# AI Theory, W2 L1/2

## Problem Reduction methods:

Humans → goal centric

we take "sub problem" approach.

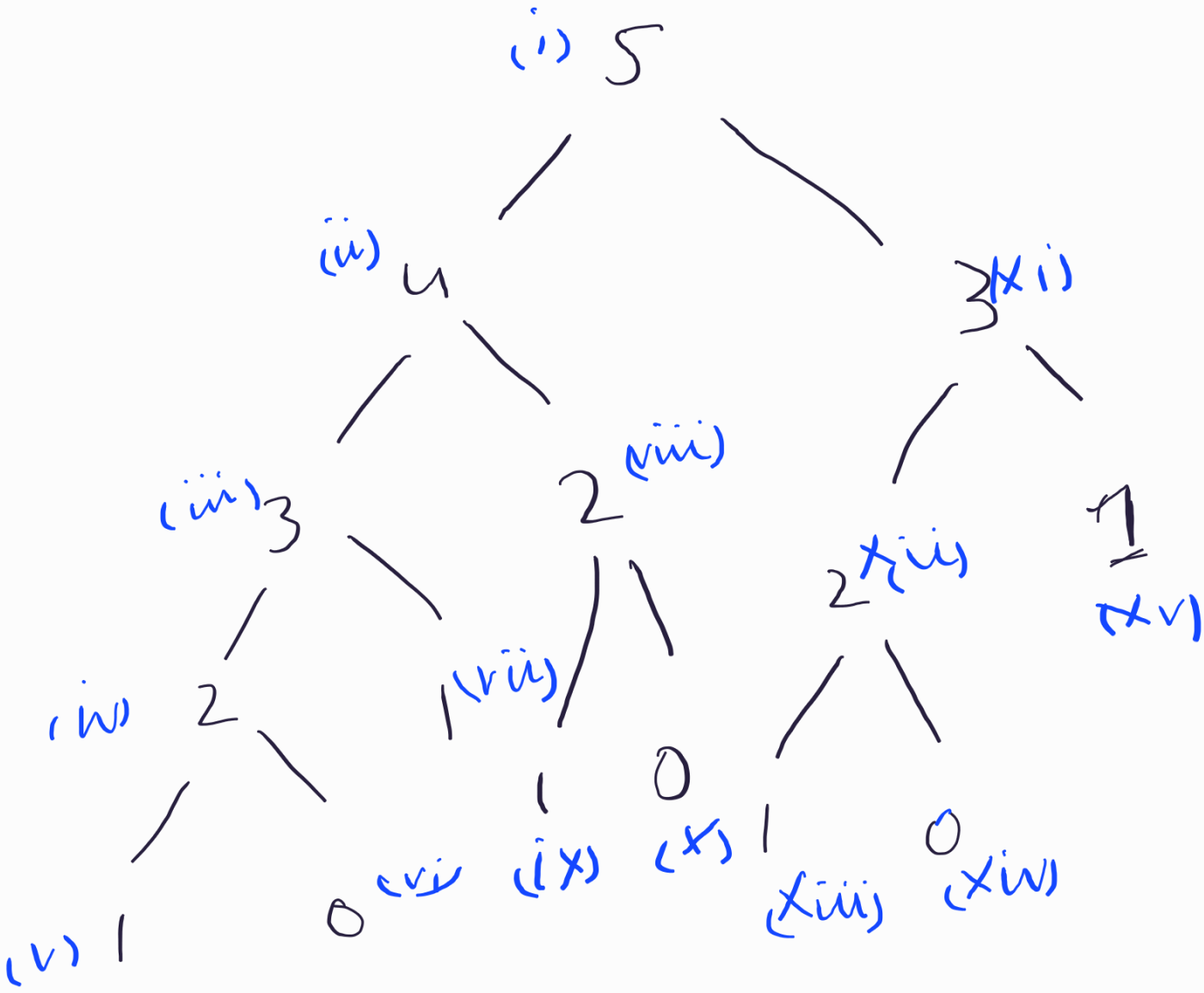
Take a big problem, divides it into chunks and delegate.

