CSC791/495

Dr. Blair D. Sullivan

September 14, 2017

Dynamic Programming (DP)

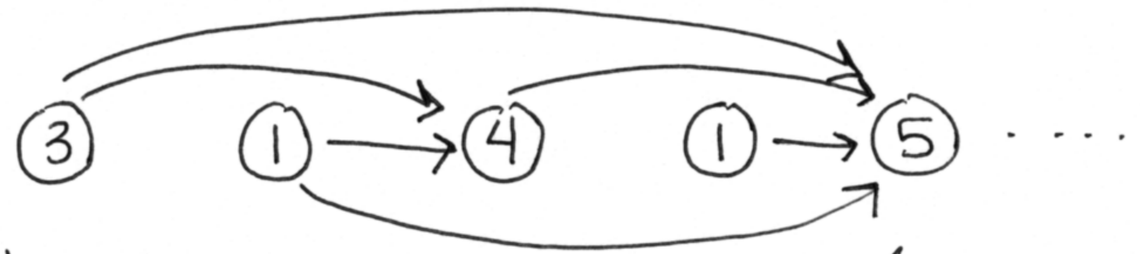
Problem Given a sequence of numbers $a_1, \ldots, a_n$, find an increasing (strictly >) subsequence (in order) of max length.

Example: 3 1 4 1 5 9 2 6 5 → 3 4 5 9 (length 4)

Approach check all subsequences? 2^n ☹

Q: what if we knew the answer for all but last entry? Not enough – but *if* we knew longest ending at *every* element before 5, this would be sufficient

Model "compatibility" w/ a DAG G



what is a longest increasing subseq. in G?

It's a longest path. Define $L(j)$ = longest path ending at vertex j (assoc. w/ a_j)

$$L(j) = 1 + \max\{L(i) : (i,j) \in E\}, \quad L(1) = 1$$

Time Complexity: $O(n)$ to sweep over j & $O(n)$ to compute max $\Rightarrow$ $O(n^2)$

Solution Recovery: store prev(j) & use backtracking at end. Same time complexity.

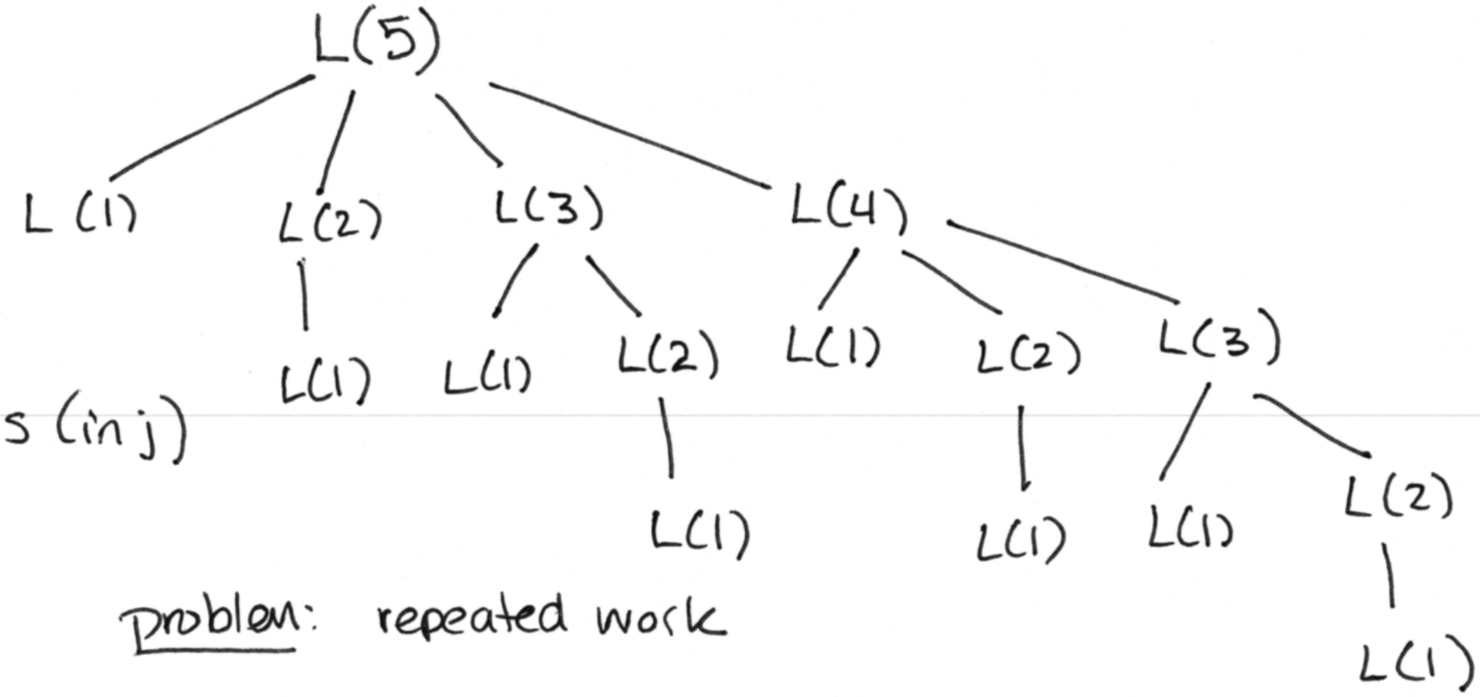
* Fundamental idea of dynamic programming: break down into (bounded) subproblems so you can combine solutions cleverly.

