National University of Singapore
School of Computing
CS2109S: Introduction to AI and Machine Learning
Semester 2, 2024/2025

Tutorial 2
Informed Search

These questions will be discussed during the tutorial session. Please be prepared to answer them.

Summary of Key Concepts

In this tutorial, we will discuss and explore the following key learning points/lessons from Lecture:

1. Heuristic functions
   (a) Formulation of heuristic functions
   (b) Admissible and consistent heuristics
   (c) Dominance
2. A* Search
   (a) Optimality of `SEARCH` vs `SEARCH WITH VISITED MEMORY`

Problems

A Pacman

Pacman is a maze action video game. For this question, we will take a look at a simplified version of Pacman. In this version, Players control Pacman, with the objective of eating all the pellets in the maze while executing the least amount of moves. More formally, the initial state is a game state where Pacman is at a starting location and pellets are scattered around the maze (refer to Figure 1). Available actions are 4-directional movement unless blocked by walls, and if Pacman moves to a square with a pellet, he will automatically eat it with no additional cost (each movement has a cost of 1). The goal state is when Pacmac has eaten all the pellets. Recall that A* search finds an optimal solution with an admissible heuristic.

1. Thus, devise a non-trivial admissible heuristic for this problem.

Solution:

A simple admissible heuristic $h(n)$ would be the number of pellets left at state n. Since Pacman can only eat at most one pellet per move, a state with n pellets will at least be n pellets away from the goal state. This means that $h(n) \leq h^*(n)$ where h^* is the true cost.

A way to formulate admissible heuristics is to look at the cost of the optimal solution to a relaxed version of the problem — in this case, the relaxed problem

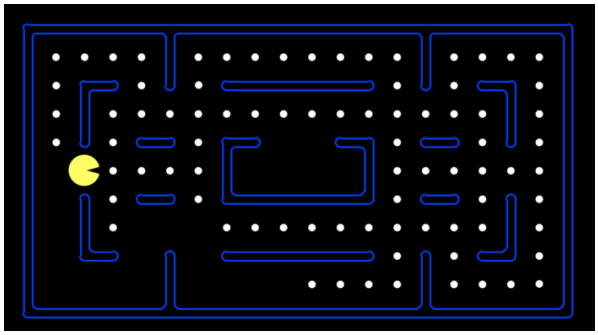


Figure 1: A state of Pacman.

would be that Pacman can move to any square without restrictions.

More formally, we can view it as adding edges (less restrictions means extra moves) to the original graph. By definition, a solution to the original problem is also a solution to the relaxed problem as the original edges remain in the relaxed problem, but may not be optimal as the new edges may provide better paths to the goal node. This also means that the cost of optimal solution to the relaxed problem is always lower than the cost of the optimal solution to the original problem, making it an admissible heuristic for the original problem.

Extra: since the heuristic is an exact cost for the relaxed problem, it satisfies the triangle inequality which also makes it a consistent heuristic as well.

B Route Finding

We learnt that for route-finding problems, a simple admissible heuristic is the straight line distance. More formally, given a graph $G = (V, E)$ where each node v_n having coordinates (x_n, y_n), each edge (v_i, v_j) having weight equals to the distance between v_i and v_j, and a unique goal node v_g with coordinates (x_g, y_g), the straight line distance heuristic is given by:

$$h_{SLD}(n) = \sqrt{(x_n - x_g)^2 + (y_n - y_g)^2}$$

Consider the following two heuristic functions

$$h_1(n) = \max\{|x_n - x_g|, |y_n - y_g|\}$$

$$h_2(n) = |x_n - x_g| + |y_n - y_g|$$

1. Is $h_1(n)$ an admissible and consistent heuristic? If yes, prove it; otherwise, provide a counterexample.

Solution:

Admissibility:
Let $h^*(n)$ be the actual path distance from v_n to v_g. Since h_{SLD} is admissible, we

have

$$h^*(n) \geq h_{SLD}(n) = \sqrt{(x_n - x_g)^2 + (y_n - y_g)^2}$$

Note that

$$\sqrt{(x_n - x_g)^2 + (y_n - y_g)^2} \geq \sqrt{(x_n - x_g)^2} = |x_n - x_g|$$

and similarly, $h_{SLD} \geq |y_n - y_g|$. Thus, $h_{SLD} \geq max\{|x_n - x_g|, |y_n - y_g|\} = h_1(n)$. This means that $h^*(n) \geq h_{SLD}(n) \geq h_1(n)$, so $h_1(n)$ is admissible.

