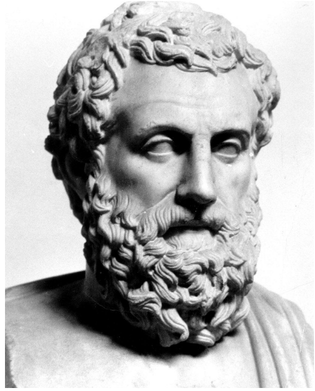
Artificial Intelligence For NLP Lesson-03

人工智能与自然语言处理课程组

2019.April. 14



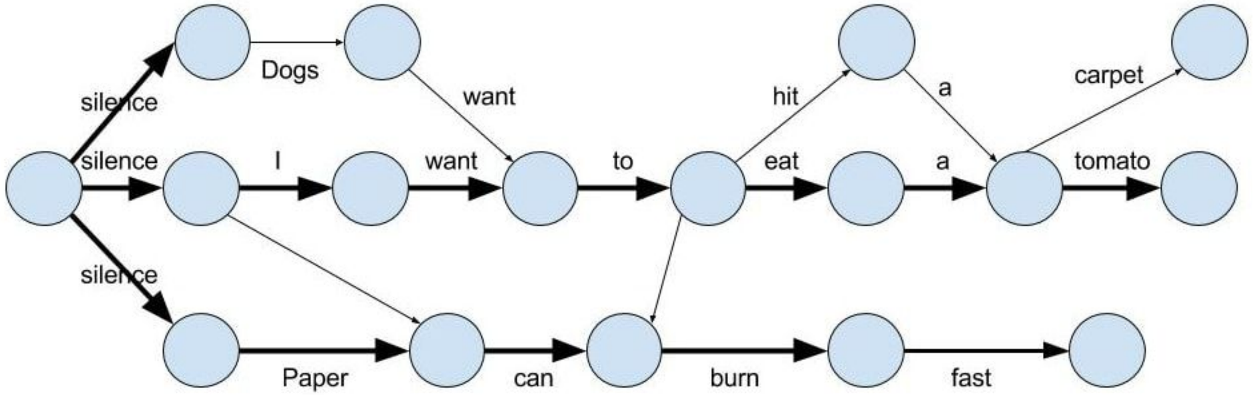
Outline

- 1. Review
  - Syntax Tree: Rule Based System
  - BFS Search, DFS Search: Graph Based System
  - Language Model: Probability Based System
- 2. Machine Learning Based System
- 3. How to make search faster:
  - 1. Heuristic Search, A*
  - 2. Dynamic Programming

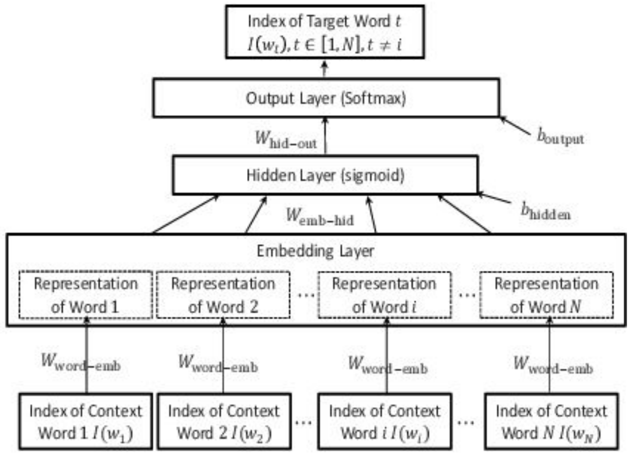
Generator?

4. Probability Based

- Language Model:
  - How likely is a given string in a given 'language'



