Dynamic Programming

Text Segmentation

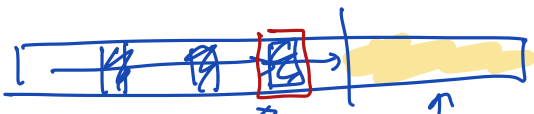
Decisions: 

Subproblems: Suffixes

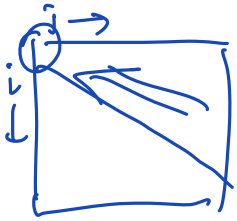
Eval order 

Prefixes

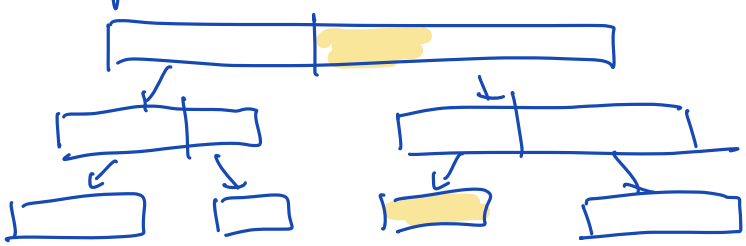
LIS:

Decisions: 

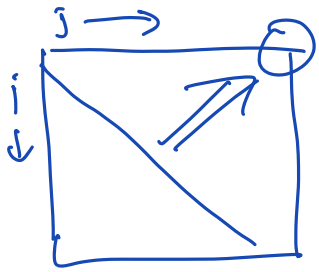
Subproblems: sentinel$_i$ + suffix$_j$



Woodcutters problem:



Subproblems = intervals $[i \ldots j]$

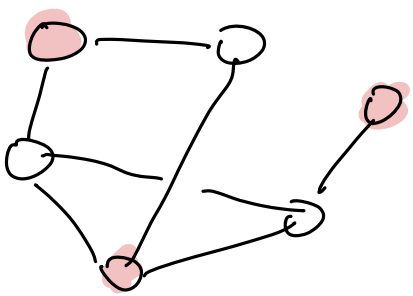


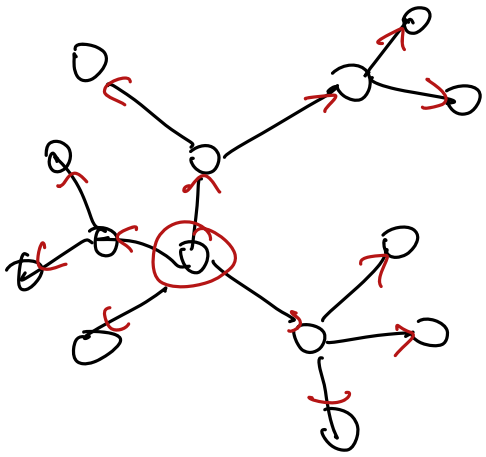
Maximum Independent Set

NP-hard

Fast algos for some special cases

TREES!



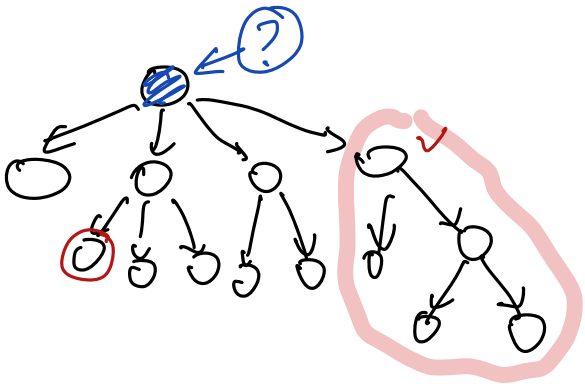


Tree = Connected acyclic graph.

Choose a root node

Direct edges away from root

Rooted tree = node with a set of rooted (sub)trees.



