Informed (Heuristic) Search Strategies

Faculty of DS & AI
Autumn semester, 2025

Trong-Nghia Nguyen

Business AI Lab

2025-08

Content

- Evaluation function
- Informed (Heuristic) Search
  - Best first search
  - Beam search
  - Hill climbing search

Content

- Evaluation function
- Informed (Heuristic) Search
  - Best first search
  - Beam search
  - Hill climbing search

Evaluation function

- Estimate function, evaluating the level of good/bad, the ability to reach the destination of each state.

- Based on experience.

- For a state u,:
    - g(u) is the **cost** of going from the starting state (past information, **known**)
    - h(u) is the **remaining** cost to go to the goal (future information or heuristic, **estimate**). The **smaller** this value is the **better**.
    - g(u) + h(u) is the **total cost** of going from the starting state to the goal through that state (common information, **estimate**). The **smaller** this value is the **better**.

Evaluation function

- Can exploit only 1 or both evaluation information about the past and future
  - $f(u) = g(u) + h(u)$
  - $f(u) = g(u)$
  - $f(u) = h(u)$

- **Search techniques** using the evaluation function $h(u)$ are generally called **heuristic search**

- Optimal search techniques using the evaluation function $f(u)=g(u)+h(u)$

- The better the evaluation function (**closer to reality**), the more effective the search

Evaluation function

Heuristic Function

- Tic-Tac-Toe game

X		O
	X	

You are X. To calculate the heuristic (probability of winning), should evaluate:

- How many paths (rows, columns, diagonals) are there that X can win?
- A valid path must have at least 1 X and no O

⇒ chance to win (Big or small) ?

- Row 1: X _ O → blocked by O → eliminate
- Row 2: _ X _ → no O → count 1
- Row 3: _ _ _ → no O → count 1
- Column 1: X _ _ → no O → count 1
- Column 2: _ X _ → no O → count 1
- Column 3: O _ _ → blocked by O → eliminate
- Diagonal 1: X X _ → no O → count 1
- Diagonal 2: O X _ → have O → eliminate

➡ Total: 5 valid paths → h(u) = 5

Evaluation function

Heuristic Function

- Tic-Tac-Toe game

X		O
O		

You are X. To calculate the heuristic (probability of winning), should evaluate:

- How many paths (rows, columns, diagonals) are there that X can win?
- A valid path must have at least 1 X and no O

- Row 1: X _ O → blocked by O → eliminate
- Row 2: _ _ _ → no O → count 1
- Row 3: O _ _ → blocked by O → eliminate
- Column 1: X _ O → blocked by O → eliminate
- Column 2: _ _ _ → no O → count 1
- Column 3: O _ _ → blocked by O → eliminate
- Diagonal 1: X _ _ → no O → count 1
- Diagonal 2: O _ O → have O → eliminate

Total: 3 valid paths → h(u) = 3

Evaluation function

Method 1: Calculate $h_1(u)$ total number of cells that differ from the target state

Heuristic Function

- 8-Puzzle game

2	8	3
1	6	4
7		5

state u

1	2	3
4	5	6
7	8	

goal state

h(u) ?

⇒ Shortest path ?

Pos	u	Goal	Wrong?
(1,1)	2	1	✅
(1,2)	8	2	✅
(1,3)	3	3	❌
(2,1)	1	4	✅
(2,2)	6	5	✅
(2,3)	4	6	✅
(3,1)	7	7	❌
(3,2)	_	8	✅ (not count _)
(3,3)	5	_	✅

➡ Total errors: 6 cells => $h_1(u) = 6$

Evaluation function

Method 2: Calculate $h_2(u)$ — Total Manhattan Distance

Heuristic Function

- 8-Puzzle game

2	8	3
1	6	4
7		5

state u

1	2	3
4	5	6
7	8	

goal state

h(u) ?

Shortest path ?

Cell number	Pos (u)	Pos (goal)	Distance
1	(2,1)	(1,1)	1
2	(1,1)	(1,2)	1
3	(1,3)	(1,3)	0
4	(2,3)	(2,1)	2
5	(3,3)	(2,2)	2
6	(2,2)	(2,3)	1
7	(3,1)	(3,1)	0
8	(1,2)	(3,2)	2

