L25

reductions

4102

11.21.2013

Nathan

a new technique for algorithm design

MergeSort(n)

```
<base case>
 MergeSort(n/2)    <left half>
 MergeSort(n/2)    <right half>
 Merge(left,right)  <combine>
```

MergeSort(n)

<base case>

MergeSort(n/2) <left half>

MergeSort(n/2) <right half>

Merge(left,right) <combine>

$$T(n) = 2T(n/2) + O(n)$$

MergeSort(n) $<$f(n) 2MergeSort(n/2)

Typesetting

$$\mathrm{BEST}_n = \min \left\{ \begin{array}{c} \mathrm{BEST}_0 + S_{1,n}^2 \\ \mathrm{BEST}_1 + S_{2,n}^2 \\ \mathrm{BEST}_2 + S_{3,n}^2 \\ \dots \\ \mathrm{BEST}_{\ell-1} + S_{\ell,n}^2 \\ \\ \dots \\ \\ \mathrm{BEST}_{n-1} + S_{n,n}^2 \end{array} \right.$$

Typesetting

$$\mathrm{BEST}_n = \min \left\{ \begin{array}{c} \mathrm{BEST}_0 + S_{1,n}^2 \\ \mathrm{BEST}_1 + S_{2,n}^2 \\ \mathrm{BEST}_2 + S_{3,n}^2 \\ \cdots \\ \mathrm{BEST}_{\ell-1} + S_{\ell,n}^2 \\ \\ \cdots \\ \\ \mathrm{BEST}_{n-1} + S_{n,n}^2 \end{array} \right.$$

solving BESTn can be reduced to solving n-1 BESTi problems and combining the answer in linear time.

HUFFMAN

Finding an optimal code for an X character alphabet

solved by

can be reduced to

Finding an optimal code for an X-1 character alphabet

WE HAVE BEEN SOLVING PROBLEM A BY SOLVING SMALLER VERSIONS OF PROBLEM A

GENERAL IDEA:

SOLVE PROBLEM A BY SOLVING PROBLEM B

REDUCTION

$$\text{PROBLEM}_a \leq_{f(n)} \text{PROBLEM}_b$$

problem a reduces to problem b. “A solution to problem B implies there is a solution to A”
The time it takes to solve A(n) is <= f(n) + time it takes to solve B(n)