Sounds a lot like recursion?

well:

## Recursion:

```
Fibonacci(n):
    if n <= 1
        return n

    return Fib(n-1) + Fib(n-2)
```




⇒ Roman numeral indicates no. of call at that index.

This method is not effecient in memory.

Can we use dynamic Prog. or DFS Algorithm for this purpose?

## A bit Terminology:

### AND Goal:

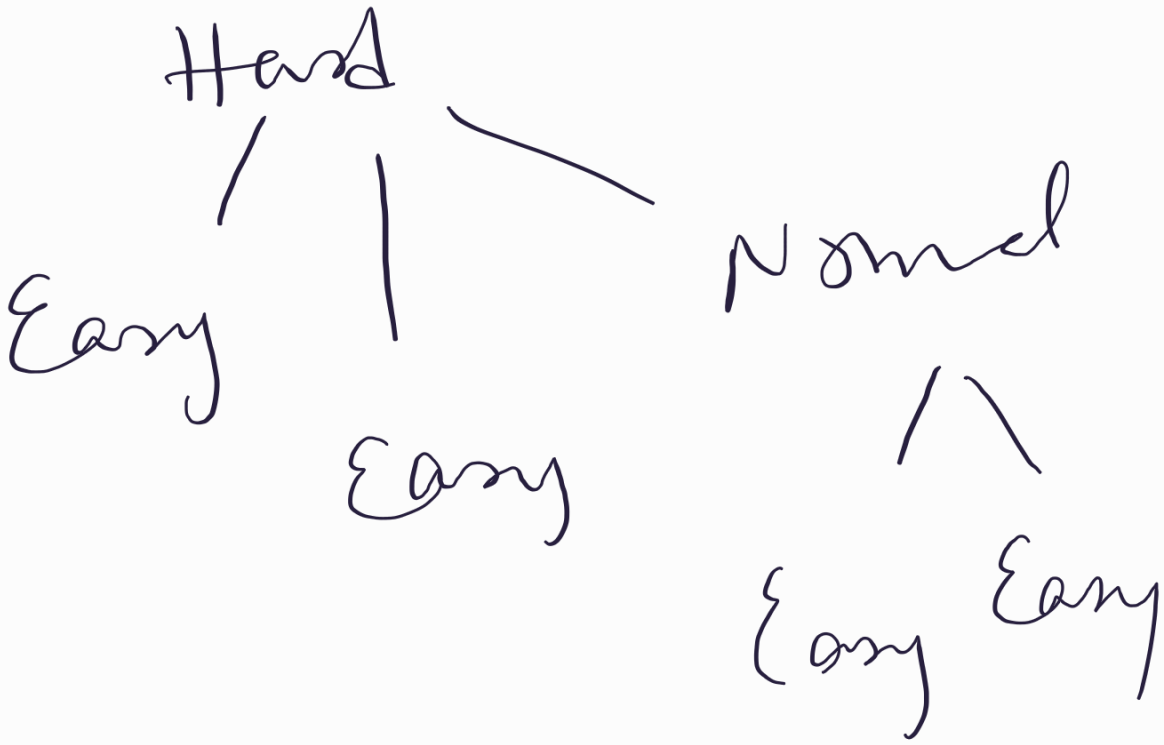
satisfy when all immideate subgoals are satisfied

### OR Goal:

OR operation on goals

## Goal Tree:

A semantic tree representing problem and its solution space in context of goals.




## Back To fibonacci:

Use Dynamic Programing to lessen the number of recursive calls.

```
cache = { }

def recurse (num):
    if exists cache[num]
        return cache[num]

    if num <= 1:
        cache [num] = num
        return num

ans = recurse(num -1) + recurse
                        (num -2)
    cache [num] = ans
    return ans
```

A practical application of Recursion: Edit Distance

If Last characters is same, then recurse on string EXCLUDING Last characters.

Else:

Three cases.

