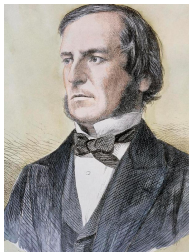


Independent Set

Boolean logic $x, y \in \{\text{TRUE}, \text{FALSE}\} = \{1, 0\}$

"x and y" $x \wedge y$
 "x or y" $x \vee y$
 "not x" $\bar{x}$ (or $\neg x$)



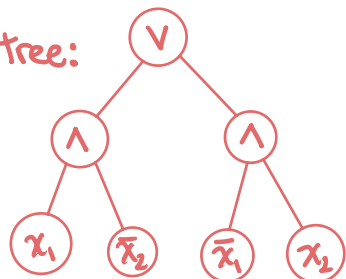
George Boole,
1815-1864

XOR ("exclusive-or")

(TRUE iff $x_1 \neq x_2$)

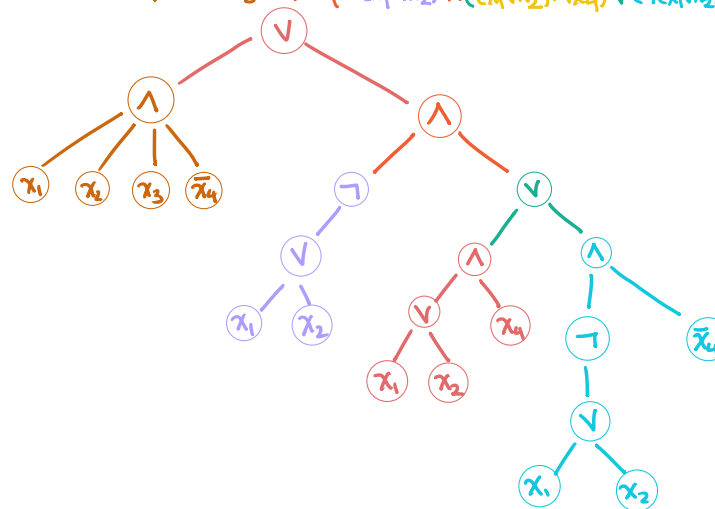
$$f(x_1, x_2) = (x_1 \wedge \bar{x}_2) \vee (\bar{x}_1 \wedge x_2)$$

as a tree:



Adding two bits $f(x_1, x_2, x_3, x_4) = \begin{cases} 1 & \text{if } x_1 + x_2 = x_3 x_4 \\ 0 & \text{otherwise} \end{cases}$

$$f(x_1, x_2, x_3, x_4) = (x_1 \wedge x_2 \wedge x_3 \wedge \bar{x}_4) \vee (\neg(x_1 \wedge x_2) \wedge ((x_1 \vee x_2) \wedge x_4) \vee (\neg(x_1 \vee x_2) \wedge \bar{x}_4))$$



"SAT Programming Language"

"x and y" $x \wedge y$
 "x or y" $x \vee y$
 "not x" $\bar{x}$ (or $\neg x$)

★ "z = x" $(z \wedge x) \vee (\bar{z} \wedge \bar{x})$

★ "if x then y else z" $(x \wedge y) \vee (\bar{x} \wedge z)$

So Boolean algebra is very expressive

Boolean Satisfiability

Input: Boolean formula $\varphi(x_1, x_2, \dots, x_n)$

• n variables • "size" m

e.g. $\varphi(\overset{\text{(false)}}{0}, \overset{\text{(true)}}{1}, \overset{\text{(false)}}{0}, \overset{\text{(true)}}{1}, \neg, \overset{\text{(false)}}{0}, \overset{\text{(true)}}{1}) = 0$

Goal: decide if φ is "satisfiable";
 i.e., if there exists $x_1, x_2, \dots, x_n \in \{0, 1\}$
 s.t. $\varphi(x_1, x_2, \dots, x_n) = 1$

"search problem": we can recognize an answer (e.g. satisfying assignment x) when we see it (by evaluating $f(x)$).

- at the very least: we can try all x .
- harder "search problem" than "over/under"

Boolean Satisfiability

Input: Boolean formula $\varphi(x_1, x_2, \dots, x_n)$

- n variables
- "size" m

e.g. $\varphi(\overset{\text{(false)}}{0}, \overset{\text{(true)}}{1}, 0, 0, \neg, \overset{\text{(false)}}{1}, 1) = 0$

Boolean logic $x, y \in \{\text{TRUE}, \text{FALSE}\} = \{1, 0\}$

"x and y" $x \wedge y$

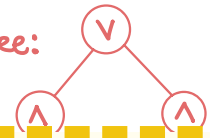
"x or y" $x \vee y$

"not x" $\bar{x}$ (or $\neg x$)

XOR ("exclusive-or") (TRUE iff $x_1 \neq x_2$)

$$f(x_1, x_2) = (x_1 \wedge \bar{x}_2) \vee (\bar{x}_1 \wedge x_2)$$

as a tree:



Goal: The Big Question

Is there a polynomial time algorithm for Boolean satisfiability?

x_2

$x_2 = x_3 x_4$

wise

$x_3 \wedge \bar{x}_4$

x_2

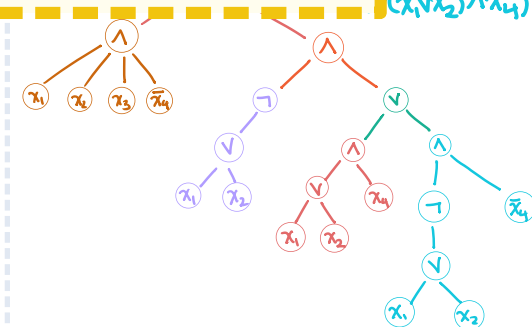
$x_2 \wedge x_4$

$(x_1 \vee x_2) \wedge \bar{x}_4$

answer (e.g. satisfying assignment x) when we see it (by evaluating $f(x)$).

- at the very least: we can try all x .

- harder "search problem" than "over/under"



"SAT Programming Language"

★ "z = x"

"both true or both false"
 $(z \wedge x) \vee (\bar{z} \wedge \bar{x})$

★ "if x then y else z"

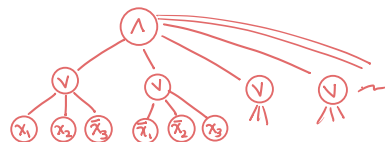
"(x=1, y=1) or (x=0, z=1)"
 $(x \wedge y) \vee (\bar{x} \wedge z)$

CNF (Conjunctive Normal Form)

"and-of-or's"

$$f(x_1, \dots, x_n) =$$

$$(\bar{x}_1 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee x_3) \wedge (x_1 \vee x_2) \wedge \dots$$



(CNF-) SAT problem

Given boolean formula f in CNF, decide if f is satisfiable

More modest Question

Is there a polynomial time algorithm for CNF-satisfiability?

"CNF-SAT Programming Language"

"x and y"

$$x \wedge y$$

"x or y"

$$x \vee y$$

"not x"

$$\bar{x} \quad (\text{or } \neg x)$$

★ "z = x"

$$(z \wedge x) \vee (\bar{z} \wedge \bar{x})$$

$$= (z \vee (\bar{z} \wedge \bar{x})) \wedge (x \vee (\bar{z} \wedge \bar{x}))$$

$$= (z \vee \bar{z}) \wedge (z \vee \bar{x}) \wedge (x \vee \bar{z}) \wedge (x \vee \bar{x})$$

$$= (z \vee \bar{x}) \wedge (x \vee \bar{z})$$

★ "if x then y else z"

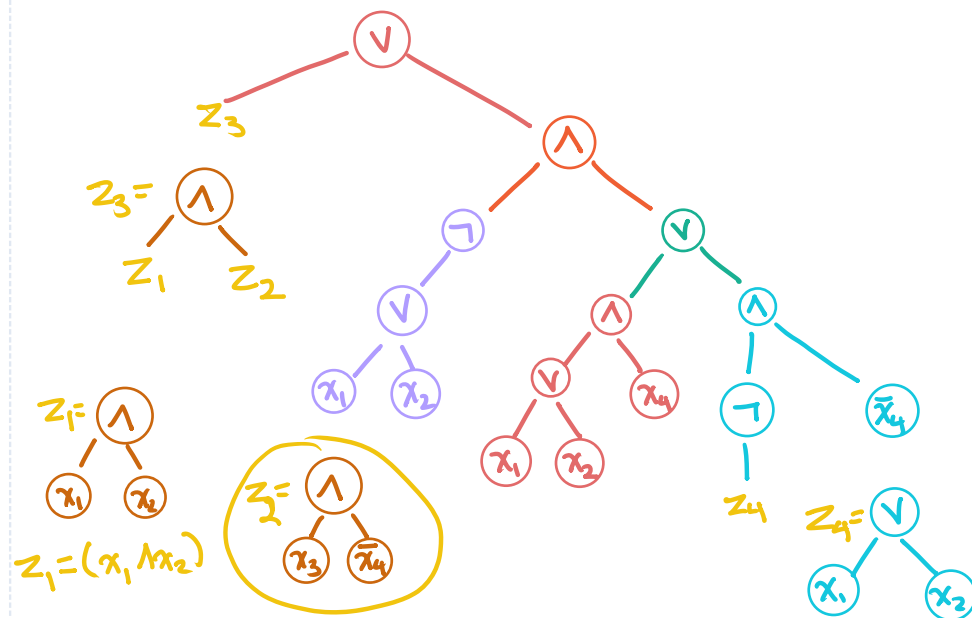
$$(x \wedge y) \vee (\bar{x} \wedge z)$$

$$= (x \vee \bar{x}) \wedge (y \vee \bar{x}) \wedge (x \vee z) \wedge (x \vee \bar{z})$$

$$= (y \vee \bar{x}) \wedge (x \vee z) \wedge (x \vee \bar{z})$$

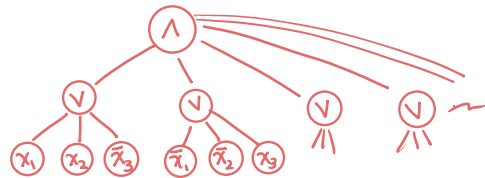
So CNF is also very expressive

Theorem: there is a poly-time algo for Boolean SAT iff there is a poly-time algo for CNF-SAT.



more special case: 3-SAT

like CNF-SAT, except w/ exactly 3 variables per clause.



$$f(x_1, \dots, x_n) =$$

$$(\bar{x}_1 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee x_3) \wedge (\sim \vee \sim \vee \sim) \wedge \sim$$

"clause"

Is there a polytime algo for 3SAT?

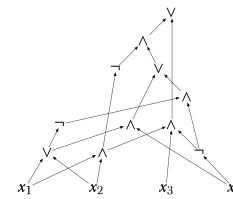
Theorem: there is a poly-time algo for Boolean SAT iff there is a poly-time algo for 3-SAT.

Conclusion

We want, but don't have, polynomial time algos for

Boolean SAT, CNF SAT, 3-SAT, or Circuit SAT

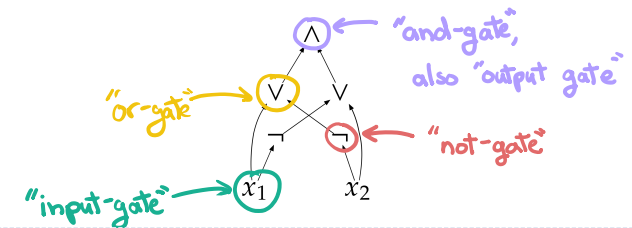
But we do know they are equivalent up to polynomial time!



