Reasoning Systems
Day 1

Course Manager: GU Zhan (Sam)
zhan.gu@nus.edu.sg

Reasoning Systems

Day 1

Day 1 Agenda

1.1 Reasoning Systems Overview

- 1.1.1 Reasoning Systems: Retrieval vs. Generative
- 1.1.2 Hybrid Reasoning Systems
- 1.1.3 Exercise

1.2 Reasoning Using Uninformed Search

- 1.2.1 Uninformed Search Techniques (Tree)
- 1.2.2 Uninformed Search Techniques (Graph)
- 1.2.3 Exercise

1.3 Search Representation [Workshop]

- 1.3.1 Tree Search & Graph Search
- 1.3.2 Knowledge Graph Reasoner
- 1.3.3 Workshop Submission

Google

who is sam gu zhan

All | Images | News | Maps | Videos | More | Settings | Tools

About 335,000 results (0.61 seconds)

Analytics & Intelligent Systems - NUS-ISS
https://www.iss.nus.edu.sg/about-us/iss-team/teaching.../analytics-intelligent-systems
Ms. FAN **Zhen Zhen**. Senior Lecturer & Consultant, Analytics & Intelligent ... **GU Zhan**. Mr. **GU Zhan**. Lecturer & Consultant, Analytics & Intelligent Systems ...

GU Zhan - NUS-ISS
https://www.iss.nus.edu.sg/about-us/staff/detail/201/GU%20Zhan
GU Zhan (**Sam**) lectures Master of Technology programme in the areas of data science, machine intelligence, soft computing, and applied deep learning. Prior to ...

Zhan GU | LinkedIn
https://sg.linkedin.com/in/zhan-gu-27a82823
As a lecturer and consultant in applied machine intelligence, **Zhan GU** (**Sam**) engages communities and schools to help organizations making sense of their ...

sam gu - Principal Manager - Qualcomm | LinkedIn
https://www.linkedin.com/in/sam-gu-97b65b102
San Diego, California - Qualcomm
View **sam gu's** profile on LinkedIn, the world's largest professional community. ... See the complete profile on LinkedIn and discover sam's connections and jobs at similar companies. ... Li **Zhang**. Vice President Engineering at Qualcomm Inc ...
Missing: ~~zhan~~ | Must include: zhan

GitHub - telescopeuser/GCP-SamGu
https://github.com/telescopeuser/GCP-SamGu
Contribute to telescopeuser/GCP-**SamGu** development by creating an account on GitHub. ... Gu **Zhan** support TensorFlow-v1.6. Latest commit 1dd2d5e on Apr ...

Images for who is sam gu zhan

More images for who is sam gu zhan — Report images

Google

how to pass exam

- how to pass exam**s**
- how to pass exam**s without studying**
- how to pass exam**s pdf**
- how to pass exam**s easily**
- how to pass exam**s dominic o'brien pdf**
- how to pass exam **p**
- how to pass exam**s with top grades**
- how to pass exam **in one night**
- how to pass exam **fm**
- how to pass exam**s without reading**

4. Use all of your senses while **studying**:
5. Partner up with someone in your class:
6. Wear a watch:
7. Teach a class of stuffed animals:
8. Drink water and eat fruit:

More items...

Report inappropriate predictions

10 Study Hacks That Will Help You Ace Your Final Exams - Business ...
https://www.businessinsider.com/study-hacks-for-final-exams-2013-12

About this result | Feedback

People also ask

- How do you get good grades without studying?
- Can you pass GED without studying?
- How can I pass the exam?
- How can I study effectively at the last minute?

Feedback

4 Ways to Pass a Class Without Really Studying - wikiHow
https://www.wikihow.com/Pass-a-Class-Without-Really-Studying
★★★☆☆ Rating: 56% - 121 votes
Studying may not be your forte, but that shouldn't prevent you from **passing** your class! ... most out of your class time, you may be able to **pass** your class **without studying**. ... Sitting in the same seat may help trigger your memory on **exam** days.

1.1 Reasoning Systems Overview

1.1.1 Reasoning Systems: Retrieval vs. Generative

1.1.2 Hybrid Reasoning Systems

1.1.3 Exercise

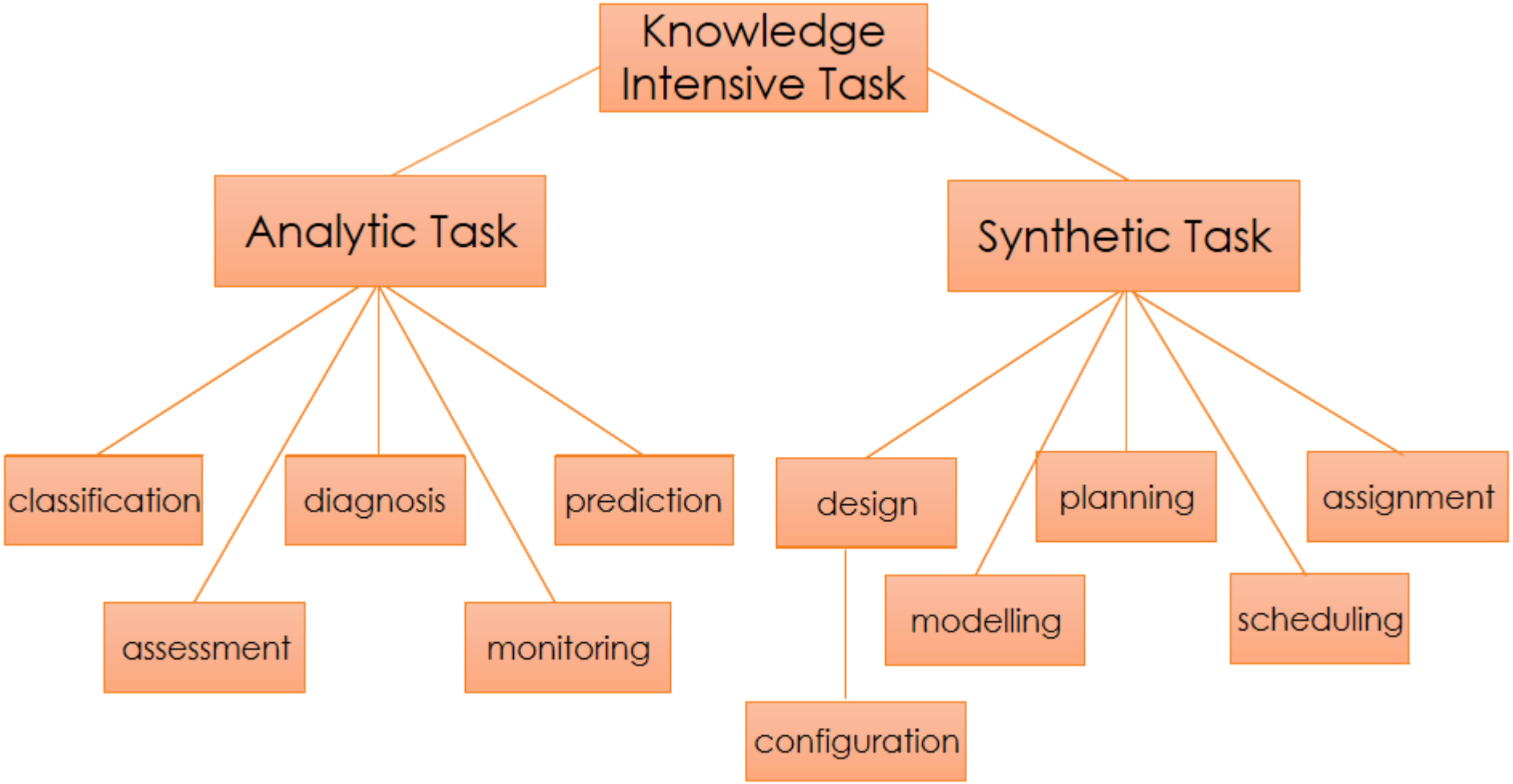
1.1 Reasoning Systems Overview

1.1.1 Reasoning Systems: Retrieval vs. Generative

1.1.2 Hybrid Reasoning Systems

1.1.3 Exercise

AI for Knowledge Intensive Task



Reasoning Systems: Retrieval vs. Generative

- **Analytic Tasks** **E.g. Coronavirus (COVID-19) diagnosis**
  - System/Solution to be analysed pre-exists, but usually not completely "known".
  - Input: some data to trigger the system (e.g. patient symptoms)
  - Output: some characterization or behaviours about the system (e.g. cause of illness)
  - - E.g. Medical diagnosis

  A reasoning system which can retrieve/calculate a (known) solution from/using a knowledge base, containing solutions. Obtain knowledge (using knowledge discovery & ML); Represent knowledge (based on various forms of reasoning types); Obtain solution by using knowledge on new problem scenario (inference engine or calling functions);

- **Synthetic Tasks** **E.g. Coronavirus (COVID-19) vaccination creation**
  - System/Solution does not yet exist.
  - Input: requirements about system to be constructed
  - Output: constructed system description

  A reasoning system which can generate a (new) solution using a knowledge base, containing goal test or solution evaluation knowledge. Generate candidate solutions systematically; Test candidate solution using goal test knowledge; Rank candidate solutions based on evaluation function/knowledge; Alter solutions to generate new candidate solutions; Repeat Test candidate… until a termination criteria, e.g. one minute, is satisfied;

Reasoning Systems: Retrieval vs. Generative

- **Analytic Tasks** (Retrieval Reasoning Systems)

  Identification, Classification, Prediction, Clustering/Grouping, …

- **Techniques**

  Heuristic Business Rules

  Decision Trees

  Similarity Based Reasoning

  Fuzzy Logic

  Knowledge Discovery

  …

Suspect of depression by overwhelming daily stresses, but shy to consult a doctor?

Empower you to conduct professional assessment DIY, with:

Depression Screener

[Health] Depression Screener

Source: https://github.com/IRS-MR/IRS-MR-2019-01-19-IS1PT-GRP-Pepper-DepressionScreeningSystem

Reasoning Systems: Retrieval vs. Generative

- **Synthetic Tasks** (Generative Reasoning Systems)

  Planning, Scheduling, Optimisation, Design, …

- **Techniques**

  Uninformed (brute force / blind) Search

  Informed (heuristic) Search

  Simulations

  Genetic Algorithms

  Reinforcement Learning

  …

Given a scenario, can you tell whether it is a analytic or synthetic task?

Want to consult a depression specialist, but worried about shy personality, Chinese speaking only, and other concerns... Which doctor to see?

Empower you to meet a doctor suitable to your needs, with:

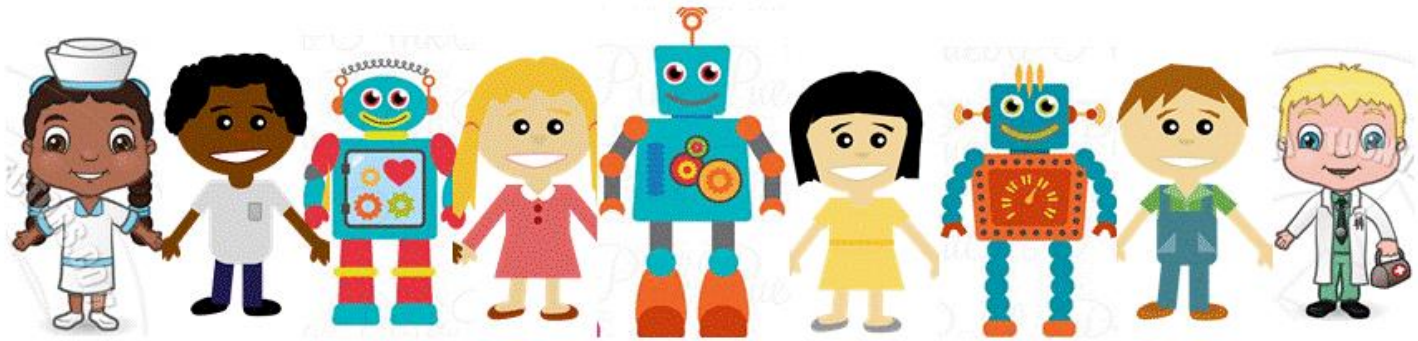
Patient-Doctor Matcher

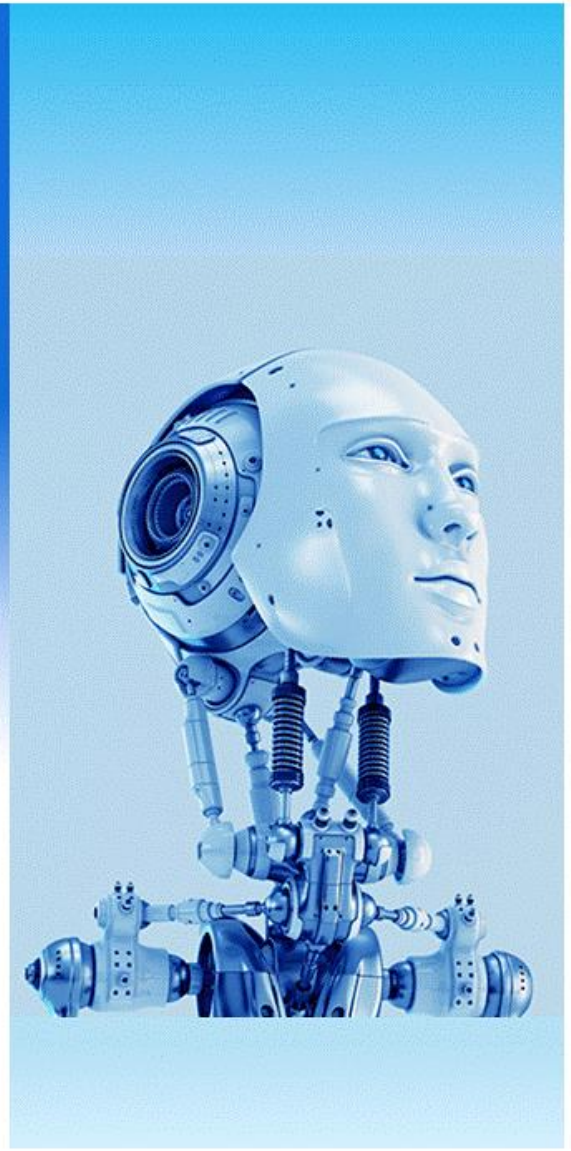
[Health] Patient-Doctor Matcher

Pepper Project Group

CAO LIANG	A0012884E
GENG LIANGYU	A0195278M
HAN DONGCHOU FRANCIS	A0195414A
ONG BOON PING	A0195172B
TAN CHIN GEE	A0195296M

Patient Matching System





Source: https://github.com/IRS-RS/IRS-RS-2019-03-09-IS1PT-GRP-Pepper-PatientMatchingSystem

1.1 Reasoning Systems Overview

1.1.1 Reasoning Systems: Retrieval vs. Generative

1.1.2 Hybrid Reasoning Systems

1.1.3 Exercise

Hybrid Reasoning Systems

Hybrid Intelligent systems are intelligent systems which combine two or more machine reasoning techniques:

- **Deductive Reasoning used in Analytic Problem Solving**

  Techniques: Knowledge-driven Rule/Process Systems; Information Retrieval; Fuzzy Logic; Belief Desire Intention (BDI) Cognitive framework

- **Inductive Reasoning used in Data Mining & Machine Learning**

  Techniques: Decision Tree; Association Rule; Neural Networks; Bayesian Net; Knowledge Graph; Reinforcement Learning

- **Planning & Optimization used in Synthetic Problem Solving**

  Techniques: Search; Genetic Algorithms; Constrain Satisfaction; Swarm Intelligence;

☺ **Another common term for hybrid intelligent systems is "Hybrid Soft Computing Systems".**

Hybrid Reasoning Systems

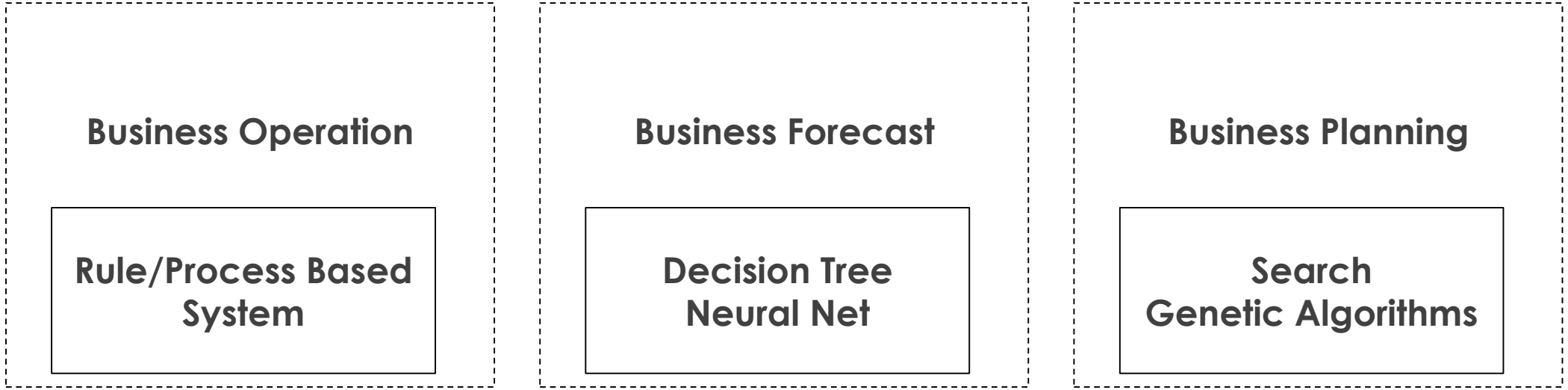
Four broad types of system architectures

1. Independent Sub-systems
2. Competing Experts
3. Self-tuning
4. Cooperating Experts

Hybrid Reasoning Systems

1. Independent Sub-systems

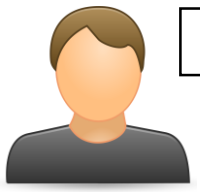
- **Sub-divide problem into independent parts.**
- **Each is solved by an appropriate technique or sub-system.**
- **No cooperation is required, e.g. a decision support system has several independent sub-systems (functions).**



Hybrid Reasoning Systems

2. Competing Experts

- **Different solution strategies (experts) offer alternative solutions.**
- **Another process decides which solution to accept or how to combine the solutions, e.g. majority vote algorithm or a rule-based system. (auction, tender, bidding)**



Rule Based System

Decision Tree

Genetic Algorithm

Search System

Individual Experts

Arbitrator System → Solution / Decision / Forecasting

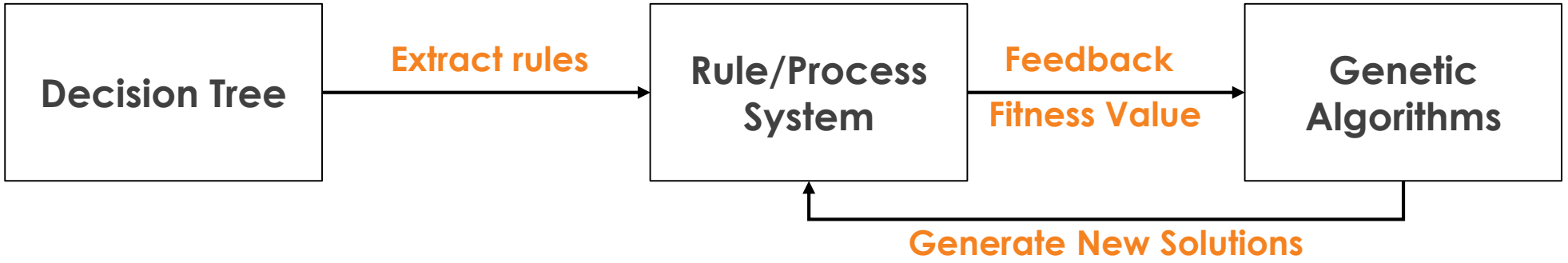
Simple Committee:
If majority of experts agree then good decision, else unreliable decision.

Ensemble Arbitrator:
Weight expert judgments according to circumstances.

Hybrid Reasoning Systems

3. Self-tuning

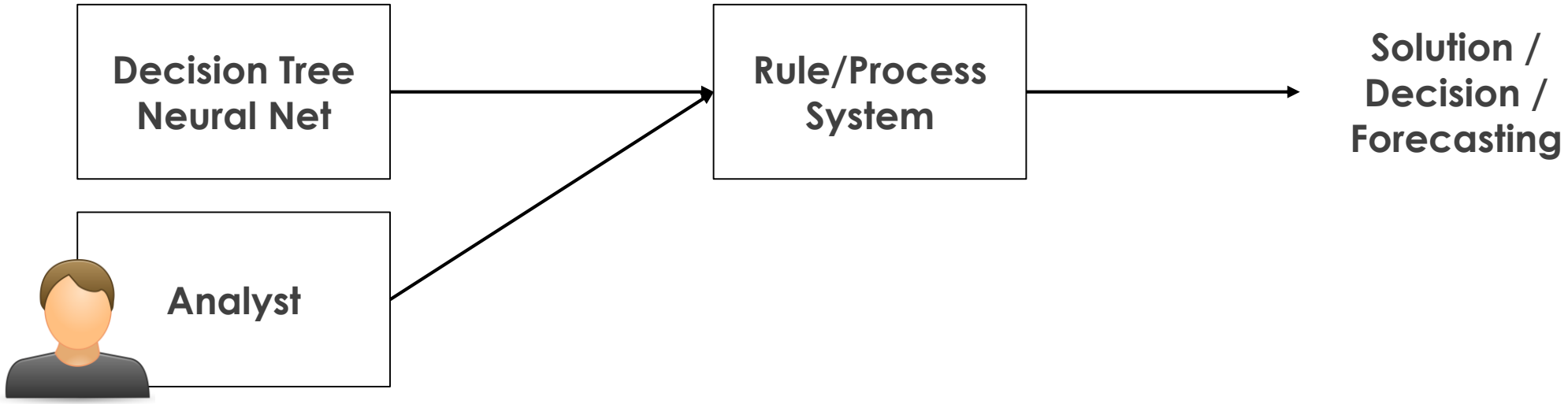
- **One technique is used to tune or learn the architecture for another, e.g. Decision Tree used to learn a ruleset, e.g. trading strategies; Genetic Algorithms used to optimise initial solution, e.g. investment portfolio, based on predicted/simulated value from rule based system, in order to maximize trading profit.**



Hybrid Reasoning Systems

4. Co-operating Experts

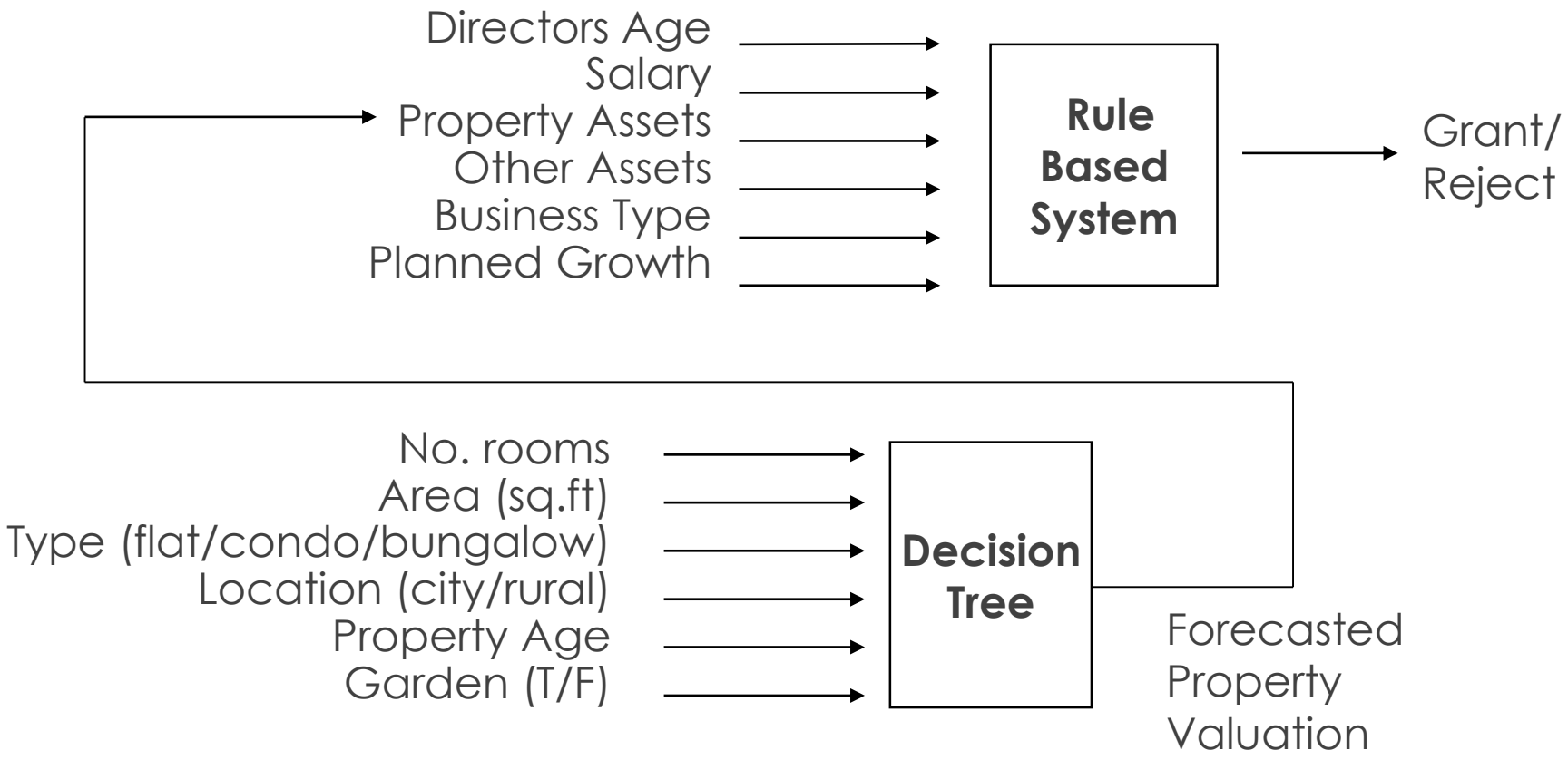
- **Pooled Expertise: Different techniques/sub-systems work together as a team to produce a single solution. No single technique/expert is sufficient alone, e.g. Machine learning and human analyst both provide inputs required to forecast macroeconomics based on business rules and weights, under different market conditions.**



Hybrid Reasoning Systems

4. Co-operating Experts

- **Use case: Approving business loan to a small company**



Hybrid Reasoning Systems

Summary

- **Hybrid Systems offer solutions to a greater range of problems: "The sum is greater than the parts".**

- **Four categories of typical hybrid system architectures have been introduced. These are not exhaustive: Other patterns are possible.**

1.1 Reasoning Systems Overview

1.1.1 Reasoning Systems: Retrieval vs. Generative

1.1.2 Hybrid Reasoning Systems

1.1.3 Exercise

[Exercise] Reasoning Systems Overview

Case Study

- A soup manufacturer plans to produce a new type of canned soup. The new soup will contain up to **27** ingredients, namely **10** types of meat/fish, **7** types of vegetables, **5** flavour enhancers, **3** types of preservative, salt and sugar.
- To determine the relative quantities of each of the ingredients, the company conducts a market survey. Several hundred volunteers taste various prototype soups in which the **27** ingredients are mixed in different ratios, e.g. **30%** chicken feet, **14%** fish eyes, **15%** turnips. Each taster is given **5** prototype soups to taste and asked to assign each to one of **7** categories.

7 Categories for describing the prototype soup

(a) horrible taste
(b) would only eat if very hungry
(c) weak taste
(d) average taste
(e) good taste
(f) very good taste
(g) heavenly taste

Discussion

- Suggest a top-level design for a hybrid system that uses the results of the market survey to determine the mix of raw ingredients likely to achieve the highest consumer rating.

[Exercise] Reasoning Systems Overview

- **What's the end solution look like? (knowledge/solution representation)**

Start with an assumption of the user of system

reasonable consumers:
Restaurant owners,
Generic public

Provide a soup recipe

What is the structure of the final solution: percentage of the ingredient

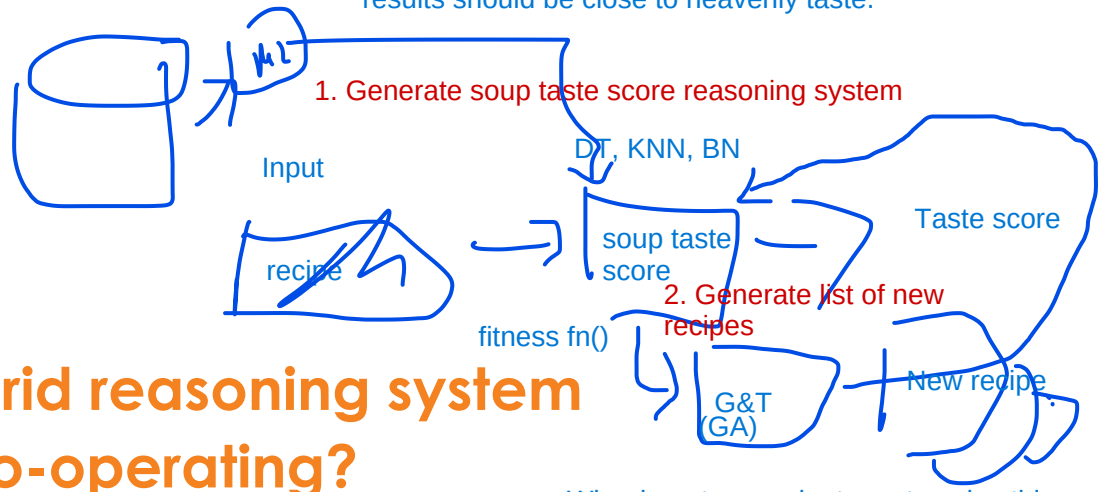
results should be close to heavenly taste.

- **Analytic task or Synthetic task?**

Synthetic task if we assume that tasks are not defined.
Analytic task if we consider only the 5 prototype recipes.

- **Retrieval system or Generative system?**

Likewise, it could be a generative or retrieval tass.



- **Decompose into sub-systems, using hybrid reasoning system architecture, competing? self-tuning? co-operating?**

Identify the problem that you are trying to solve. This is a classification problem.

Identify a classification algorithm suitable.

Why do not use volunteers to solve this issue?

Approach is to collect data from a set of recipes

The generative and Test is actually an optimisation problem.

- **Design the sub-systems using knowledge models, e.g. box diagram, process flowchart, etc.**

Generate new recipe

random forest is good at identifying feature importance.

1.2 Reasoning Using Uninformed Search

1.2.1 Uninformed Search Techniques (Tree)

1.2.2 Uninformed Search Techniques (Graph)

1.2.3 Exercise

1.2 Reasoning Using Uninformed Search

1.2.1 Uninformed Search Techniques (Tree)

1.2.2 Uninformed Search Techniques (Graph)

1.2.3 Exercise

Uninformed Search Techniques

Solving Problem by Search

- Synthesis of a new valid solution is performed by searching through the **(search/solution) space**, which defines all possible solutions;
- Each possible solution is evaluated to see whether it is **valid** and/or the **optimum** (best solution found so far), e.g. a valid employee schedule, a valid vehicle delivery route, an optimal (shortest) vehicle delivery route;
- Validity of solution involves satisfaction of a set of **constraints/business requirements** on the solution variables (**goal test**);
- Optimality is measured by a **user-defined evaluation function** which measures the "goodness" of the solution, e.g. the shorter delivery route the better;

Uninformed Search Techniques

Solving Problem by Search

(1) Create a pool of solution candidates (search space)

(2) Pick up one candidate solution from pool

(3) Check whether this candidate is valid (constraints satisfied?)

- (3)=True If valid, continue
- (3)=False If not valid, go to (2)

(4) Check whether this candidate is the best till now (optimal solution?)

- (4)=True If best, save this solution as the best then continue
- (4)=False If not best, discard this solution then continue

(5) Go to (2). Repeat the cycle until a stopping criteria is met.

Uninformed Search Techniques (Tree)

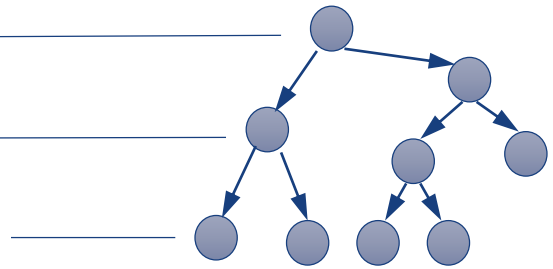
Search Tree Representation

- **Search is illustrated using a search space with a particular restricted structure**
- **Solutions (search space) can be represented as a Tree**
  - Nodes in tree represent

    an initial state

    an intermediate state

    a final state (feasible solution, or failure)



  - Connection between nodes represents a search step

Uninformed Search Techniques (Tree)

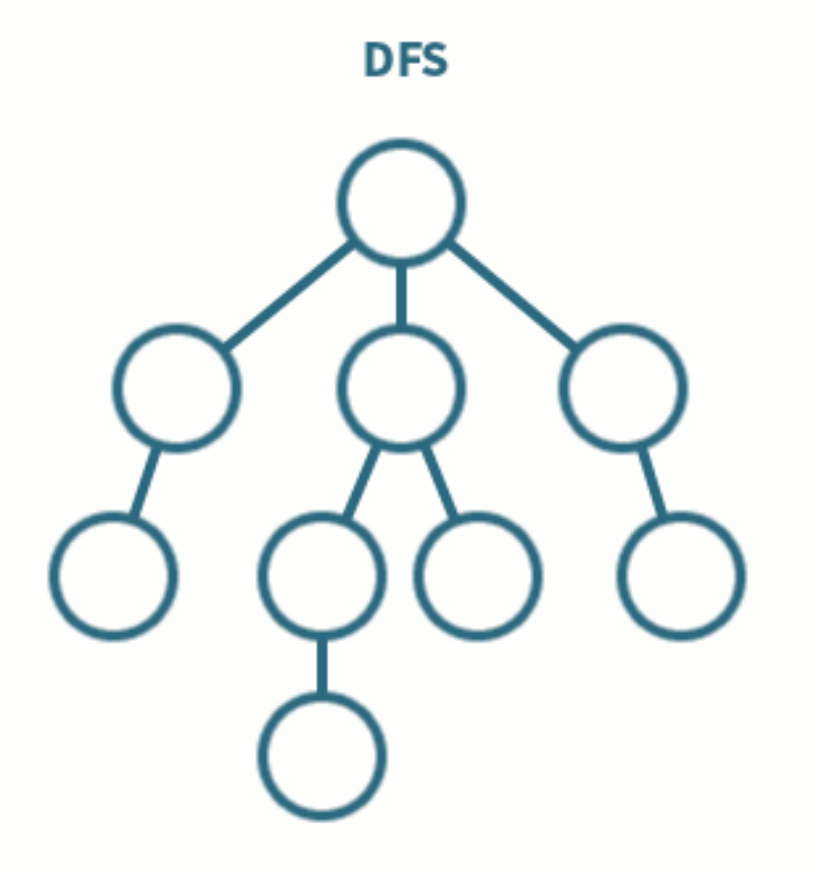
Depth First Search (DFS)

- **Always prefers to search deeper in the search tree rather than wider.**

Uninformed Search Techniques (Tree)

Depth First Search (DFS)

- **Visit order**



https://medium.com/@kenny.hom27/breadth-first-vs-depth-first-tree-traversal-in-javascript-48df2ebfc6d1

Uninformed Search Techniques (Tree)

Depth First Search (DFS)

Algorithm Pseudo Code

(1) Set N to be a list (open list/frontier list) of initial nodes

(2) If N is empty, then exit and signal failure

(3) Set n to be the first node in N, and remove n from N

(4) Check n:

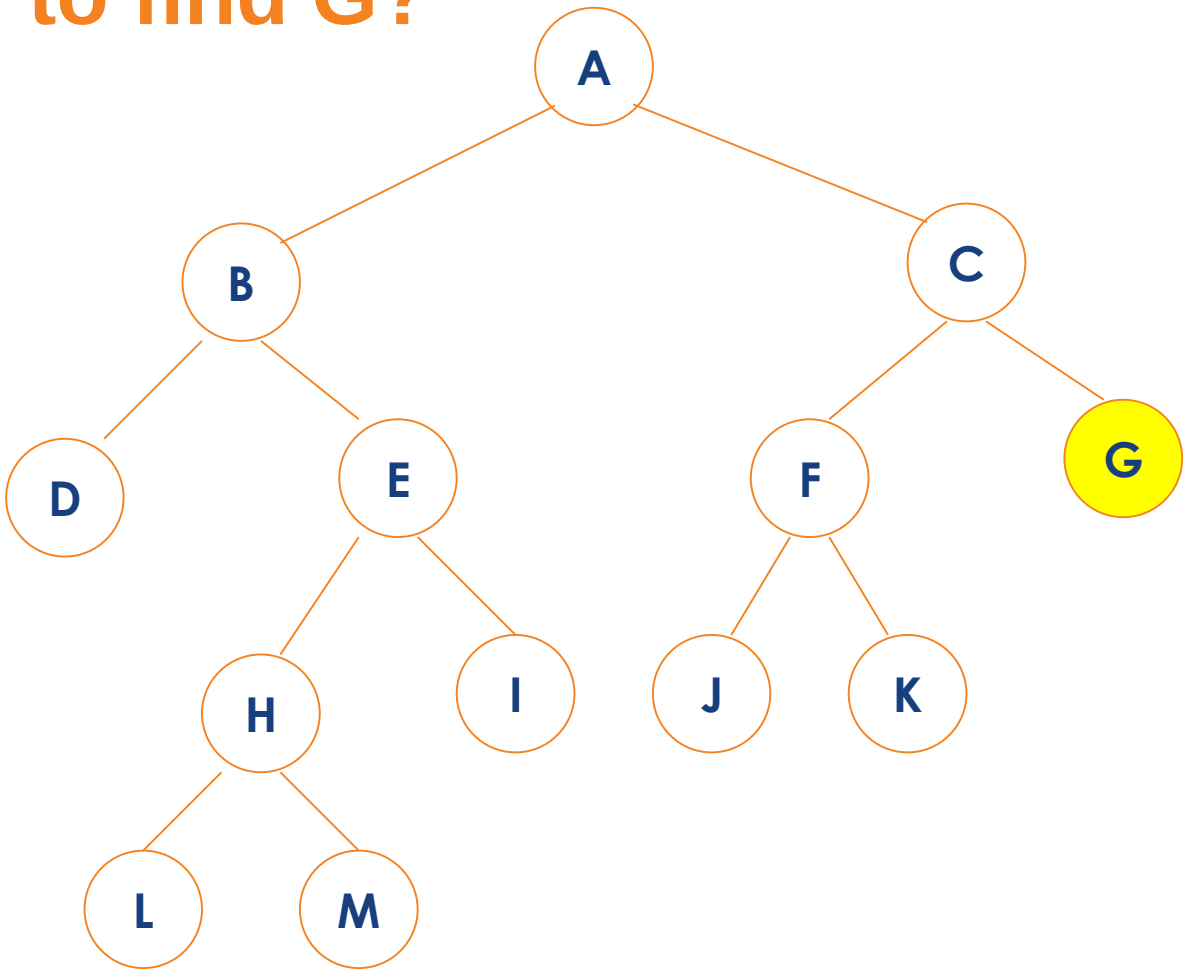
(4.1) If **n** is a goal node, then exit and signal success

(4.2) Otherwise, add the children of **n** to the **front** of **N** then go to step **(2)**

Uninformed Search Techniques (Tree)

Depth First Search (DFS)

- **Visit order to find G?**



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

Uninformed Search Techniques (Tree)

Breadth First Search (BFS)

- **Explores all the nodes at a given depth before processing deeper in the search tree.**

Uninformed Search Techniques (Tree)

Breadth First Search (BFS)

- **Visit order**

https://medium.com/@kenny.hom27/breadth-first-vs-depth-first-tree-traversal-in-javascript-48df2ebfc6d1

Uninformed Search Techniques (Tree)

Breadth First Search (BFS)