1. Insertion
2. Deletion
3. Substitution

Edit Distance(A, B, a, b):

len(A) len(B)

# your code

#

Pseudo code :

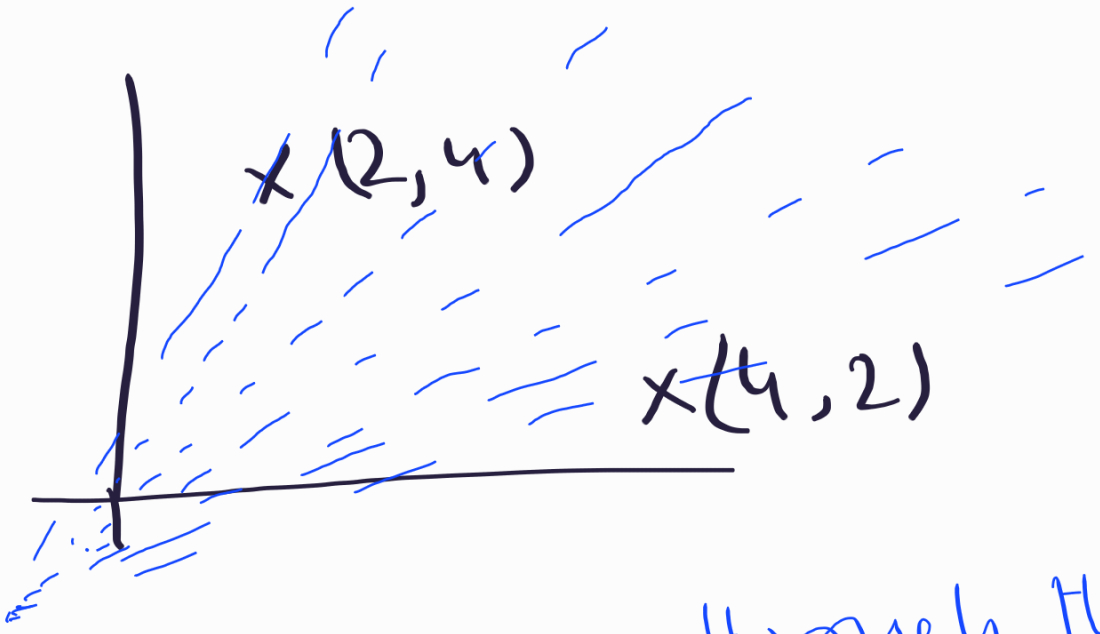
```
if (last chars of A & B are
    same):
    EditDistance(A, B, a-1, b-1)

else:
    c1 = Edit(A, B, a-1, b)
                # insertion case

    c2 = Edit(A, B, a, b-1)
                # deletion case

    c3 = Edit(A, B, a-1, b-1)
                # substitution case

return min(c1, c2, c3)
```

Challenge: implement a dynamic programming version of this edit distance algorithm.

# LEAST SQUARES LINES

★ important ★

Suppose we have a plot.




Several lines pass through the plot, shown in blue.

⇒ Which of these dotted lins best FITS the data?

The definition of fit is as follow:

"given a prediction line, the error should be minimum".