(definition)

logical gates arranged as vertices in a directed acyclic graph

"gates": take some inputs bits, output 1 bit



Circuit SAT

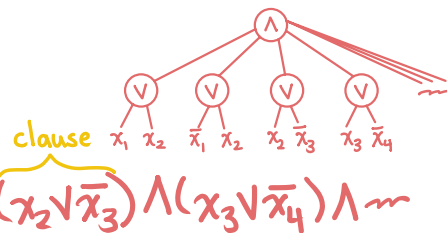
given a circuit, decide if it is satisfiable

Theorem: there is a poly-time algo for Boolean SAT iff there is a poly-time algo for Circuit SAT

2-SAT

like CNF-SAT, except w/ exactly 2 variables per clause.

$$f(x_1, \dots, x_n) =$$



$$(x_1 \vee x_2) \wedge (\bar{x}_1 \vee \bar{x}_2) \wedge (x_2 \vee \bar{x}_3) \wedge (x_3 \vee \bar{x}_4) \wedge \dots$$

Max 2-SAT

like 2-SAT, except goal is to satisfy
maximum number of clauses.
(not necessarily all)

Is there a polytime algo for Max 2-SAT?

let C be a sink SCC.

Complement $\bar{C}$ is a source SCC (why?)



Assign (all literals in) $C = \text{true}$, $\bar{C} = \text{false}$

Remove C and $\bar{C}$ and recurse on
remaining SCC's

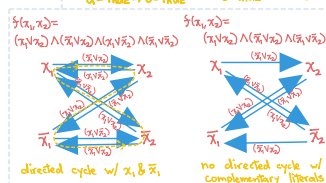
Key idea: assigning sinks to be true,
sources to be false
does not violate any edges

Implication graph

• vertex for each $x_i, \bar{x}_i$

• for clause $(a \vee b)$,

$\bar{a} \xrightarrow{(a \vee b)} b$ $\bar{b} \xrightarrow{(a \vee b)} a$
"a = TRUE $\Rightarrow$ b = TRUE" "b = TRUE $\Rightarrow$ a = TRUE"



all literals in SCC have same assignment



each SCC C has complement SCC $\bar{C}$
w/ negated literals & reversed edges

Polytime Algo

Hard as SAT

Abstain

2SAT

66%

44%

2

Max 2-SAT

60%

35%

5

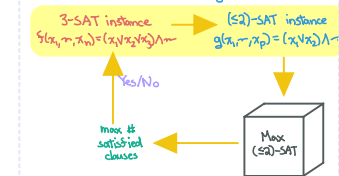
Fix a clause $(a \vee b \vee c)$
introduce new variable y

consider the 10 clauses

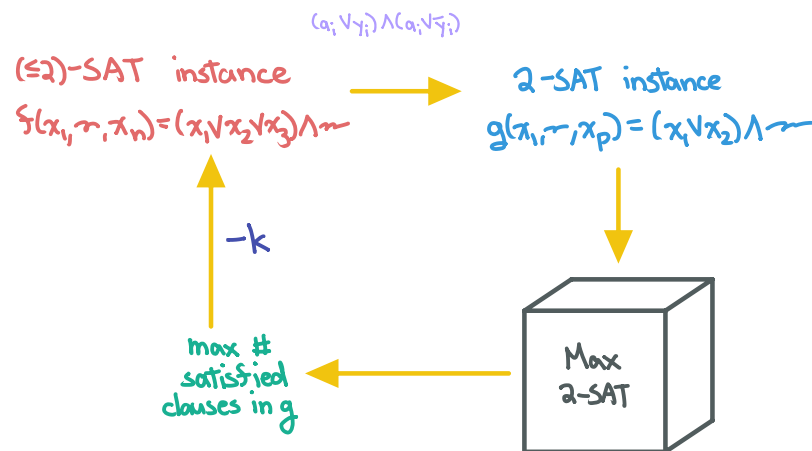
$(a), (b), (c), (\bar{a} \vee \bar{b}), (\bar{b} \vee \bar{c}), (\bar{c} \vee \bar{a}),$
 $(y), (a \vee \bar{y}), (b \vee \bar{y}), (c \vee \bar{y})$

$(a, b, c) =$	(f, f, f)	(t, f, f)	(t, t, f)	(t, t, t)
$(a), (b), (c)$	0	1	2	3
$(\bar{a} \vee \bar{b}), (\bar{b} \vee \bar{c}), (\bar{c} \vee \bar{a})$	3	3	2	0
$\max_{y \in \{t, f\}} (y), (a \vee \bar{y}), (b \vee \bar{y}), (c \vee \bar{y})$	3	3	3	4
total	6	7	7	7

Theorem: A polytime algorithm for max (2)-SAT
implies a polynomial time algorithm for SAT.



Theorem: A polytime algorithm for max 2-SAT
implies a polynomial time algorithm for SAT.



Subset Sum

Input: $x_1, x_2, \dots, x_n \in \mathbb{N}, T \in \mathbb{N}$
Goal: find a subset of x_i 's that sum to T
(decision version: decide if such a subset sum exists)
e.g. 14, 13, 17, 7, 18, 18, 10, 4, 11, 10
T=52

Another "search" problem

	Polytime Algo	Hard as SAT (CNF)	Abstain
Subset Sum	10%	30%	○

★Dynamic Programming★

Recursive Spec
(Subset Sum)
SS(i, S) = TRUE if there is a subset of $x_1, \dots, x_n$ summing to S, and FALSE otherwise

Subset Sum
Input: $x_1, x_2, \dots, x_n \in \mathbb{N}$
Goal: Find a subset of x_i 's that sum to T
(decision version: decide if such a subset sum exists)
e.g. 14, 13, 17, 7, 18, 18, 10, 4, 11, 10
T=52

Implementation

SS(i, S)
1. If $S < 0$ then return FALSE
2. If $S = 0$ then return TRUE
3. If $n = 0$ then return FALSE
4. Return $SS(i+1, S) \vee SS(i+1, S - x_i)$

Running time:
 $O(nT)$ subproblems
 $O(1)$ time per subproblem

 $O(nT)$ time overall

Theorem: A polytime algo for subset-sum implies a polytime algo for SAT

Subset Sum
Input: $x_1, x_2, \dots, x_n \in \mathbb{N}$
Goal: Find a subset of x_i 's that sum to T
(decision version: decide if such a subset sum exists)
e.g. 14, 13, 17, 7, 18, 18, 10, 4, 11, 10
T=52

Literals $x = \{x_1, \bar{x}_1, \dots, x_n, \bar{x}_n\}$ Long #'s $\{A_1, \bar{A}_1, \dots, A_m, \bar{A}_m\}$ and $\{B_1, \bar{B}_1, \dots, B_n, \bar{B}_n\}$
T = 33 m clauses 3 1 1 1 1 n variables
3-SAT instance $\phi(x_1, \dots, x_n) = (x_1 \vee x_2 \vee x_3) \wedge \dots$
Yes/No for $x_1, x_2, \dots, x_p \in \mathbb{N}$ T ∈ N
Subset Sum
Key idea: numbers are bit-strings too! use many digits each digit encodes 1 constraint

Key idea: numbers are bit-strings too!
use many digits each digit encodes 1 constraint

2. Start making #'s:
we use mtn digits base 4
for the literal x_j :

$A_j = \underbrace{0 \ 1 \ 1}_{m \text{ slots for each clause}} \underbrace{0 \ 0 \ 1}_{n \text{ slots for each variable}}$
(models) x_j

1. Recast 3SAT as "subset" problem
 $X = \text{literals } \{x_1, \bar{x}_1, x_2, \bar{x}_2, \dots, x_n, \bar{x}_n\}$
clause $(x_1 \vee \bar{x}_2 \vee x_3) = \text{set } \{x_1, \bar{x}_2, x_3\} \subset X$
Goal: Find SCX st.
a) for each clause $C_i, C_i \cap S \neq \emptyset$
b) for each x_j, S contains 1 of $\{x_j, \bar{x}_j\}$ (not $\bar{x}_j$)

for each clause C_i , set C_i th digit to 1 if $x_j \in C_i$, else 0
set " x_j th digit" to 1, else 0

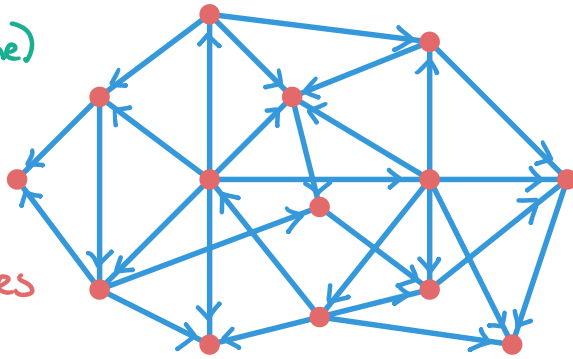
(vertices cannot repeat)

today: longest path problem

(eg. # edges)

find the (length of the) longest path in G

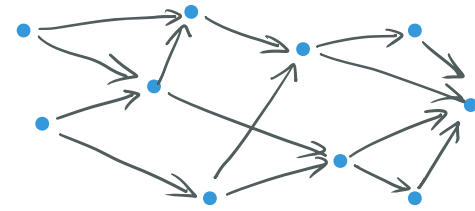
Hamiltonian path:
path visiting all vertices



does G have a Hamiltonian path?

related problems: longest walk, shortest paths

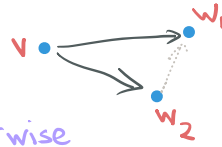
Longest walk in DAGs



Recursion

$LW(v)$ = length of Longest Walk from v

$$LW(v) = \begin{cases} 0 & \text{if } v \text{ is a sink} \\ 1 + \max_{(v,w) \in E} LW(w) & \text{otherwise} \end{cases}$$



Running time w/ caching

$$\sum_{v \in V} (\text{time for } LW(v)) = \sum_{v \in V} 1 + \deg^+(v) = O(m+n)$$

Three settings:

1. DAG

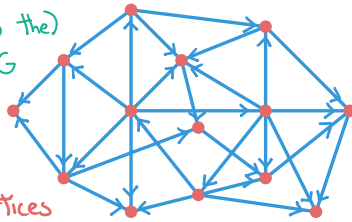
2. Directed

3. Undirected

today: longest path problem

find the (length of the) longest path in G

Hamiltonian path:
path visiting all vertices

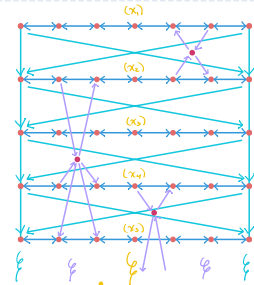


does G have a Hamiltonian path?