Why not recursion?

$$L(j) = 1 + \max\{L(i) \mid (i,j) \in E\}$$

what if DAG had all possible edges? (sorted seq).

If we apply recursion:



how big can this tree be?

exponentially many nodes (in j)

Problem: repeated work

can we fix it? store the results (memoization)

typically, we save results in a hash table (indexed by j)

& at each recursive call (1) check hash table (2) compute & store if not there

new running time: $O(n^2)$ only visit each node once SAME as DP.

⚠ can be faster: don't necessarily solve all subproblems.

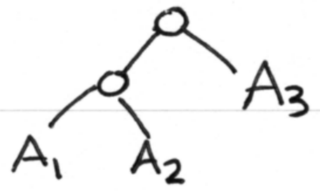
Linear Algebra Throwback

Suppose you want to multiply some matrices...

Problem: Given $A_1 \times A_2 \times \cdots A_n$ and the dimensions $m_i \times n_i$ of A_i, what is the best way to parenthesize the product (assuming $O(mn\ell)$ to multiply $m \times n$ by $n \times \ell$)?

observe: order matters $\underset{100\times 20}{A_1} \times \underset{20\times 100}{A_2} \times \underset{100\times 20}{A_3}$

$(A_1A_2)A_3$ or $A_1(A_2A_3)$



5x faster

brute force: $\binom{2(n-1)}{n-1}/n$

- Think recursively! $A_1 \times \cdots \times A_n \Rightarrow (A_1 \times \cdots A_j)(A_{j+1} \times \cdots \times A_n)$

so there is a choice of j for the last mult. which is optimal $(j = 1, \ldots, n-1)$

This creates two subproblems (smaller!) that must be solved.

How many subproblems? how many distinct (i, j) with $i \le j$ and $1 \le i, j \le n$ $O(n^2)$

how to parenthesize a sub-interval of matrices $(A_i, \ldots, A_j)$

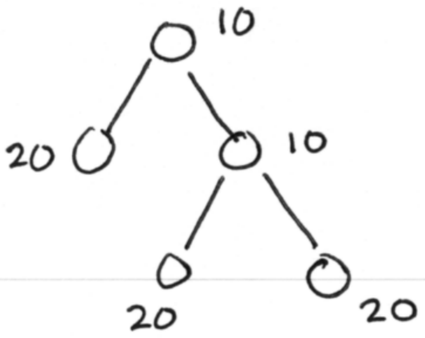
Overall (recursion w/ memoization) $O(n^3)$

bottom-up start w/ adjacent pairs: $j - i = 1$. Now you can solve $j - i = 2$...

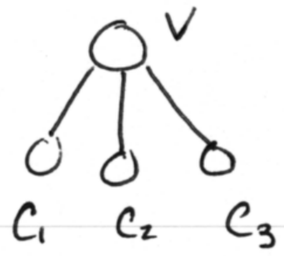
In-Class Exercise

Problem (MWIS) Given a graph G and $\omega: V(G) \to \mathbb{Z}^+ \cup \{0\}$, find the maximum weight of an independent set in G.

* Give a polynomial time algorithm for MWIS on trees (using DP).