Algorithm Pseudo Code

(1) Set N to be a list (open list/frontier list) of initial nodes

(2) If N is empty, then exit and signal failure

(3) Set n to be the first node in N, and remove n from N

(4) Check n:

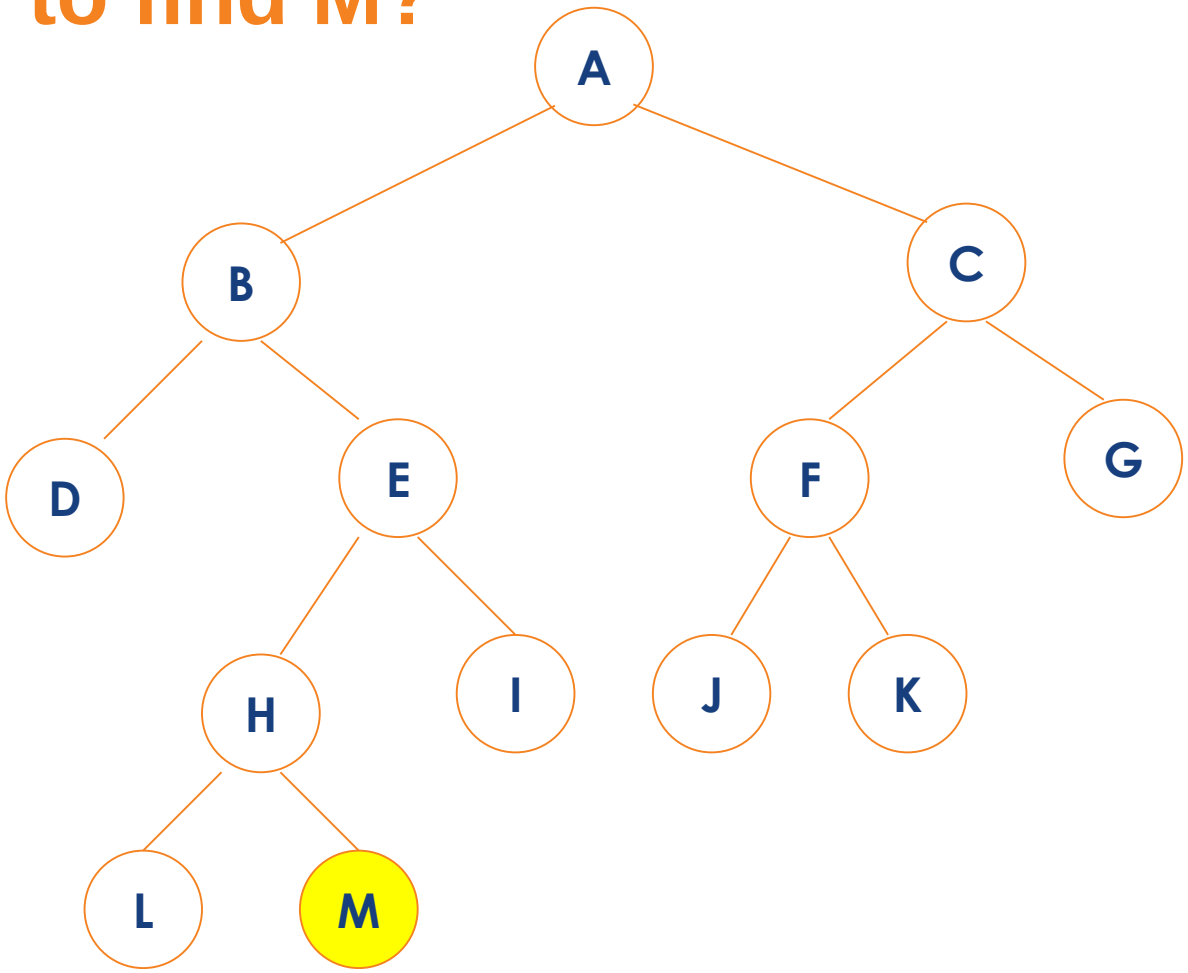
(4.1) If **n** is a goal node, then exit and signal success

(4.2) Otherwise, add the children of **n** to the **end** of **N** then go to step **(2)**

Uninformed Search Techniques (Tree)

Breadth First Search (BFS)

- **Visit order to find M?**



Inference by Search Example

Propositional logic

1. WHEN BodyCoverType = HasHair THEN AnimalType = Mammal
2. WHEN FeedType = FeedMilk THEN AnimalType = Mammal
3. WHEN BodyCoverType = HasFeather THEN AnimalType = Bird
4. WHEN MoveType = CanFly AND ReproduceType = LayEgg THEN AnimalType = Bird
5. WHEN AnimalType = Mammal AND FeedType = EatMeat THEN AnimalType = Carnivore
6. WHEN AnimalType = Mammal AND ToothType = HasPointedTeeth AND FootType = HasClaws AND EyeType = HasForwardEyes THEN AnimalType = Carnivore
7. WHEN AnimalType = Mammal AND FeedType = EatGrass THEN AnimalType = Herbivore
8. WHEN AnimalType = Mammal AND FootType = HasHooves THEN AnimalType = Herbivore
9. WHEN AnimalType = Carnivore AND ColorType = HasColorTawny AND MarkingType = HasDarkSpots THEN AnimalName = Cheetah
10. WHEN AnimalType = Carnivore AND ColorType = HasColorTawny AND MarkingType = HasDarkStripes THEN AnimalName = Tiger
11. WHEN AnimalType = Herbivore AND ColorType = HasColorTawny AND MarkingType = HasDarkSpots AND NeckType = HasLongNeck THEN AnimalName = Giraffe
12. WHEN AnimalType = Herbivore AND ColorType = HasColorBlackWhite THEN AnimalName = Zebra
13. WHEN AnimalType = Bird AND MoveType = CanWalk AND ColorType = HasColorBlackWhite AND NeckType = HasLongNeck THEN AnimalName = Ostrich
14. WHEN AnimalType = Bird AND MoveType = CanWwim AND ColorType = HasColorBlackWhite THEN AnimalName = Penguin
15. WHEN AnimalType = Bird AND MoveType = CanFly AND ColorType = HasColorBlackWhite THEN AnimalName = Albatross

Inference by Search Example

Predicate logic

X is an (instance of) animal (class).

1. HasHair(X) → Mammal(X)
2. FeedMilk(X) → Mammal(X)
3. HasFeather(X) → Bird(X)
4. CanFly(X) ^ LayEgg(X) → Bird(X)
5. Mammal(X) ^ EatMeat(X) → Carnivore(X)
6. Mammal(X) ^ HasPointedTeeth(X) ^ HasClaws(X) ^ HasForwardEyes(X) → Carnivore(X)
7. Mammal(X) ^ EatGrass(X) → Herbivore(X)
8. Mammal(X) ^ HasHooves(X) → Herbivore(X)
9. Carnivore(X) ^ HasColorTawny(X) ^ HasDarkSpots(X) → Cheetah(X)
10. Carnivore(X) ^ HasColorTawny(X) ^ HasDarkStripes(X) → Tiger(X)
11. Herbivore(X) ^ HasColorTawny(X) ^ HasDarkSpots(X) ^ HasLongNeck(X) → Giraffe(X)
12. Herbivore(X) ^ HasColorBlackWhite(X) → Zebra(X)
13. Bird(X) ^ CanWalk(X) ^ HasColorBlackWhite(X) ^ HasLongNeck(X) → Ostrich(X)
14. Bird(X) ^ CanSwim(X) ^ HasColorBlackWhite(X) → Penguin(X)
15. Bird(X) ^ CanFly(X) ^ HasColorBlackWhite(X) → Albatross(X)

Inference by Search Example

Predicate logic

- **Knowledge Base (KB)**
  - Rule sets plus below Facts:
  - HasHair(X)
  - HasClaws(X)
  - HasPointedTeeth(X)
  - HasForwardEyes(X)
  - HasColorTawny(X)
  - HasDarkSpots(X)

An example of DFS, going backwards.

- **Clouse form conversion : $p \rightarrow q \equiv \neg p \vee q$**
- **Hypothesis to prove is : α = Cheetah(X)**
- **Refutation of hypothesis is : $\neg\alpha = \neg$Cheetah(X)**

Inference by Search Example

Predicate logic

1. HasHair(X) → Mammal(X)
2. FeedMilk(X) → Mammal(X)
3. HasFeather(X) → Bird(X)
4. CanFly(X) ^ LayEgg(X) → Bird(X)
5. Mammal(X) ^ EatMeat(X) → Carnivore(X)
6. Mammal(X) ^ HasPointedTeeth(X) ^ HasClaws(X) ^ HasForwardEyes(X) → Carnivore(X)
7. Mammal(X) ^ EatGrass(X) → Herbivore(X)
8. Mammal(X) ^ HasHooves(X) → Herbivore(X)
9. Carnivore(X) ^ HasColorTawny(X) ^ HasDarkSpots(X) → Cheetah(X)
10. Carnivore(X) ^ HasColorTawny(X) ^ HasDarkStripes(X) → Tiger(X)
11. Herbivore(X) ^ HasColorTawny(X) ^ HasDarkSpots(X) ^ HasLongNeck(X) → Giraffe(X)
12. Herbivore(X) ^ HasColorBlackWhite(X) → Zebra(X)
13. Bird(X) ^ CanWalk(X) ^ HasColorBlackWhite(X) ^ HasLongNeck(X) → Ostrich(X)
14. Bird(X) ^ CanSwim(X) ^ HasColorBlackWhite(X) → Penguin(X)
15. Bird(X) ^ CanFly(X) ^ HasColorBlackWhite(X) → Albatross(X)

Facts:
HasHair(X)
HasClaws(X)
HasPointedTeeth(X)
HasForwardEyes(X)
HasColorTawny(X)
HasDarkSpots(X)
¬ Cheetah(X)

Inference by Search Example

Predicate logic

1. ¬ HasHair(X) v Mammal(X)
2. ¬ FeedMilk(X) v Mammal(X)
3. ¬ HasFeather(X) v Bird(X)
4. ¬ CanFly(X) v ¬ LayEgg(X) v Bird(X)
5. ¬ Mammal(X) v ¬ EatMeat(X) v Carnivore(X)
6. ¬ Mammal(X) v ¬ HasPointedTeeth(X) v ¬ HasClaws(X) v ¬ HasForwardEyes(X) v Carnivore(X)
7. ¬ Mammal(X) v ¬ EatGrass(X) v Herbivore(X)
8. ¬ Mammal(X) v ¬ HasHooves(X) v Herbivore(X)
9. ¬ Carnivore(X) v ¬ HasColorTawny(X) v ¬ HasDarkSpots(X) v Cheetah(X)
10. ¬ Carnivore(X) v ¬ HasColorTawny(X) v ¬ HasDarkStripes(X) v Tiger(X)
11. ¬ Herbivore(X) v ¬ HasColorTawny(X) v ¬ HasDarkSpots(X) v ¬ HasLongNeck(X) v Giraffe(X)
12. ¬ Herbivore(X) v ¬ HasColorBlackWhite(X) v Zebra(X)
13. ¬ Bird(X) v ¬ CanWalk(X) v ¬ HasColorBlackWhite(X) v ¬ HasLongNeck(X) v Ostrich(X)
14. ¬ Bird(X) v ¬ CanSwim(X) v ¬ HasColorBlackWhite(X) v Penguin(X)
15. ¬ Bird(X) v ¬ CanFly(X) v ¬ HasColorBlackWhite(X) v Albatross(X)

Facts:
HasHair(X)
HasClaws(X)
HasPointedTeeth(X)
HasForwardEyes(X)
HasColorTawny(X)
HasDarkSpots(X)
¬ Cheetah(X)

Inference by Search Example

Predicate logic

1. ¬ HasHair(X) v Mammal(X)
2. ¬ FeedMilk(X) v Mammal(X)
3. ¬ HasFeather(X) v Bird(X)
4. ¬ CanFly(X) v ¬ LayEgg(X) v Bird(X)
5. ¬ Mammal(X) v ¬ EatMeat(X) v Carnivore(X)
6. ¬ Mammal(X) v ¬ HasPointedTeeth(X) v ¬ HasClaws(X) v ¬ HasForwardEyes(X) v Carnivore(X)
7. ¬ Mammal(X) v ¬ EatGrass(X) v Herbivore(X)
8. ¬ Mammal(X) v ¬ HasHooves(X) v Herbivore(X)
9. ¬ Carnivore(X) v ¬ HasColorTawny(X) v ¬ HasDarkSpots(X) v Cheetah(X) ^ ¬ Cheetah(X)
10. ¬ Carnivore(X) v ¬ HasColorTawny(X) v ¬ HasDarkStripes(X) v Tiger(X)
11. ¬ Herbivore(X) v ¬ HasColorTawny(X) v ¬ HasDarkSpots(X) v ¬ HasLongNeck(X) v Giraffe(X)
12. ¬ Herbivore(X) v ¬ HasColorBlackWhite(X) v Zebra(X)
13. ¬ Bird(X) v ¬ CanWalk(X) v ¬ HasColorBlackWhite(X) v ¬ HasLongNeck(X) v Ostrich(X)
14. ¬ Bird(X) v ¬ CanSwim(X) v ¬ HasColorBlackWhite(X) v Penguin(X)
15. ¬ Bird(X) v ¬ CanFly(X) v ¬ HasColorBlackWhite(X) v Albatross(X)

Facts:
HasHair(X)
HasClaws(X)
HasPointedTeeth(X)
HasForwardEyes(X)
HasColorTawny(X)
HasDarkSpots(X)
¬ Cheetah(X)

Inference by Search Example

Predicate logic

1. ¬ HasHair(X) v Mammal(X)
2. ¬ FeedMilk(X) v Mammal(X)
3. ¬ HasFeather(X) v Bird(X)
4. ¬ CanFly(X) v ¬ LayEgg(X) v Bird(X)
5. ¬ Mammal(X) v ¬ EatMeat(X) v Carnivore(X)
6. ¬ Mammal(X) v ¬ HasPointedTeeth(X) v ¬ HasClaws(X) v ¬ HasForwardEyes(X) v Carnivore(X)
7. ¬ Mammal(X) v ¬ EatGrass(X) v Herbivore(X)
8. ¬ Mammal(X) v ¬ HasHooves(X) v Herbivore(X)
9. ¬ Carnivore(X) v ¬ HasColorTawny(X) v ¬ HasDarkSpots(X) v {}
10. ¬ Carnivore(X) v ¬ HasColorTawny(X) v ¬ HasDarkStripes(X) v Tiger(X)
11. ¬ Herbivore(X) v ¬ HasColorTawny(X) v ¬ HasDarkSpots(X) v ¬ HasLongNeck(X) v Giraffe(X)
12. ¬ Herbivore(X) v ¬ HasColorBlackWhite(X) v Zebra(X)
13. ¬ Bird(X) v ¬ CanWalk(X) v ¬ HasColorBlackWhite(X) v ¬ HasLongNeck(X) v Ostrich(X)
14. ¬ Bird(X) v ¬ CanSwim(X) v ¬ HasColorBlackWhite(X) v Penguin(X)
15. ¬ Bird(X) v ¬ CanFly(X) v ¬ HasColorBlackWhite(X) v Albatross(X)

Facts:
HasHair(X)
HasClaws(X)
HasPointedTeeth(X)
HasForwardEyes(X)
HasColorTawny(X)
HasDarkSpots(X)
¬ Cheetah(X)

Inference by Search Example

Predicate logic

1. ¬ HasHair(X) v Mammal(X)
2. ¬ FeedMilk(X) v Mammal(X)
3. ¬ HasFeather(X) v Bird(X)
4. ¬ CanFly(X) v ¬ LayEgg(X) v Bird(X)
5. ¬ Mammal(X) v ¬ EatMeat(X) v Carnivore(X)
6. ¬ Mammal(X) v ¬ HasPointedTeeth(X) v ¬ HasClaws(X) v ¬ HasForwardEyes(X) v Carnivore(X)
7. ¬ Mammal(X) v ¬ EatGrass(X) v Herbivore(X)
8. ¬ Mammal(X) v ¬ HasHooves(X) v Herbivore(X)
9. ¬ Carnivore(X) v ¬ HasColorTawny(X) v ¬ HasDarkSpots(X)
10. ¬ Carnivore(X) v ¬ HasColorTawny(X) v ¬ HasDarkStripes(X) v Tiger(X)
11. ¬ Herbivore(X) v ¬ HasColorTawny(X) v ¬ HasDarkSpots(X) v ¬ HasLongNeck(X) v Giraffe(X)
12. ¬ Herbivore(X) v ¬ HasColorBlackWhite(X) v Zebra(X)
13. ¬ Bird(X) v ¬ CanWalk(X) v ¬ HasColorBlackWhite(X) v ¬ HasLongNeck(X) v Ostrich(X)
14. ¬ Bird(X) v ¬ CanSwim(X) v ¬ HasColorBlackWhite(X) v Penguin(X)
15. ¬ Bird(X) v ¬ CanFly(X) v ¬ HasColorBlackWhite(X) v Albatross(X)

Facts:
HasHair(X)
HasClaws(X)
HasPointedTeeth(X)
HasForwardEyes(X)
HasColorTawny(X)
HasDarkSpots(X)
¬ Cheetah(X)

Inference by Search Example

Predicate logic

1. ¬ HasHair(X) v Mammal(X)
2. ¬ FeedMilk(X) v Mammal(X)
3. ¬ HasFeather(X) v Bird(X)
4. ¬ CanFly(X) v ¬ LayEgg(X) v Bird(X)
5. ¬ Mammal(X) v ¬ EatMeat(X) v Carnivore(X)
6. ¬ Mammal(X) v ¬ HasPointedTeeth(X) v ¬ HasClaws(X) v ¬ HasForwardEyes(X) v Carnivore(X)
7. ¬ Mammal(X) v ¬ EatGrass(X) v Herbivore(X)
8. ¬ Mammal(X) v ¬ HasHooves(X) v Herbivore(X)
9. ¬ Carnivore(X) v ¬ HasColorTawny(X) v ¬ HasDarkSpots(X) ^ HasDarkSpots(X)
10. ¬ Carnivore(X) v ¬ HasColorTawny(X) v ¬ HasDarkStripes(X) v Tiger(X)
11. ¬ Herbivore(X) v ¬ HasColorTawny(X) v ¬ HasDarkSpots(X) v ¬ HasLongNeck(X) v Giraffe(X)
12. ¬ Herbivore(X) v ¬ HasColorBlackWhite(X) v Zebra(X)
13. ¬ Bird(X) v ¬ CanWalk(X) v ¬ HasColorBlackWhite(X) v ¬ HasLongNeck(X) v Ostrich(X)
14. ¬ Bird(X) v ¬ CanSwim(X) v ¬ HasColorBlackWhite(X) v Penguin(X)
15. ¬ Bird(X) v ¬ CanFly(X) v ¬ HasColorBlackWhite(X) v Albatross(X)

Facts:
HasHair(X)
HasClaws(X)
HasPointedTeeth(X)
HasForwardEyes(X)
HasColorTawny(X)
HasDarkSpots(X)
¬ Cheetah(X)

Inference by Search Example

Predicate logic

1. ¬ HasHair(X) v Mammal(X)
2. ¬ FeedMilk(X) v Mammal(X)
3. ¬ HasFeather(X) v Bird(X)
4. ¬ CanFly(X) v ¬ LayEgg(X) v Bird(X)
5. ¬ Mammal(X) v ¬ EatMeat(X) v Carnivore(X)
6. ¬ Mammal(X) v ¬ HasPointedTeeth(X) v ¬ HasClaws(X) v ¬ HasForwardEyes(X) v Carnivore(X)
7. ¬ Mammal(X) v ¬ EatGrass(X) v Herbivore(X)
8. ¬ Mammal(X) v ¬ HasHooves(X) v Herbivore(X)
9. ¬ Carnivore(X) v ¬ HasColorTawny(X) v {}
10. ¬ Carnivore(X) v ¬ HasColorTawny(X) v ¬ HasDarkStripes(X) v Tiger(X)
11. ¬ Herbivore(X) v ¬ HasColorTawny(X) v ¬ HasDarkSpots(X) v ¬ HasLongNeck(X) v Giraffe(X)
12. ¬ Herbivore(X) v ¬ HasColorBlackWhite(X) v Zebra(X)
13. ¬ Bird(X) v ¬ CanWalk(X) v ¬ HasColorBlackWhite(X) v ¬ HasLongNeck(X) v Ostrich(X)
14. ¬ Bird(X) v ¬ CanSwim(X) v ¬ HasColorBlackWhite(X) v Penguin(X)
15. ¬ Bird(X) v ¬ CanFly(X) v ¬ HasColorBlackWhite(X) v Albatross(X)

Facts:
HasHair(X)
HasClaws(X)
HasPointedTeeth(X)
HasForwardEyes(X)
HasColorTawny(X)
HasDarkSpots(X)
¬ Cheetah(X)

Inference by Search Example

Predicate logic

1. ¬ HasHair(X) v Mammal(X)
2. ¬ FeedMilk(X) v Mammal(X)
3. ¬ HasFeather(X) v Bird(X)
4. ¬ CanFly(X) v ¬ LayEgg(X) v Bird(X)
5. ¬ Mammal(X) v ¬ EatMeat(X) v Carnivore(X)
6. ¬ Mammal(X) v ¬ HasPointedTeeth(X) v ¬ HasClaws(X) v ¬ HasForwardEyes(X) v Carnivore(X)
7. ¬ Mammal(X) v ¬ EatGrass(X) v Herbivore(X)
8. ¬ Mammal(X) v ¬ HasHooves(X) v Herbivore(X)
9. ¬ Carnivore(X) v ¬ HasColorTawny(X) ^ HasColorTawny(X)
10. ¬ Carnivore(X) v ¬ HasColorTawny(X) v ¬ HasDarkStripes(X) v Tiger(X)
11. ¬ Herbivore(X) v ¬ HasColorTawny(X) v ¬ HasDarkSpots(X) v ¬ HasLongNeck(X) v Giraffe(X)
12. ¬ Herbivore(X) v ¬ HasColorBlackWhite(X) v Zebra(X)
13. ¬ Bird(X) v ¬ CanWalk(X) v ¬ HasColorBlackWhite(X) v ¬ HasLongNeck(X) v Ostrich(X)
14. ¬ Bird(X) v ¬ CanSwim(X) v ¬ HasColorBlackWhite(X) v Penguin(X)
15. ¬ Bird(X) v ¬ CanFly(X) v ¬ HasColorBlackWhite(X) v Albatross(X)

Facts:
HasHair(X)
HasClaws(X)
HasPointedTeeth(X)
HasForwardEyes(X)
HasColorTawny(X)
HasDarkSpots(X)
¬ Cheetah(X)

Inference by Search Example

Predicate logic

1. ¬ HasHair(X) v Mammal(X)
2. ¬ FeedMilk(X) v Mammal(X)
3. ¬ HasFeather(X) v Bird(X)
4. ¬ CanFly(X) v ¬ LayEgg(X) v Bird(X)
5. ¬ Mammal(X) v ¬ EatMeat(X) v Carnivore(X)
6. ¬ Mammal(X) v ¬ HasPointedTeeth(X) v ¬ HasClaws(X) v ¬ HasForwardEyes(X) v Carnivore(X)
7. ¬ Mammal(X) v ¬ EatGrass(X) v Herbivore(X)
8. ¬ Mammal(X) v ¬ HasHooves(X) v Herbivore(X)
9. ¬ Carnivore(X) v {}
10. ¬ Carnivore(X) v ¬ HasColorTawny(X) v ¬ HasDarkStripes(X) v Tiger(X)
11. ¬ Herbivore(X) v ¬ HasColorTawny(X) v ¬ HasDarkSpots(X) v ¬ HasLongNeck(X) v Giraffe(X)
12. ¬ Herbivore(X) v ¬ HasColorBlackWhite(X) v Zebra(X)
13. ¬ Bird(X) v ¬ CanWalk(X) v ¬ HasColorBlackWhite(X) v ¬ HasLongNeck(X) v Ostrich(X)
14. ¬ Bird(X) v ¬ CanSwim(X) v ¬ HasColorBlackWhite(X) v Penguin(X)
15. ¬ Bird(X) v ¬ CanFly(X) v ¬ HasColorBlackWhite(X) v Albatross(X)

Facts:
HasHair(X)
HasClaws(X)
HasPointedTeeth(X)
HasForwardEyes(X)
HasColorTawny(X)
HasDarkSpots(X)
¬ Cheetah(X)

Inference by Search Example

Predicate logic

1. ¬ HasHair(X) v Mammal(X)
2. ¬ FeedMilk(X) v Mammal(X)
3. ¬ HasFeather(X) v Bird(X)
4. ¬ CanFly(X) v ¬ LayEgg(X) v Bird(X)
5. ¬ Mammal(X) v ¬ EatMeat(X) v Carnivore(X)
6. ¬ Mammal(X) v ¬ HasPointedTeeth(X) v ¬ HasClaws(X) v ¬ HasForwardEyes(X) v Carnivore(X)
7. ¬ Mammal(X) v ¬ EatGrass(X) v Herbivore(X)
8. ¬ Mammal(X) v ¬ HasHooves(X) v Herbivore(X)
9. ¬ Carnivore(X)
10. ¬ Carnivore(X) v ¬ HasColorTawny(X) v ¬ HasDarkStripes(X) v Tiger(X)
11. ¬ Herbivore(X) v ¬ HasColorTawny(X) v ¬ HasDarkSpots(X) v ¬ HasLongNeck(X) v Giraffe(X)
12. ¬ Herbivore(X) v ¬ HasColorBlackWhite(X) v Zebra(X)
13. ¬ Bird(X) v ¬ CanWalk(X) v ¬ HasColorBlackWhite(X) v ¬ HasLongNeck(X) v Ostrich(X)
14. ¬ Bird(X) v ¬ CanSwim(X) v ¬ HasColorBlackWhite(X) v Penguin(X)
15. ¬ Bird(X) v ¬ CanFly(X) v ¬ HasColorBlackWhite(X) v Albatross(X)

Facts:
HasHair(X)
HasClaws(X)
HasPointedTeeth(X)
HasForwardEyes(X)
HasColorTawny(X)
HasDarkSpots(X)
¬ Cheetah(X)
¬ Carnivore(X)

Inference by Search Example

Predicate logic

1. ¬ HasHair(X) v Mammal(X)
2. ¬ FeedMilk(X) v Mammal(X)
3. ¬ HasFeather(X) v Bird(X)
4. ¬ CanFly(X) v ¬ LayEgg(X) v Bird(X)
5. ¬ Mammal(X) v ¬ EatMeat(X) v Carnivore(X) ^ ¬ Carnivore(X)
6. ¬ Mammal(X) v ¬ HasPointedTeeth(X) v ¬ HasClaws(X) v ¬ HasForwardEyes(X) v Carnivore(X) ^ ¬ Carnivore(X)
7. ¬ Mammal(X) v ¬ EatGrass(X) v Herbivore(X)
8. ¬ Mammal(X) v ¬ HasHooves(X) v Herbivore(X)
9. ¬ Carnivore(X)
10. ¬ Carnivore(X) v ¬ HasColorTawny(X) v ¬ HasDarkStripes(X) v Tiger(X)
11. ¬ Herbivore(X) v ¬ HasColorTawny(X) v ¬ HasDarkSpots(X) v ¬ HasLongNeck(X) v Giraffe(X)
12. ¬ Herbivore(X) v ¬ HasColorBlackWhite(X) v Zebra(X)
13. ¬ Bird(X) v ¬ CanWalk(X) v ¬ HasColorBlackWhite(X) v ¬ HasLongNeck(X) v Ostrich(X)
14. ¬ Bird(X) v ¬ CanSwim(X) v ¬ HasColorBlackWhite(X) v Penguin(X)
15. ¬ Bird(X) v ¬ CanFly(X) v ¬ HasColorBlackWhite(X) v Albatross(X)

Facts:
HasHair(X)
HasClaws(X)
HasPointedTeeth(X)
HasForwardEyes(X)
HasColorTawny(X)
HasDarkSpots(X)
¬ Cheetah(X)
¬ Carnivore(X)

Inference by Search Example

Predicate logic

1. ¬ HasHair(X) v Mammal(X)
2. ¬ FeedMilk(X) v Mammal(X)
3. ¬ HasFeather(X) v Bird(X)
4. ¬ CanFly(X) v ¬ LayEgg(X) v Bird(X)
5. ¬ Mammal(X) v ¬ EatMeat(X) v {} ← branch 2
6. ¬ Mammal(X) v ¬ HasPointedTeeth(X) v ¬ HasClaws(X) v ¬ HasForwardEyes(X) v {} ← branch 1
7. ¬ Mammal(X) v ¬ EatGrass(X) v Herbivore(X)
8. ¬ Mammal(X) v ¬ HasHooves(X) v Herbivore(X)
9. ¬ Carnivore(X)
10. ¬ Carnivore(X) v ¬ HasColorTawny(X) v ¬ HasDarkStripes(X) v Tiger(X)
11. ¬ Herbivore(X) v ¬ HasColorTawny(X) v ¬ HasDarkSpots(X) v ¬ HasLongNeck(X) v Giraffe(X)
12. ¬ Herbivore(X) v ¬ HasColorBlackWhite(X) v Zebra(X)
13. ¬ Bird(X) v ¬ CanWalk(X) v ¬ HasColorBlackWhite(X) v ¬ HasLongNeck(X) v Ostrich(X)
14. ¬ Bird(X) v ¬ CanSwim(X) v ¬ HasColorBlackWhite(X) v Penguin(X)
15. ¬ Bird(X) v ¬ CanFly(X) v ¬ HasColorBlackWhite(X) v Albatross(X)

Facts:
HasHair(X)
HasClaws(X)
HasPointedTeeth(X)
HasForwardEyes(X)
HasColorTawny(X)
HasDarkSpots(X)
¬ Cheetah(X)
¬ Carnivore(X)

Inference by Search Example

Predicate logic

1. ¬ HasHair(X) v Mammal(X)
2. ¬ FeedMilk(X) v Mammal(X)
3. ¬ HasFeather(X) v Bird(X)
4. ¬ CanFly(X) v ¬ LayEgg(X) v Bird(X)
5. ¬ Mammal(X) v ¬ EatMeat(X) v {} ← branch 2
6. ¬ Mammal(X) v ¬ HasPointedTeeth(X) v ¬ HasClaws(X) v ¬ HasForwardEyes(X) v {} ← branch 1
7. ¬ Mammal(X) v ¬ EatGrass(X) v Herbivore(X)
8. ¬ Mammal(X) v ¬ HasHooves(X) v Herbivore(X)
9. ¬ Carnivore(X)
10. ¬ Carnivore(X) v ¬ HasColorTawny(X) v ¬ HasDarkStripes(X) v Tiger(X)
11. ¬ Herbivore(X) v ¬ HasColorTawny(X) v ¬ HasDarkSpots(X) v ¬ HasLongNeck(X) v Giraffe(X)
12. ¬ Herbivore(X) v ¬ HasColorBlackWhite(X) v Zebra(X)
13. ¬ Bird(X) v ¬ CanWalk(X) v ¬ HasColorBlackWhite(X) v ¬ HasLongNeck(X) v Ostrich(X)
14. ¬ Bird(X) v ¬ CanSwim(X) v ¬ HasColorBlackWhite(X) v Penguin(X)
15. ¬ Bird(X) v ¬ CanFly(X) v ¬ HasColorBlackWhite(X) v Albatross(X)

Facts:
HasHair(X)
HasClaws(X)
HasPointedTeeth(X)
HasForwardEyes(X)
HasColorTawny(X)
HasDarkSpots(X)
¬ Cheetah(X)
¬ Carnivore(X)

Inference by Search Example

Predicate logic

1. ¬ HasHair(X) v Mammal(X)
2. ¬ FeedMilk(X) v Mammal(X)
3. ¬ HasFeather(X) v Bird(X)
4. ¬ CanFly(X) v ¬ LayEgg(X) v Bird(X)
5. ¬ Mammal(X) v ¬ EatMeat(X) v {} ← branch 2
6. ¬ Mammal(X) v {} ← branch 1
7. ¬ Mammal(X) v ¬ EatGrass(X) v Herbivore(X)
8. ¬ Mammal(X) v ¬ HasHooves(X) v Herbivore(X)
9. ¬ Carnivore(X)
10. ¬ Carnivore(X) v ¬ HasColorTawny(X) v ¬ HasDarkStripes(X) v Tiger(X)
11. ¬ Herbivore(X) v ¬ HasColorTawny(X) v ¬ HasDarkSpots(X) v ¬ HasLongNeck(X) v Giraffe(X)
12. ¬ Herbivore(X) v ¬ HasColorBlackWhite(X) v Zebra(X)
13. ¬ Bird(X) v ¬ CanWalk(X) v ¬ HasColorBlackWhite(X) v ¬ HasLongNeck(X) v Ostrich(X)
14. ¬ Bird(X) v ¬ CanSwim(X) v ¬ HasColorBlackWhite(X) v Penguin(X)
15. ¬ Bird(X) v ¬ CanFly(X) v ¬ HasColorBlackWhite(X) v Albatross(X)

Which branch to search/check next? Use conflict resolution strategy:

- FIFO?
- Randomly?
- User defined: Less sub-goals?

Facts:
HasHair(X)
HasClaws(X)
HasPointedTeeth(X)
HasForwardEyes(X)
HasColorTawny(X)
HasDarkSpots(X)
¬ Cheetah(X)
¬ Carnivore(X)

Inference by Search Example

Predicate logic

1. ¬ HasHair(X) v Mammal(X)
2. ¬ FeedMilk(X) v Mammal(X)
3. ¬ HasFeather(X) v Bird(X)
4. ¬ CanFly(X) v ¬ LayEgg(X) v Bird(X)
5. ¬ Mammal(X) v ¬ EatMeat(X) v {} ← branch 2
6. ¬ Mammal(X) ← branch 1
7. ¬ Mammal(X) v ¬ EatGrass(X) v Herbivore(X)
8. ¬ Mammal(X) v ¬ HasHooves(X) v Herbivore(X)
9. ¬ Carnivore(X)
10. ¬ Carnivore(X) v ¬ HasColorTawny(X) v ¬ HasDarkStripes(X) v Tiger(X)
11. ¬ Herbivore(X) v ¬ HasColorTawny(X) v ¬ HasDarkSpots(X) v ¬ HasLongNeck(X) v Giraffe(X)
12. ¬ Herbivore(X) v ¬ HasColorBlackWhite(X) v Zebra(X)
13. ¬ Bird(X) v ¬ CanWalk(X) v ¬ HasColorBlackWhite(X) v ¬ HasLongNeck(X) v Ostrich(X)
14. ¬ Bird(X) v ¬ CanSwim(X) v ¬ HasColorBlackWhite(X) v Penguin(X)
15. ¬ Bird(X) v ¬ CanFly(X) v ¬ HasColorBlackWhite(X) v Albatross(X)

Facts:
HasHair(X)
HasClaws(X)
HasPointedTeeth(X)
HasForwardEyes(X)
HasColorTawny(X)
HasDarkSpots(X)
¬ Cheetah(X)
¬ Carnivore(X)
¬ Mammal(X)

Inference by Search Example

Predicate logic

1. ¬ HasHair(X) v Mammal(X) ^ ¬ Mammal(X) ← branch 1.1
2. ¬ FeedMilk(X) v Mammal(X) ^ ¬ Mammal(X) ← branch 1.2
3. ¬ HasFeather(X) v Bird(X)
4. ¬ CanFly(X) v ¬ LayEgg(X) v Bird(X)
5. ¬ Mammal(X) v ¬ EatMeat(X) v {} ← branch 2
6. ¬ Mammal(X) ← branch 1
7. ¬ Mammal(X) v ¬ EatGrass(X) v Herbivore(X)
8. ¬ Mammal(X) v ¬ HasHooves(X) v Herbivore(X)
9. ¬ Carnivore(X)
10. ¬ Carnivore(X) v ¬ HasColorTawny(X) v ¬ HasDarkStripes(X) v Tiger(X)
11. ¬ Herbivore(X) v ¬ HasColorTawny(X) v ¬ HasDarkSpots(X) v ¬ HasLongNeck(X) v Giraffe(X)
12. ¬ Herbivore(X) v ¬ HasColorBlackWhite(X) v Zebra(X)
13. ¬ Bird(X) v ¬ CanWalk(X) v ¬ HasColorBlackWhite(X) v ¬ HasLongNeck(X) v Ostrich(X)
14. ¬ Bird(X) v ¬ CanSwim(X) v ¬ HasColorBlackWhite(X) v Penguin(X)
15. ¬ Bird(X) v ¬ CanFly(X) v ¬ HasColorBlackWhite(X) v Albatross(X)

Facts:
HasHair(X)
HasClaws(X)
HasPointedTeeth(X)
HasForwardEyes(X)
HasColorTawny(X)
HasDarkSpots(X)
¬ Cheetah(X)
¬ Carnivore(X)
¬ Mammal(X)

Inference by Search Example

Predicate logic

1. ¬ HasHair(X) v {} ← branch 1.1
2. ¬ FeedMilk(X) v {} ← branch 1.2
3. ¬ HasFeather(X) v Bird(X)
4. ¬ CanFly(X) v ¬ LayEgg(X) v Bird(X)
5. ¬ Mammal(X) v ¬ EatMeat(X) v {} ← branch 2
6. ¬ Mammal(X) ← branch 1
7. ¬ Mammal(X) v ¬ EatGrass(X) v Herbivore(X)
8. ¬ Mammal(X) v ¬ HasHooves(X) v Herbivore(X)
9. ¬ Carnivore(X)
10. ¬ Carnivore(X) v ¬ HasColorTawny(X) v ¬ HasDarkStripes(X) v Tiger(X)
11. ¬ Herbivore(X) v ¬ HasColorTawny(X) v ¬ HasDarkSpots(X) v ¬ HasLongNeck(X) v Giraffe(X)
12. ¬ Herbivore(X) v ¬ HasColorBlackWhite(X) v Zebra(X)
13. ¬ Bird(X) v ¬ CanWalk(X) v ¬ HasColorBlackWhite(X) v ¬ HasLongNeck(X) v Ostrich(X)
14. ¬ Bird(X) v ¬ CanSwim(X) v ¬ HasColorBlackWhite(X) v Penguin(X)
15. ¬ Bird(X) v ¬ CanFly(X) v ¬ HasColorBlackWhite(X) v Albatross(X)

Facts:
HasHair(X)
HasClaws(X)
HasPointedTeeth(X)
HasForwardEyes(X)
HasColorTawny(X)
HasDarkSpots(X)
¬ Cheetah(X)
¬ Carnivore(X)
¬ Mammal(X)

Inference by Search Example

Predicate logic

1. ¬ HasHair(X) ^ HasHair(X) ← branch 1.1
2. ¬ FeedMilk(X) v {} ← branch 1.2
3. ¬ HasFeather(X) v Bird(X)
4. ¬ CanFly(X) v ¬ LayEgg(X) v Bird(X)
5. ¬ Mammal(X) v ¬ EatMeat(X) v {} ← branch 2
6. ¬ Mammal(X) ← branch 1
7. ¬ Mammal(X) v ¬ EatGrass(X) v Herbivore(X)
8. ¬ Mammal(X) v ¬ HasHooves(X) v Herbivore(X)
9. ¬ Carnivore(X)
10. ¬ Carnivore(X) v ¬ HasColorTawny(X) v ¬ HasDarkStripes(X) v Tiger(X)
11. ¬ Herbivore(X) v ¬ HasColorTawny(X) v ¬ HasDarkSpots(X) v ¬ HasLongNeck(X) v Giraffe(X)
12. ¬ Herbivore(X) v ¬ HasColorBlackWhite(X) v Zebra(X)
13. ¬ Bird(X) v ¬ CanWalk(X) v ¬ HasColorBlackWhite(X) v ¬ HasLongNeck(X) v Ostrich(X)
14. ¬ Bird(X) v ¬ CanSwim(X) v ¬ HasColorBlackWhite(X) v Penguin(X)
15. ¬ Bird(X) v ¬ CanFly(X) v ¬ HasColorBlackWhite(X) v Albatross(X)