Total distance: 1 + 1 + 0 + 2 + 2 + 1 + 0 + 2 = 9 => $h_2(u) = 9$

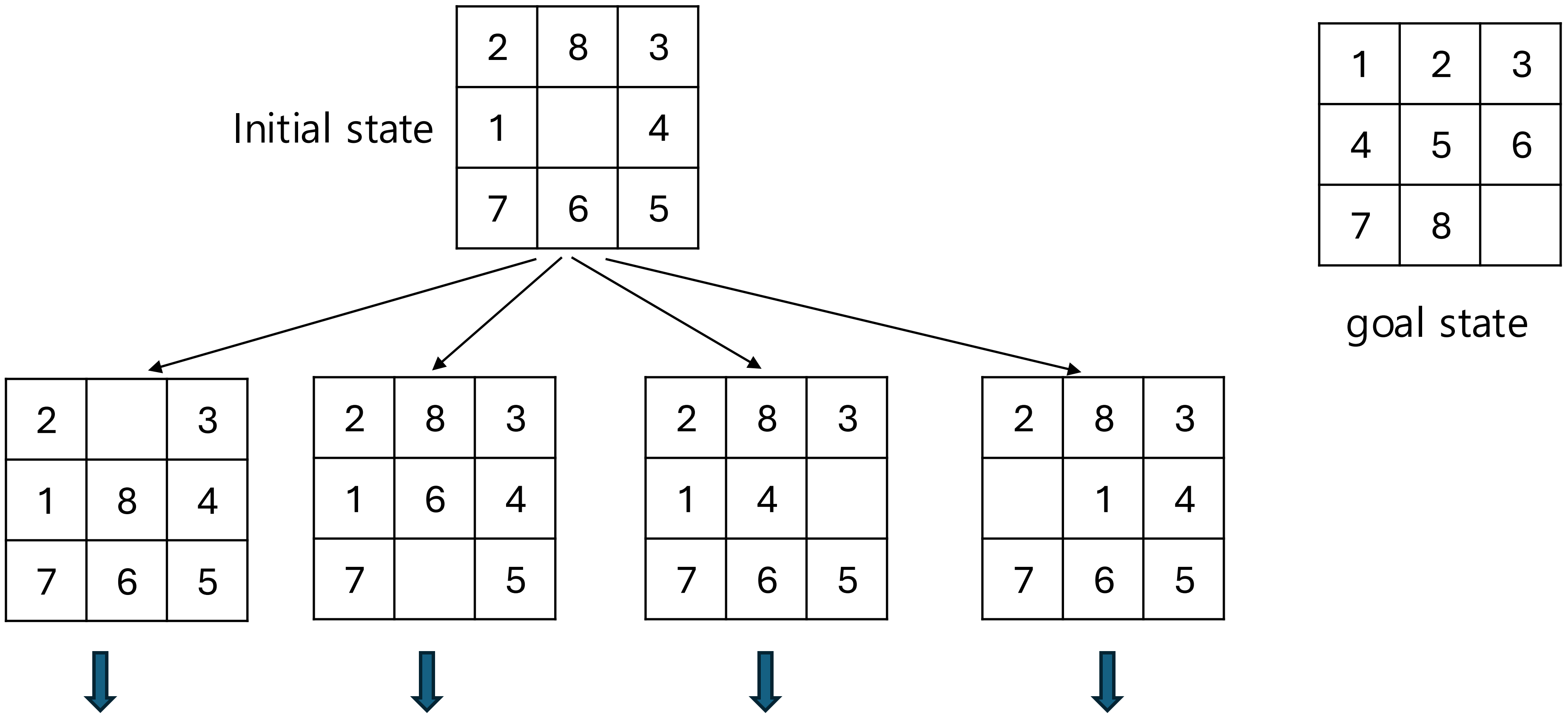
Evaluation function

Compare h_1 and h_2

Heuristic	Definition	Meaning	Advantages	Limitations
h_1	Number of tiles that are not in their goal position (ignores the blank)	A coarse estimate: counts how many tiles are "wrong"	Very simple, fast to compute	Too rough, cannot distinguish between tiles slightly misplaced vs. far away
h_2	Sum of the Manhattan distances of each tile from its goal position (ignores the blank)	A more realistic estimate: how far each tile needs to move	More accurate, closer to the real cost, **usually expands fewer states**	Slightly more expensive to compute than h_1

Evaluation function

Compare h_1 and h_2



g(u) = 1 state u

$f_1(u) = g(u) + h_1(u)$	7	7	7	7
$f_2(u) = g(u) + h_2(u)$	10	10	10	12

Evaluation function

Compare h_1 and h_2

Expanded states at $g = 2$

Move sequence	State at $g = 2$	h_2	$f_2 = 2 + h_2$
Up → Left	`_ 2 3 / 1 8 4 / 7 6 5`	**8**	**10**
Up → Right	`2 3 _ / 1 8 4 / 7 6 5`	10	12
Down → Left	`2 8 3 / 1 6 4 / _ 7 5`	10	12
Down → Right	`2 8 3 / 1 6 4 / 7 5 _`	**8**	**10**
Right → Up	`2 8 _ / 1 4 3 / 7 6 5`	10	12
Right → Down	`2 8 3 / 1 4 5 / 7 6 _`	**8**	**10**

h_2 clearly distinguishes "better" states (closer to the goal) than h_1, so in A* it is common to expand states with low f_2 first.

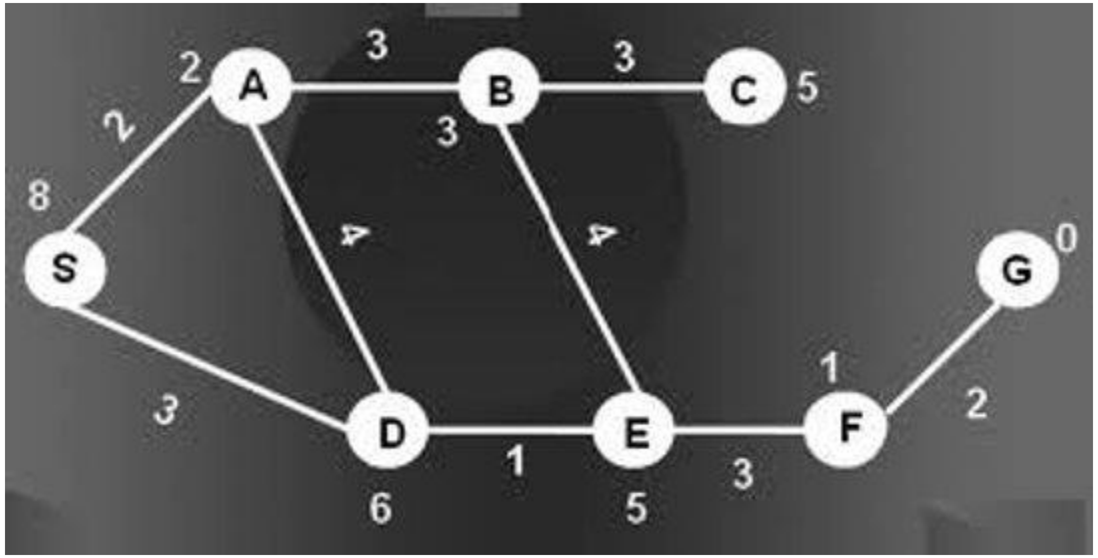
Evaluation function

Mapping to searching problem

Traveler's problem: find the shortest path from a starting city to a destination city

Evaluation function

- Past: g(u) = cost of traveling from starting city to city u
- Heuristic: h(u) = travel cost at vertex u.