REDUCTION

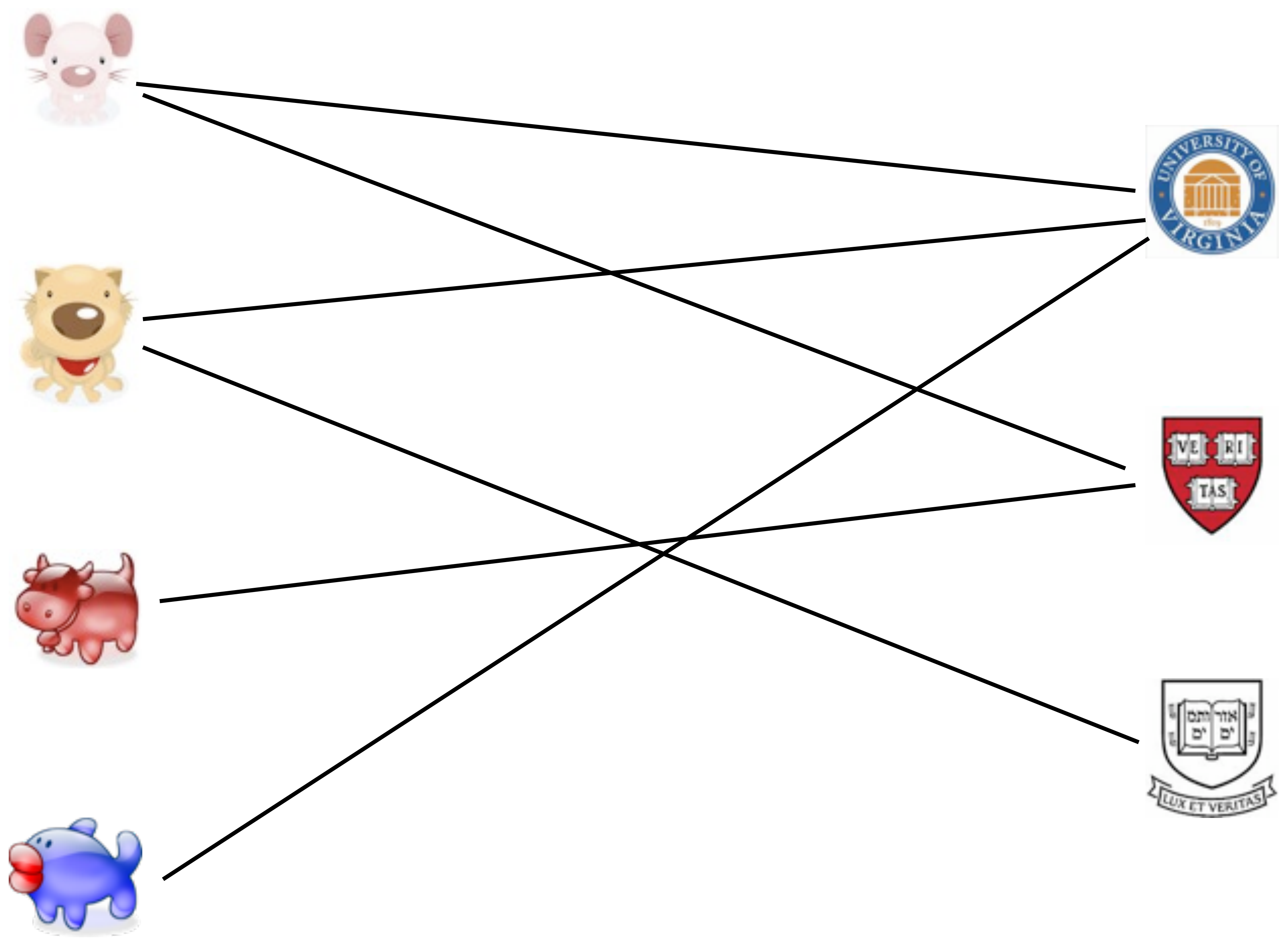
$$\text{PROBLEM}_a \leq_{f(n)} \text{PROBLEM}_b$$

