A* & Heuristic Search

CMPUT 355: Games, Puzzles, and Algorithms

Lecture Outline

1. Logistics & Recap
2. Map puzzle
3. A* search
4. Sliding tile heuristics

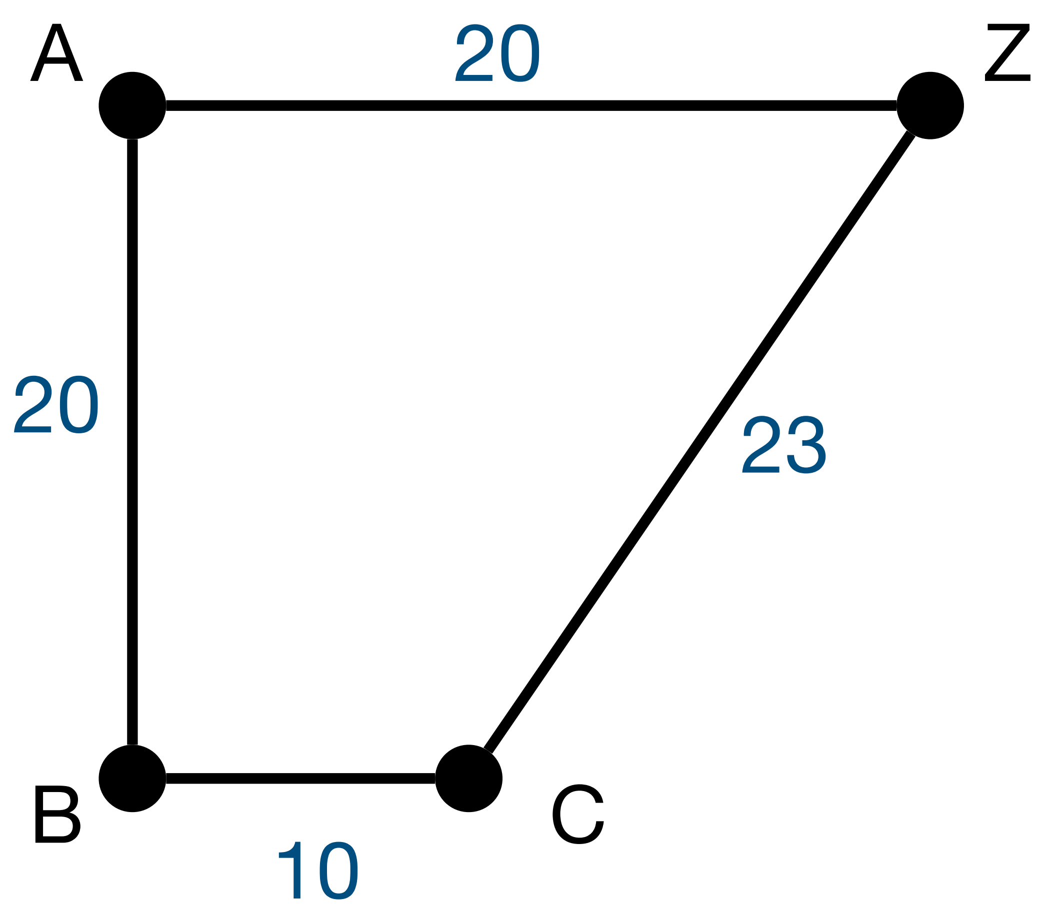
Logistics

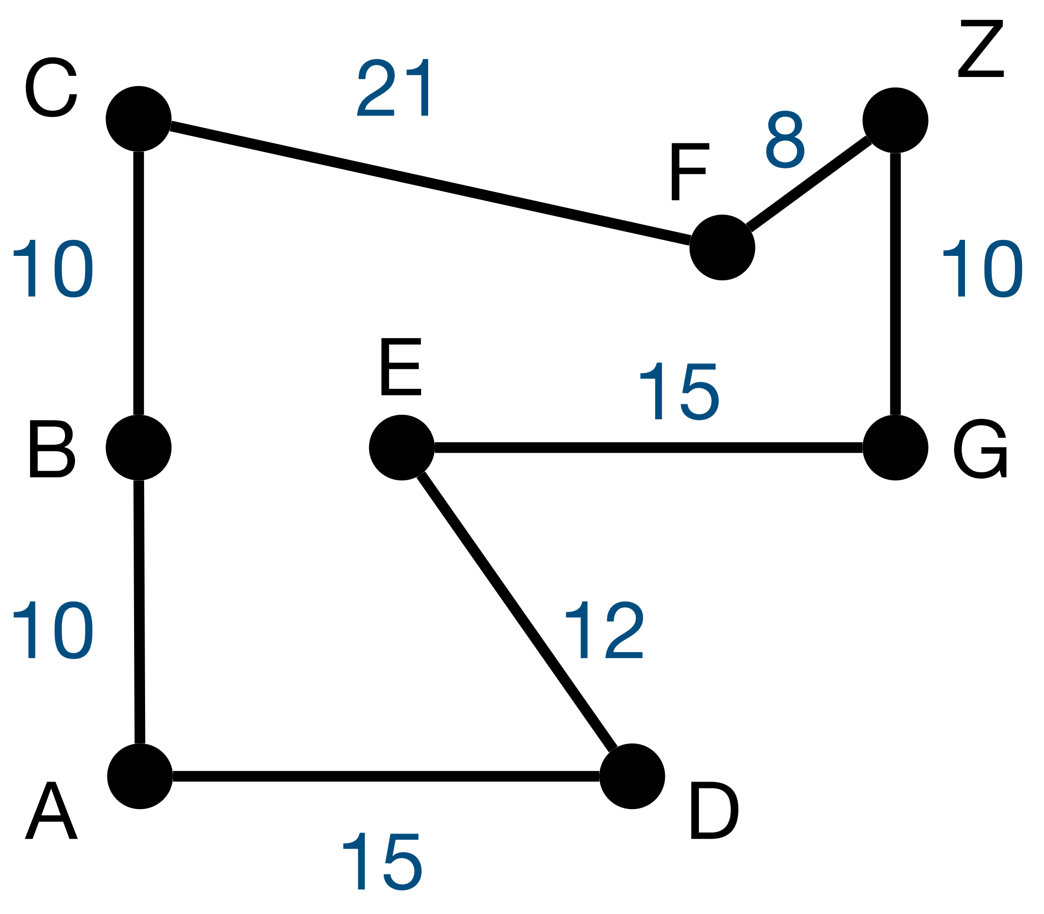
- **Practice quiz questions #2:**
  - Will be posted on Friday (Jan 30)
  - Answers will be posted following Tuesday (Feb 3)
- **Quiz 2:** Next Friday, **Feb 6**
  - In-class, full 50 minutes
  - **No need to email** if you have to miss it;
    up to 3 missed quizzes replaced by final exam **automatically**
  - Questions will be very similar to practice questions

Recap: Breadth-First Search

- Exhaustively explores **every possible position**
  - In order of number of moves from start position (shortest paths first)
  - Good news: Will eventually find a solution (if it exists)
  - Bad news: Might take until heat death of universe
- Example: Sliding tile puzzle
  - BFS solves 3×3 puzzle basically immediately
  - We estimate that it would take about a year to solve 4×4 (worst-case)
  - But people solve puzzles of this size all the time!
- People explore "promising" neighbours before unpromising ones

Roadmap Example





- Want to get from A to Z
- As usual, edge between neighbours
- **New:** label each edge with a **cost**
- **Goal:** Path from A to Z with **least cost**
- **Question:** Is BFS guaranteed to return the **least-cost** path?