Facts:
HasHair(X)
HasClaws(X)
HasPointedTeeth(X)
HasForwardEyes(X)
HasColorTawny(X)
HasDarkSpots(X)
¬ Cheetah(X)
¬ Carnivore(X)
¬ Mammal(X)

Inference by Search Example

Predicate logic

1. {} ← branch 1.1
2. ¬ FeedMilk(X) v {} ← branch 1.2
3. ¬ HasFeather(X) v Bird(X)
4. ¬ CanFly(X) v ¬ LayEgg(X) v Bird(X)
5. ¬ Mammal(X) v ¬ EatMeat(X) v {} ← branch 2
6. ¬ Mammal(X) ← branch 1
7. ¬ Mammal(X) v ¬ EatGrass(X) v Herbivore(X)
8. ¬ Mammal(X) v ¬ HasHooves(X) v Herbivore(X)
9. ¬ Carnivore(X)
10. ¬ Carnivore(X) v ¬ HasColorTawny(X) v ¬ HasDarkStripes(X) v Tiger(X)
11. ¬ Herbivore(X) v ¬ HasColorTawny(X) v ¬ HasDarkSpots(X) v ¬ HasLongNeck(X) v Giraffe(X)
12. ¬ Herbivore(X) v ¬ HasColorBlackWhite(X) v Zebra(X)
13. ¬ Bird(X) v ¬ CanWalk(X) v ¬ HasColorBlackWhite(X) v ¬ HasLongNeck(X) v Ostrich(X)
14. ¬ Bird(X) v ¬ CanSwim(X) v ¬ HasColorBlackWhite(X) v Penguin(X)
15. ¬ Bird(X) v ¬ CanFly(X) v ¬ HasColorBlackWhite(X) v Albatross(X)

Facts:
HasHair(X)
HasClaws(X)
HasPointedTeeth(X)
HasForwardEyes(X)
HasColorTawny(X)
HasDarkSpots(X)
¬ Cheetah(X)
¬ Carnivore(X)
¬ Mammal(X)

Inference by Search Example

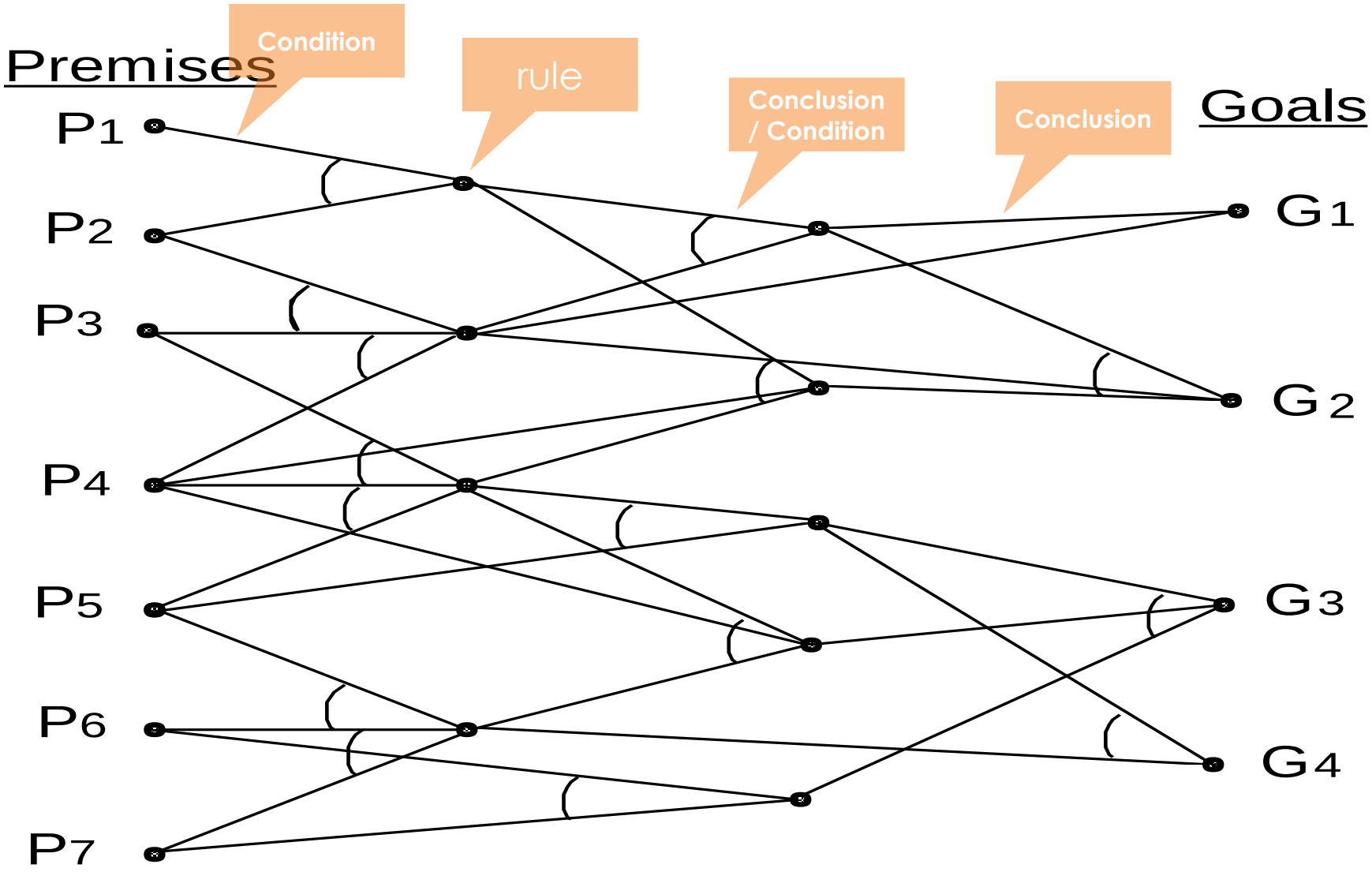
Predicate logic

1. {} ← a = Cheetah(X) proved by refutation ← branch 1.1
2. ¬ FeedMilk(X) v {} ← branch 1.2
3. ¬ HasFeather(X) v Bird(X)
4. ¬ CanFly(X) v ¬ LayEgg(X) v Bird(X)
5. ¬ Mammal(X) v ¬ EatMeat(X) v {} ← branch 2
6. ¬ Mammal(X) ← branch 1
7. ¬ Mammal(X) v ¬ EatGrass(X) v Herbivore(X)
8. ¬ Mammal(X) v ¬ HasHooves(X) v Herbivore(X)
9. ¬ Carnivore(X)
10. ¬ Carnivore(X) v ¬ HasColorTawny(X) v ¬ HasDarkStripes(X) v Tiger(X)
11. ¬ Herbivore(X) v ¬ HasColorTawny(X) v ¬ HasDarkSpots(X) v ¬ HasLongNeck(X) v Giraffe(X)
12. ¬ Herbivore(X) v ¬ HasColorBlackWhite(X) v Zebra(X)
13. ¬ Bird(X) v ¬ CanWalk(X) v ¬ HasColorBlackWhite(X) v ¬ HasLongNeck(X) v Ostrich(X)
14. ¬ Bird(X) v ¬ CanSwim(X) v ¬ HasColorBlackWhite(X) v Penguin(X)
15. ¬ Bird(X) v ¬ CanFly(X) v ¬ HasColorBlackWhite(X) v Albatross(X)

Facts:
HasHair(X)
HasClaws(X)
HasPointedTeeth(X)
HasForwardEyes(X)
HasColorTawny(X)
HasDarkSpots(X)
¬ Cheetah(X)
¬ Carnivore(X)
¬ Mammal(X)

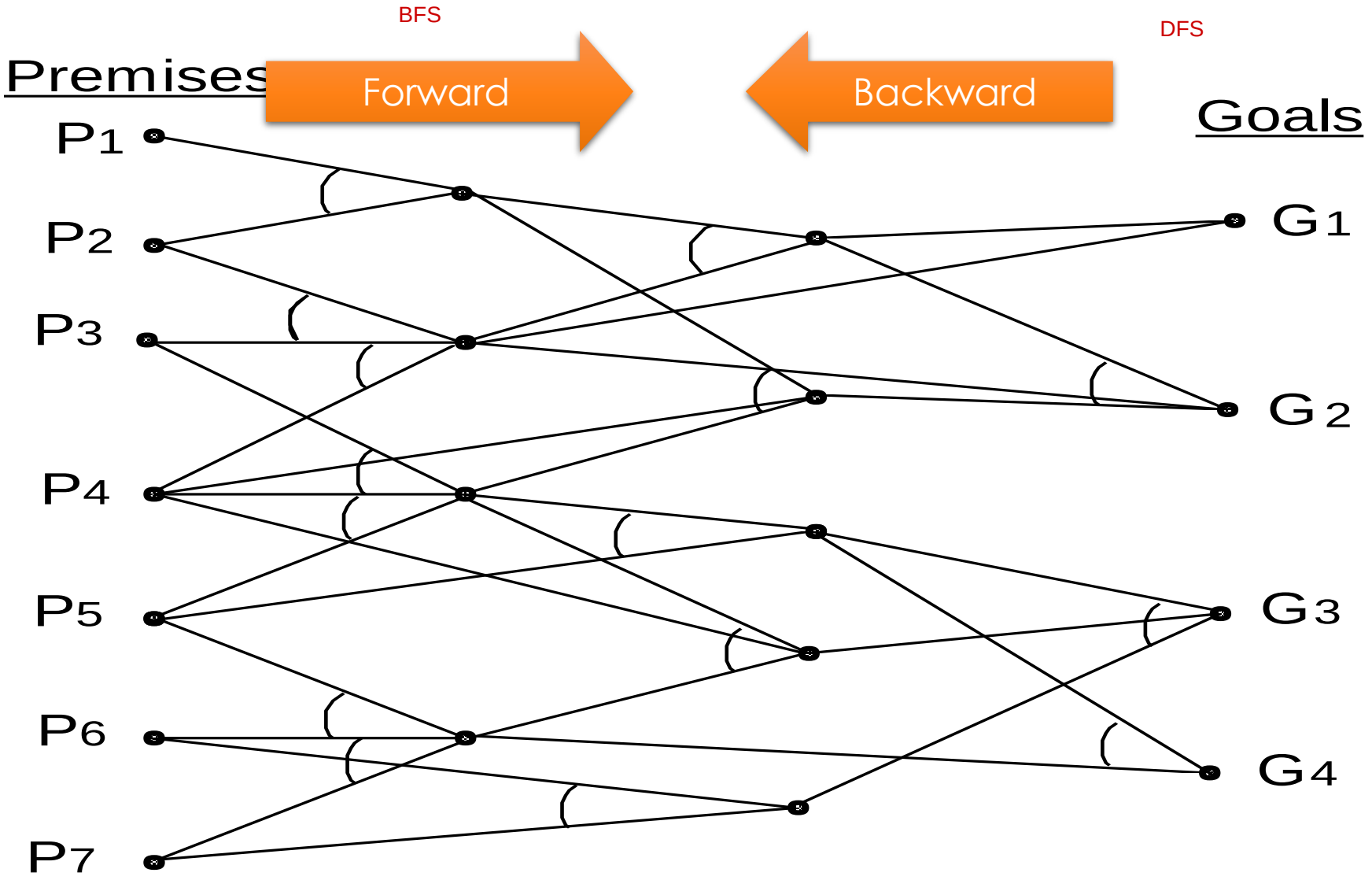
Inference by Search Example

Knowledge base (rules) in tree representation



Inference by Search Example

Forward Chaining (BFS) vs. Backward Chaining (DFS)



1.2 Reasoning Using Uninformed Search

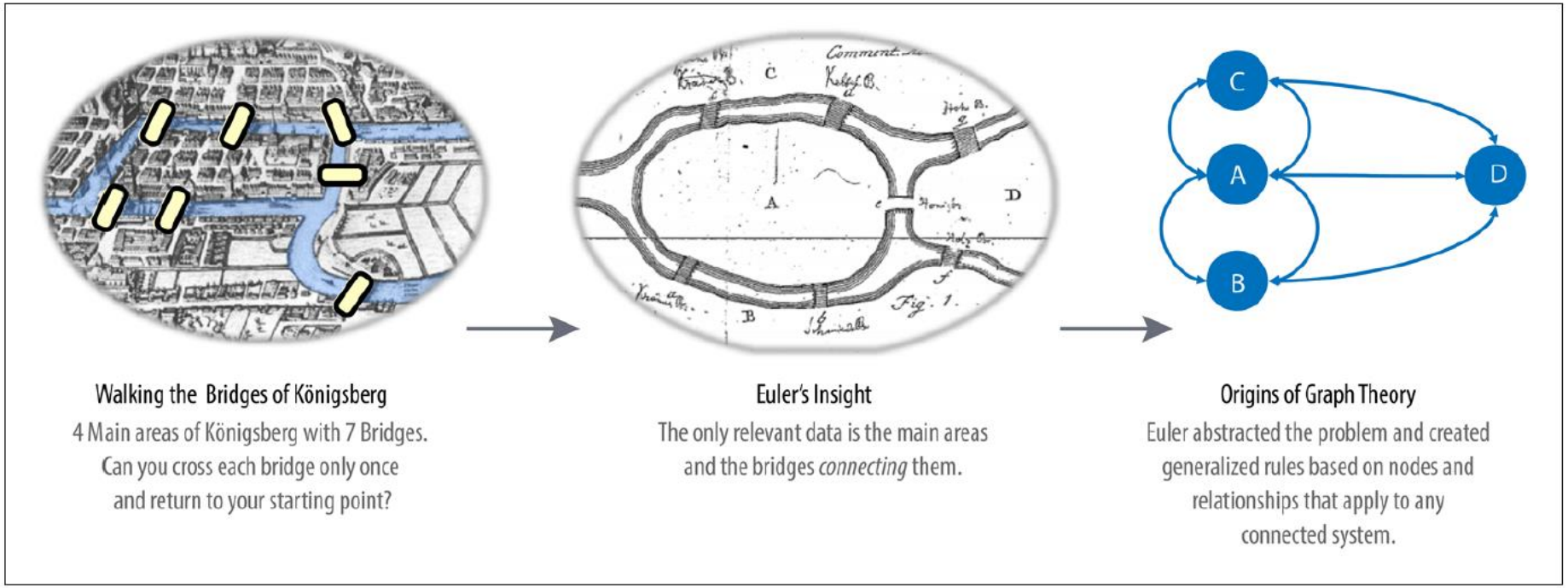
1.2.1 Uninformed Search Techniques (Tree)

1.2.2 Uninformed Search Techniques (Graph)

1.2.3 Exercise

Uninformed Search Techniques (Graph)

Graphs dating back to 1736



Leonhard Euler

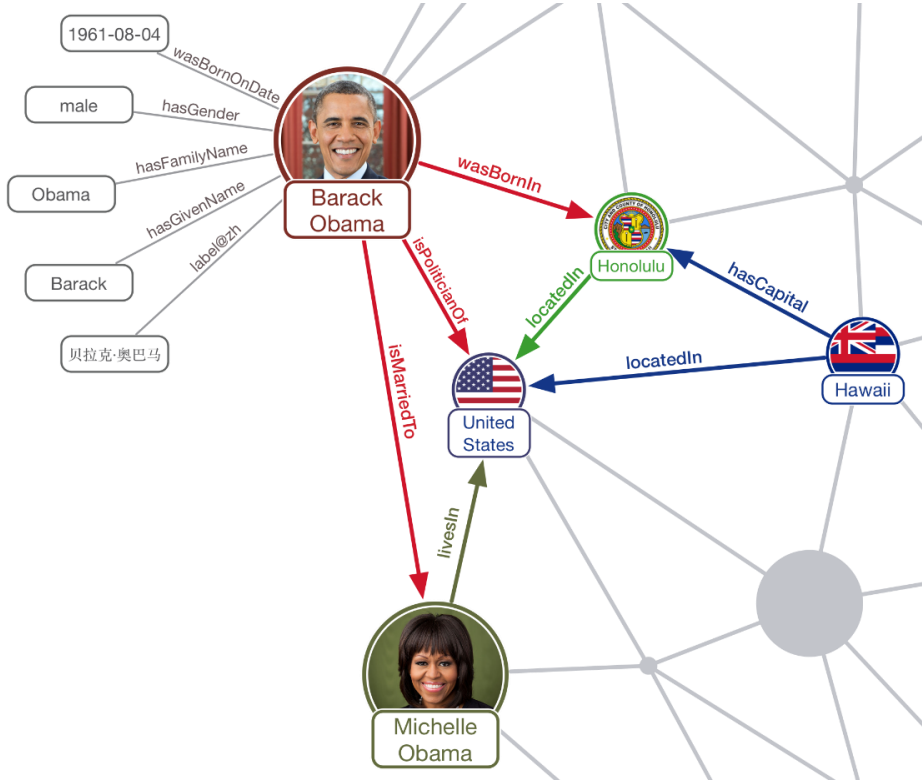
Figure 1-1. The origins of graph theory. The city of Königsberg included two large islands connected to each other and the two mainland portions of the city by seven bridges. The puzzle was to create a walk through the city, crossing each bridge once and only once.

Source: Graph Algorithms: Practical Examples in Apache Spark and Neo4j
Book by Amy E. Hodler and Mark Needham

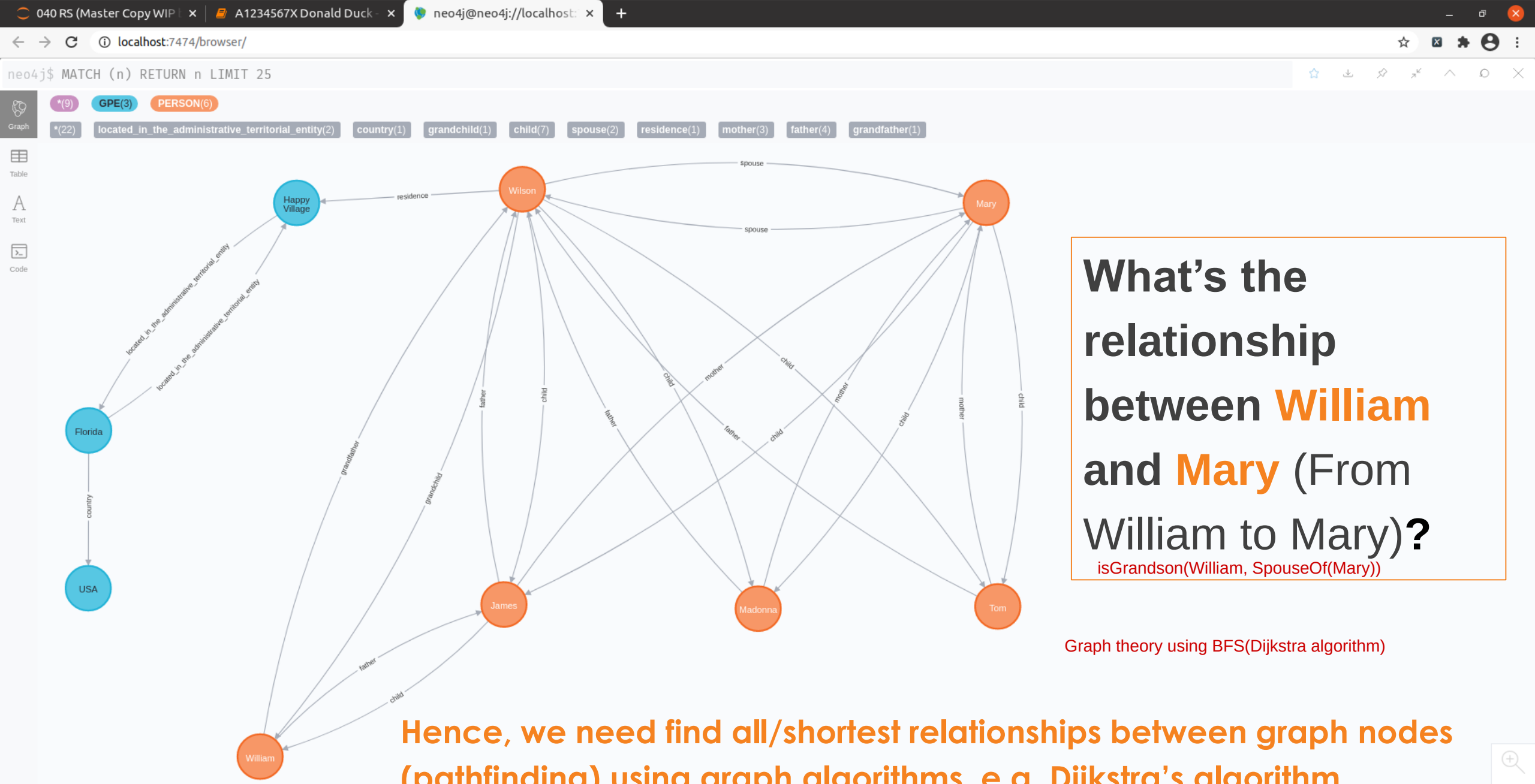
Uninformed Search Techniques (Graph)

Knowledge Graph / Semantic Web / Ontology

- **A model (data structure) for word concepts in human cognition, consisting of nodes, links and labels**
  - **Nodes (vertices)** represent objects, concepts, or situations. They can be instances (individual objects as in Knowledge Graph) or classes (generic objects as in Semantic Web)
  - **Links (edges)** between nodes represent a relationship/predicate
  - **Labels (attributes)**
    - Labels on nodes indicate the name of the object, concept, etc.
    - Labels on links describe the type of relationship between nodes
- **Search objective: Find out the relationship between Barack and Hawaii.**



Source: https://www.ambiverse.com/wp-content/uploads/2017/01/KnowledgeGraph-Relations-cropped2.png



What's the relationship between William and Mary (From William to Mary)**?**

isGrandson(William, SpouseOf(Mary))

Graph theory using BFS(Dijkstra algorithm)

Hence, we need find all/shortest relationships between graph nodes (pathfinding) using graph algorithms, e.g. Dijkstra's algorithm.

Uninformed Search Techniques (Graph)

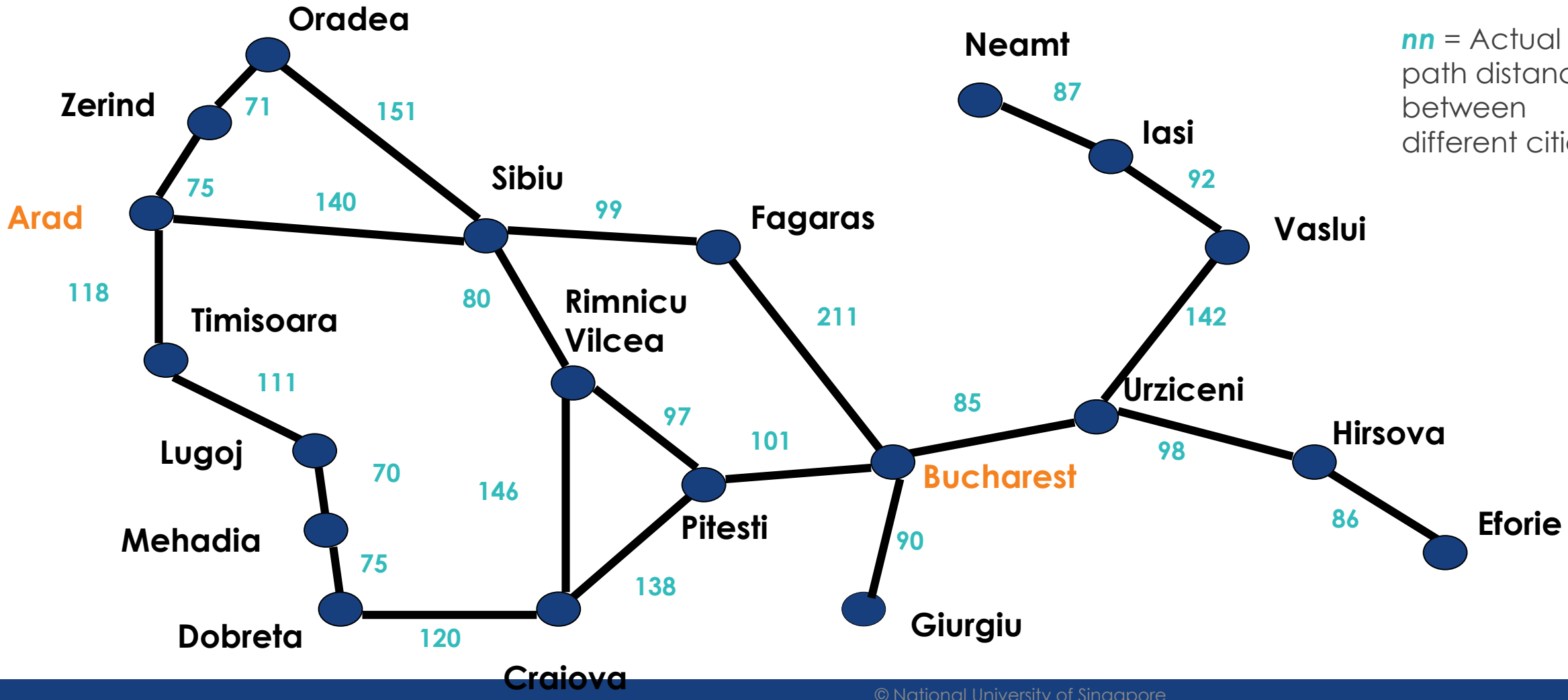
Dijkstra's algorithm conducts a breadth-first search (BFS) with a higher level of analysis (calculation of path cost) in order to find the shortest path between two nodes in a graph. Here's how it works:

1. Pick the **start** and **end nodes** and add the start node to the set of **solved nodes** with a value of **0**. Solved nodes are the set of nodes with the shortest known path from the start node. The start node has a value of 0 because it is 0 path-length away from itself.
2. Traverse breadth-first from the start node to its **nearest neighbors (frontier nodes)** and record/calculate the path length against each neighboring node.
3. Pick the **shortest path to one of these neighbors** and mark that node as solved. In case of a tie, Dijkstra's algorithm picks at random.
4. Visit the nearest neighbors (frontier nodes) to the **set of solved nodes** and record/accumulate the path lengths from the start node against these new neighbors. Don't visit any neighboring nodes that have already been solved, as we already know the shortest paths to them.
5. Repeat steps three and four until the **end/destination node** has been marked as "**solved node**", or a termination criteria is satisfied.

: https://neo4j.com/blog/graph-search-algorithm-basics/

Uninformed Search Techniques (Graph)

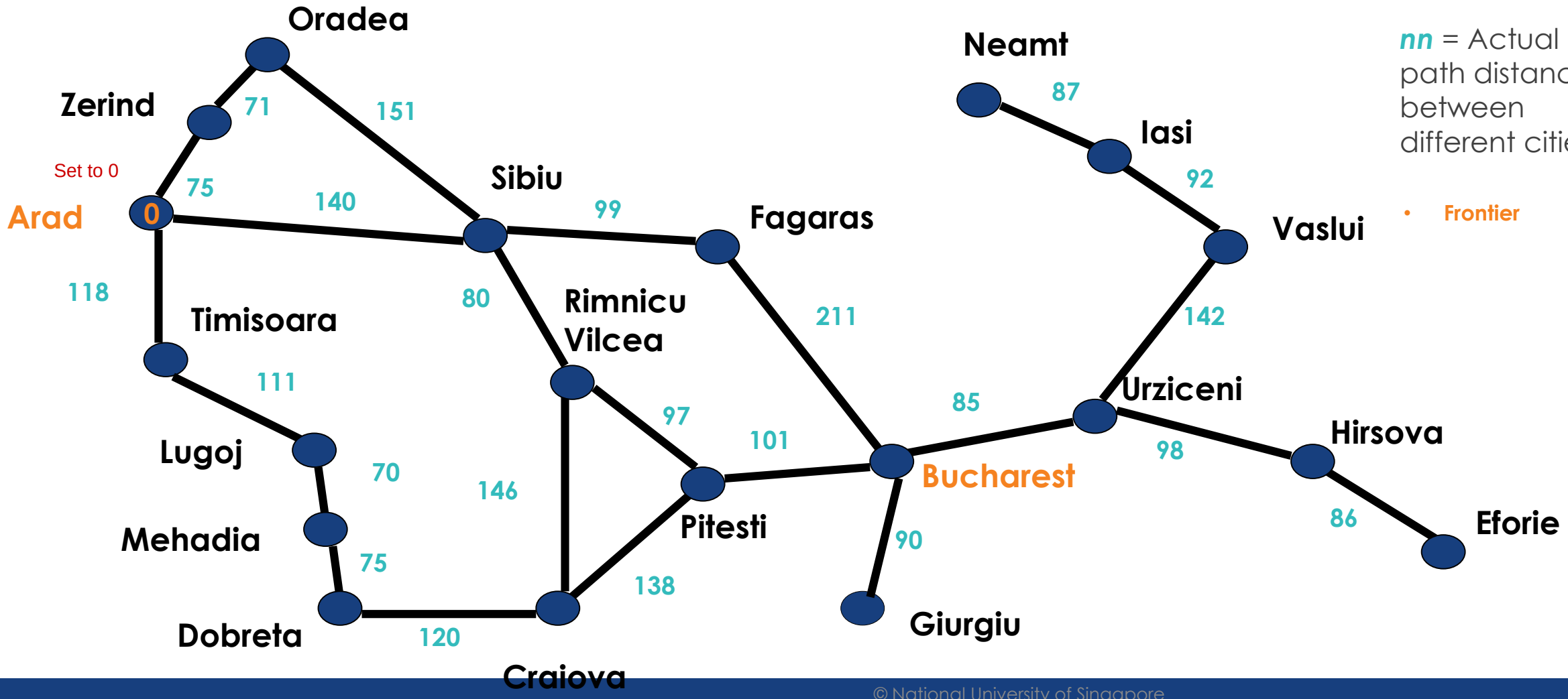
Goal: find shortest path from Arad to Bucharest.



nn = Actual path distance between different cities

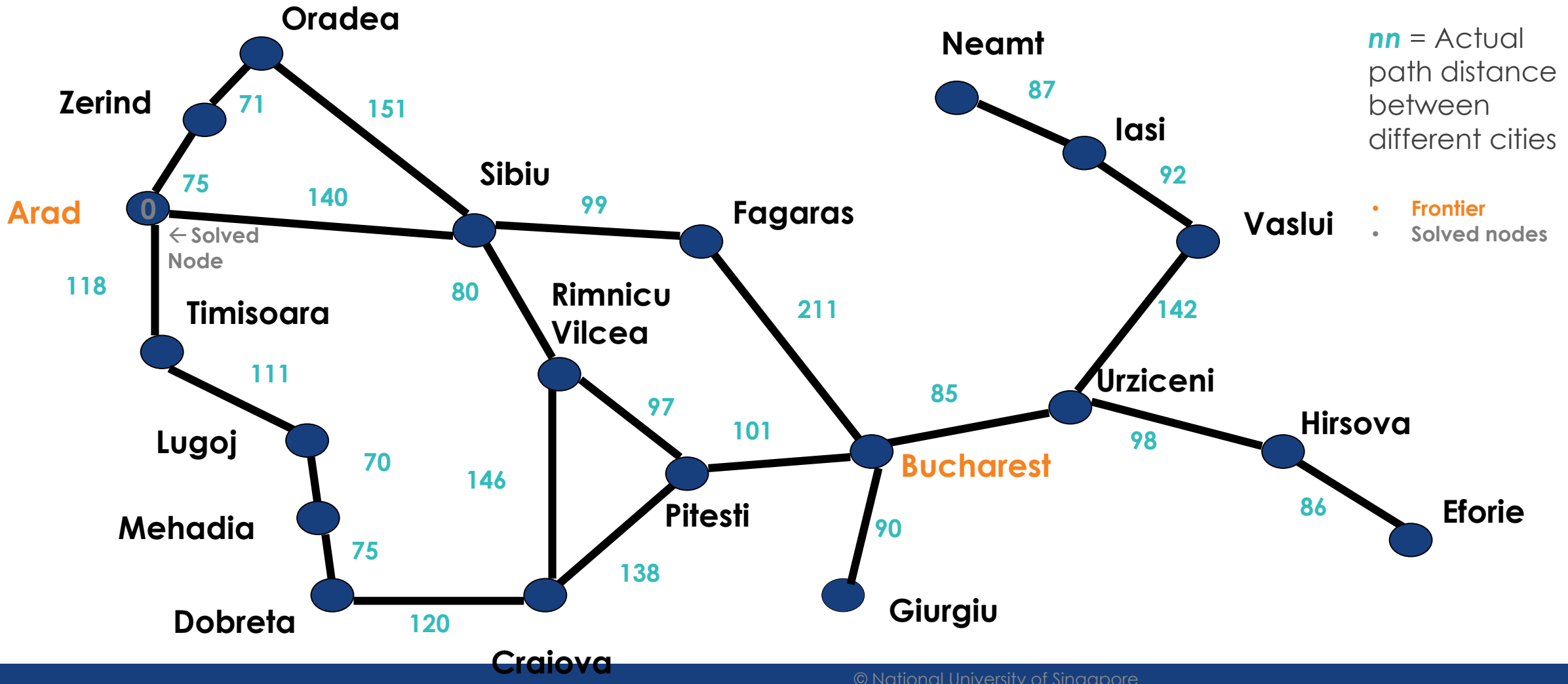
Uninformed Search Techniques (Graph)

Initialize Start Node (Frontier Node**): Arad; Shortest distance: Zero;**



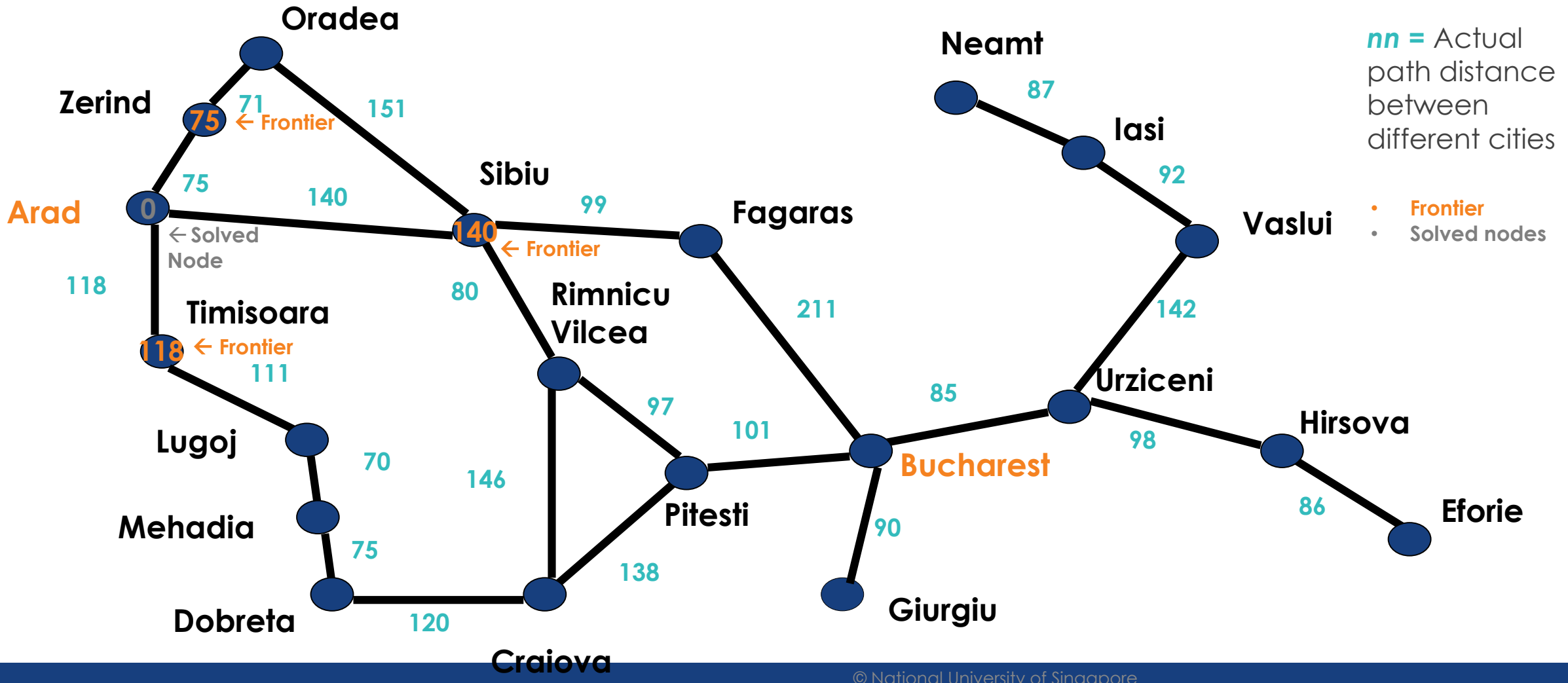
Uninformed Search Techniques (Graph)

Update list of Solved Nodes;



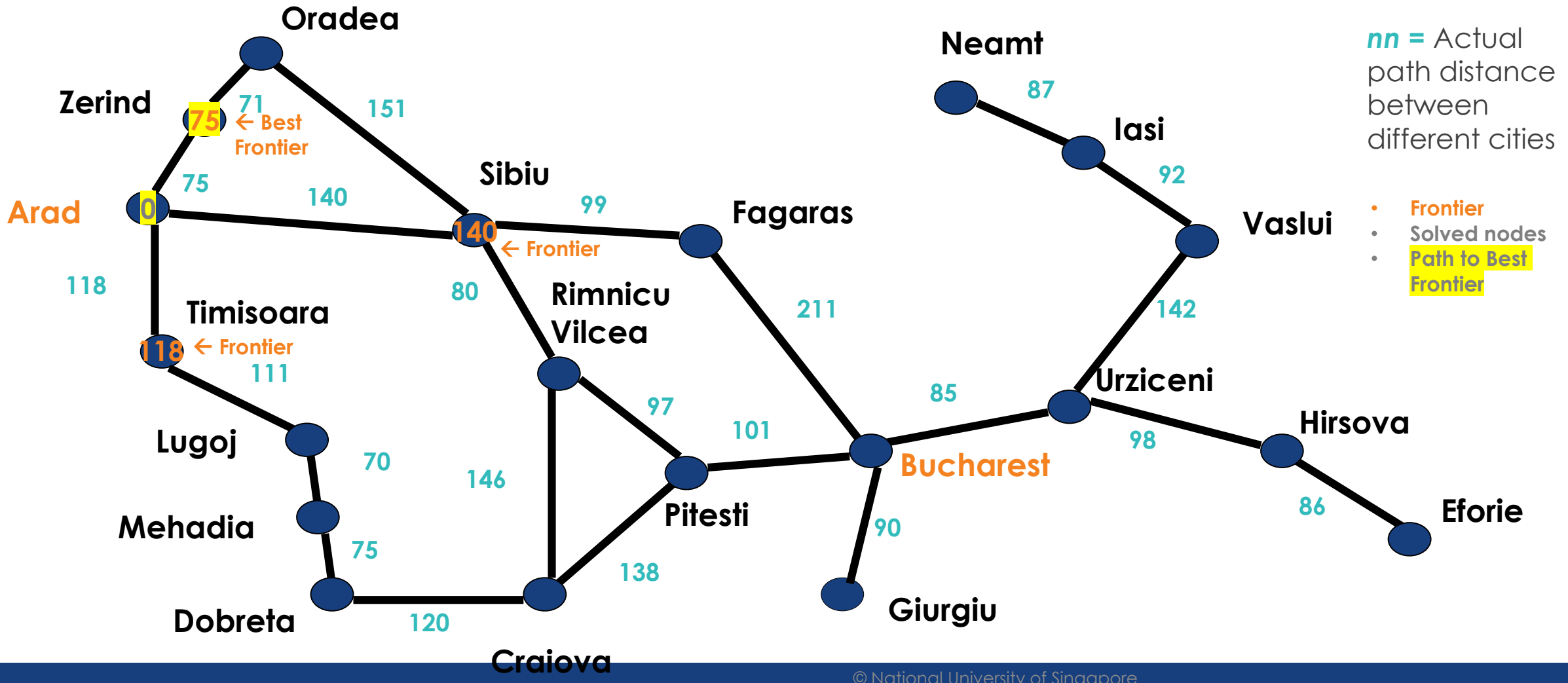
Uninformed Search Techniques (Graph)