Key observation:



if $v \in I \Rightarrow$

$c_1, c_2, c_3 \notin I$

Strategy: work from leaves up to the root, storing $A(v)$, $B(v)$

- $A(v)$: max weight of a set including v (in the subtree)
- $B(v)$: max weight excluding v.

root: chosen arbitrarily.

easily $O(n^2)$ { n nodes; $\leq n$ children

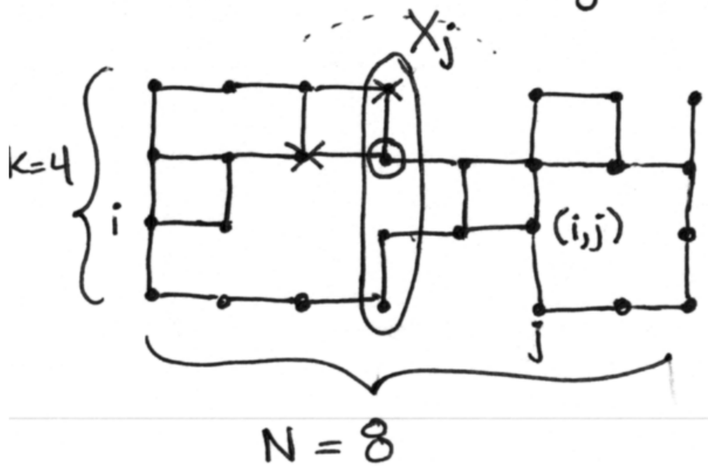
Show: $O(n)$ every node can be "used" at most twice $\Rightarrow$ not actually n^2.

If you finish early, use DP to get $O(n^2 2^n)$ for TSP (naive alg is $n!$)

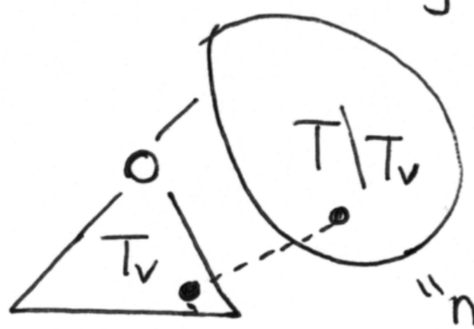
Gridlock

→ Trees are very "thin" - can we use a similar approach on "thick" graphs like grids?

* solve MWIS on graphs which are subgraphs of a k×N grid



Idea: In trees



"no cycles"

V breaks T into two pieces w/ no edges between. is a separator

now columns are natural separators. DP strategy: sweep from L to R across columns of grid. Let G_j = graph consisting of 1^{st} j columns

- what can a MWIS look like inside a column? subset, & independent

need a way to find max weight of an indep set in G_j w/ forbidden vertices in X_j

- what goes in a table? $c[j, Y]$ = max weight of an indep. set in $G_j \setminus Y$.

(j: curr. col.; Y: forbidden vertices in X_j)

Try to compute at j+1: iterate over all $S \subseteq X_{j+1}$, independent.

not quite ← only $S \subseteq X_{j+1} \setminus Y$

$c[j+1, \cdot]$: $Y \subseteq X_{j+1}$

$$c[j+1, Y] = \max_{\substack{S \subseteq X_{j+1} \setminus Y \\ S \text{ indep.}}} \{ w(S) + c[j, N(S) \cap X_j] \}$$

Natural question:

Can we combine our approaches to handle "fat trees"?

"Tree-like", more formally

Definition A **tree decomposition** of a graph $G=(V,E)$ (V: vertices) is a pair $(T, \{X_i\})$ ($\{X_i\}$: bags)

where $T=(I,F)$ is a tree and $\{X_i\}_{i=1}^{\ell}$ are subsets of V, AND

(I: nodes $= \{1, \ldots, \ell\}$)

① $\forall v \in V \quad \exists i$ s.t. $v \in X_i$ "cover all vertices"

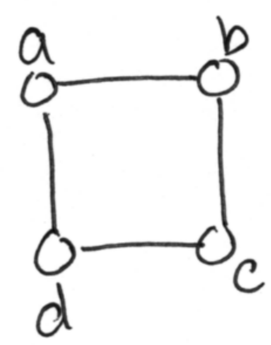
② $\forall (u,v) \in E \quad \exists j$ s.t. $\{u,v\} \subseteq X_j$ "cover all edges"

③ $\forall v \in V$, $\{i : v \in X_i\}$ form a connected subtree of T. "continuity"

ALT ③: If $v \in X_i$ and $v \in X_j \Rightarrow v \in X_k$ for all k on the path from i to j (in T)

Defn the **width** of a TD is $\max\{|X_i| - 1\}$

Defn the **treewidth** of a graph G ($tw(G)$) is the minimum width of any valid TD.