Consistency:
Define $h_3(n) = |x_n - x_g|$, and $h_4(n) = |y_n - y_g|$. Using the triangle inequality, we can demonstrate that both $h_3(n)$ and $h_4(n)$ are consistent. We have

$$h_3(n) - h_3(n') = |x_n - x_g| - |x'_n - x_g| \leq |x_n - x'_n| \leq c(n, a, n')$$

$$h_3(n) \leq c(n, a, n') + h_3(n')$$

Similarly, $h_4(n)$ is consistent. Since $h_1(n)$ is the maximum of $h_3(n)$ and $h_4(n)$, $h_1(n)$ is also consistent.

2. Is $h_2(n)$ an admissible heuristic? If yes, prove it; otherwise, provide a counter-example.

Solution:

Consider a graph where v_s has coordinates $(0, 0)$ and v_g has coordinates $(1, 1)$. $h^*(v_s) = \sqrt{2}$ while $h_2(v_s) = 2$. Since $h_2(v_s) = 2 > h^*(v_s)$, $h_2(n)$ is not an admissible heuristic.

3. Out of $h_1(n)$, $h_2(n)$ and $h_{SLD}(n)$, which heuristic function would you choose for A* search? Justify your answer.

Solution:

$h_2(n)$ is not admissible, so we may not get an optimal solution if we use it. Meanwhile, from part (a), we have proven that $h_{SLD}(n) \geq h_1(n)$, i.e. $h_{SLD}(n)$ dominates $h_1(n)$, which means that $h_{SLD}(n)$ is closer to the true cost $h^*(v_s)$ than $h_1(n)$, implying that we will incur less search cost (on average) if we use $h_{SLD}(n)$. Hence, we pick $h_{SLD}(n)$.

C Admissibility and Consistency

1. Given that a heuristic h is such that $h(G) = 0$, where G is any goal state, prove that if h is consistent, then it must be admissible. (Hint: Think of the path the algorithm takes)

Solution:

Consistency is defined as $h(v) \leq c(v, a, u) + h(u)$, where $h(v) =$ heuristic/estimate from node v to goal node, $c(v, a, u) =$ actual cost from v to u by applying action a,

and $h(u) =$ heuristic from u to the goal node.

Let v_n be an arbitrary node and let $v_n, v_{n-1}, \ldots, v_1, G$ be the path with minimal cost to a goal state.

Let a_n be the action applied from v_n to v_{n-1}.

By using the fact that h is consistent multiple times, we can get:

$$h(v_n) \leq c(v_n, a_n, v_{n-1}) + h(v_{n-1})$$

$$h(v_{n-1}) \leq c(v_{n-1}, a_{n-1}, v_{n-2}) + h(v_{n-2})$$

$$\ldots$$

$$h(v_1) \leq \ c(v_1, a_1, G) + h(G)$$

Note that $h(G) = h^*(G) = 0$, so we can drop $h(G)$.

Combined,

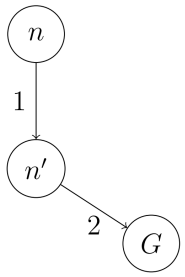
$$\begin{aligned} & h(v_n) \\ & \leq c(v_n, a_n, v_{n-1}) + h(v_{n-1}) \\ & \leq c(v_n, a_n, v_{n-1}) + c(v_{n-1}, a_{n-1}, v_{n-2}) + h(v_{n-2}) \\ & \ldots \\ & \leq c(v_n, a_n, v_{n-1}) + c(v_{n-1}, a_{n-1}, v_{n-2}) + \ldots + c(v_1, a_1, G) \end{aligned}$$

Because we chose the minimal cost path, $c(v_n, a_n, v_{n-1}) + c(v_{n-1}, a_{n-1}, v_{n-2}) + \ldots + c(v_1, a_1, G)$ equals to the true cost $h^*(n)$. Hence we have shown $h(v_n) \leq h^*(v_n)$ for any node v_n, and proven that if h is consistent, then it must also be admissible.

2. Give an example of an admissible heuristic function that is not consistent.

Solution:

Consider the graph below.



