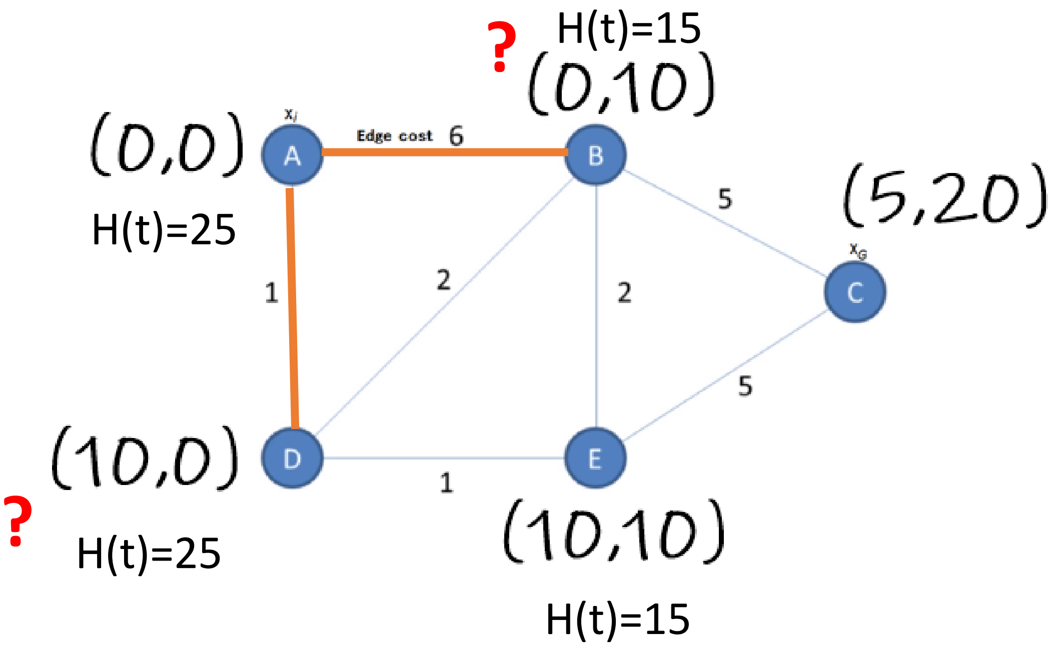
*C(x): Optimal cost-to-come
*PPR: preferred previous state

f (n) = h(n).



Searching Algorithms: GBFs

- Step 1: (start from least C(x) which is A again)



unvisited DB

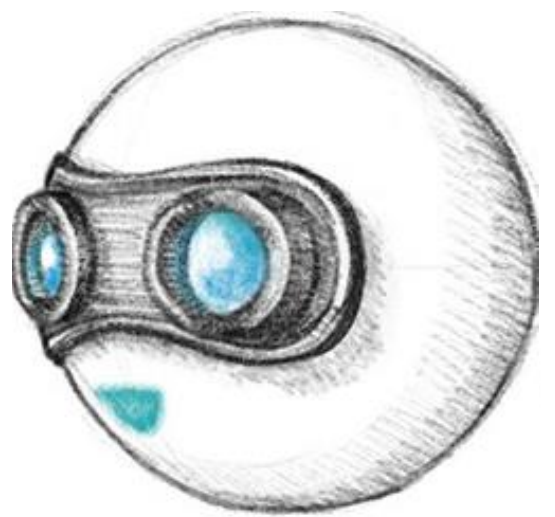
B
C
D
E

Visited DB

A

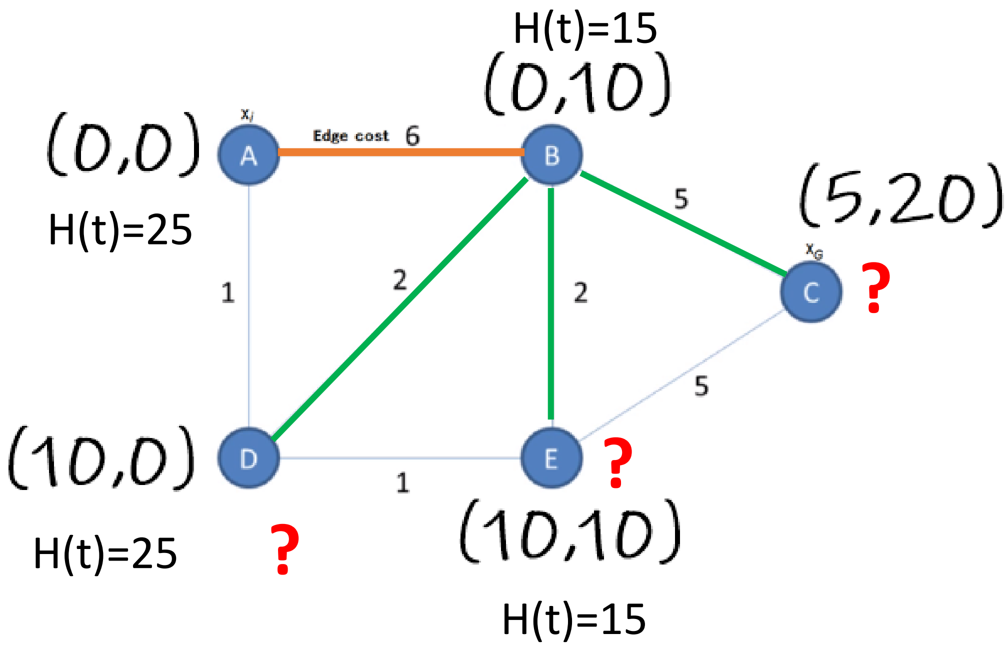
State	H(n)	f(n)	PPR
A	25	0	-
B	15	6	A
C	0	∞	NULL
D	25	1	A
E	15	∞	NULL

f (n) = h(n).



