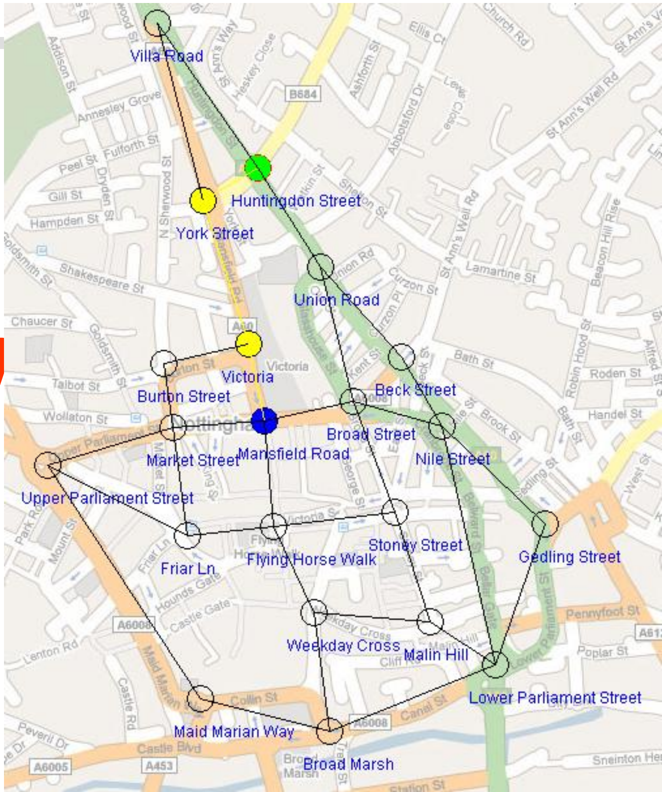
# Introduction to Artificial Intelligence (G51IAI)

Dr Rong Qu

## Heuristic Search



# Blind Search vs. Heuristic Searches

- Blind search
  - Randomly choose where to search in the search tree
  - When problems get large, not practical any more
- Heuristic search
  - Explore the node which is more likely to lead to the goal state
  - Quite often, using knowledge

# Heuristic Searches - Characteristics

- Heuristic searches work by deciding which is the next best node to expand
  - Has some domain knowledge
  - Use a function to tell us how close the node is to the goal state
- Usually more efficient than blind searches
- Sometimes called an informed search
- There is no guarantee that it is the best node

# Heuristic Searches - Characteristics

- Heuristic searches estimate the cost to the goal from its current position. It is usual to denote the heuristic evaluation function by $h(n)$
- Compare this with something like Uniform Cost Search which chooses the lowest code node thus far ( $g(n)$ )

# Heuristic Searches - Characteristics

- Heuristic searches vs. Uniform Cost Search
  - Uniform cost search
    - expand the path with the lowest path cost
    - chooses the lowest cost node thus far
  - Heuristic search
    - estimate how close the solution is to the goal
    - not how cheap the solution is *thus far*

# Heuristic Searches - Characteristics

- Heuristic searches vs. Uniform Cost Search
  - Heuristic searches evaluation function
    - *h*(n): how close is the current node to the solution
  - Uniform Cost Search path cost function
    - *g*(n): the cost of the path thus far

# Heuristic Searches - Definition

- Heuristics are "rules of thumb", educated guesses, intuitive judgments or simply common sense.

- A heuristic method is particularly used to rapidly come to a solution that is hoped to be close to the best possible answer, or 'optimal solution'.

*- Wikipedia*

# Heuristic Searches - methods

- Tree searches (G51IAI)
  - A way to reduce the search effort by pushing search in good directions
  - Not losing completeness

- Search algorithms
  - Not complete
  - Find good solutions quickly
  - Genetic Algorithms, Tabu Search, Ant Algorithms

# Heuristic Searches - Implementation 1

- Implementation is achieved by sorting the nodes based on the evaluation function: $h(n)$
  - Search is based on the order of the nodes

# Heuristic Searches - Implementation 2

Function BEST-FIRST-SEARCH(problem, EVAL-FN) returns a solution sequence

Inputs: a problem

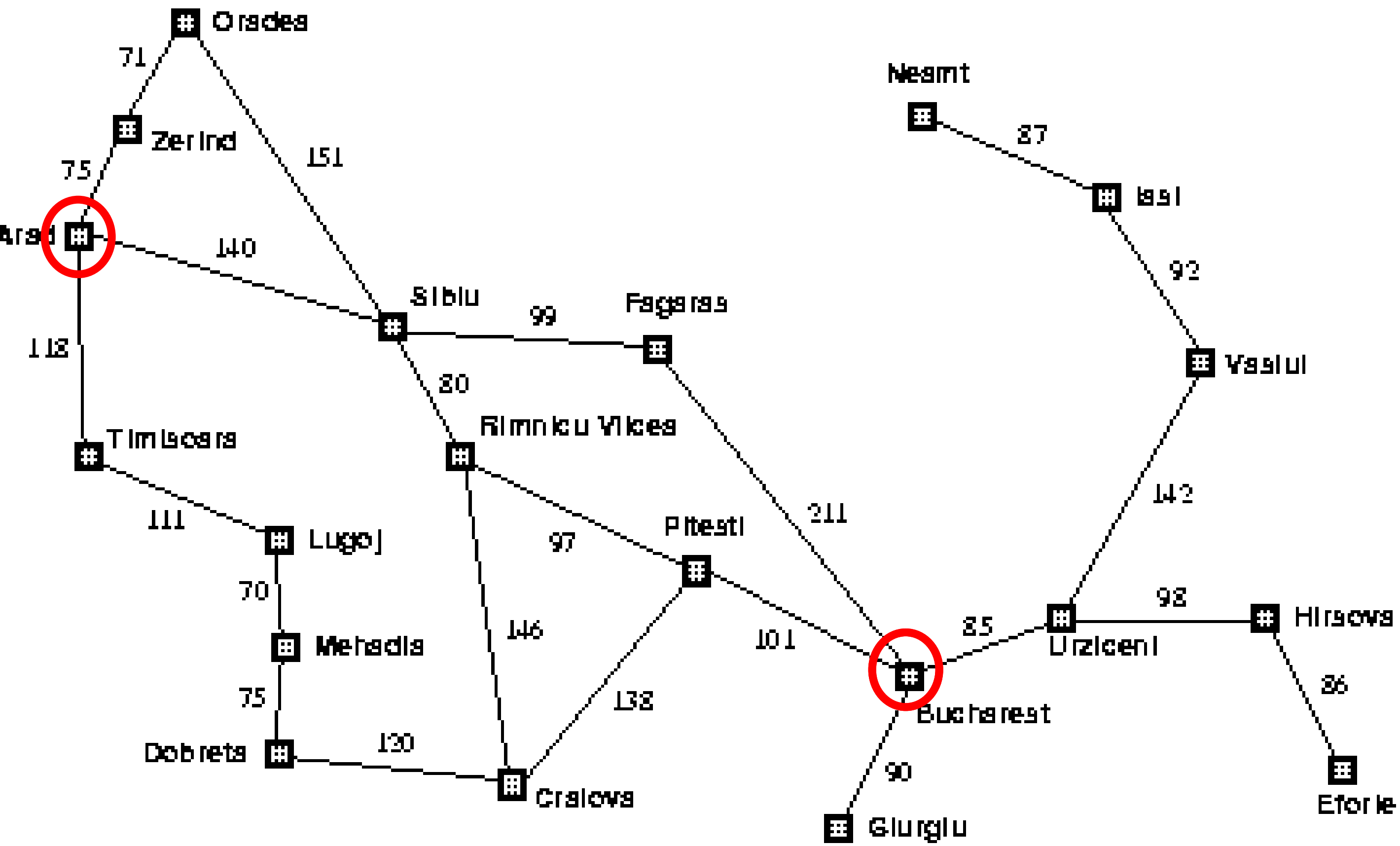
Eval-Fn: an evaluation function

Queuing-Fn: a function that orders nodes by EVAL-FN

Return GENERAL-SEARCH (problem, Queuing-Fn)

