CSE-411
Classical Search

Al Mehdi Saadat Chowdhury
North East University Bangladesh

Summer-2022

Outline

Outline

Dimension of Complexity of Classical Search

Dimension	Values
Modularity	flat, modular, hierarchical
Planning Horizon	non-planning, finite stage, indefinite stage, infinite stage
Representation	states, features, relations
Computational Limits	perfect rationality, bounded rationality
Learning	knowledge is given, knowledge is learned
Sensing Uncertainty	fully observable, partially observable
Effect Uncertainty	deterministic, stochastic
Preference	goals, complex preferences
Number of Agents	single agent, multiple agents
Interaction	offline, online

Application of Classical Search — Map

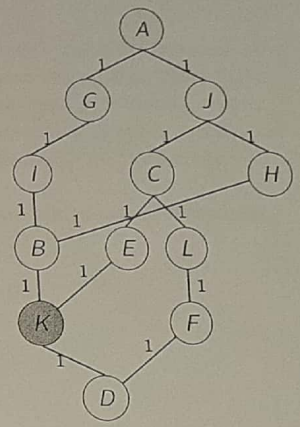
Map of Moulvibazar District

Application of Classical Search — Grid World

- Consider moving a knight on a 3×4 board, with start and goal states labeled as ***A*** and ***K*** respectively.

A	B	C	D
E	F	G	H
I	J	K	L

- The corresponding search graph (without repeating the same node in a path) is given in the right hand side:



Classical Search — Problem Formulation

- A search problem consists of:
  - a set of states.
    - a state contains all of the information necessary to predict the effects of an action along with its associated cost and to determine whether a state satisfies the goal
  - a subset of states called the start states
  - for each state, a set of actions available to the agent in that state
  - a goal specified as a Boolean function, $goal(s)$, that is true when state s satisfies the goal, in which case we say that s is a goal-state
  - a criterion that specifies the quality of an acceptable solution
    - a solution that is best according to some criterion is called an optimal solution. However, we do not always need an optimal solution.

Search — Generic Algorithm

Function search(G, s, $goal(n)$)

Data: Given: a list *frontier* that holds set of paths from the start state

Input: G: a graph with nodes N and edges A where each node N has 4 components - $N.state$, $N.parent$, $N.actions$, and $N.pathcost$; s: start node; $goal(n)$: a boolean function that tests for goal node

Output: path from s to a node for which $goal()$ is true; $\perp$ if there are no solution paths

```
1 frontier ← {⟨s⟩};
2 while frontier ≠ {} do
3     select and remove ⟨n_0, ..., n_k⟩ from frontier;
4     if goal(n_k) then
5         return ⟨n_0, ..., n_k⟩
6     frontier ← frontier ∪ {⟨n_0, ..., n_k, n⟩ : ⟨n_k, n⟩ ∈ A}
7 return ⊥
```

Outline

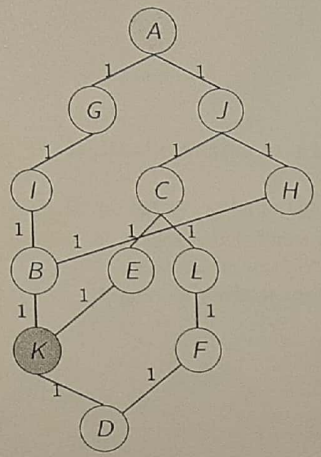
What and How

- "Uninformed search, also called blind search or unguided search, is a class of general purpose search algorithms that operate in a brute-force way. The term 'uninformed' means that they have no additional information about states beyond that provides in the problem definition." — Wikiversity.
- We will look at the following uninformed search techniques:
  - Breadth First Search (BFS)
  - Depth First Search (DFS)
  - Iterative Deepening Search (IDS)
    - Depth Limited Search (DLS)
  - Uniform Cost Search (UCS)

Breadth First Search (BFS)

- Treats the *frontier* as a FIFO queue.
- If the list of paths in the *frontier* is $[p_1, p_2, ..., p_r]$, then:
  - p_1 is selected. Its neighbors are added to the end of the FIFO queue, after p_r.
  - p_2 is selected next.