Try throw everything in one bag. $T = \bullet_1$

① ✓

② ✓

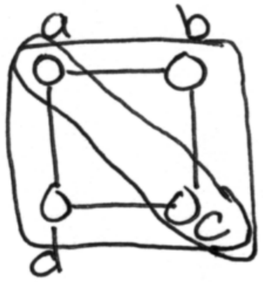
③ ✓ trivially

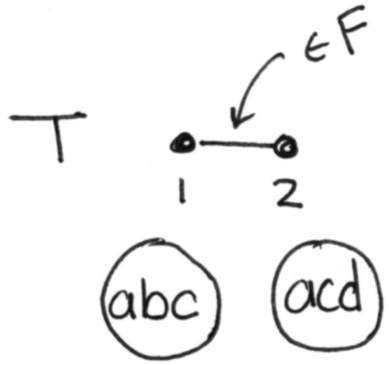
$tw(G) \leq n-1$

$X_1 = \{a, b, c, d\}$

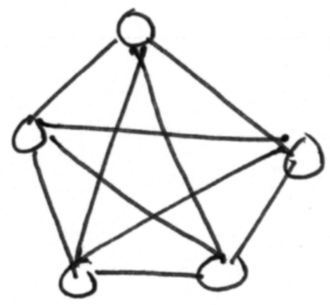
width: 3

Examples





width 2



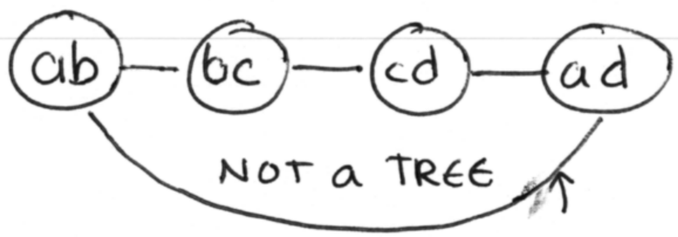
why is width $4(n-1)$ optimal? no separators

must cover all edges.
so add any vertex v
it has n-1 nbrs, all
of which are connected
(& can't be dropped) until
all others are added.

