Administrivia

- Save, then quit
- 9:41
- Piazza 
- LaTeX template RSN™

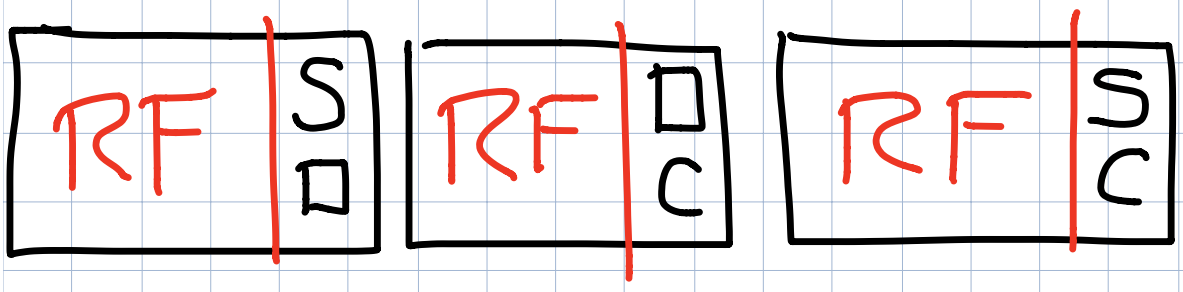
Dynamic Programming 2

1. Recurrence
   - English description first
2. Memoize
   - Identify subproblems
   - Data structures
3. On Purpose
   - Dependencies
   - Order
4. Analysis

(5. Optimize)

Edit distance

min # insertions
deletions
replacements
to change A into B



$$Edit(A[1..m], B[1..n]) =$$

$$\min \begin{cases} Edit(A[1..m-1], B[1..n]) + 1 \\ Edit(A[1..m], B[1..n-1]) + 1 \\ Edit(A[1..m-1], B[1..n-1]) + [A[m] \neq B[n]] \end{cases}$$

ALGO
ALT

Edit(i, j) =
Edit distance
between A[1..i]
and B[1..j]

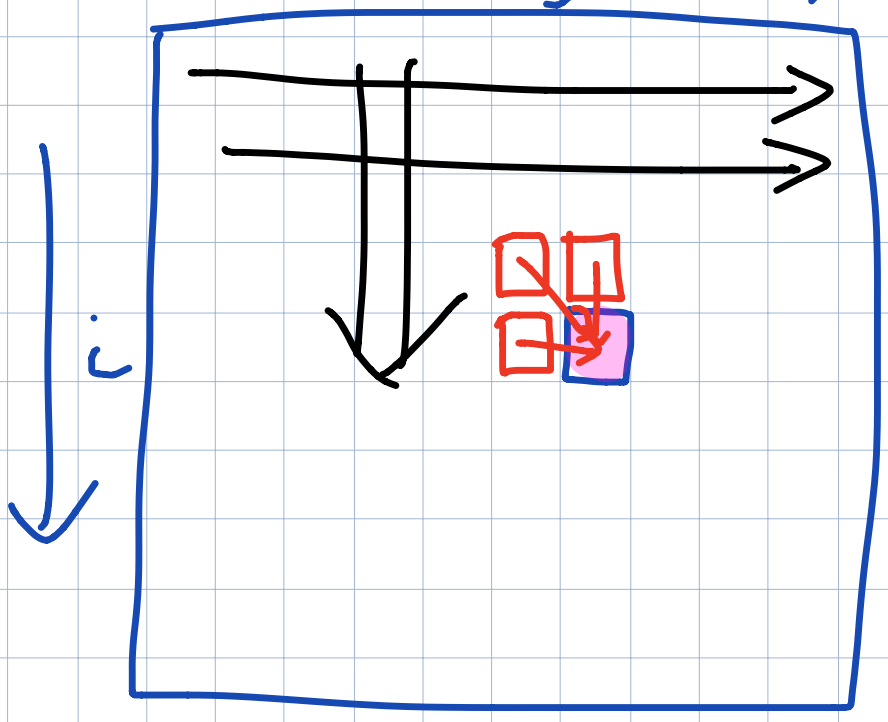
!