$\exists c, d$

$$T(\textsc{problem}_a(n)) \leq f(n) + cT(\textsc{problem}_b(dn))$$

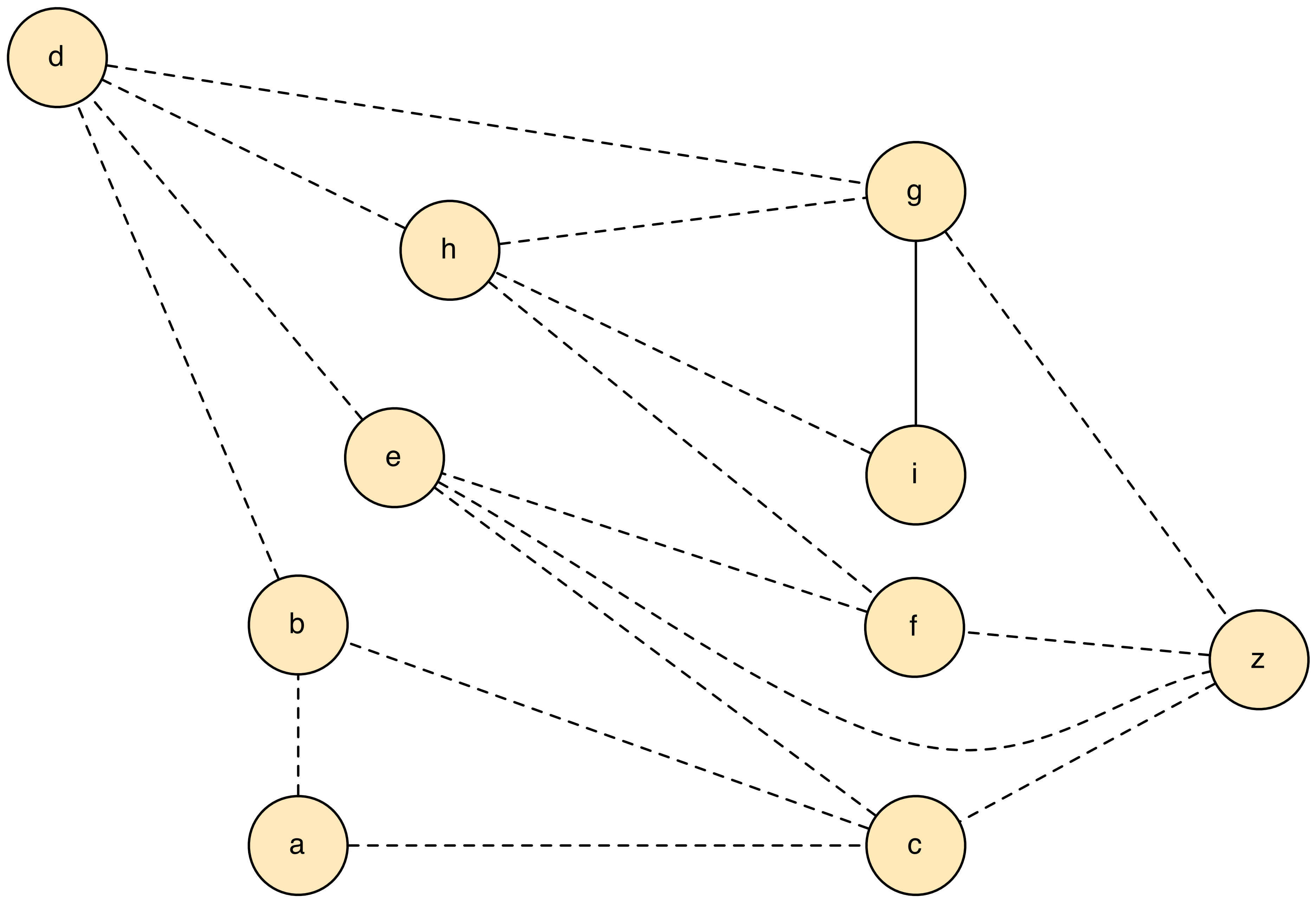
notice this is a very natural generalization of divide and conquer...in which case A and B are THE SAME problem

MAXIMUM BIPARTITE MATCHING



Can solve maxbipartite by solving max-flow, min-cut.

EDGE-DISJOINT PATHS



$\textsc{maxBipartite} <_{E+V} \textsc{maxFlow}$

$\textsc{maxEdgeDisj} <_{E+V} \textsc{maxFlow}$

TRIPLET PROBLEM

given numbers $(x_1, \ldots, x_n)$

determine whether there is a triplet (x_i, x_j, x_k)

such that $x_i + x_j + x_k = 0$

```
3,-6, 5 ,2,6,8,-1,12,7,-10,-3,14
```

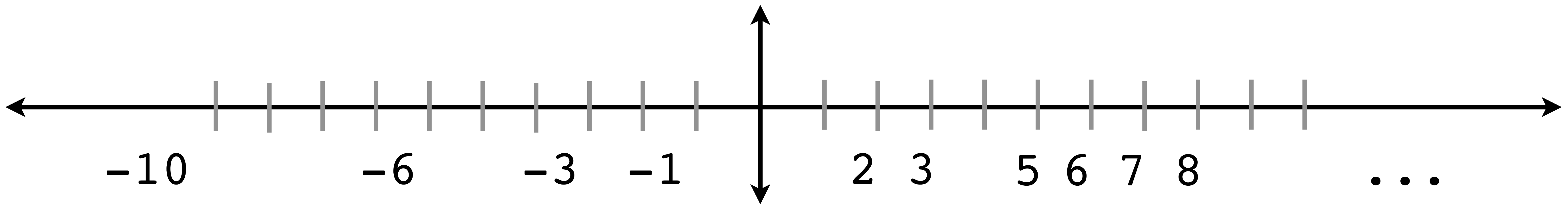
EASY TO SOLVE IN

$$O(n^3)$$

now ask the class to spend 5-7 minutes thinking about a way to solve this problem in n^2 time.
after two minutes: hint: the answer must involve either one negative and 2 positives, or one pos and two negatives.
for each negative, how much time does it take to determine if this negative is part of a solution?
can you do it in O(n)?