the thickness of the arrows indicate the value of the transition probability



Language Model

- Conditional Probability

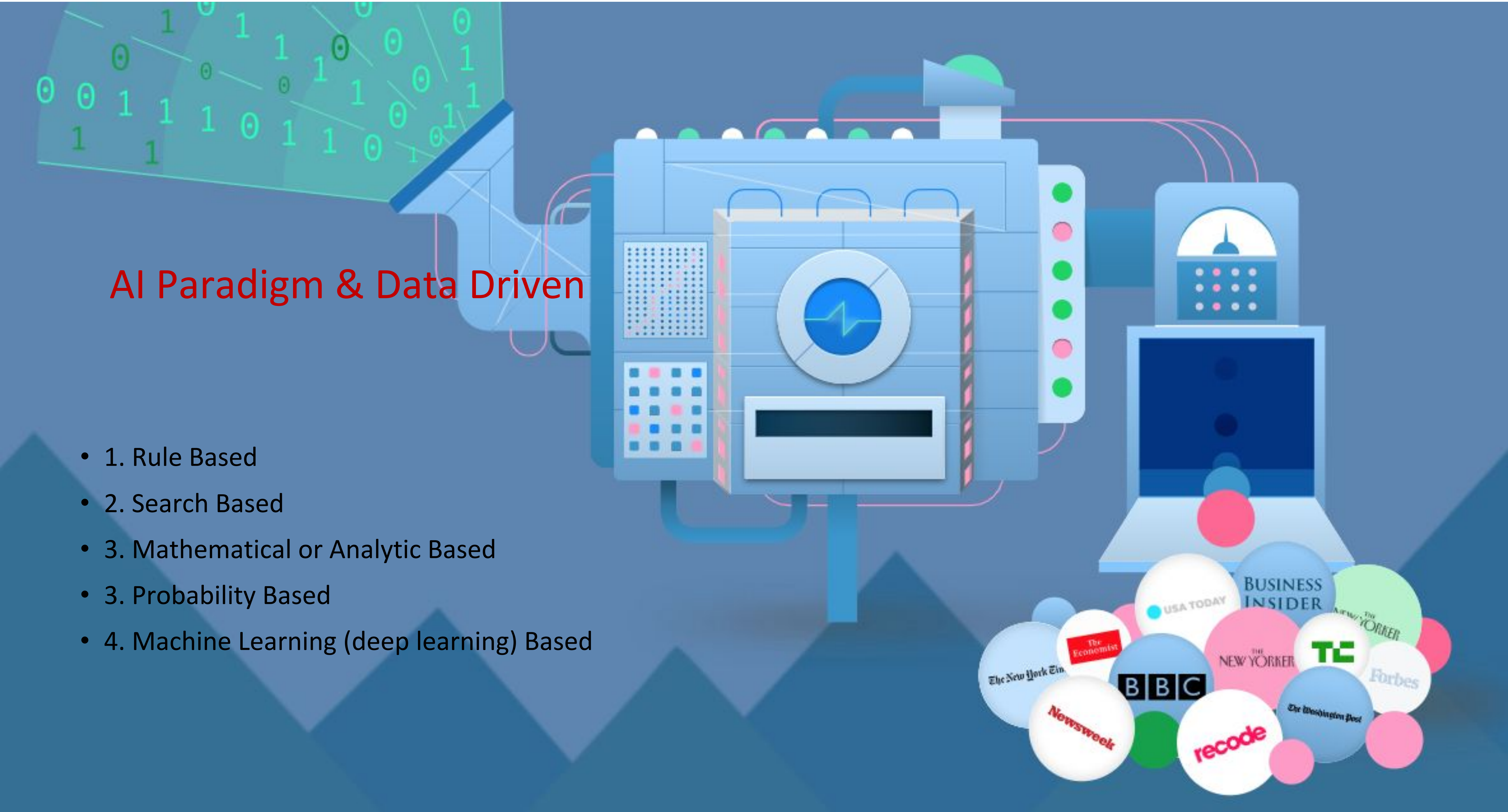
$$P(A \mid B) = \frac{P(A \cap B)}{P(B)}$$

5. Machine Learning (deep learning) Based

- 1. From Probability What We know?
- 2. Data Driven
- 3. Linear Regression
- 4. Classification

6. Plus. Logic Reasoning System

- Talk: Come up with a *new* scenario with AI methods.