# Heuristic Searches – Example

Go to the city which is nearest to the goal city




Straight–line distance to Bucharest

| | |
|---|---|
| **Arad** | 366 |
| **Bucharest** | 0 |
| **Craiova** | 160 |
| **Dobreta** | 242 |
| **Eforie** | 161 |
| **Fagaras** | 178 |
| **Giurgiu** | 77 |
| **Hirsova** | 151 |
| **Iasi** | 226 |
| **Lugoj** | 244 |
| **Mehadia** | 241 |
| **Neamt** | 234 |
| **Oradea** | 380 |
| **Pitesti** | 98 |
| **Rimnicu Vilcea** | 193 |
| **Sibiu** | 253 |
| **Timisoara** | 329 |
| **Urziceni** | 80 |
| **Vaslui** | 199 |
| **Zerind** | 374 |

**$H_{sld}$(*n*) = straight line distance between *n* and the goal location**

# Heuristic Searches - Greedy Search

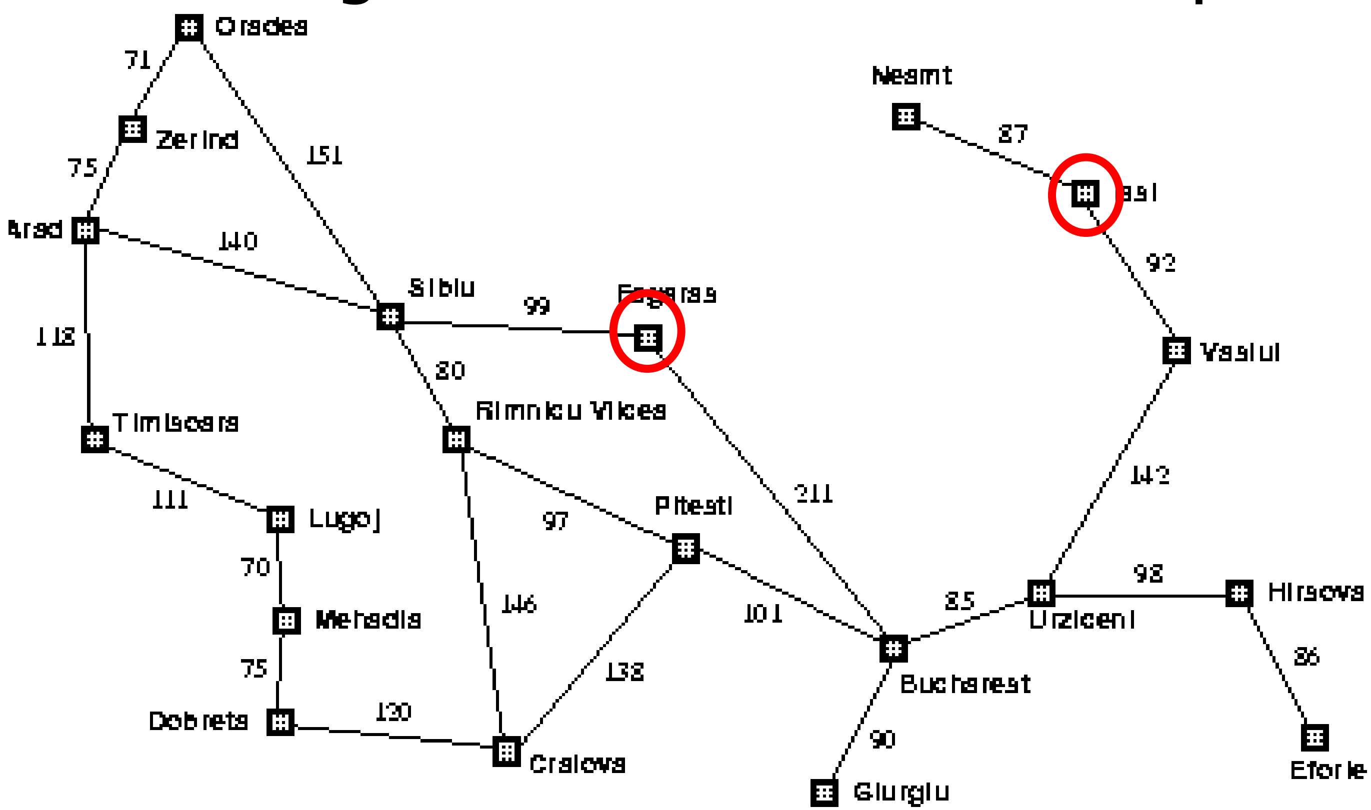
- So named as it takes the biggest "bite" it can out of the problem.
- That is, it seeks to minimise the estimated cost to the goal by expanding the node estimated to be closest to the goal state

Function GREEDY-SEARCH(*problem*) returns a solution of failure

  Return BEST-FIRST-SEARCH(*problem,h*)

# Heuristic Searches - Greedy Search

- It is only concerned with short term aims
- It is possible to get stuck in an infinite loop