An example of an admissible heuristic function that is not consistent is as follow:

$$h(n) = 3$$

$$h(n') = 1$$

$$h(G) = 0$$

h is admissible since

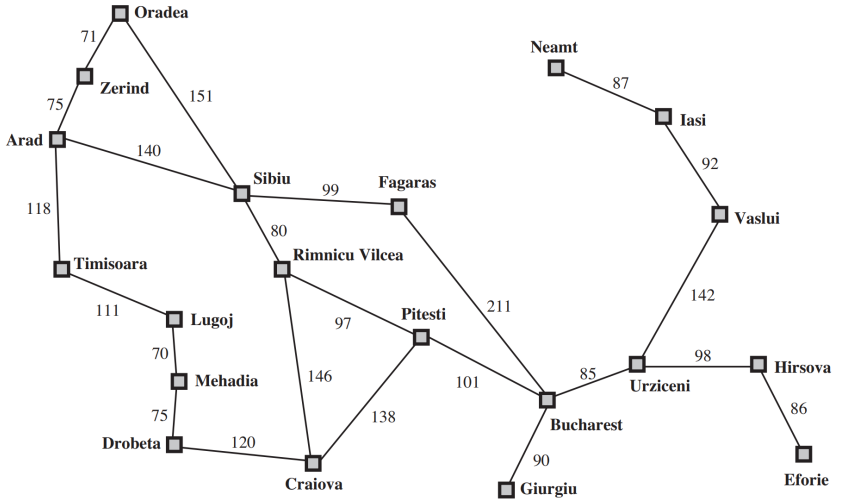
$$h(n) = 3 \leq h^*(n) = 1 + 2 = 3$$

$$h(n') = 1 \leq h^*(n') = 2$$

However, h is not consistent as $h(n) = 3 > c(n, a, n') + h(n') = 1 + 1 = 2$. Thus, admissibility doesn't imply consistency.

D Romania

Refer to the Figure 2 below. Apply the A* search algorithm to find a path from Fagaras to Craiova where $h(n) = |h_{SLD}(\text{Craiova}) - h_{SLD}(n)|$ and $h_{SLD}(n)$ is the straight-line distance from any city n to Bucharest given in Figure 2.



Values of h_{SLD} - straight-line distances to Bucharest

Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Figure 2: Graph of Romania.

1. Trace the A* search algorithm by showing the nodes and their priorities in the fringe (frontier) as each node is expanded, based on the `SEARCH` algorithm below in its first *three* iterations. Also, draw the search trees for the first three iterations.

Solution:

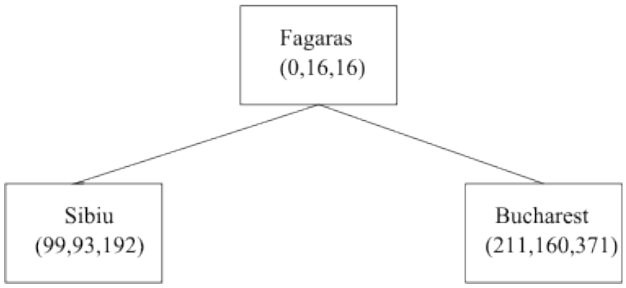
Iteration	Node(s) in Fringe
0	Fagaras (16)
1	Sibiu (192), Bucharest (371)
2	Rimnicu Vilcea (212), Fagaras (214), Bucharest (371), Arad (445), Oradea (470)
3	Fagaras (214), Craiova (325), Pitesti (336), Sibiu (352), Bucharest (371), Arad (445), Oradea (470)
4	Craiova (325), Pitesti (336), Sibiu (352), Bucharest (371), Sibiu (390), Arad (445), Oradea (470), Bucharest (569)
5	[Goal Reached]

The search trees are as follows (the 3-tuple in each node denotes the $g(n)$, $h(n)$, and $f(n)$ values respectively):

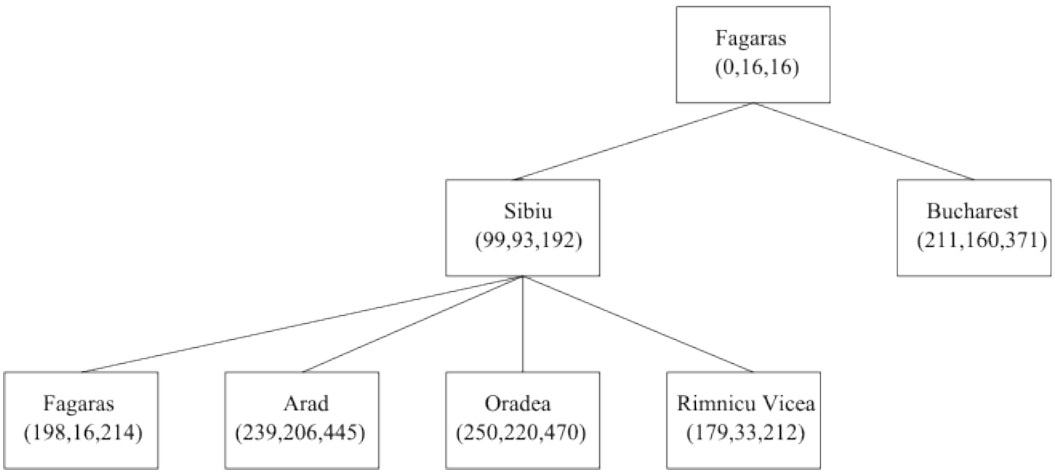
0.