	Polytime Algo	Hard as SAT ^(CNF)	Abstain	Lions
DAG	90%	2		≥3
Directed	15%	85% - 1		
Undirected	0%	99%		

Theorem 7.1. A polynomial time algorithm for the Hamiltonian path problem implies a polynomial time algorithm for SAT.

$$\varphi(x_1, \dots, x_n) = (x_1 \vee \bar{x}_2) \wedge (x_2 \vee x_3 \vee \bar{x}_4) \wedge \dots$$

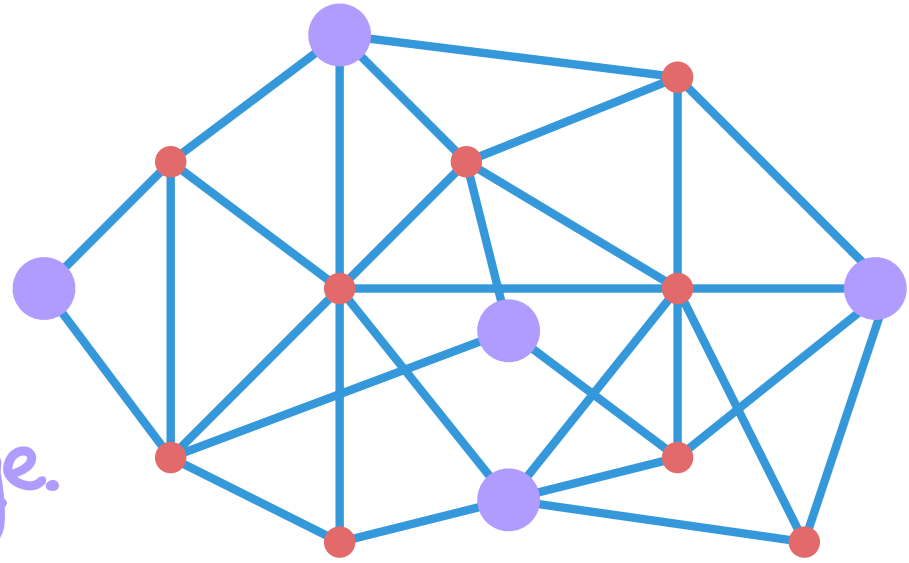


Independent set

Input: undirected $G=(V,E)$
(optionally) weights $w: V \rightarrow \mathbb{R}_{\geq 0}$

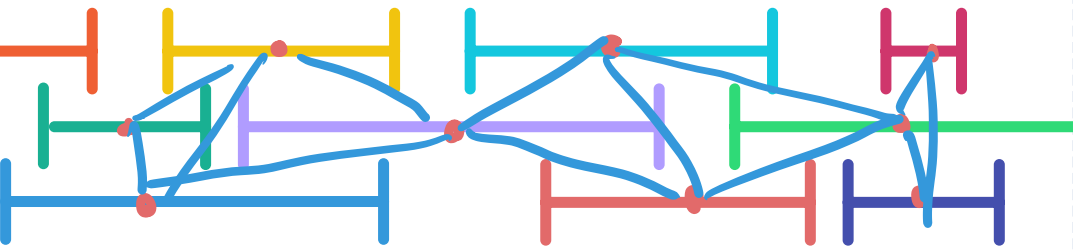
$S \subseteq V$ is independent if no $u,v \in S$ are connected by an edge.

Goal: max weight independent set
(decision version: is there an independent set of weight $\geq k$?)



Special cases:

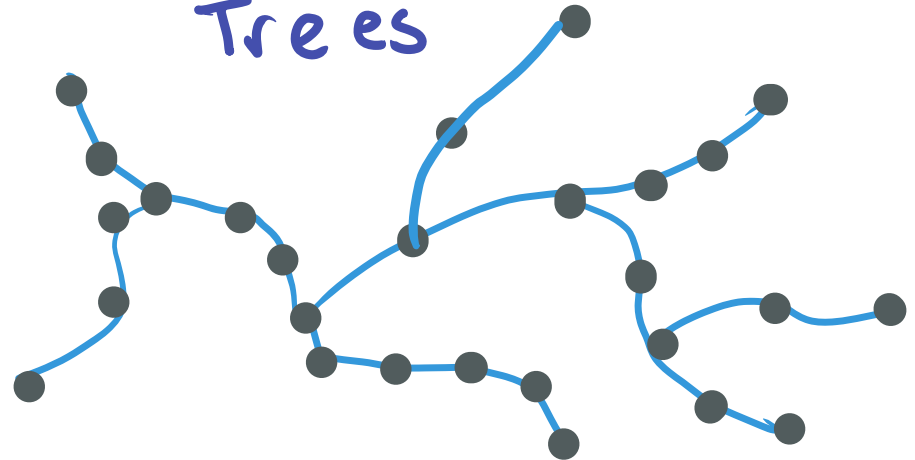
intervals



"independent" = non-overlapping



Trees

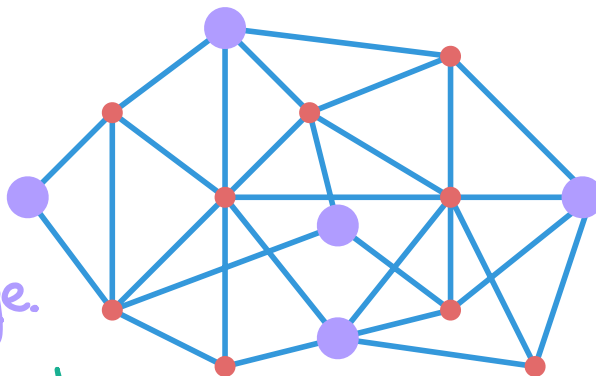


Independent set

Input: undirected $G=(V,E)$
(optionally) weights $w: V \rightarrow \mathbb{R}_{\geq 0}$

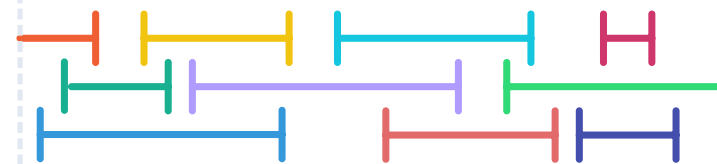
$S \subseteq V$ is independent if no $u,v \in S$ are connected by an edge.

Goal: max weight independent set
(decision version: is there an independent set of weight $\geq k$?)



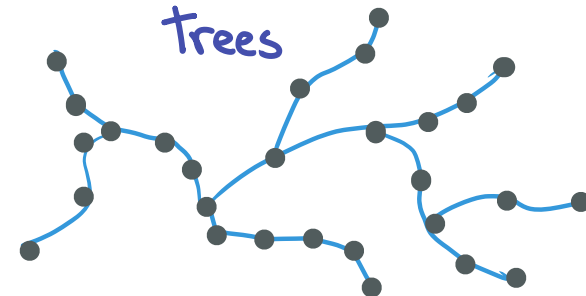
Special cases:

intervals



"independent" = non-overlapping

trees



Independent set in { graphs, intervals, trees }

Hard as SAT Polytime algo

100% → 3	3
20%	80%
2	100% → 2

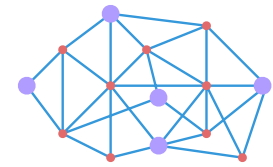
Abstain



Theorem: A polytime algo for max
(unweighted)
cardinality independent set implies a
polytime algo for SAT

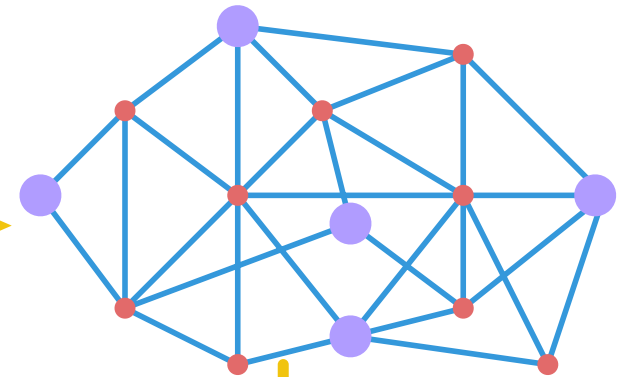
Proof setup

Independent set
Input: undirected $G=(V,E)$
(optionally) weights $w: V \rightarrow \mathbb{R}_{\geq 0}$

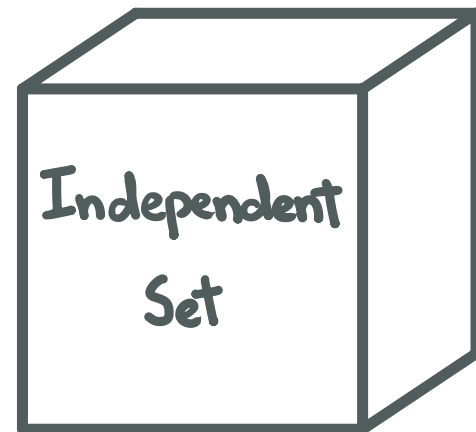


$S \subseteq V$ is independent if no
 $u,v \in S$ are connected by an edge.
Goal: max weight independent set
(decision version: is there an independent set of weight $\geq k$?)

3-SAT instance
 $\{ (x_1, \neg, x_n) = (x_1 \vee x_2 \vee x_3) \wedge \dots$

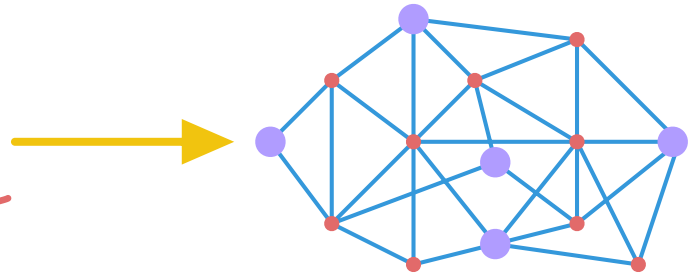


there is/isn't an
independent set
of size k



Goal: transform

3-SAT instance

$$\zeta(x_1, x_2, x_3) = (x_1 \vee x_2 \vee x_3) \wedge \dots$$


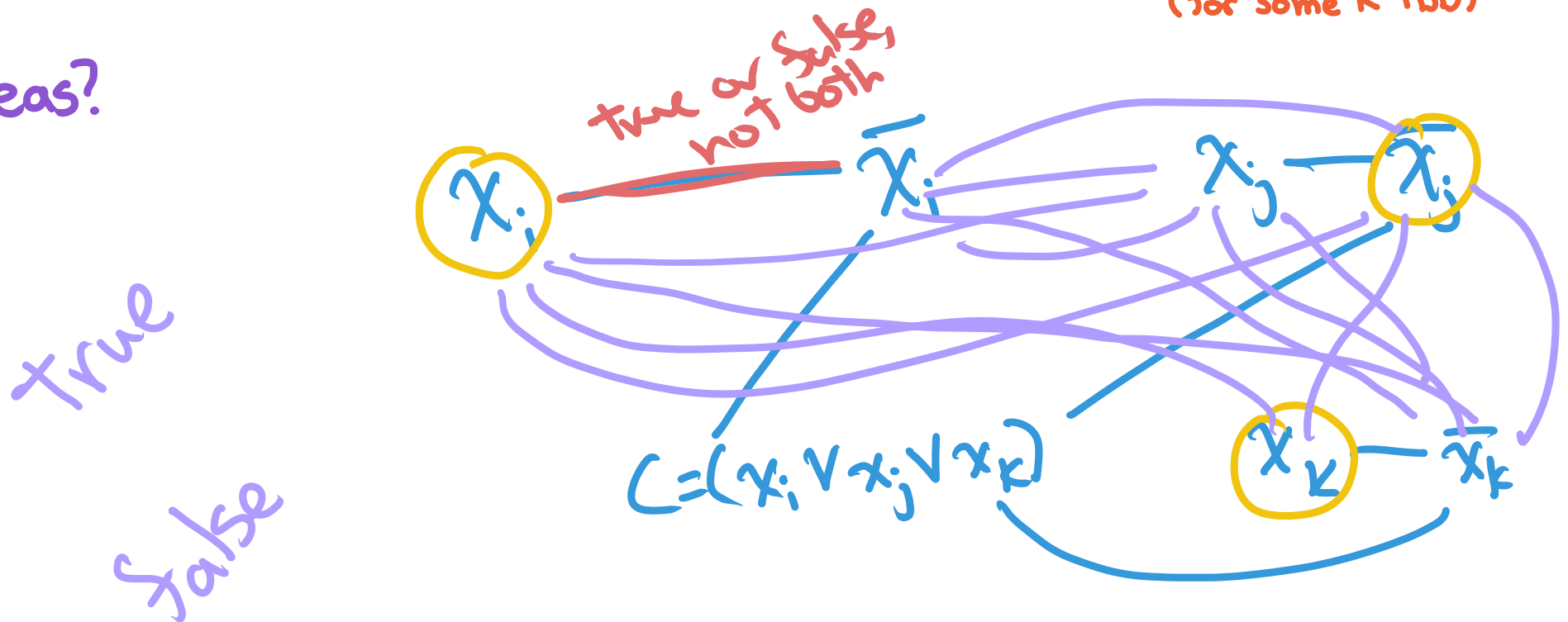
st. size of max indep. set informs satisfiability of f

e.g.