EASY TO SOLVE IN $O(n^2)$

3,-6, 5 ,2,6,8,-1,12,7,-10,-3,14



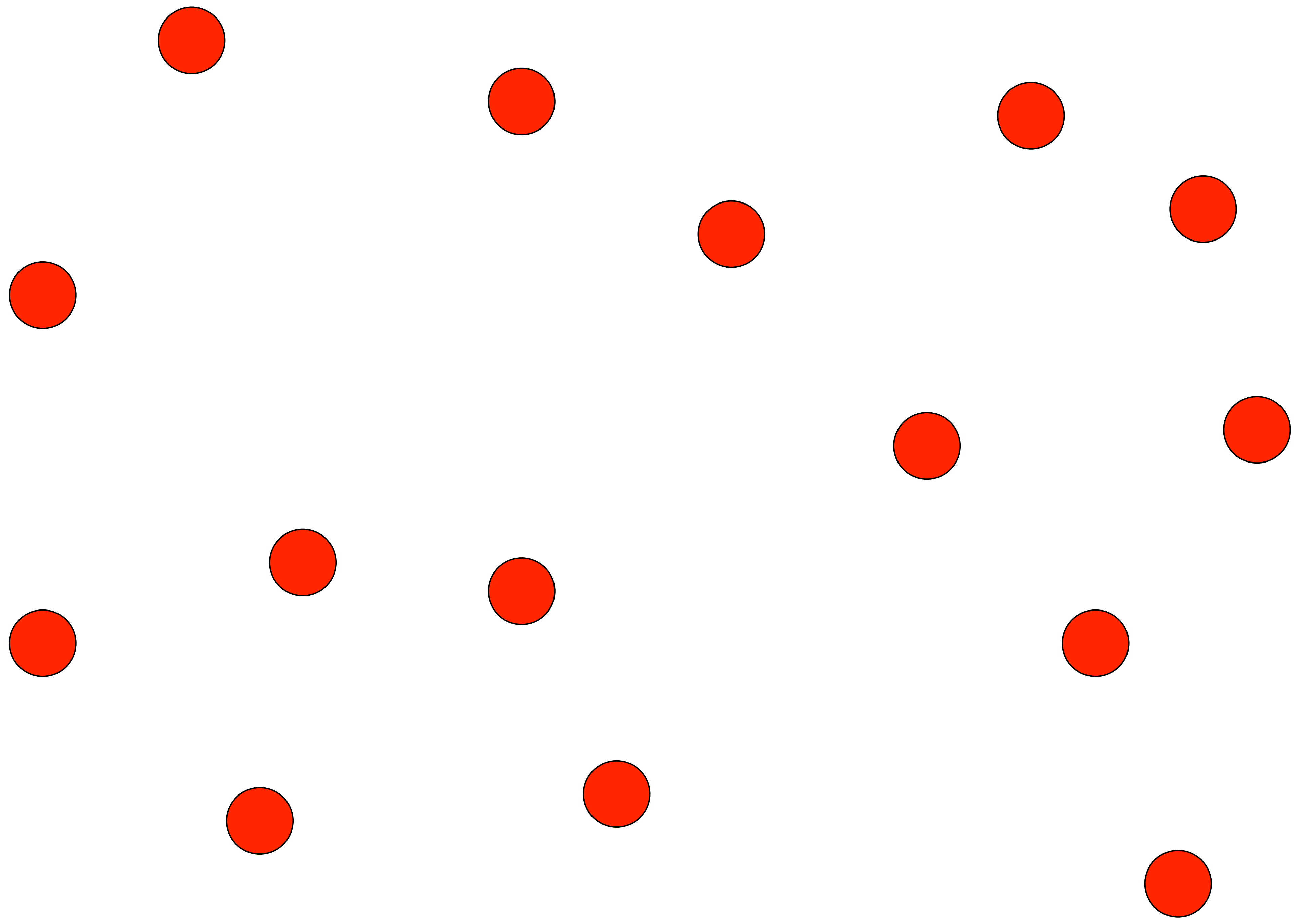
Every triplet needs two neg and one pos or vice versa. Split the input into negs and pos. Sort these. now lets test if the solution consists of two negs and a pos. In particular, lets test if the solution consists of first neg, another neg, and a pos. keep a left pointer at the second neg, and a right pointer at the first pos. in O(n) steps, we can see if there is a solution involving this first negative as follows: if n1+n2 + p1 <0, then move right pointer (selecting a larger pos). else, move left pointer selecting a smaller neg n2. if equality, you are done. at most 2n steps to get to the end of both lists. since there are n possible inputs, O(n^2) time.