Fagaras
(0,16,16)

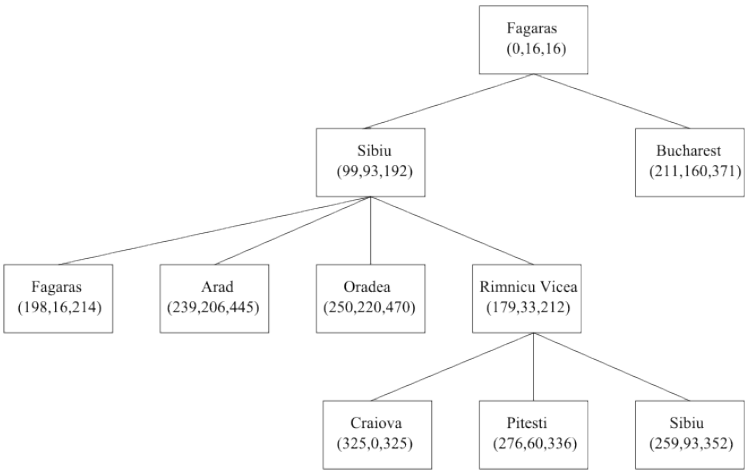
1.

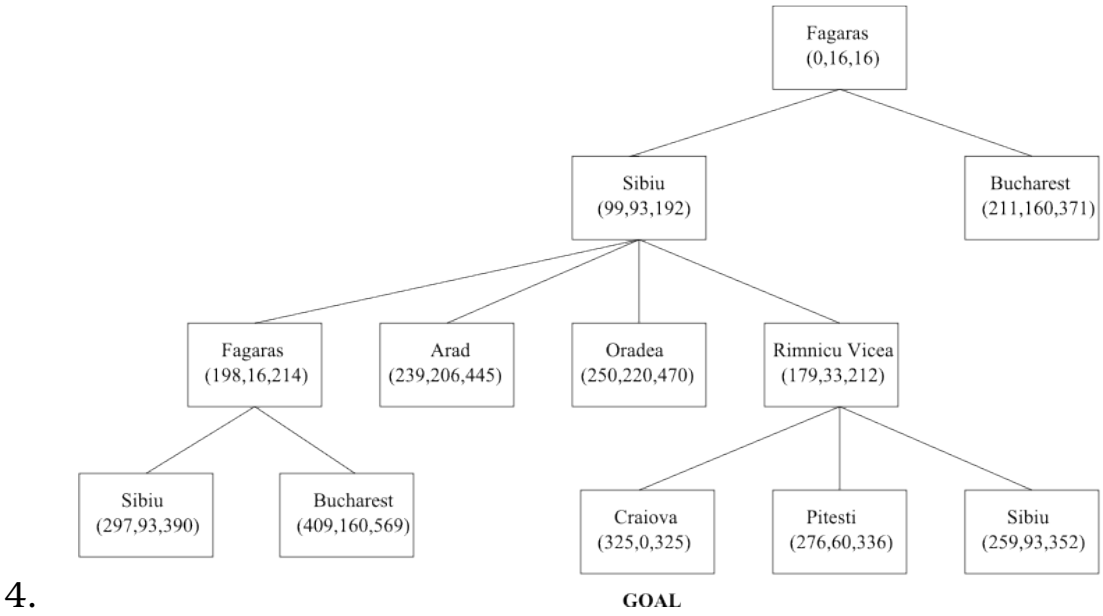

Fagaras
(0,16,16)
Sibiu
(99,93,192)
Bucharest
(211,160,371)

2.