Evaluation function

Searching with evaluation function

- **Heuristic Search (Experience-Based Search):** uses the evaluation function f(u)=h(u)
  - ➢ *Best-First Search* = Breadth-First Search + h(u)
  - ➢ *Hill-Climbing Search* = Depth-First Search + h(u)

- **Optimal Search:** uses the evaluation function f(u)=g(u)+h(u)
  - ➢ *A** = Best-First Search + f(u)
  - ➢ *Branch and Bound* = Hill-Climbing Search + f(u)

Content

- Evaluation function
- Informed (Heuristic) Search
  - Best first search
  - Beam search
  - Hill climbing search

Informed (Heuristic) Search

Best first search (BestFS)

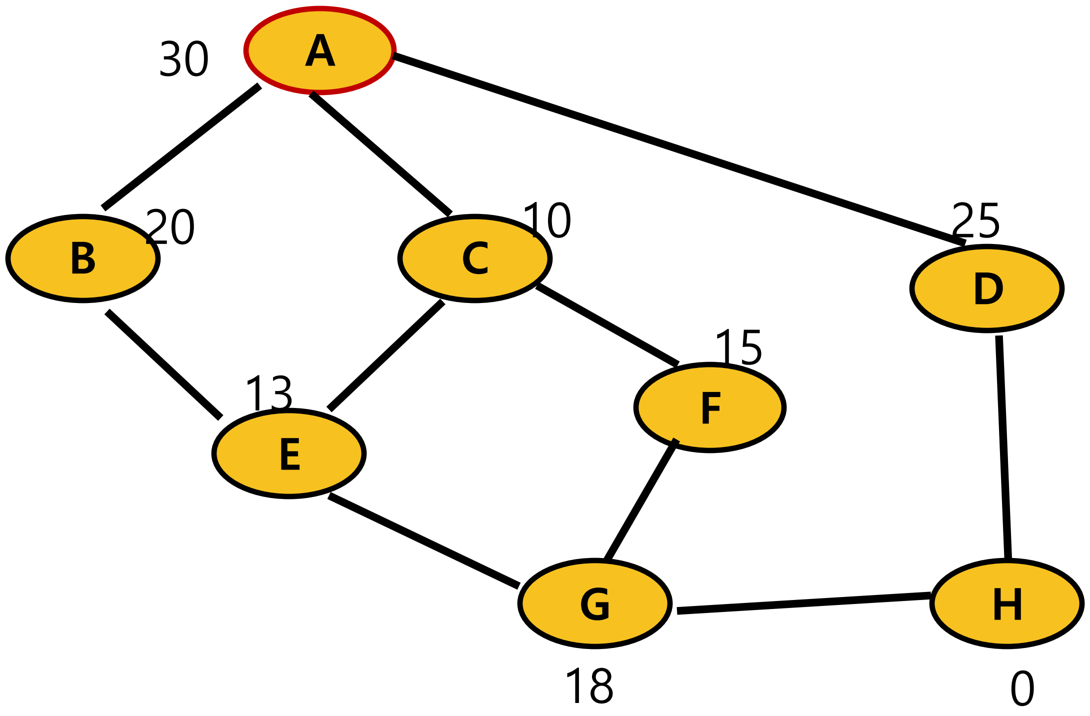
- **BestFS** = A **BFS** guided by the evaluation **function h(u)** (this mean f(u) = h(u)), in this case, could be call as Greedy best-first search (GBFS).
- **OPEN list**: set of nodes to be expanded, sorted in ascending order of the evaluation function.
- At each step:
  - Select node uuu in OPEN with the smallest evaluation value.
  - **Insert neighbors of u** into OPEN and **keep it sorted by the evaluation function**.

```
procedure Best_First_Search
Begin
 1. Initialize list OPEN = {initial state};
 2. while true do
     2.1 if (OPEN is empty) then {search failed; stop};
     2.2 Remove state u from the beginning of the OPEN list;
     2.3 if u is the end state then {search succeeded; stop};
     2.4 Insert adjacent vertices of u into OPEN so that OPEN is sorted in the ascending order of the evaluation function g;
   end
```

Informed (Heuristic) Search

Best first search (BestFS)

Find the shortest path from A → H

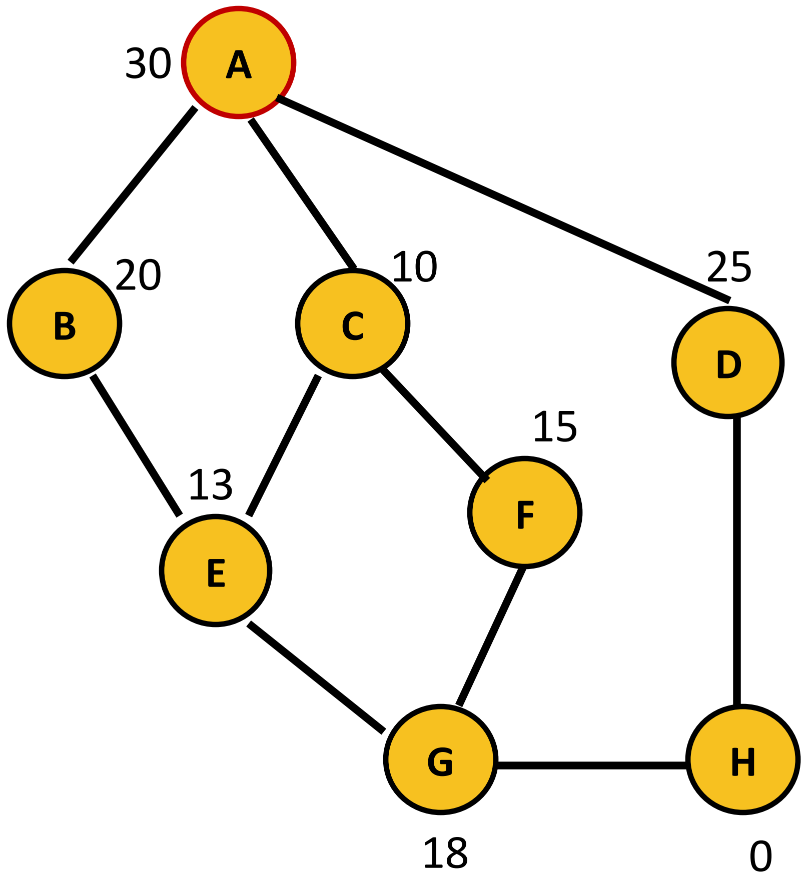


The value associated with each node is the heuristic evaluation h(u)

Informed (Heuristic) Search

Best first search (BestFS)