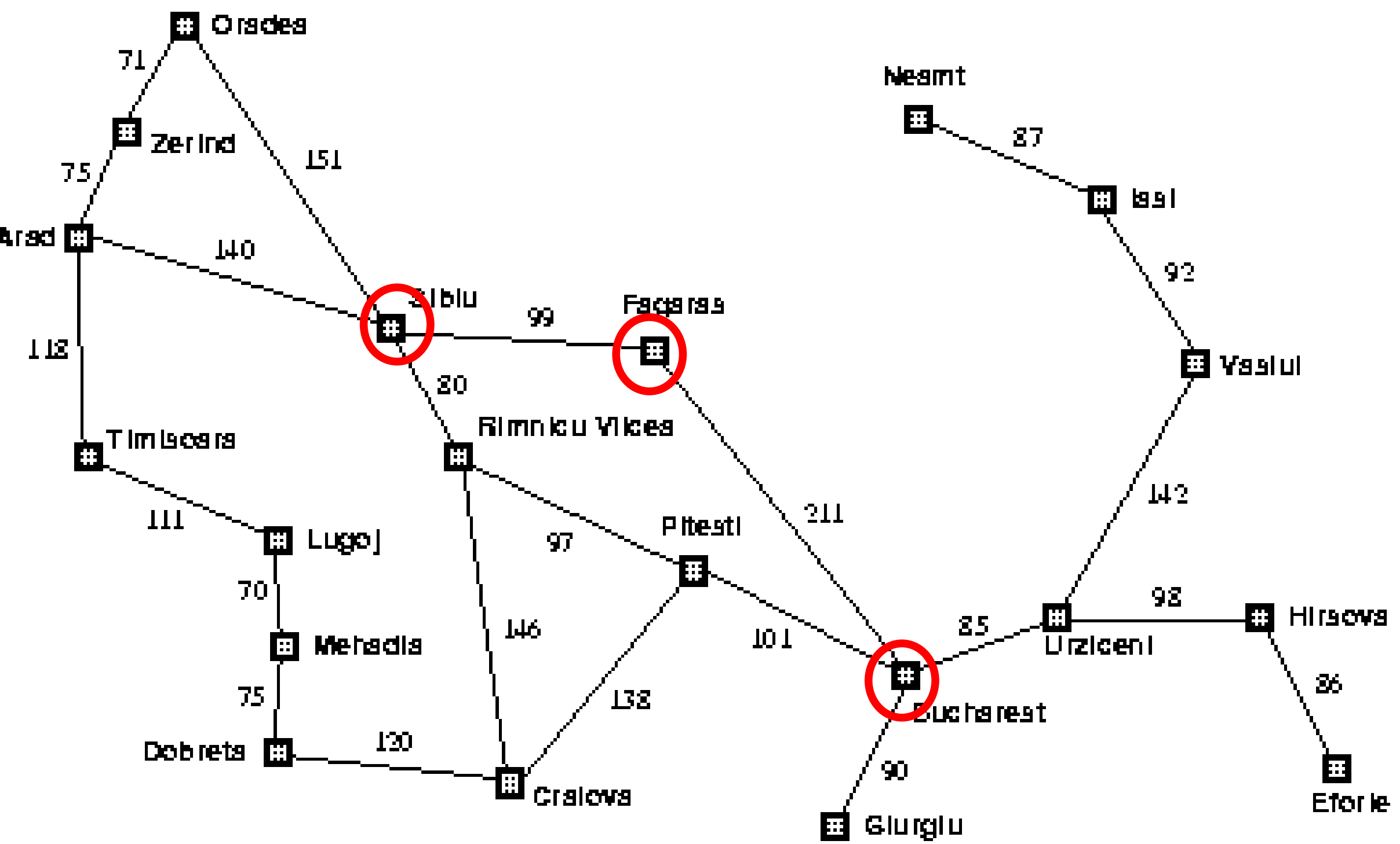


# Heuristic Searches - Greedy Search

- It is not optimal
- It is not complete

Time and space complexity is $O(b^m)$; where *m* is the depth of the search tree

# Greedy Search




Straight–line distance to Bucharest

| | |
|---|---|
| Arad | 366 |
| Bucharest | 0 |
| Craiova | 160 |
| Dobreta | 242 |
| Eforie | 161 |
| Fagaras | 178 |
| Giurgiu | 77 |
| Hirsova | 151 |
| Iasi | 226 |
| Lugoj | 244 |
| Mehadia | 241 |
| Neamt | 234 |
| Oradea | 380 |
| Pitesti | 98 |
| Rimnicu Vilcea | 193 |
| Sibiu | 253 |
| Timisoara | 329 |
| Urziceni | 80 |
| Vaslui | 199 |
| Zerind | 374 |