Fagaras
(0,16,16)
Sibiu
(99,93,192)
Bucharest
(211,160,371)
Fagaras
(198,16,214)
Arad
(239,206,445)
Oradea
(250,220,470)
Rimnicu Vicea
(179,33,212)

3.


Fagaras
(0,16,16)
Sibiu
(99,93,192)
Bucharest
(211,160,371)
Fagaras
(198,16,214)
Arad
(239,206,445)
Oradea
(250,220,470)
Rimnicu Vicea
(179,33,212)
Craiova
(325,0,325)
Pitesti
(276,60,336)
Sibiu
(259,93,352)

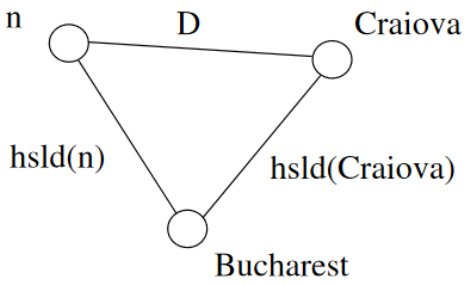


4.

2. (Optional) Prove that $h(n)$ is an admissible heuristic.

Solution:

Now consider the following triangle:



Let D be the straight-line distance, which means the shortest path possible between n and Craiova. Based on the triangle inequality,

$$D + h_{SLD}(n) \geq h_{SLD}(Craiova)$$
$$D \geq |h_{SLD}(Craiova) - h_{SLD}(n)|$$
$$Hence, D \geq h(n).$$

Let $h^*(n)$ be the actual true cost between n and Craiova. Then, $h^*(n) \geq D \geq h(n)$, and hence $h(n)$ is admissible.

Note: The triangle inequality states that the sum of the lengths of any two sides must be greater than or equal to the length of the third side.

E Admissibility and Inconsistency

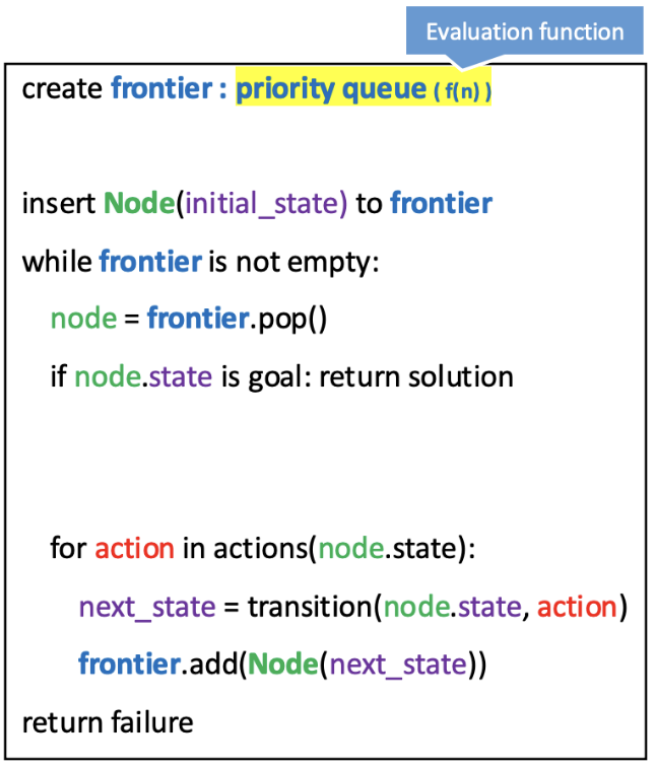


Figure 3: A* search (`SEARCH` implementation).