Subproblem = vertex

Define MIS(v, p)

size of max independent set in the subtree rooted at v

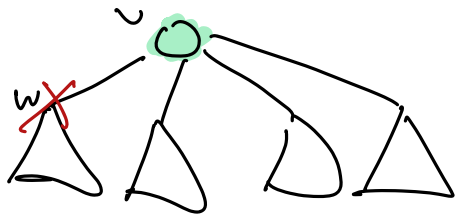
if parent of v is included (p=True)
parent of v is excluded (p=False)

This works but awkward ⟹

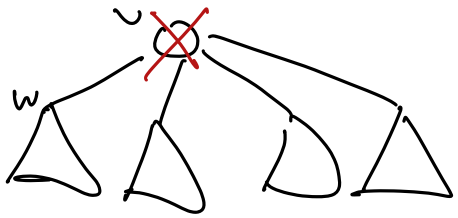
Define MIS yes(v) = size of largest ind. set in subtree rooted at v that includes v

MIS no(v) = size of largest ind. set in subtree rooted at v that excludes v

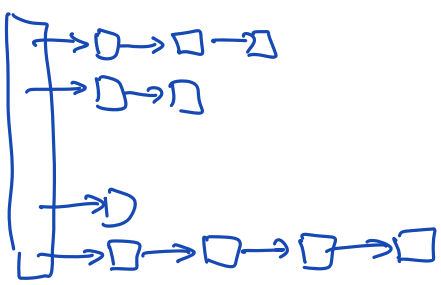
$$MIS_{yes}(v) = 1 + \sum_{w \text{ of } v} MIS_{no}(w)$$

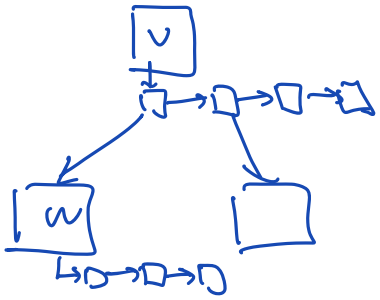


$$MIS_{no}(v) = \sum_{w \text{ of } v} \max\{MIS_{yes}(w), MIS_{no}(w)\}$$



We want $\max\{MIS_{yes}(root), MIS_{no}(root)\}$

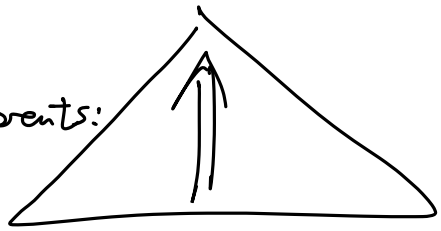




Memoize into the given tree data structure

v. MIS$_{yes}$
v. MIS$_{no}$

Evaluate children before parents:



① Reverse BFS: level by level
② DFS at root
Postorder

$O(n)$ ← # nodes

Recurrence defines a dependency graph
nodes = subproblems edges = recursive calls
MUST be acyclic

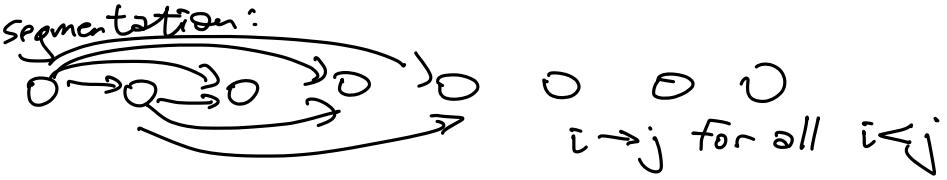
```
MEMOIZE(x):
  if value[x] is undefined
      initialize value[x]

      for all subproblems y of x
          MEMOIZE(y)
          update value[x] based on value[y]
      finalize value[x]
```

```
DFS(v):
  if v is unmarked
      mark v
      PREVISIT(x)
      for all edges v→w
          DFS(w)

      POSTVISIT(x)
```

Text segmentation:



$i \to j$ for all $i < j$

```
DYNAMICPROGRAMMING(G):
  for all subproblems x in postorder
      initialize value[x]
      for all subproblems y of x
          update value[x] based on value[y]
      finalize value[x]
```

Longest Path in DAG

LLP(v) = length of longest path $s \to v$