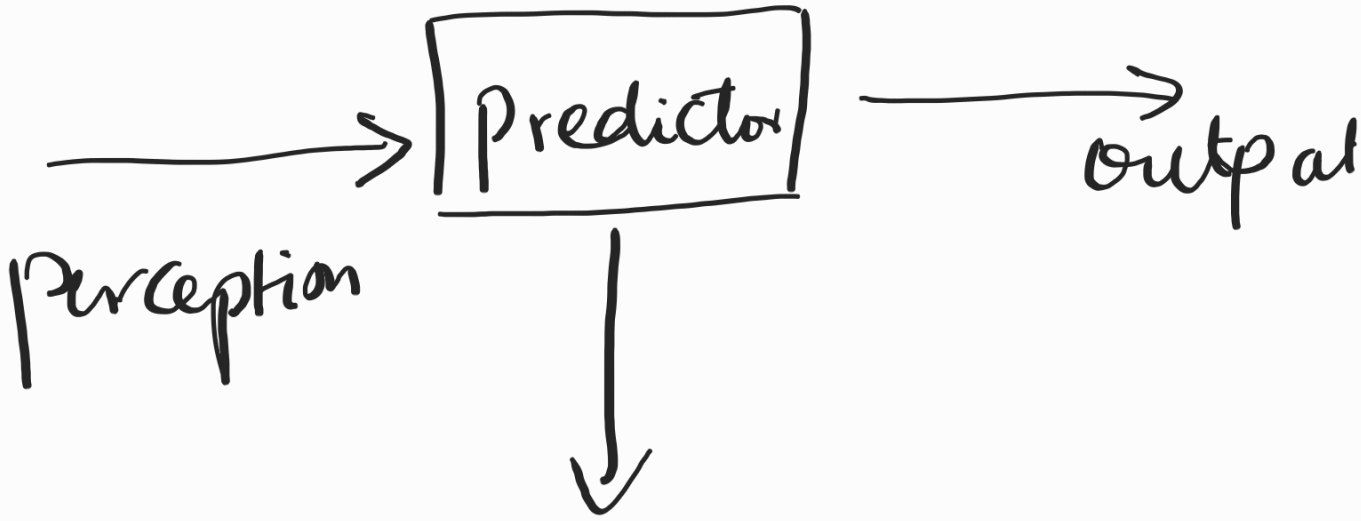
AI Theory, W311



What is the Type of this model?

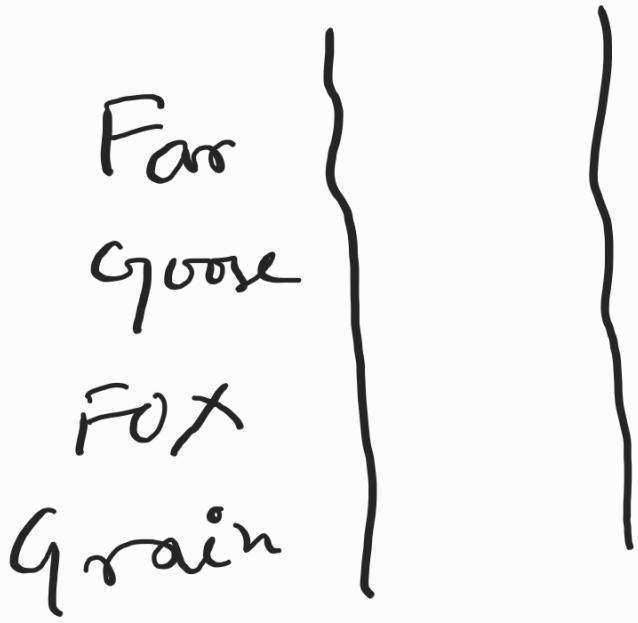
Reflex based models are classifiers (table, chair etc).

Search-based models return sequence of actions as output.

Search based Models:

(we'll start with these for AI).

Story: Fox, River, Goose, Grain and Farmer



What should be the optimal solution so that all goods are transported?

⇒ Seven crossings required to cross river.

Important:

We considered safe and unsafe

states of the environment, as a search parameter.

↓

What is defined as a state?

Summary of all actions taken of the environment at any given time "t".

In our story example: 16 states

But how do we define the term "state" and "actions"? (define in a mathematical sense).

Let's define using an example.

Route finding:

Start state : UET

Goal state : Waqas Bijni

Transition function : Gimena

state, proceed to the next.

What are the states?

All possible nodes in the way.

Actions? All Left/Right Turns in the way.

Machine Translation:

La maison bleue

↓

the blue house

action => translate one of

words per iteration

In the multiple translations problem of languages (عربي) find the best translations.

<u>Roadmap</u>:

Dynamic Programing

Uniform Cost Search

Search Tree

<u>Recall</u>:

pillars of solving an AI

problem:

Representation

↓

Inference

↓

Learning

Transportation Example:

⇒ blocks 1 .... n.

⇒ s ... s+1 takes cost 1.

⇒ s .... 2s takes cost 2.