$$(\mathcal{F} \text{ is satisfiable}) \iff (\text{there is indep. set of size } k)$$

(for some k TBD)

ideas?



clause gadgets

SAT

Graphs

literals
(in clauses)



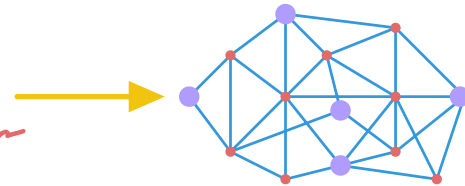
vertices

clauses



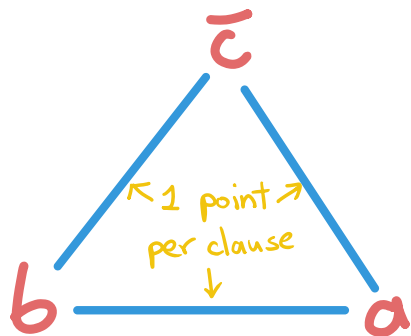
triangles

3-SAT instance
 $\mathcal{S}(x_1, x_2, x_3) = (x_1 \vee x_2 \vee x_3) \wedge \dots$

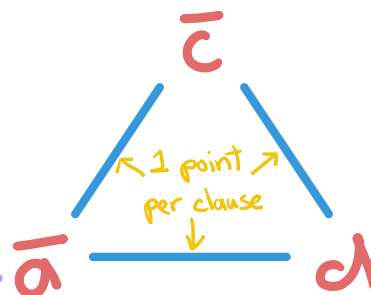


$\left(\mathcal{S} \text{ is satisfiable} \right) \Leftrightarrow \left(\text{there is indep. set of size } k \right)$
(for some k TBD)

$a \vee b \vee \bar{c}$

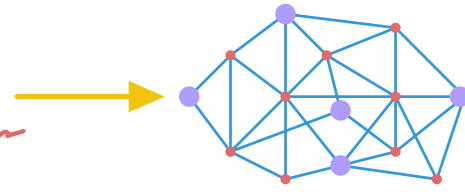


$\bar{a} \vee \bar{c} \vee d$



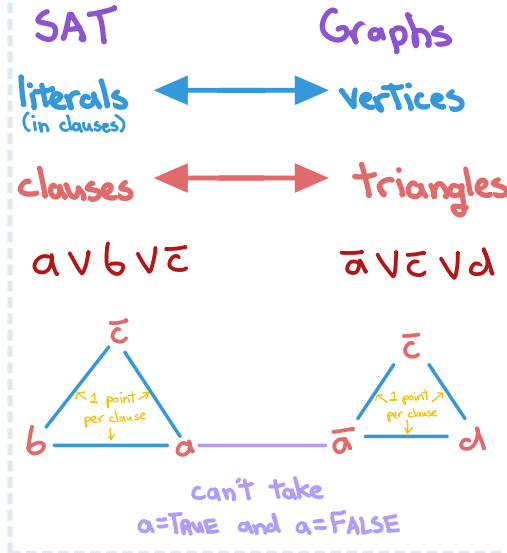
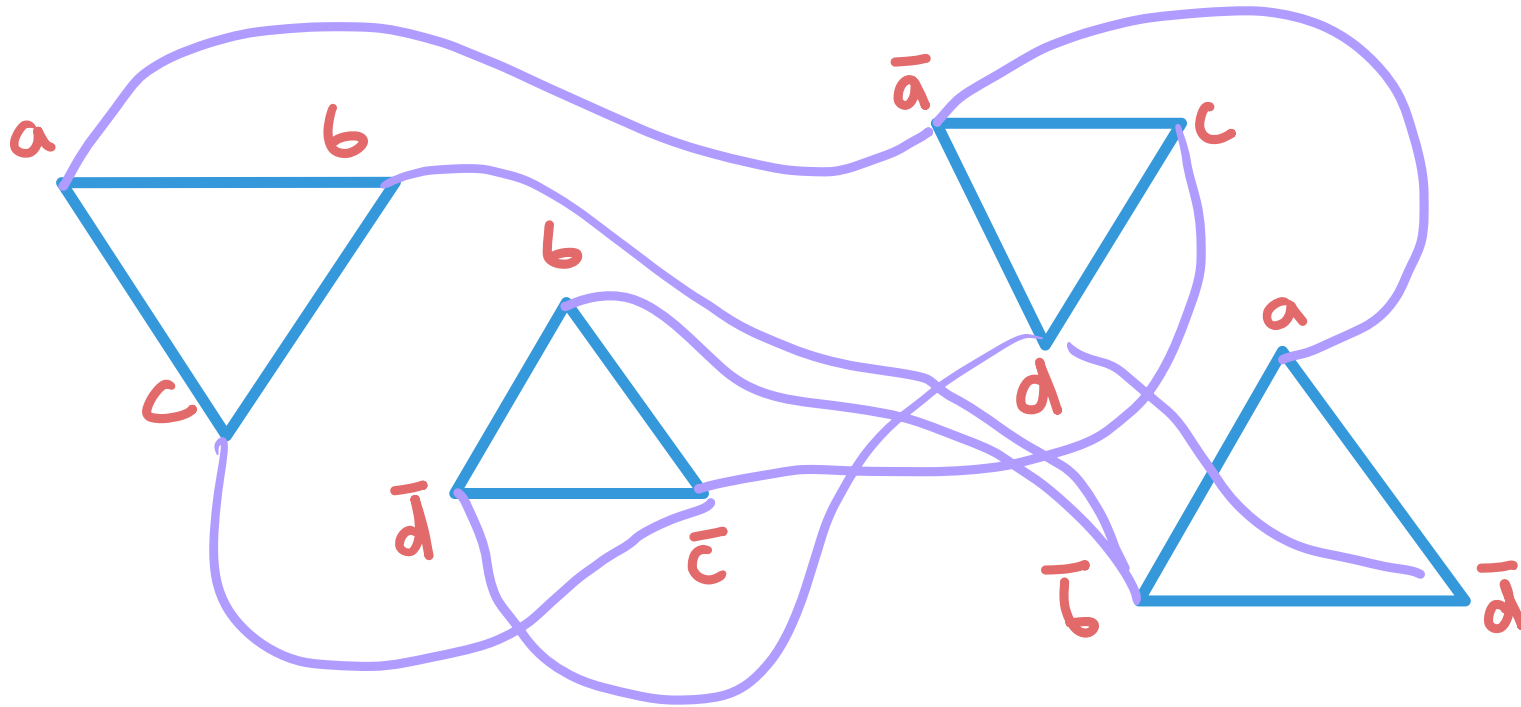
can't take
 $a = \text{TRUE}$ and $a = \text{FALSE}$

3-SAT instance
 $\mathcal{F}(x_1, x_2, x_3) = (x_1 \vee x_2 \vee x_3) \wedge \dots$



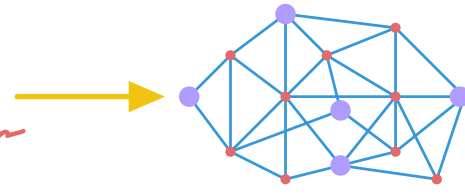
$(\mathcal{F} \text{ is satisfiable}) \iff (\text{there is indep. set of size } k)$
 (for some k TBD)

$$(a \vee b \vee c) \wedge (b \vee \bar{c} \vee \bar{d}) \wedge (\bar{a} \vee c \vee d) \wedge (a \vee \bar{b} \vee \bar{d})$$



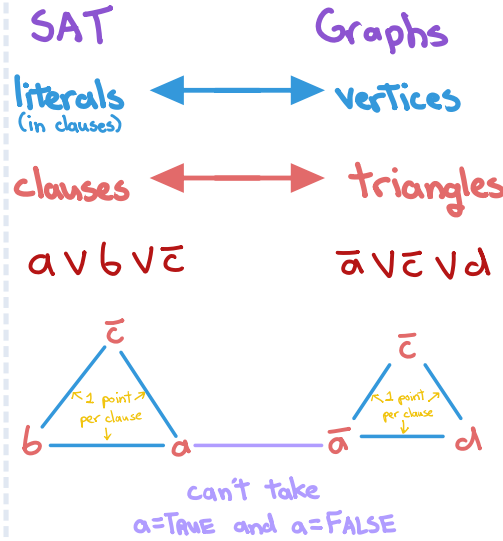
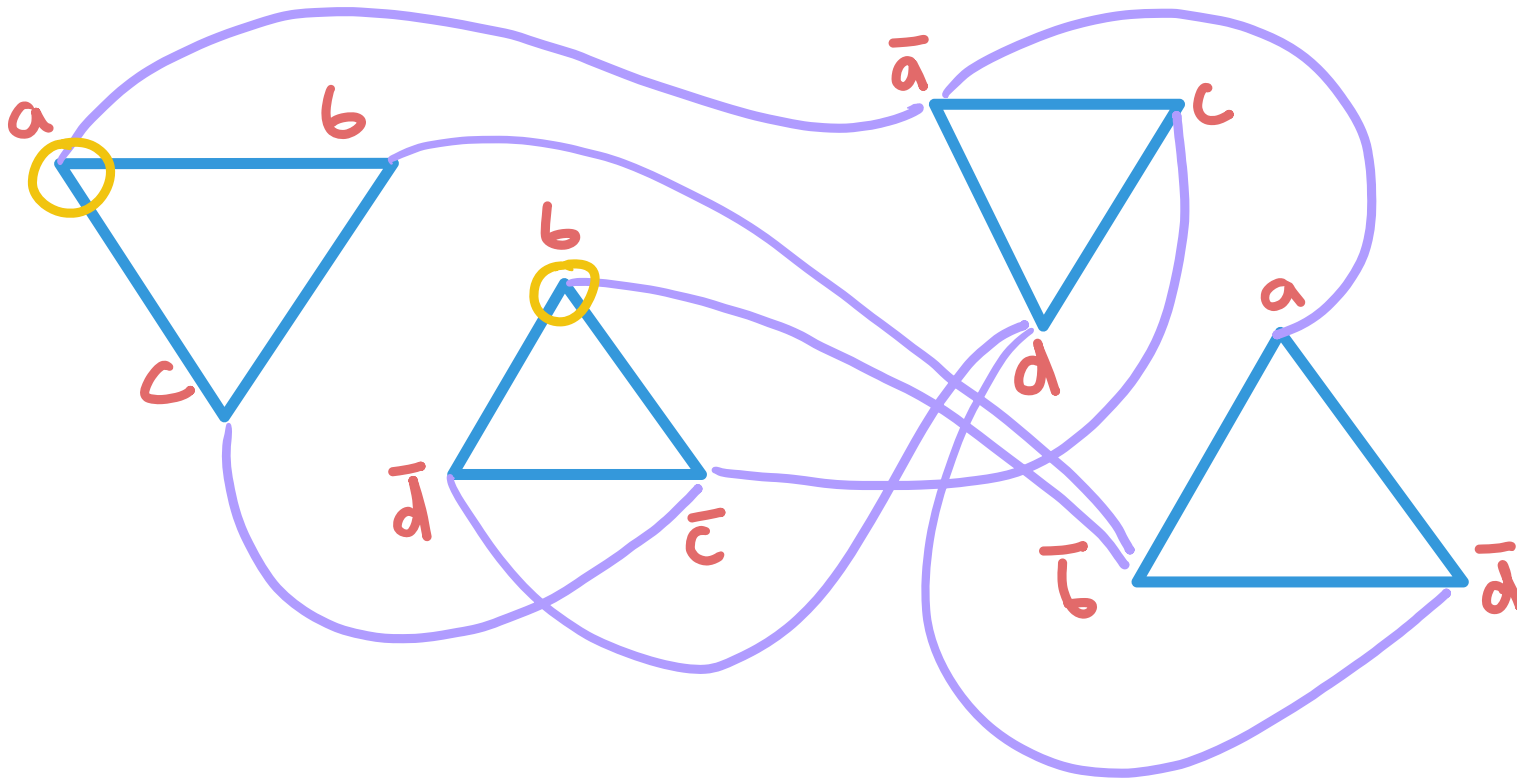
m clause $\rightarrow m$ triangles