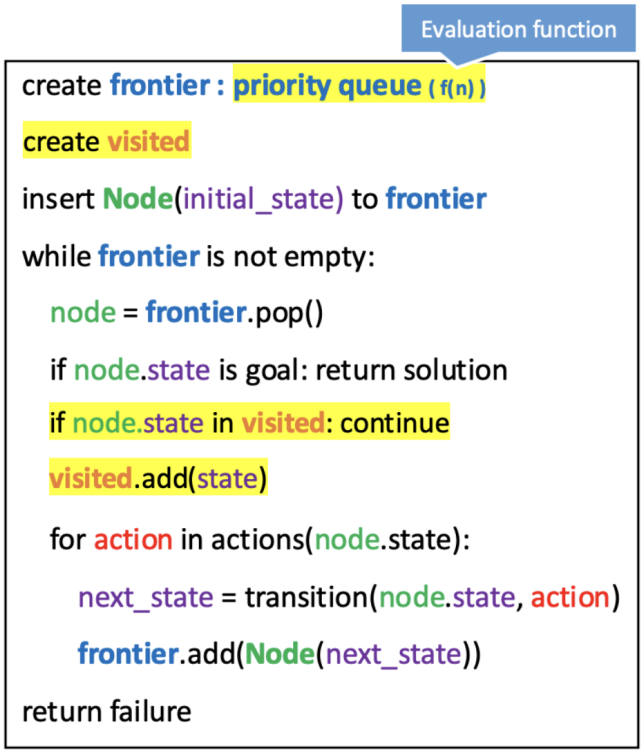


Figure 4: A* search (`SEARCH WITH VISITED MEMORY` implementation).

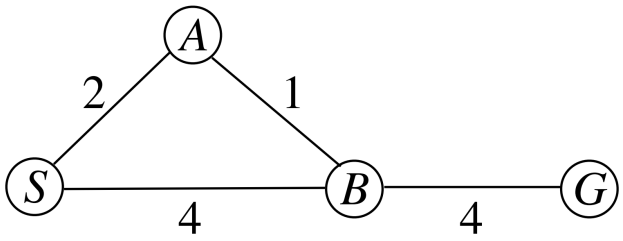


Figure 5: Graph

You have learned before that A* using search with visited memory is optimal if $h(n)$ is consistent. Does this optimality still hold if $h(n)$ is admissible but inconsistent? Use the heuristic function $h(n)$ defined as follows:

$$h(n) = \begin{cases} 7 & \text{if } n = S \\ 0 & \text{if } n = B \\ 3 & \text{if } n = A \\ 0 & \text{if } n = G \end{cases}$$

You may refer to Figure 3 and Figure 4 for A* Search vs. A* Search with visited memory implementation.

1. Using the graph in Figure 5, show that A* using `SEARCH WITH VISITED MEMORY` returns a non-optimal solution path from start node S to goal node G when using an admissible but inconsistent $h(n)$.
2. Show that search will return the optimal solution with the same heuristic.

Solution:

First let us recall that A* maintains a priority queue in increasing order of $f(n) = g(n)+h(n)$ (ties broken arbitrarily). The key difference between `SEARCH` and `SEARCH WITH VISITED MEMORY` is that under `SEARCH WITH VISITED MEMORY`, once a node gets explored, it will never be added to the priority queue again. The intuition here is that if we do not revisit nodes, we may miss out on some shorter paths that we mistakenly think are long after reaching the node as the heuristic function violates the triangle inequality. There are several possible solutions one might consider. First, let us consider $h(n)$. A* will first add $(S, 7, (null))$ to the priority queue and explore it as it is the only element in the queue, which adds $(A, 2+3, (S))$ and $(B, 4+0, (S))$ into the priority queue, resulting in

$$|(B, 4, (S)), (A, 5, (S))|$$

It will then explore B, finding A, G and S. Since B has not been visited, it will add $(A, 8)$, $(G, 8)$ and $(S, 15)$ into the priority queue resulting in

$$|(A, 5, (S)), (A, 8, (S, B)), (G, 8, (S, B)), (S, 15, (S, B))|$$

Now, it explores A, finding S and B. Since A has not been visited, it will add $(B, 3)$ and $(S, 9)$ into the priority queue resulting in

$$|(B, 3, (S, A)), (A, 8, (S, B)), (G, 8, (S, B)), (S, 11, (S, A)), (S, 15, (S, B))|$$

Next, it explores B again. Since B has been visited, it adds nothing into the priority queue resulting in

$$|(A, 8, (S, B)), (G, 8, (S, B)), (S, 11, (S, A)), (S, 15, (S, B))|$$

Then, it explores A again. Since A has been visited, it adds nothing into the priority queue resulting in

$$|(G, 8, (S, B)), (S, 11, (S, A)), (S, 15, (S, B))|$$

Finally, it explores G, ending up with a suboptimal solution of (S, B, G).