Performed well, but not optimal

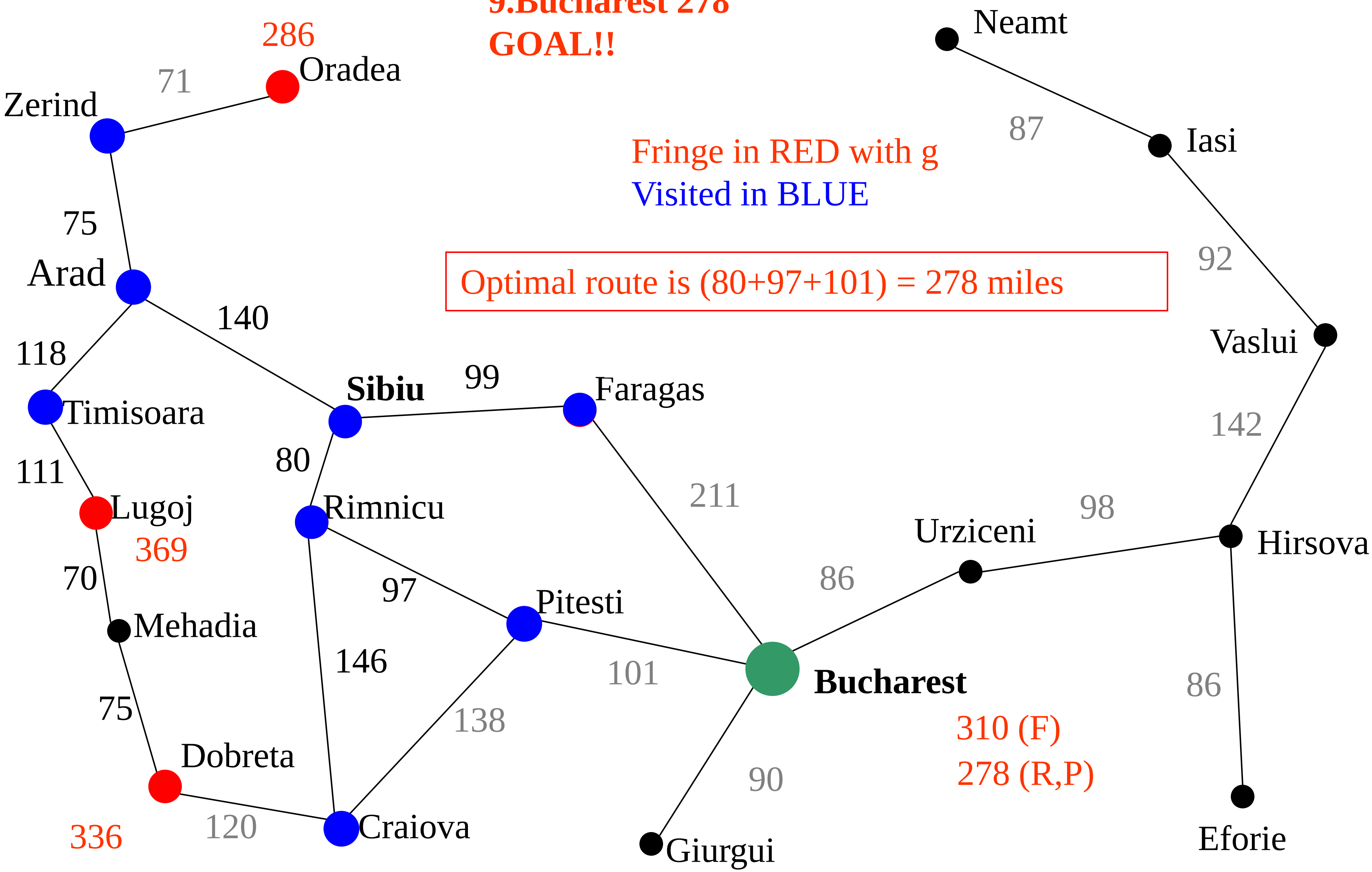
# UCS

Nodes Expanded

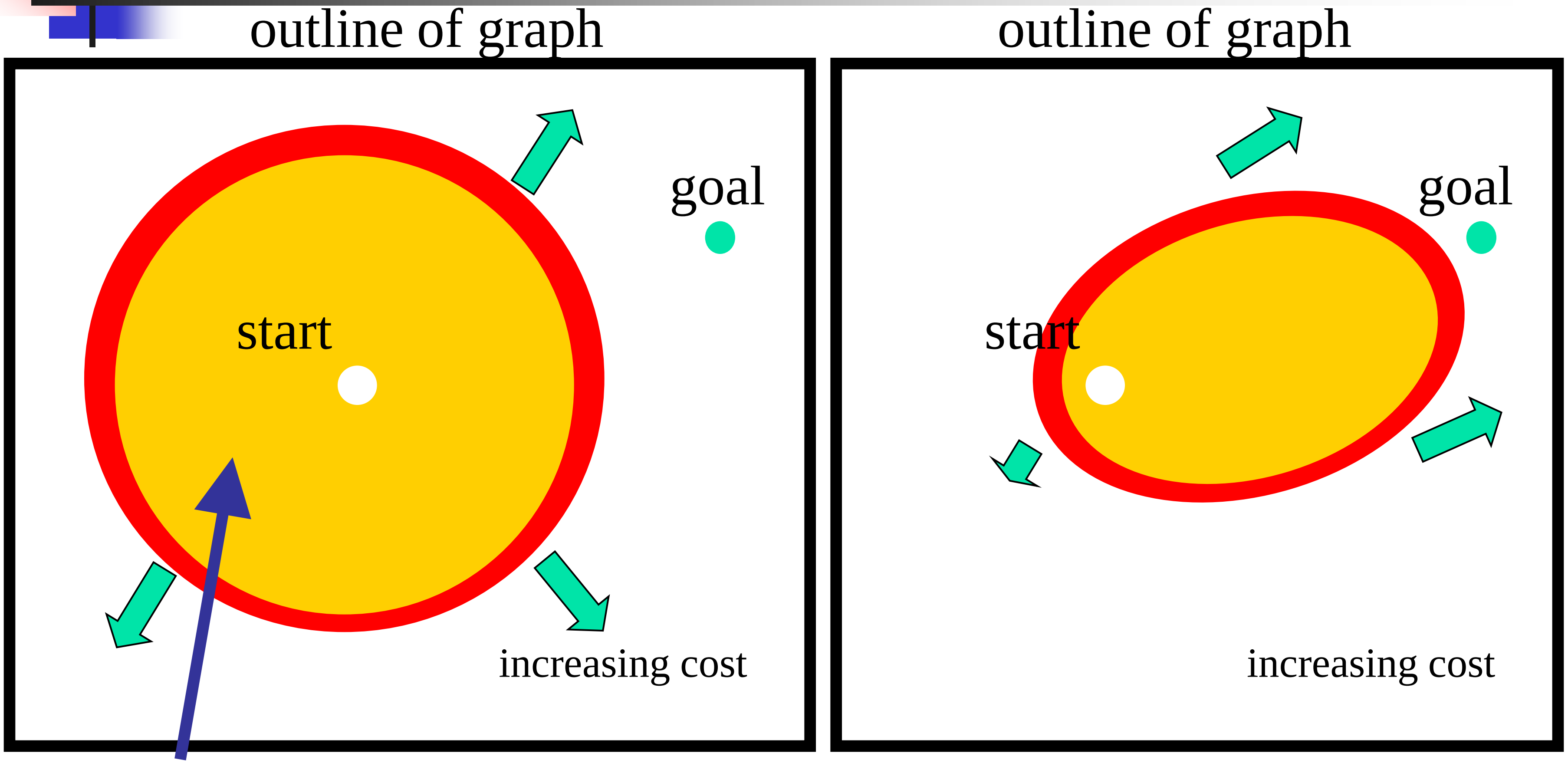
1.Sibiu 2.Rimnicu 3.Faragas 4.Arad 5.Pitesti 6.Zerind 7.Craiova 8.Timisoara

**9.Bucharest 278**

**GOAL!!**



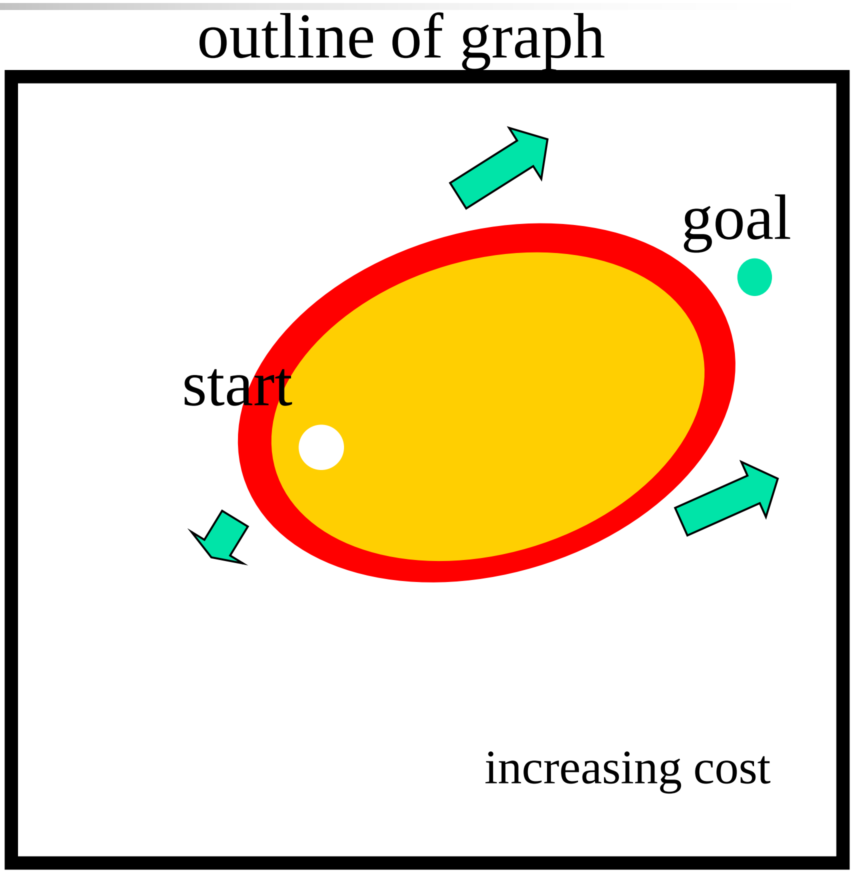
# Heuristic Searches vs. UCS




**This region is basically wasted effort**

# Heuristic Searches vs. UCS

- Want to achieve this but stay
  - complete
  - optimal
- If bias the search "too much" then could miss goals or miss shorter paths