COLINEARITY

given points in the plane $((x_1, y_1), \ldots, (x_n, y_n))$

determine whether any 3 are co-linear but not horizontal.

finding lines in a set of points...



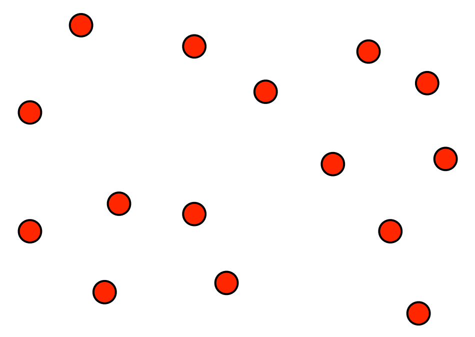
example of the problem... very interesting practical problem...what does it have to do with 3sum?
we are going to show an interesting connection

HOW CAN WE COMPARE 2 PROBLEMS?

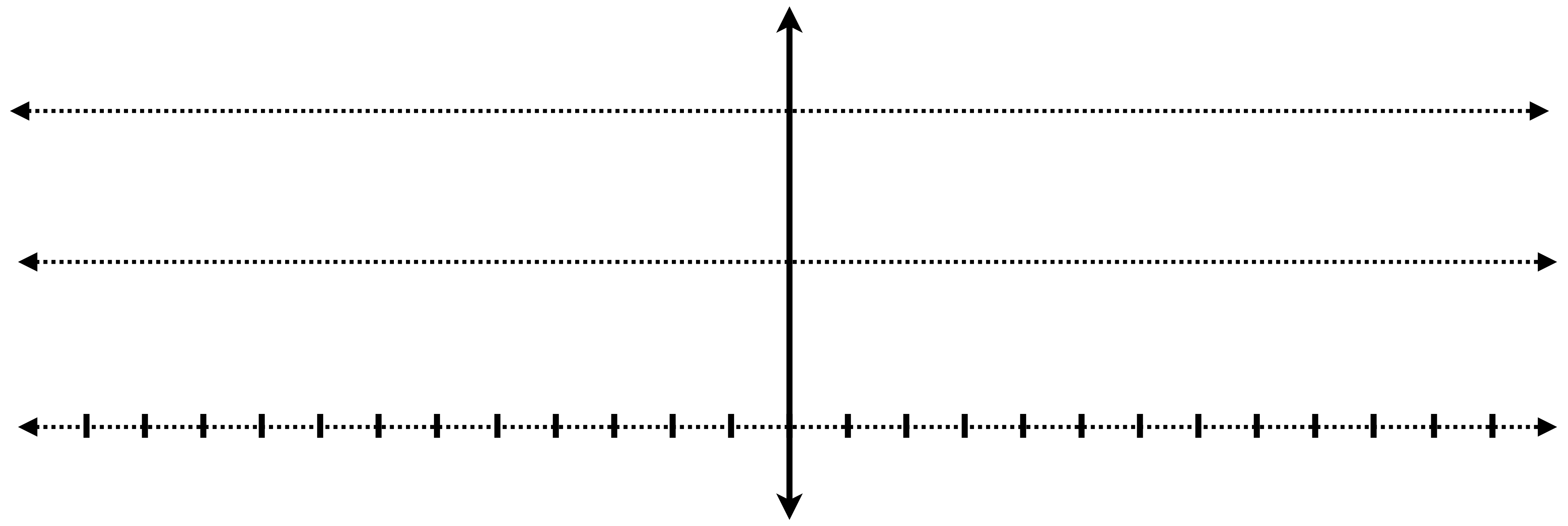
$$\text{PROBLEM}_a \leq_{f(n)} \text{PROBLEM}_b$$