Expand and calculate Frontier Nodes;



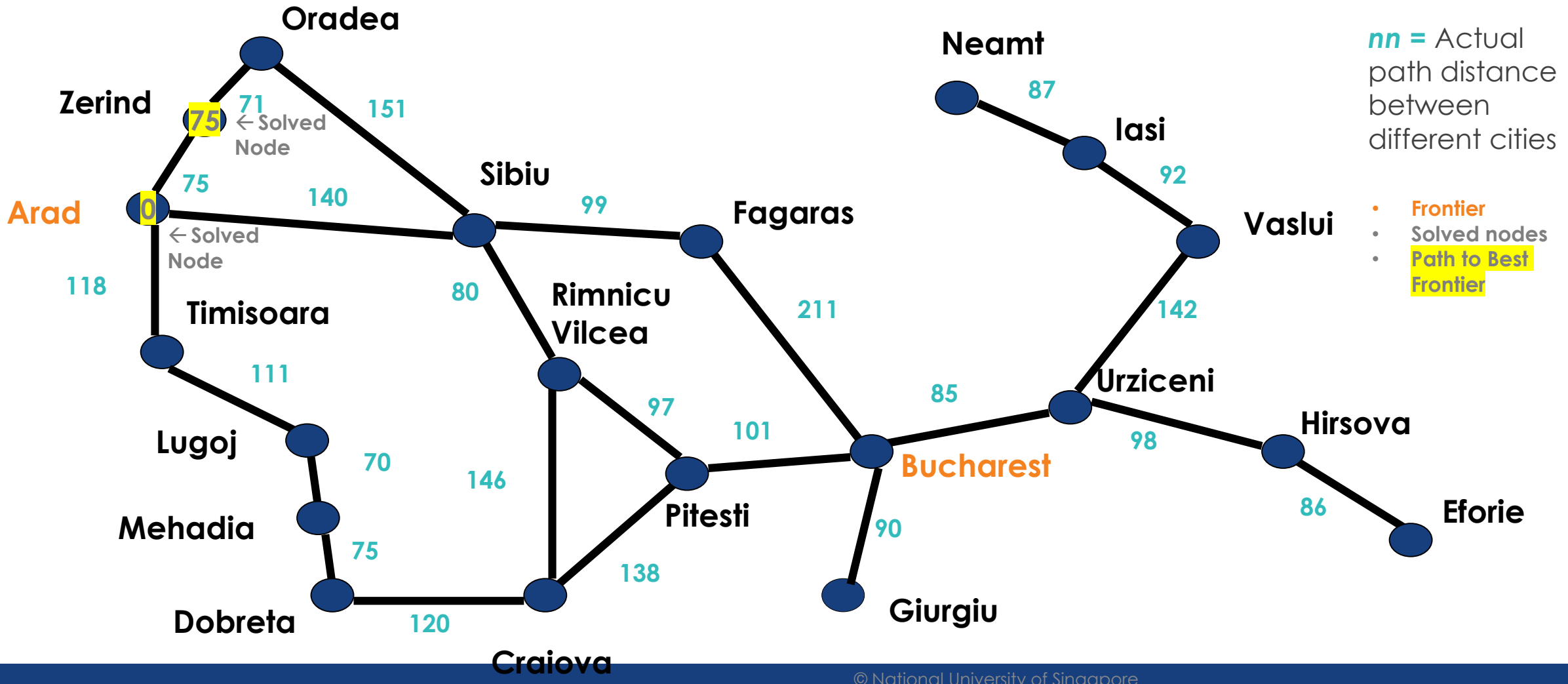
Uninformed Search Techniques (Graph)

Determine Best Frontier Node;



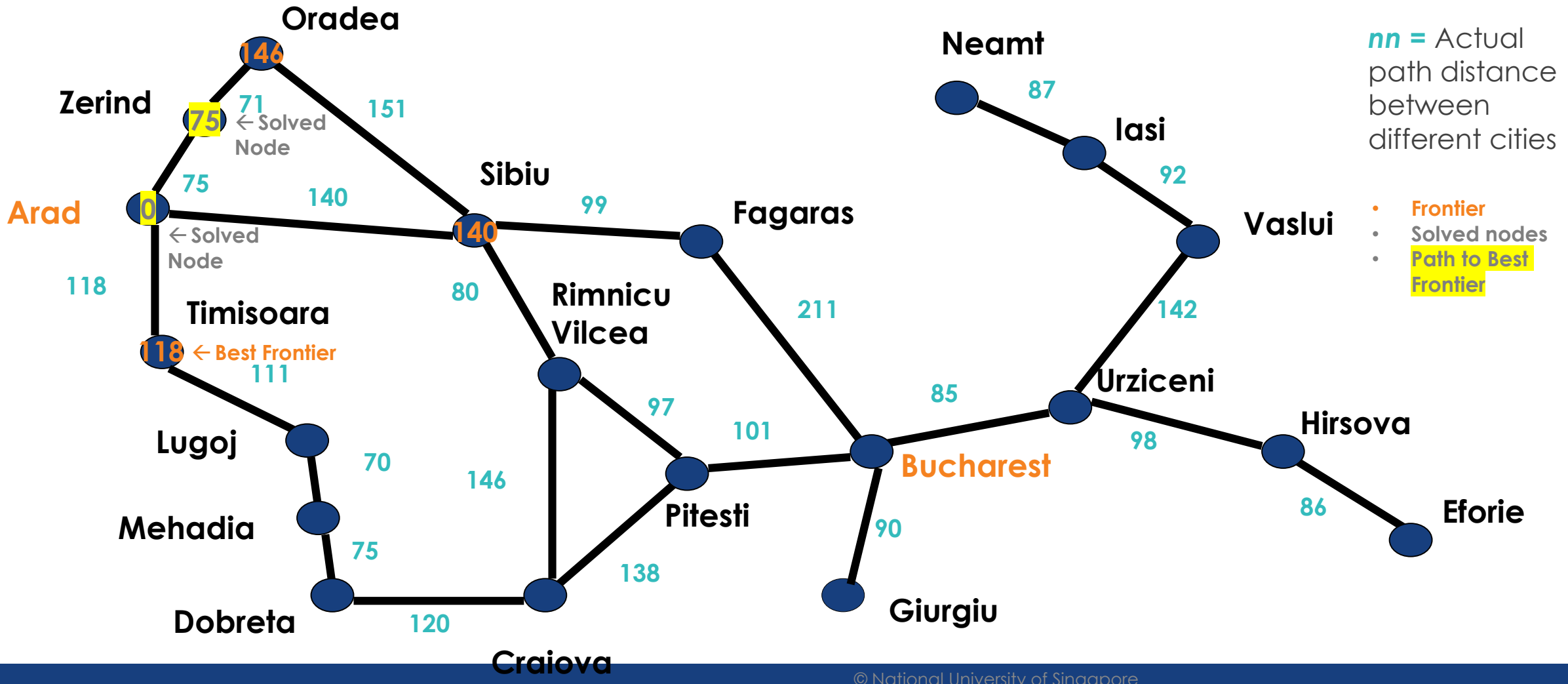
Uninformed Search Techniques (Graph)

Update list of Solved Nodes;



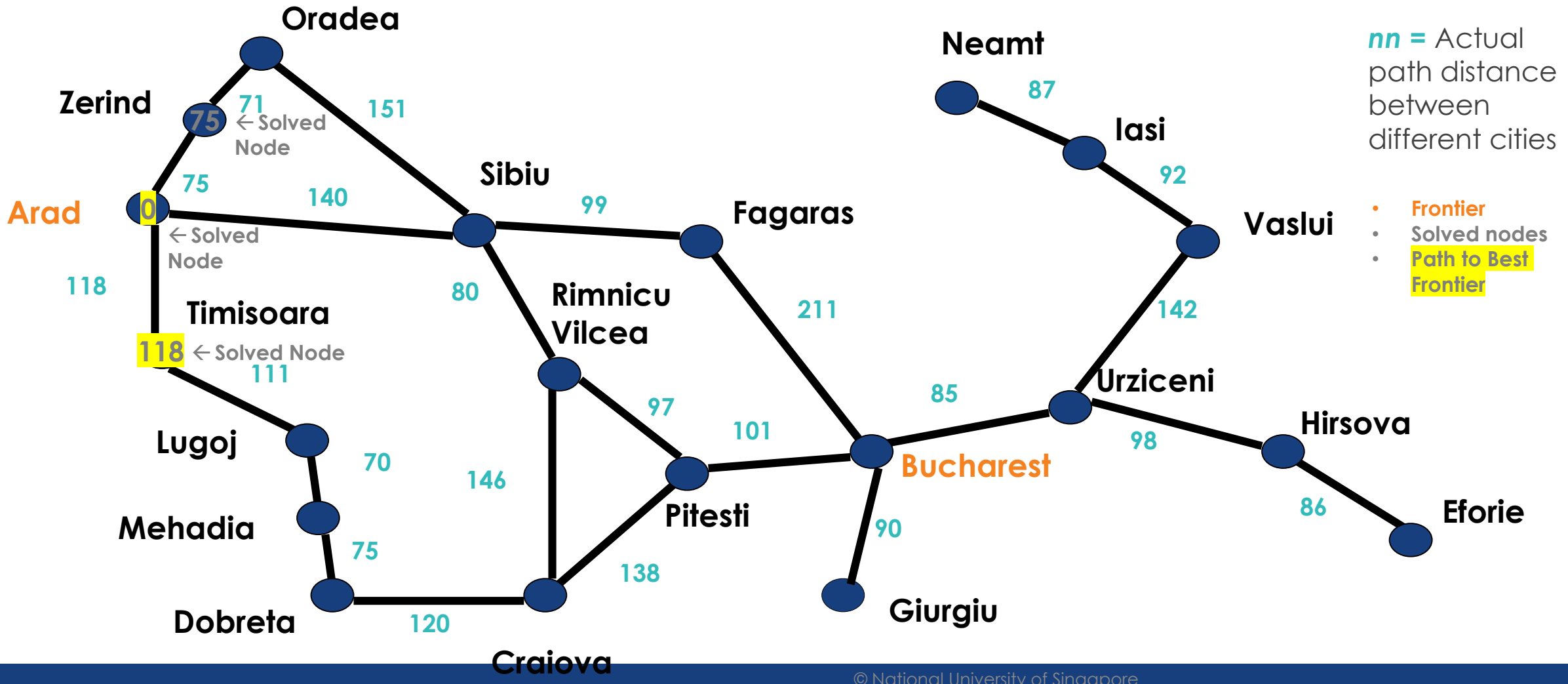
Uninformed Search Techniques (Graph)

Expand and calculate Frontier Nodes; Determine Best Frontier Node;



Uninformed Search Techniques (Graph)

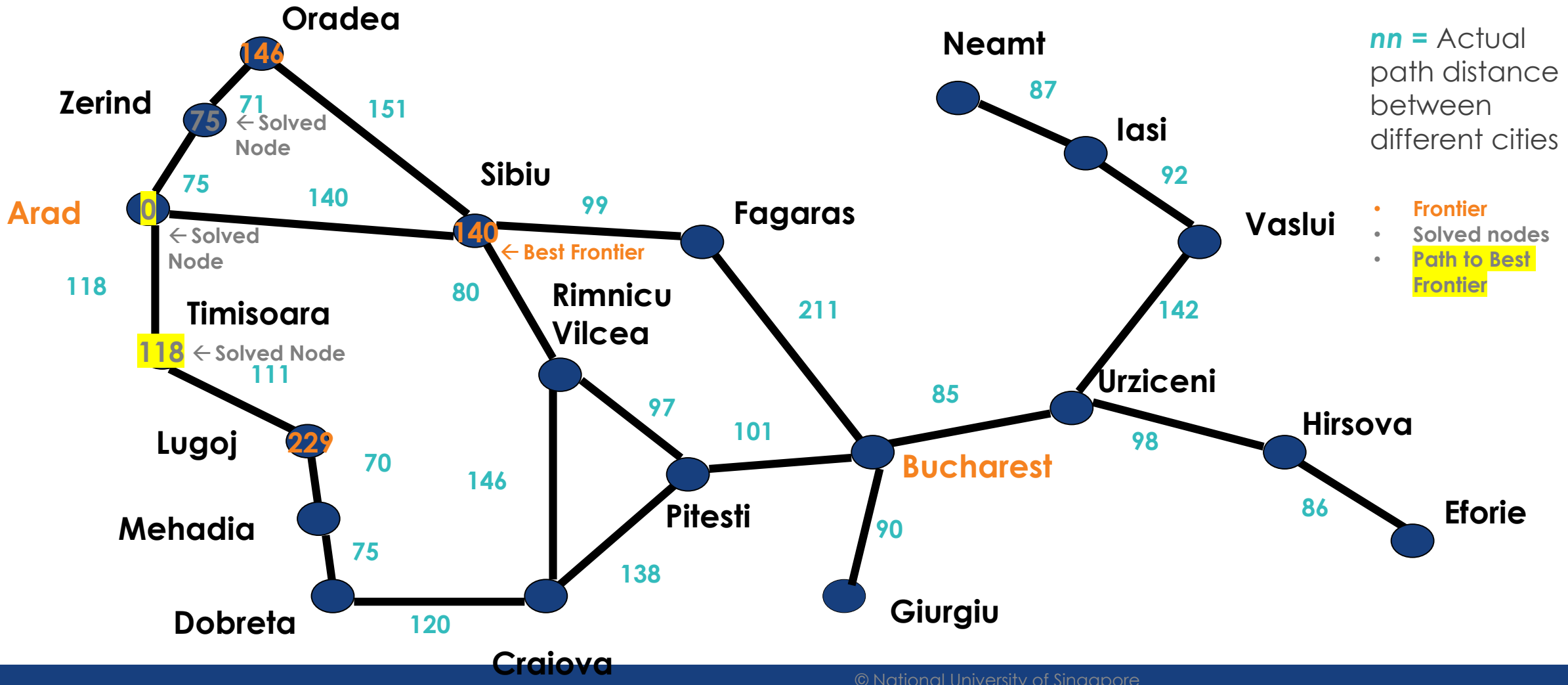
Update list of Solved Nodes;



Uninformed Search Techniques (Graph)

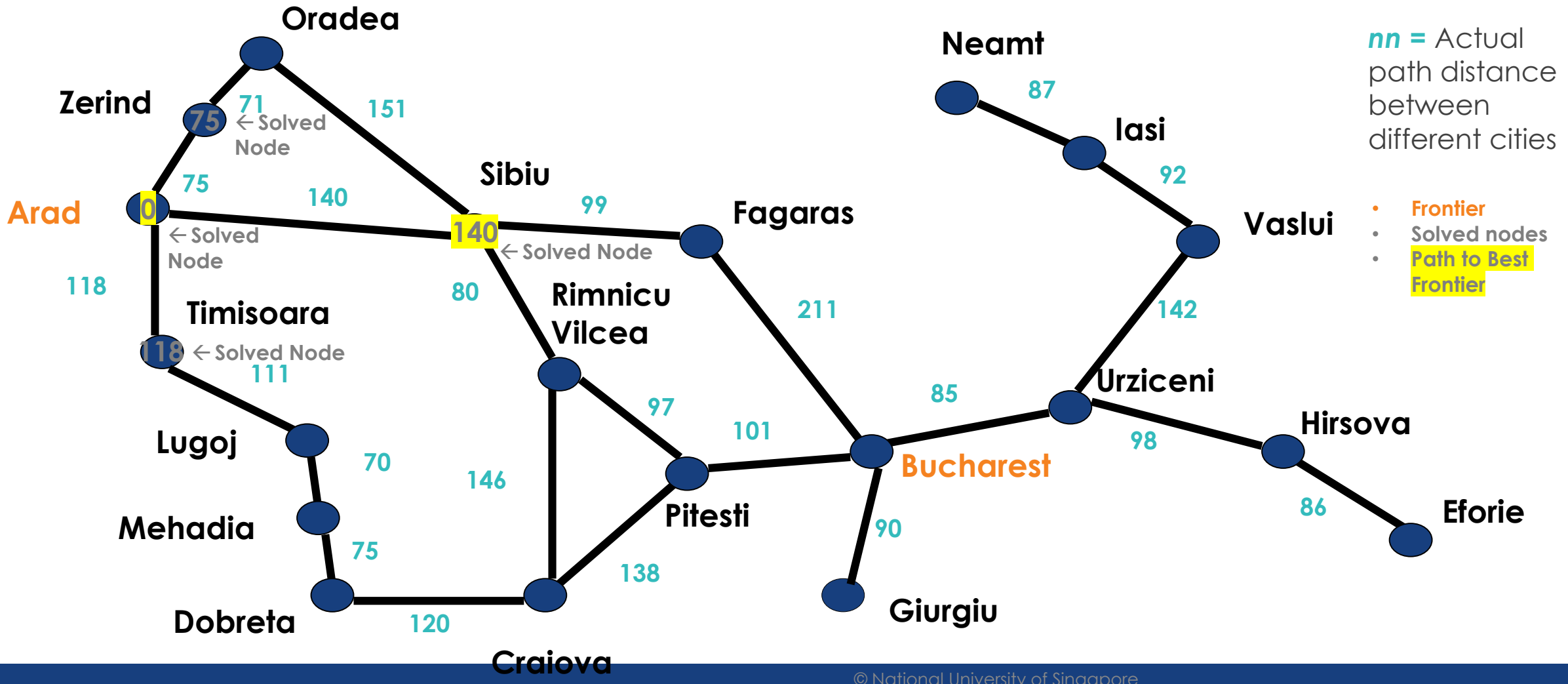
Graph theory typically do BFS because DFS may have loops.

Expand and calculate Frontier Nodes; Determine Best Frontier Node;



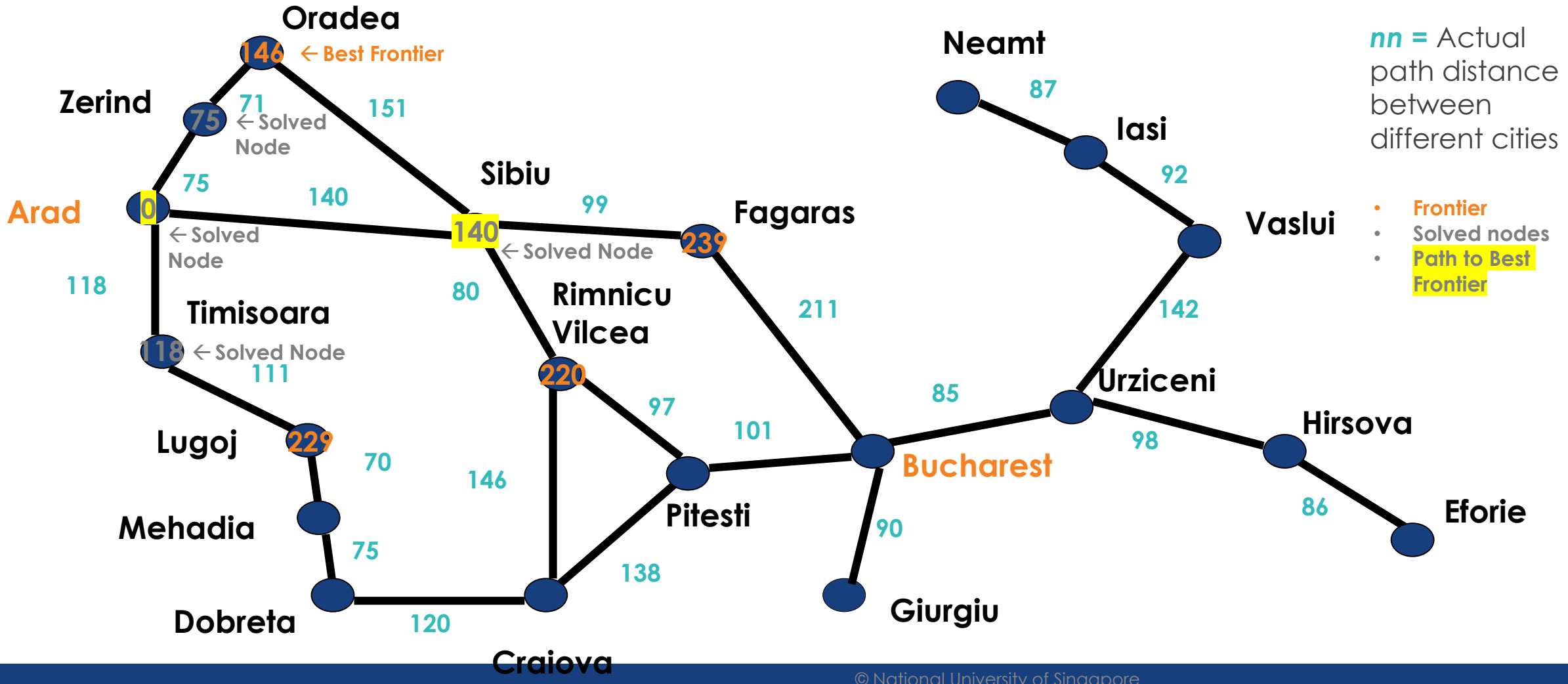
Uninformed Search Techniques (Graph)

Update list of Solved Nodes;



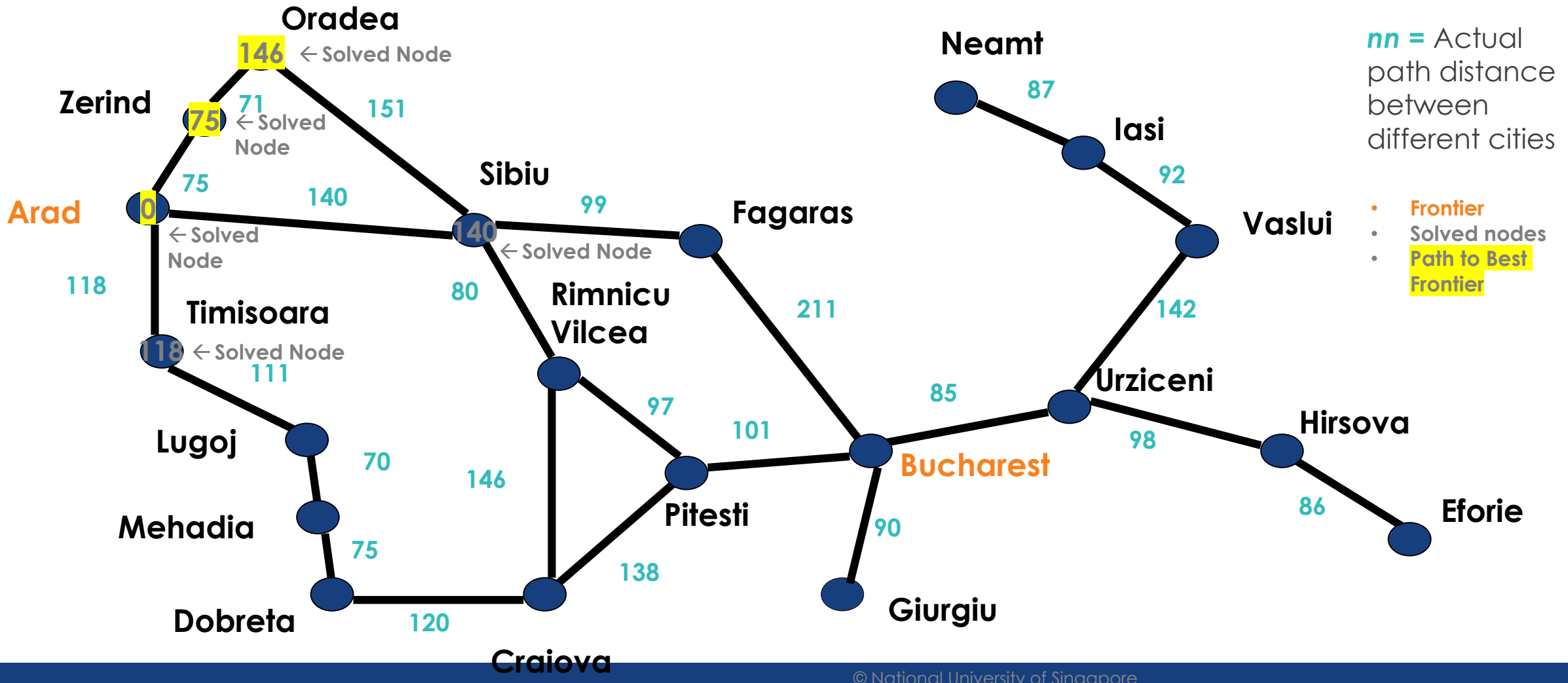
Uninformed Search Techniques (Graph)

Expand and calculate Frontier Nodes; Determine Best Frontier Node;



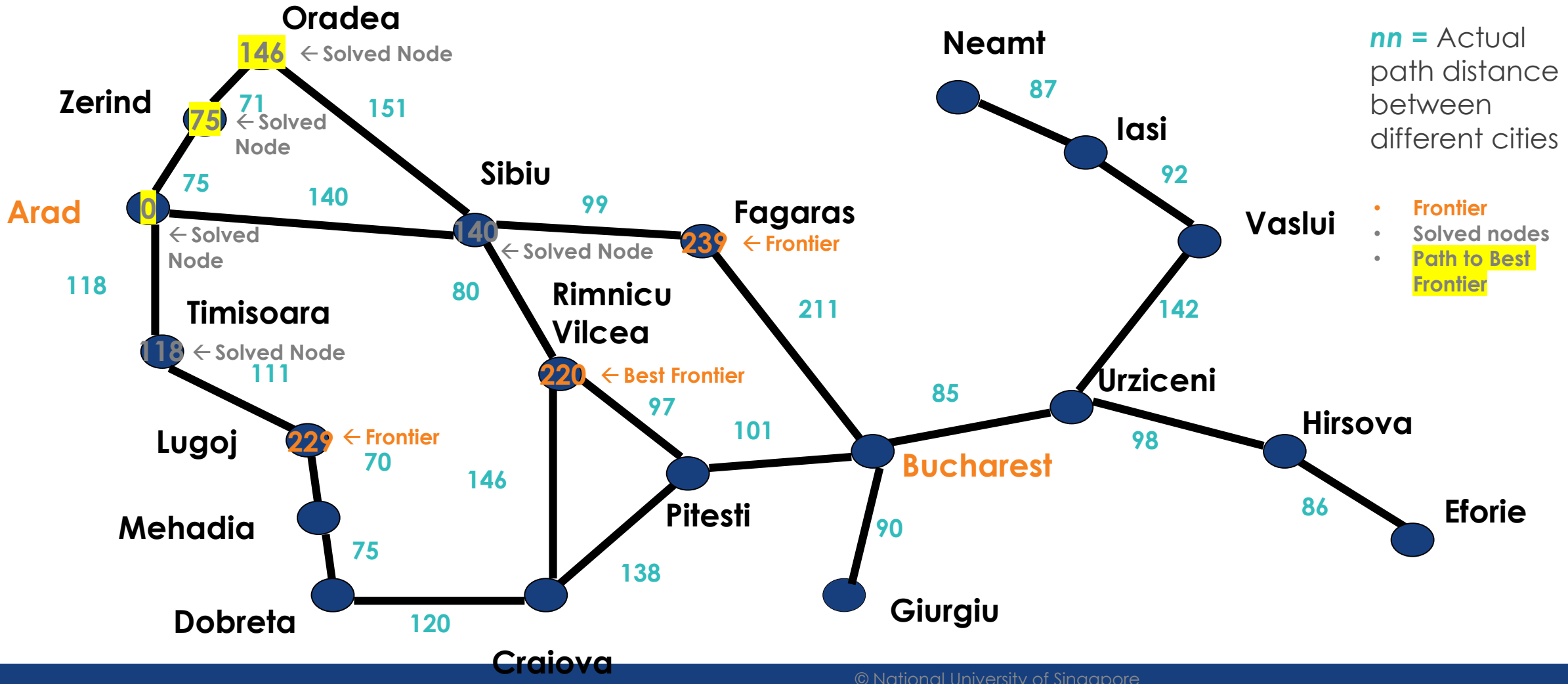
Uninformed Search Techniques (Graph)

Update list of Solved Nodes;



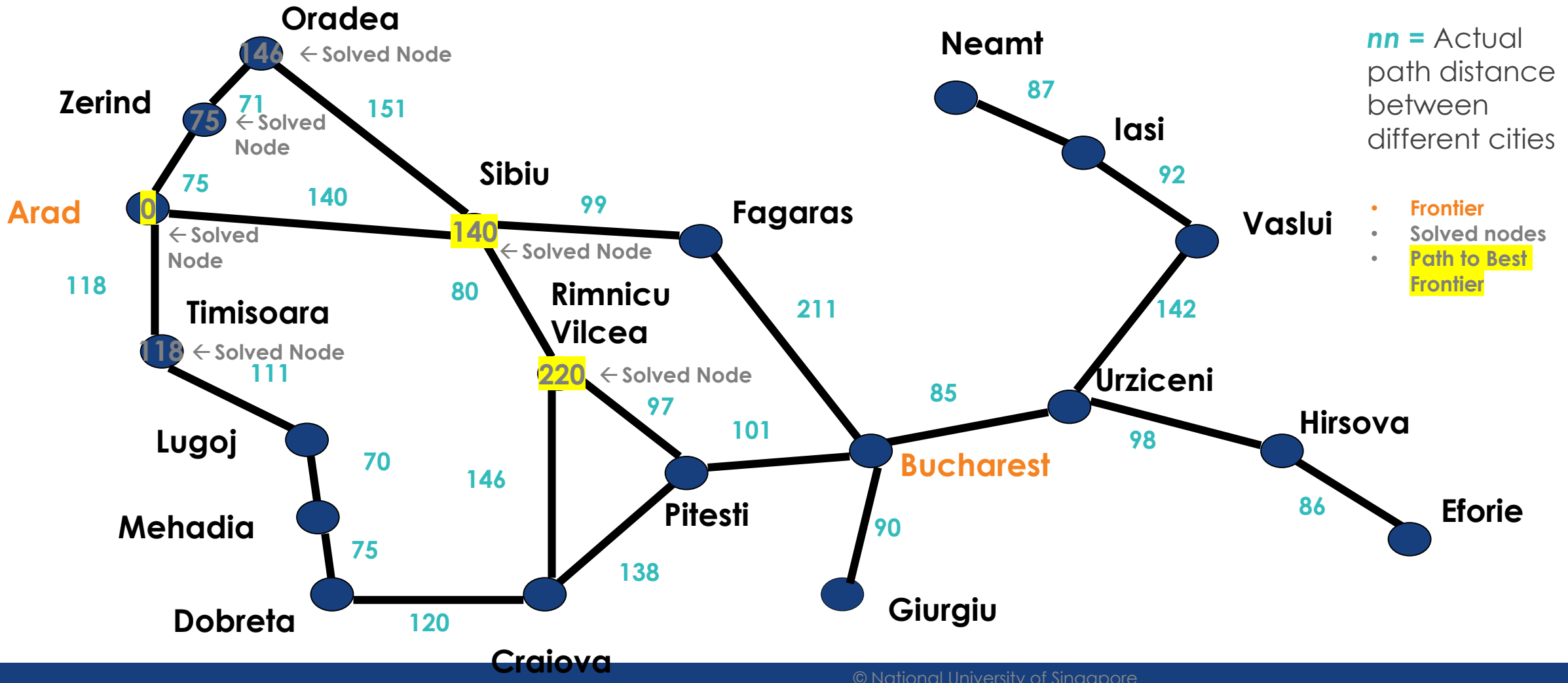
Uninformed Search Techniques (Graph)

Expand and calculate Frontier Nodes; Determine Best Frontier Node;



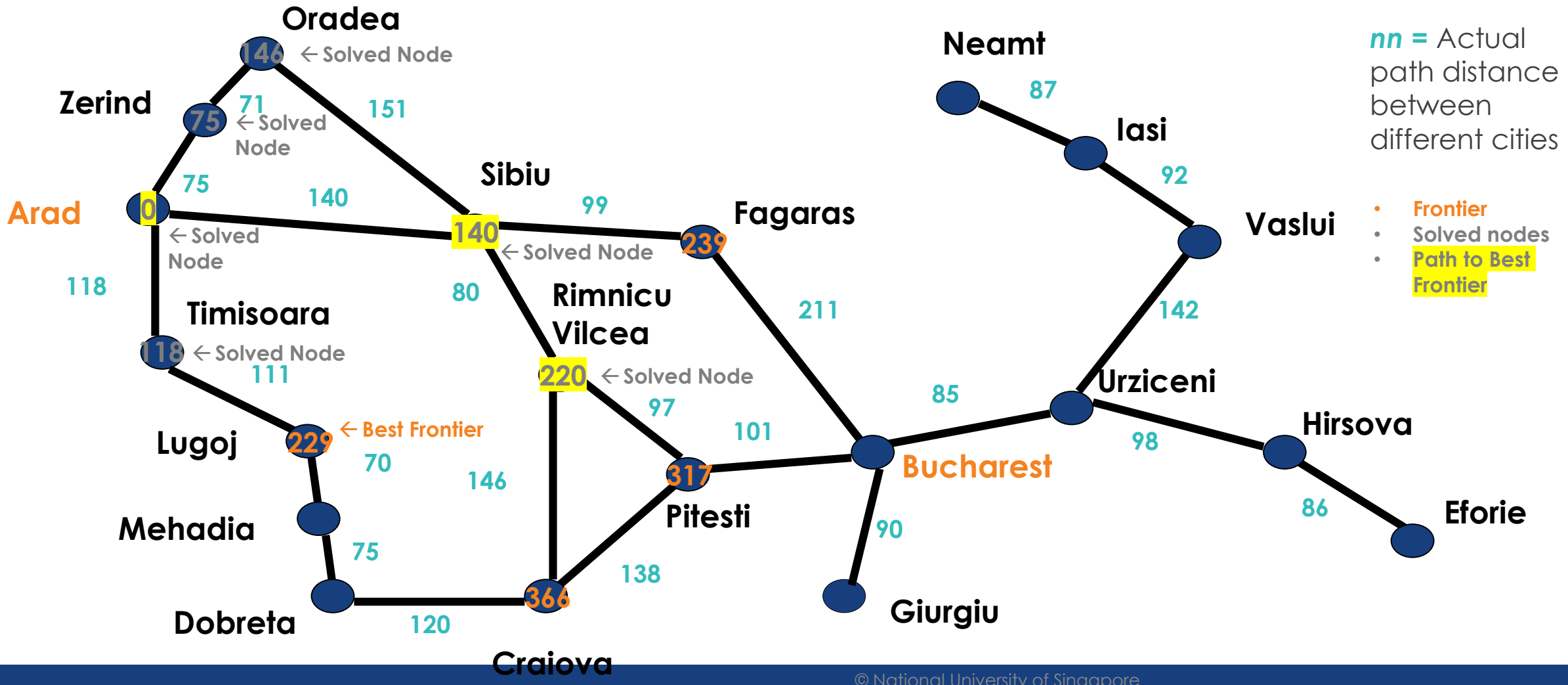
Uninformed Search Techniques (Graph)

Update list of Solved Nodes;



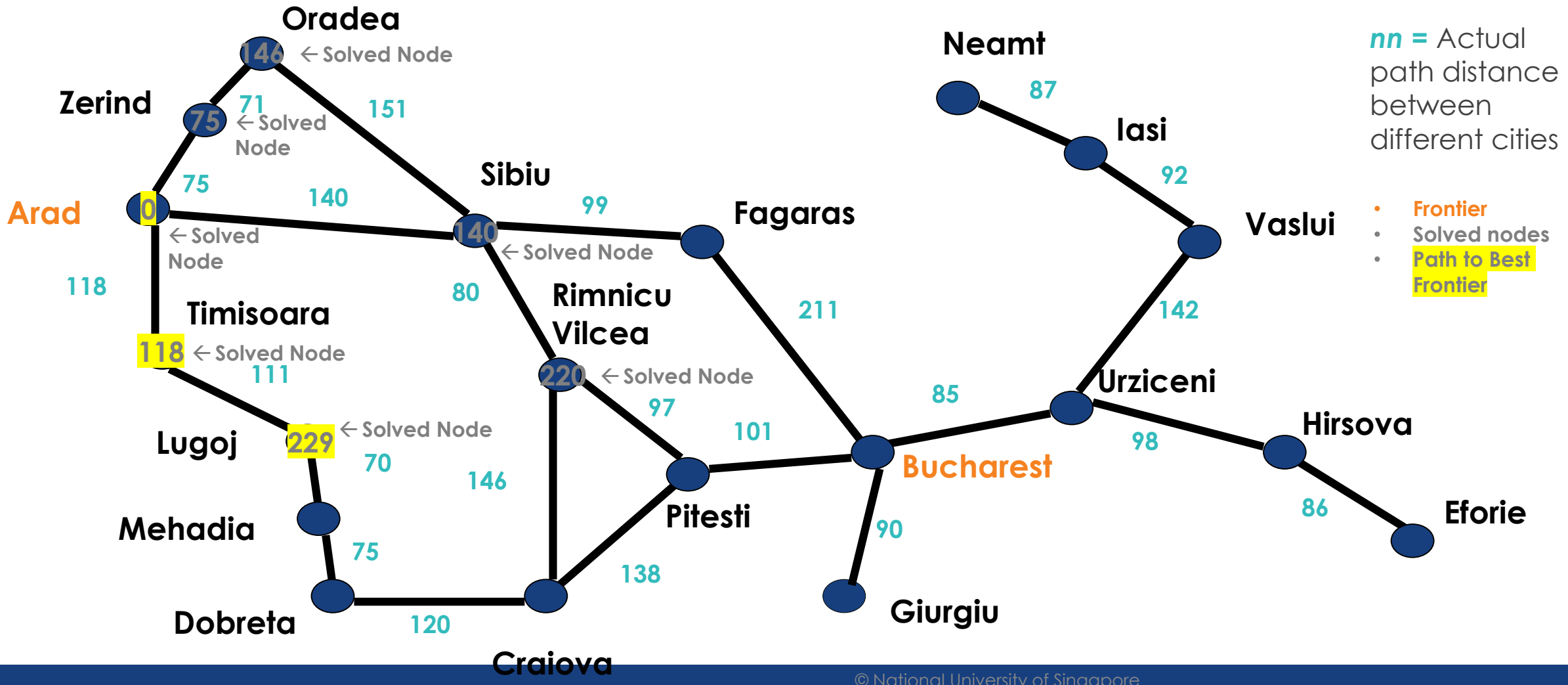
Uninformed Search Techniques (Graph)

Expand and calculate Frontier Nodes; Determine Best Frontier Node;