# Heuristic Searches - A* Algorithm

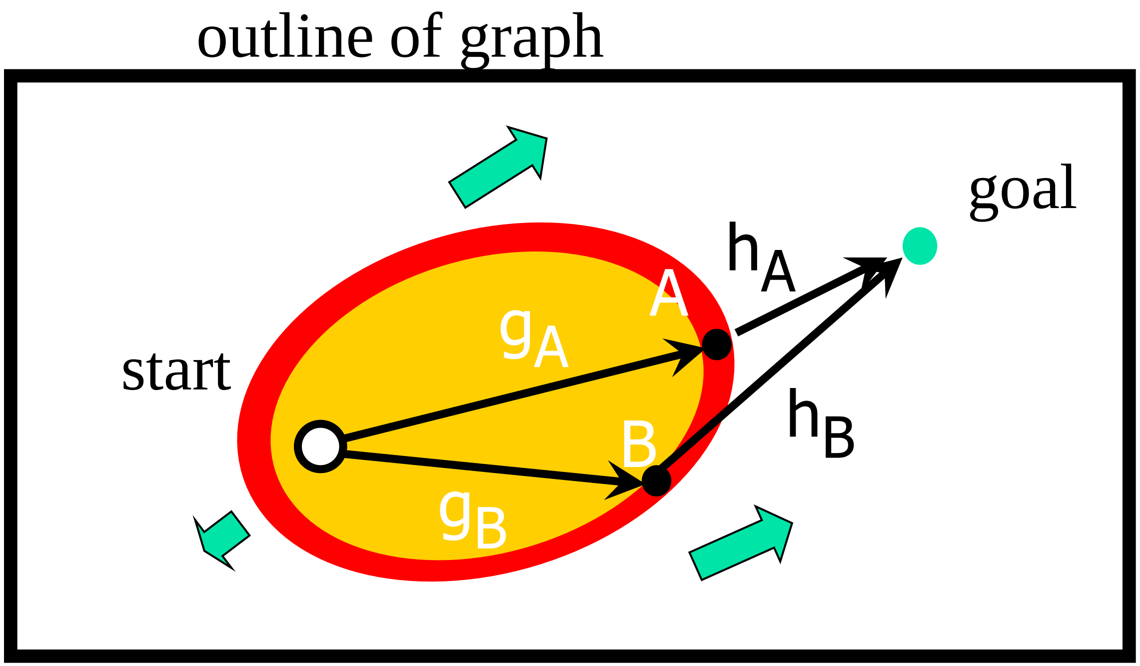
- Combines the cost so far and the estimated cost to the goal

That is fn = $g(n) + h(n)$

This gives us an estimated cost of the cheapest solution through *n*

# Heuristic Searches - A* Algorithm

- We need to have a proper way to estimate *h*



# Heuristic Searches - A* Algorithm

- A search algorithm to find the shortest path through a search space to a goal state using a heuristic.

$$f = g + h$$

  - **f** - function that gives an evaluation of the state
  - **g** - the cost of getting from the initial state to the current state
  - **h** - the cost of getting from the current state to a goal state

# Heuristic Searches - A* Algorithm

- A search algorithm to find the shortest path through a search space to a goal state using a heuristic
  - h=0 A* becomes UCS
    - complete & optimal* but search pattern undirected
  - h too large
    - if *h* is large enough to dominate *g* then becomes like Greedy, lose optimality

*when cost along path never decrease

# Heuristic Searches - A* Algorithm

- It can be proved to be optimal and complete providing that the heuristic is admissible.
- That is the heuristic must never over estimate the cost to reach the goal
  - h(n) must provide a valid lower bound on cost to the goal
- But, the number of nodes that have to be searched still grows exponentially

# Heuristic Searches - A* Algorithm

Function A*-SEARCH(problem) returns a solution of failure

Return BEST-FIRST-SEARCH(problem, $g + h$)

# Straight Line Distances to Bucharest

| Town | SLD |
|---|---|
| Arad | 366 |
| Bucharest | 0 |
| Craiova | 160 |
| Dobreta | 242 |
| Eforie | 161 |
| Fagaras | 178 |
| Giurgiu | 77 |
| Hirsova | 151 |
| Iasi | 226 |
| Lugoj | 244 |

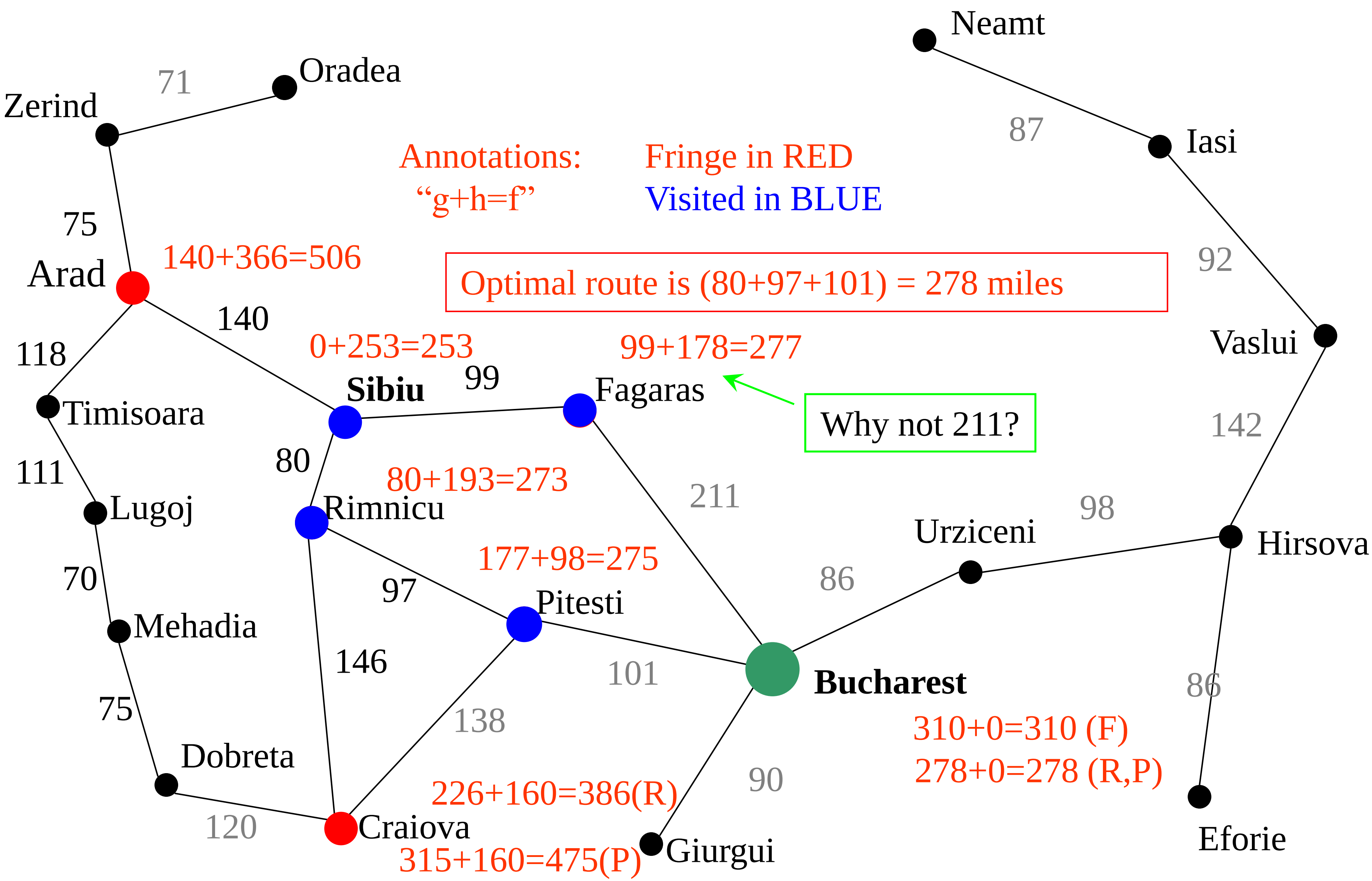
| Town | SLD |
|---|---|
| Mehadai | 241 |
| Neamt | 234 |
| Oradea | 380 |
| Pitesti | 98 |
| Rimnicu | 193 |
| Sibiu | 253 |
| Timisoara | 329 |
| Urziceni | 80 |
| Vaslui | 199 |
| Zerind | 374 |