BFS — Example



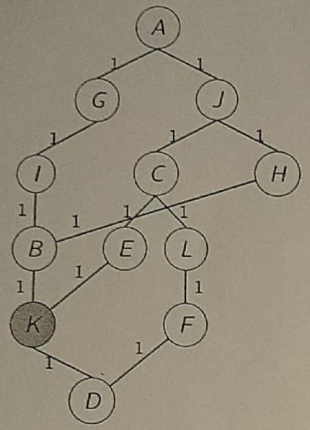
frontier = {A}
Visited = *None*

frontier = {G, J}
Visited = {A}

frontier = {J, I}
Visited = {A, G}

frontier = {I, C, H}
Visited = {A, G, J}

BFS — Example



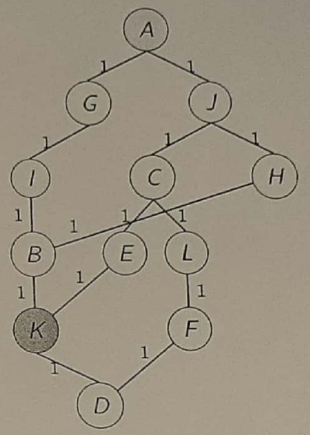
frontier = {C, H, B}
Visited = {A, G, J, I}

frontier = {H, B, E, L}
Visited = {A, G, J, I, C}

frontier = {B, E, L}
Visited = {A, G, J, I, C, H}

frontier = {E, L, K}
Visited = {A, G, J, I, C, H, B}

BFS — Example



frontier = {L, K}
Visited = {A, G, J, I, C, H, B, E}

frontier = {K, F}
Visited = {A, G, J, I, C, H, B, E, L}

Goal Found
frontier = {F}
Visited = {A, G, J, I, C, H, B, E, L, K}

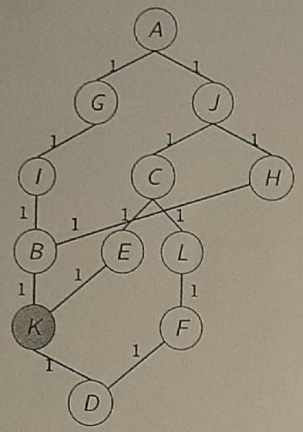
BFS — Complexity

- Does BFS guarantee to find the path with fewest arcs?
  - Yes.
- What happens on infinite graphs or on graphs with cycles if there is a solution?
  - For infinite graph, goal will be found if it is on a finite depth. For graph with cycles, the algorithm may not halt, but the goal will eventually be found.
- What is the time complexity as a function of the length of the path selected?
  - $O(B^L)$, where B is the branching factor, L is the length of the solution path.
- What is the space complexity as a function of the length of the path selected?
  - Exponential, $O(B^L)$. All nodes resides on the *frontier* memory.

Depth First Search (DFS)

- Treats the *frontier* as a stack.
- If the list of paths in the *frontier* is $[p_1, p_2, ...]$, then:
  - p_1 is selected. Paths that extend p_1 are added to the front of the stack (in front of p_2).
  - p_2 is only selected when all paths from p_1 have been explored.