3-SAT instance
 $\mathcal{F}(x_1, x_2, x_3) = (x_1 \vee x_2 \vee x_3) \wedge \dots$



$(\mathcal{F} \text{ is satisfiable}) \iff (\text{there is indep. set of size } k)$
 (for some k TBD)

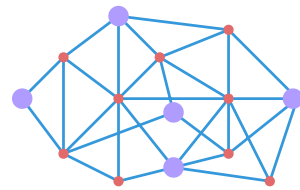
$$(a \vee b \vee c) \wedge (b \vee \bar{c} \vee \bar{d}) \wedge (\bar{a} \vee c \vee d) \wedge (a \vee \bar{b} \vee \bar{d})$$



$m \text{ clause} \rightarrow m \text{ triangles}$

claim: $\mathcal{F}$ is satisfiable iff there is an independent set of size m
 (where m is the # of clauses)

3-SAT instance
 $\mathcal{F}(x_1, \neg x_1, x_2) = (x_1 \vee x_2 \vee x_3) \wedge \dots$



$(\mathcal{F} \text{ is satisfiable}) \iff (\text{there is indep. set of size } k)$
 (for some k TBD)

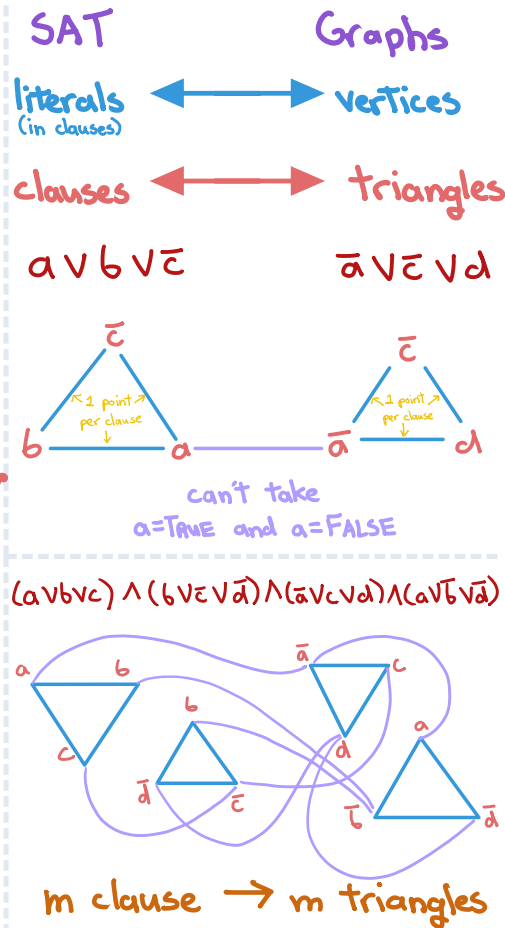
Proof. ($\Rightarrow$) Suppose $\mathcal{F}$ is satisfiable

for each clause of the form $(a \vee b \vee c)$, either a , b , or c is true. Take the vertex corresponding to the true literal

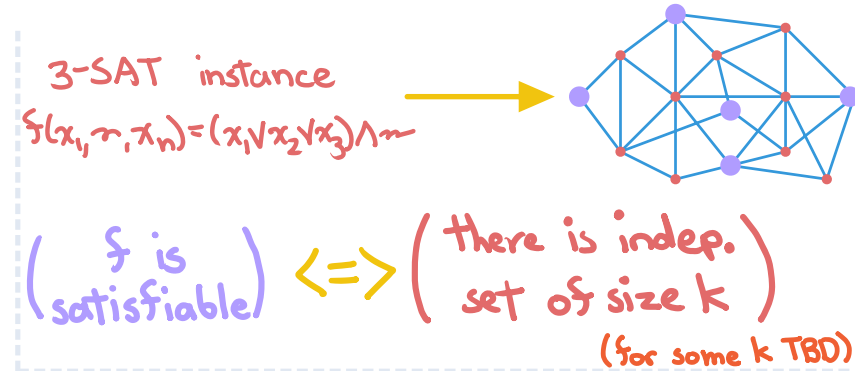
(a) $\Rightarrow m$ vertices

(b) clause edges $\checkmark$

(c) $x_i - \bar{x}_i$ edges are OK b/c assign only takes one of them

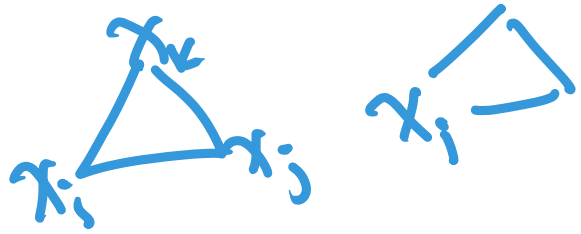


claim: $\mathcal{F}$ is satisfiable iff there is an independent set of size m
 (where m is the # of clauses)



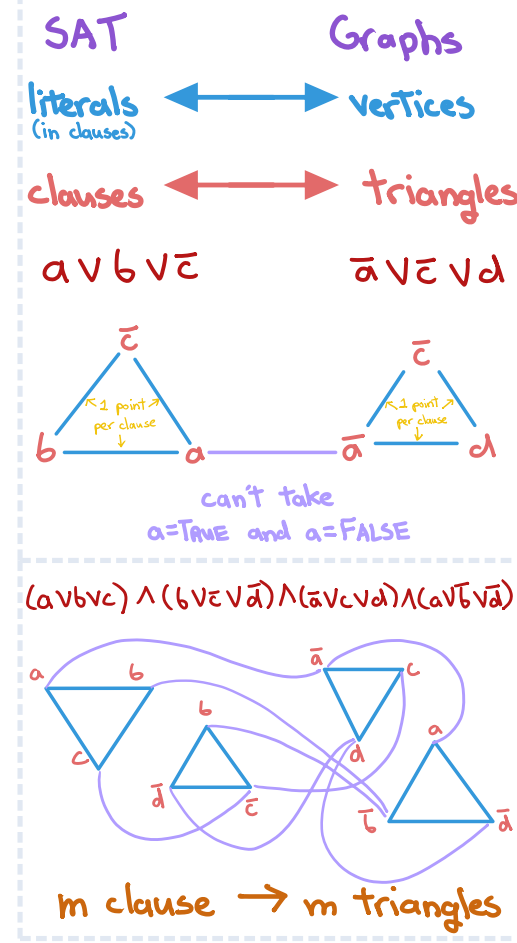
Proof. ($\Leftarrow$) Suppose there is an independent set of size m

for each set x_i ,



set $x_i = \text{true}$ if we select an " x_i " vertex
 set $x_i = \text{false}$ if we select a " $\bar{x}_i$ " vertex

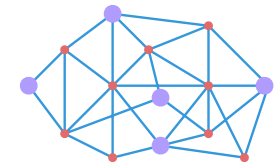
note that we can't have both x_i and $\bar{x}_i$ edges
 otherwise either



Theorem: A polytime algo for max
(unweighted)
cardinality independent set implies a
polytime algo for SAT

Proof setup

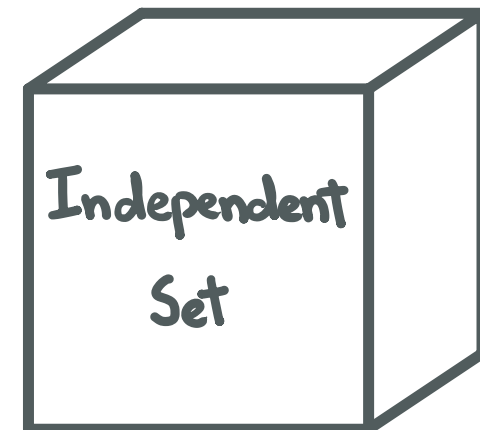
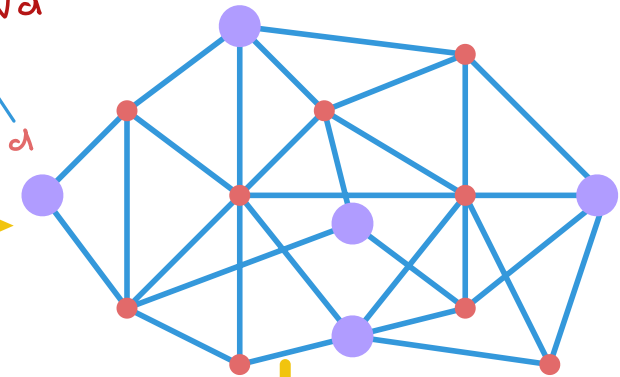
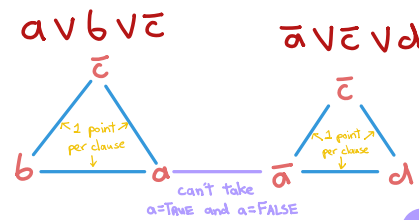
Independent set
Input: undirected $G=(V,E)$
(optionally) weights $w: V \rightarrow \mathbb{R}_{\geq 0}$



$S \subseteq V$ is independent if no
 $u,v \in S$ are connected by an edge.
Goal: max weight independent set
(decision version: is there an independent set of weight $\geq k$?)

3-SAT instance

$$\mathcal{F}(x_1, \dots, x_n) = (x_1 \vee x_2 \vee x_3) \wedge \dots$$



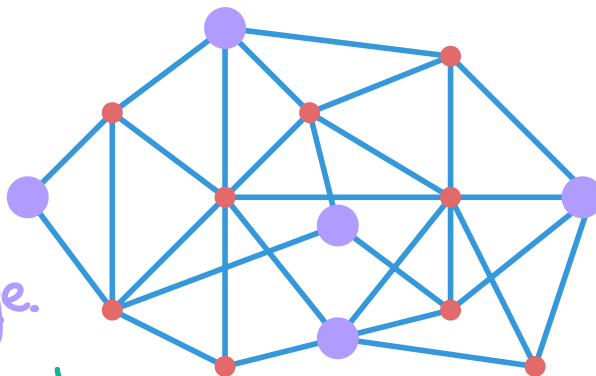
There is/isn't an
independent set
of size m

Independent set

Input: undirected $G=(V,E)$
(optionally) weights $w: V \rightarrow \mathbb{R}_{\geq 0}$

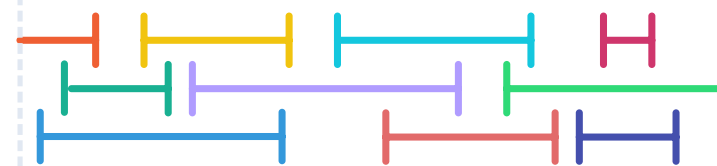
$S \subseteq V$ is independent if no
 $u,v \in S$ are connected by an edge.