Searching Algorithms: GBFs

- Step 2: (start from least f(n) which is B)



unvisited DB

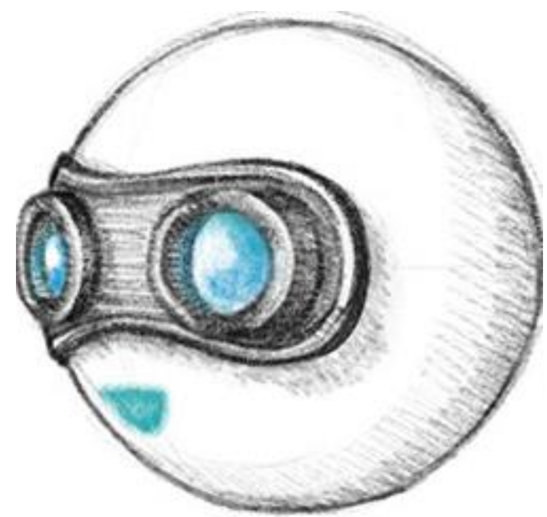
- C
- D
- E

Visited DB

- A
- B

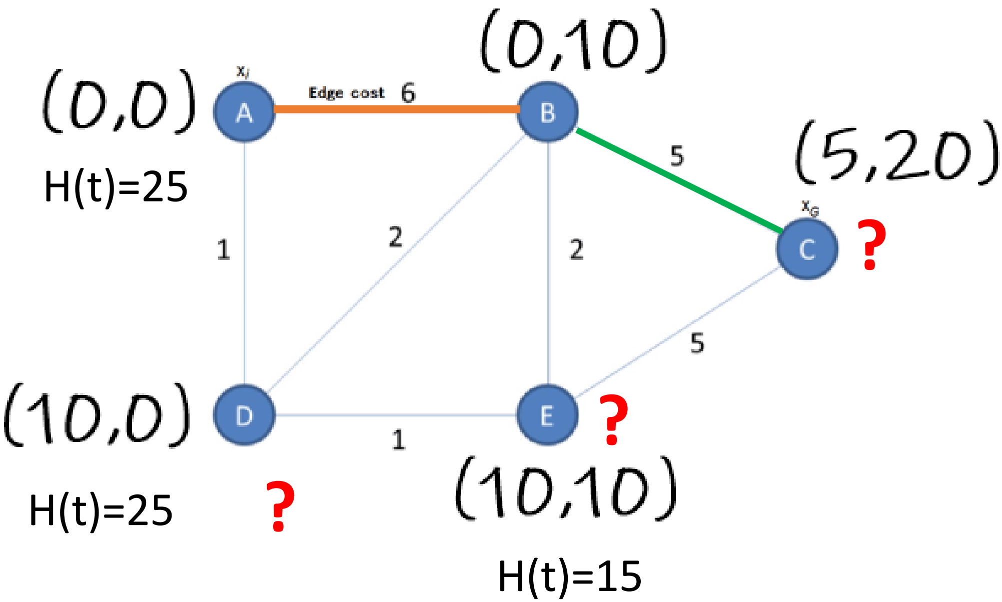
State	H(n)	f(n)	PPR
A	25	25	-
B	15	6	A
C	0	11	B
D	25	1	A
E	15	∞	NULL

f (n) = h(n).



Searching Algorithms: GBFs

- Step 3: (start from least f(n) which is C)