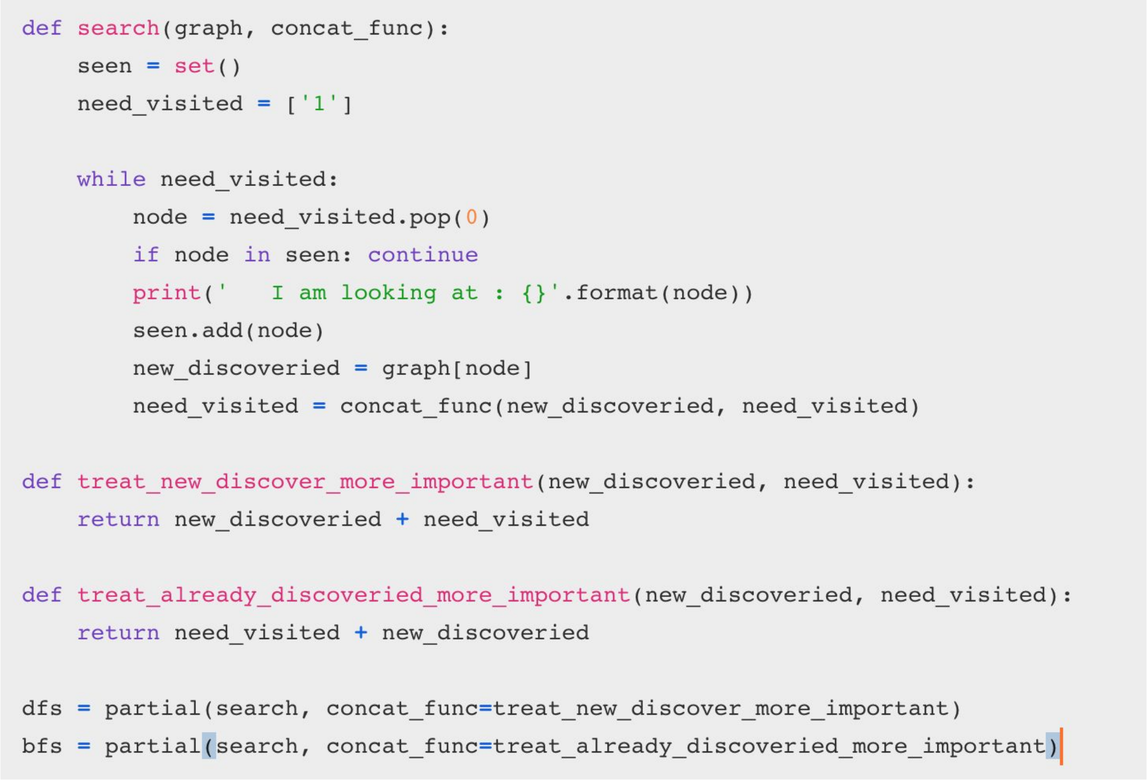


AI Paradigm & Data Driven

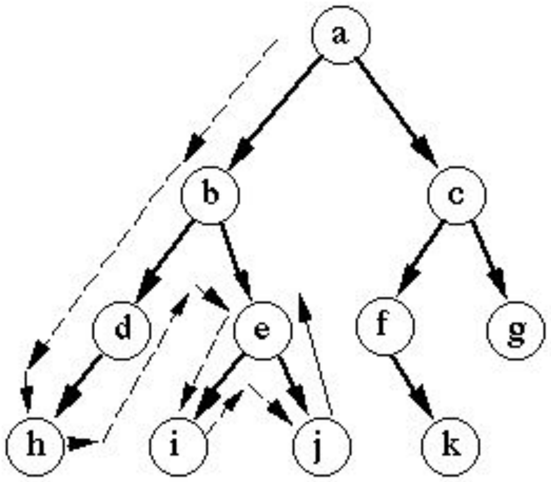
- 1. Rule Based
- 2. Search Based
- 3. Mathematical or Analytic Based
- 3. Probability Based
- 4. Machine Learning (deep learning) Based

The Problem of Search

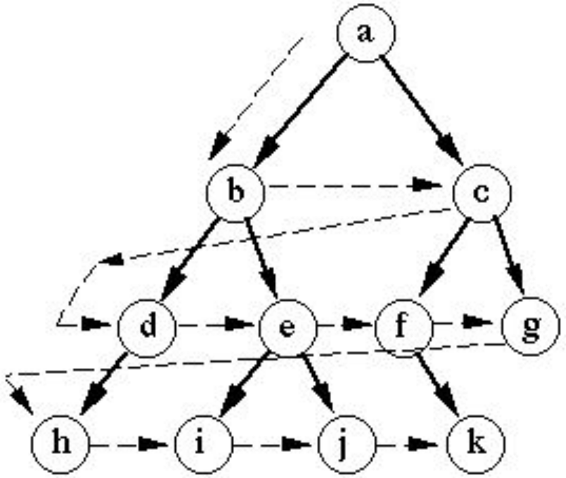
- Heuristic Function
- Search Problem, Search Tree



```
def search(graph, concat_func):
    seen = set()
    need_visited = ['1']

    while need_visited:
        node = need_visited.pop(0)
        if node in seen: continue
        print('   I am looking at : {}'.format(node))
        seen.add(node)
        new_discoveried = graph[node]
        need_visited = concat_func(new_discoveried, need_visited)

def treat_new_discover_more_important(new_discoveried, need_visited):
    return new_discoveried + need_visited

def treat_already_discoveried_more_important(new_discoveried, need_visited):
    return need_visited + new_discoveried

dfs = partial(search, concat_func=treat_new_discover_more_important)
bfs = partial(search, concat_func=treat_already_discoveried_more_important)
```