Goal: max weight independent set
(decision version: is there an independent set of weight $\geq k$?)



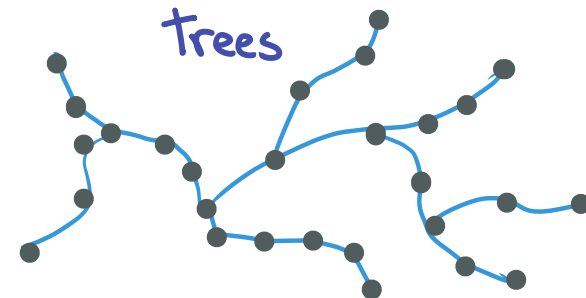
Special cases:

intervals



"independent" = non-overlapping

trees



Independent set in { graphs, intervals, trees }

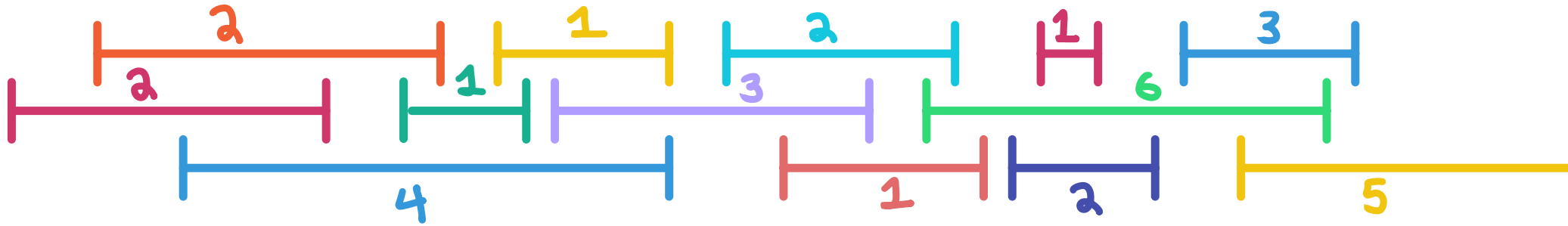
Hard as SAT Polytime algo

graphs	✓	
intervals	1%	93%
trees		

Abstain

Maximum weight independent set of intervals

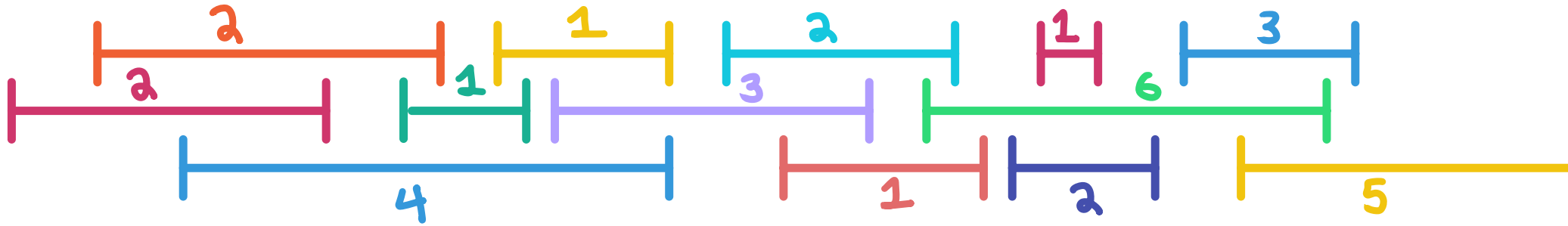
("independent" = non-overlapping)



ideas?

Maximum weight independent set of intervals

("independent" = non-overlapping)

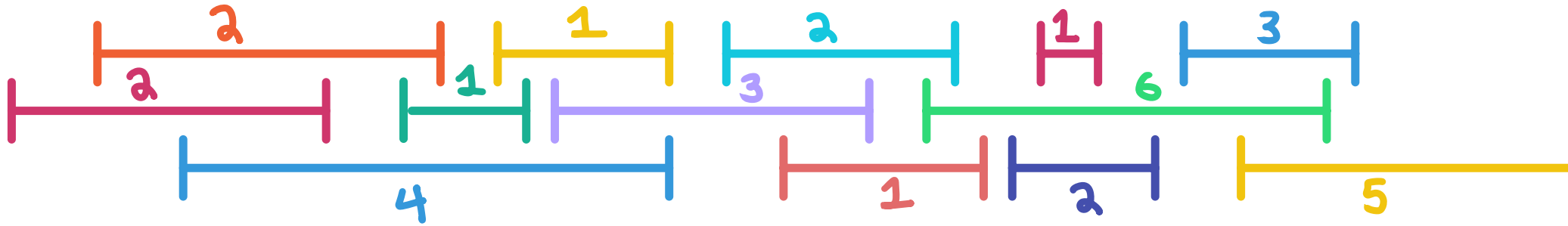


Recursive spec: For a family of weighted intervals I ,

$$MWIS(I) =$$

$I' =$ intervals disjoint from I

Maximum weight independent set of intervals
("independent" = non-overlapping)

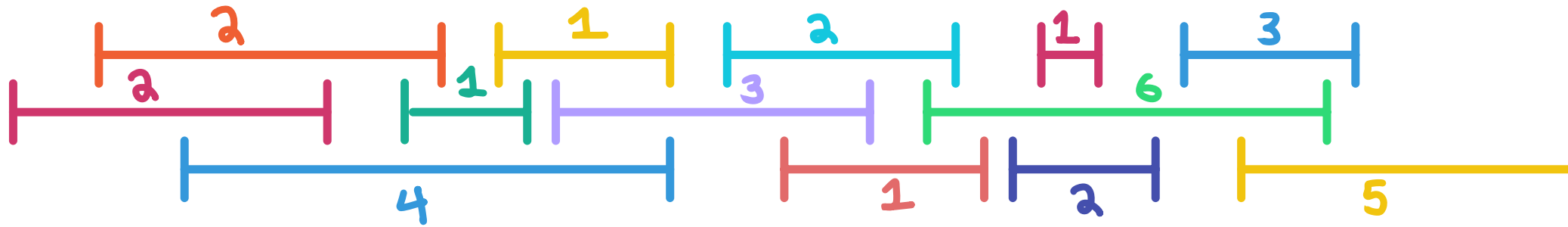


Recursive spec: for a family of weighted intervals $\mathcal{I}$,
 $MWIS(\mathcal{I}) = \max \text{ weight of any independent subset of } \mathcal{I}$

$\mathcal{I}' = \text{intervals disjoint from } \mathcal{I}$

Maximum weight independent set of intervals

("independent" = non-overlapping)



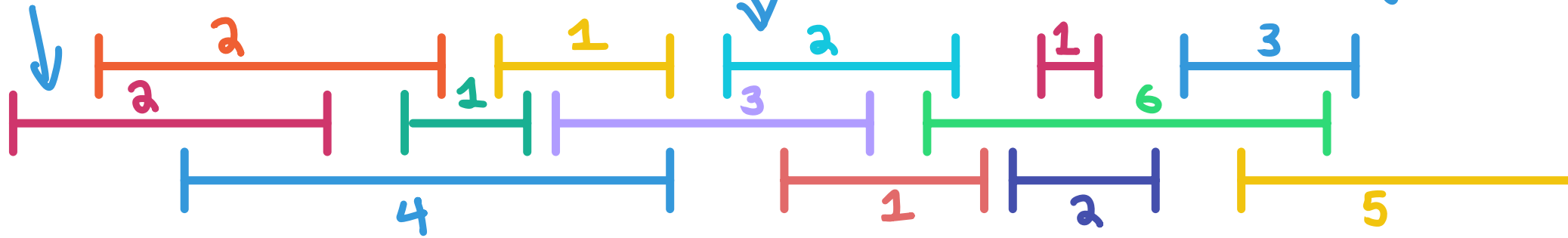
Recursive spec: for a family of weighted intervals I ,
 $MWIS(I) = \text{max weight of any independent subset of } I$

Implementation:

$MWIS(I)$

Maximum weight independent set of intervals

("independent" = non-overlapping)



Recursive spec: for a family of weighted intervals I ,
 $MWIS(I) = \text{max weight of any independent subset of } I$

Implementation:

$MWIS(I)$

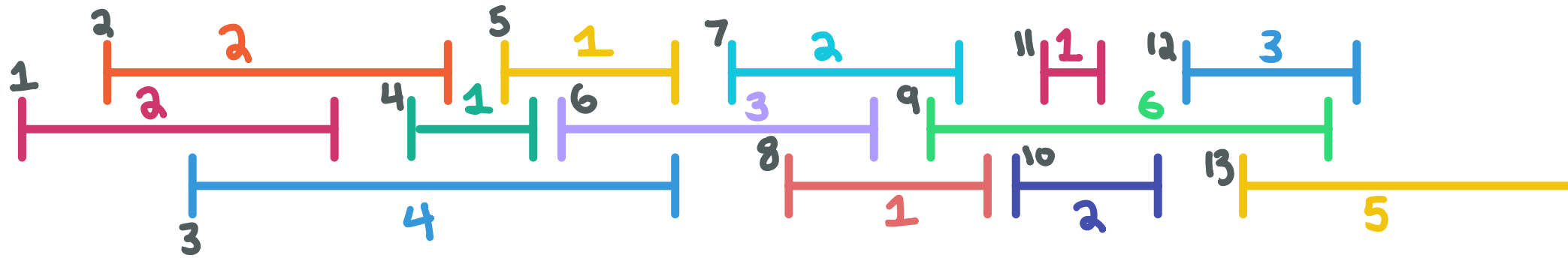
1. if $I = \emptyset$ return 0.
2. let $I \in I$ be any interval.
3. return max of
 - a. $MWIS(I \setminus \{I\})$
 - b. $\text{weight}(I) + MWIS(I')$ w/ $I' = \text{intervals disjoint from } I$

Running time

$$T(n) = 2T(n-1) + O(1) = 2^n$$

how to make this faster?