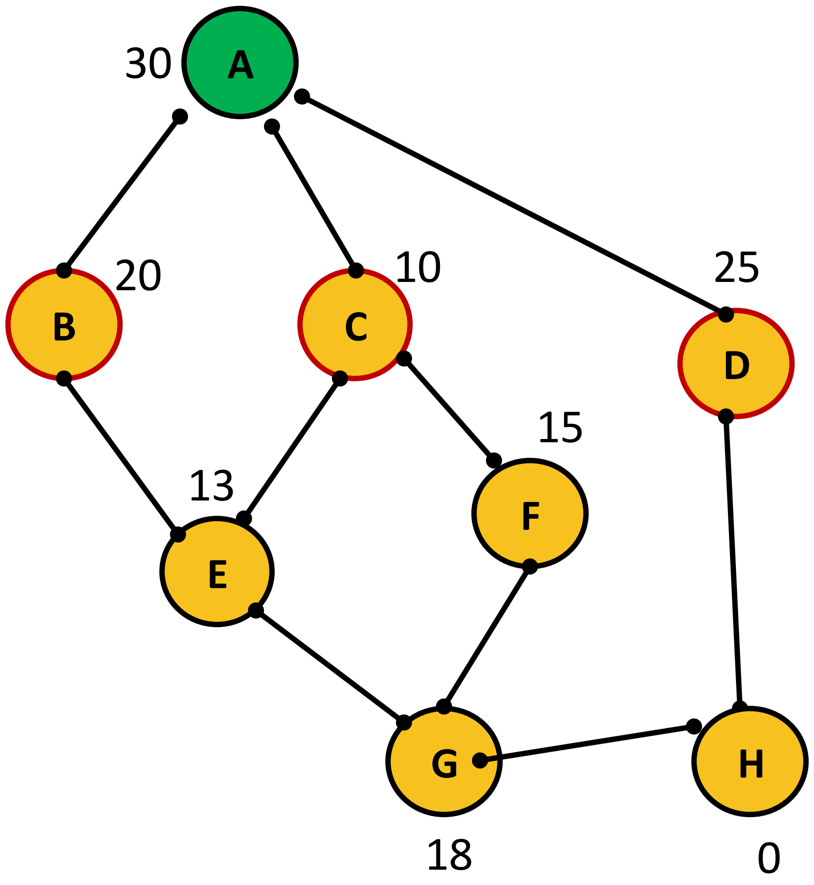
Step	u	Egde(u)	OPEN
0			A30



Informed (Heuristic) Search

Best first search (BestFS)

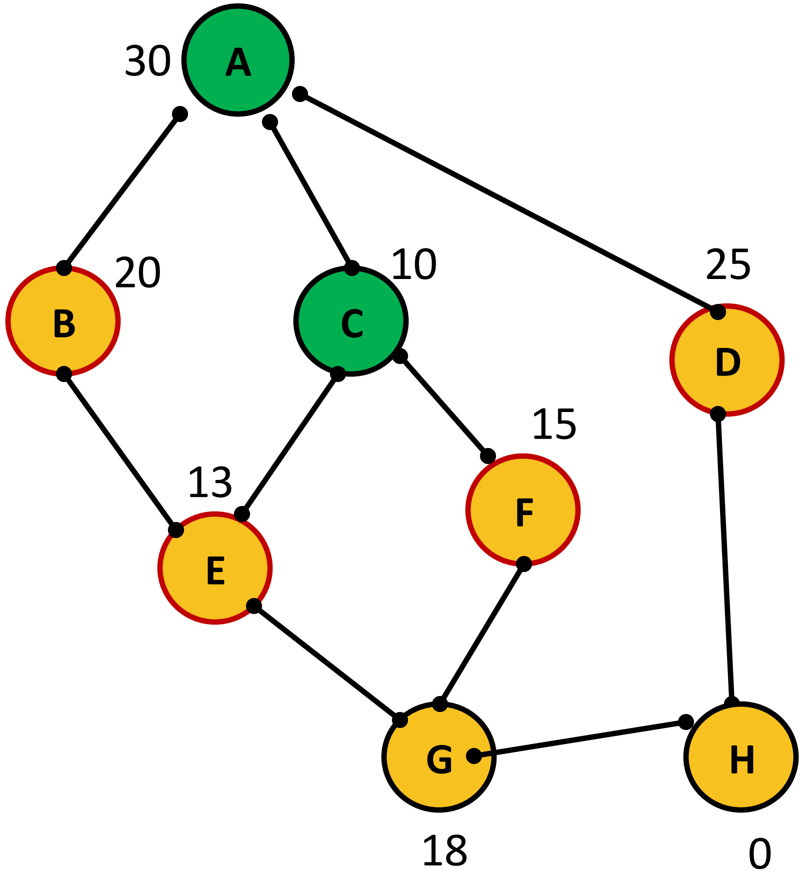
Step	u	Egde(u)	OPEN
0			A^{30}
1	A^{30}	B^{20},C^{10},D^{25}	C^{10}, B^{20},D^{25}



Informed (Heuristic) Search

Best first search (BestFS)

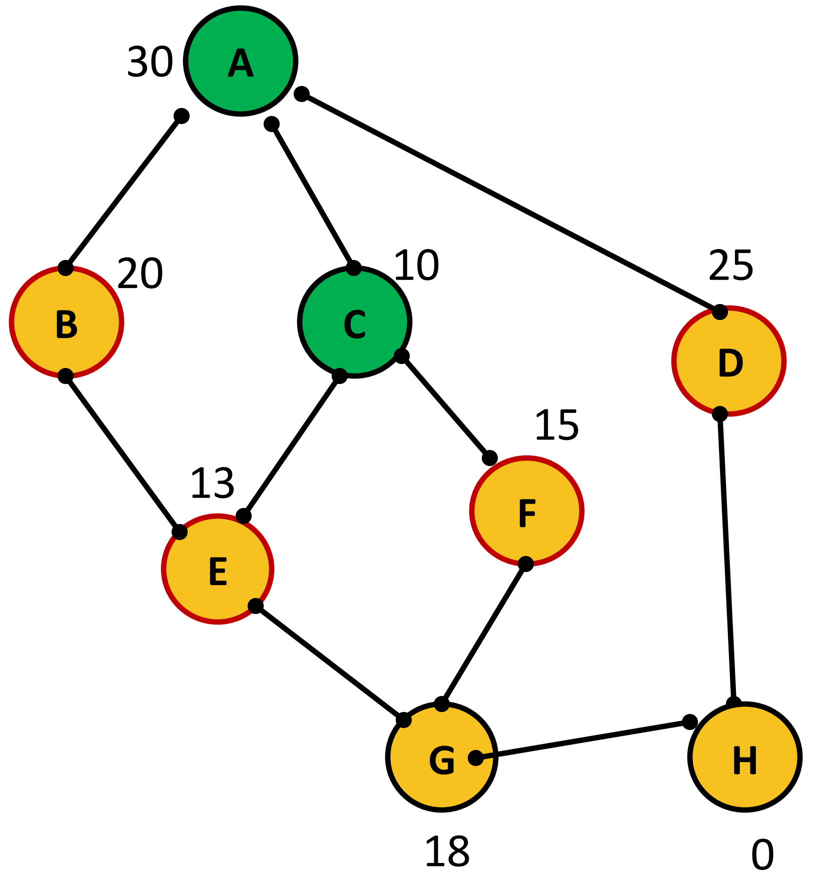
Step	u	Edge(u)	OPEN
0			A^{30}
1	A^{30}	B^{20},C^{10},D^{25}	C^{10}, B^{20},D^{25}
2	C^{10}	A^{30}, E^{13},F^{15}	B^{20},D^{25},