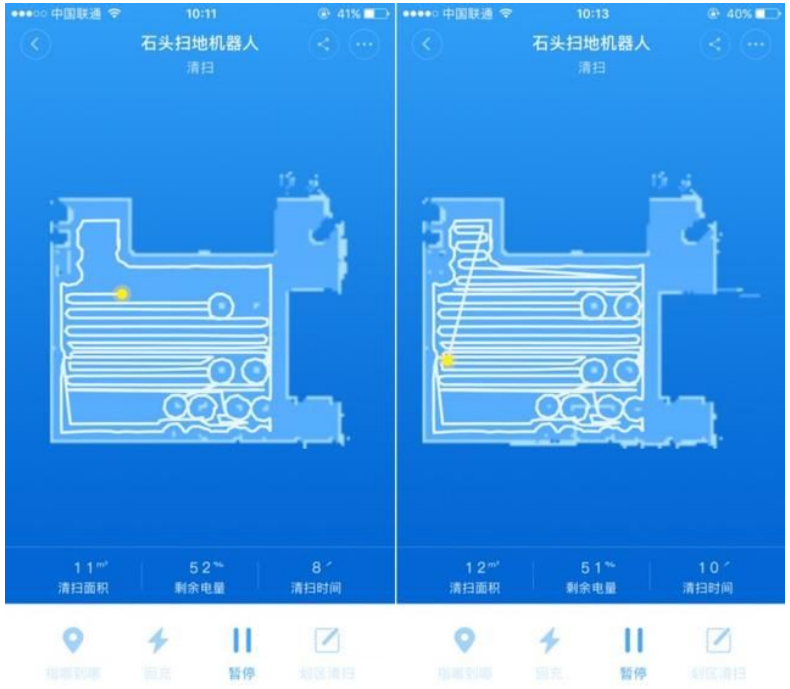


Depth-first search



Breadth-first search

Applications

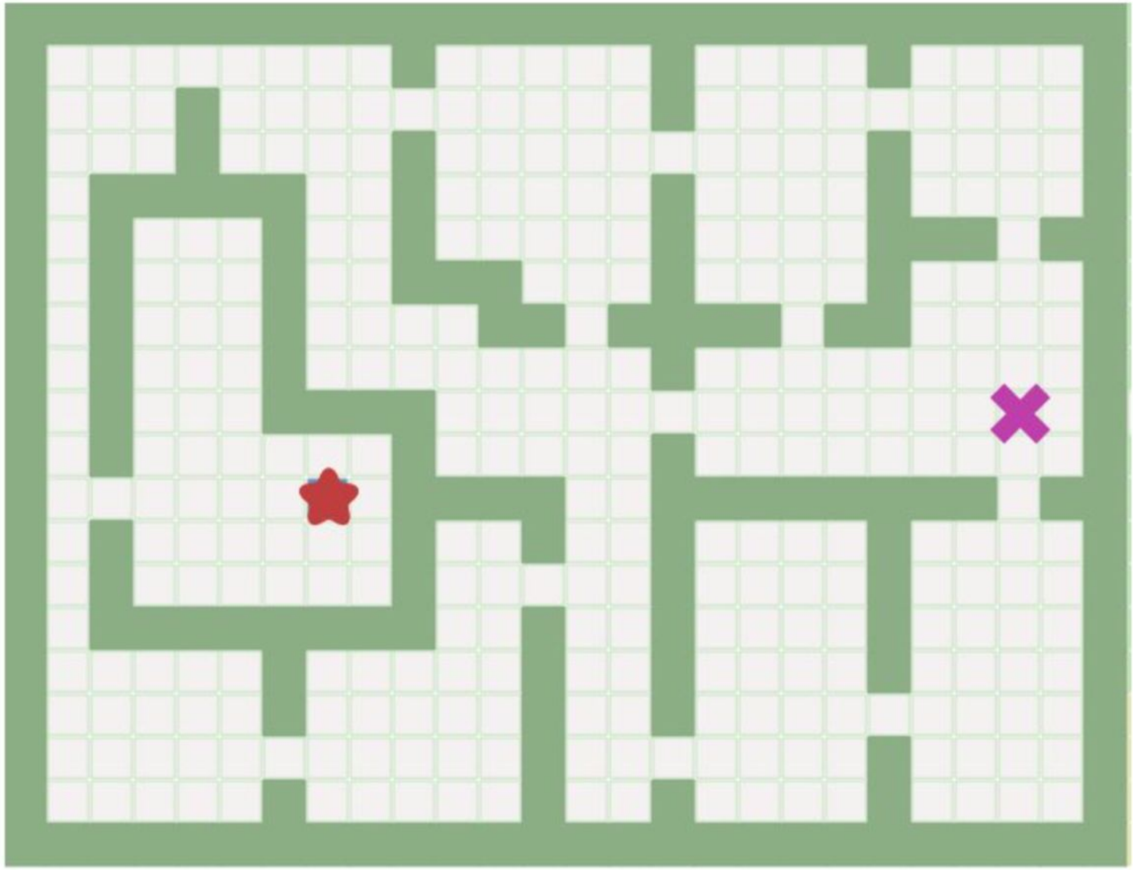


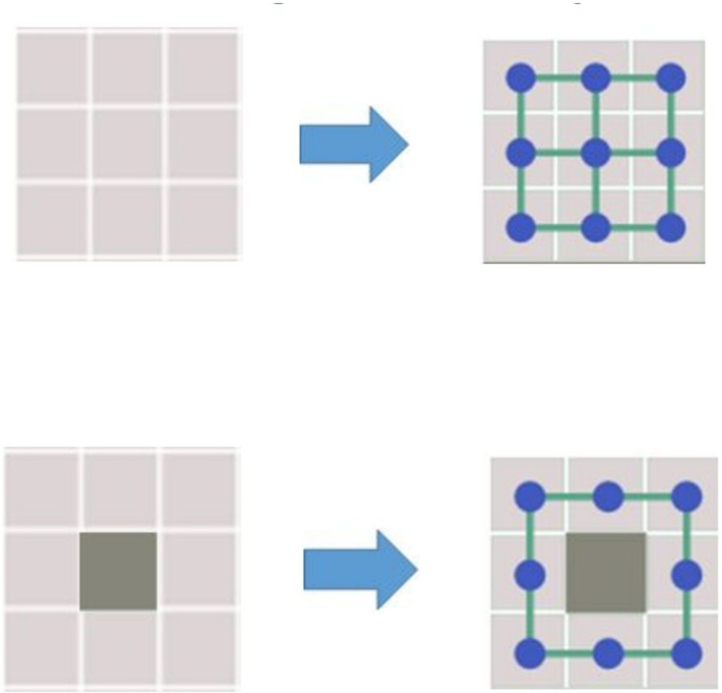