A* Pseudocode

```
# item, priority
fringe = PQ()
fringe.add(start, 0)

# Preceding location
parent = {}
parent[start] = None

# Cost so far
cost = {}
cost[start] = 0

# Have processed
done = {}
```

```
while not fringe.empty():
  current = fringe.remove() # min priority
  done.add(current)
  if current == target: break
  for next in nbrs(current):
    if next not in done:
      new_cost = cost[current] + wt(current, next)
        if next not in cost or new_cost < cost[next]:
          cost[next] = new_cost
          priority = new_cost + heuristic(target, next)
          fringe.add(next, priority)
          parent[next] = current
```

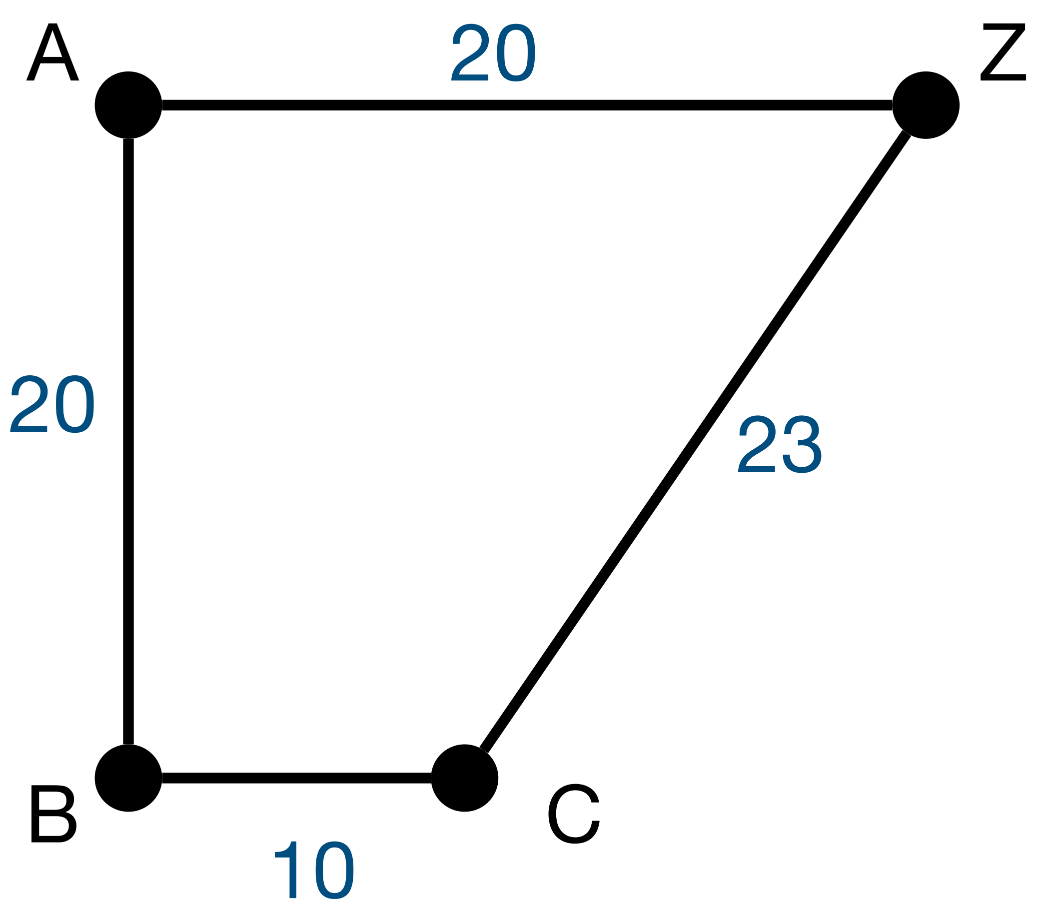
A* Pseudocode

```
while not fringe.empty():
  current = fringe.remove() # min priority
  done.add(current)
  if current == target: break
  for next in nbrs(current):
    if next not in done:
      new_cost = cost[current] + wt(current, next)
        if next not in cost or new_cost < cost[next]:
          cost[next] = new_cost
          priority = new_cost + heuristic(target, next)
          fringe.add(next, priority)
          parent[next] = current
```