With search, we add nodes that we have already visited back into the priority queue. At the start, we add S into the priority queue:

$$|(S, 7, (null))|$$

Exploring it, we find A and B, adding them into the priority queue, resulting in

$$|(B, 4, (S)), (A, 5, (S))|$$

It still explores B first, finding S, A and G, resulting in

$$|(A, 5, (S)), (A, 8, (S, B)), (G, 8, (S, B)), (S, 15, (S, B))|$$

Then, it explores A, finding S and B, resulting in

$$|(B, 3, (S, A)), (A, 8, (S, B)), (G, 8, (S, B)), (S, 11, (S, A)), (S, 15, (S, B))|$$

This time, it explores B first (from the path from A), finding S, A and G, resulting in

$$|(G, 7, (S, A, B)), (A, 7, (S, A, B)), (A, 8, (S, B)), (G, 8, (S, B)), ..., (S, 15, (S, B))|$$

Finally, G is explored and we get the optimal path (S, A, B, G).

F (Extra & Optional) Negativity

1. Would A* work with negative edge weights? Assume no negative cycles. If yes, prove it; otherwise, provide a counterexample. What if there are negative cycles?

Solution:

As shown through the proof of the optimality of A* search when used with an admissible heuristic, negative weights do not affect the optimality of A*. Here, the same proof applies.

The proof for `SEARCH` is reproduced here:

Suppose A* is not optimal and it will return the suboptimal goal G_2. For this to happen, G_2 must been generated at some point and is in the frontier. We have

the following equations:

$$f(G) = g(G) + h(G) = g(G) + 0 \quad (1)$$
$$f(G_2) = g(G_2) + h(G_2) = g(G_2) + 0 \quad (2)$$
$$g(G_2) > g(G) \text{ since } G_2 \text{ is suboptimal by our assumptions} \quad (3)$$
$$f(G_2) > f(G) \quad (4)$$

We know that G cannot be in the frontier before G_2 since it would otherwise be expanded before G_2.

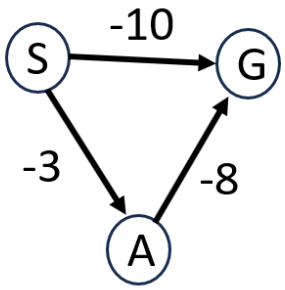
Let n be an unexpanded node in the frontier such that n is on a shortest path to an optimal goal G. The first n could be the initial node. Then:

$$h(n) \leq h^*(n) \text{ due to admissibility of } h \quad (5)$$
$$f(n) = g(n) + h(n) \leq g(n) + h^*(n) = g(G) = f(G) \quad (6)$$
$$f(n) \leq f(G) < f(G_2) \text{ from (4) so } G_2 \text{ will never be expanded} \quad (7)$$

This means that the nodes on the shortest path to G will all be expanded before G_2, so G_2 will never be expanded before G.

Notice that this proof does not assume that the weights are positive. Therefore, the proof still holds.

One question that naturally arises would be: why would A* work for negative weights if Dijkstra does not? Consider the following graph:



If we naively assume that $h(n) = 0$ for all nodes and apply Dijkstra, it won't work since once we expand S, we will get G as the next node on the frontier since -10 < -3. However, notice that $h(n) = 0$ is not admissible. In fact, for h to be admissible, h needs to satisfy the following:

$$h(G) \leq 0 \quad (8)$$
$$h(A) \leq -8 \quad (9)$$
$$h(S) \leq -11 \quad (10)$$

If we assume $h(G) = 0$, $h(A) = -8$, and $h(S) = -11$, and we apply A*, we will get the correct optimal path. In other words, A* works because if h is admissible, then h will somehow "make up" for the negative weights. Regular Dijkstra does not have this extra heuristic to "make up" for the negative weights.

For `SEARCH WITH VISITED MEMORY` we first prove that for a path from start to goal, the f-scores are increasing, that is if node m comes after node n, $f(n) \leq f(n')$.

1. Due to consistency, $h(n) \leq c(n, a, n') + h(n')$
2. $f(n) = g(n) + h(n) \leq g(n) + c(n, a, n') + h(n') = g(n') + h(n') = f(n')$

Then prove by induction that A* never expands a node with a lower f-score than a node that has already been expanded.

1. **Inductive Statement**: For any node n expanded by A*, the f-score of n is higher than or equal to the f-scores of all nodes that have been previously expanded.
2. **Base Case**: Consider the initial node n_0. A* starts by expanding the initial node, and since there are no nodes that have been previously expanded, the base case is trivially true.
3. **Inductive Step**: Assume that the inductive statement holds for any node n_k that has been expanded, where k is any positive integer. Now, let's prove that the statement holds for the node n_{k+1} that A* expands next.

   Let $f(n_{k+1})$ be the f-score of n_{k+1}. By the inductive assumption, the f-score of n_k (the previously expanded node) is higher than or equal to the f-scores of all nodes that have been previously expanded.