- The *start* state

- The *goal* state

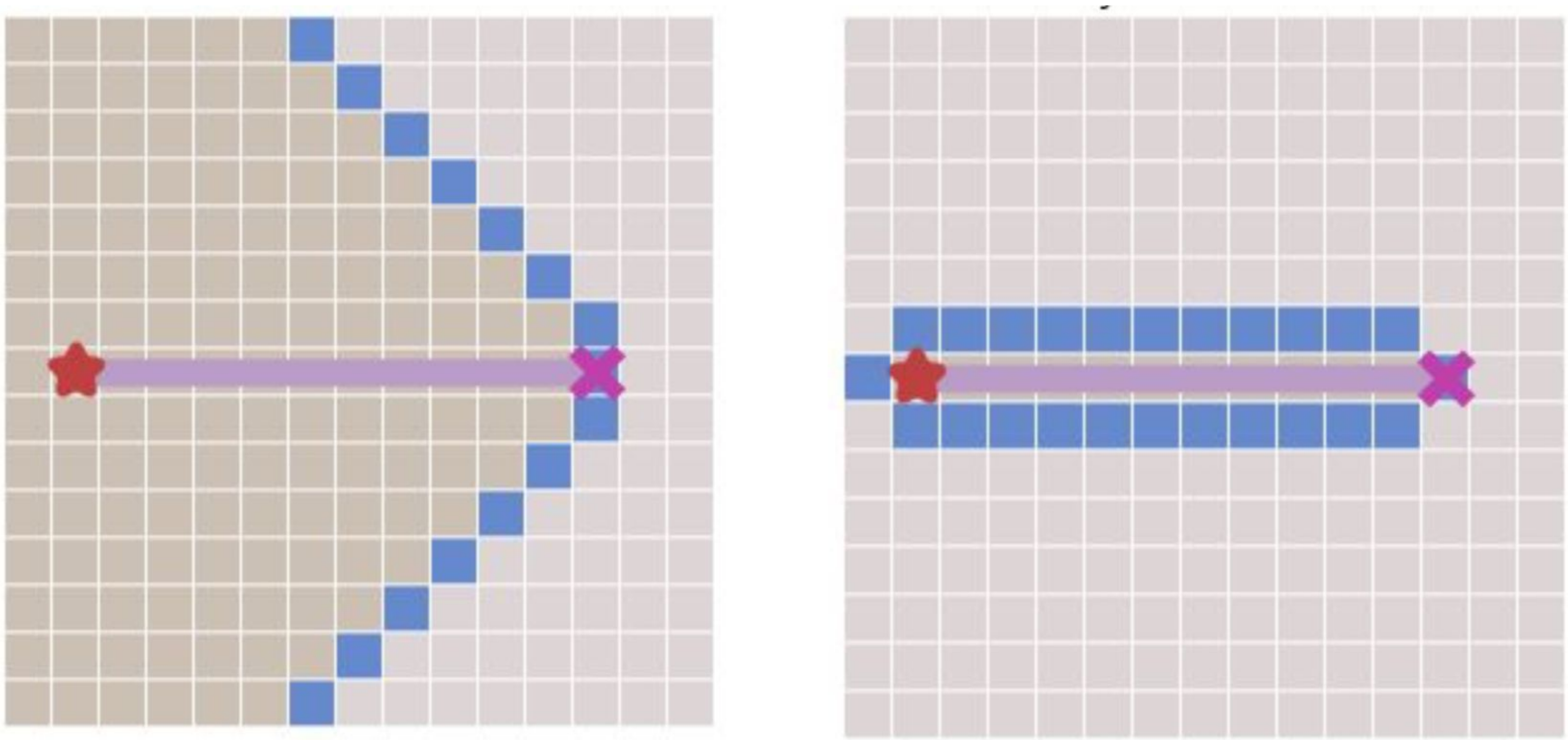
- The *successors*

- The strategy that determines the order in which we search.