Edit(0, j) = j

Edit(i, 0) = i

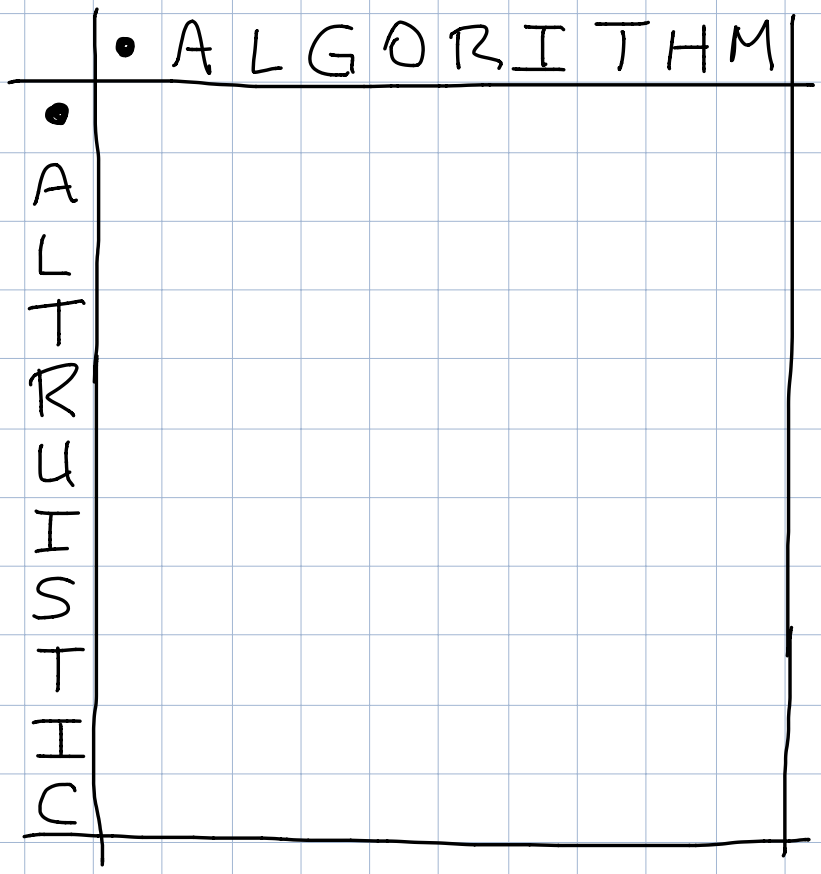
$$Edit(i,j) = \begin{cases} i & \text{if } j = 0 \\ j & \text{if } i = 0 \\ \min \left\{ \begin{array}{l} Edit(i-1,j)+1, \\ Edit(i,j-1)+1, \\ Edit(i-1,j-1)+[A[i] \neq B[j]] \end{array} \right\} & \text{otherwise} \end{cases}$$