Idea: make problem "sequential" by sorting intervals

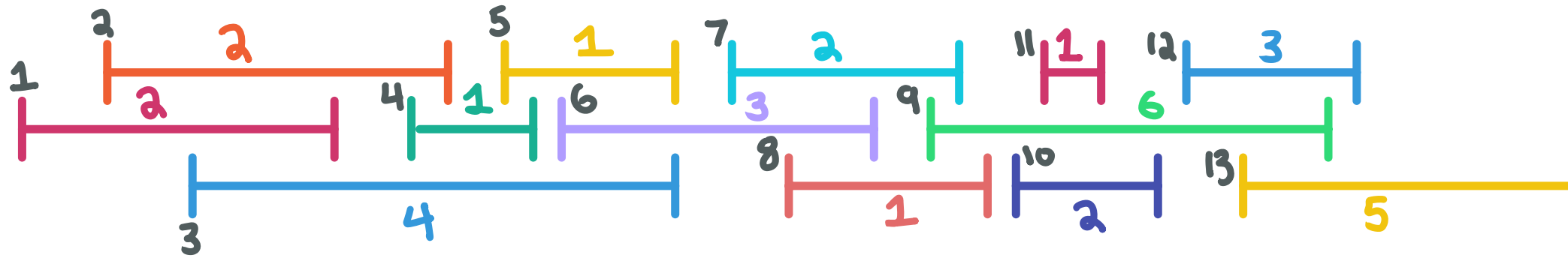


let $I_1=[a_1, b_1]$, $I_2=[a_2, b_2]$, ..., $I_n=[a_n, b_n]$ number the intervals in increasing order of left endpoint a_i

Recursive spec:

$MWIS(k) = \text{max weight of any indep. subset of } I_k, \dots, I_n$

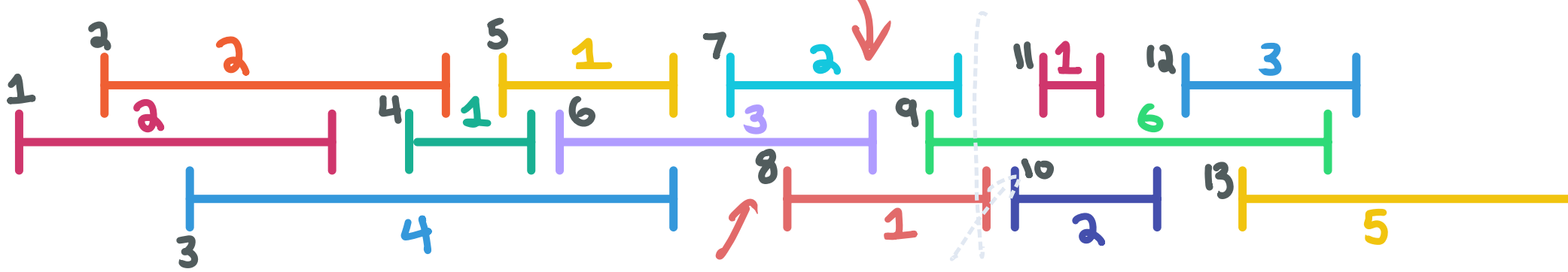
Idea: make problem "sequential" by sorting intervals



let $I_1=[a_1, b_1]$, $I_2=[a_2, b_2]$, ..., $I_n=[a_n, b_n]$ number the intervals in increasing order of left endpoint a_i

Recursive spec:

$MWIS(k) = \max \text{ weight of any indep. subset of } I_k, \dots, I_n$



let $I_1=[a_1, b_1]$, $I_2=[a_2, b_2]$, ..., $I_n=[a_n, b_n]$ number the intervals in increasing order of left endpoint a_i

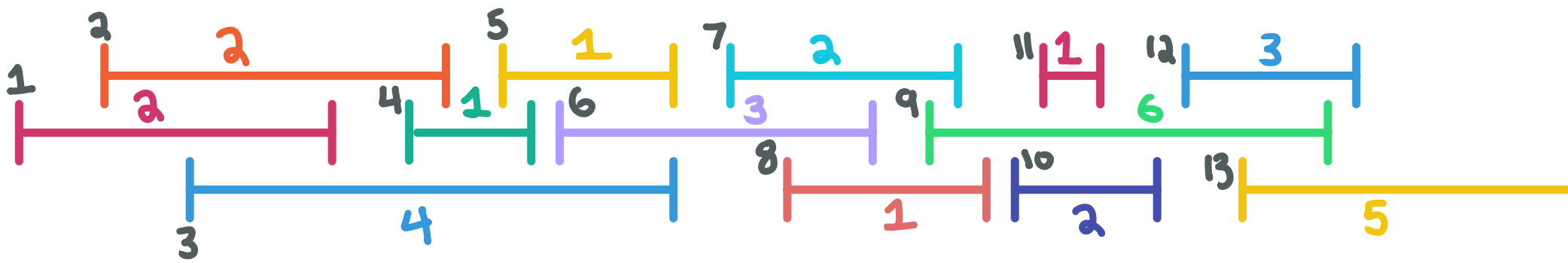
Recursive spec:

$MWIS(k) = \max \text{ weight of any indep. subset of } I_k, \dots, I_n$

Implementation:

$MWIS(k)$

if $k > n$ return 0
 if $k = n$ return $\text{weight}(I_n)$



let $I_1=[a_1, b_1]$, $I_2=[a_2, b_2]$, ..., $I_n=[a_n, b_n]$ number the intervals in increasing order of left endpoint a_i

Recursive spec:

$MWIS(k) = \max$ weight of any indep. subset of $I_k, \dots, I_n$

Implementation:

$MWIS(k)$

0 if $k > n$

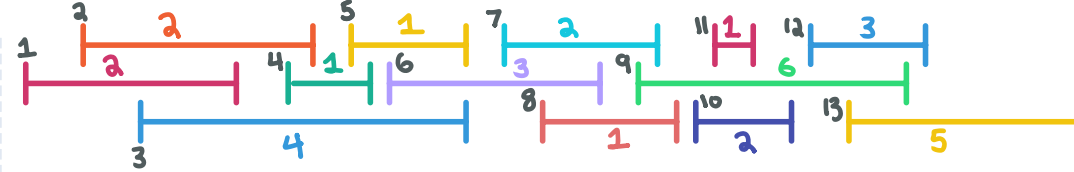
let $l > k$ be index of first interval disjoint from I_k

$\max \begin{cases} MWIS(k+1) \\ \text{weight}(I_k) + MWIS(l) \end{cases}$ otherwise

Recursive spec:

$MWIS(k) = \text{max weight}$

of any indep. subset of $I_k, \dots, I_n$



let $I_1=[a_1, b_1]$, $I_2=[a_2, b_2]$, $\dots$, $I_n=[a_n, b_n]$ number the intervals in increasing order of left endpoint a_i

Implementation:

$MWIS(k)$

0 if $k > n$

let $l > k$ be index of first interval disjoint from I_k

$\max \begin{cases} MWIS(k+1) \\ \text{weight}(I_k) + MWIS(l) \end{cases}$ otherwise

Running time (w/ caching)

$O(n)$ subproblems

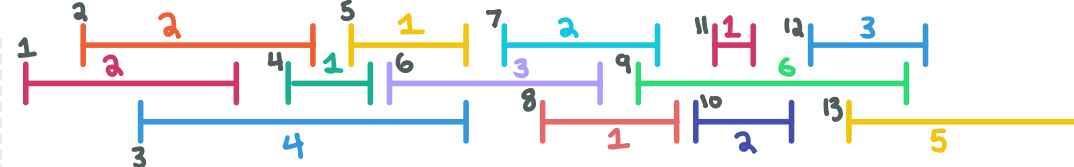
$O(n)$ time per problem

$\Rightarrow O(n^2)$

Recursive spec:

MWIS(k) = max weight

of any indep. subset of $I_k, \dots, I_n$



let $I_1=[a_1, b_1], I_2=[a_2, b_2], \dots, I_n=[a_n, b_n]$ number the intervals in increasing order of left endpoint a_i

Implementation:

MWIS(k)

0 if $k > n$

↙ binary search!

let $l > k$ be index of first interval disjoint from I_k

$\max \begin{cases} \text{MWIS}(k+1) \\ \text{weight}(I_k) + \text{MWIS}(l) \end{cases}$ otherwise

Running time (w/ caching)

$\Rightarrow O(n \log n)$

$O(n \log n)$ to sort

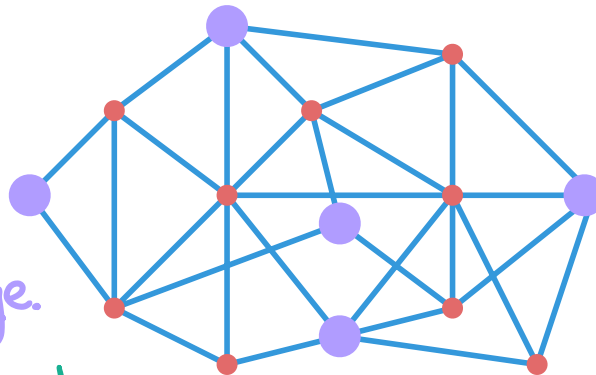
$O(n)$ subproblems $\times O(\log n)$ time per problem

Independent set

Input: undirected $G=(V,E)$
(optionally) weights $w: V \rightarrow \mathbb{R}_{\geq 0}$

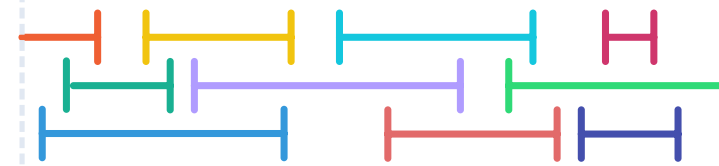
$S \subseteq V$ is independent if no
 $u,v \in S$ are connected by an edge.

Goal: max weight independent set
(decision version: is there an independent set of weight $\geq k$?)



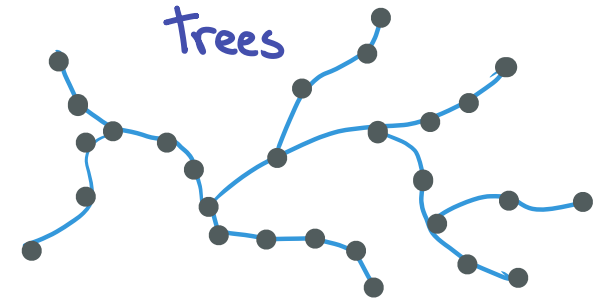
Special cases:

intervals



"independent" = non-overlapping

trees



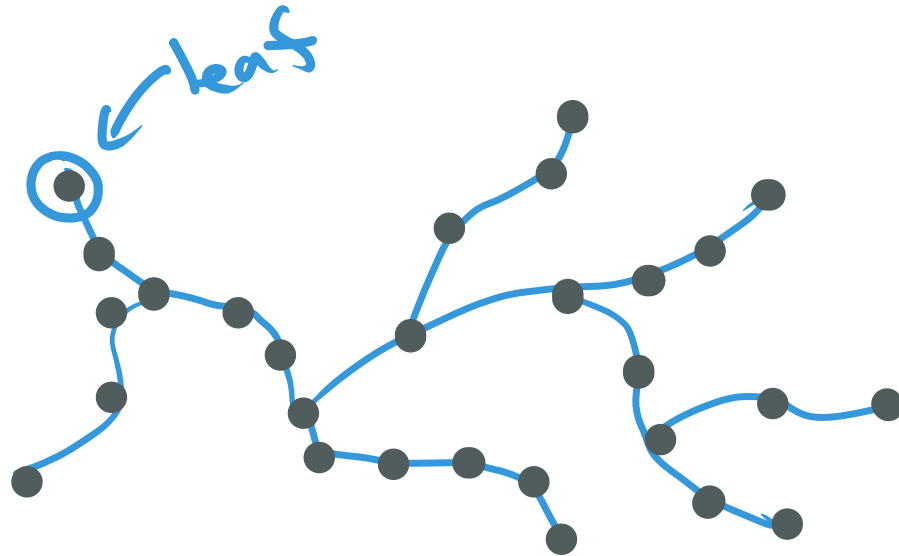
Independent set in { graphs, intervals, trees }