⇒ what is the $\boxed{\text{LEAST TIME}}$ to travel from 1 ... n?

intuition: 6 cost minimum

Walk → Tram → Walk → Tram

1 2 1 2

Can we decrease the cost to less than 6?

```
Class TransportationProblem:
                    n=0
    def sstart(self):
        n = 1
        return 1

    def isGoal(self, s):
        if s == self.n:
            return True
```

```
def Actions (self, s):
# return the actions corresponding
to the branches of the subtree
rooted at state s of the
tree.

actions = [ ]

    if s+1 <=n:
        actions.push ("walk")
    if s*2 <=n:
        actions.push ("tram")

return actions
```

```
def Cost(self, s, a):
    """
        s = state
        a = action
    """

return
    cost == "tram" ? 2 : 1
```

```
def transition_function(self,
    s, a):

    if a = "tram"
        return 2s
    elif a = "walk":
        return s+1
    return None
```

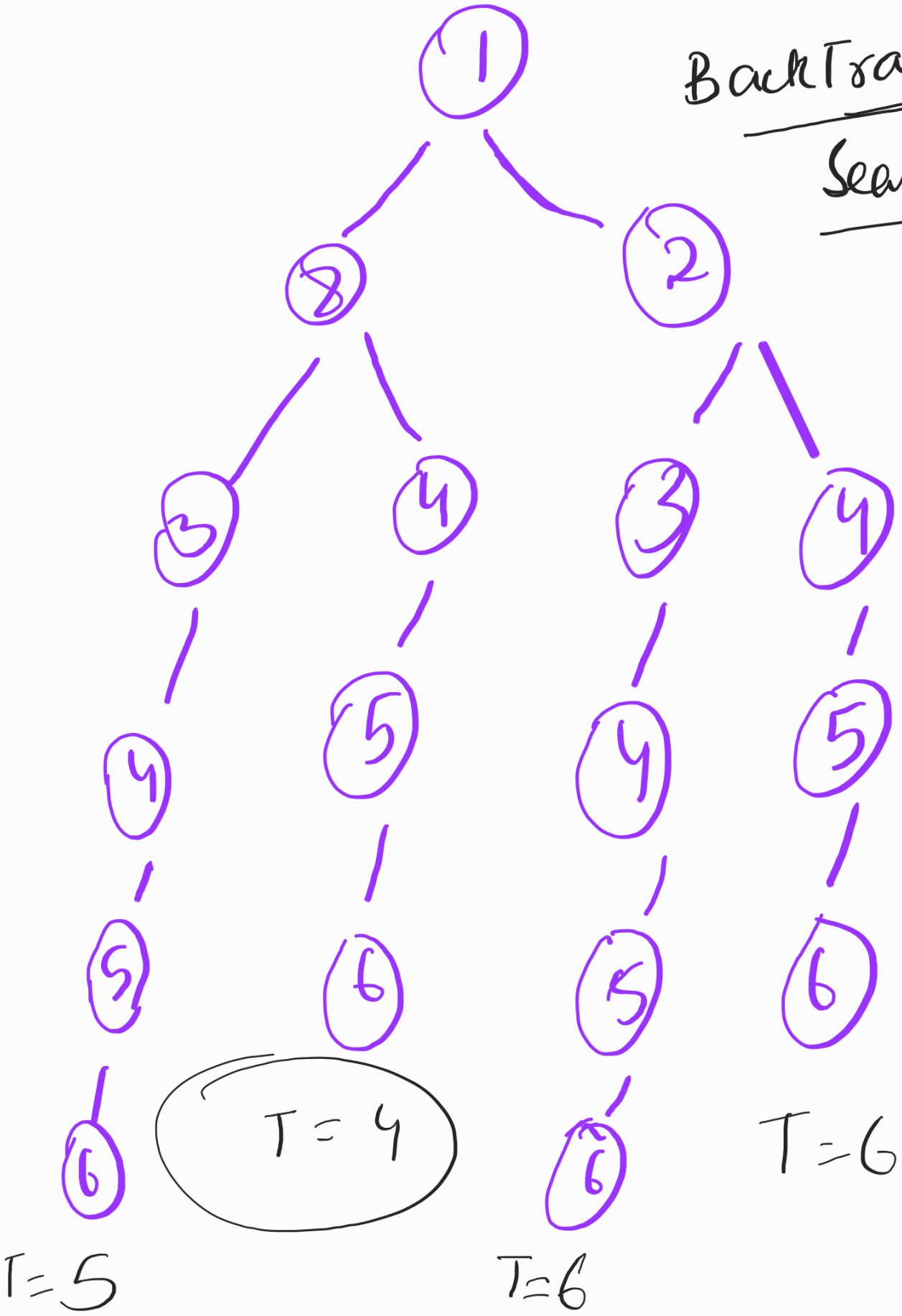
up until this phase, we have <u>modelled</u> the problem.

```
def fn(s):
    if self.isGoal(s):
        return

    else:
        actions = self.actions(s)
        for act in actions:
            new_s = transition(act)
            fn(new_s)
```

The search tree will be as follows: $n=6$

Backtracking Search:



left cost = 1

right cost = 2

T = 4 is the minimum cost.

How do we modify the code such that we get our answer?

=>

```
for act in actions:
    new_s = transition(s)
    cost = cost(s, act)
    fn(new_s)
```

MODIFY f(n) as follows:

```
fn(s, running_cost):
    if isGoal(s):
```

```
if c < self.mincost:
    self.mincost = c
else:
    actions = actions(s)
    for act in actions:
        new_s = transition(s)
        cost = cost_func(s, a)
        fn(new_s, running_cost
                + cost)

return self.min_cost
```

* This whole code is written in an inner function

So:

```
fn(st, sum_cost):
    self.min = ∞
    self.history = []

    def recurse(s, c, hist):
        if isGoal(s):
            if c < self.min:
                self.min = c
                self.history = hist
        else:
            actions = actions(s):
            for act in actions:
```

```
new_s = transition(s)
cost = cost(s, a)

recurse(new_s,
        cost + c,
        hist +
        [(s, act,
          new_s)])

recurse(st, 0, [])
return(self.min, self.history)
```

This kind of solving is called BACKTRACKING.

DFS:

just BackTracking Search, but Stop $\boxed{\text{recursion}}$ on first goal instance.

BFS:

↳ explore each level completely

Next class:

BFS, back_iterative, progressive searches