Informed (Heuristic) Search

Best first search (BestFS)

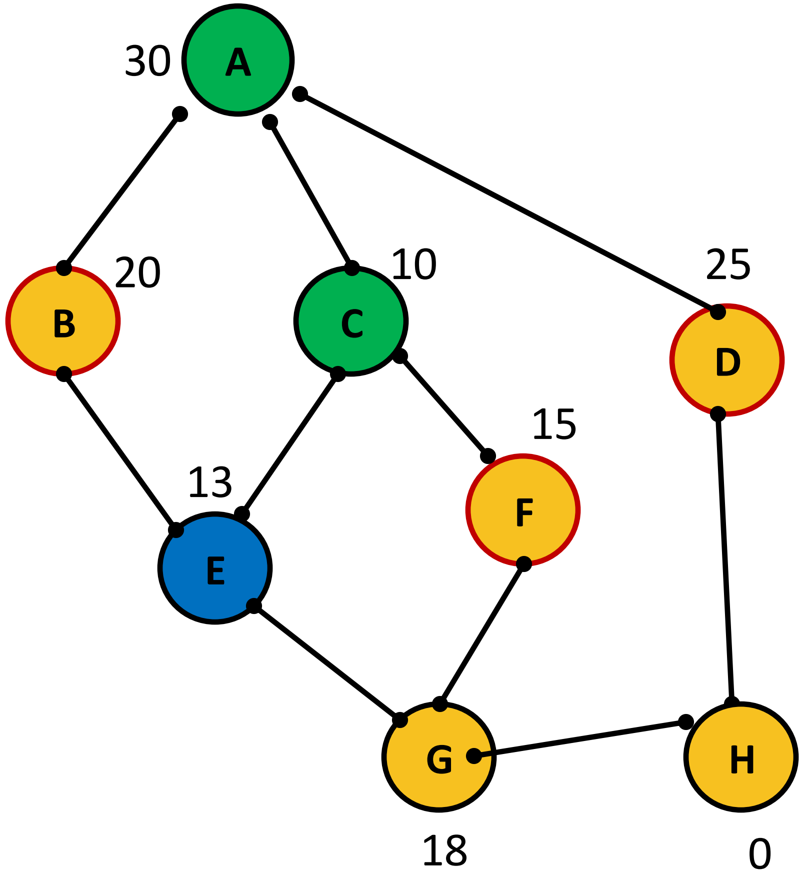
Step	u	Edge(u)	OPEN
0			A^{30}
1	A^{30}	B^{20},C^{10},D^{25}	C^{10}, B^{20},D^{25}
2	C^{10}	A^{30}, E^{13},F^{15}	E^{13},F^{15},B^{20},D^{25}



Informed (Heuristic) Search

Best first search (BestFS)

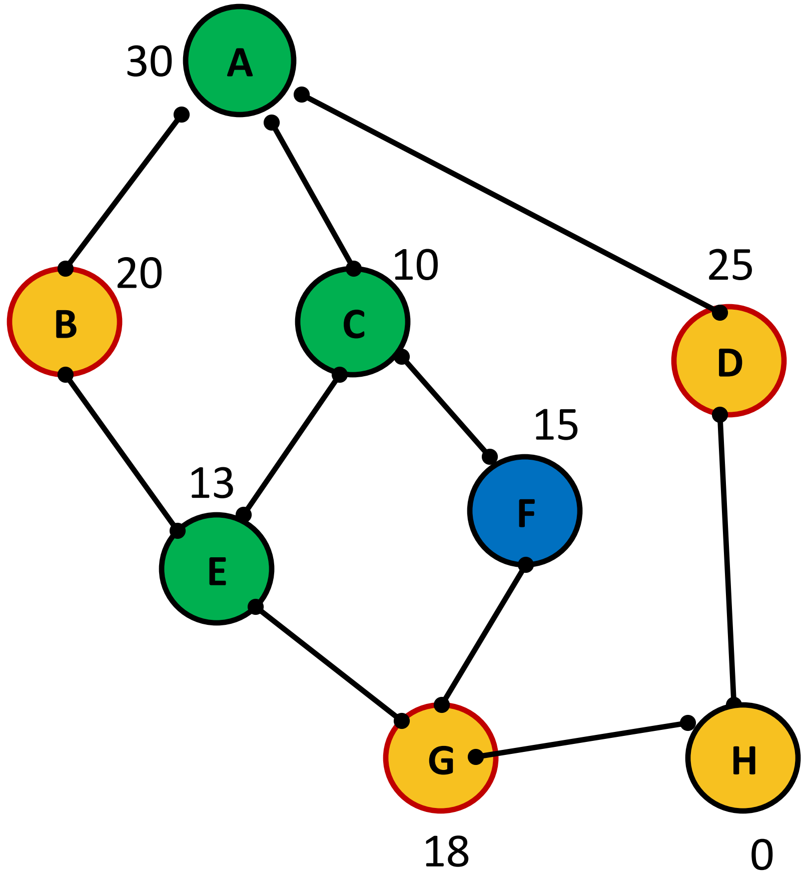
Step	u	Edge(u)	OPEN
0			A^{30}
1	A^{30}	B^{20},C^{10},D^{25}	C^{10}, B^{20},D^{25}
2	C^{10}	A^{30}, E^{13},F^{15}	~~E^{13}~~,F^{15},B^{20},D^{25},
3	E^{13}	G^{18}, B^{20}, C^{10}	F^{15},B^{20},D^{25}