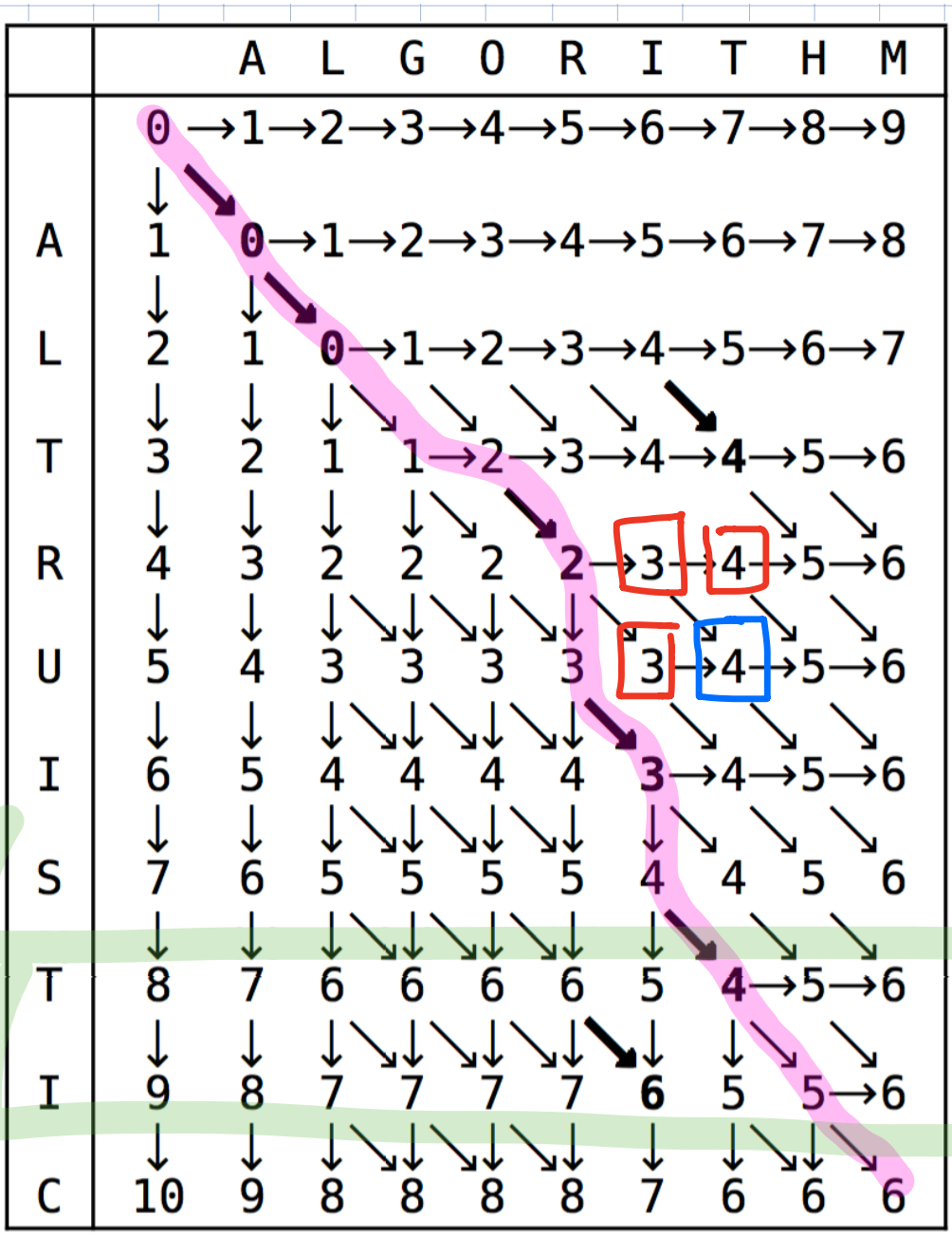
Memoize into 2d array



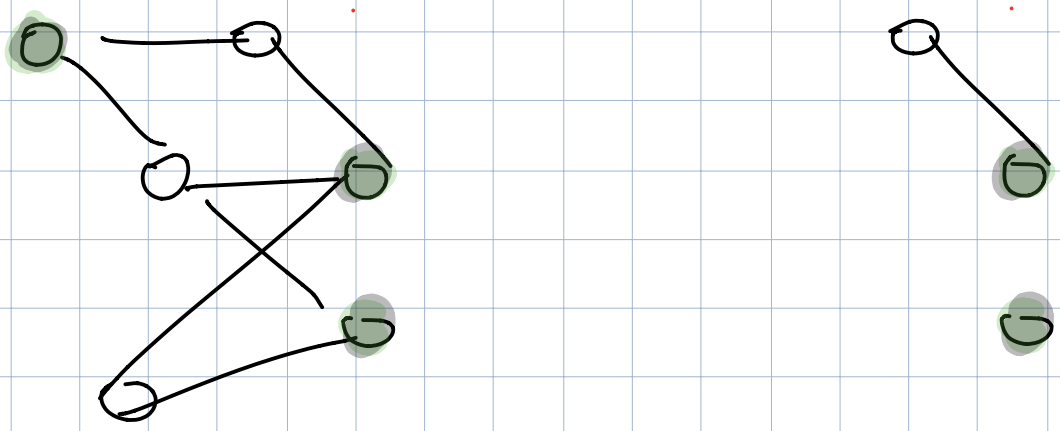

```
EditDistance(A[1..m], B[1..n]):
  for j ← 1 to n
      Edit[0, j] ← j
  for i ← 1 to m
      Edit[i, 0] ← i
      for j ← 1 to n
          if A[i] = B[j]
              Edit[i, j] ← min{Edit[i−1, j] + 1, Edit[i, j−1] + 1, Edit[i−1, j−1]}
          else
              Edit[i, j] ← min{Edit[i−1, j] + 1, Edit[i, j−1] + 1, Edit[i−1, j−1] + 1}
  return Edit[m, n]
```

O(nm) time


• ALGORITHM
• ALTRUISTIC


A L G O R I T H M
0 1 2 3 4 5 6 7 8 9
A 1 0 1 2 3 4 5 6 7 8
L 2 1 0 1 2 3 4 5 6 7
T 3 2 1 1 2 3 4 4 5 6
R 4 3 2 2 2 2 3 4 5 6
U 5 4 3 3 3 3 3 4 5 6
I 6 5 4 4 4 4 3 4 5 6
S 7 6 5 5 5 5 4 4 5 6
T 8 7 6 6 6 6 5 4 5 6
I 9 8 7 7 7 7 6 5 5 6
C 10 9 8 8 8 8 7 6 6 6

NP-hard!