From Breadth First Search to Best First Search

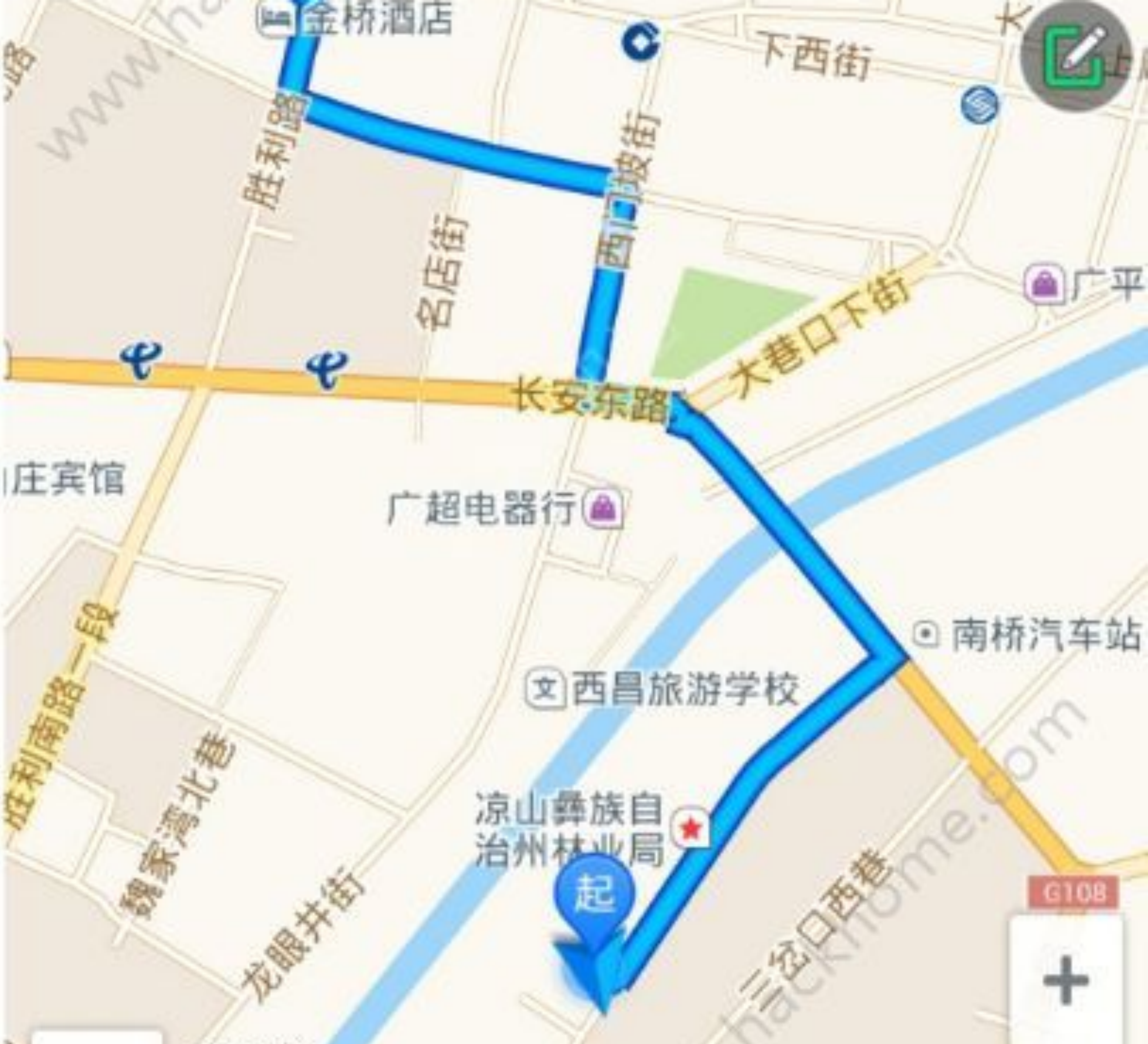


A *heuristic* is an approximate measure of how close you are to the target

Activity

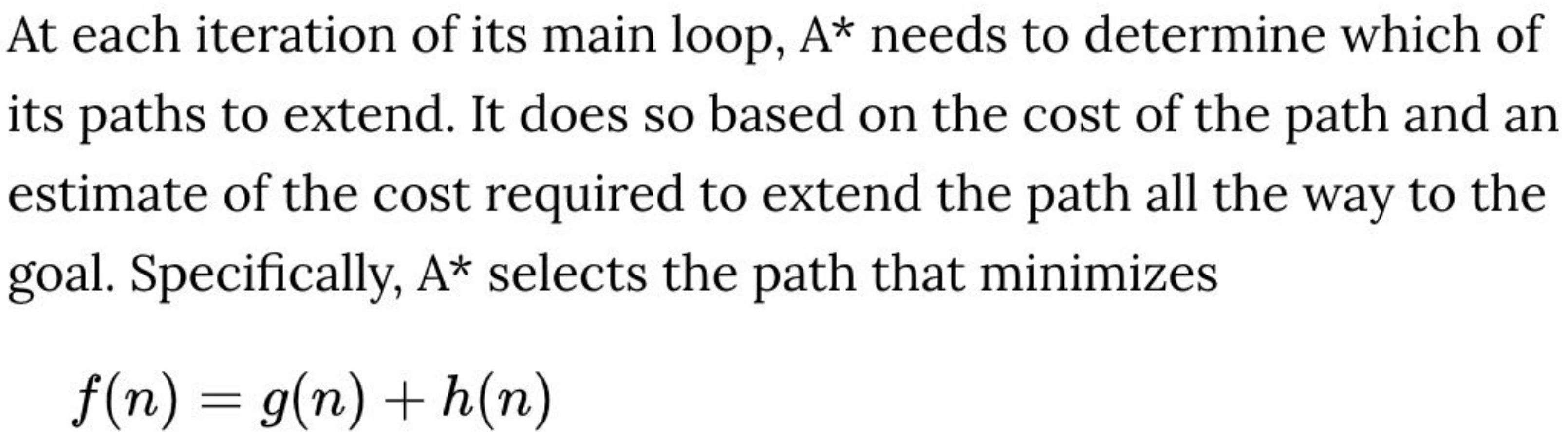
- What’s the heuristic function of one problem?