DFS — Example



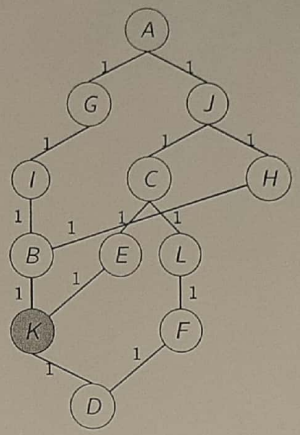
frontier = {*A*}
Visited = *None*

frontier = {*G*, *J*}
Visited = {*A*}

frontier = {*G*, *C*, *H*}
Visited = {*A*, *J*}

frontier = {*G*, *C*, *B*}
Visited = {*A*, *J*, *H*}

DFS — Example



frontier = {*G*, *C*, *K*}
Visited = {*A*, *J*, *H*, *B*}

Goal Found
frontier = {*G*, *C*}
Visited = {*A*, *J*, *H*, *B*, *K*}

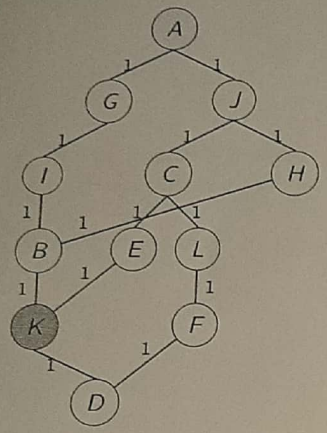
DFS — Complexity

- Does DFS guarantee to find the path with fewest arcs?
  - No.
- What happens on infinite graphs or on graphs with cycles if there is a solution?
  - DFS won't be able to find a solution. Therefore, DFS is not COMPLETE.
- What is the time complexity?
  - $O(B^M)$, where B is the branching factor, M is the maximum depth of the search graph.
- What is the space complexity as a function of the length of the path selected?
  - Linear, $O(L \times (B-1))$, L is the length of the path, B is the branching factor.

Iterative Deepening Search (IDS)

- IDS is nothing but running DFS iteratively.
- First apply DFS to depth-1 by building paths of length 1 in depth-first manner.
- If that does not find a solution, then build paths to depth-2, then depth-3, and so on until a solution is found.
- The algorithm is basically like BFS; however, rather than storing elements, it computes them from the frontier.
- IDS, combining benefits of both BFS and DFS, is the best uninformed search strategy known.

IDS — Example



Pass-1: *depthCount* = 1

frontier = {}
Visited = {A}
Goal not found. Hence, Failure.

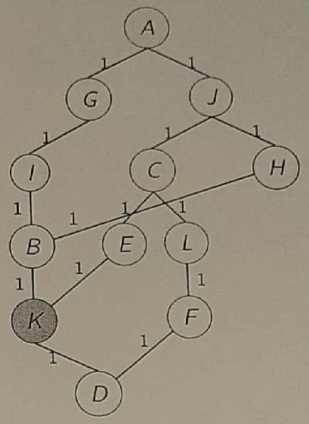
Pass-2: *depthCount* = 2

frontier = {G, J}
Visited = {A}

frontier = {G}
Visited = {A, J}

frontier = {}
Visited = {A, J, G}
Goal not found. Hence, Failure.

IDS — Example



Pass-3: *depthCount* = 3

frontier = {G, J}
Visited = {A}

frontier = {G, C, H}
Visited = {A, J}

frontier = {G, C}
Visited = {A, J, H}

frontier = {G}
Visited = {A, J, H, C}

Rest is your homework...

IDS — Complexity

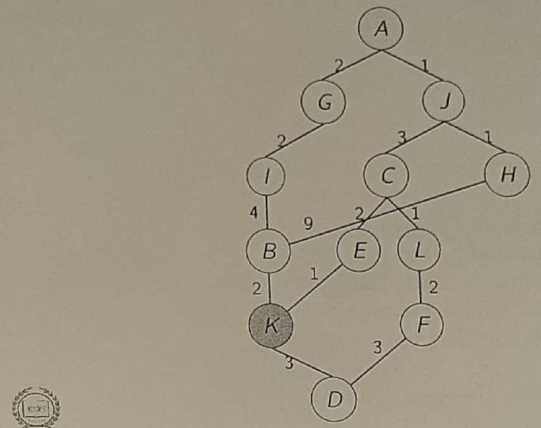
- Does IDS guarantee to find the path with fewest arcs?
  - Yes.
- What happens on infinite graphs or on graphs with cycles if there is a solution?
  - Like BFS, it will find the solution path.
- What is the time complexity as a function of the length of the path selected?
  - $O(B^L)$, where B is the branching factor, L is the length of the solution path.
- What is the space complexity as a function of the length of the path selected?
  - Linear, $O(L \times (B-1))$, L is the length of the path, B is the branching factor.