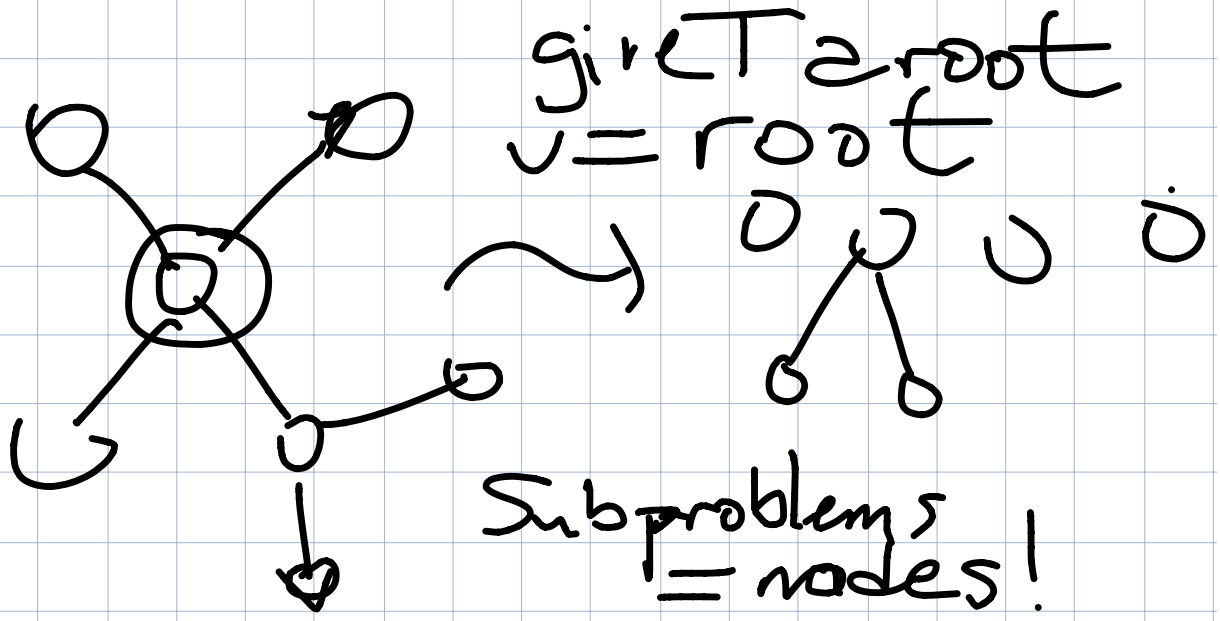
Max. Indep. Set

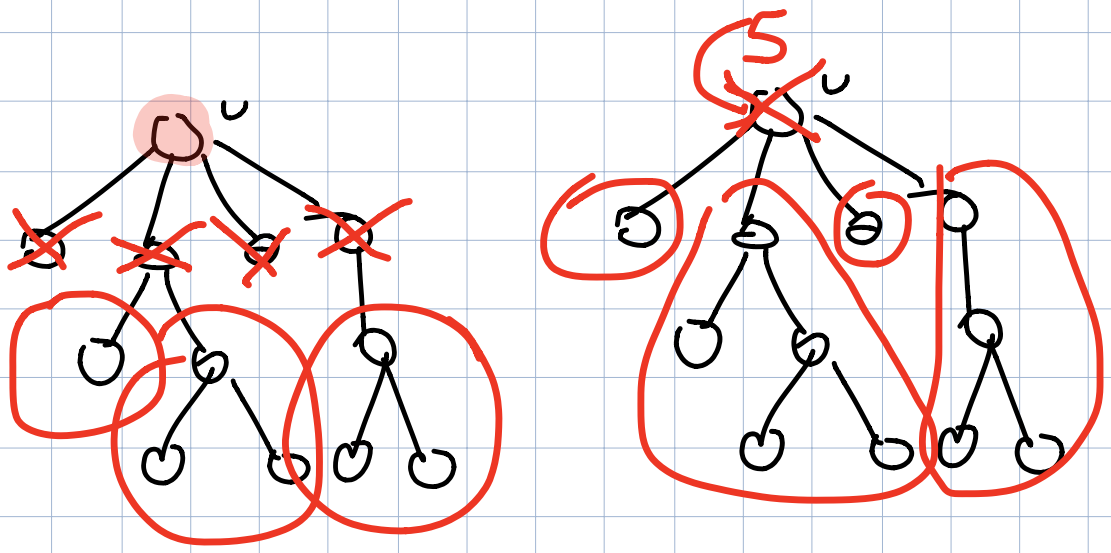
```
MaximumIndSetSize(G):
    if G = ∅
        return 0

    v ← any node in G
    withv ← 1 + MaximumIndSetSize(G \ N(v))
    withoutv ← MaximumIndSetSize(G \ {v})
    return max{withv, withoutv}.
```

