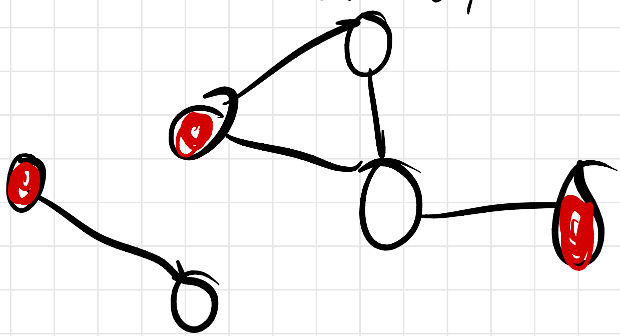
Independent set of a graph is a subset of the vertices with no edge between two members of the subset

maximum independent set:

given a graph $G = (V, E)$.

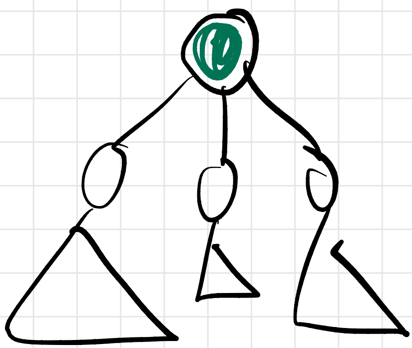
Find an independent set of max size.



Suppose we're given a tree T (an acyclic connected graph)

n: # vertices

Let's root T. Ask if the root should be in the max ind set.



If no to root, recursively find max set in each child subtree.

If yes to root, we cannot take children, so find max ind sets in grandchild subtrees

MIS(v): size of max ind set in subtree rooted at v.

$$MIS(v) = \max\left\{ \sum_{w \downarrow v} MIS(w),\ 1 + \sum_{w \downarrow v} \sum_{x \downarrow w} MIS(x) \right\}$$

"w is a child of v"

Subproblems: One per vertex v in T.

Memoization: $v.MIS$ for each node v in T

Dependencies: Children + grandchildren.

Eval order: postorder

r: root of T

Need to return $MIS(r)$

Space: $O(n)$

Time: $O(n)$

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

Returns MIS(v)

Class Scheduling:

Take as many classes as possible so no two overlap in time.

(all in one day)

Given $S[1..n]$: start times

$F[1..n]$: finish times

Class i starts at $S[i]$ & finishes at $F[i]$.

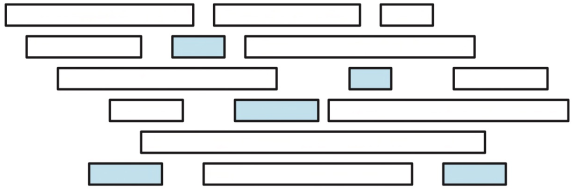
$(0 \le S[i] < F[i])$

Want a maximal conflict-free schedule X.

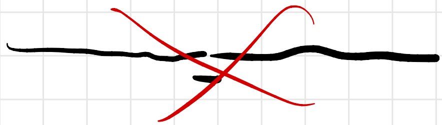
$\underset{\text{size}}{\text{max}}$ $X \subseteq \{1, \ldots, n\}$ such that

for each $i, j \in X$,

either $S[i] > F[j]$ or $S[j] > F[i]$



Greedy strategy:



not shortest class

Earlies to finish ✓

Lemma: At least one maximal conflict free schedule includes the class that finishes first.

Let f be the class that finishes first. Let X be a maximal conflict free scheduling.

If $f \in X$, we're done!

Otherwise, let g be the first class to finish <u>in X</u>.

f finishes before g, so

f does not conflict with any class in $X \setminus \{g\}$.

So remove g & replace it with f to get X'.

X' is conflict free & $|X'| = |X|$ ✓

So, we safely grab f as a greedy choice.

Remove those that conflict from input set.

"Recursively" find best set from rest of the input.

$O(n \log n)$

$O(n)$

```
GreedySchedule(S[1..n], F[1..n]):
    sort F and permute S to match
    count ← 1
    X[count] ← 1
    for i ← 2 to n
        if S[i] > F[X[count]]
            count ← count + 1
            X[count] ← i
    return X[1..count]
```

So $O(n \log n)$ total.

Greedy Algorithms:

Backtracking without going back.

Make first decision without trying all choices.

Recurse.

Done!

- easy to think of plausible greedy algorithms
- much harder to pick a correct one & justify it

Proof is usually an exchange argument:

1) Start with some optimal solution X.

If X agrees with our greedy choice, great!

2) Otherwise, do some kind of exchange so our choice goes into the optimal solution.

3) Argue solution is still valid + optimal after the exchange.

You usually want to do dynamic programming instead.

I will warn you if you <u>should</u> be greedy.