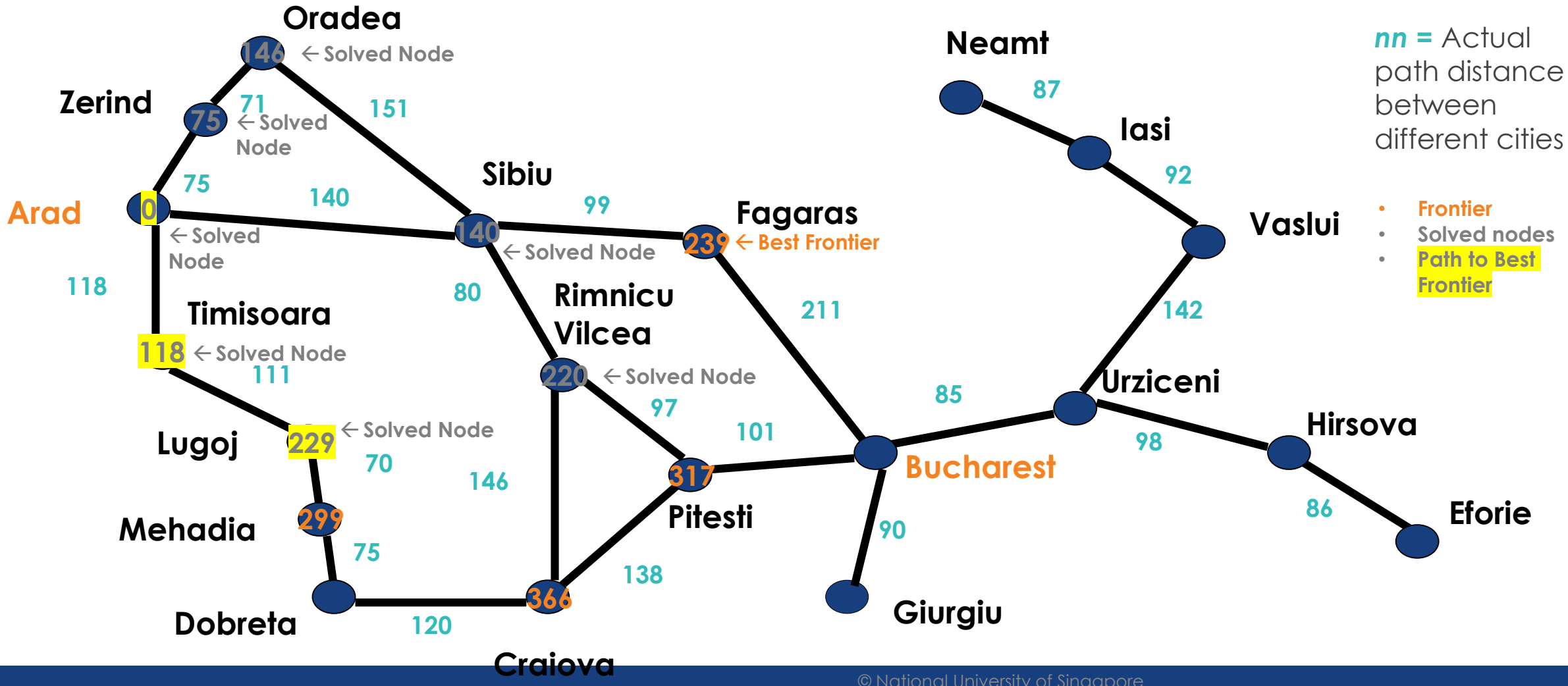
Uninformed Search Techniques (Graph)

Expand and calculate Frontier Nodes; Determine Best Frontier Node;



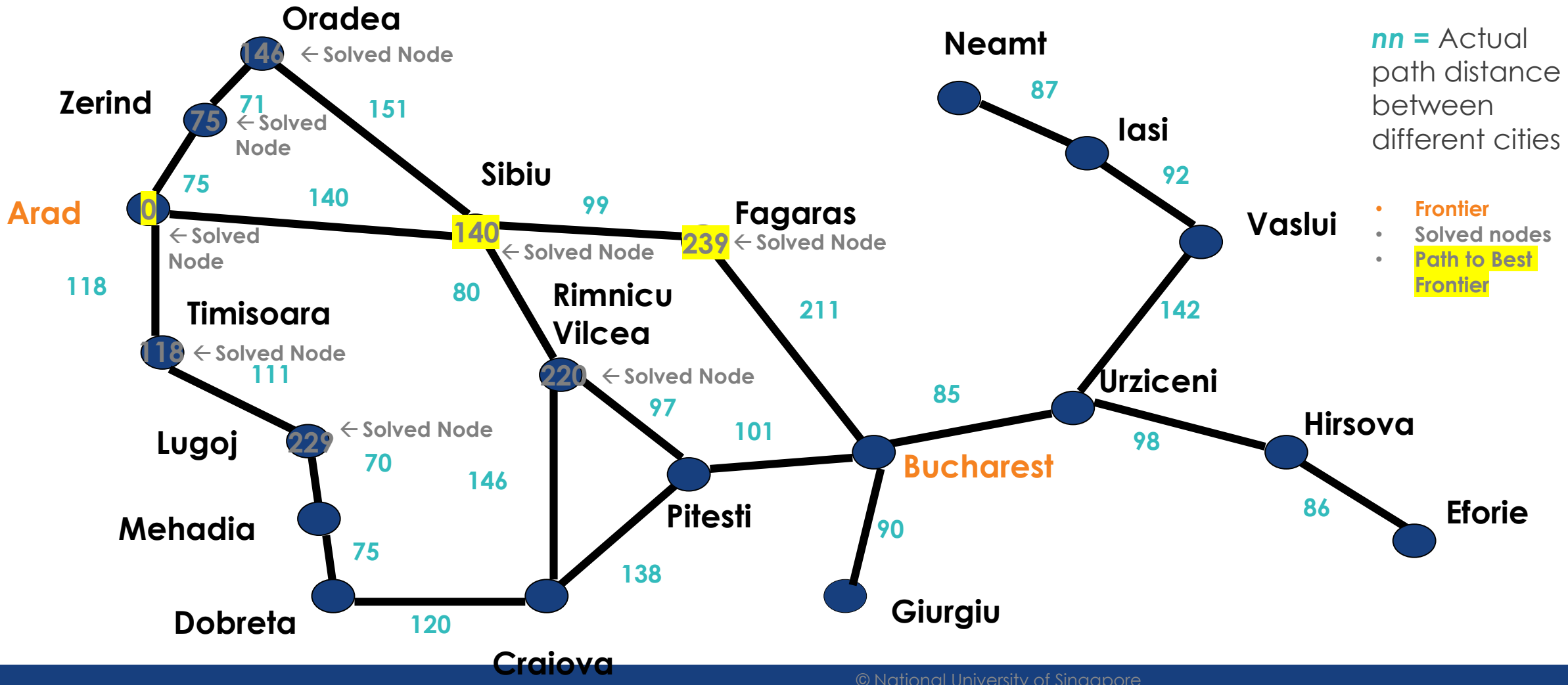
Uninformed Search Techniques (Graph)

Expand and calculate Frontier Nodes; Determine Best Frontier Node;



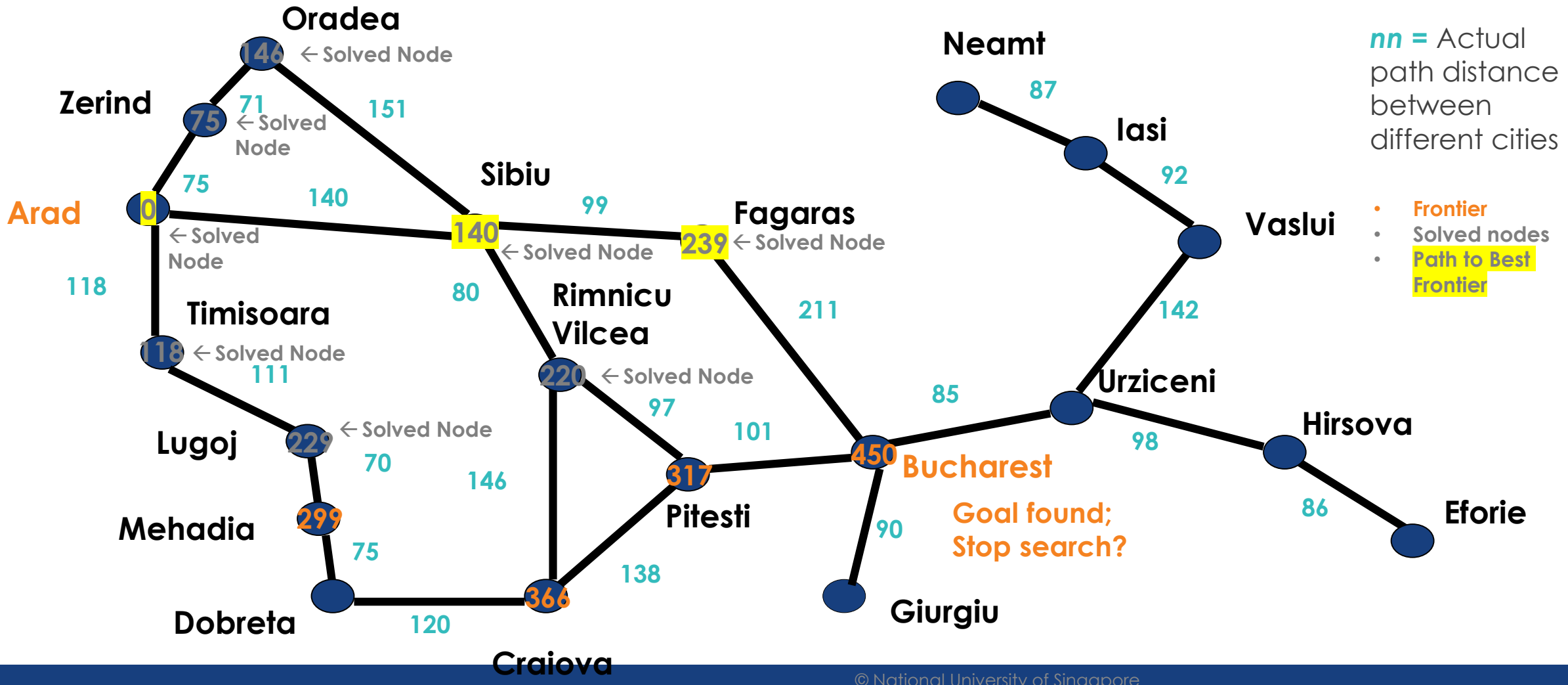
Uninformed Search Techniques (Graph)

Update list of Solved Nodes;



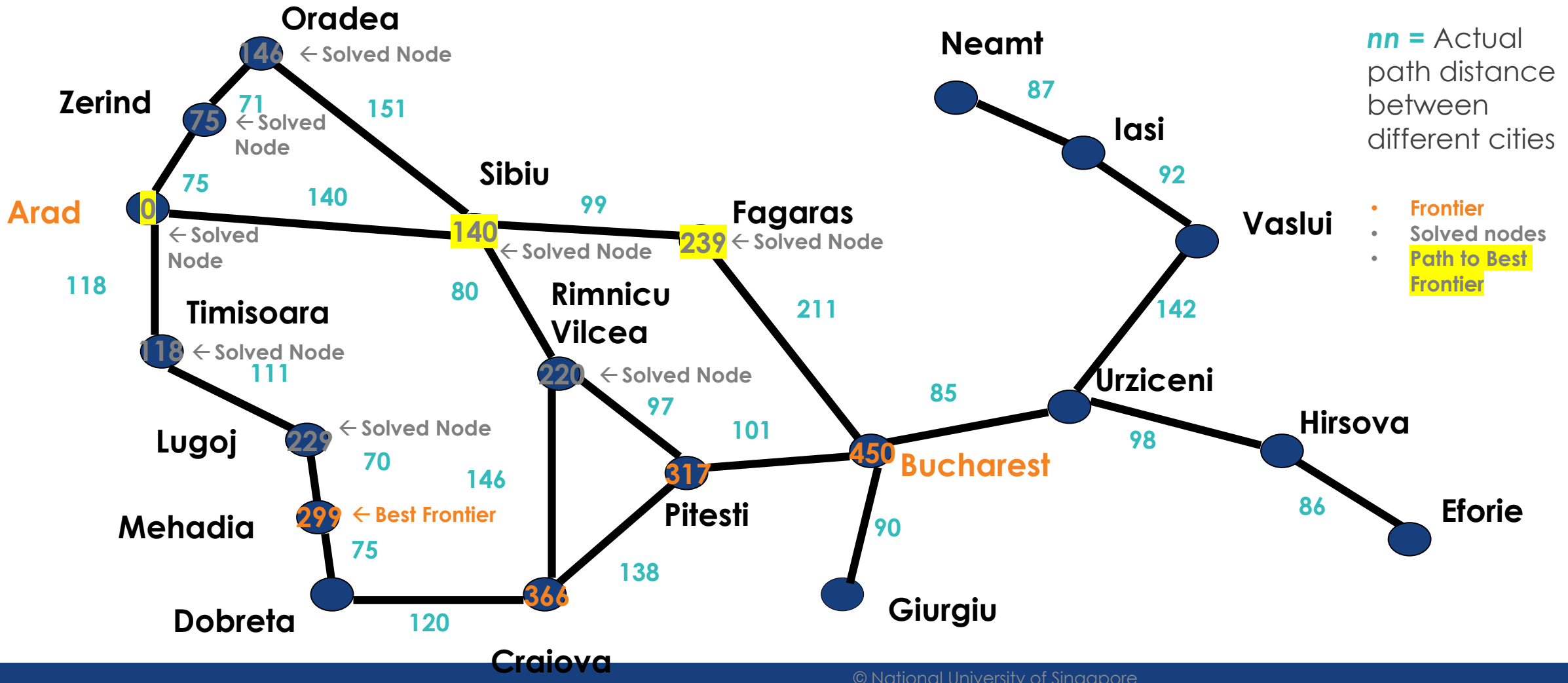
Uninformed Search Techniques (Graph)

Expand and calculate Frontier Nodes;



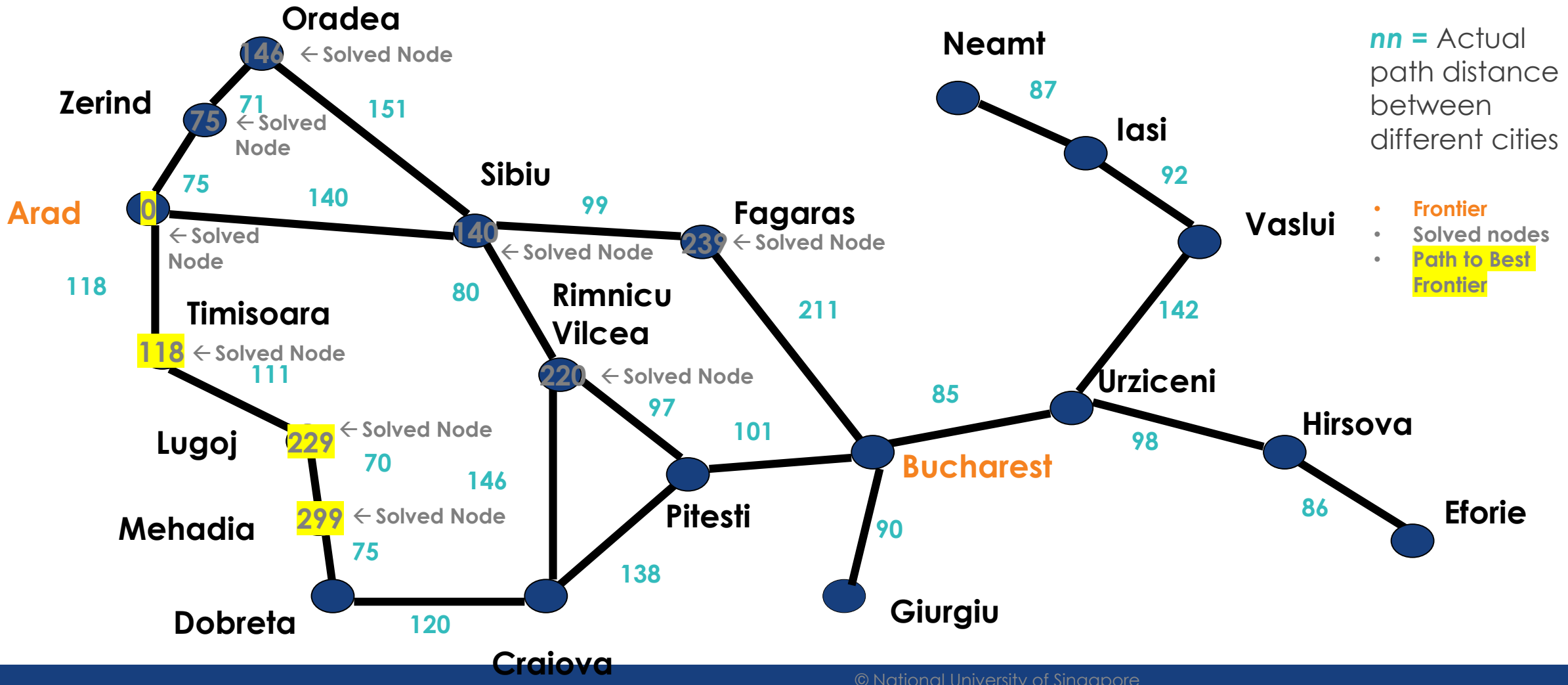
Uninformed Search Techniques (Graph)

Determine Best Frontier Node;



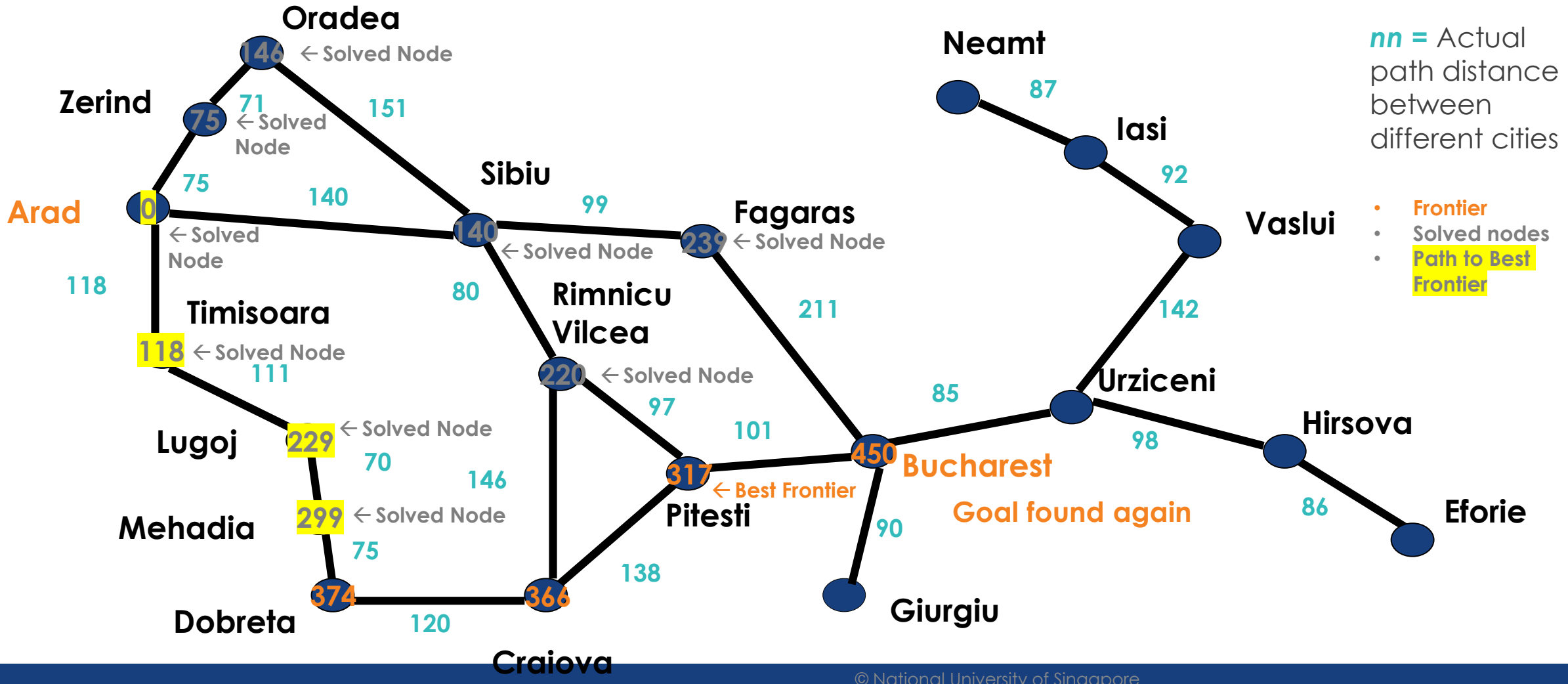
Uninformed Search Techniques (Graph)

Update list of Solved Nodes;



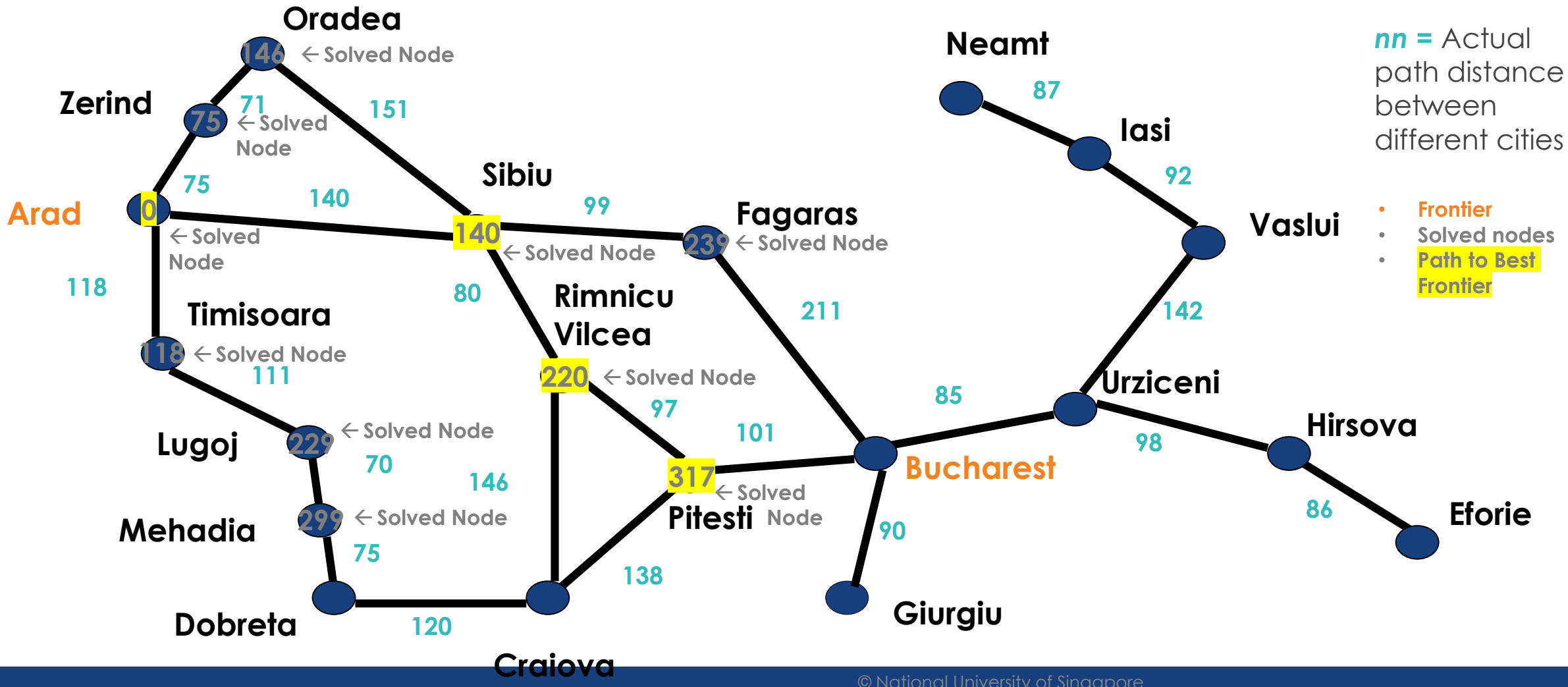
Uninformed Search Techniques (Graph)

Expand and calculate Frontier Nodes; Determine Best Frontier Node;



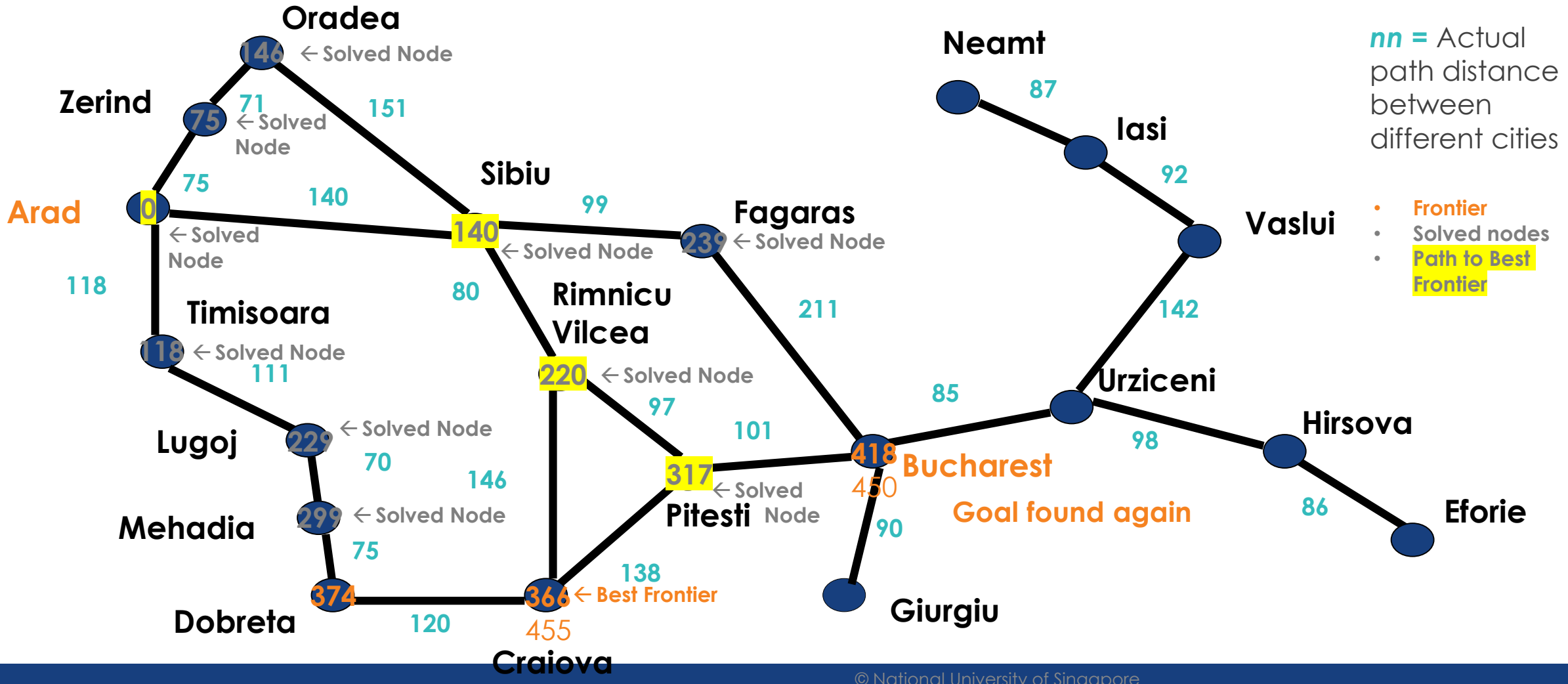
Uninformed Search Techniques (Graph)

Update list of Solved Nodes;



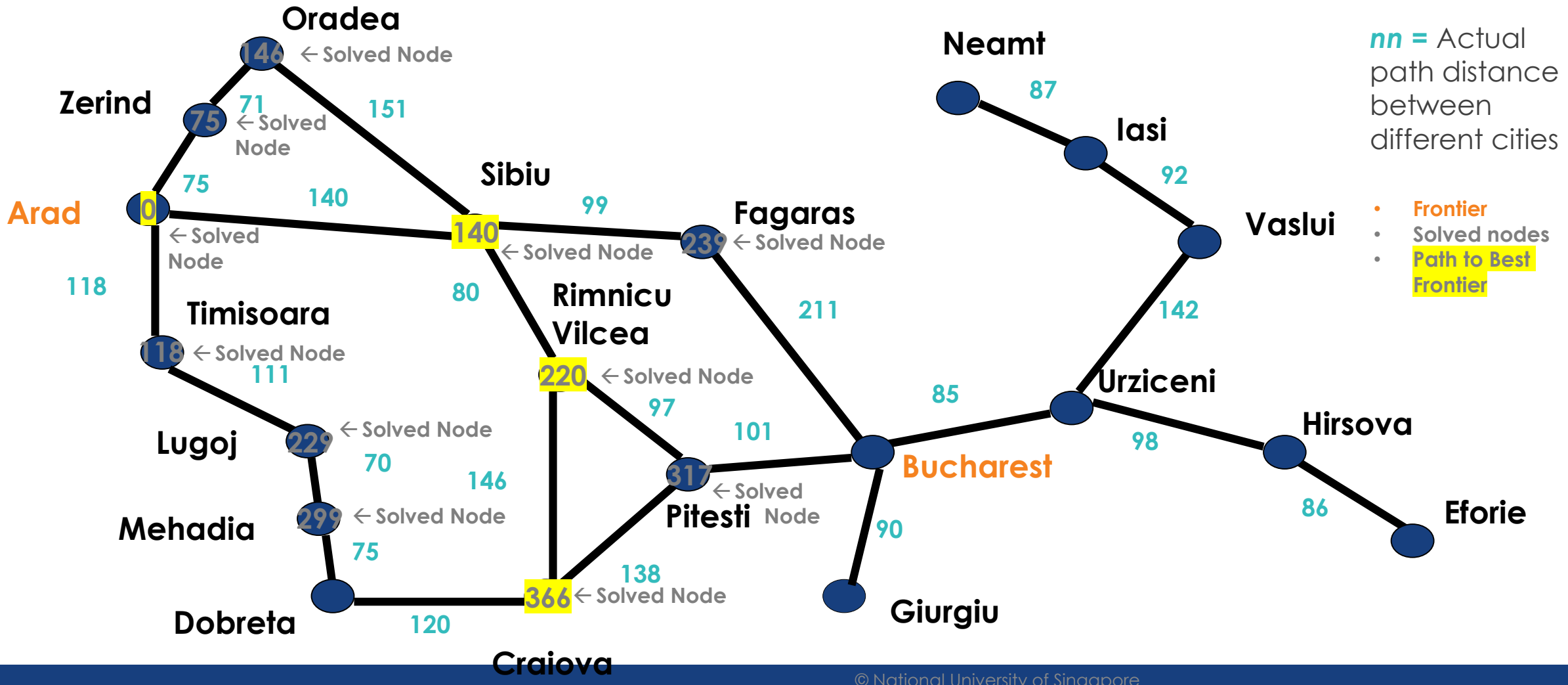
Uninformed Search Techniques (Graph)

Expand and calculate Frontier Nodes; Determine Best Frontier Node;



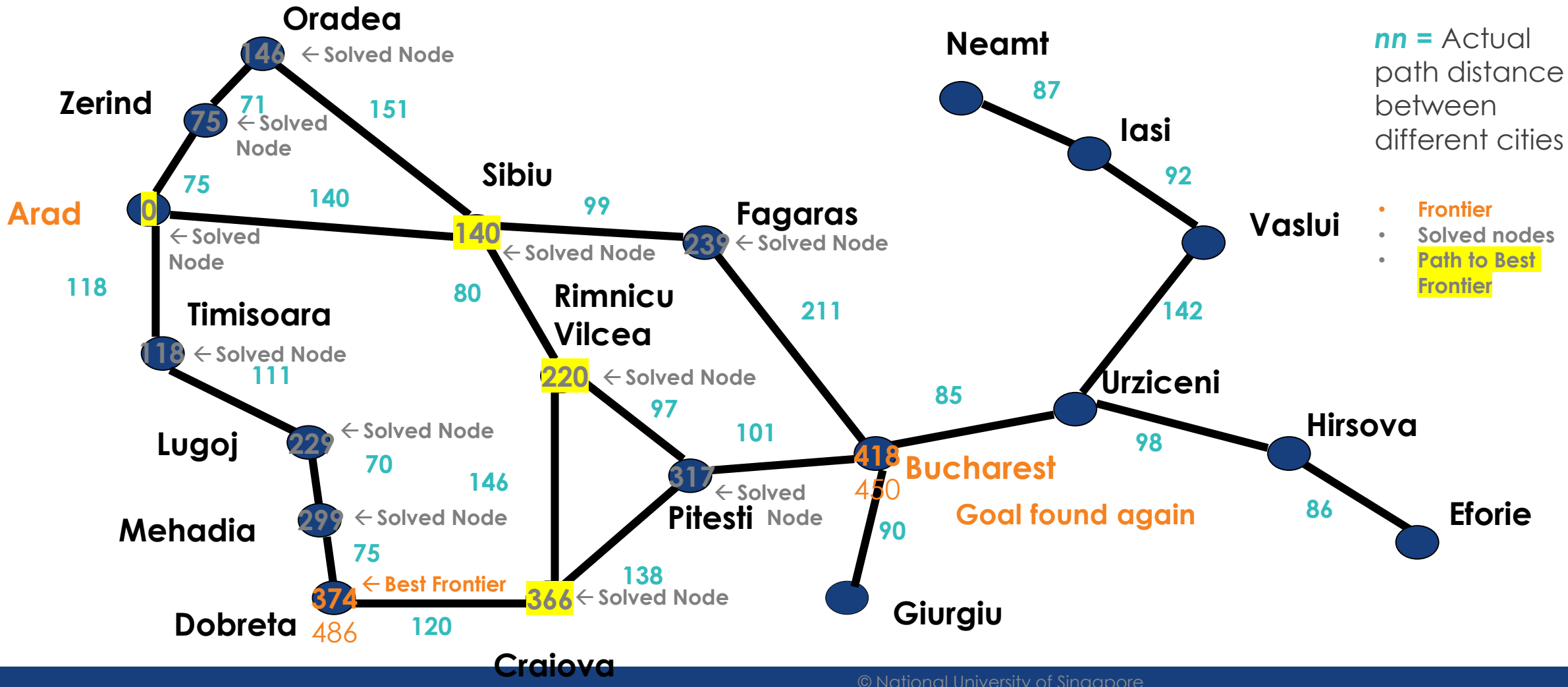
Uninformed Search Techniques (Graph)

Update list of Solved Nodes;



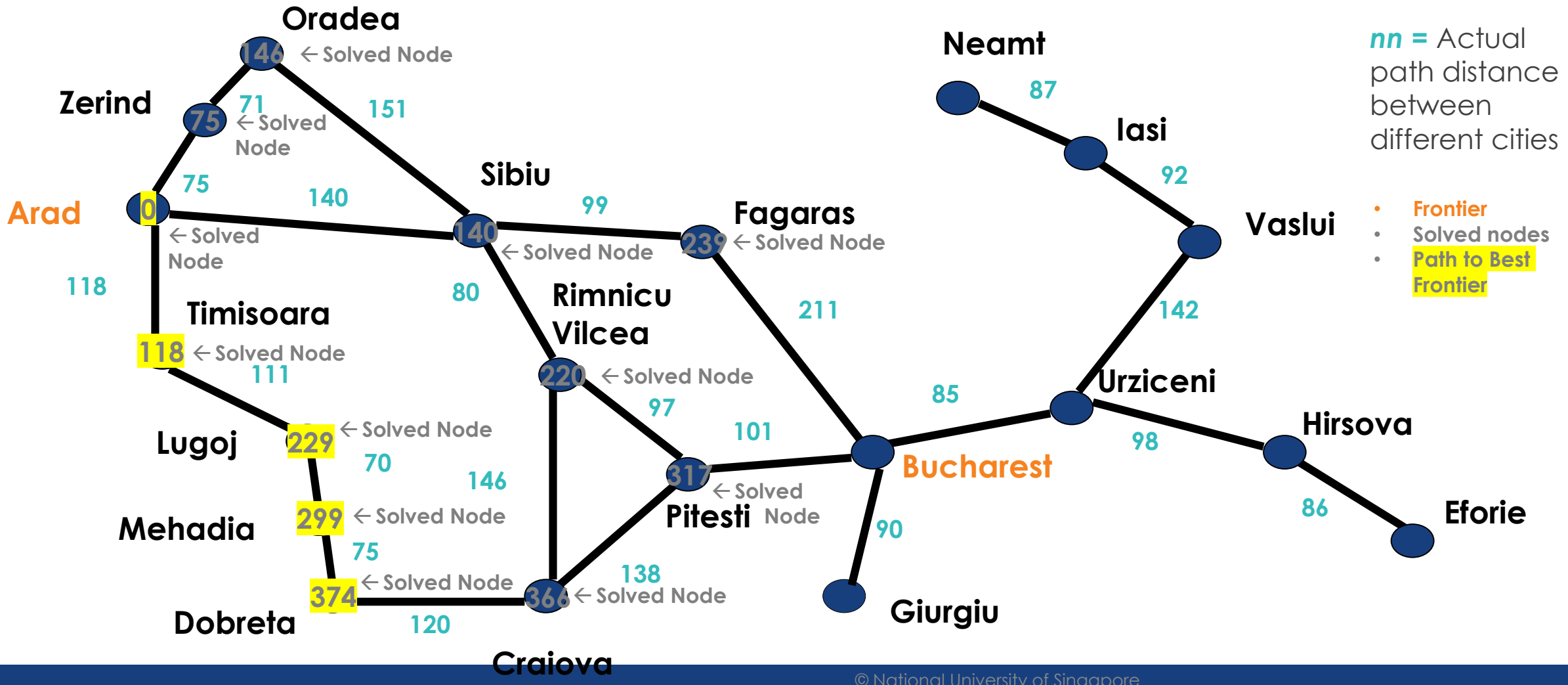
Uninformed Search Techniques (Graph)

Expand and calculate Frontier Nodes; Determine Best Frontier Node;



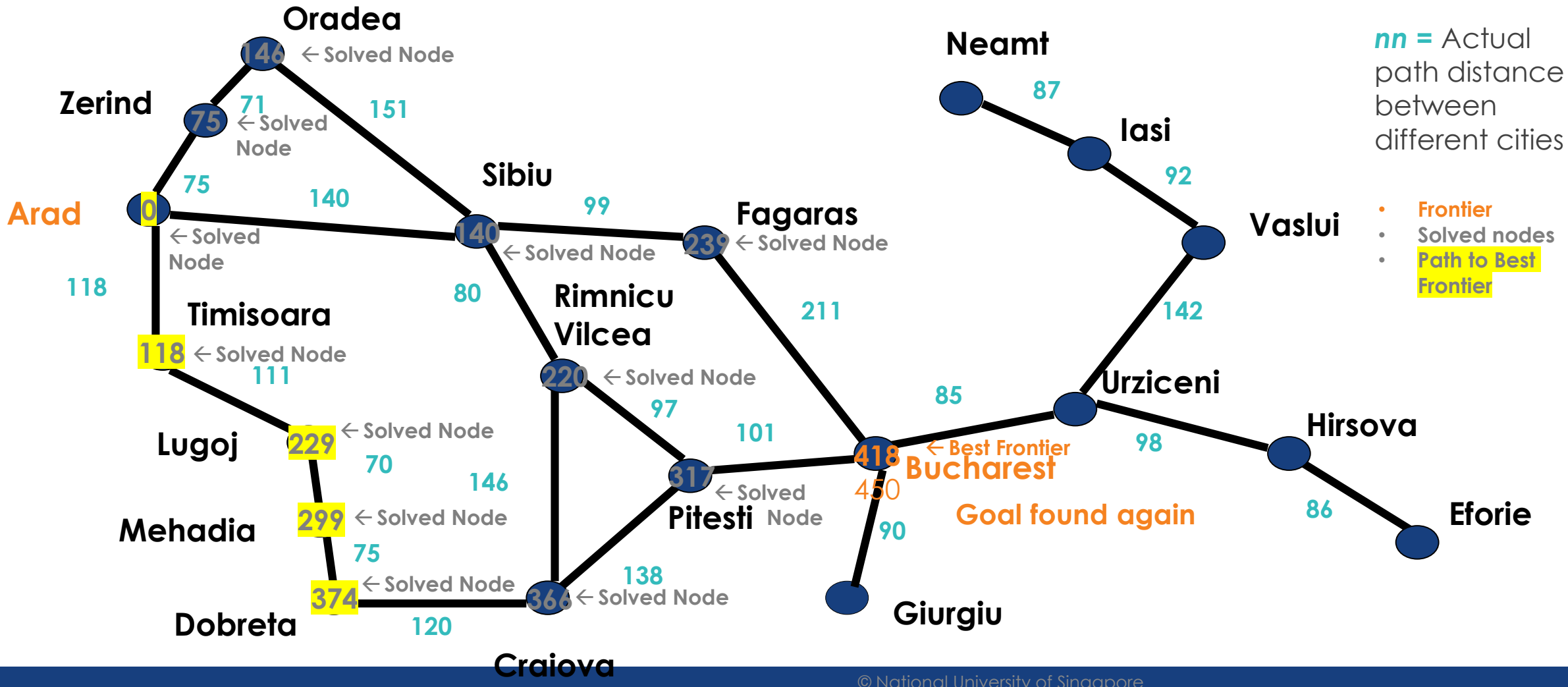
Uninformed Search Techniques (Graph)