Informed (Heuristic) Search

Best first search (BestFS)

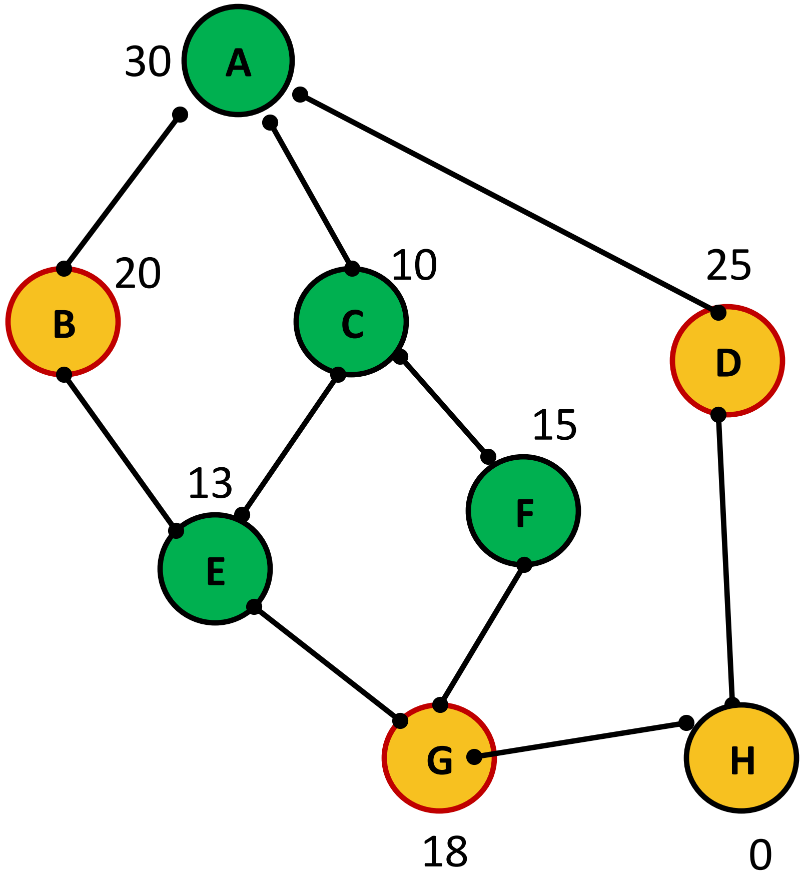
Step	u	Edge(u)	OPEN
0			A^{30}
1	A^{30}	B^{20},C^{10},D^{25}	C^{10}, B^{20},D^{25}
2	C^{10}	A^{30}, E^{13},F^{15}	E^{13},F^{15},B^{20},D^{25},
3	E^{13}	G^{18}, B^{20}, C^{10}	~~F^{15}~~, G^{18},B^{20},D^{25}
4	F^{15}		G^{18},B^{20},D^{25}



Informed (Heuristic) Search

Best first search (BestFS)

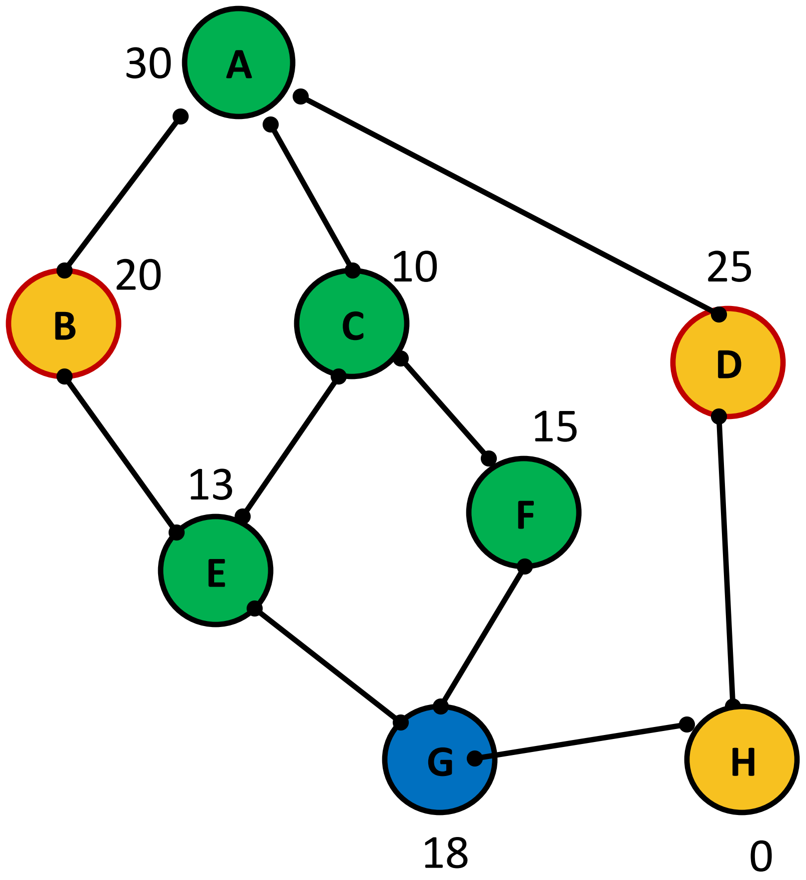
Step	u	Edge(u)	OPEN
0			A^{30}
1	A^{30}	B^{20}, C^{10}, D^{25}	C^{10}, B^{20}, D^{25}
2	C^{10}	A^{30}, E^{13}, F^{15}	$E^{13}, F^{15}, B^{20}, D^{25},$
3	E^{13}	$G^{18,}$ B^{20}, C^{10}	~~F^{15}~~, G^{18}, B^{20}, D^{25}
4	F^{15}		G^{18}, B^{20}, D^{25}



Informed (Heuristic) Search

Best first search (BestFS)