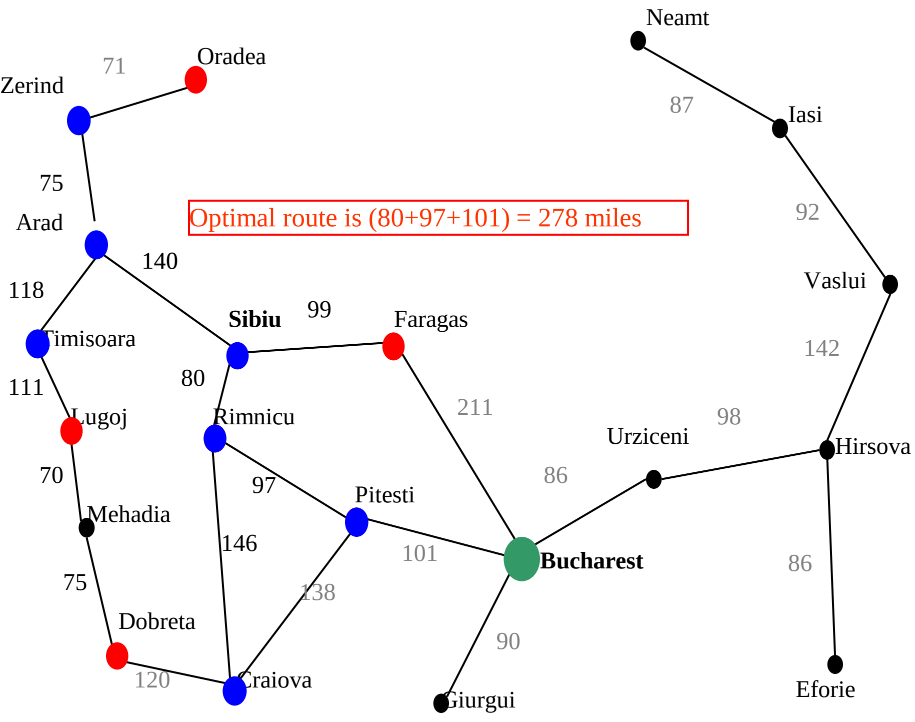
We can use straight line distances as an admissible heuristic as they will never overestimate the cost to the goal. This is because there is no shorter distance between two cities than the straight line distance.

# ANIMATION OF A*.

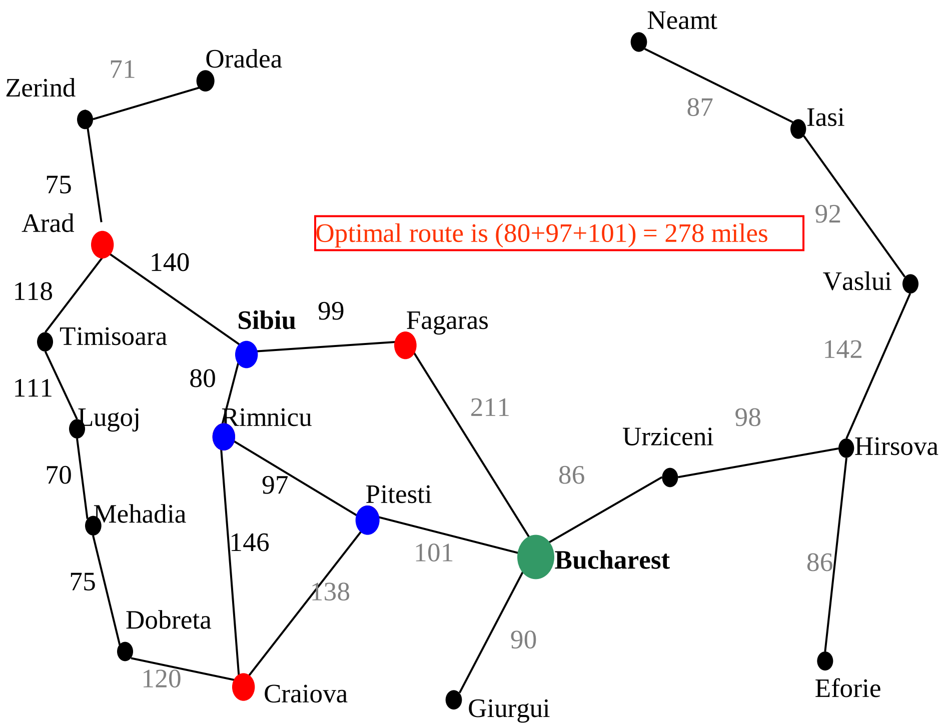
Nodes Expanded

1.Sibiu 2.Rimnicu 3.Pitesti 4.Fagaras **5.Bucharest 278 GOAL!!**






## UCS




## A*

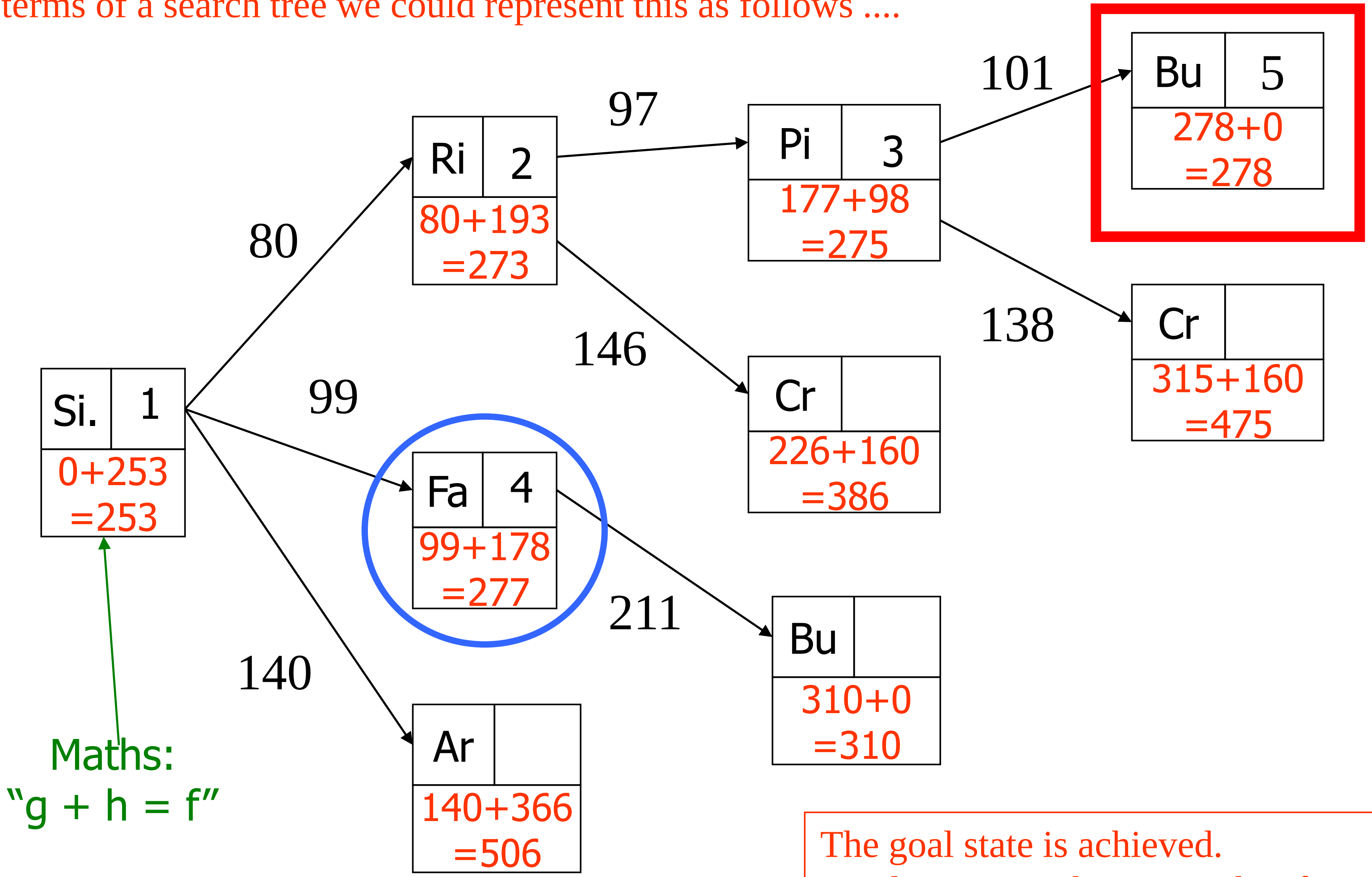
Nodes expanded:

1.Sibiu; 2.Rimnicu; 3.Faragas; 4.Arad; 5.Pitesti; 6.Zerind; 7.Craiova; 8.Timisoara; **9.Bucharest 278**

Nodes Expanded:

1.Sibiu; 2.Rimnicu; 3.Pitesti; 4.Fagaras; **5.Bucharest 278**

In terms of a search tree we could represent this as follows ....




The goal state is achieved.
In relation to path cost, A* has found the optimal route with 5 expansions.
Press space to end.

**Press space to begin the search**

**A* SEARCH TREE**

# Heuristic Searches - A* Algorithm

- Clearly the expansion of the fringe is much more directed towards the goal

- The number of expansions is significantly reduced

# Heuristic Searches - A* Algorithm

- A* is optimal and complete, but it is not all good news
  - It can be shown that the number of nodes that are searched is still exponential to the size of most problems
  - This has implications not only for the time taken to perform the search but also the space required
  - Of these two problems the space complexity is more serious

# Heuristic Searches - A* Algorithm

- If you examine the animation on the previous slide you will notice an interesting phenomenon
  - Along any path from the root, the *f*-cost never decreases
  - This is no accident
  - It holds true for all admissible heuristics

# Heuristic Searches - A* Example

Initial State

| | | |
|---|---|---|
| 1 | 3 | 4 |
| 8 | 6 | 2 |
| 7 | | 5 |

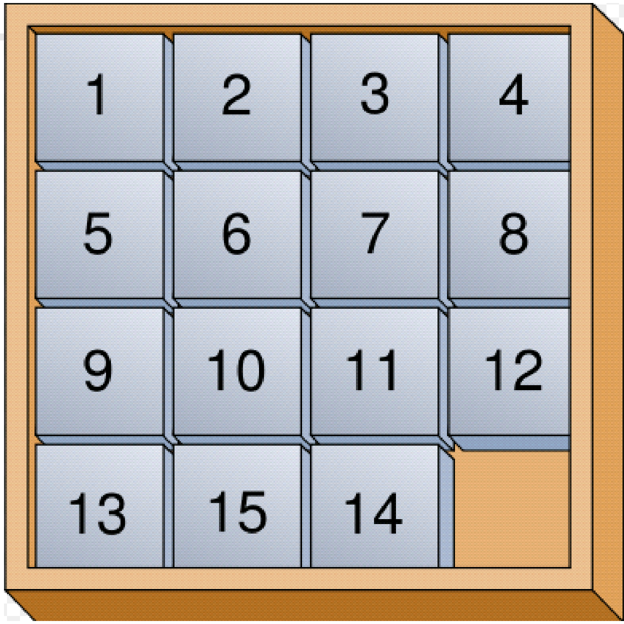
Goal State

| | | |
|---|---|---|
| 1 | 2 | 3 |
| 8 | | 4 |
| 7 | 6 | 5 |

# Heuristic Searches - A* Example

- Typical solution is about twenty steps
- Branching factor is approximately three. Therefore a complete search would need to search $3^{20}$ states. But by keeping track of repeated states we would only need to search 9! (362,880) states
- But even this is a lot (imagine having all these in memory)
- Our aim is to develop a heuristic that does not over estimate (it is admissible) so that we can use A* to find the optimal solution

# Heuristic Searches - A* Example




- 8 puzzle and 15 puzzle
  - Online demo of A* algorithm for 8 puzzle
  - Noyes Chapman's 15 puzzle

# Heuristic Searches
## - Possible Heuristics in A* Algorithm

- $H_1$
  = the number of tiles that are in the wrong position
- $H_2$
  = the sum of the distances of the tiles from their goal positions using the Manhattan Distance

- *We need admissible heuristics (never over estimate)*
  - *Both are admissible but which one is better?*

# Heuristic Searches
## - Possible Heuristics in A* Algorithm

- $H_1$

  = the number of tiles that are in the wrong position (=4)

- $H_2$

  = the sum of the distances of the tiles from their goal positions using the Manhattan Distance (=5)

| | | |
|---|---|---|
| 1 | 3 | 4 |
| 8 | 6 | 2 |
| 7 | | 5 |

| | | |
|---|---|---|
| 1 | 2 | 3 |
| 8 | | 4 |
| 7 | 6 | 5 |

# Heuristic Searches
## - Possible Heuristics in A* Algorithm

| | | |
|---|---|---|
| 1 | 3 | 4 |
| 8 | 6 | 2 |
| 7 | | 5 |

5

| | | |
|---|---|---|
| 1 | 3 | 4 |
| 8 | 6 | 2 |
| 7 | | 5 |

4

- $H_1$ = the number of tiles that are in the wrong position (=4)
- $H_2$ = the sum of the distances of the tiles from their goal positions using the Manhattan Distance (=5)

# Possible Heuristics in A* Algorithm

| | | | |
|---|---|---|---|
| 1 | 3 | 4 | 5 |
| 8 | 6 | 2 | |
| 7 | | 5 | |

What's wrong with this search?
is it A*?

**$H_2$ = the sum of the distances of the tiles from their goal positions using the Manhattan Distance (=5)**

# Possible Heuristics in A* Algorithm

| | | |
|---|---|---|
| 1 | 3 | 4 |
| 8 | 6 | 2 |
| 7 | | 5 |

4

What’s wrong with this search?
is it A*?