Update list of Solved Nodes;



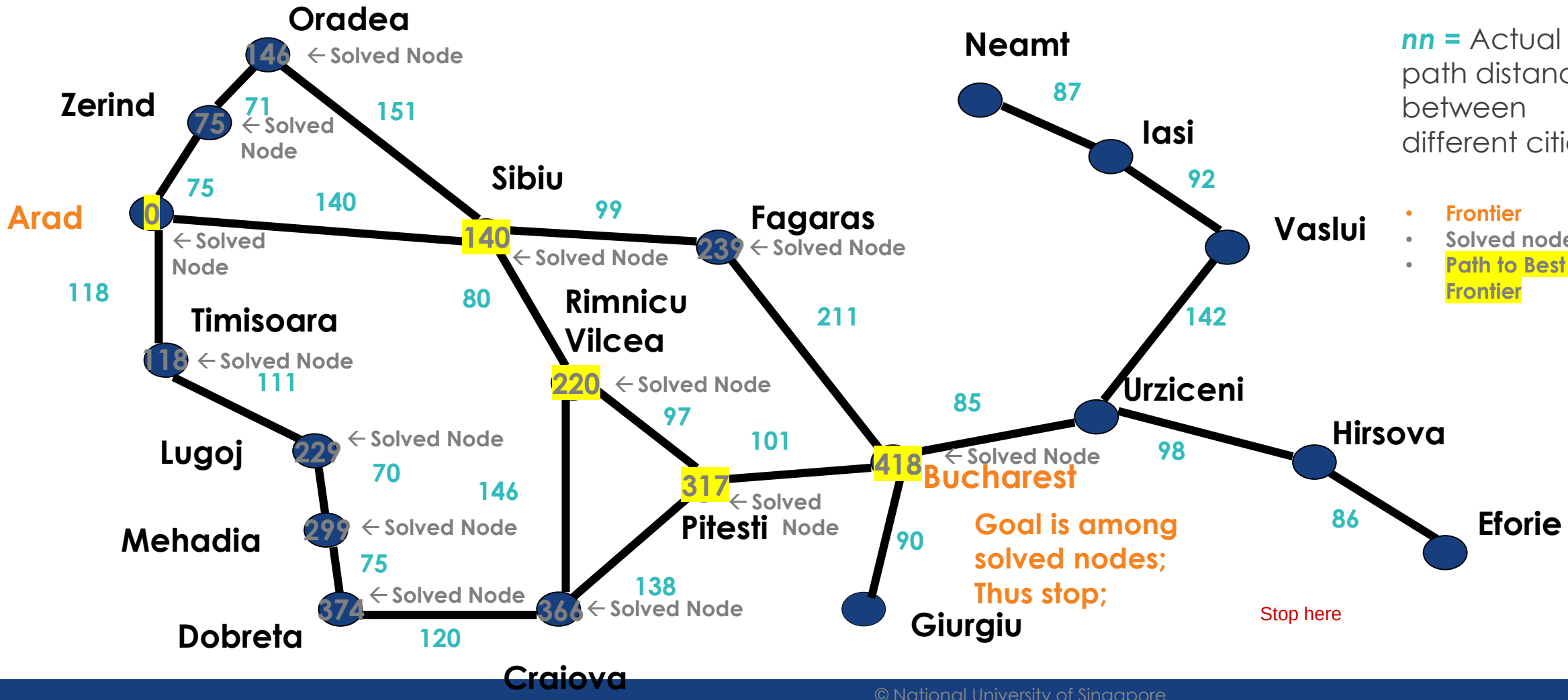
Uninformed Search Techniques (Graph)

Expand and calculate Frontier Nodes; Determine Best Frontier Node;



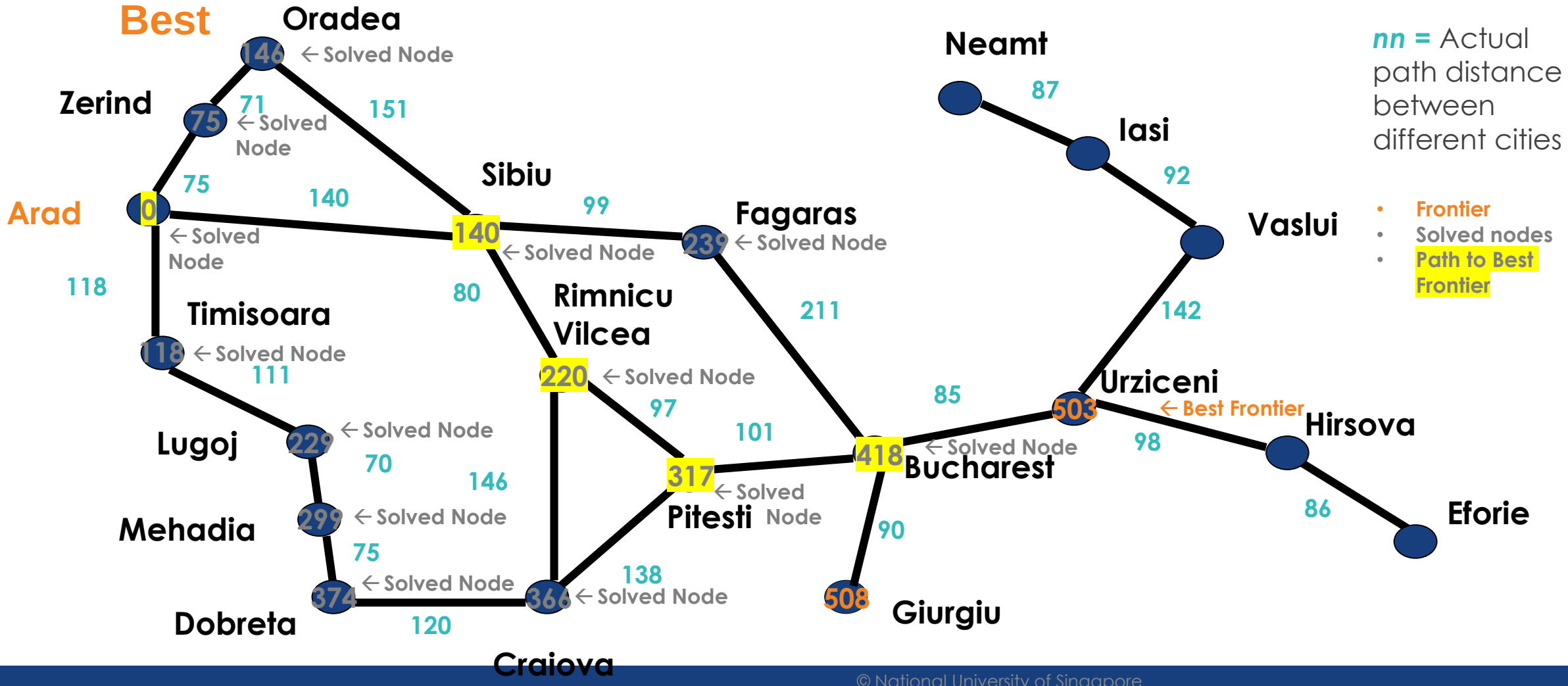
Uninformed Search Techniques (Graph)

Update list of Solved Nodes; Best Arad→Bucharest path is found;



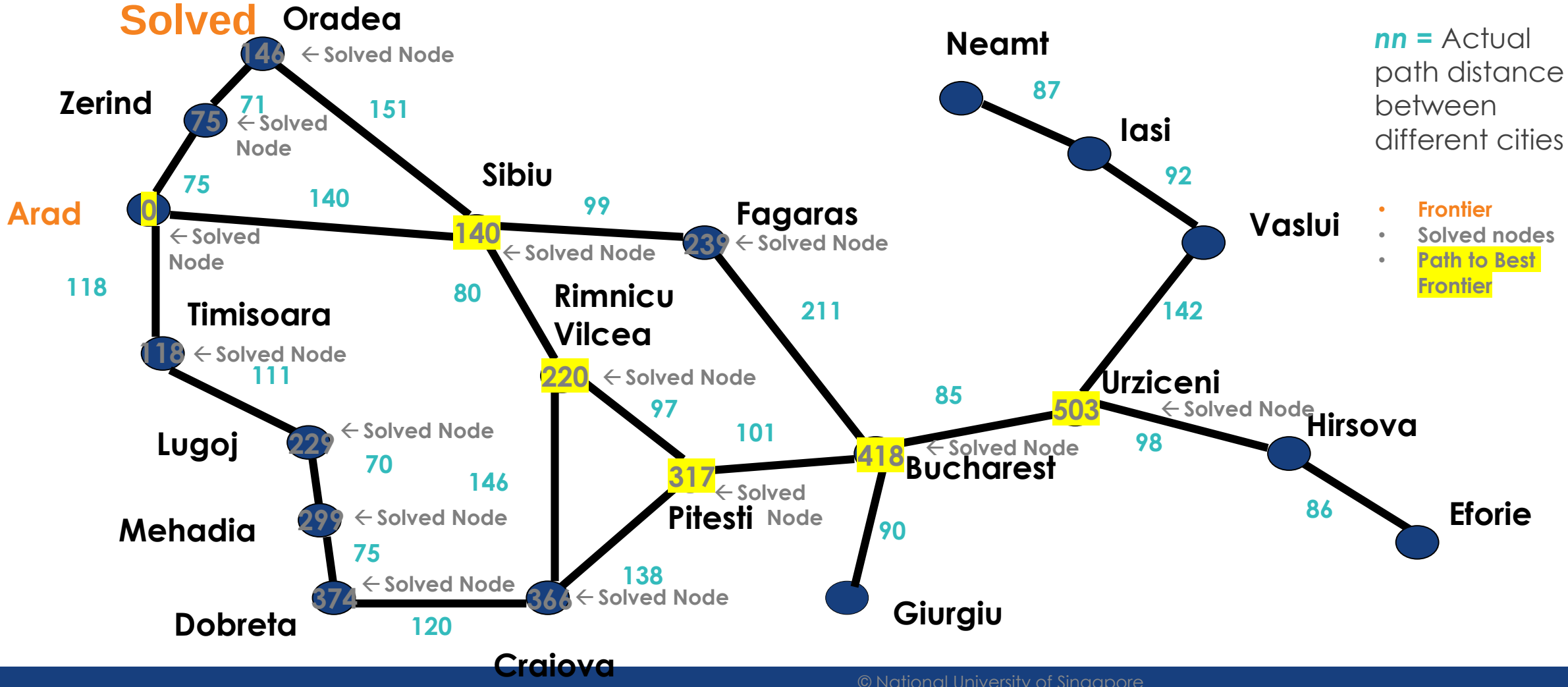
Uninformed Search Techniques (Graph)

Continue for all shortest paths starting from Arad to any other city…



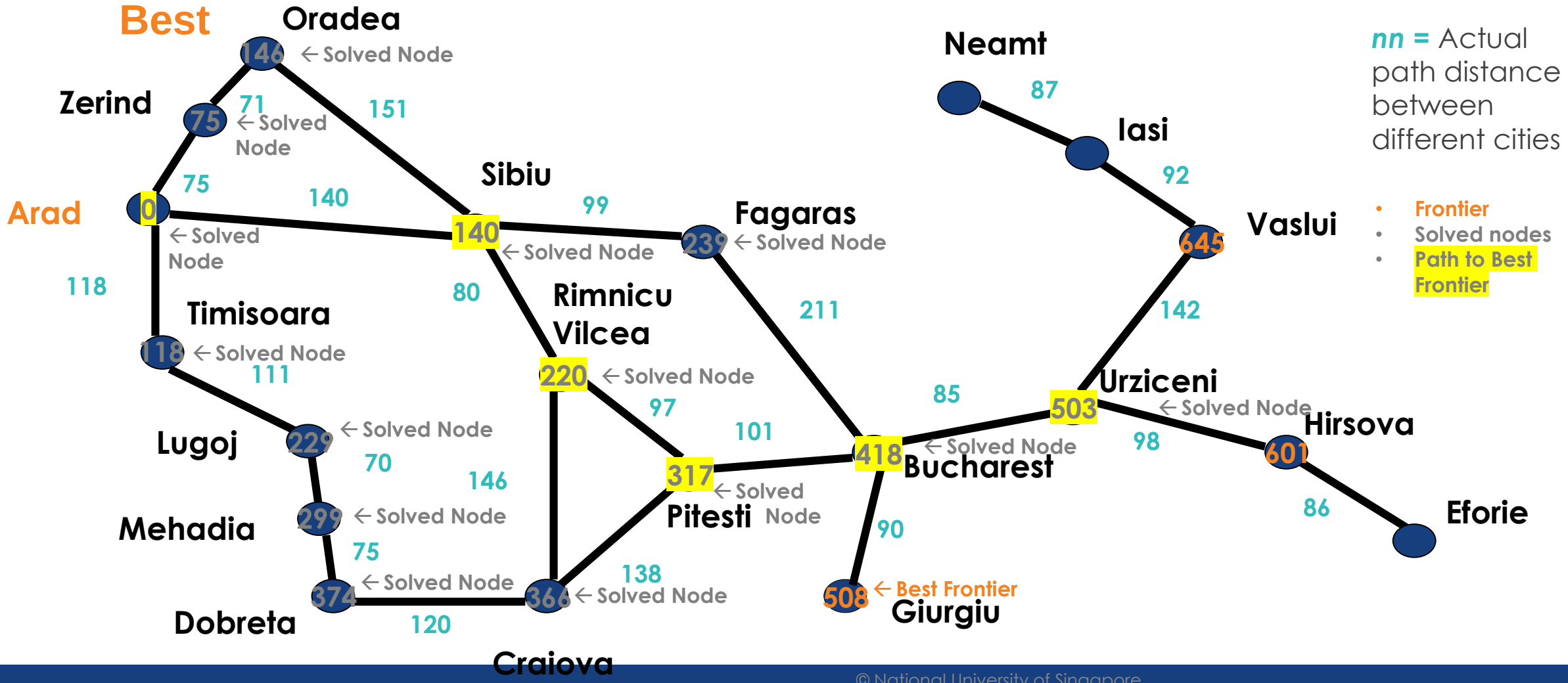
Uninformed Search Techniques (Graph)

Continue for all shortest paths starting from Arad to any other city…



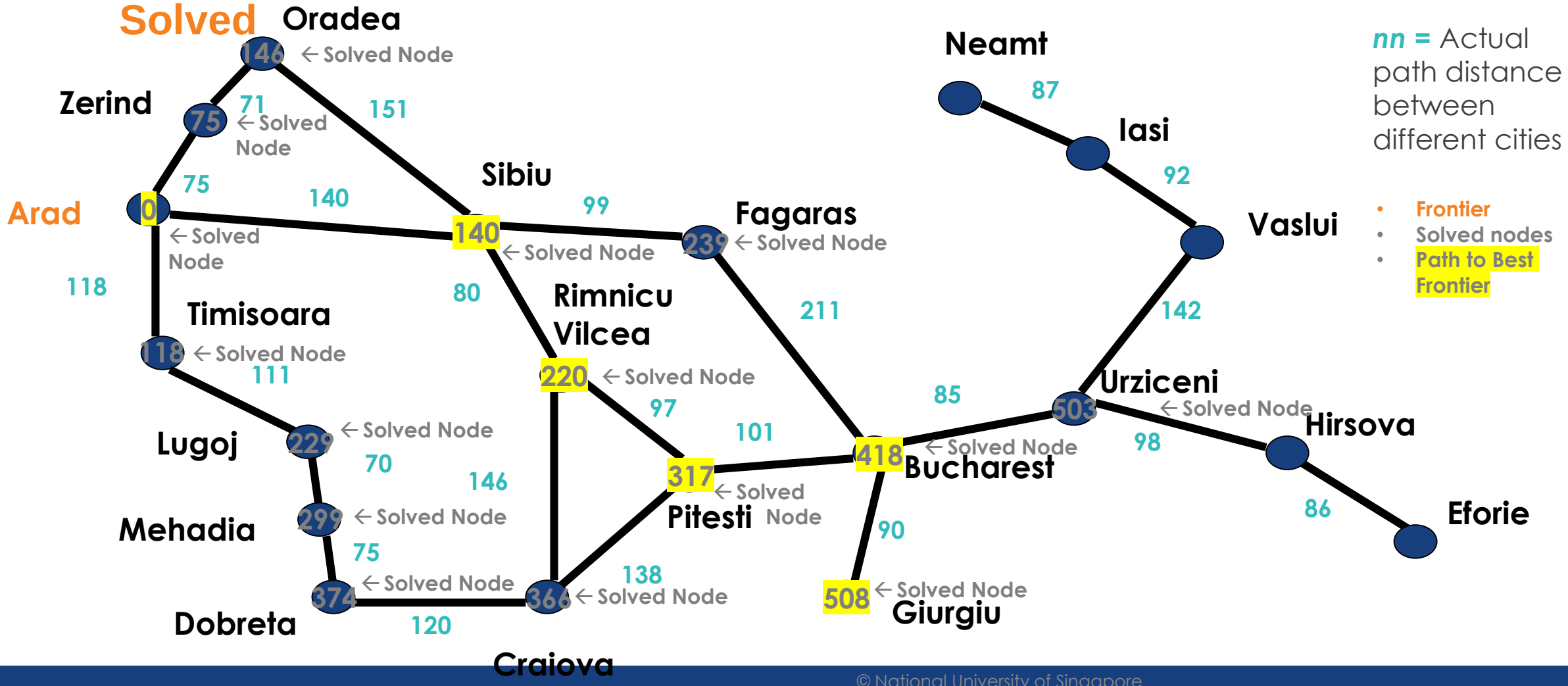
Uninformed Search Techniques (Graph)

Continue for all shortest paths starting from Arad to any other city…



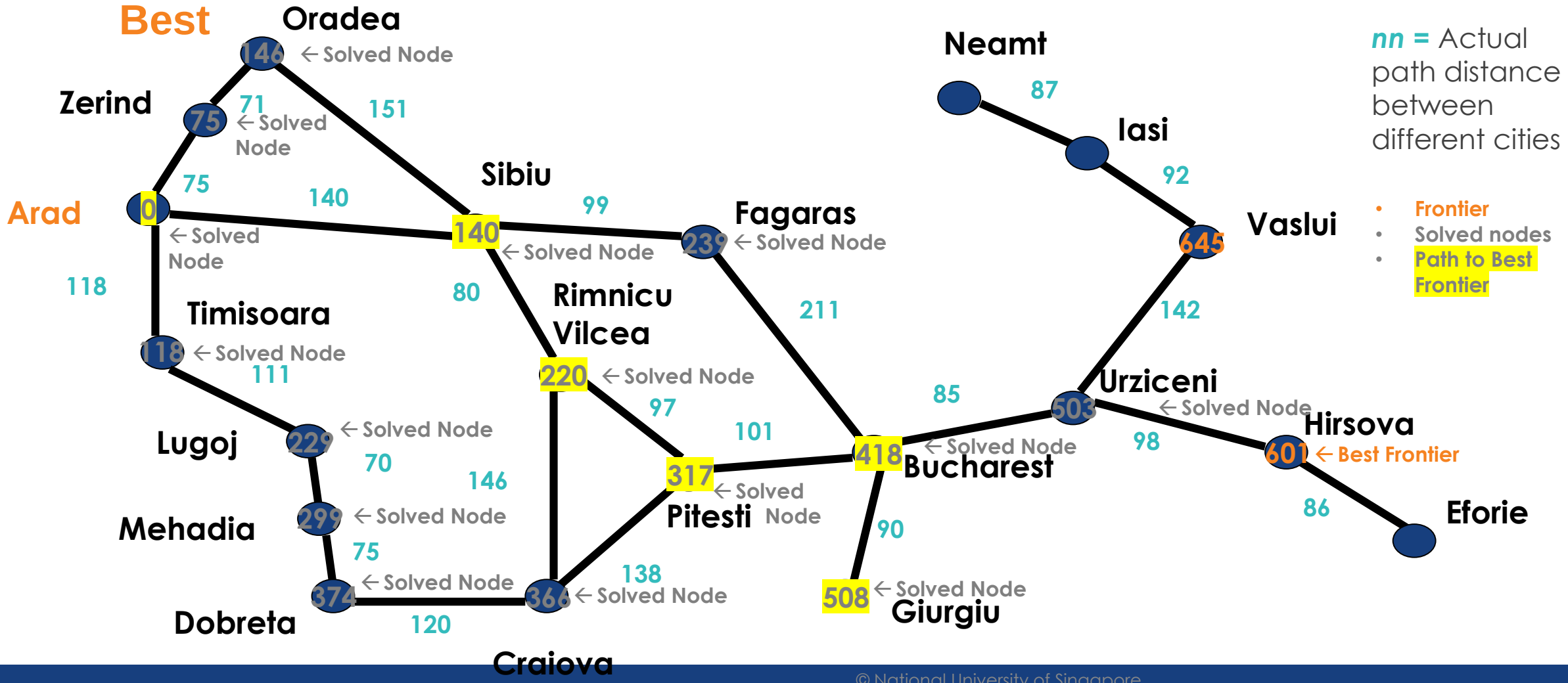
Uninformed Search Techniques (Graph)

Continue for all shortest paths starting from Arad to any other city…



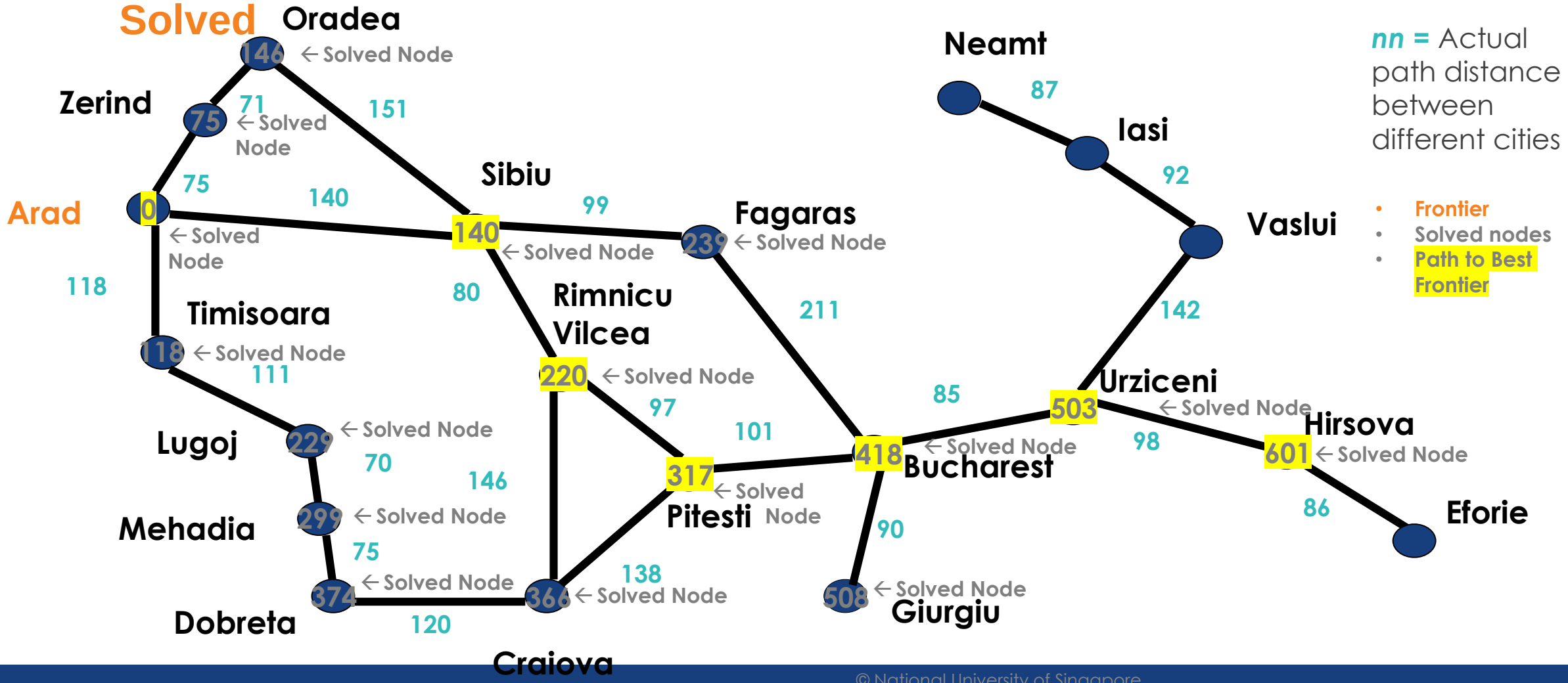
Uninformed Search Techniques (Graph)

Continue for all shortest paths starting from Arad to any other city…



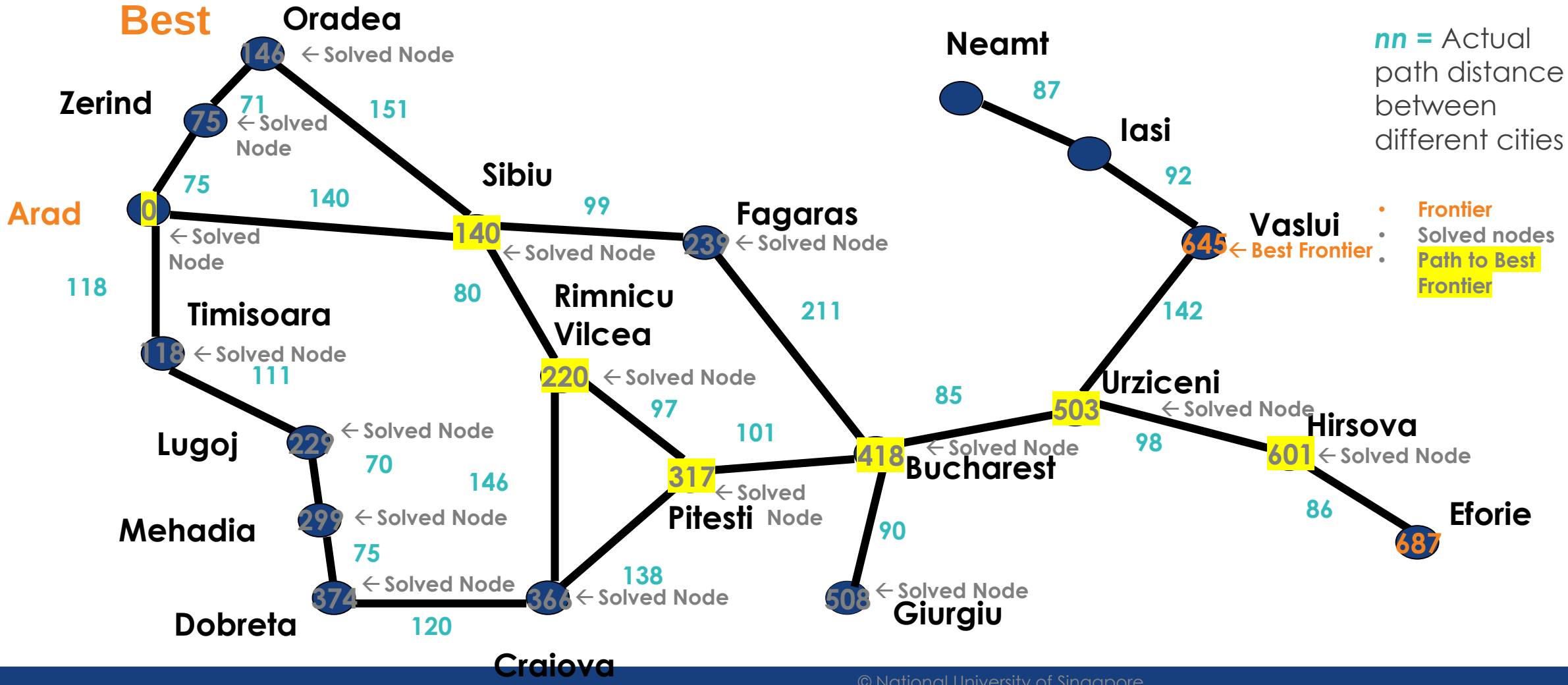
Uninformed Search Techniques (Graph)

Continue for all shortest paths starting from Arad to any other city…



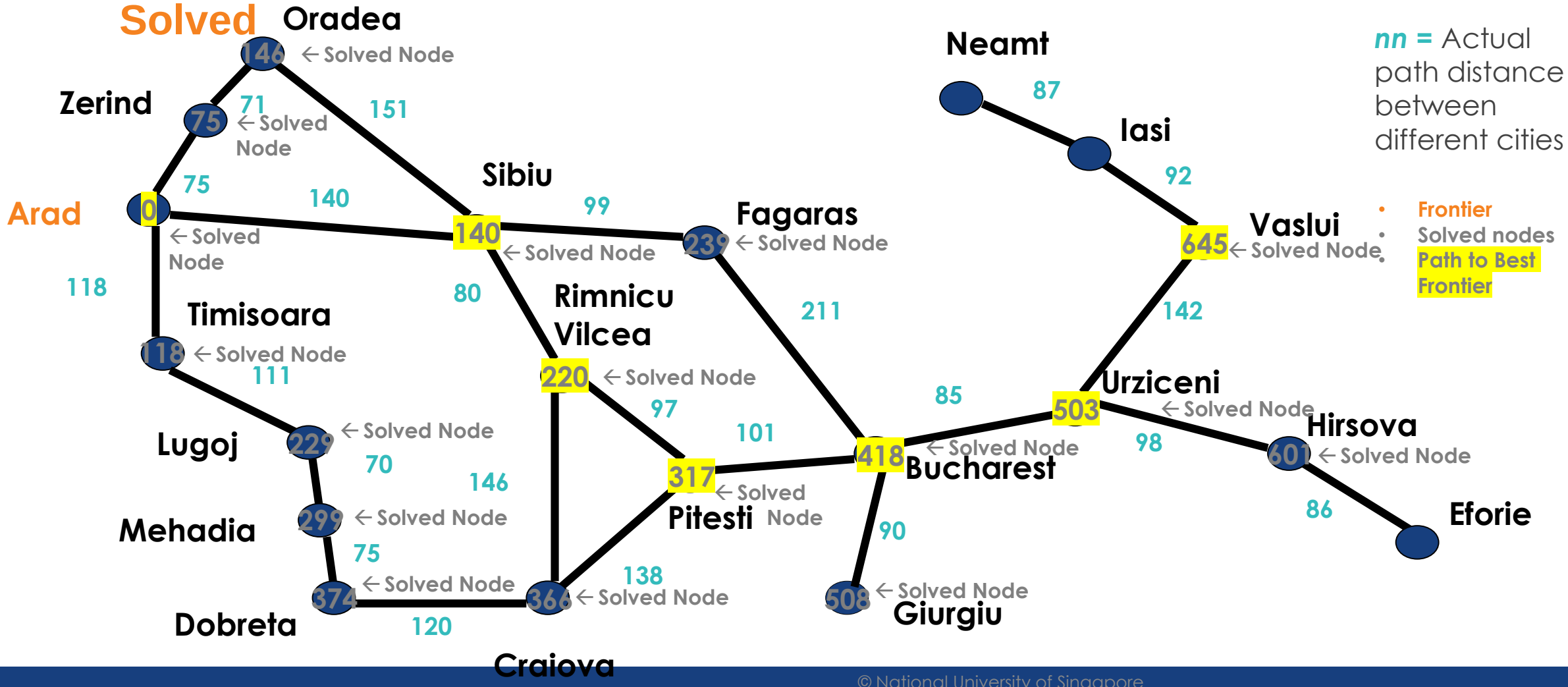
Uninformed Search Techniques (Graph)

Continue for all shortest paths starting from Arad to any other city…



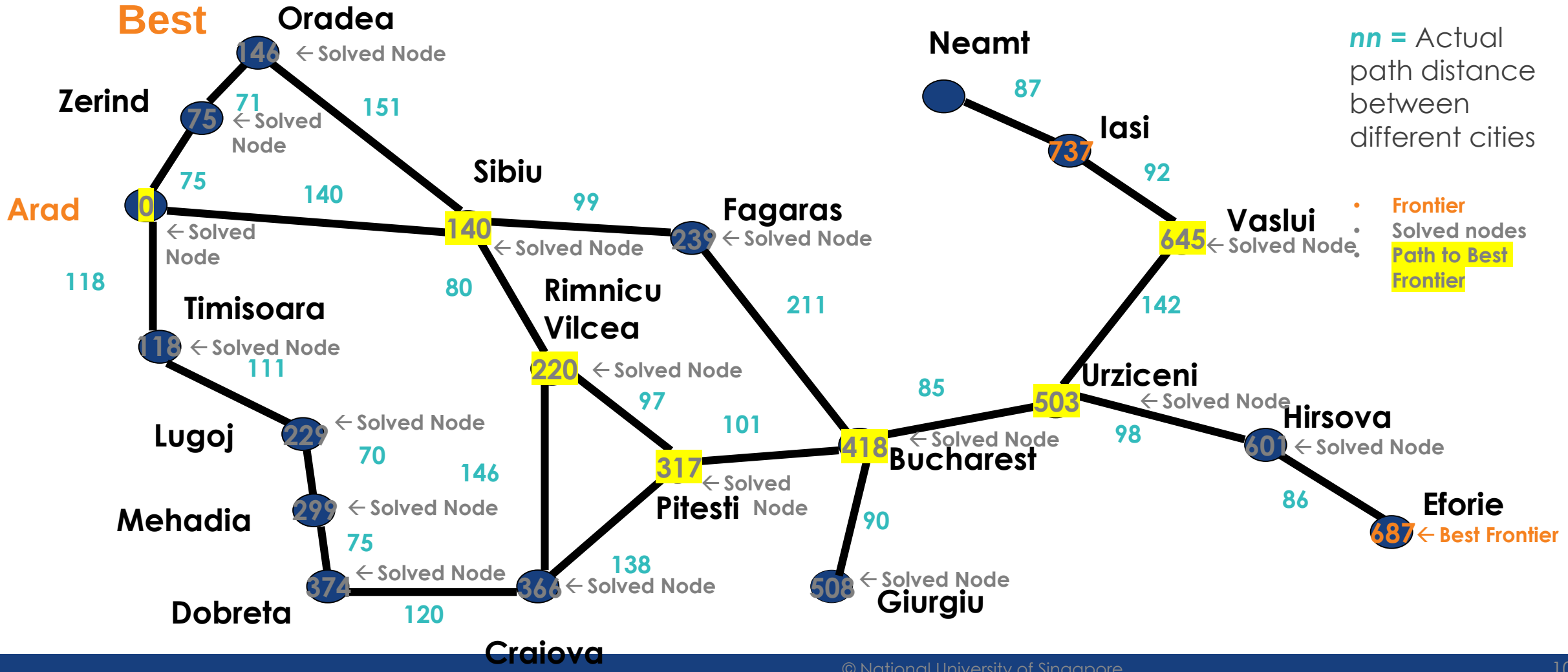
Uninformed Search Techniques (Graph)

Continue for all shortest paths starting from Arad to any other city…



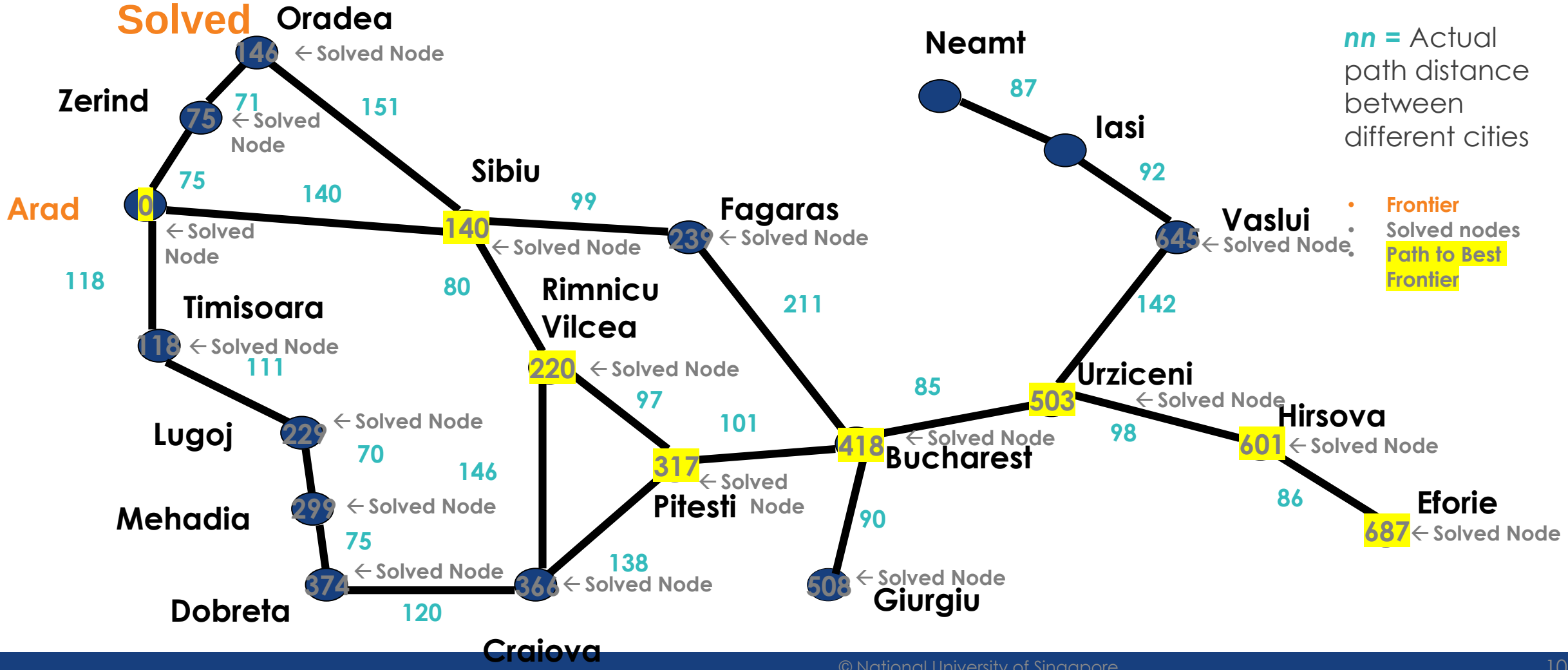
Uninformed Search Techniques (Graph)

Continue for all shortest paths starting from Arad to any other city…



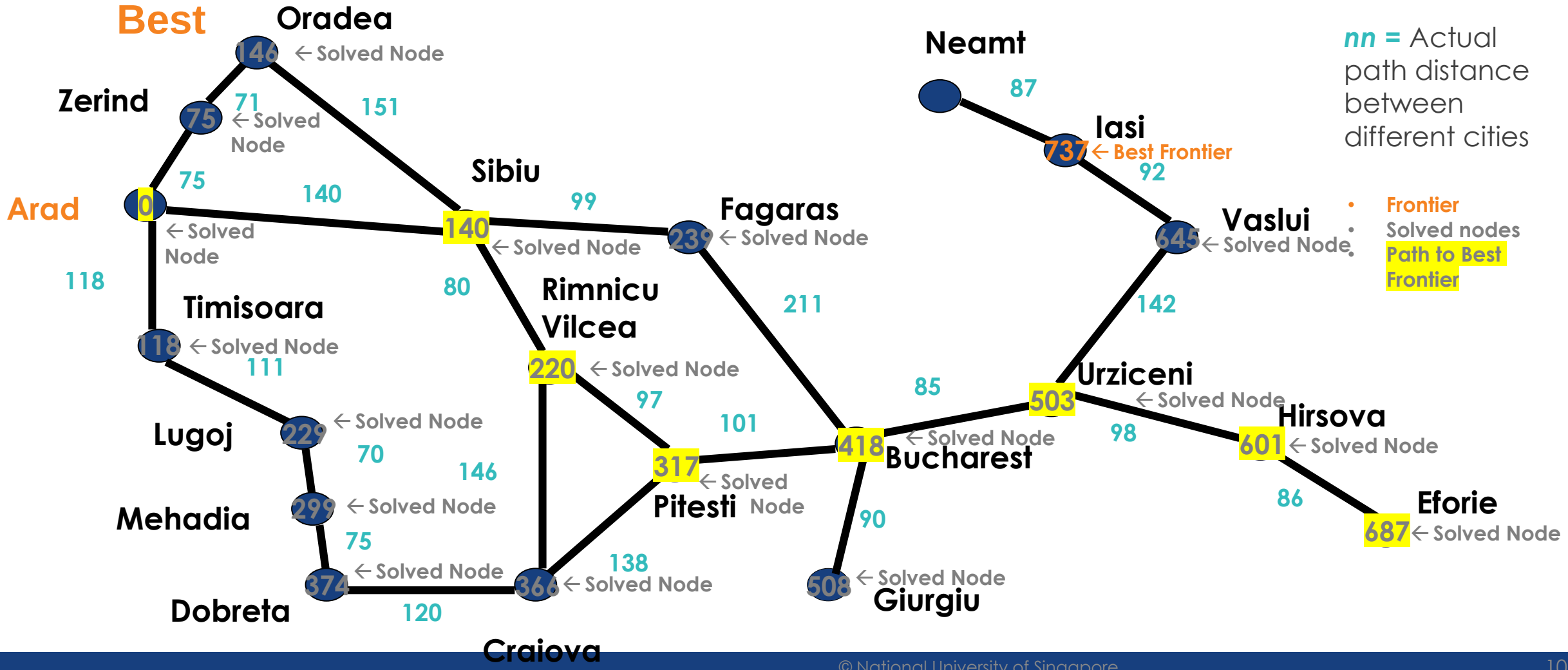
Uninformed Search Techniques (Graph)