- Map Routing


金桥酒店
下西街
胜利路
西门坡街
名店街
长安东路
大巷口下街
广平
庄宾馆
广超电器行
南桥汽车站
西昌旅游学校
胜利南路一段
魏家湾北巷
凉山彝族自治州林业局
起
龙眼井街
三岔口西巷
G108
+

- Find the person-2-person
- Connection in a social web.

A* search



At each iteration of its main loop, A* needs to determine which of its paths to extend. It does so based on the cost of the path and an estimate of the cost required to extend the path all the way to the goal. Specifically, A* selects the path that minimizes

$$f(n) = g(n) + h(n)$$

Like BFS, it finds the shortest path, and like Greedy Best First, it's fast.

Each iteration, A* chooses the node on the frontier which minimizes:

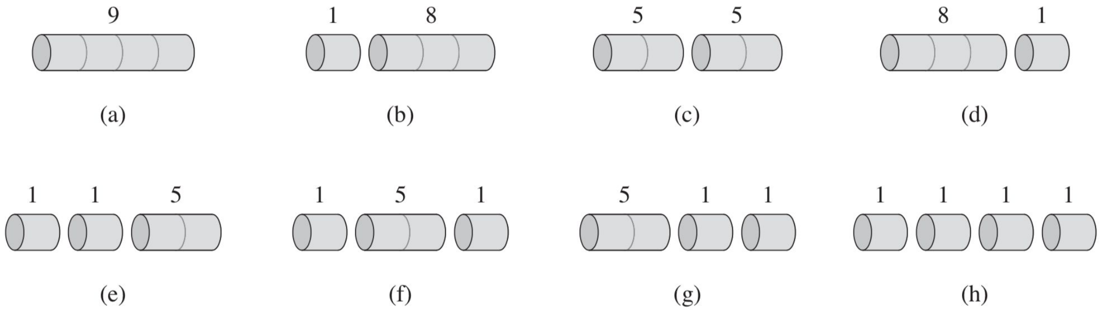
steps from source + approximate steps to target

From Search to DeepLearning

Decision Making and Dynamic Programming

Dynamic Programming

- 1. Rob Cutting Problem
- 2. Edit-Distance Problem
- 3. Key Characteristics for Dynamic Programming
- 4. The Travel Salesman Problem

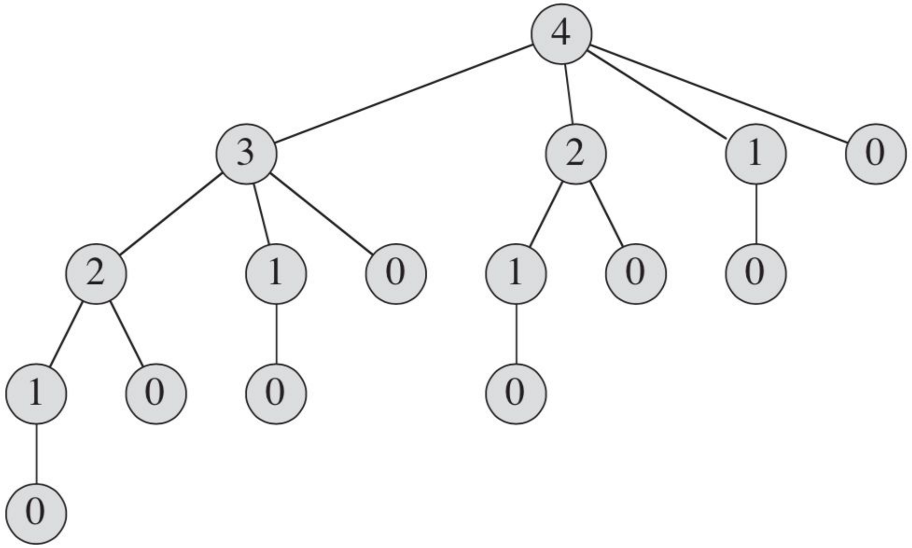


(a) (b) (c) (d)

(e) (f) (g) (h)