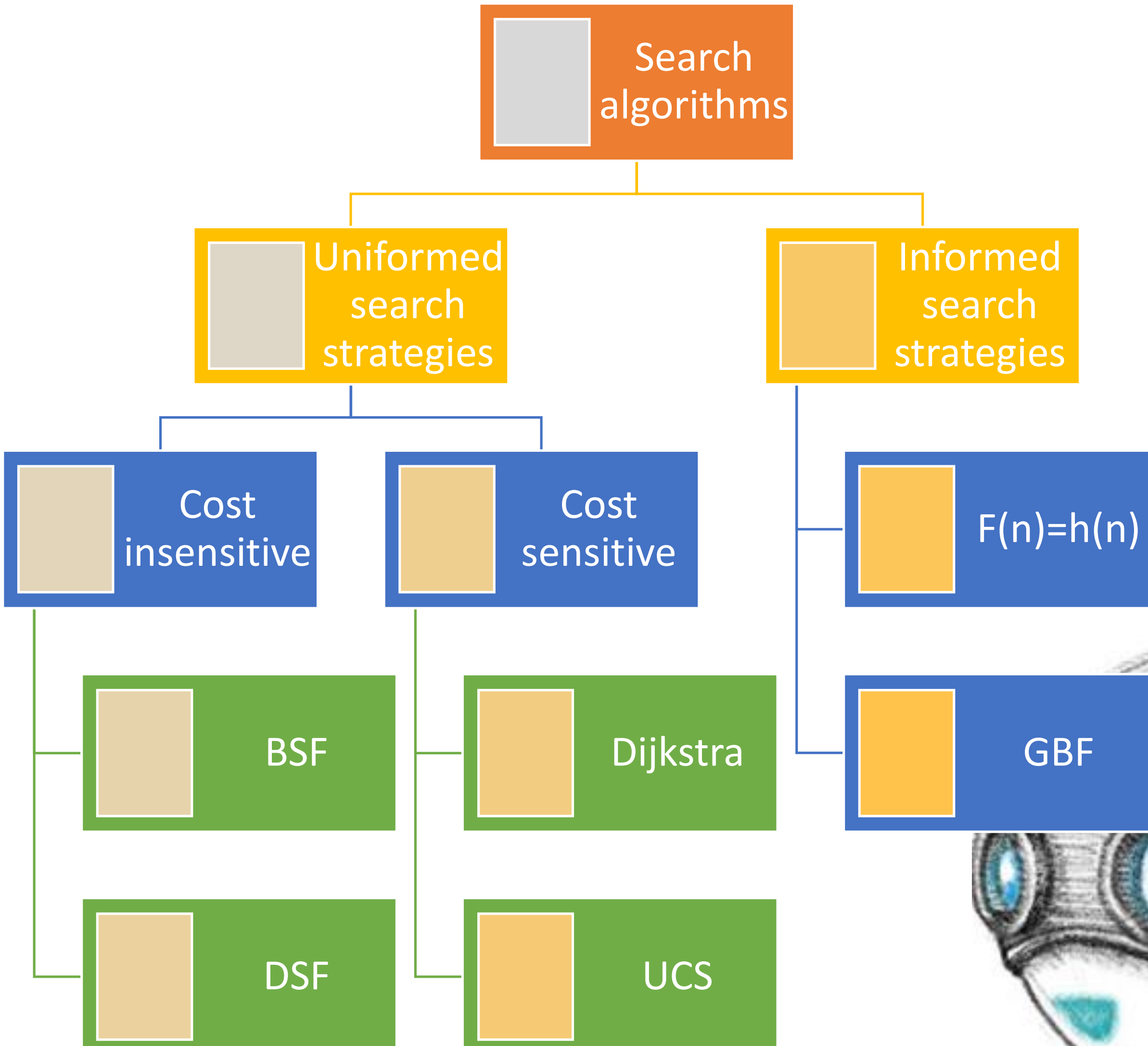
unvisited DB
D
E

Visited DB
A
B
C

State	H(n)	f(n)	PPR
A	25	25	-
B	15	6	A
C	0	11	B
D	25	1	A
E	15	8	NULL

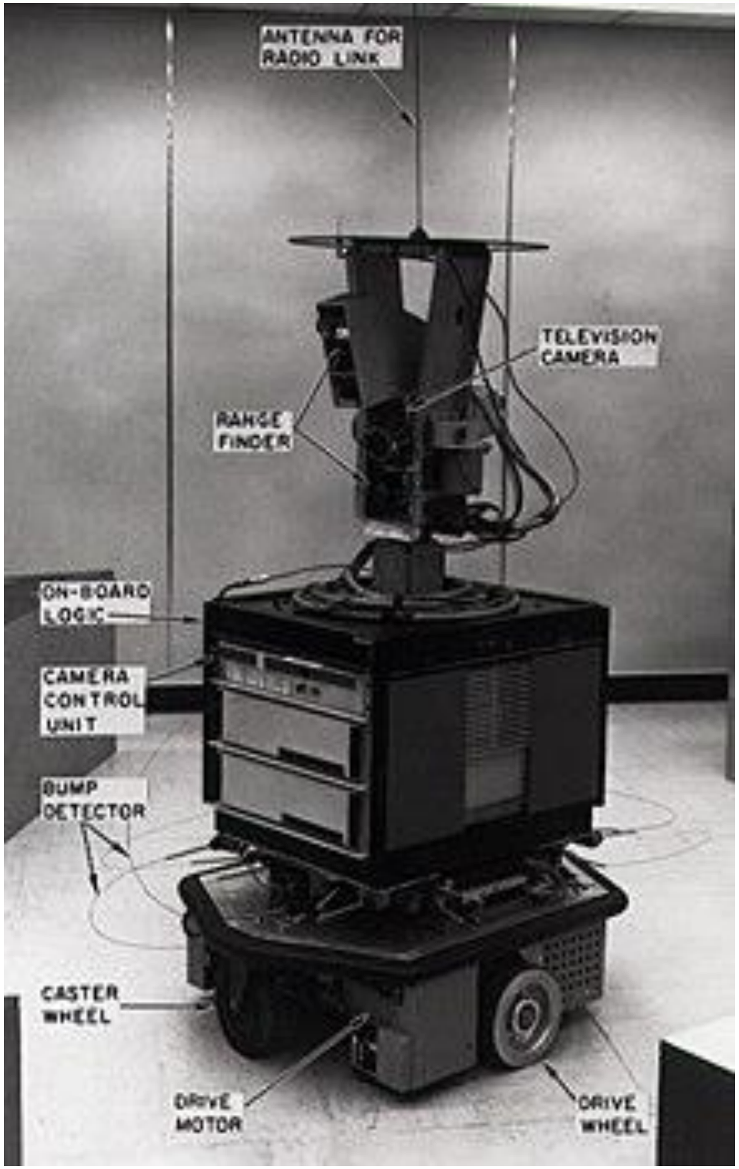
f (n) = h(n).

Search Tree



Search algorithms
Uniformed search strategies
Informed search strategies
Cost insensitive
Cost sensitive
F(n)=h(n)
BSF
Dijkstra
GBF
DSF
UCS

A* Algorithm (for SHAKEY)



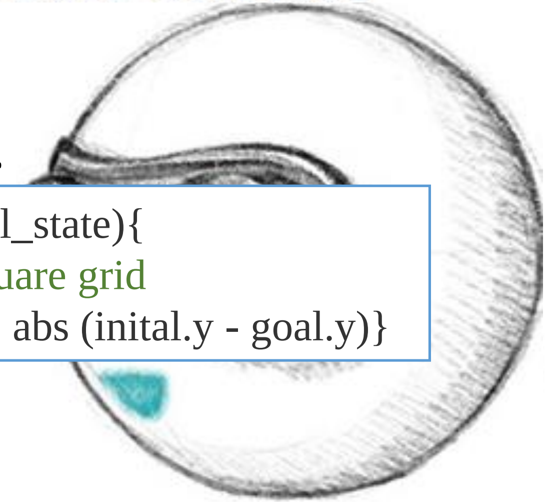
A Formal Basis for the Heuristic Determination of Minimum Cost Paths