Depth Limited Search (DLS)

- If we fix the *depthCount* to certain depth (instead of iteratively incrementing depths), then we get DLS.

Uniform Cost Search (UCS)

- The *frontier* is a priority queue ordered by path cost. UCS always selects a path on the frontier with lowest cost.
- When arc costs are equal, UCS is nothing but BFS.



UCS — Example

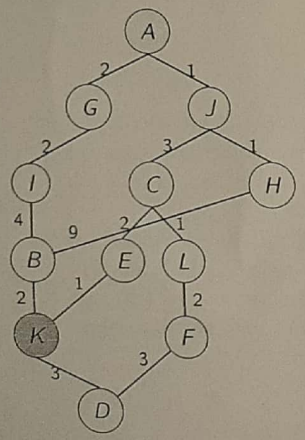
$frontier = \{A^{\{0\}}\}$
Visited = *None*

$frontier = \{J^{\{0+1=1\}}, G^{\{0+2=2\}}\}$
Visited = $\{A^{\{0\}}\}$

$frontier = \{G^{\{2\}}, H^{\{1+1=2\}}, C^{\{1+3=4\}}\}$
Visited = $\{A^{\{0\}}, J^{\{1\}}\}$

$frontier = \{H^{\{2\}}, C^{\{4\}}, I^{\{2+2=4\}}\}$
Visited = $\{A^{\{0\}}, J^{\{1\}}, G^{\{2\}}\}$

UCS — Example



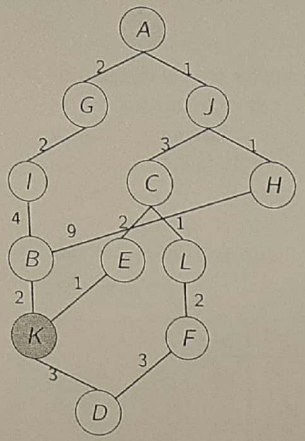
$f = \{C^{\{4\}}, I^{\{4\}}, B^{\{2+9=11\}}\}$
$V = \{A^{\{0\}}, J^{\{1\}}, G^{\{2\}}, H^{\{2\}}\}$

$f = \{I^{\{4\}}, L^{\{4+1=5\}}, E^{\{4+2=6\}}, B^{\{11\}}\}$
$V = \{A^{\{0\}}, J^{\{1\}}, G^{\{2\}}, H^{\{2\}}, C^{\{4\}}\}$

$f = \{L^{\{5\}}, E^{\{6\}}, B^{\{11\}}, B^{\{4+4=8\}}\}$
$V = \{A^{\{0\}}, J^{\{1\}}, G^{\{2\}}, H^{\{2\}}, C^{\{4\}}, I^{\{4\}}\}$

$f = \{E^{\{6\}}, F^{\{5+2=7\}}, B^{\{8\}}\}$
$V = \{A^{\{0\}}, J^{\{1\}}, G^{\{2\}}, H^{\{2\}}, C^{\{4\}}, I^{\{4\}}, L^{\{5\}}\}$

UCS — Example



$f = \{F^{\{7\}}, K^{\{6+1=7\}}, B^{\{8\}}\}$
$V = \{A^{\{0\}}, J^{\{1\}}, G^{\{2\}}, H^{\{2\}}, C^{\{4\}}, I^{\{4\}}, L^{\{5\}}, E^{\{6\}}\}$