**$H_1$ = the number of tiles that are in the wrong position (=4)**

## *Test from 100 runs with varying solution depths using h1 and h2*

| | Search Cost | | |
|---|---|---|---|
| **Depth** | **IDS** | **A*($h_1$)** | **A*($h_2$)** |
| 2 | 10 | 6 | 6 |
| 4 | 112 | 13 | 12 |
| 6 | 680 | 20 | 18 |
| 8 | 6384 | 39 | 25 |
| 10 | 47127 | 93 | 39 |
| 12 | 364404 | 227 | 73 |
| 14 | 3473941 | 539 | 113 |
| 16 | | 1301 | 211 |
| 18 | | 3056 | 363 |
| 20 | | 7276 | 676 |
| 22 | | 18094 | 1219 |
| 24 | | 39135 | 1641 |

| | | |
|---|---|---|
| 5 | 4 | |
| 6 | 1 | 8 |
| 7 | 3 | 2 |

| | | |
|---|---|---|
| 1 | 2 | 3 |
| 8 | | 4 |
| 7 | 6 | 5 |

$H_2$ looks better as fewer nodes are expanded. But why?

# Effective Branching Factor

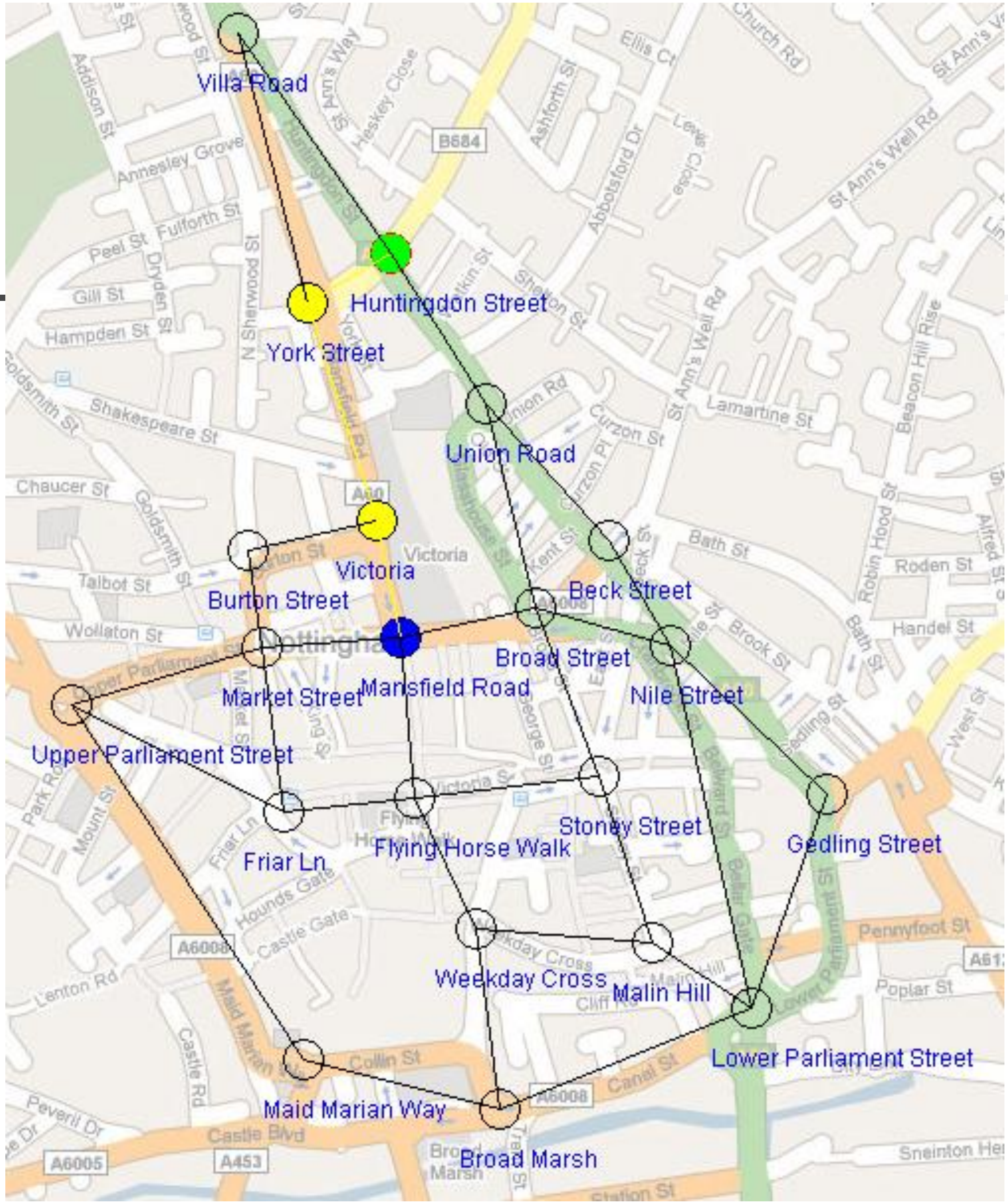
| | Search Cost | | | EBF | | |
|---|---|---|---|---|---|---|
| **Depth** | **IDS** | **A*($h_1$)** | **A*($h_2$)** | **IDS** | **A*($h_1$)** | **A*($h_2$)** |
| 2 | 10 | 6 | 6 | 2.45 | 1.79 | 1.79 |
| 4 | 112 | 13 | 12 | 2.87 | 1.48 | 1.45 |
| 6 | 680 | 20 | 18 | 2.73 | 1.34 | 1.30 |
| 8 | 6384 | 39 | 25 | 2.80 | 1.33 | 1.24 |
| 10 | 47127 | 93 | 39 | 2.79 | 1.38 | 1.22 |
| 12 | 364404 | 227 | 73 | 2.78 | 1.42 | 1.24 |
| 14 | 3473941 | 539 | 113 | 2.83 | 1.44 | 1.23 |

- Effective branching factor: average number of branches expanded
- $H_2$ has a lower branching factor and so fewer nodes are expanded
- Therefore, one way to measure the quality of a heuristic is to find its average branching factor
- $H_2$ has a lower EBF and is therefore the better heuristic

# Domination

| | Search Cost | | | EBF | | |
|---|---|---|---|---|---|---|
| **Depth** | **IDS** | **A*($h_1$)** | **A*($h_2$)** | **IDS** | **A*($h_1$)** | **A*($h_2$)** |
| 2 | 10 | 6 | 6 | 2.45 | 1.79 | 1.79 |
| 4 | 112 | 13 | 12 | 2.87 | 1.48 | 1.45 |
| 6 | 680 | 20 | 18 | 2.73 | 1.34 | 1.30 |
| 8 | 6384 | 39 | 25 | 2.80 | 1.33 | 1.24 |
| 10 | 47127 | 93 | 39 | 2.79 | 1.38 | 1.22 |
| 12 | 364404 | 227 | 73 | 2.78 | 1.42 | 1.24 |
| 14 | 3473941 | 539 | 113 | 2.83 | 1.44 | 1.23 |

- For any node: h2(n) <= h1(n)
- h2 dominates h1


Villa Road
Huntingdon Street
York Street
Union Road
Victoria
Burton Street
Beck Street
Broad Street
Market Street
Mansfield Road
Nile Street
Upper Parliament Street
Stoney Street
Gedling Street
Friar Ln
Flying Horse Walk
Weekday Cross
Malin Hill
Lower Parliament Street
Maid Marian Way
Broad Marsh

Use Uniform Cost Search to find the shortest path from A to F in the map below (not drawn to scale). You should not re-visit a node that you have just come from.

Show at each step what fringe nodes are in the queue (5 marks).

Show the list of nodes that are expanded (3 marks).

State the shortest route you take and its cost (2 marks).

Can Uniform Cost Search guarantee to find the optimal solution for this problem? Explain the reason. (3 marks)