PETER E. HART, MEMBER, IEEE, NILS J. NILSSON, MEMBER, IEEE, AND BERTRAM RAPHAEL

Nils Nilsson
USA 1933-2019

Peter Hart
USA 1940-2005

Bertram Raphael
USA 1936-



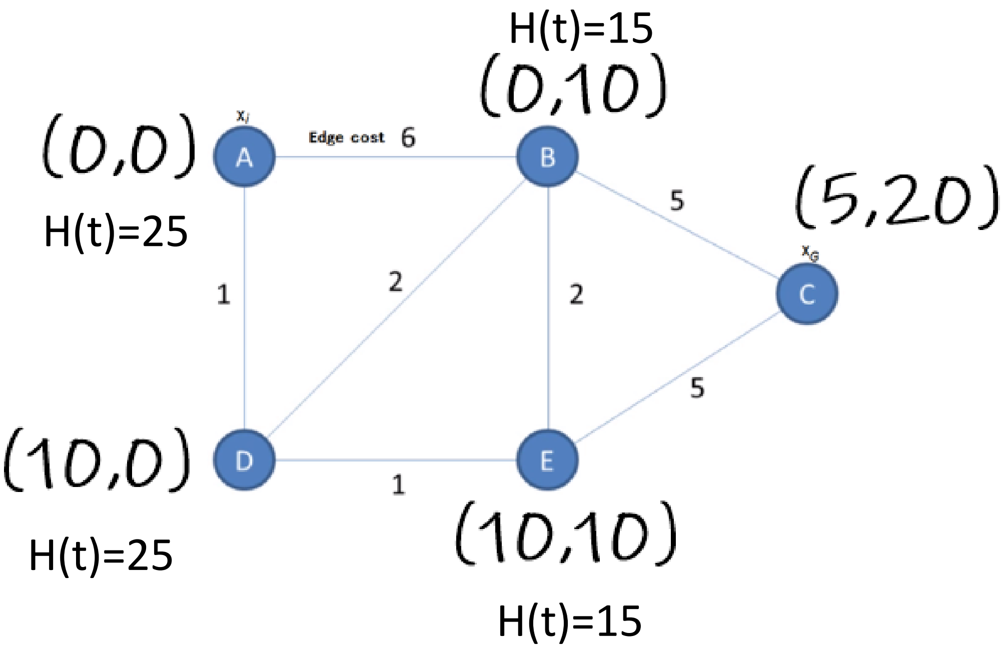
- Admissible heuristic
- Heuristic function *h*(*n*).

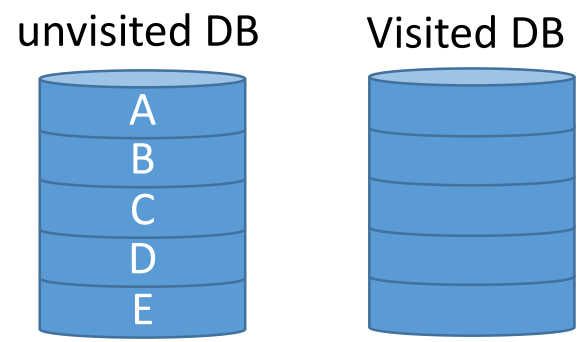
```
Int heuristic(initial_state, goal_state){
# Manhattan distance on a square grid
return abs (inital.x - goal.x) + abs (inital.y - goal.y)}
```

- Published 1968

Searching Algorithms: A*

- Initialization: (start from initial State)





*C(x): Optimal cost-to-come

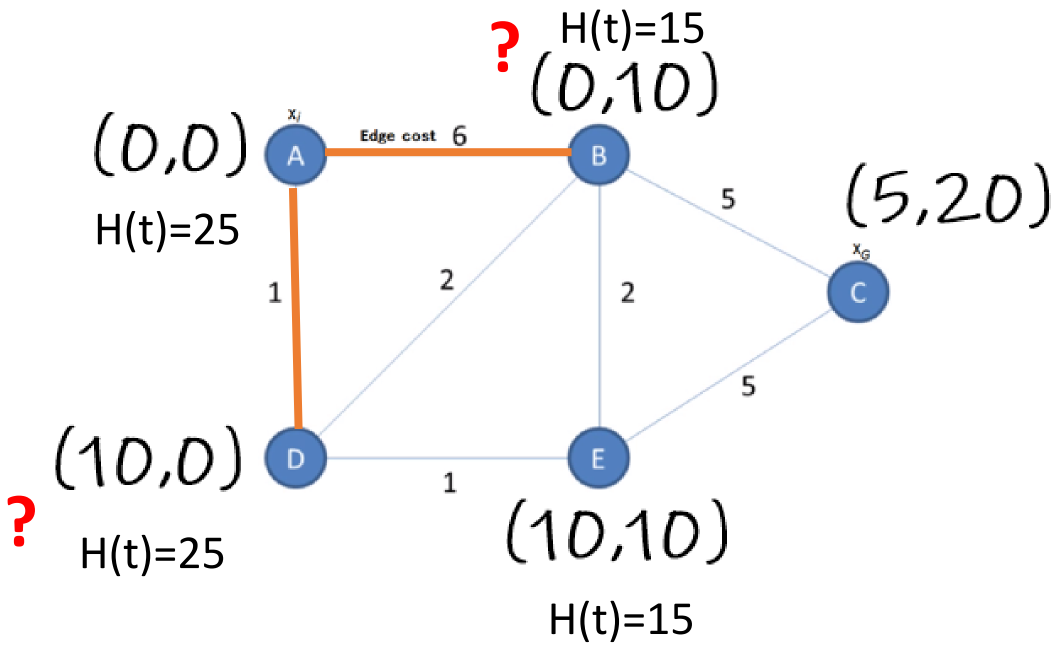
*PPR: preferred previous state

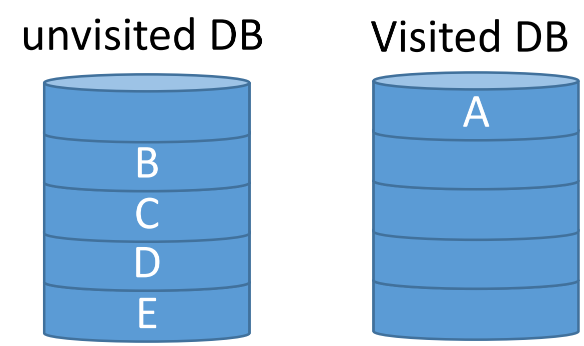
State	H(n)	f(n)	PPR
A	25	25	-
B	15	∞	NULL
C	0	∞	NULL
D	25	∞	NULL
E	15	∞	NULL

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

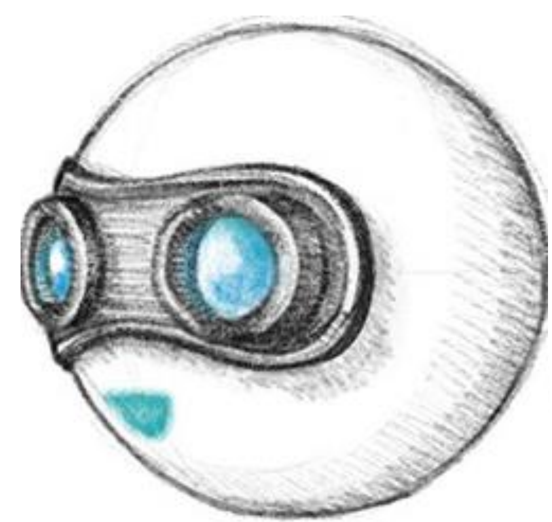
Searching Algorithms: A*

- Step 1: (start from least C(x) which is A again)