$f = \{K^{\{7\}}, B^{\{8\}}, D^{\{7+3=10\}}\}$
$V = \{A^{\{0\}}, J^{\{1\}}, G^{\{2\}}, H^{\{2\}}, C^{\{4\}}, I^{\{4\}}, L^{\{5\}}, E^{\{6\}}, F^{\{7\}}\}$

$f = \{B^{\{8\}}, D^{\{10\}}\}$
$V = \{A^{\{0\}}, J^{\{1\}}, G^{\{2\}}, H^{\{2\}}, C^{\{4\}}, I^{\{4\}}, L^{\{5\}}, E^{\{6\}}, F^{\{7\}}, K^{\{7\}}\}$
Goal Found.

UCS — Complexity

- Does UCS guarantee to find the path with fewest arcs? least-cost path?
  - No for fewest arcs. Yes for least-cost path.
- What happens on infinite graphs or on graphs with cycles if there is a solution?
  - Solution will be found.
- What is the time complexity as a function of the length of the path selected?
  - Exponential, same as BFS.
- What is the space complexity as a function of the length of the path selected?
  - Exponential, same as BFS.

Outline

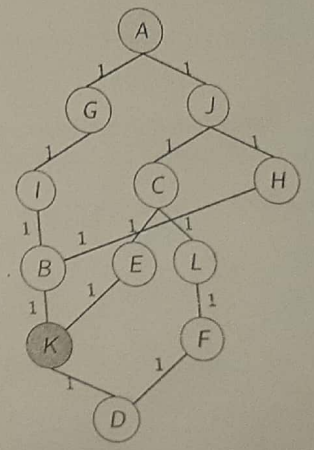
What is Heuristics?

- A heuristic function $h(n)$ takes a node n and returns a non-negative real number that is an estimate of the cost of the least-cost path from node n to a goal node.
- Admissible Heuristic: The function $h(n)$ is an admissible heuristic if $h(n)$ is always less than or equal to the actual cost of a lowest-cost path from node n to a goal node (always under-estimate, never over-estiamte).

Is the following heuristics admissible?

- Consider moving a knight on a 3×4 board, with start and goal states labeled as **A** and **K** respectively. In this grid world, subscript of each letter denotes heuristic values.

$A_{[3]}$	$B_{[1]}$	$C_{[1]}$	$D_{[1]}$
$E_{[1]}$	$F_{[2]}$	$G_{[2]}$	$H_{[2]}$
$I_{[1]}$	$J_{[2]}$	$K_{[0]}$	$L_{[3]}$



But, how to find heuristic values?

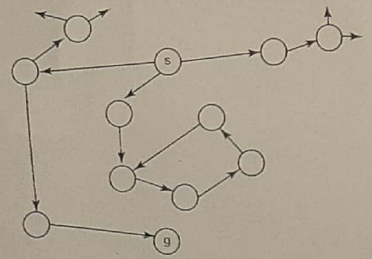
- Always use the value that can be readily obtained about a node.
- A standard way to derive a heuristic function is to solve a simpler problem (one with fewer constraints) and use the cost to the goal of this simplified problem as heuristic values of the original problem.
- For example:
  - When cost is distance and solution is a path to the goal, you can use straight-line Euclidean distance between nodes as heuristics.

Informed search algorithms we will read

- Heuristic Depth First Search (HDFS)
- Greedy Best First Search (GBFS)
- A* Search
- Iterative Deepening A* Search (IDA*)

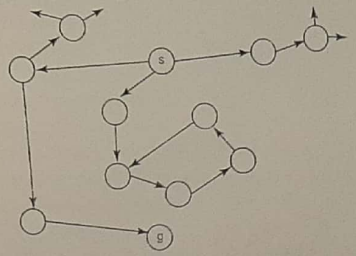
Heuristic Depth First Search (HDFS)

- Add neighbors to the *frontier* so that the best neighbor is selected first.
- What happens to the following graph? s is the starting state, g is the goal state, Euclidean distance is the heuristic function.