Continue for all shortest paths starting from Arad to any other city…



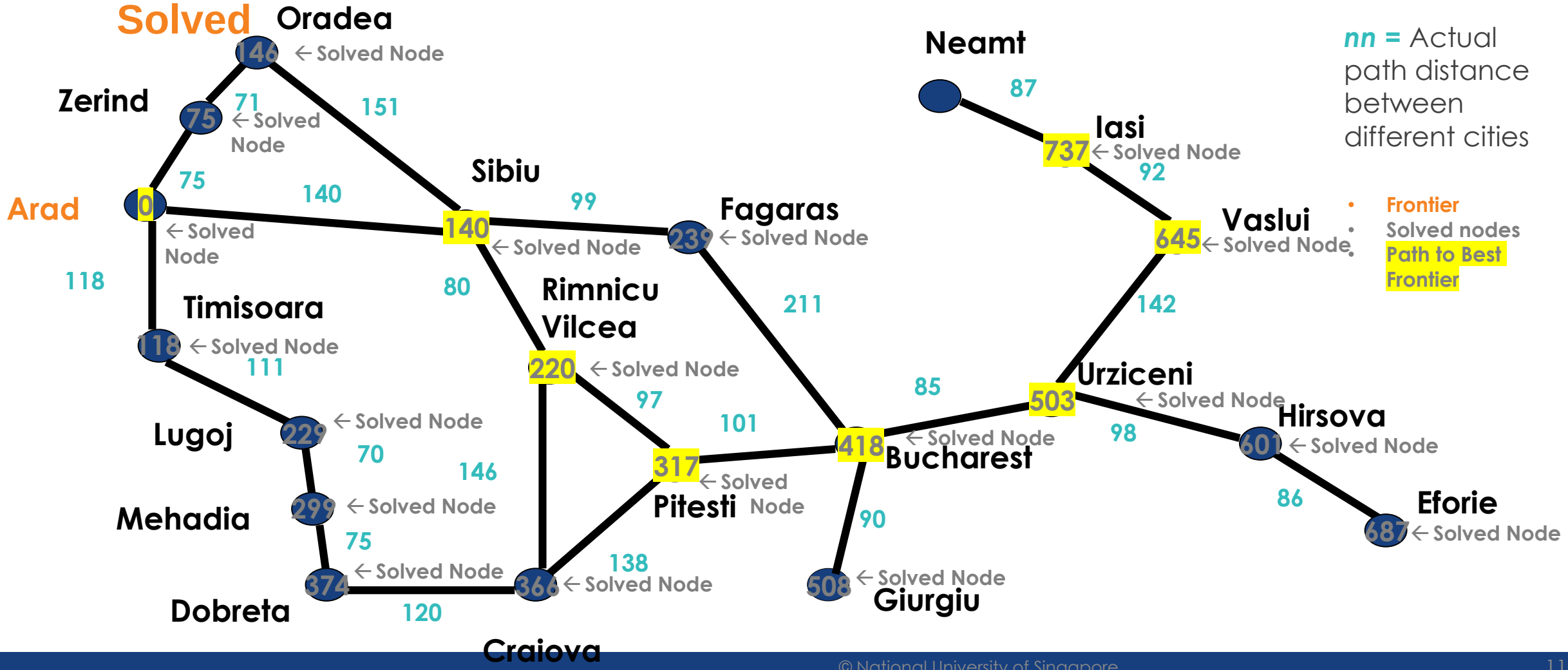
Uninformed Search Techniques (Graph)

Continue for all shortest paths starting from Arad to any other city…



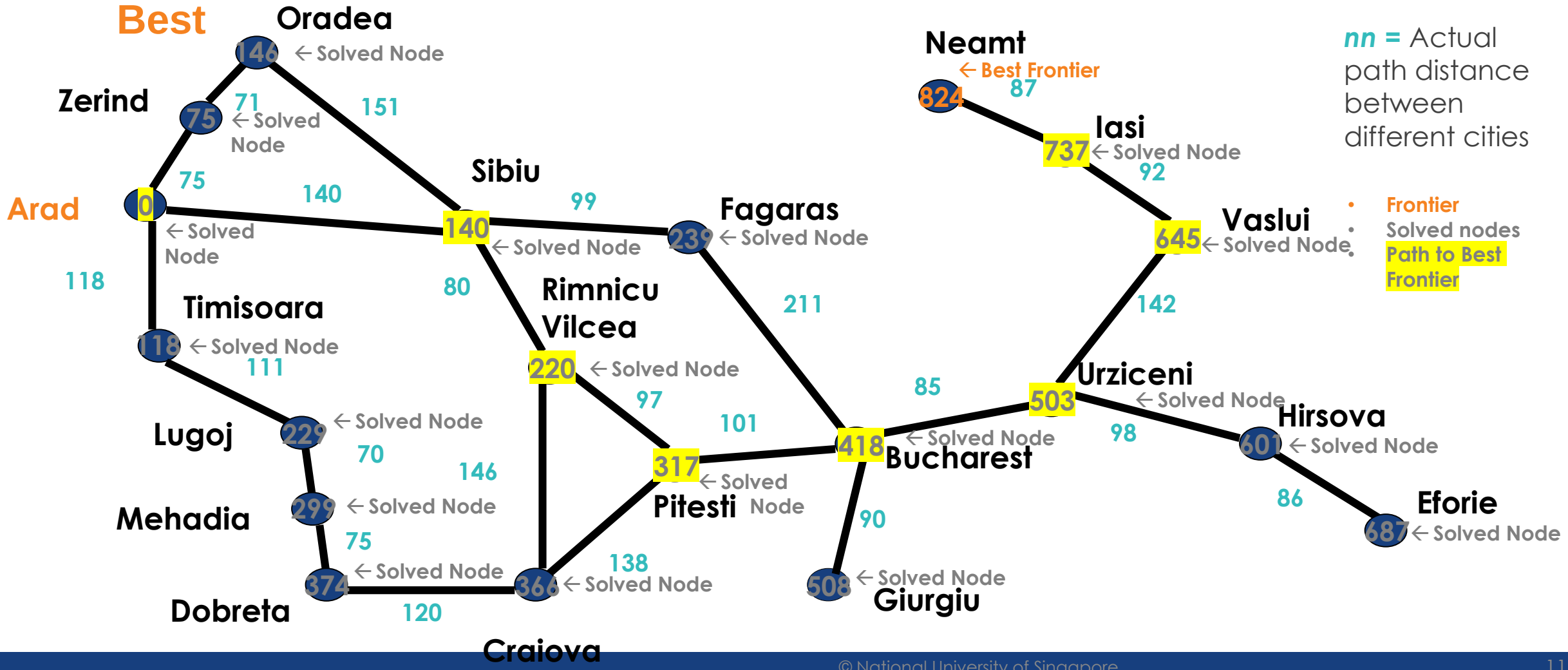
Uninformed Search Techniques (Graph)

Continue for all shortest paths starting from Arad to any other city…



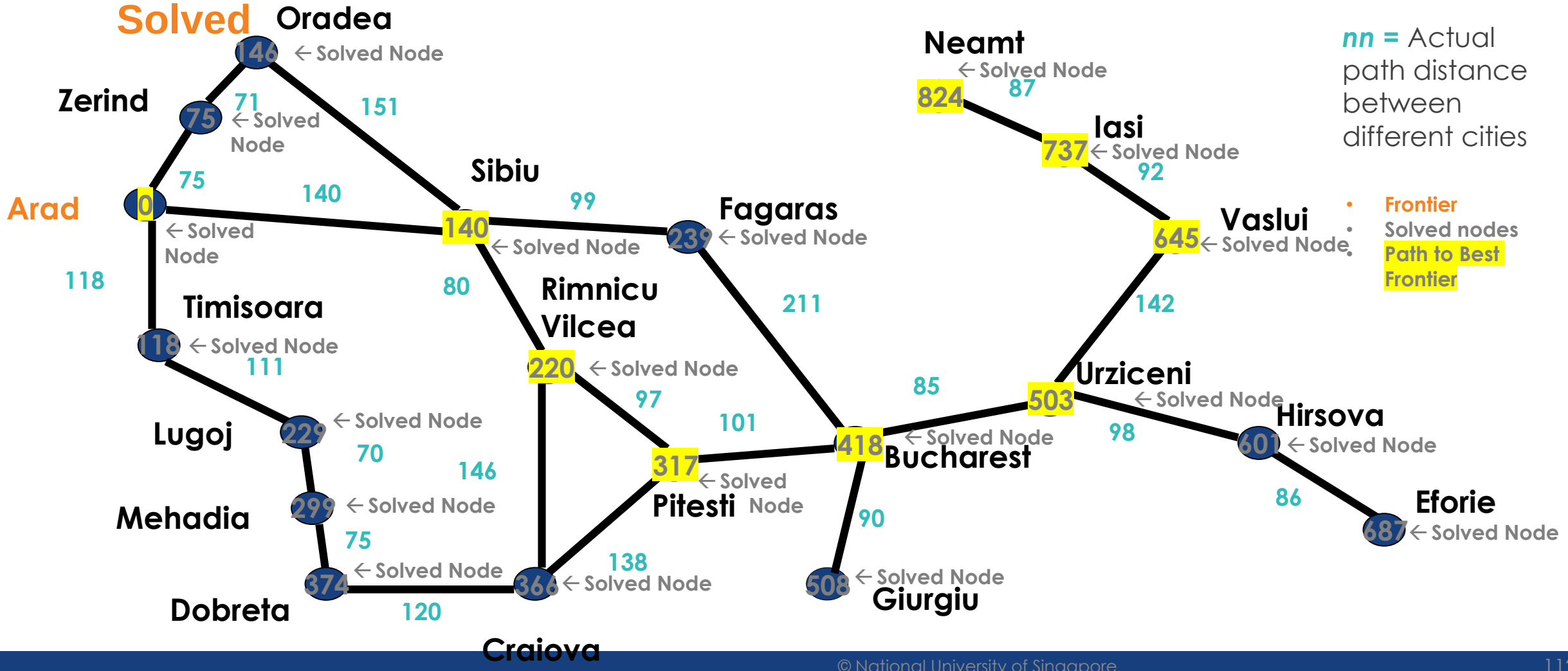
Uninformed Search Techniques (Graph)

Continue for all shortest paths starting from Arad to any other city…



Uninformed Search Techniques (Graph)

Continue for all shortest paths starting from Arad to any other city…



Uninformed Search Techniques (Graph)

iGraph for graph analysis. Supports both R and python

Various graph algorithms:

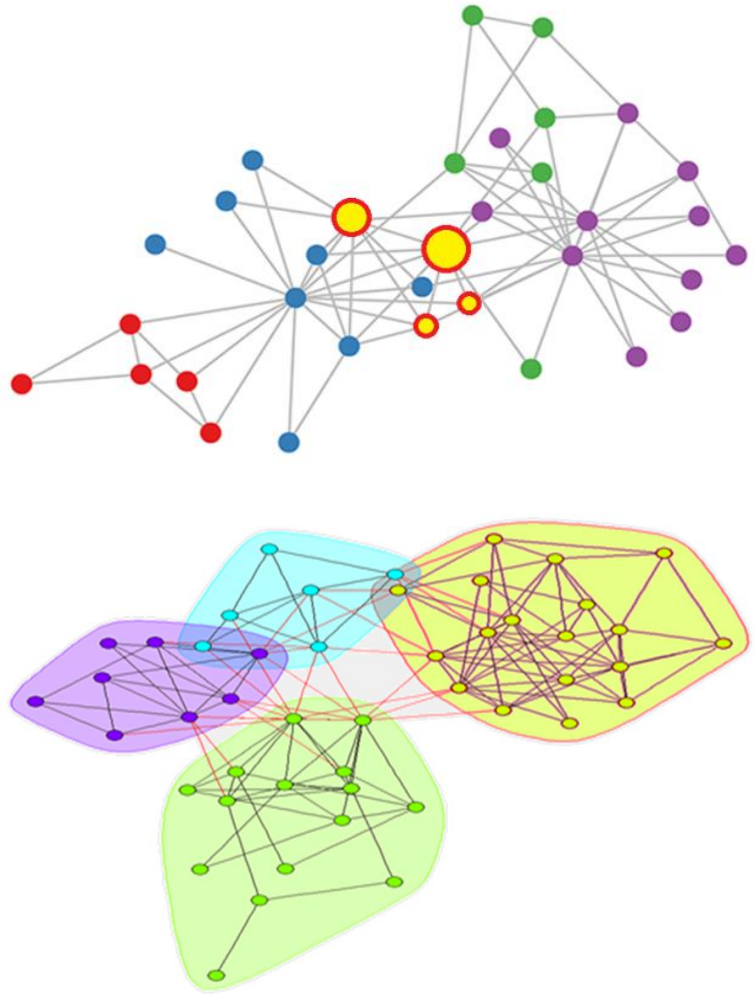
- **Pathfinding:** Paths are fundamental to graph analytics and algorithms, so this is where we'll start our chapters with specific algorithm examples. Finding shortest paths is probably the most frequent task performed with graph algorithms and is a precursor for several different types of analysis. The shortest path is the traversal route with the fewest hops or lowest weight. If the graph is directed, then it's the shortest path between two nodes as allowed by the relationship directions.

Finding structurally most important nodes. example links to other important nodes.

- **Centrality:** Centrality is all about understanding which nodes are more important in a network. But what do we mean by importance? There are different types of centrality algorithms created to measure different things, such as the ability to quickly spread information versus bridging distinct groups.

- **Community Detection:** Connectedness is a core concept of graph theory that enables a sophisticated network analysis such as finding communities. Most real-world networks exhibit substructures (often quasi-fractal) of more or less independent subgraphs.

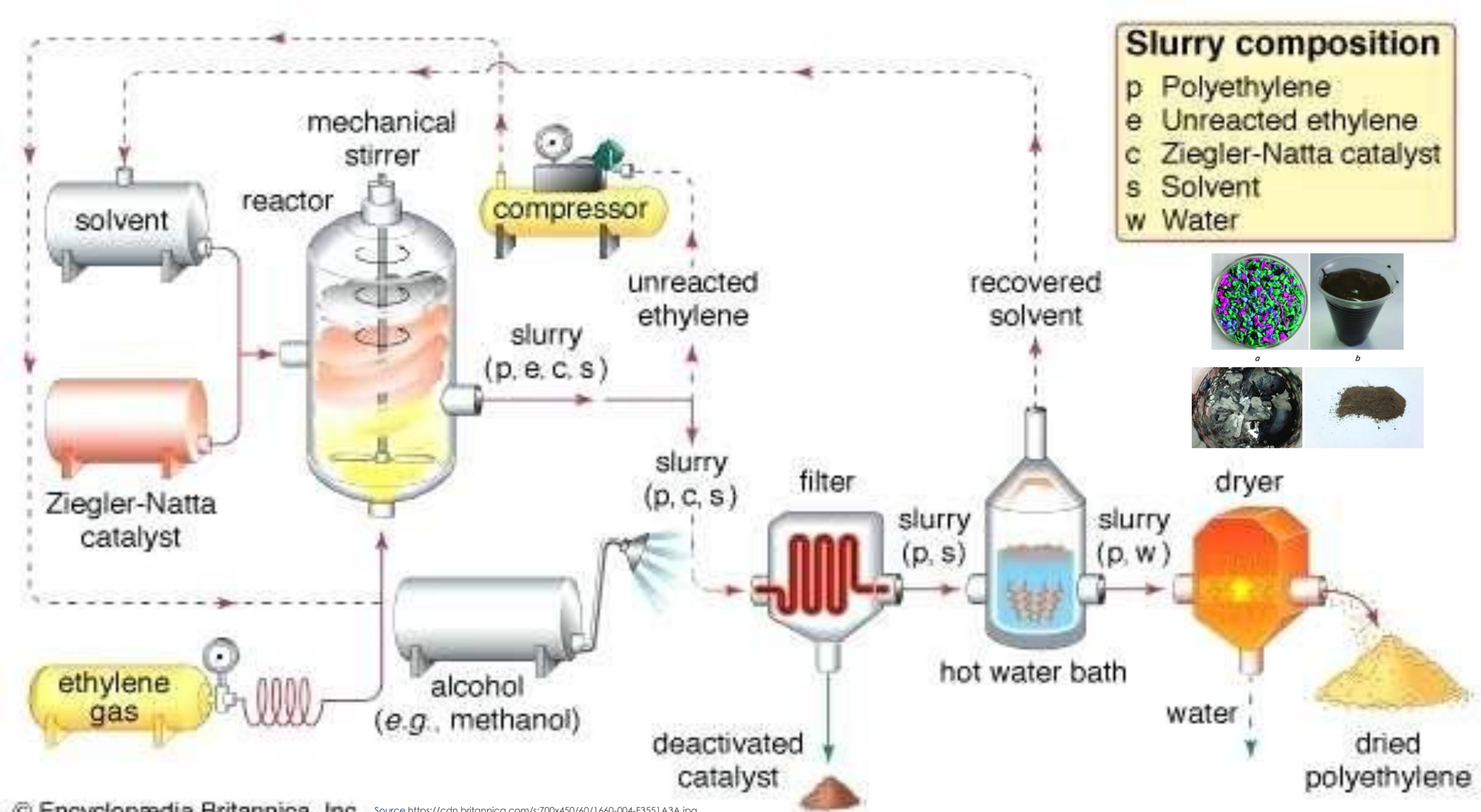
To find substructure of communities



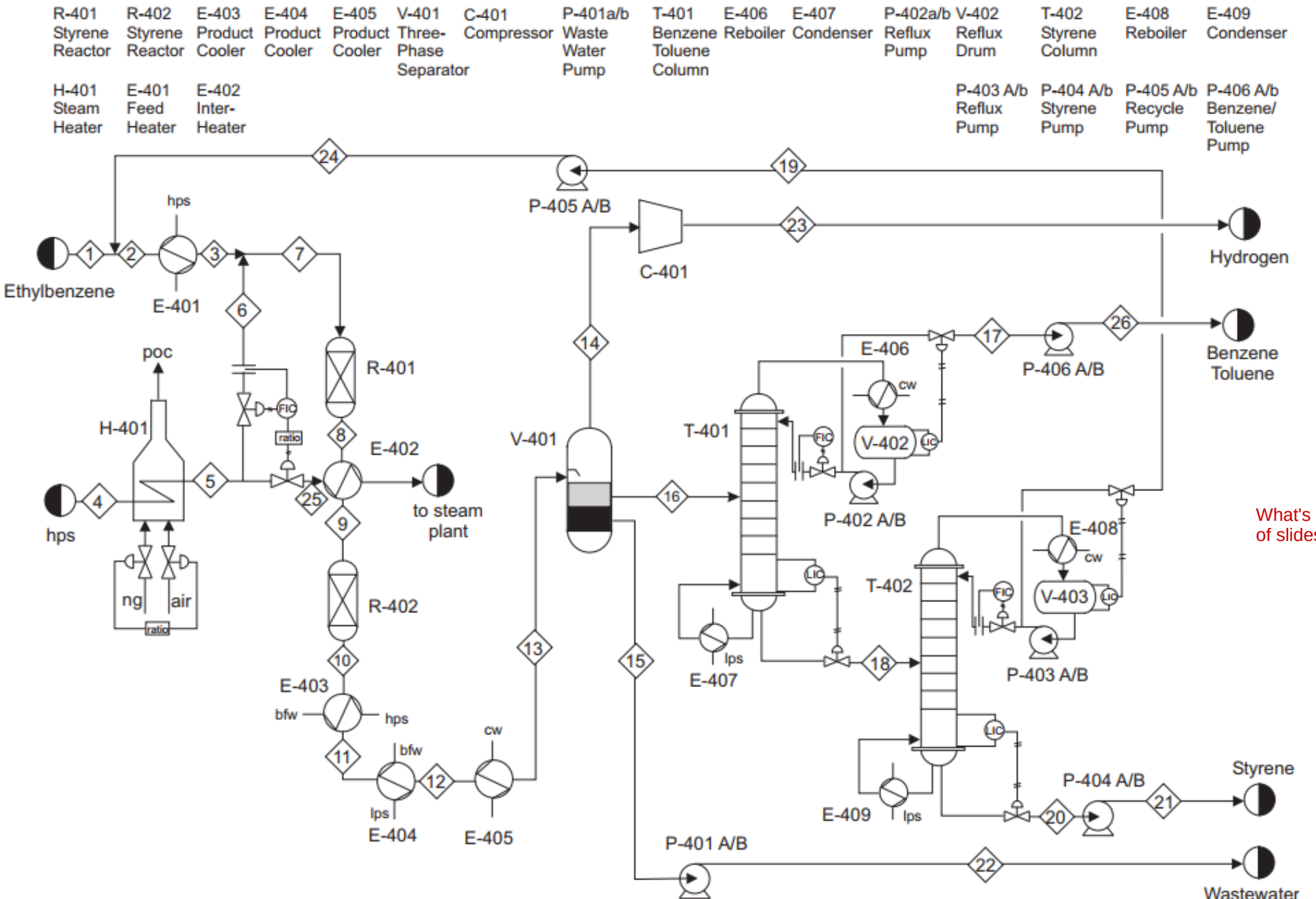
Source: Graph Algorithms: Practical Examples in Apache Spark and Neo4j Book by Amy E. Hodler and Mark Needham

Hybrid Reasoning System for Predictive Maintenance


http://www.revue...fr/photo/art/grande/8412702-13201261.jpg?v=1463652853



Source https://cdn.britannica.com/s:700x450/60/1660-004-F3551A3A.jpg
Source https://www.researchgate.net/profile/Nadegda_Rykusova/publication/332931678/figure/fig1/AS:756211943227392@1557306385389/Composite-materials-a-high-density-polyethylene-waste-b-waste-mud-c-dried-mud-d.png



What's the purpose for this 3 set of slides?

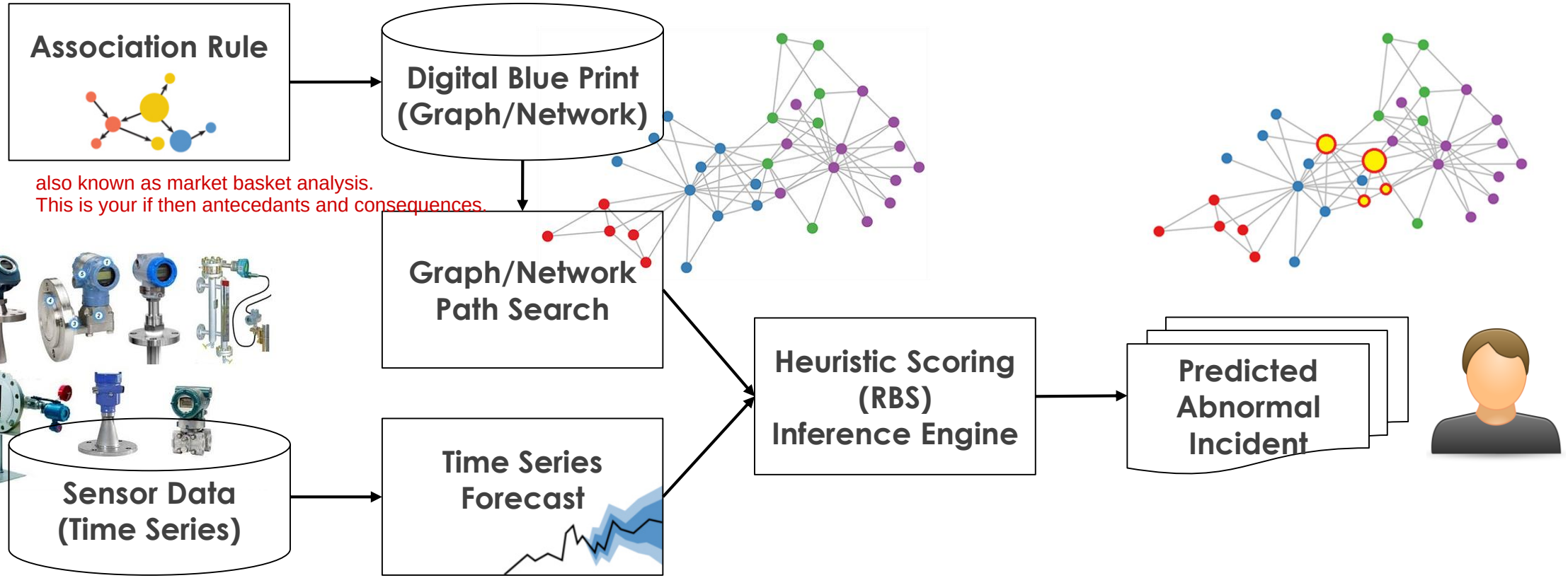
https://d2vlcm61l7u1fs.cloudfront.net/media%2F5ef%2F5ef158e9-bf05-47fa-b20c-fbd72b122e61%2FphpA1w6Ci.png

Hybrid Reasoning Systems

Co-operating Experts

- **Use case: Plant Abnormal Situation Prediction (Predictive Maintenance)**

  Sub-systems: Time Series Forecast, Association Rule, Graph Search, Heuristic Score



https://4.imimg.com/data4/QI/WE/MY-24712837/different-types-of-level-transmitters-500x500.jpg
http://tkipf.github.io/graph-convolutional-networks/images/karate.png
https://www.microsoft.com

1.2 Reasoning Using Uninformed Search

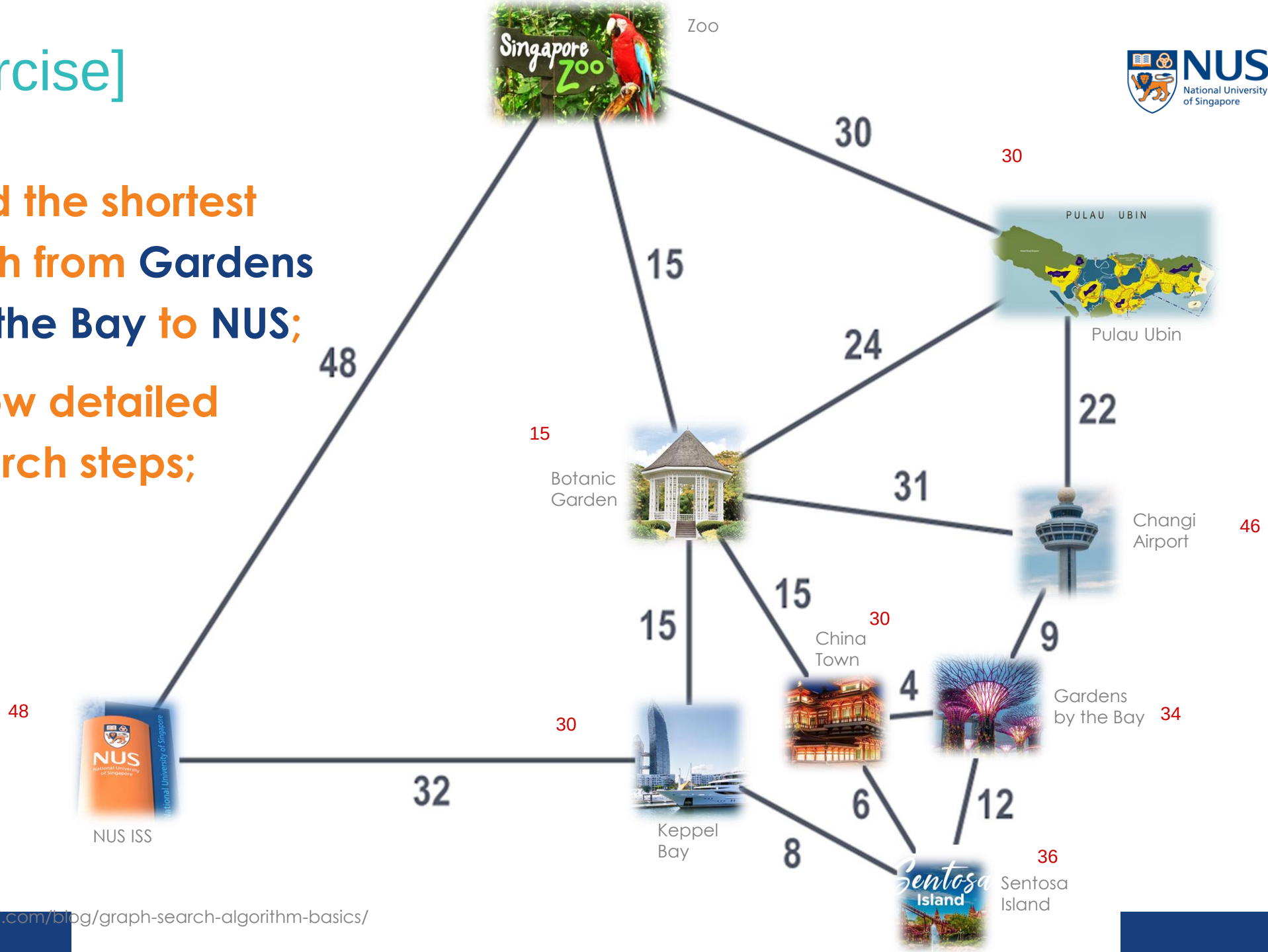
1.2.1 Uninformed Search Techniques (Tree)

1.2.2 Uninformed Search Techniques (Graph)

1.2.3 Exercise

[Exercise]

- **Find the shortest path from Gardens by the Bay to NUS;**
- **Show detailed search steps;**



: https://neo4j.com/blog/graph-search-algorithm-basics/

1.3 Search Representation [Workshop]

1.3.1 Tree Search & Graph Search

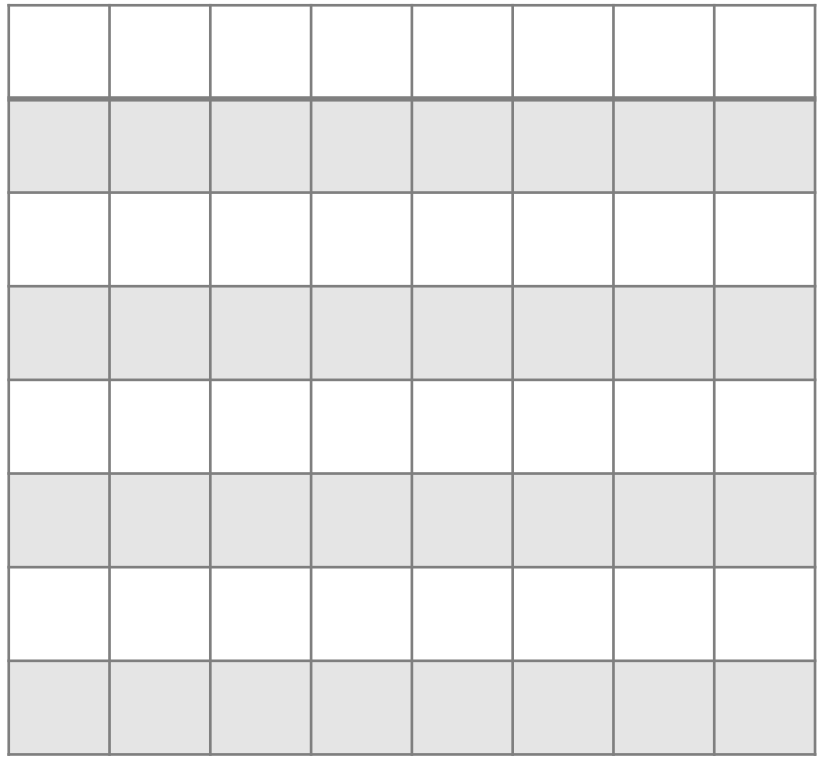
1.3.2 Knowledge Graph Reasoner

1.3.3 Workshop Submission

Tree Search & Graph Search

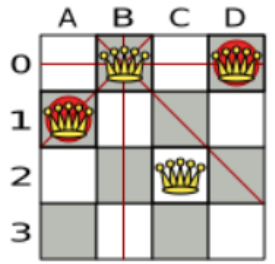
Tree representation & search

Goal: Find all solution(s) of eight queen problem using tree search.

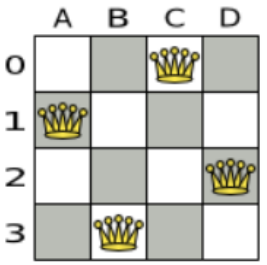


Business Requirements/Constraints:

- **Place eight queens on this chessboard;**
- **No two queens can attack each other;**



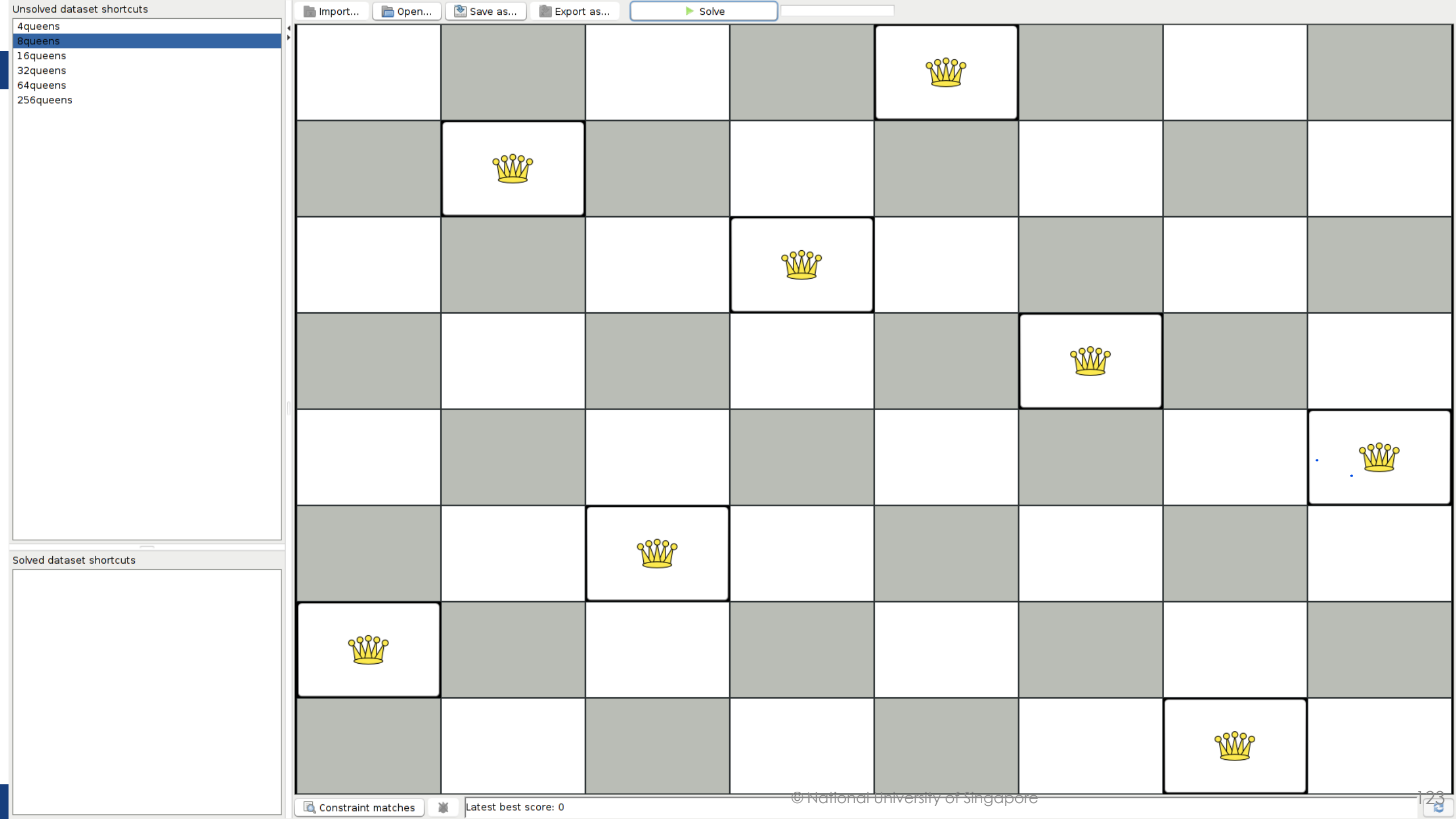
Bad



Good

Reference:

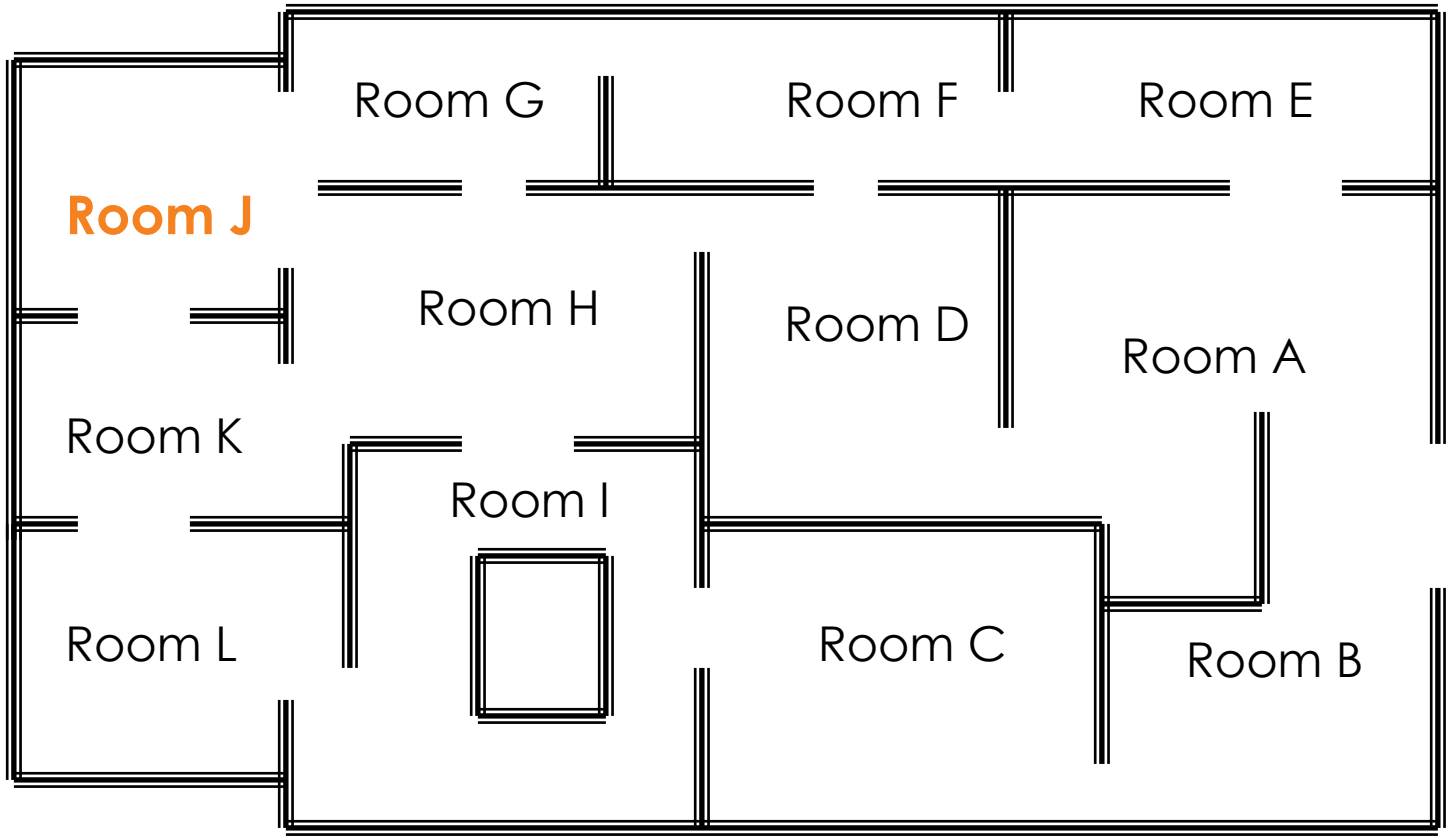
- https://en.wikipedia.org/wiki/Eight_queens_puzzle
- https://developers.google.com/optimization/cp/queens

