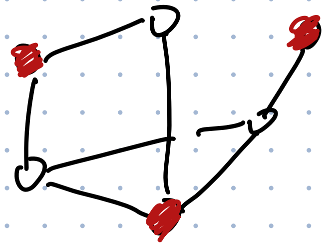
Dynamic Programming

Max Independent Set

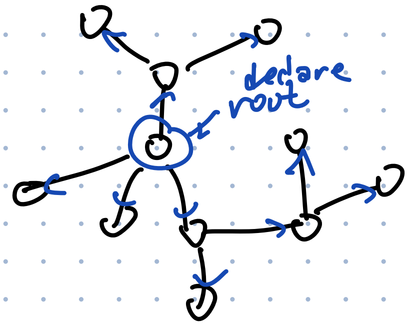
NP-hard

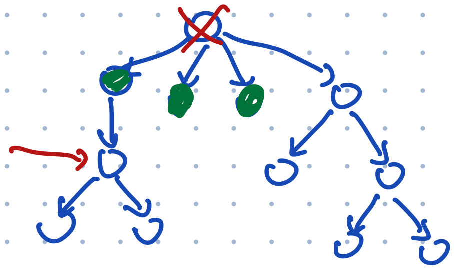


max # vertices in G with no edges between them

Trees

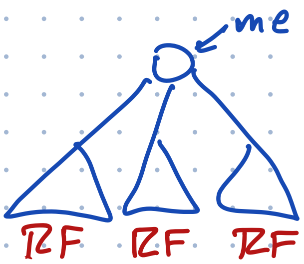
connected
no cycles





rooted tree

node + set of rooted trees



Intuitively: subproblem = subtree = node

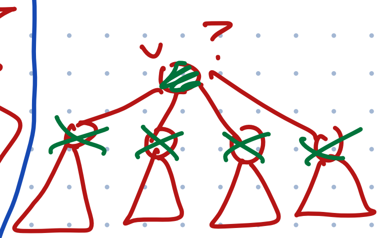
LIS(v, p) = size of largest ind set in subtree rooted at v ~~assuming parent(v) is already in IS~~

where v must be excluded if p=TRUE

we want $\boxed{\text{LIS(root, FALSE)}}$

$$LIS(v,T) = \sum_{w \text{ child of } v} LIS(w,F)$$

$$LIS(v,F) = \max \begin{cases} \sum_{\text{child } w} LIS(w,F) \\ 1 + \sum_{w} LIS(w,T) \end{cases}$$

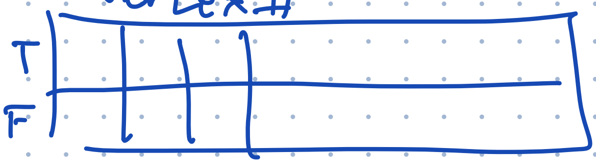


when v is a leaf $\sum_{w} = 0$

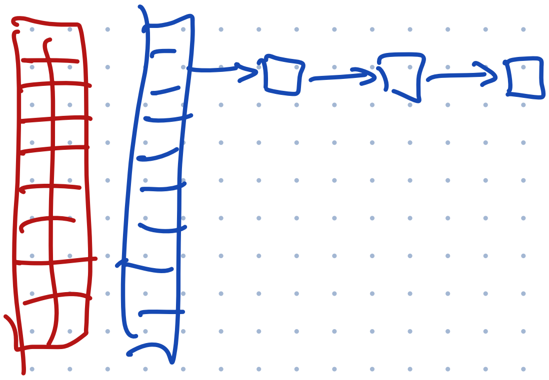
Memoize?

2d array

vertex #



Tree is represented in adj list



But what if T is a ptr-based data structure?

1 hash table

2 in the tree data struct

⟹ v.allowed
v.forbidden

MEMOIZE INTO THE TREE

Eval order? Reverse level order, postorder — depth-first search

Time: read from memo structure ≤ 3 times per node

write ≤ 2 times per node

$\Rightarrow$ O(n) time

At each node v in postorder
compute LIS(v, –) from (LIS(w, –) for children w)

Equivalently: LIS(v) returns both values

1 DP via postorder

2 Memo recursion via DFS

3 Recursion via DFS returning multiple values

dependency graph

Equivalent to DFS if G is a dag

```
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]
```

return value[x]

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

        POSTVISIT(v)
```

(even if G is not a dag)

```
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]
```

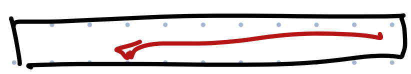
= reversed top. order

String splitting

G: node for each ~~prefix~~ suffix / index i

edge $i \to j$ whenever $i<j$

transitive tournament



array scanning

$O(n^2)$

DFS in dag with
n verts and $O(n^2)$ edges

LIS: $LIS(i)$ = length of LIS of $A[i..n]$ including $A[i]$

$E = \{i \to j \mid i<j \text{ and } A[i] < A[j]\}$

Sequence DP ⇄ optimal path thru dependency dag

Longest path in a dag G from s to t

$LLP(v)$ = length of longest path in G starting with v

$$LLP(v) = \begin{cases} 0 & \text{if } v = t, \\ \max\left\{\ell(v \to w) + LLP(w) \;\middle|\; v \to w \in E\right\} & \text{otherwise} \end{cases}$$

$-\infty$ if v is a sink but $v \neq t$

$\boxed{\max \emptyset = -\infty}$

```
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
```

$O(V+E)$

```
LongestPath(s, t):
    for each node v in postorder
        if v = t
            v.LLP ← 0
        else
            v.LLP ← −∞
            for each edge v→w
                v.LLP ← max{v.LLP, ℓ(v→w) + w.LLP}
    return s.LLP
```