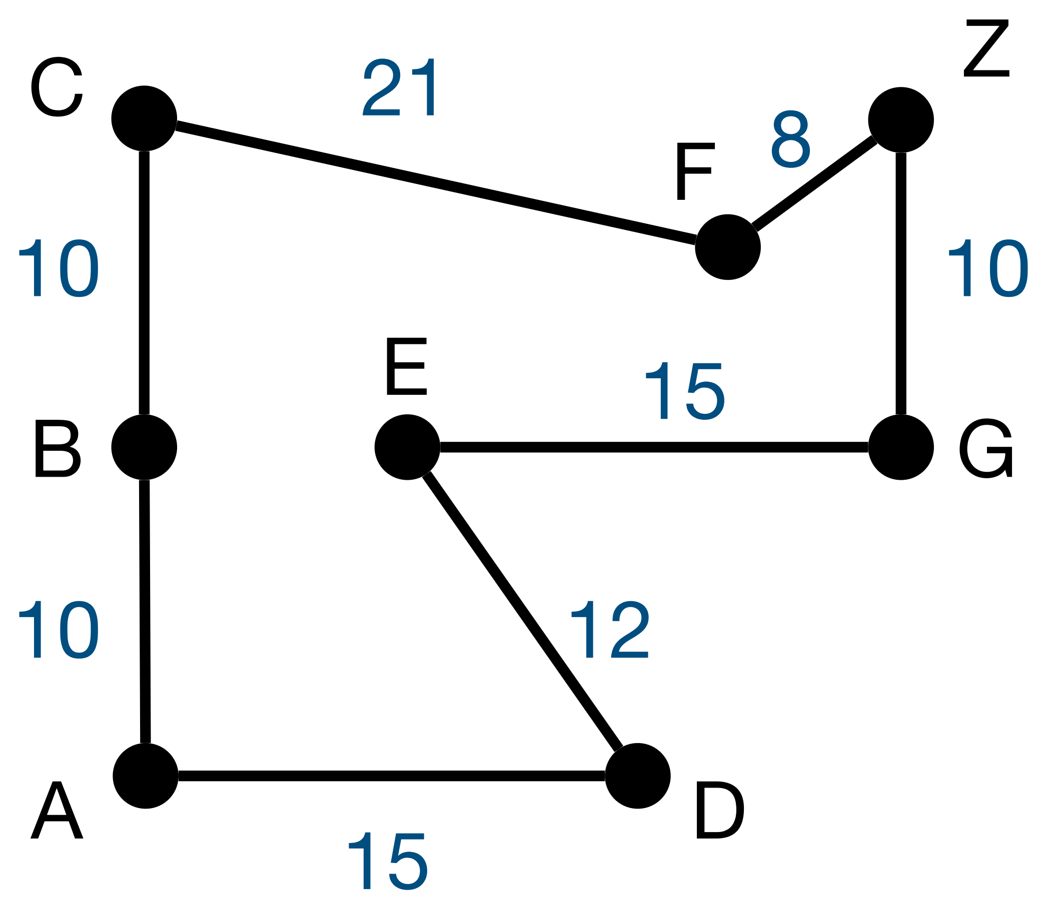
Questions:

1. Why would we need to update an existing cost?
2. What can go wrong if the heuristic is an over-estimate?
3. Is A* guaranteed to return the **least-cost** solution? (**why or why not?**)