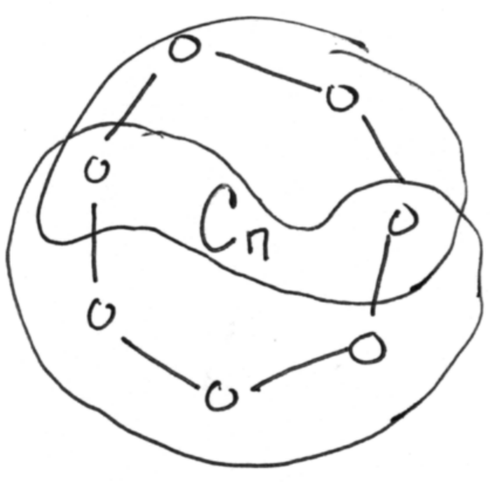
$$tw \geq \omega(G) - 1$$

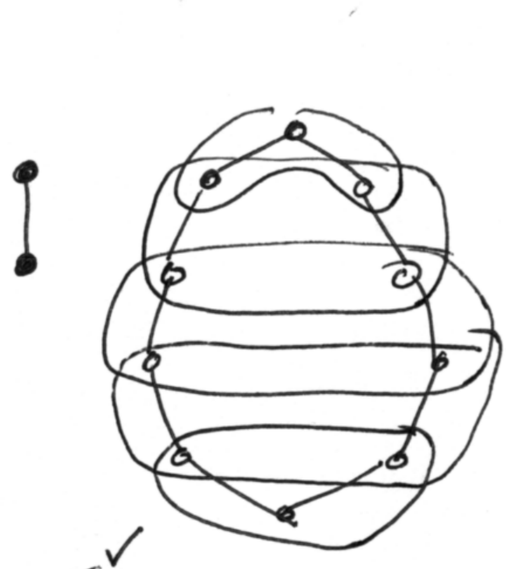
↑ clique #

Can we get width 1? NO ⇒ only trees have tw 1



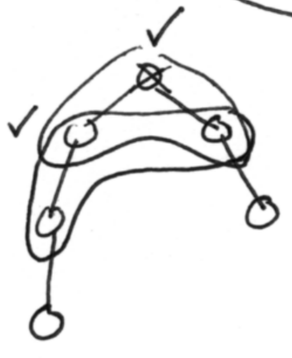
covers V
covers E



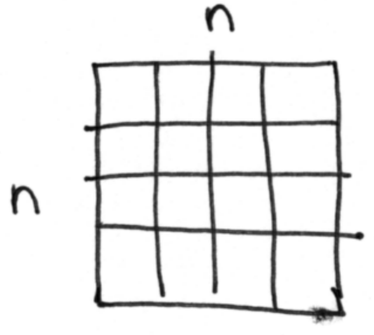


width 3

Can we do better?



try to build
width 2.



grids

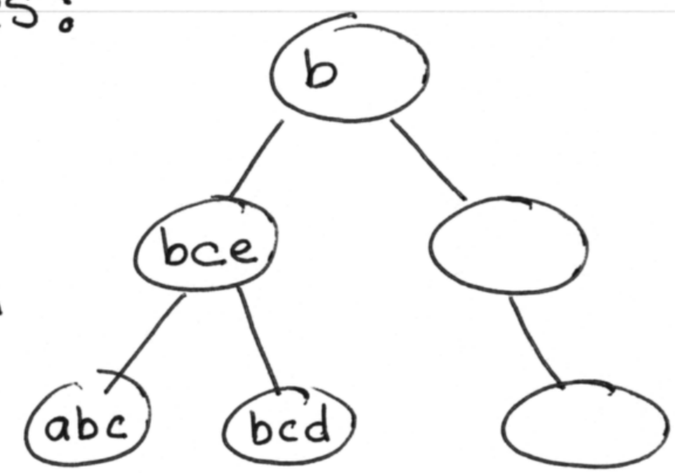
DP on Tree Decompositions

Idea: TDs come w/ natural separators (bags) & a nice order for subproblems (subtrees).

So can we extend our MWIS algorithm for trees?

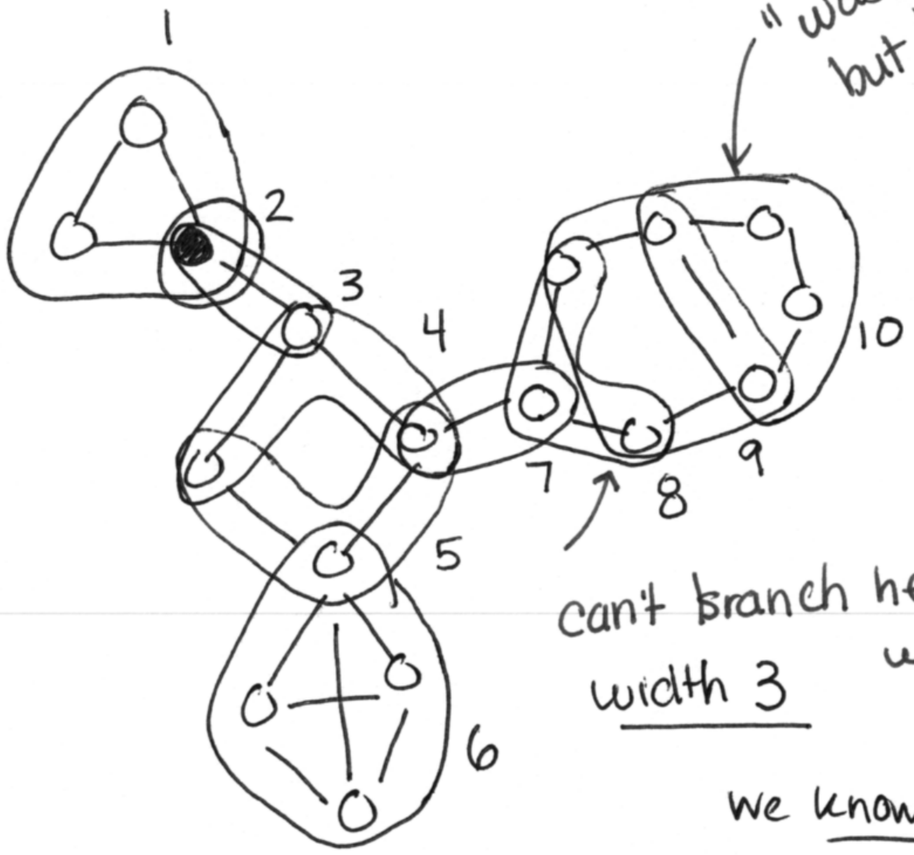
at a leaf: in a tree, MWIS including v & excluding v