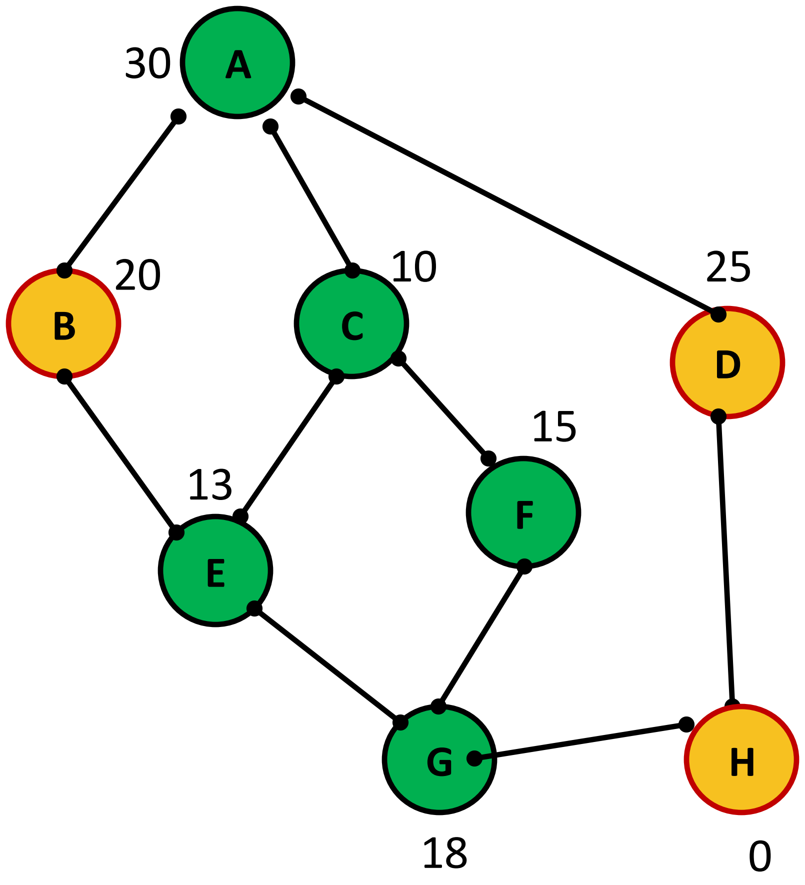
Step	u	Edge(u)	OPEN
0			A^{30}
1	A^{30}	B^{20},C^{10},D^{25}	C^{10}, B^{20},D^{25}
2	C^{10}	A^{30}, E^{13},F^{15}	E^{13},F^{15},B^{20},D^{25},
3	E^{13}	G^{18}, B^{20}, C^{10}	F^{15}, G^{18},B^{20},D^{25}
4	F^{15}	G^{18}, C^{10}	~~G~~^{18},B^{20},D^{25}
5	G^{18}		B^{20},D^{25}



Informed (Heuristic) Search

Best first search (BestFS)

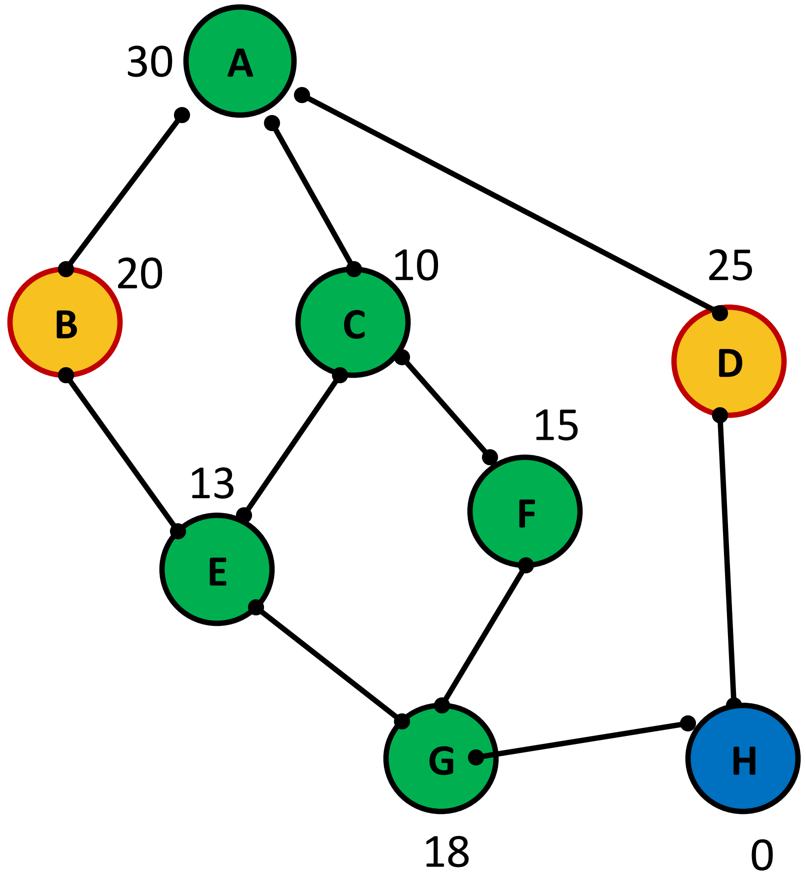
Step	U	Edge(u)	OPEN
0			A^{30}
1	A^{30}	B^{20}, C^{10}, D^{25}	C^{10}, B^{20}, D^{25}
2	C^{10}	A^{30}, E^{13}, F^{15}	$E^{13}, F^{15}, B^{20}, D^{25},$
3	E^{13}	G^{18}, B^{20}, C^{10}	F^{15}, G^{18}, B^{20}, D^{25}
4	F^{15}	G^{18}, C^{10}	~~G~~$^{18}, B^{20}, D^{25}$
5	G^{18}	H^{0}, E^{13}, F^{15}	H^{0}, B^{20}, D^{25}



Informed (Heuristic) Search

Best first search (BestFS)

Step	u	Edge(u)	OPEN
0			A^{30}
1	A^{30}	B^{20}, C^{10}, D^{25}	C^{10}, B^{20}, D^{25}
2	C^{10}	A^{30}, E^{13}, F^{15}	$E^{13}, F^{15}, B^{20}, D^{25}$
3	E^{13}	G^{18}, B^{20}, C^{10}	$F^{15}, G^{18}, B^{20}, D^{25}$
4	F^{15}	G^{18}, C^{10}	~~G^{18}~~$, B^{20}, D^{25}$
5	G^{18}	H^{0}, E^{13}, F^{15}	~~H^{0}~~$, B^{20}, D^{25}$
6	$H^{0} \equiv$		B^{20}, D^{25}