Hard as SAT Polytime algo

✓	
	✓

Abstain

MWIS in trees

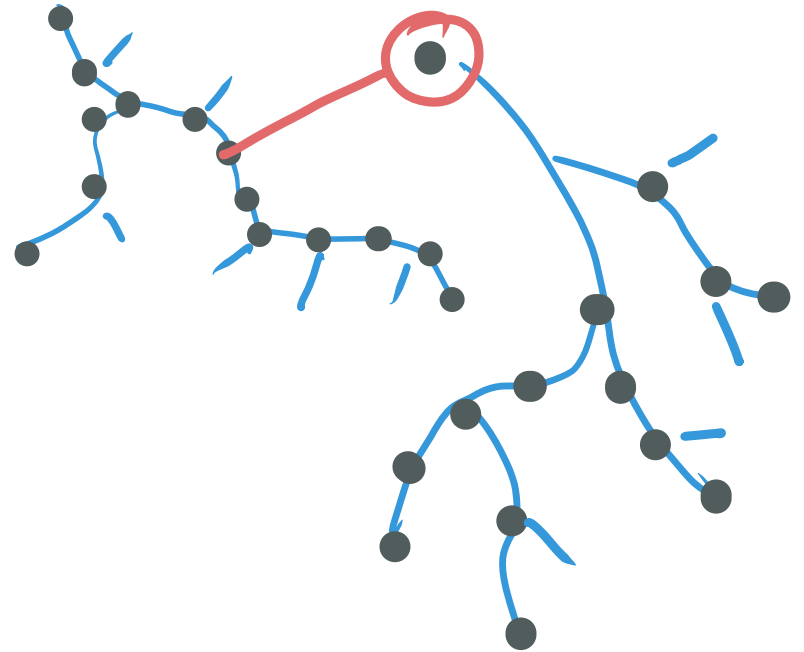
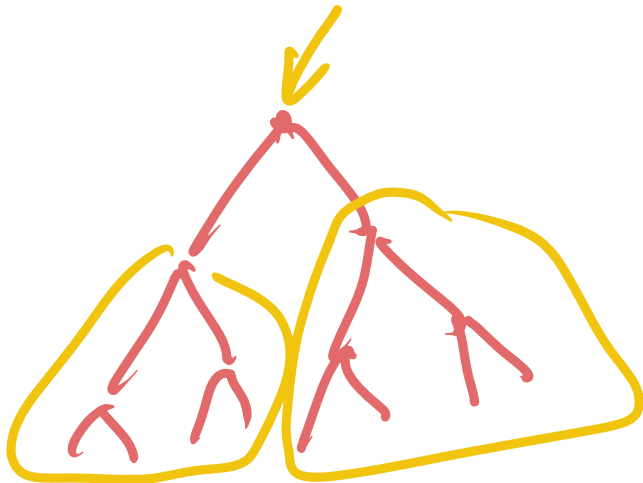


ideas?

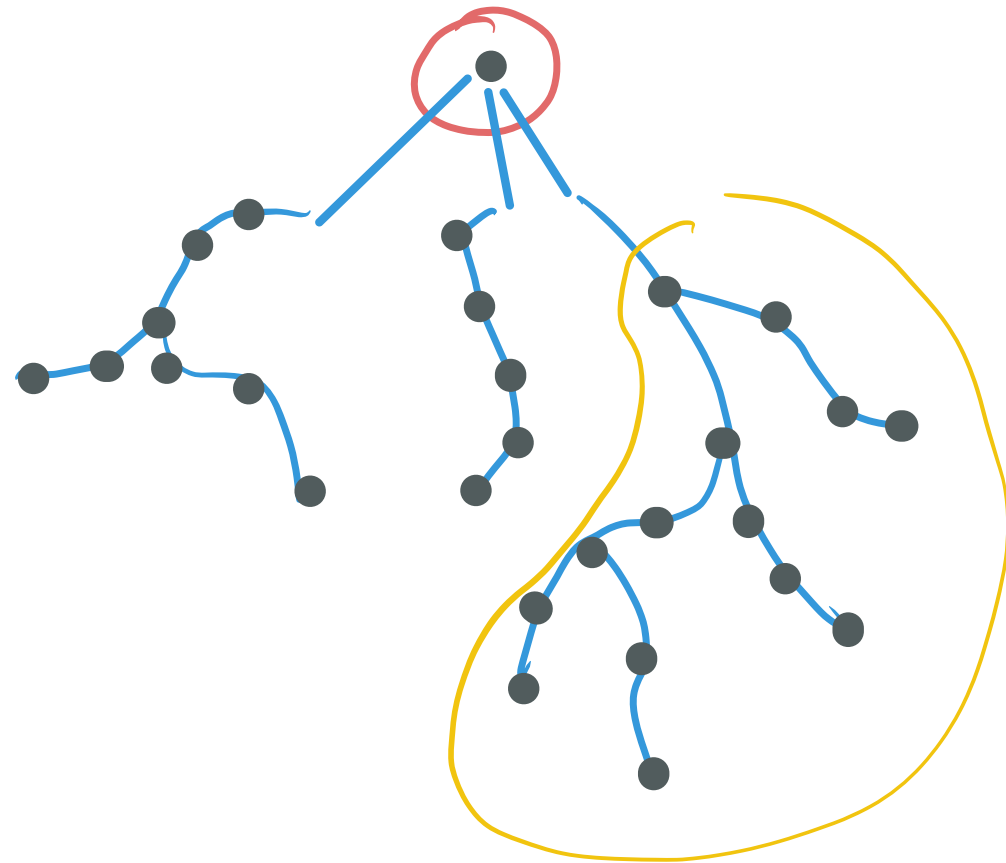
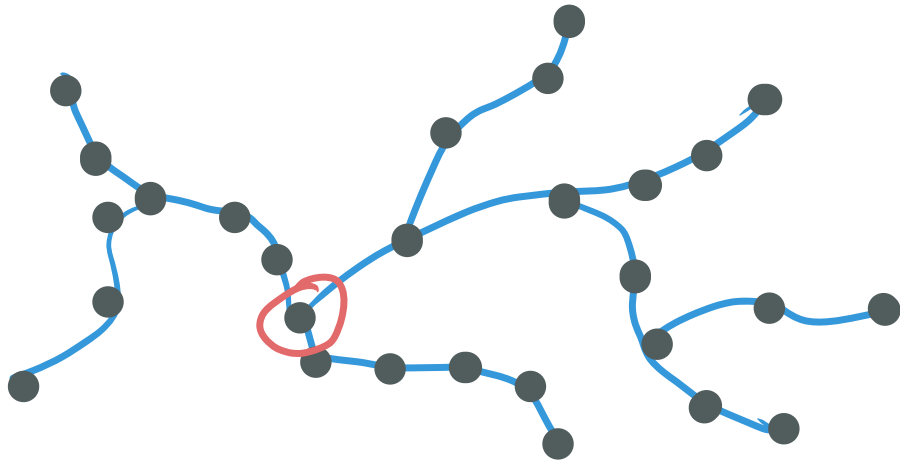
MWIS in trees

ideas?

how to order subproblems?



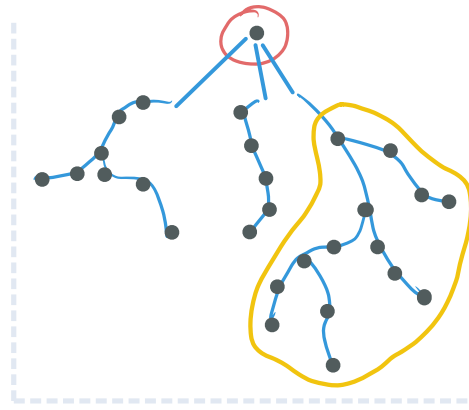
Root it!



Subtree

Recursive Spec:

$MWIS(v)$ = MWIS of the subtree rooted at v



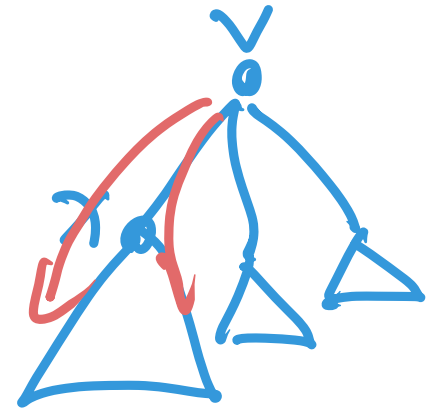
Recursive implementation

$MWIS(v) =$

max

$\sum_{\text{child } x} MWIS(x)$

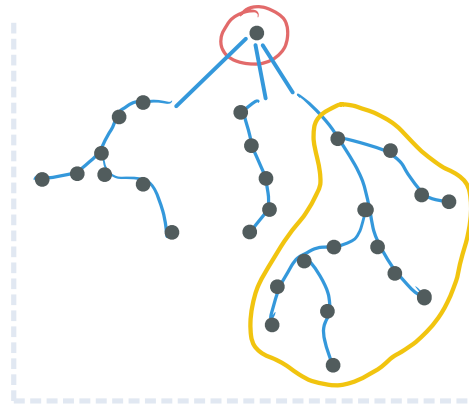
$\left\{ \begin{array}{l} \text{weight}(v) + \sum_{\text{grandchild } x} MWIS(x) \end{array} \right.$



Recursive Spec:

$MWIS(v, \text{...}) =$

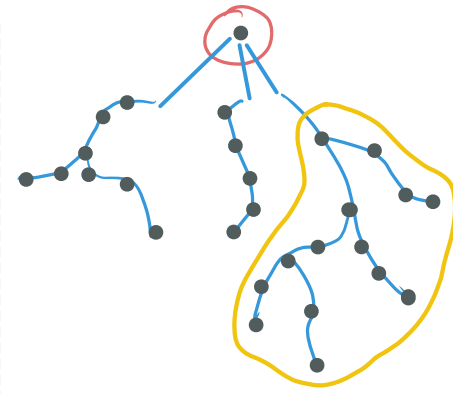
MWIS of the subtree rooted at v ,



Recursive Spec:

$MWIS(v, \text{exclude-root}^{(\text{flag})}) =$

MWIS of the subtree rooted at v , excluding v if exclude-root is true.



Recursive implementation

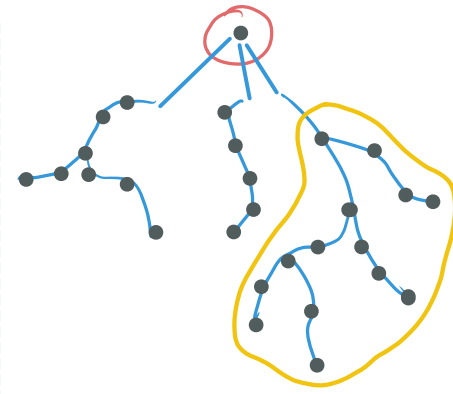
$MWIS(v, \text{exclude-root}) =$

Running time

Recursive Spec:

$MWIS(v, \text{exclude-root}^{(\text{flag})}) =$

MWIS of the subtree rooted at v , excluding v if exclude-root is true.



Recursive implementation

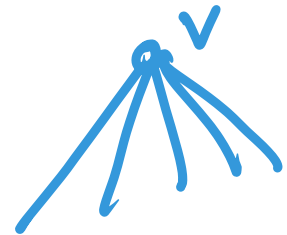
$MWIS(v, \text{exclude-root}) =$

if exclude-root then return sum of $MWIS(x, \text{false})$ over all children x of v

else return max of $MWIS(v, \text{true})$

and $\text{weight}(v) + \sum_{\text{children } x} MWIS(x, \text{true})$

Running time



$$O(\sum \# \text{child } v)$$

$$= O(m) = O(n)$$