Euclidean Heuristic



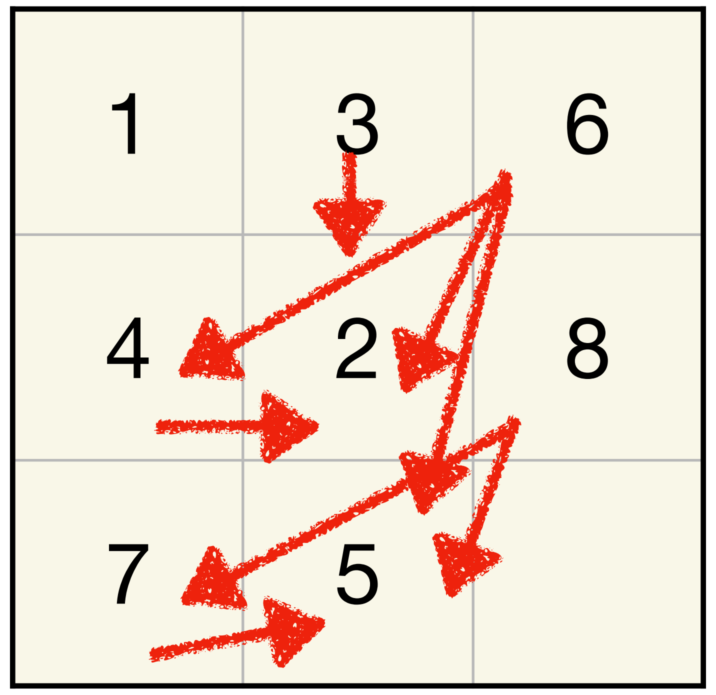
Node	A	B	C	Z
ETD	28	20	22	0



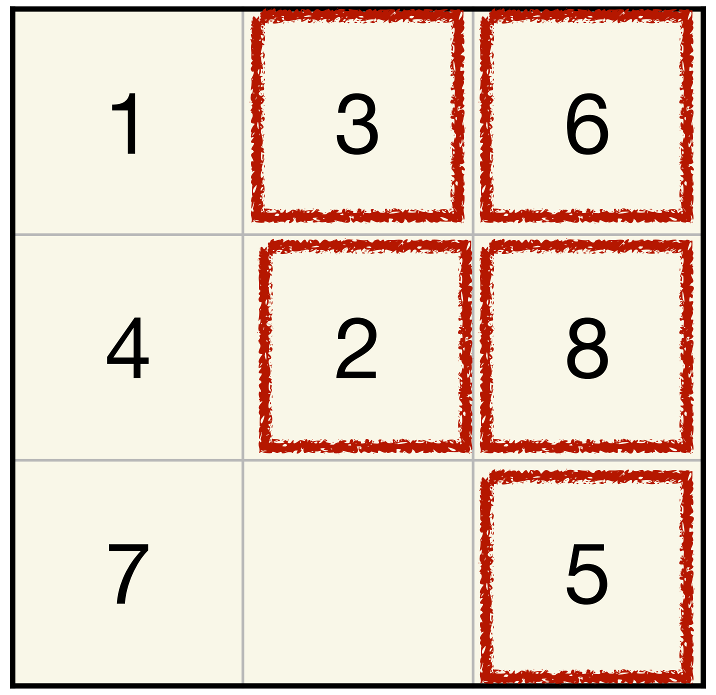
Node	A	B	C	D	E	F	G	Z
ETD	28	26	24	22	18	7	10	0

- A natural heuristic for the roadmap example is **Euclidean distance** ("as the crow flies")
- Guaranteed to be a **lower bound** on remaining cost (**why?**)
- **Demo:** `stile/astar.py` for the above problems (`wg_pq2` and `wg_pq` respectively)