Greedy Best First Search (GBFS)

- Always select a path on the *frontier* with the lowest heuristic value.
- What happens to the following graph? s is the starting state, g is the goal state, Euclidean distance is the heuristic function.

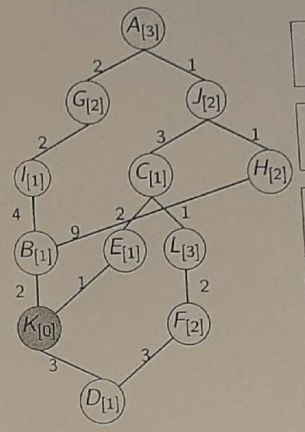


A* Search

- Treats the *frontier* as a priority queue ordered by $f(p) = cost(p) + h(p)$.

$$\underbrace{\underbrace{start \xrightarrow{\text{path } p} n}_{cost(p)} \underbrace{\xrightarrow{\text{estimate}} goal}_{h(p)}}_{f(p)}$$

A* Search — Example



frontier = $\{A^{\{0+3=3\}}\}$
Visited = *None*

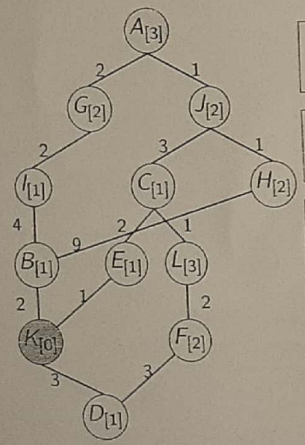
frontier = $\{J^{\{1+2=3\}}, G^{\{2+2=4\}}\}$
Visited = $\{A^{\{3\}}\}$

frontier = $\{G^{\{4\}}, H^{\{2+2=4\}}, C^{\{4+1=5\}}\}$
Visited = $\{A^{\{3\}}, J^{\{3\}}\}$

frontier = $\{H^{\{4\}}, C^{\{5\}}, I^{\{4+1=5\}}\}$
Visited = $\{A^{\{3\}}, J^{\{3\}}, G^{\{4\}}\}$

frontier = $\{C^{\{5\}}, I^{\{5\}}, B^{\{11+1=12\}}\}$
Visited = $\{A^{\{3\}}, J^{\{3\}}, G^{\{4\}}, H^{\{4\}}\}$

A* Search — Example



$f = \{I^{\{5\}}, E^{\{6+1=7\}}, L^{\{5+3=8\}}, B^{\{12\}}\}$
$V = \{A^{\{3\}}, J^{\{3\}}, G^{\{4\}}, H^{\{4\}}, C^{\{5\}}\}$

$f = \{E^{\{7\}}, L^{\{8\}}, B^{\{8+1=9\}}, B^{\{12\}}\}$
$V = \{A^{\{3\}}, J^{\{3\}}, G^{\{4\}}, H^{\{4\}}, C^{\{5\}}, I^{\{5\}}\}$

$f = \{K^{\{7+0=7\}}, L^{\{8\}}, B^{\{9\}}\}$
$V = \{A^{\{3\}}, J^{\{3\}}, G^{\{4\}}, H^{\{4\}}, C^{\{5\}}, I^{\{5\}}, E^{\{7\}}\}$

$f = \{L^{\{8\}}, B^{\{9\}}\}$
$V = \{A^{\{3\}}, J^{\{3\}}, G^{\{4\}}, H^{\{4\}}, C^{\{5\}}, I^{\{5\}}, E^{\{7\}}, K^{\{7\}}\}$
Goal Found!

A* Search — Complexity

- Does A* search guarantee to find the path with fewest arcs? least-cost path?
  - No for fewest arcs. Yes for least-cost path.
- What happens on infinite graphs or on graphs with cycles if there is a solution?
  - Solution will be found. If $h(n)$ is admissible, the first solution found will also be an optimal solution.
- What is the time complexity as a function of the length of the path selected?
  - All such path for which $cost(p) + h(p) < L$ will be expanded, and some paths for which $cost(p) + h(p) = L$ will also be expanded. This is exponential time bound.
- What is the space complexity as a function of the length of the path selected?
  - Exponential.