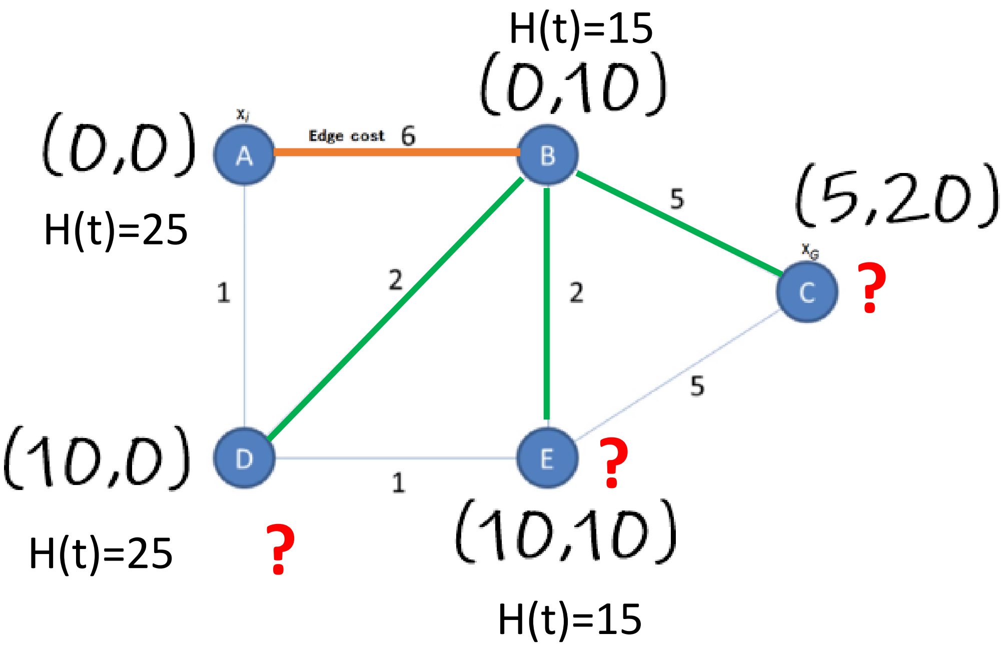


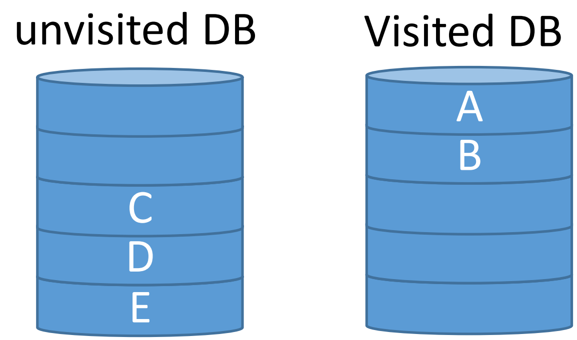
State	H(n)	f(n)	PPR
A	25	25	-
B	15	21	A
C	0	∞	NULL
D	25	26	A
E	15	∞	NULL



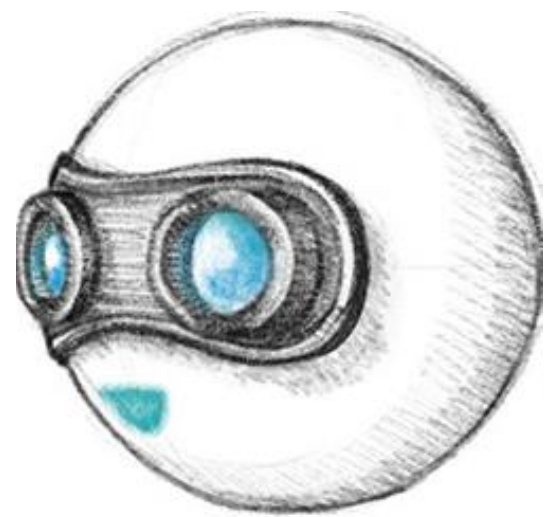
Searching Algorithms: A*

- Step 2: (start from least f(n) which is B)



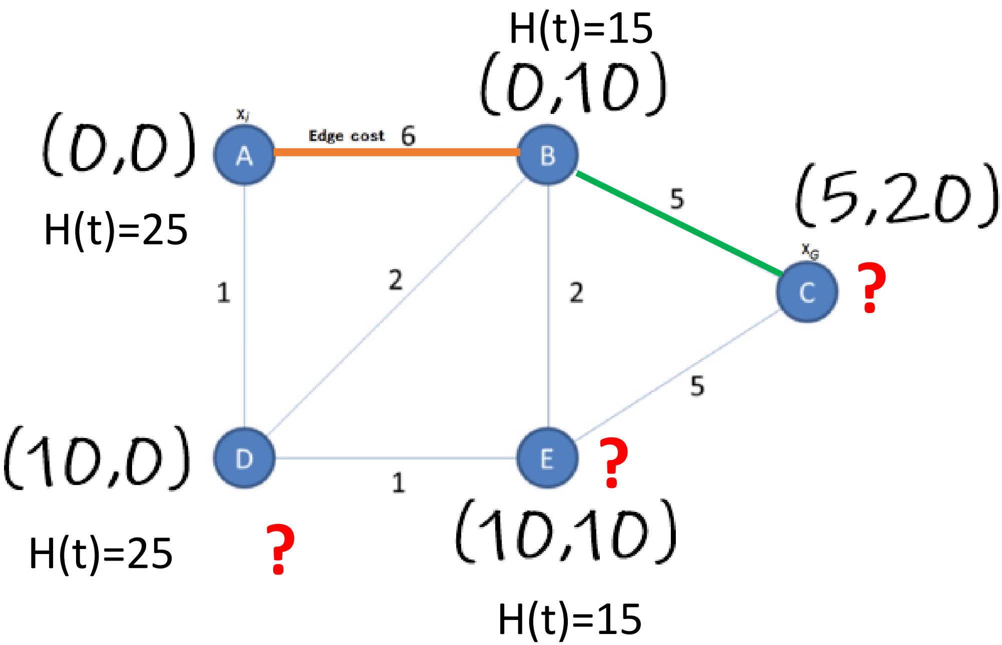


State	H(n)	f(n)	PPR
A	25	25	-
B	15	21	A
C	0	11	B
D	25	33	B
E	15	23	B



Searching Algorithms: A*

- Step 3: (start from least f(n) which is C)