why? b/c this allowed us to "extend" a partial solution at v to the subtree below it.



[postponed]

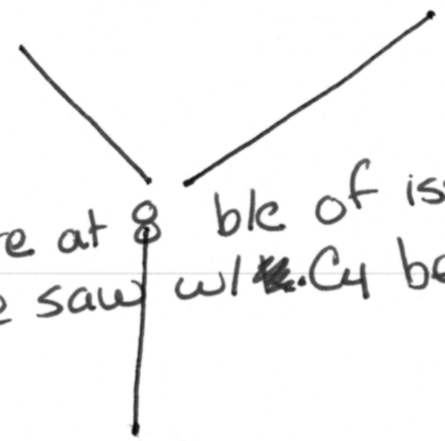
More Examples



"wasteful" but irrelevant to overall width given bag 6

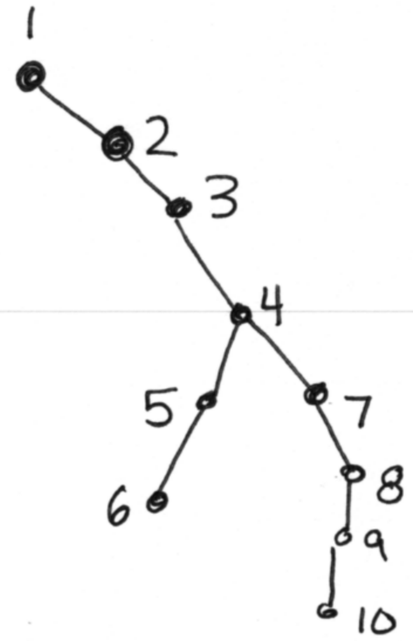
can't branch here at 8 b/c of issue we saw w/ K_4 before

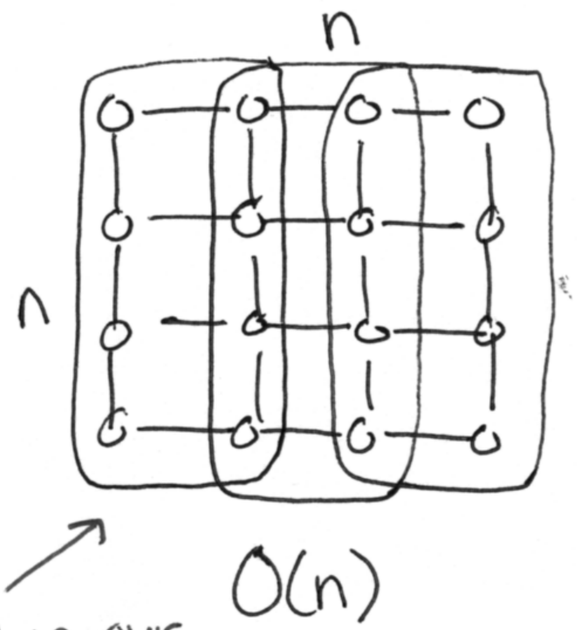
width 3



We know this is optimal b/c of K_4.

recall: we had a choice of where to connect 7.

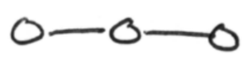




O(n)

bulk up our column idea to cover horizontal edges.

- width $2n-1$



exercise: $tw \leq n$

Sidebar: Computing treewidth (see section 7.6 of platypus book)

notes added to match discussion

- treewidth is NP-hard to compute, but still useful for parameterized algs:

  ① FPT w.r.t. treewidth [$O(n) \cdot tw^{O(tw^3)}$ algorithm by Bodlaender]

  ② exact (optimal) value not needed for DP - algorithms work with any tree decomposition (just might take longer) ⇒ approximation would be a win

  ③ The best fully poly-time approx. alg is $\Theta(\sqrt{\log OPT})$ by Feige et.al. (runs in poly(n)). There is also a 5-approximation that runs in $2^{O(k)} \cdot n$ by Bodlaender et.al.