Trees!

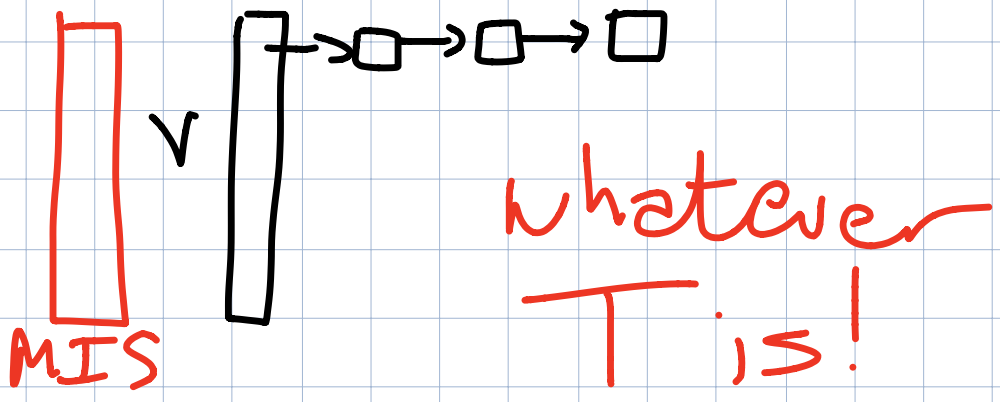
```
MaximumIndSetSize(T):
  if T = ∅
      return 0
  v ← any node in T
  withv ← 1
  for each tree T' in T \ N(v)
      withv ← withv + MaximumIndSetSize(T')
  withoutv ← 0
  for each tree T' in T \ {v}
      withoutv ← withoutv + MaximumIndSetSize(T')
  return max{withv, withoutv}.
```





```
MaximumIndSetSize(v):
  withv ← 1
  for each grandchild x of v
      withv ← withv + MaximumIndSetSize(x)
  withoutv ← 0
  for each child w of v
      withoutv ← withoutv + MaximumIndSetSize(w)
  return max{withv, withoutv}.
```

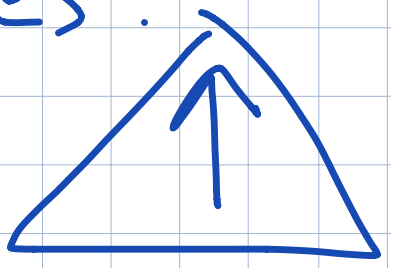
Data structure?



whatever T is!

ORDER? Post! DFS

Dependencies?



```
MaximumIndSetSize(v):
  withoutv ← 0
  for each child w of v
      withoutv ← withoutv + MaximumIndSetSize(w)
  withv ← 1
  for each grandchild x of v
      withv ← withv + x.MIS
  v.MIS ← max{withv, withoutv}
  return v.MIS
```

O(n)

For all v in postorder

[eval MIS(v)]

```
MaximumIndSetSize(v):
  v.MISno ← 0
  v.MISyes ← 1
  for each child w of v
      v.MISno ← v.MISno + MaximumIndSetSize(w)
      v.MISyes ← v.MISyes + w.MISno
  return max{v.MISyes, v.MISno}
```

O(n)

```
LongestPath(v, t):
  if v = t
      return 0
  if v.LLP is undefined
      v.LLP ← ∞
      for each edge v→w
          v.LLP ← max{v.LLP, ℓ(v→w) + LongestPath(w, t)}
      return v.LLP
```

```
LongestPath(s, t):
  for each node v in reverse topological order
      if v = t
          v.LLP ← ∞
      else
          v.LLP ← ∞
          for each edge v→w
              v.LLP ← max{v.LLP, ℓ(v→w) + w.LLP}
  return s.LLP
```