Iterative Deepening A* Search (IDA*)

- IDA* bounds on the value of $f(n)$. Initially it starts with the value of $f(s)$ where s is the start node.
- The search then carries out a depth-first depth-bounded search but never expands a path with a higher f-value than the current bound.
- If the search fails and the bound was reached, the next bound is the minimum of the f-values that exceeded the previous bound.

Outline

Cycle Pruning

- Check whether the last node on the path already appears earlier on the path from the start node to that node. Paths $\langle n_0, \ldots, n_k, n \rangle$ where $n \in \{n_0, \ldots, n_k\}$ are not added to the frontier.
- For depth-first methods, complexity is constant, by storing elements of the current path as a set.
- For search strategies that maintains multiple paths (exponential space), complexity is linear in the length of the path.

Multiple Path Pruning

- Maintain an explored set or closed set. When a path $\langle n_0, \ldots, n_k \rangle$ is selected, if n_k is already in the explored set, then this path will be discarded.
- With MPP, to ensure the search algorithm can still find a lowest-cost path to goal, one of the following must hold:
  - Make sure the first path found to any node is a lowest-cost path to that node.
  - Consider in a path $\langle s, \ldots, n, \ldots, m \rangle$, our path p has higher cost in $\langle s, \ldots, n \rangle$ than the current path p' between $\langle s, \ldots, n \rangle$. Then we can replace p with p' which will lead to a lower-cost path.

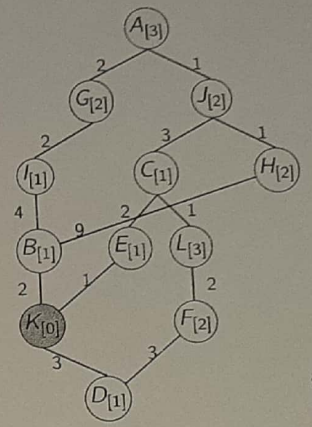
A* Search and Consistent Heuristic

- A* Search does not guarantee that when a path to a node is selected for the first time, it is the lowest-cost path to that node.
  - With an admissible heuristics, A* Search guarantee this for every path to a goal node, but not for any node.
- Consistent Heuristic: A heuristic $h(n)$ is consistent if it satisfies the constraint $h(n) \leq cost(n, n') + h(n')$ for any two nodes n and n'.
- Monotone Restriction: Consistency can be guaranteed if the heuristic function satisfies $h(n) \leq cost(n, n') + h(n')$ for any arc $\langle n, n' \rangle$.

Is the following heuristics consistent?

- Consider moving a knight on a 3×4 board, with start and goal states labeled as ***A*** and ***K*** respectively. In this grid world, subscript of each letter denotes heuristic values.

$A_{[3]}$	$B_{[1]}$	$C_{[1]}$	$D_{[1]}$
$E_{[1]}$	$F_{[2]}$	$G_{[2]}$	$H_{[2]}$
$I_{[1]}$	$J_{[2]}$	$K_{[0]}$	$L_{[3]}$



Outline

Summary of All Search Algorithms

Strategy	Selection from Frontier	Path Found	Time	Space
BFS	FIFO	Fewest arc	Exp	Exp
DFS	LIFO	No	Exp	Linear
IDS	-	Fewest arc	Exp	Linear
UCS	minimal $cost(p)$	Least cost	Exp	Exp
HDFS	LIFO	No	Exp	Linear
GBFS	minimal $h(p)$	No	Exp	Exp
A*	minimal $cost(p)+h(p)$	Least cost	Exp	~~Linear~~ Exp
IDA*	-	Least cost	Exp	Linear