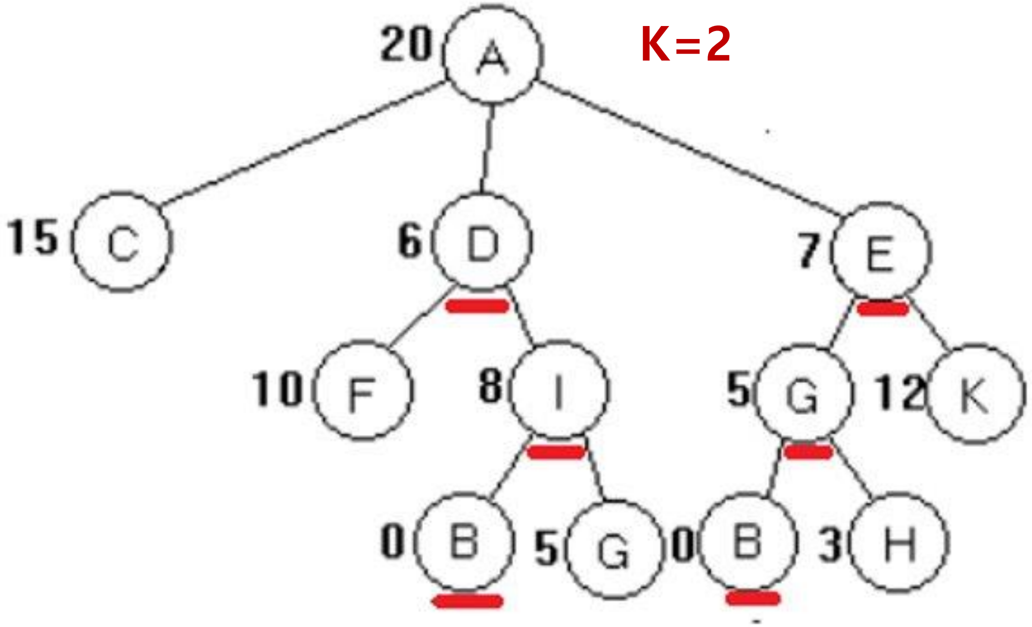


A-C-E-G-H

Informed (Heuristic) Search

Beam Search

- Similar to **Best-First Search**
- But it **restricts the number of nodes expanded** at each depth
- Main Idea:
  - Instead of expanding all successors of a node, Beam search only keeps the **k best successors** (according to heuristic value h(u)) at the same depth.
  - Parameter **k = beam width**



Best-First Search (BFS)	Beam Search
Expand node u, insert **all successors** v into OPEN	Expand node u, insert only some successors v so that the total number of nodes at the same depth in OPEN is $\leq k$

Informed (Heuristic) Search

Hill climbing

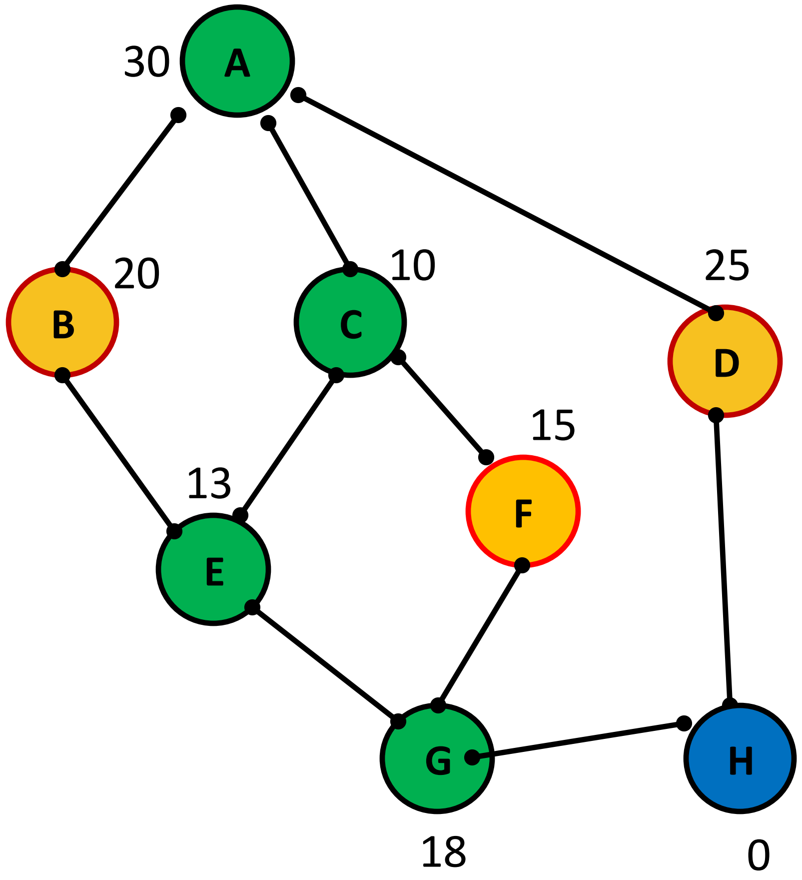
- Depth-first search under the direction of the evaluation function h(u) (remaining cost to reach the goal)
- At each step of the search:
    - Choose vertex u at the beginning of the OPEN list to traverse
    - After traversing vertex u:
    - Sort the **list of adjacent vertices** of u in the increasing order of the evaluation function
    - Insert **this adjacency list** at the **beginning** of OPEN

```
Procedure Hill_Climbing_Search begin
1. Initialize Open = {initial state};
2. while true do
     2.1 If (Open is empty) then  {failure
message; stop};
    2.2 Remove state u from the
beginning of the list Open;
    2.3 If (u is the end state) then
{success message; stop};
    2.4 for (each v adjacent to u) do
 add v to the list L;
    2.5 Sort the list L in ascending order
of the evaluation function;
    2.6 Insert L at the beginning of OPEN;
 end
```

Informed (Heuristic) Search

Hill climbing

Step	u	Edge(u)	L	Open
0				A30
1	A30	B20, C10, D25	C10, B20, D25	C10, B20, D25
2	C10	A30, E13, F15	E13, F15	E13, F15, B20, D25
3	E13	B20, C10, G18	G18	G18, F15, B20, D25
4	G18	E13, F15, H0	H0	H0, F15, B20, D25
5	H0			



A-C-E-G-H

Thank you!

You're now ready to explore the exciting world of AI!