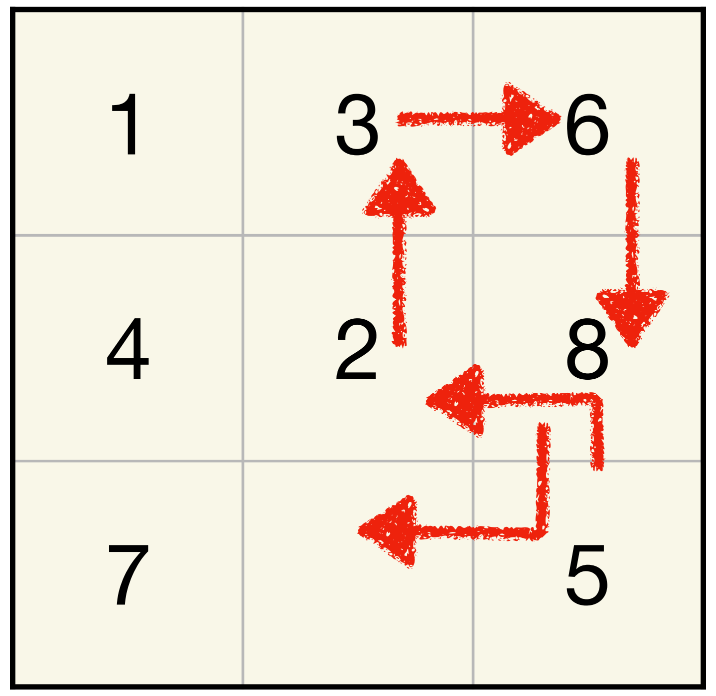
Sliding Tiles Heuristics



1. **Inversions:** Number of pairs out of order
   - **8 inversions:** {3,2}, {6,4}, {6,2}, {6,5}, {4,2}, {8,7}, {8,5}, {7,5}



2. **Misplacements:** Number of tiles in the wrong position
   - **5 misplacements:** 3, 6, 2, 8, 5
   - Does not include the blank square's placement (**why?**)



3. **Taxicab:** Distance a tile must travel to get to correct position
   - 1 + 1 + 1 + 2 + 2 = 7

Admissible Heuristics

Definition: A heuristic $h(n)$ is **admissible** if $h(n) \leq \text{cost}(n, \text{target})$ for all nodes n.
I.e., it must always be an underestimate of the remaining cost to the target.

1. **Inversions:** Number of pairs out of order
2. **Misplacements:** Number of tiles in the wrong position
3. **Taxicab:** Distance a tile must travel to get to correct position

Questions:

1. Which of the above heuristics are admissible? Why?
2. Is there a way to make the inadmissible heuristic admissible?
3. Which heuristic is likely to be most **accurate**? Why?

Combining Heuristics

- Consider two admissible heuristics, h_1 and h_2, and some node n
  - If $h_2(n) > h_1(n)$, then $h_2(n)$ **is the more accurate estimate** (**why?**)
- Some heuristics might be most accurate on different nodes
- If h_1, h_2 are both admissible, then:

1. $h_3(n) = \max\{h_1(n), h_2(n)\}$ is admissible, *and*
2. h_3 is at least as accurate as both of them

Demo: A* versus BFS for 4×4

- A solvable but hard instance:

1	2	3	4
15	14	13	12
11	10	9	8
7	5	6	

- A* solves in under 1s:

```
% time python3 15star.py -p 1  2  3  4 15 14 13 12 11 10 9 8 7 5 6
real     0m0.461s
```

- Legend says BFS is still searching to this day:

```
% echo 4 4 1 2 3 4 15 14 13 12 11 10 9 8 7 5 6 0 | time python3 stp_search2.py
...
12318701 iterations, level 23 has 10783780 nodes
23102481 iterations, level 24 has 19826318 nodes
...
```

- **Question:** What would happen if we gave 15star.py an **unsolveable** position?

Summary

- **Heuristic:** an estimate of the remaining "distance" (cost) from a node to the target
  - Not always accurate!
- **Admissible heuristic:** guaranteed to be a lower bound on remaining cost
  - i.e., a lower bound on how "bad" a given node is
- **A*** explores nodes in order of their **estimated total distance:**
  - (distance from the source to n) + (**estimated** distance from n to target)
  - First distance is real, second distance is computed by the heuristic
- **Sliding tile specific heuristics:**
  - Misplaced tiles
  - Taxicab distance
- These heuristics are good enough for A* to find solution to a **solvable** 4×4 position in under 1s
  - But **unsolvable** instances are still intractable