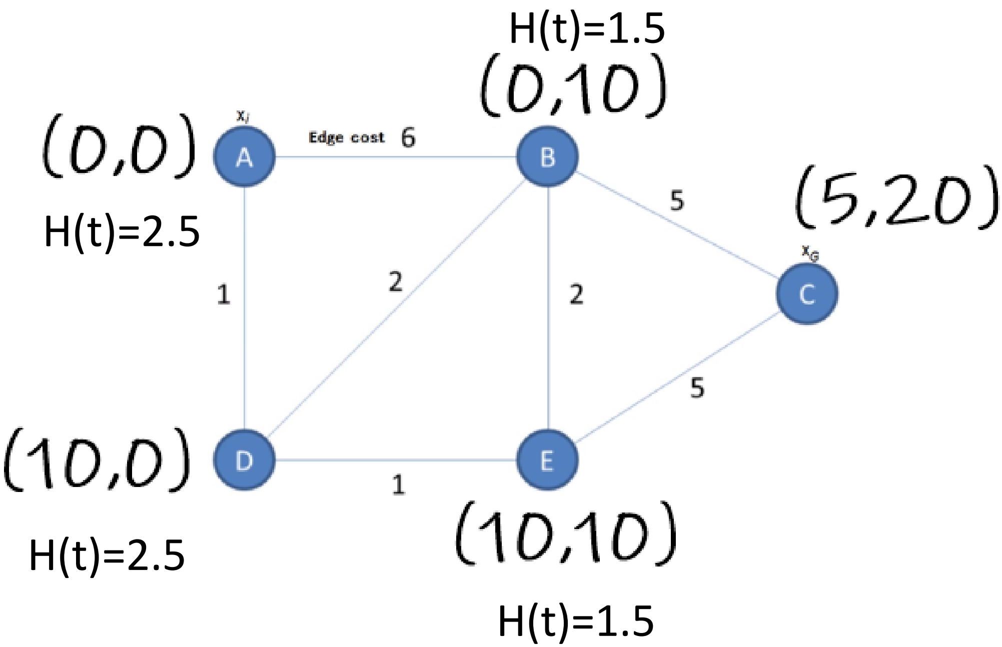
overestimating heuristic.

unvisited DB	Visited DB
	A
	B
	C
D	
E	

State	H(n)	f(n)	PPR
A	25	25	-
B	15	21	A
C	0	11	B
D	25	33	B
E	15	23	B

Searching Algorithms: A* (underestimating heuristic.)

- Initialization: (start from initial State)



unvisited DB: A, B, C, D, E

Visited DB: (empty)

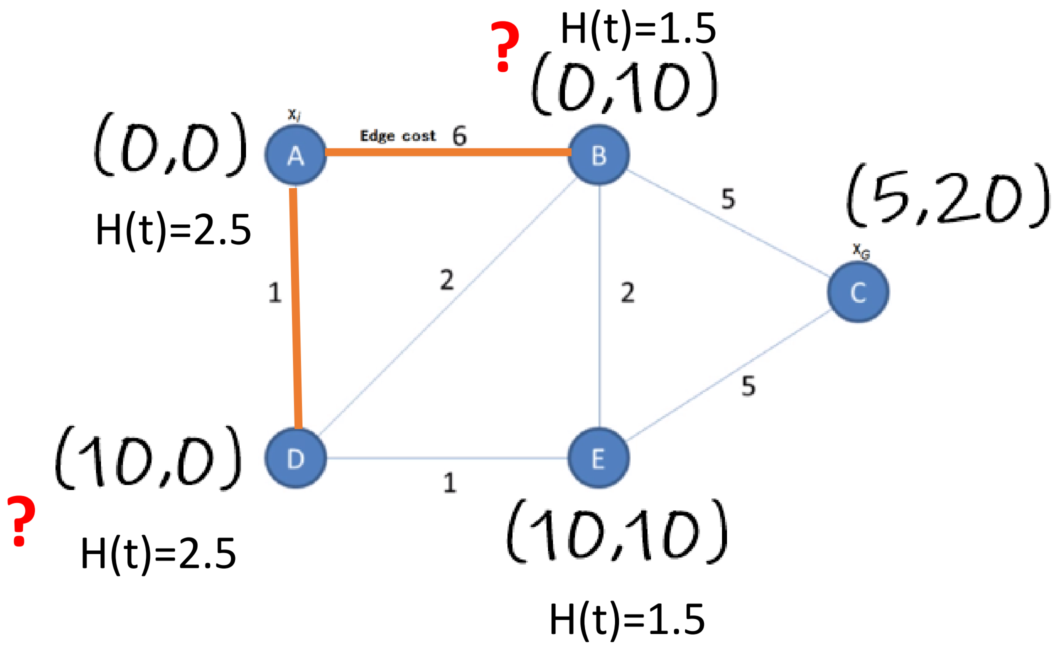
State	H(n)	f(n)	PPR
A	2.5	2.5	-
B	1.5	∞	NULL
C	0	∞	NULL
D	2.5	∞	NULL
E	1.5	∞	NULL

*C(x): Optimal cost-to-come

*PPR: preferred previous state

Searching Algorithms: A*

- Step 1: (start from least f(n) which is A again)

