Search heuristics: Heuristics can be difficult to come up with. Think about how to find a good job after graduation? Is a state of “having taken CSE 571” better than a state of “having not taken CSE 571”?

A heuristic is useful since it gives you an idea about how close you are to the goal, in other words, it gives a “forward” view.

Properties of greedy search (with tree search): A greedy search with a bad heuristic may need to expand every node so its computational complexity is $O(b^m)$. The same argument holds for space complexity. It may be stuck in loops since the heuristic can misguide you to loop forever. Although greedy search has terrible worst-case performance, the idea itself is promising.

Combining UCS and Greedy: The example shows you the problem with UCS and greedy search. UCS makes you expand nodes in every direction. A lot of nodes may not need to be expanded. A greedy search however can lead to you to a sub-optimal path. One of them looks only at what has happened (backward), and the other looks only at what will occur (forward). A* (best search) combines them.

(Node that all methods are assumed to report a solution when a goal node is expanded; not when a goal node is added to the fringe.)

Admissible Heuristic: What do you want as a heuristic? Ideally, you want the node that you expand to be the node that has the least cost to the goal so that you know that when you expand the goal node, you have found the optimal solution. This means that going through any other options that remain on the fringe will have a costlier path. This only holds if the actual path cost is costlier than the prediction!

Actual cost: Actual cost is nice to use as a heuristic but that often implies that you have already solved the problem. In reality, you want to remain close to the actual cost as much as you can. This also informs us about how to design heuristics.

Graph search: If a state has already been expanded in BFS, do we need to expand it again? No, since we must have a state that is shallower. If a state has already been expanded in UCS, do we need to expand it again? No, since the other path must be costlier. We can simply drop those nodes. In the implementation, this is often achieved by keeping a closed set.

Note that this is different from the situation when a state is already in the fringe and another node that you are expanding adds a node for the same state to the fringe.
**A* graph search gone wrong**: In A*, however, this may lead to issues. In this example, C will first get expanded for the path S->B->C before the same C is added when expanding A for the optimal path S->A->C.

**Consistency**: This is why we need a consistent heuristic. A consistent heuristic ensures that the f value always increases (or does not decrease to be more precise) along the paths. This property ensures that graph search is optimal due to the following reasoning:

To make A* not optimal, we must have a node, denoted by C, that is on the optimal path, which is however expanded for a suboptimal path before the optimal path to C enters the fringe (this will essentially remove the optimal path to C from the candidate list since it will not be considered again). Otherwise, the optimal path to C will still be considered and the admissible heuristic ensures that the optimal path will be found.

For the suboptimal path that is expanded at C, its f value must be greater than the optimal path that is expanded at C. At the same time, due to the non-decreasing property of a consistent heuristic, the f value at a node (denoted as A) that is the immediate predecessor on the optimal path to C must have a f value that is no greater than the f value at C via the optimal path. As a result, **before** C is expanded for the suboptimal path, A will have to be expanded in which case the optimal path to C will enter the fringe.

Notice that in the above explanation, we are using the search node C and a path to C interchangeably since a search node is assumed to contain the path information as well.