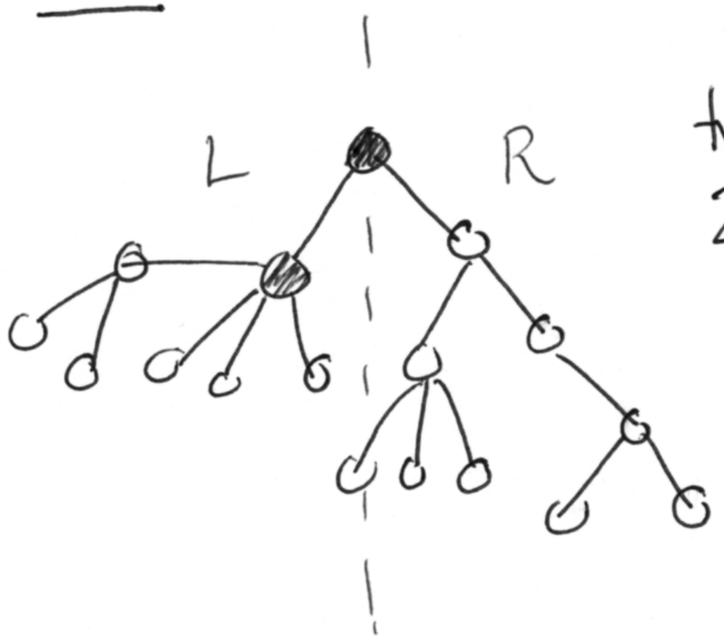
Cops & Robbers

Given a graph $G=(V,E)$ and k cops (w/ helicopters) plus one (infinitely fast) robber. Q: can the cops catch the robber?

Rules game is played in rounds. In each round, (everyone is located on vertices on G)
some of the cops may fly to another vertex (but they must file flight plans)
& robber may run along edges to any other vertex but not through cops. Robber is caught if cops can land on him.

Robber wins if he can stay free indefinitely.

Thm G has $tw \leq k \iff k+1$ cops win on G.



$tw=1$
2 cops, 1 robber

round 1: robber is somewhere, what should cops do?
→ put one at root ⇒ robber is either on L or R.
→ fly 2nd cop to child on side w/ robber
round 2: fly root cop to branch w/ robber.
repeat ? rounds depth$(T) \leq n$

Problems (postponed until next week!) – see next page

① Show OCT is FPT parameterized by treewidth.

② Show SAT is FPT parameterized by the treewidth of (a) its primal graph OR (b) its incidence graph.

Defn φ a CNF formula. The primal graph $G_p(\varphi) = (V_p, E_p)$ with $V_p = \{\text{variables}\}$ and $E_p = \{(x,y) \mid x,y \text{ co-occur in a clause}\}$. The incidence graph $G_i(\varphi) = (V_i, E_i)$ is a bipartite graph with $V_i = \{\text{variables}\} \cup \{\text{clauses}\}$ and $E_i = \{(x,C) \mid x \text{ a var occurring in } C\}$.

Reminders: (edited to reflect changes announced 9/15 on Slack)

① Proof Review Exercise ~~(instead of writeup) this week!~~
- due ~~next Friday~~ at 9 am Friday Sept. 29
- posted to github ~~this afternoon~~ Monday w/ full instructions.

② Opportunity Identification Project
- teams announced on Slack ~~today~~ Monday!
- report due 10/6

Full Details on Github Monday!

- weekly research log starts next week! (on your personal repos.)
- posters in class 10/13

Problems, take 2

READ ME!!

(*) There IS a proof writeup on treewidth this week! Due Fri, Sept 22 at 9 am (posted to Github Fri, Sept. 15)

 the homework assignment has two additional exercises that would be good practice with treewidth.

I also recommend trying to establish the treewidth of

1. outerplanar graphs
2. complete bipartite graphs
3. subgraphs (if $H \subseteq G$, is $tw(H) = tw(G)$? $\geq$? $\leq$?)