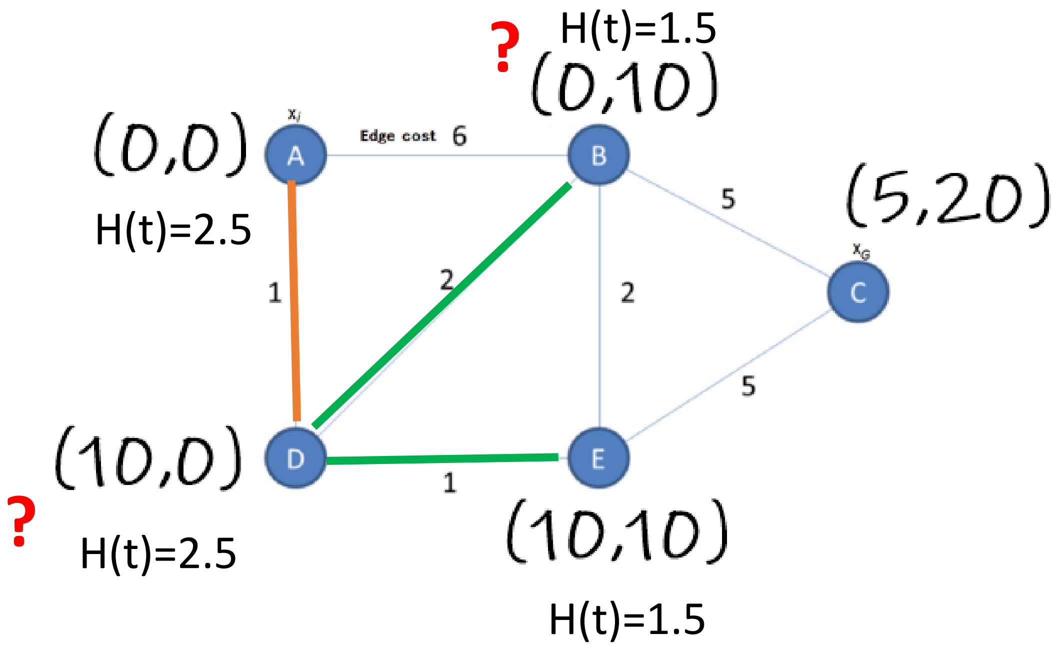


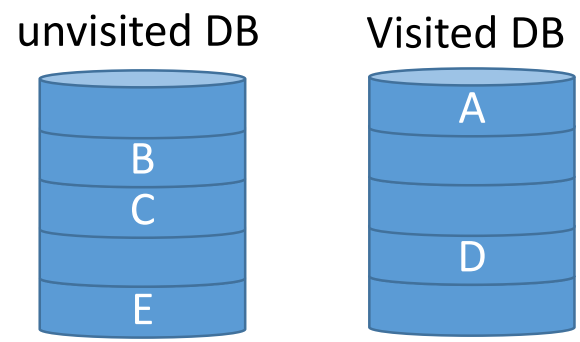
unvisited DB	Visited DB
B	A
C	
D	
E	

State	H(n)	f(n)	PPR
A	2.5	2.5	-
B	1.5	7.5	A
C	0	∞	NULL
D	2.5	3.5	A
E	1.5	∞	NULL

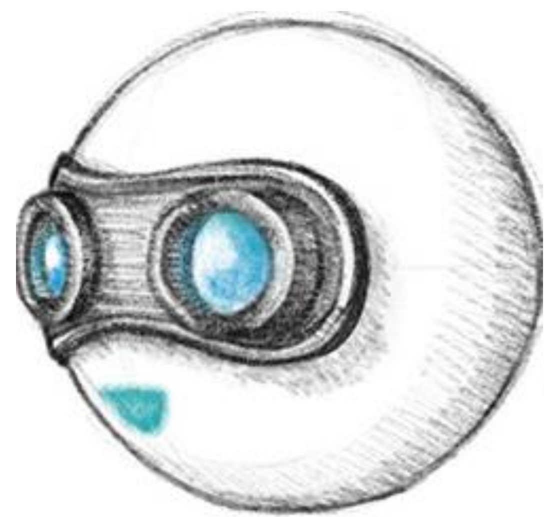
Searching Algorithms: A*

- Step 2: (start from least f(n) which is D)



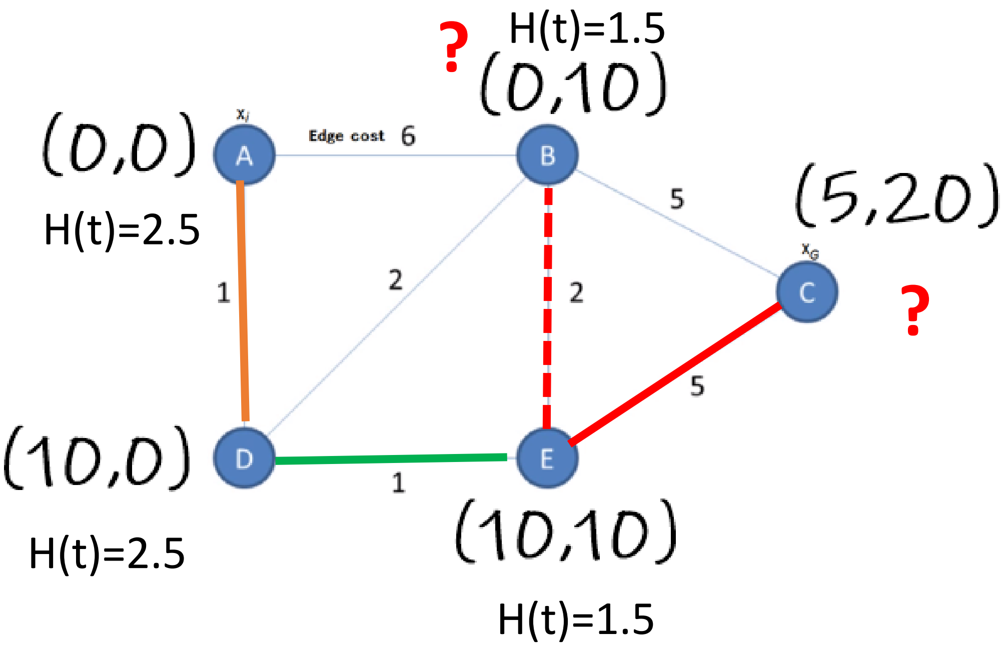


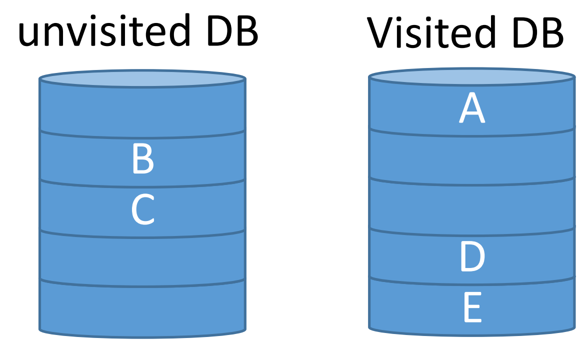
State	H(n)	f(n)	PPR
A	2.5	2.5	-
B	1.5	4.5	D
C	0	∞	NULL
D	2.5	3.5	A
E	1.5	3.5	D