   Now, there are two possibilities:

   1. Node n_{k+1} was in the frontier when n_k was expanded. Since n_k was expanded before n_{k+1}, $f(n_{k+1})$ is higher than equal to the f-score of n_k. In this case, the inductive statement holds for n_{k+1}.

   2. Node n_{k+1} was not in the frontier when n_k was expanded, but was added during the expansion of n_k. This means that n_{k+1} is the successor of n_k and hence, the path cost from n_k to n_{k+1} must be greater than equal to zero. And hence, $f(n_{k+1})$ must be higher than equal to the f-score of n_k.

This is because a consistent heuristic is used. No assumptions were made about the nature of the weights (whether they were negative). Consistency will cause nodes to be expanded in the increasing order of f-score, and thus the proof still holds.

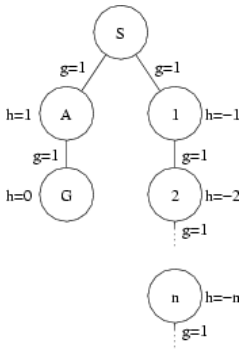
Note that A* will not work with negative cycles because we cannot find an admissible heuristic since the cost can be made infinitely low by going around in a cycle.

2. We have seen how A* performs with negative *edge weights*. A* with negative *heuristics* is different: negative heuristics are heuristic functions that can return negative values for some nodes. Can A* work with negative heuristics? If yes, prove it; otherwise, provide a counterexample.

Solution:

The requirement for admissibility is that the heuristic function must not overestimate the cost to reach the goal. While a negative heuristic may be likely to result in more nodes being expanded, it is not necessarily inadmissible. Indeed, a heuristic that only returns negative values is admissible for any graph with nonnegative edge weights. This is because the heuristic will always under-

estimate the cost to reach the goal. However, admissibility does not guarantee completeness. Consider the following graph:



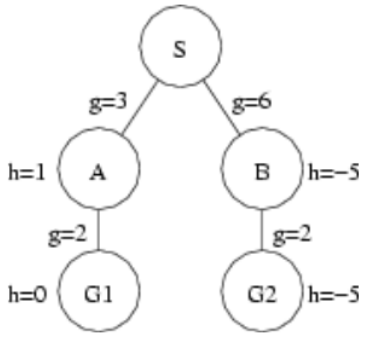
Where $h(n)$ is defined as follows:

$$h(n) = \begin{cases} 0 & \text{if } n = S \text{ or } G \\ 1 & \text{if } n = A \\ -n & \text{otherwise} \end{cases}$$

If we apply A* with this (admissible) heuristic, the search does not terminate for infinite n. This is because the priority for A is set to 2, while for all of the numeric nodes, the priority is set to 0 due to the negative heuristic. This means that the search will continue to expand nodes with negative heuristic values indefinitely, never expanding A and never reaching the goal. For reference, see the following table of the search order:

Iteration	**Node(s) in Fringe**
0	S (0)
1	1 (0), A (2)
2	2 (0), A (2)
...	...
n	n (0), A (2)

We see that A* is not guaranteed to terminate on graphs with negative heuristics. However, is it optimal on finite graphs? Consider the following graph with two goal states G_1 and G_2:



Where h(n) is defined as follows:

$$h(n) = \begin{cases} 0 & \text{if } n = S \text{ or } G_1 \\ 1 & \text{if } n = A \\ -5 & \text{otherwise} \end{cases}$$

If we apply A* with this heuristic, the search will expand B and then G_2. However, the optimal path is (S, A, G_1) with a lower cost of 5 compared to the path (S, B, G_2) with a cost of 8. For reference, see the following table of the search order:

Iteration	**Node(s) in Fringe**
0	S (0)
1	B (1), A (4)
2	G2 (3), A (4)
3	G2 is expanded [search terminates]

We discussed in lecture that A* is optimal with an admissible heuristic. However, the heuristic in this case is admissible, but the search is not optimal. This is because there is an additional assumption in the proof for optimality: that $h(g) = 0$. This assumption is required, or else the priority $(f(n))$ of the goal nodes may not necessarily correspond to the actual cost of the path. For instance, if $h(g_2)$ in the above example were 0, then the search would have continued along $A \to G_1$ and terminated with the optimal path instead of expanding G_2.

Therefore, A* is not guaranteed to be optimal or complete on graphs with negative heuristics.

The graphs for the above solution were used from the University of Oklahoma.