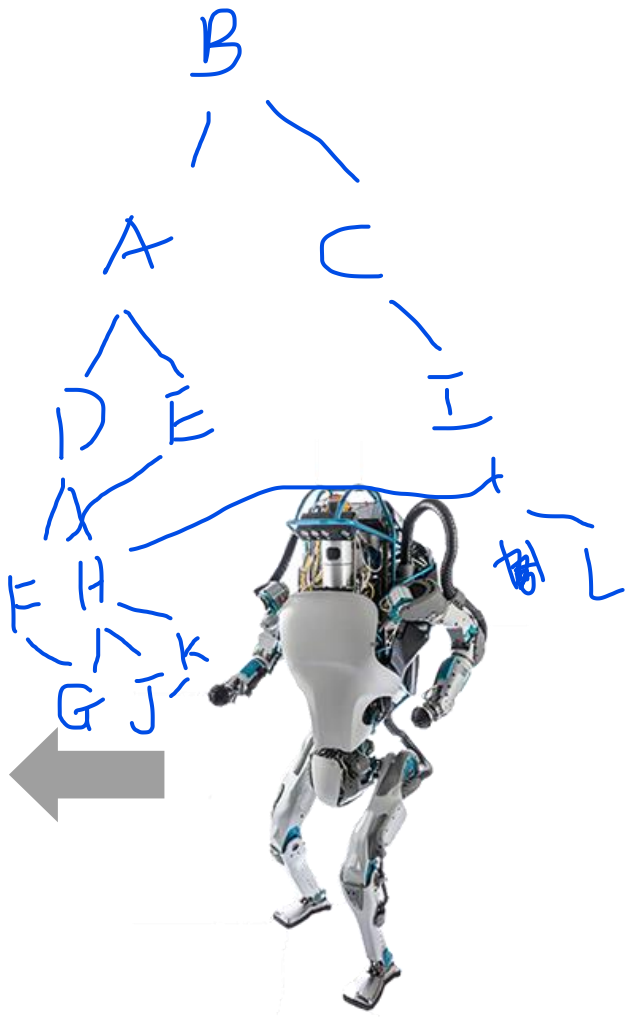


Tree Search & Graph Search

Graph representation & search

- **Robotics: How to rapidly navigate to Room J ?**



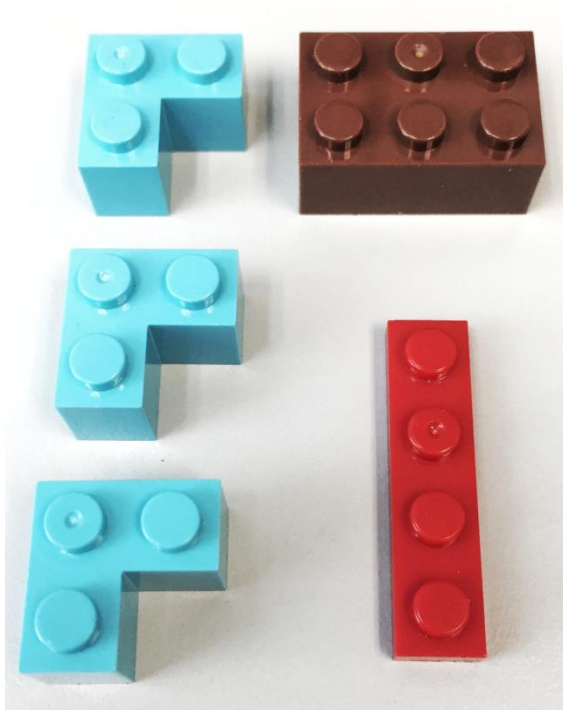


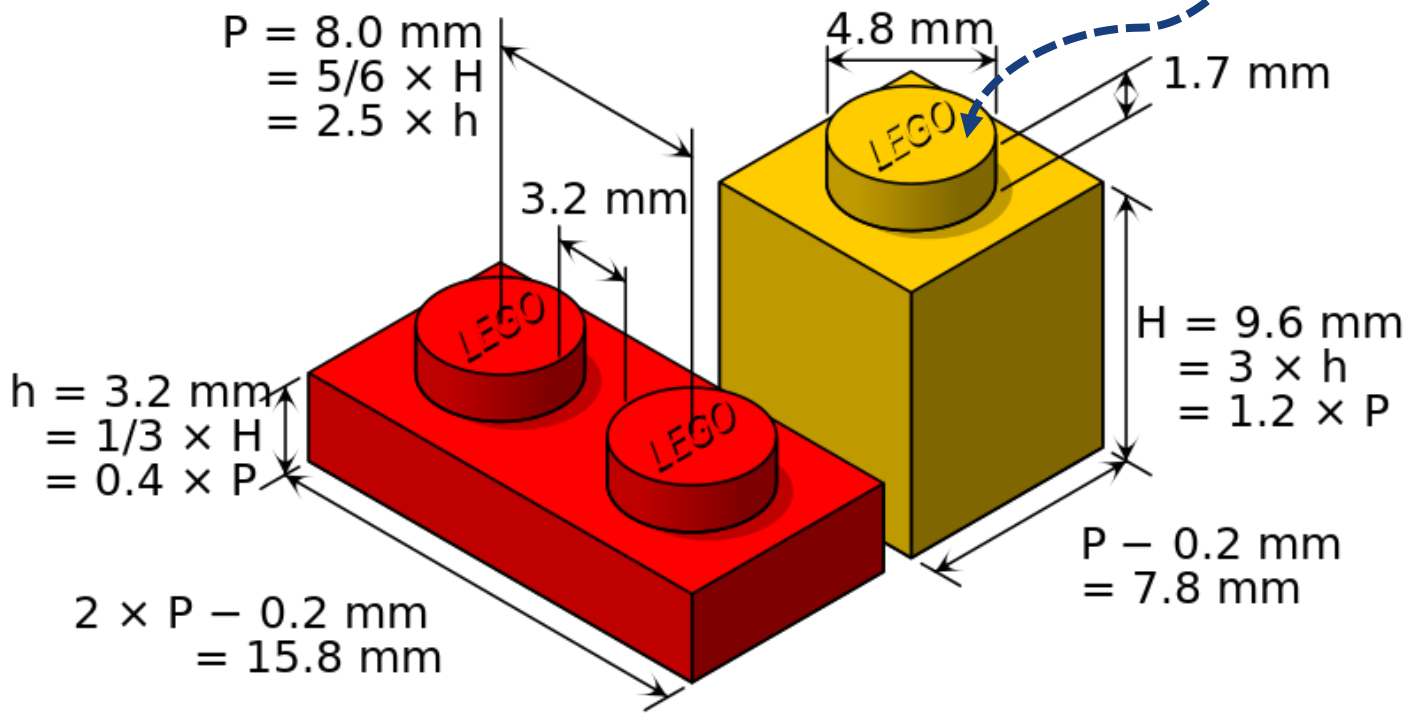
https://static1.squarespace.com/static/57c8a68a20099ef23fb19e90/t/5a32a9804192022be97f9d10/1513269668629/Atlas.png

Tree Search & Graph Search

[Optional]

- **Goal: Find a configuration with minimal total studs: little round bumps.**
- **Enhanced Goal: Lower block height without increasing total studs.**





Source https://www.quora.com/Are-the-dimensions-of-LEGO-bricks-functional-for-recreating-a-Minecraft-environment

1.3 Search Representation [Workshop]

1.3.1 Tree Search & Graph Search

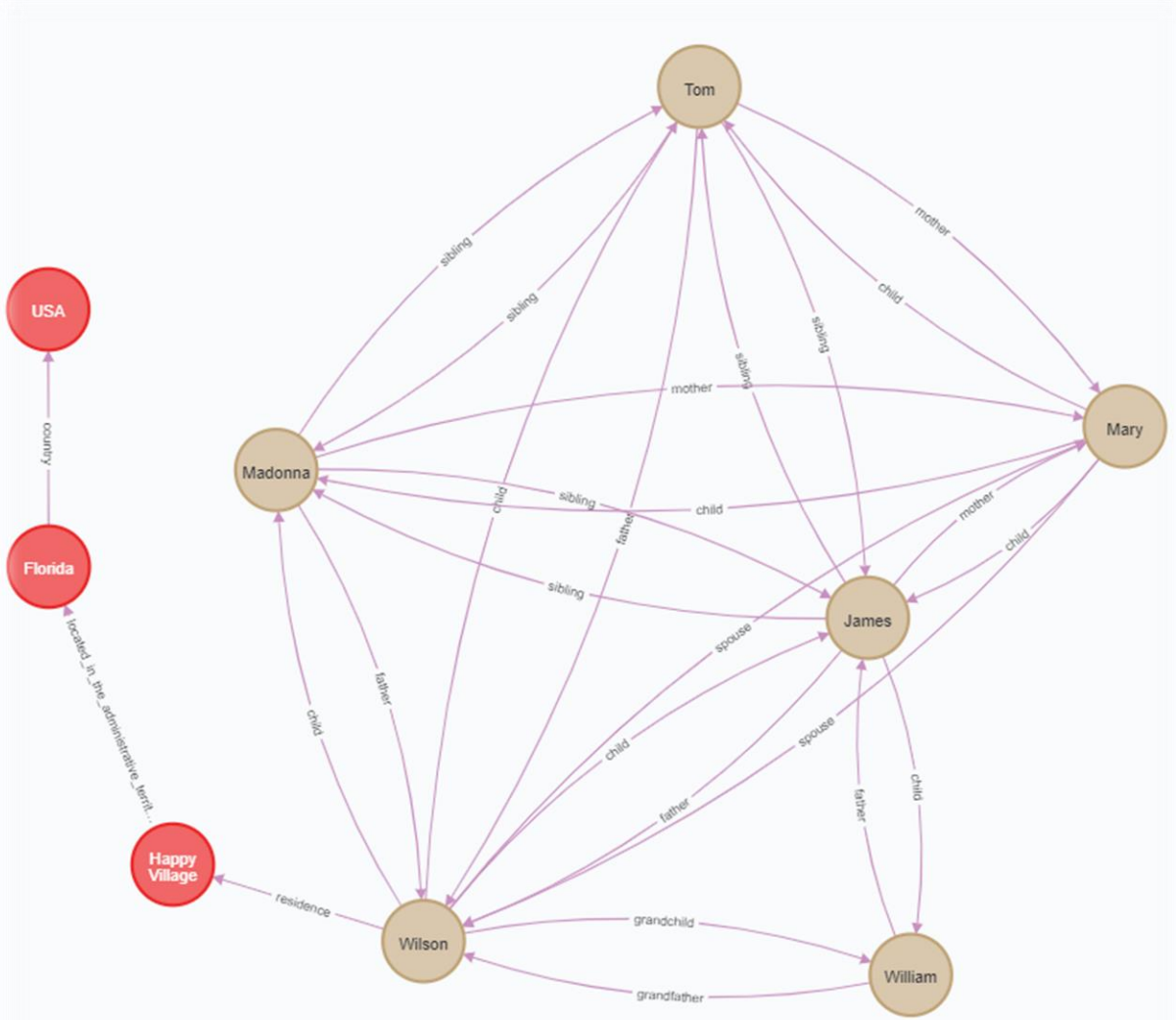
1.3.2 Knowledge Graph Reasoner

1.3.3 Workshop Submission

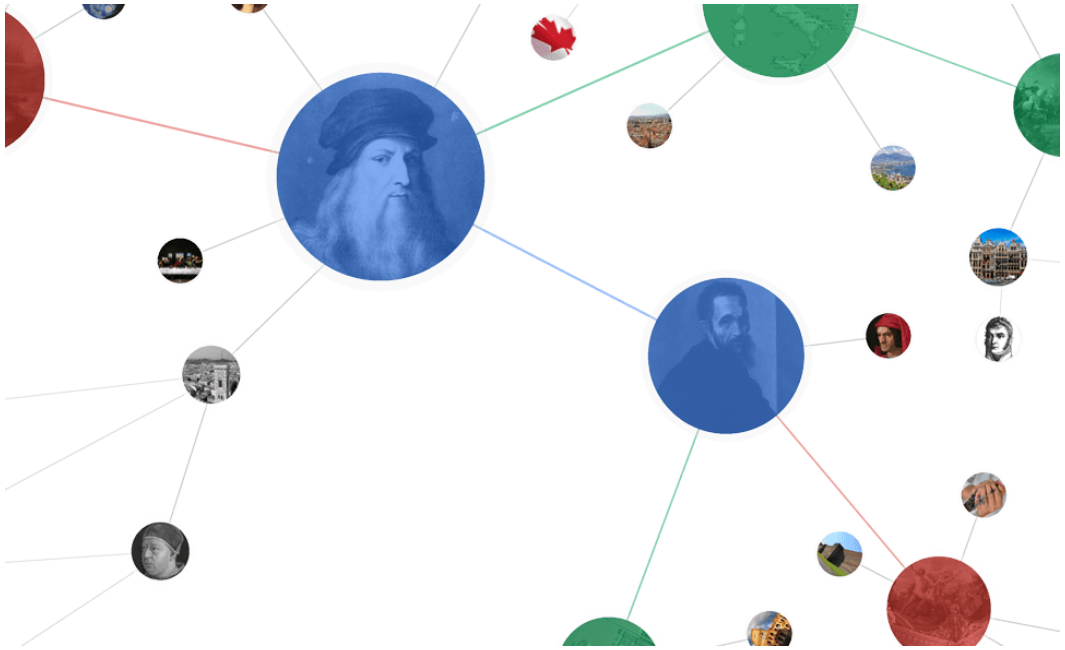
Knowledge Graph Review

- A knowledge graph is a knowledge base that uses a graph-structured data model or topology to store, integrate, represent, and query data/knowledge.

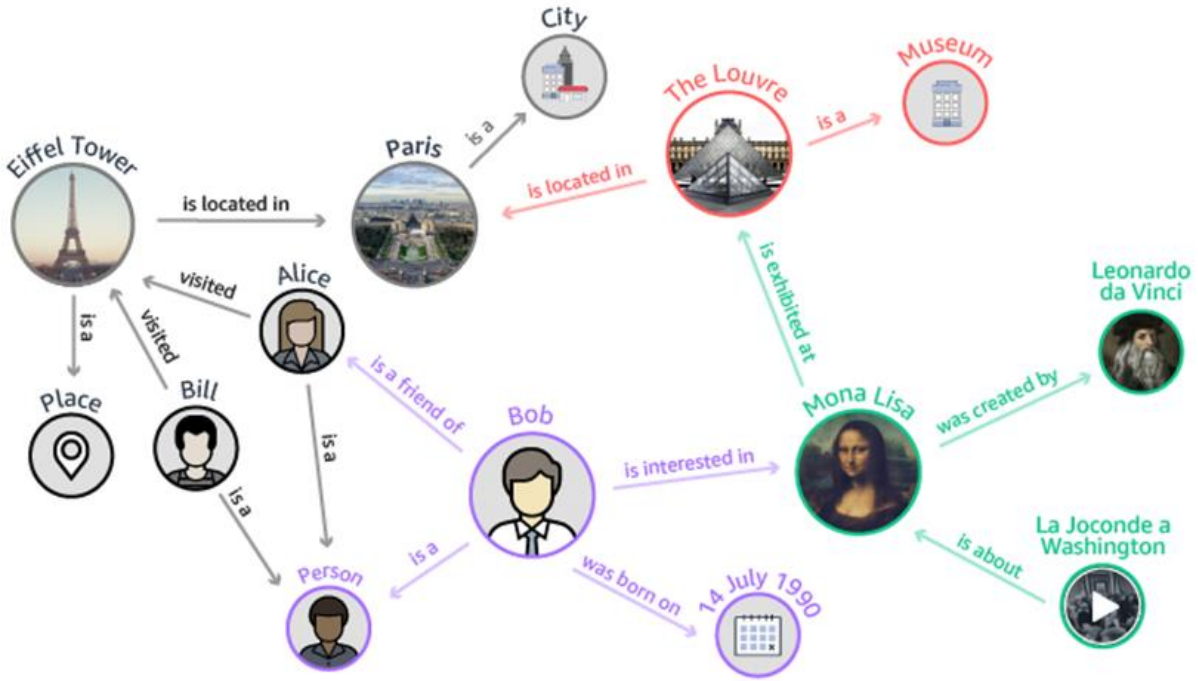
- Normally, the edge/link is relationship(predicate). The node/vertex is entity/instance.



Knowledge Graph Review



Source https://www.tampa-seo.com/wp-content/uploads/static-graph.png



Source
https://d1.awsstatic.com/products/Neptune/knowledge_graph.b0e9408219d92f2ca3c7a05cccf9a5a72e34ddbd.png

Knowledge Graph Discovery & Reasoning

- **Learning Objectives**
  - **Learn build knowledge graph (KG) in graph database**
  - **Learn do inference and query in knowledge graph**
- **Package to use**
  - **Spacy, neo4j, openNRE**

Name relationship entity

- **Technique**
  - **Entity & relationship extraction**
  - **Knowledge graph reasoning**
- **Case/Scenario**
  - **A family relationship tutorial to learn how to build KG and do inference**
  - **“Animal Farm” for workshop**
- **Requirement**
  - **Familiar with python**

What is OpenNRE?

- **In SpaCy, we call 'nlp tool' to extract the entity from a sentence**

```
raw_sentence = 'Sam lives in Singapore'
nlp = spacy.load('en_core_web_lg')

token_sentence = nlp(raw_sentence)
for i, ent in enumerate(token_sentence.ents):
    print(i,':',ent.text, ent.label_)
```

```
0 : Sam PERSON
1 : Singapore GPE
```

- **However, how to get the relationship(predicates) between two entities?**
- **We need relationship-extraction function. That's when openNRE comes. OpenNRE is a pre-trained model model to extract relationship between two entities from a sentence.**

Installation

1. Type in pip install spacy in your anaconda environment;
2. Install the neo4j following the guide of link;
3. Type in pip install neo4j in your anaconda environment (This package is used to control neo4j using python);
4. Install the OpenNRE following the guide of link;

Note: All the installation should be under your anaconda environment

Create New Graph DB in Neo4j

If necessary, in Neo4j browser/desktop:

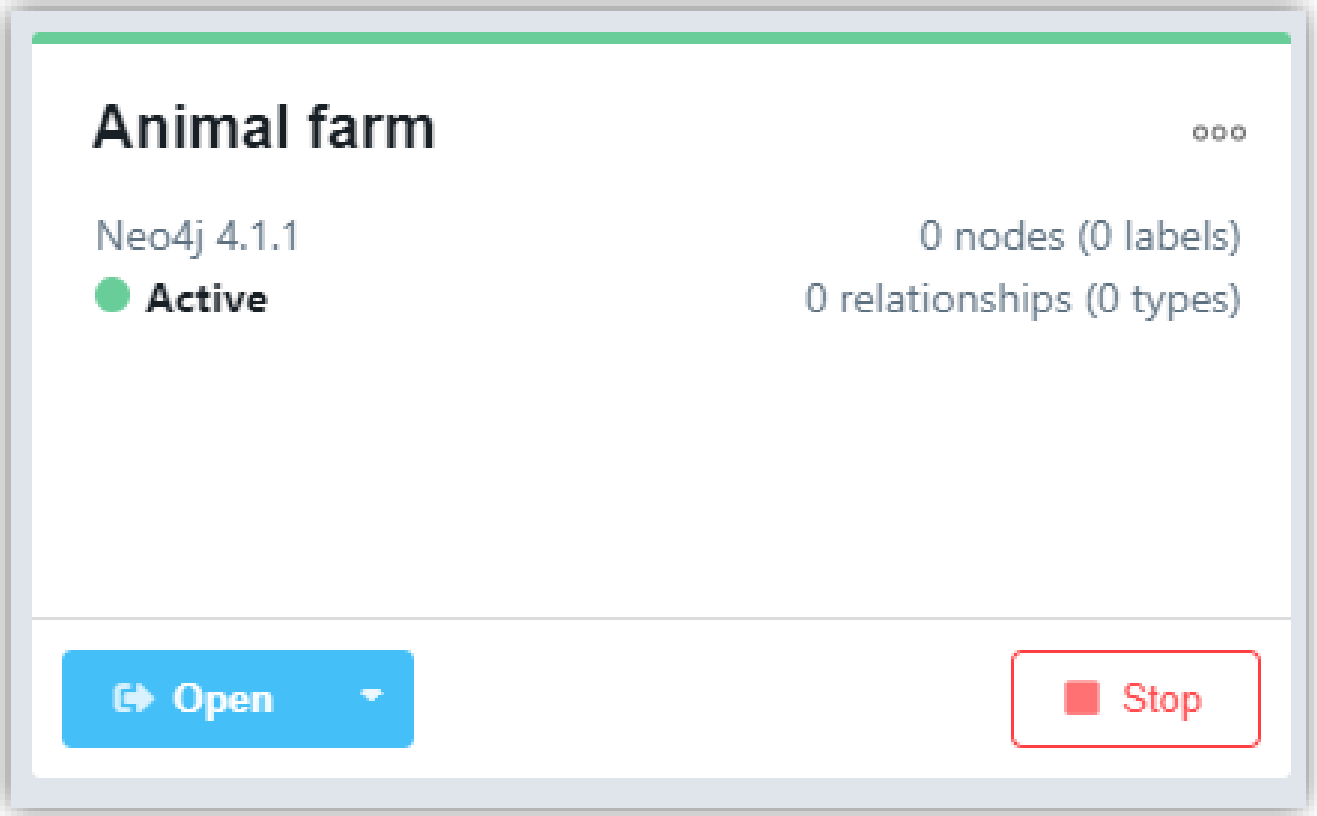
-> click **Add Database**

-> Create **Local** Database

- name '**Animal farm**'
- password '**ai-user**'

-> click Create

-> click Start



KG Construction Tutorial

- **./corpus/Family.txt contains textual resources/sentences about the relationships between some persons.**

Happy Village is located in Florida, USA.
Mr. Wilson is a well-known local silversmith in Happy Village.
He is very very old.
When Mr. Wilson was young, he came to Happy Village to seek refuge from a war.
Mr. Wilson found his wife Mary here and raised three children.
James is the elder son of Wilson.
Tom is the younger son of Wilson.
Mr. Wilson also has a daughter named Madonna.
William is James's son.

KG Construction Flow

1. Extract Entities

```
for ent in doc.ents:
    if ent.label_ in ['PERSON', 'GPE'] and ent.text not in names_of_entities:
        names_of_entities.append(ent.text)
        entities.append(ent)
```

2. Generate Nodes

```
for ent in entities:
    if exist_ent.get(ent.text) is None:
        exist_ent[ent.text] = ent.text
        query = (
            "MERGE (node: "+ent.label_+" {name: $name})"
            "RETURN node"
        )
```

3. Extract Relationships

```
for i in range(len(entities)):
    for j in range(i + 1, len(entities)):
        text_i = entities[i].text
        text_j = entities[j].text
        loc_h = sentence.find(text_i)
        loc_t = sentence.find(text_j)
        result = model.infer({'text': sentence, 'h': {'pos': (loc_h, loc_h + len(text_i))},
                              't': {'pos': (loc_t, loc_t + len(text_j))}})
```

4. Generate Relationships

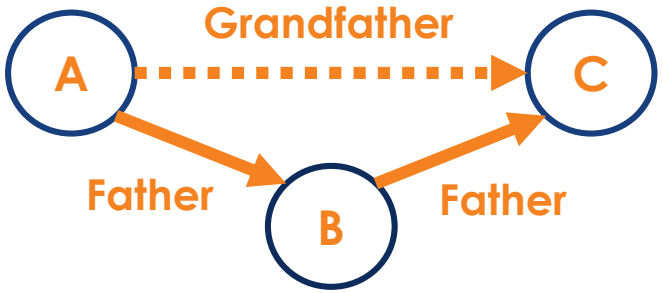
```
if record not in exist_relationship:
    exist_relationship.append(record)
    if record[3] > 0.8:
        query = (
            "MATCH (n1 {name: $name1})"
            "MATCH (n2 {name: $name2})"
            "MERGE (n1) - [r:"+record[2]+"] -> (n2)"
            "RETURN n1, n2, r"
        )
```

KG Construction: Extraction

- **In KG, we establish relationships and nodes. These information can be directly extracted from the corpus. E.g. from two raw sentences**
  - William is James's son.
  - James is the elder son of Wilson.
- **The openNRE can extract 'Father' relationship:**
  - William – HasFather → James & James – HasFather → Wilson
- **However, some relationships cannot be directly extracted from raw sentences** (due to ? Non-existence lah)**. E.g. how to generate/infer the 'Grandfather' relationship?**
  - William – HasGrandfather → Wilson

KG Construction: Extension (inference)

- **We can add inference rules (additional knowledge) to automatically extend the knowledge graph.**
- **E.g. WHEN A HasFather B AND B HasFather C THEN A HasGrandfather C**

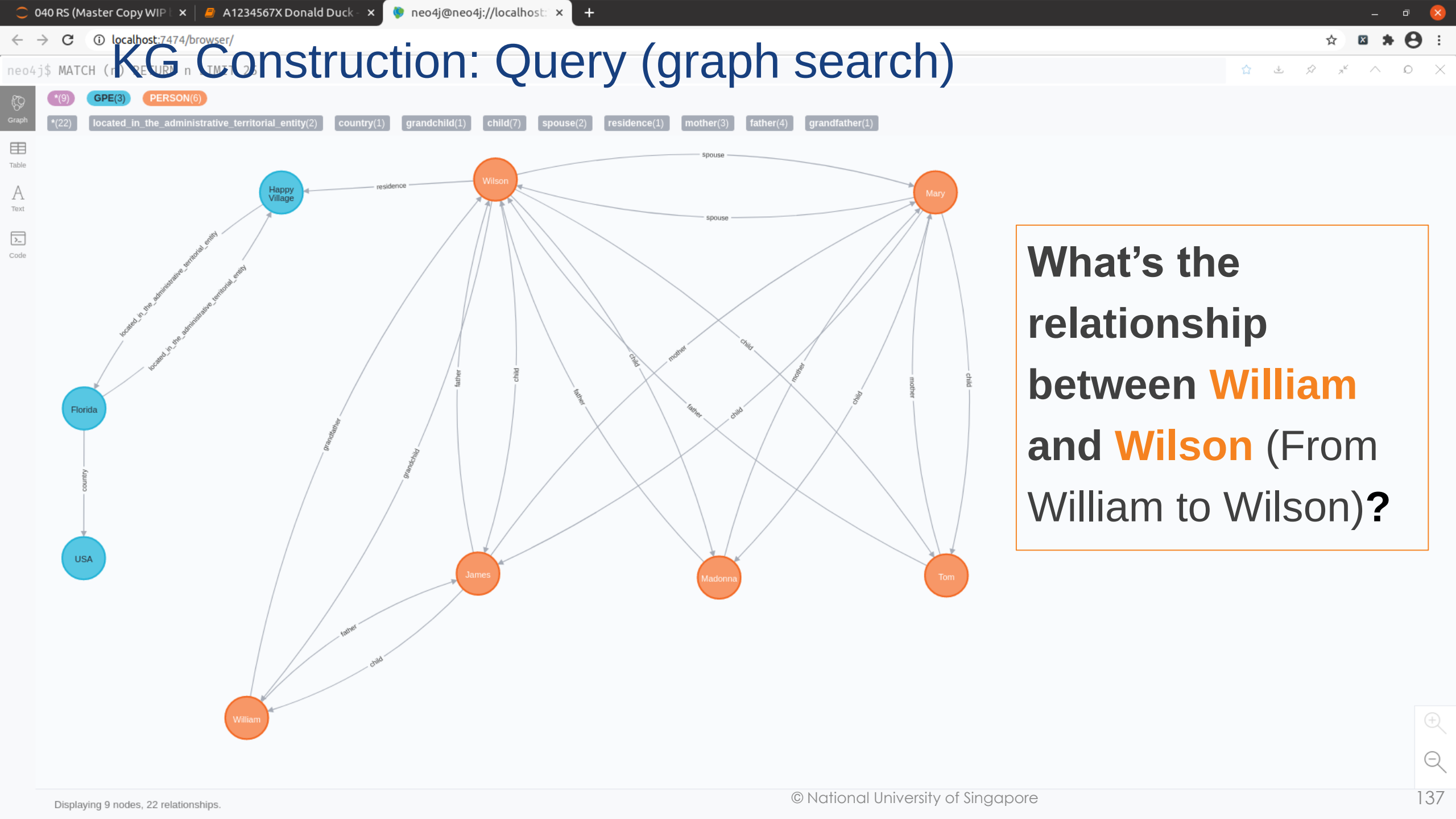



```
# Rule 3: Grandfather relationship/link

# WHEN:
# (n1:PERSON)-[r:father]->(n2:PERSON)  # n1 has a father : n2
# (n2:PERSON)-[r2:father]->(n3:PERSON) # n2 has a father : n3
# THEN:
# (n1)-[r3:grandfather]->(n3) # n1 has a grandfather : n3
# (n3)-[r4:grandchild]->(n1)  # n3 has a grandchild  : n1

query = (
        "MATCH (n1:PERSON)-[r:father]->(n2:PERSON)-[r2:father]->(n3:PERSON)"
        "MERGE (n1)-[r3:grandfather]->(n3)"
        "MERGE (n3)-[r4:grandchild]->(n1)"
        "RETURN n1, n3"
)
```

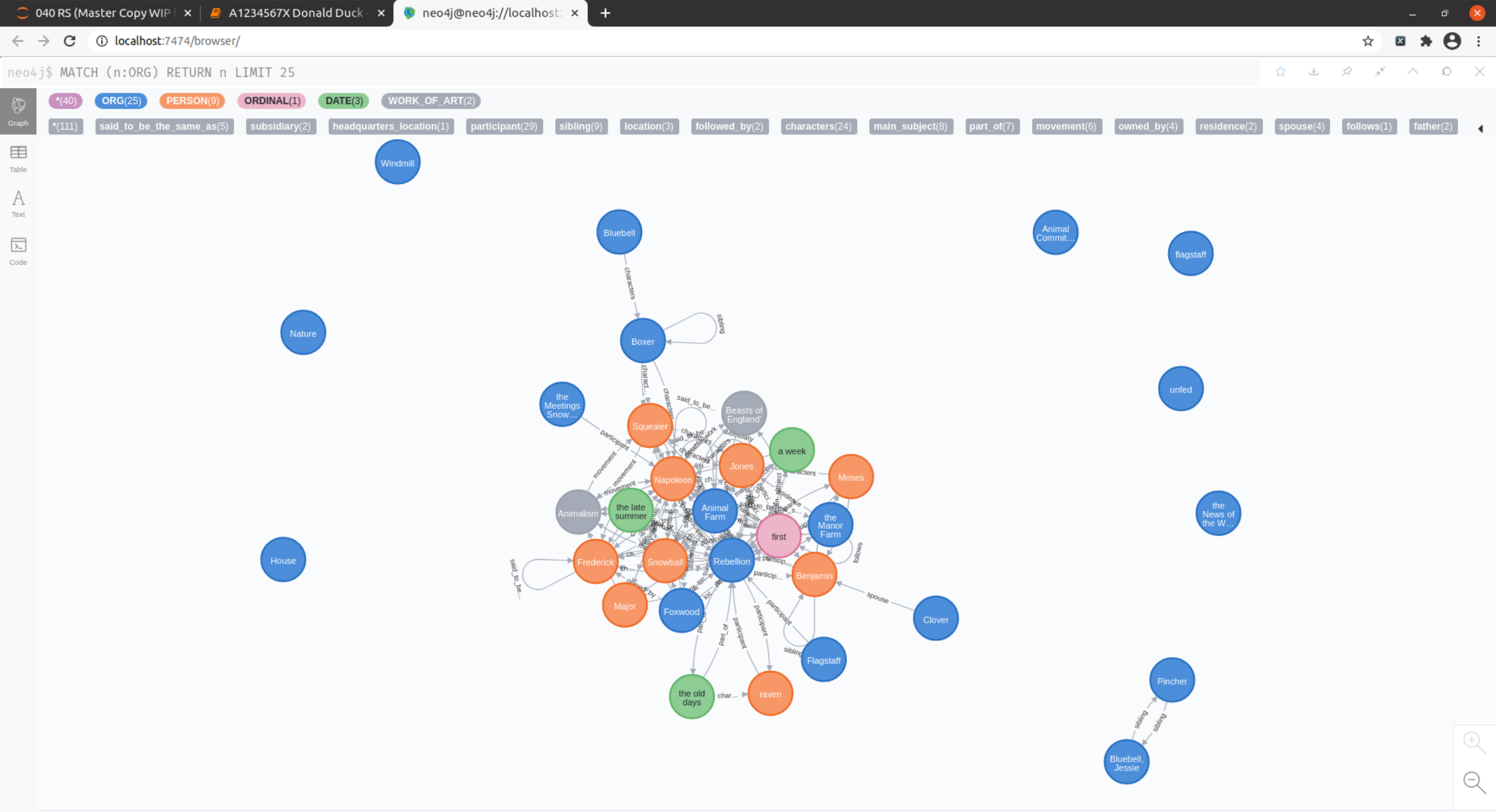
KG Construction: Query (graph search)



What's the relationship between William and Wilson (From William to Wilson)**?**

Workshop

- **Construct knowledge graph based on corpus/novel <Animal Farm> by George Orwell;**
- **Sample and observe the generated relationships, are they reasonable? Can you optimize/ modify some relationshps, e.g. many suspicious ‘spouse’ relationships?**
- **Then, add two reasonable inference rules to extend/modify the Animal Farm KG;**
- **Finally, using KG to query entity relationships and show the result, e.g. what is the relationship between pig ‘Snowball’ and horse ‘Boxer’;**



Workshop Submission

- **Naming convention: StudentID YourFullName, e.g. A1234567X Donald Duck – sln – KG Reasoner.ipynb/zip**
- **Use zip to a single file, then rename, if you plan to submit multiple files.**

References

1. OpenNRE: thunlp/OpenNRE: An Open-Source Package for Neural Relation Extraction (NRE) (github.com)

2. Usage of spacy: Architecture – SpaCy API Documentation

3. <Animal Farm> Character List: https://www.sparknotes.com/lit/animalfarm/characters/

1.3 Search Representation [Workshop]

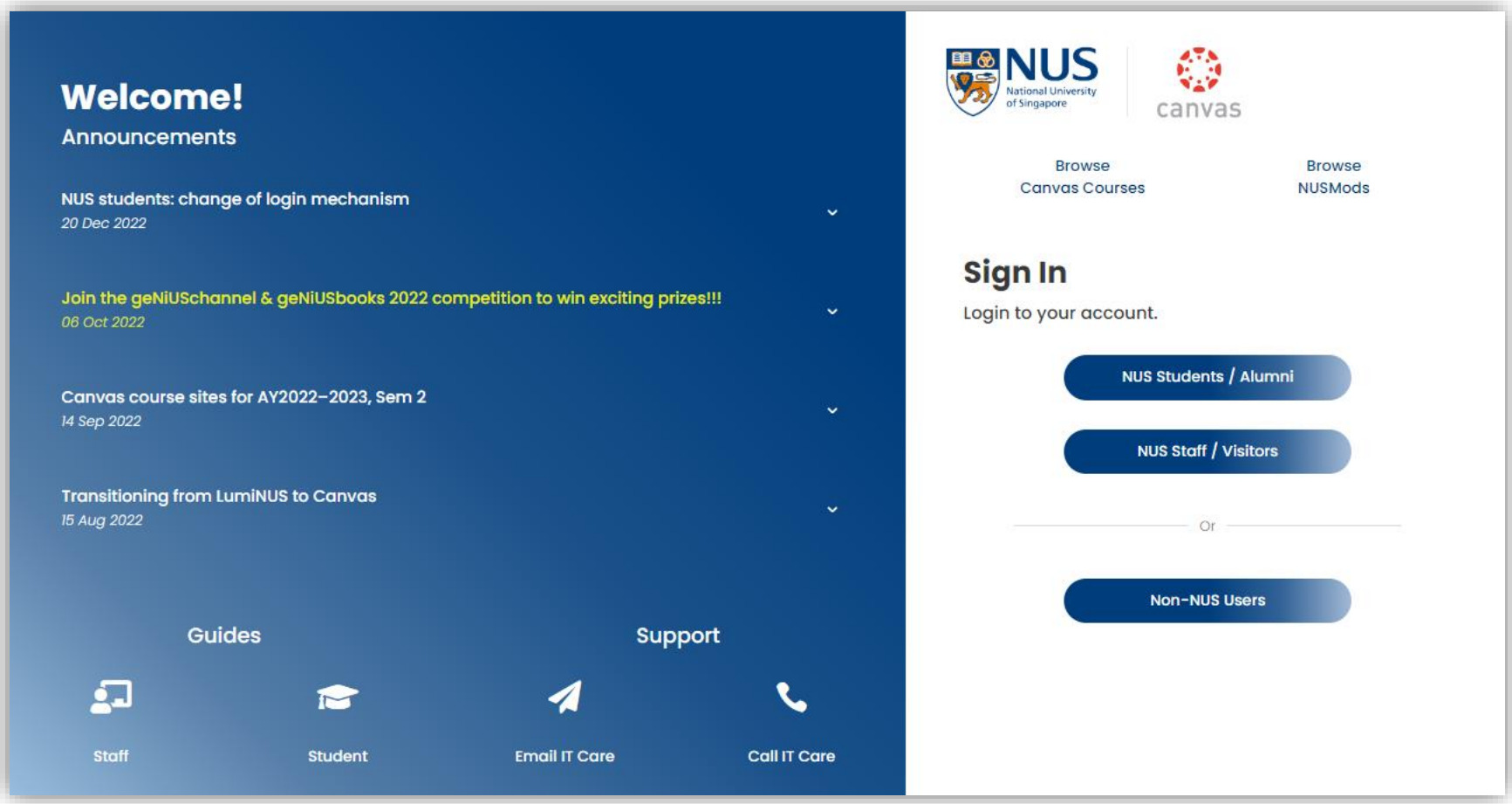
1.3.1 Tree Search & Graph Search

1.3.2 Knowledge Graph Reasoner

1.3.3 Workshop Submission

Workshop Submission

- **Naming convention: StudentID YourFullName**
- **Use zip to a single file, then rename, if you plan to submit multiple files.**



END OF NOTES

APPENDICES

ENTERPRISE KNOWLEDGE GRAPH SYSTEM

(KNOWLEDGE GRAPH SOLUTION THAT LEADS TO ENTERPRISE AI)

Team 2 Members: **YANG Lu Yi, YIN Tian Shi, YU Yu**

Source: https://github.com/IRS-PM/IRS-PM-2020-01-18-IS02PT-GRP2-Enterprise-Knowledge-Graph

YANG LU YI

END OF APPENDICES