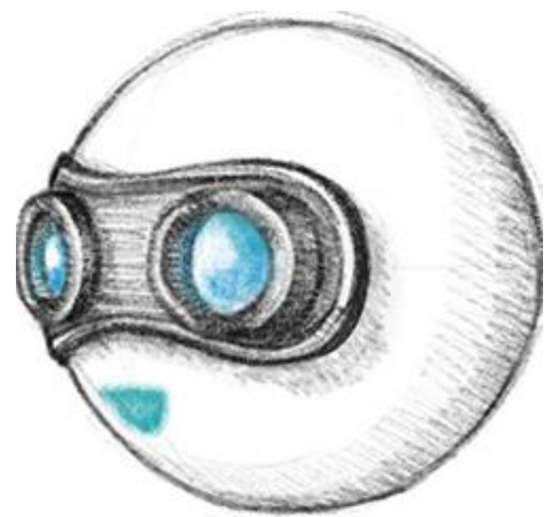
Searching Algorithms: A*

- Step 3: (start from least f(n) which is E (C is goal ,stop search))





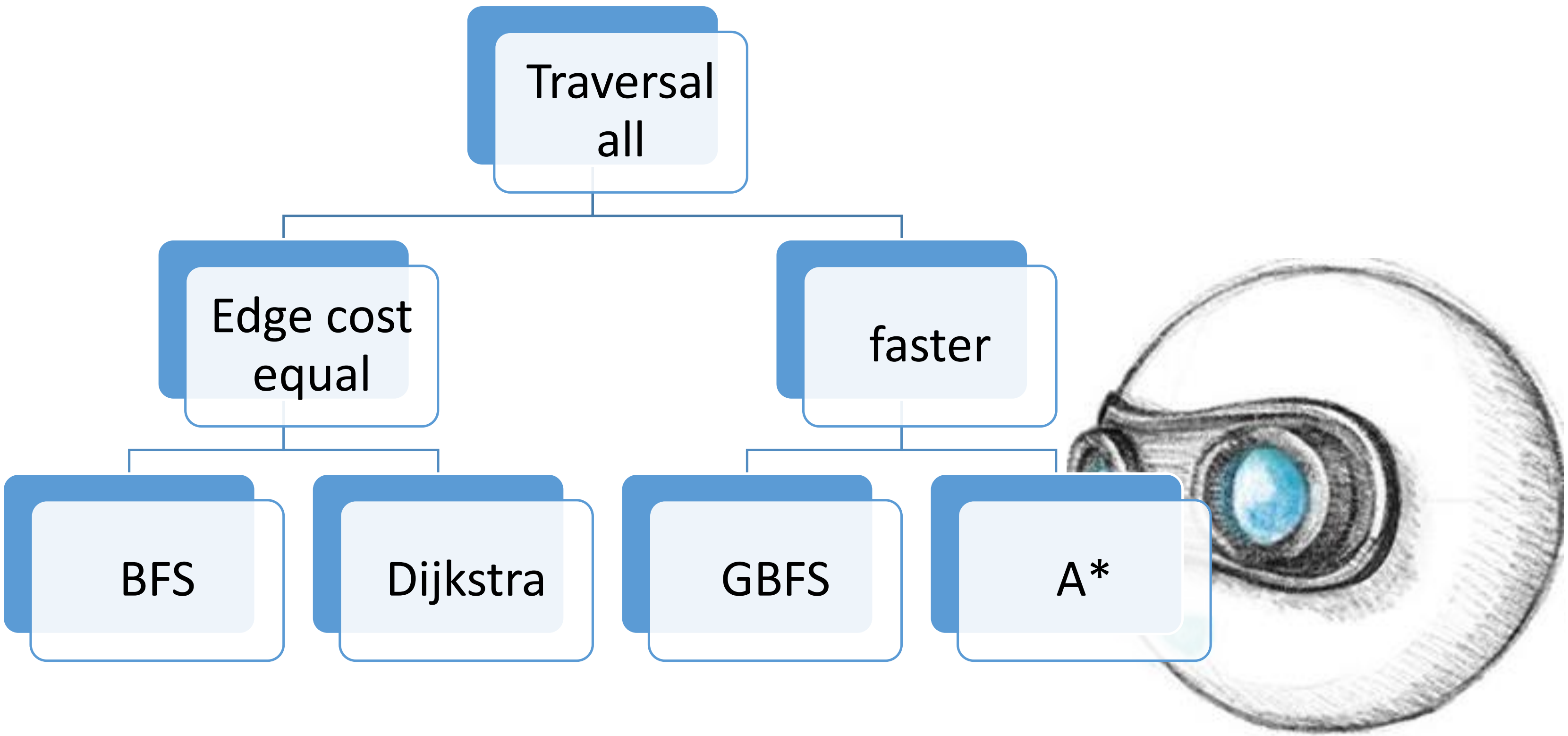
State	H(n)	f(n)	PPR
A	2.5	2.5	-
B	1.5	x	E
C	0	7	E
D	2.5	3.5	A
E	1.5	3.5	D



Planning

Search techniques :

Breadth First Search (BFS) (uniform edge cost)
Dijkstra’s algorithm (non-uniform edge cost) (surface condition)
Greedy Best First Search (nearest node **only**)
Admissible heuristic (A*) (nearest node & surface condition)



Ref.

- https://www.redblobgames.com/pathfinding/a-star/introduction.html