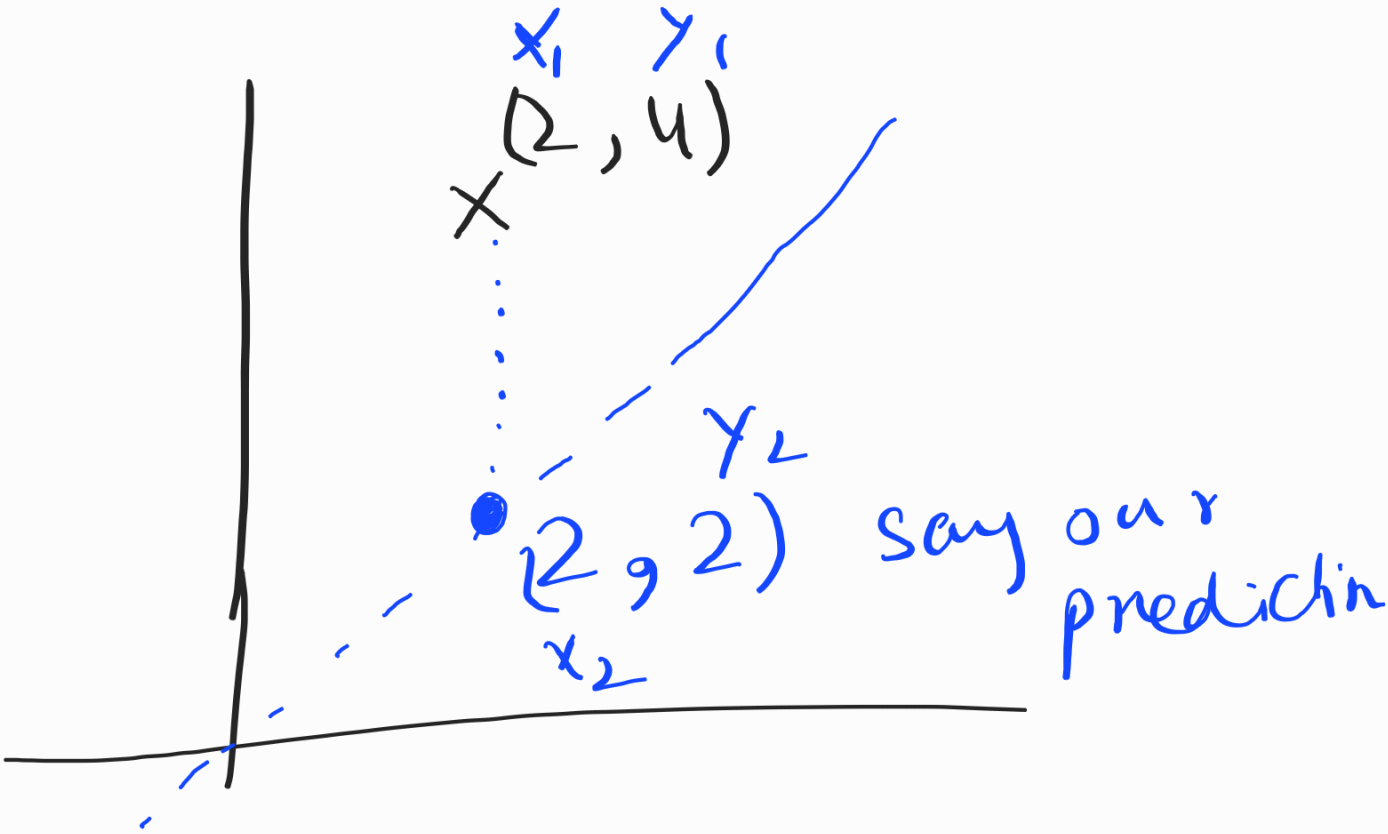
We will get to the error part later.

So, for each point, we take the euclidian criteria:

$$f(m) = (x_1 - x_2)^2 + (y_1 - y_2)^2$$

we have ommitted $\sqrt{\ }$ purposefully.

The error is a function of y. Here's how.




$$x_1 - x_2 = 0$$

So the error criteria becomes

$$f(m) = \sum_{i=1}^{m} (\hat{y}_i - y_i)^2$$

for each point in the line.

Now, a bit of derivation.

we know that

$$y = mx$$

quadratic

$$f(m) = \sum_{i=1}^{n} (x_i m - y_i)^2$$

is the mean-squared error (MSE).

This means, given a value

$m$, we can find its $f(m)$ value:

But how do we deal with the quadratic mess?

Recall calculus:

$$f(m) = (mx_i - y_i)^2$$

$$\frac{\partial}{\partial m} f(m) = \underbrace{\sum_{i=1}^{n} 2(mx_i - y_i)\, x_i}_{\text{is a linear solution}}$$

Almost done.

just one small issue. "m" value in output can be very large.

$$m = m - dF$$

dF indicates a sudden change (knee-jerk) in the direction

We control this by introducing a parameter $\alpha$. The purpose: punish knee jerk changes -

$$\boxed{m = m - dF \cdot \alpha}$$