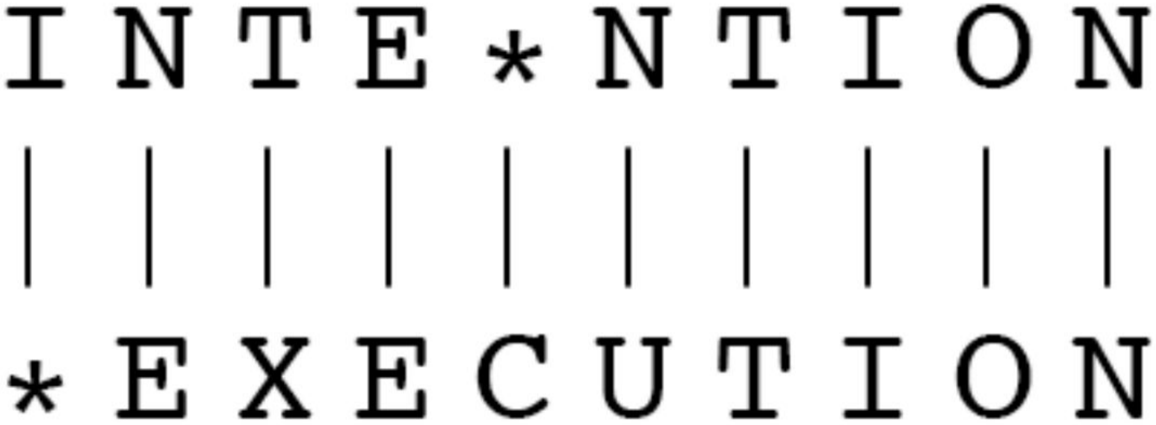
length i	1	2	3	4	5	6	7	8	9	10
price p_i	1	5	8	9	10	17	17	20	24	30

$$r_n = \max\left(p_n, r_1 + r_{n-1}, r_2 + r_{n-2}, \ldots, r_{n-1} + r_1\right) .$$



Programming Solution For This

Problem 2: Edit Distance



```
I N T E * N T I O N
| | | | | | | | | |
* E X E C U T I O N
```

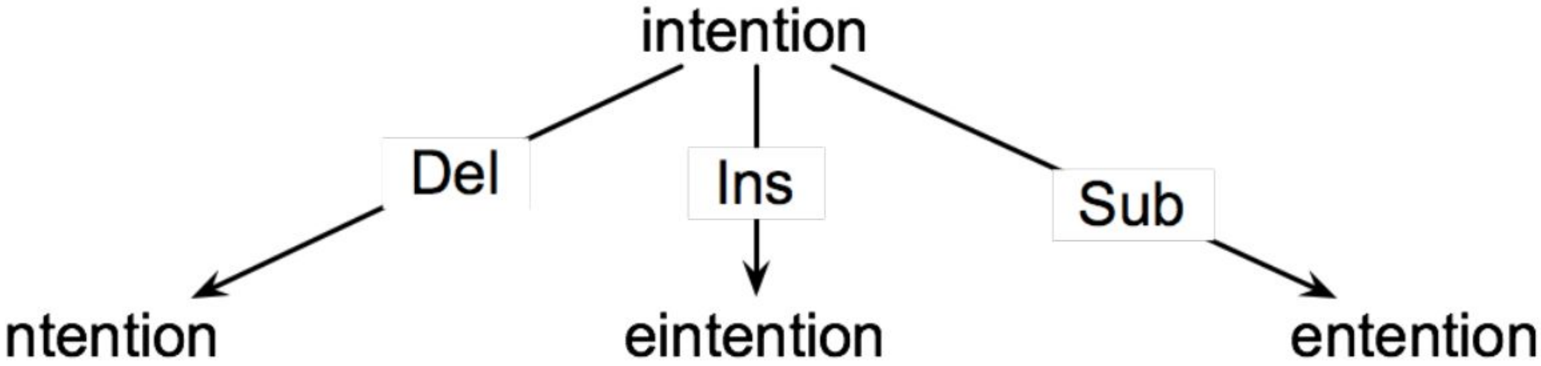
- Insertion
- Deletion
- Substitution

How similar are two strings?

- Spell Correction
  - The user typed “biejing”
  - --biejie? (别介)
  - --beijing? (北京)
  - --beijin? (北金)

- Evaluating Machine Translation and speech recognition
  - Spokesman confirms senior government adviser was shot.
  - Spokesman said   the senior   adviser was shot dead.

Search Graph is Huge



Defining Min Edit Distance

- For two strings
  - X of length *n*
  - Y of length *m*

- We define D*(i, j)*
  - the edit distance between X[1…i] and Y[1..j]
  - The edit distance between X and Y is thus D*(n, m)*

Defining Min Edit Distance (Levenshtein)

- Initialization
  - D(i, 0) = i
  - D(0, j) = j

- $$D\ (i, j) = \min \begin{cases} D(i - 1, j) + 1 \\ D(i, j - 1) + 1 \\ D(i - 1, j - 1) + 2 \text{ if } X(i) \mathrel{!=} Y(i) \text{ else } D(i - 1, j - 1) \end{cases}$$

Assignments

- 0. Review the programming task
- 1. Beijing Subway Routing