$$T(\text{PROBLEM}_a(n)) \leq f(n) + cT(\text{PROBLEM}_b(dn))$$

```
3,-6,5,2,6,8,-1,12,7,-10,-3,14
```

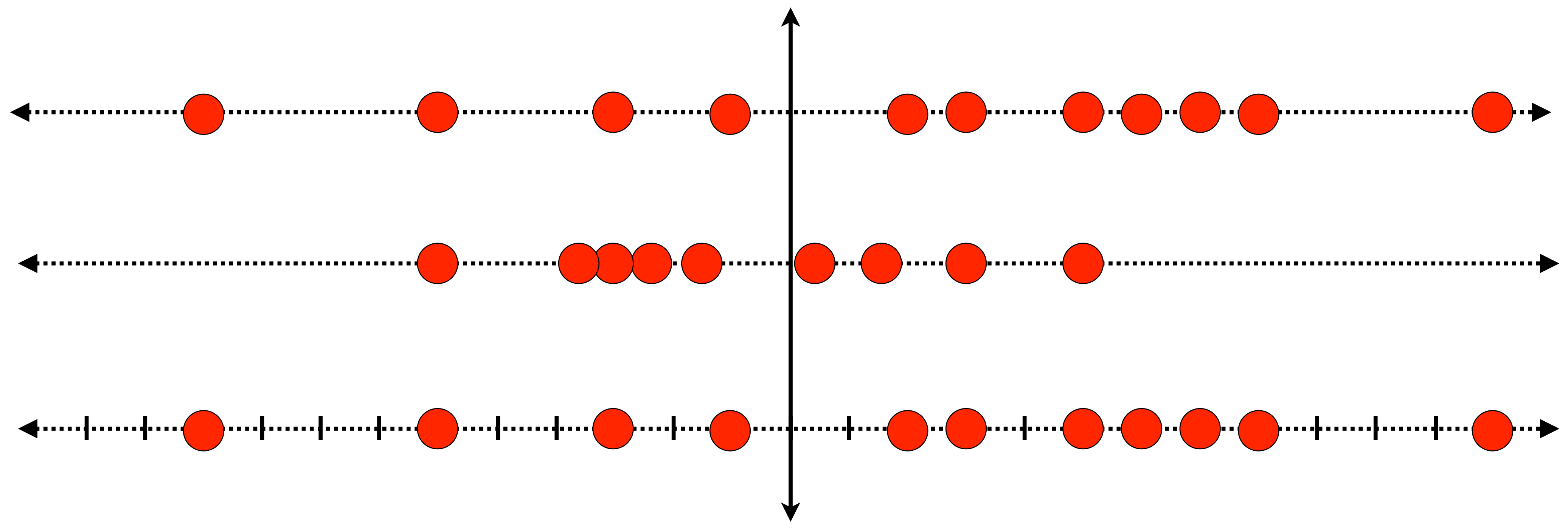


$T = \{ 3,-6,5,2,6,8,-1,12,7,-10,-3,14 \}$



take every number x in the set T and add the points (x,0), (–x/2,1), (x,2).
If T has a triplet (a,b,c), st a+b+c=0, then (a,0) (–b/2,1), (c,2) are on a line.
(–b/2–a) is the “run” in the first step
(c–(–b/2)) is the “run” in the second step. = (c+b/2) = (–b/2–a) because (c+b/2)+(b/2)+a = 0.
Similarly, if there is a line (x,0)(y,1)(z,2), then one can map this to a triple, x,2y,z in the set that adds to zero.

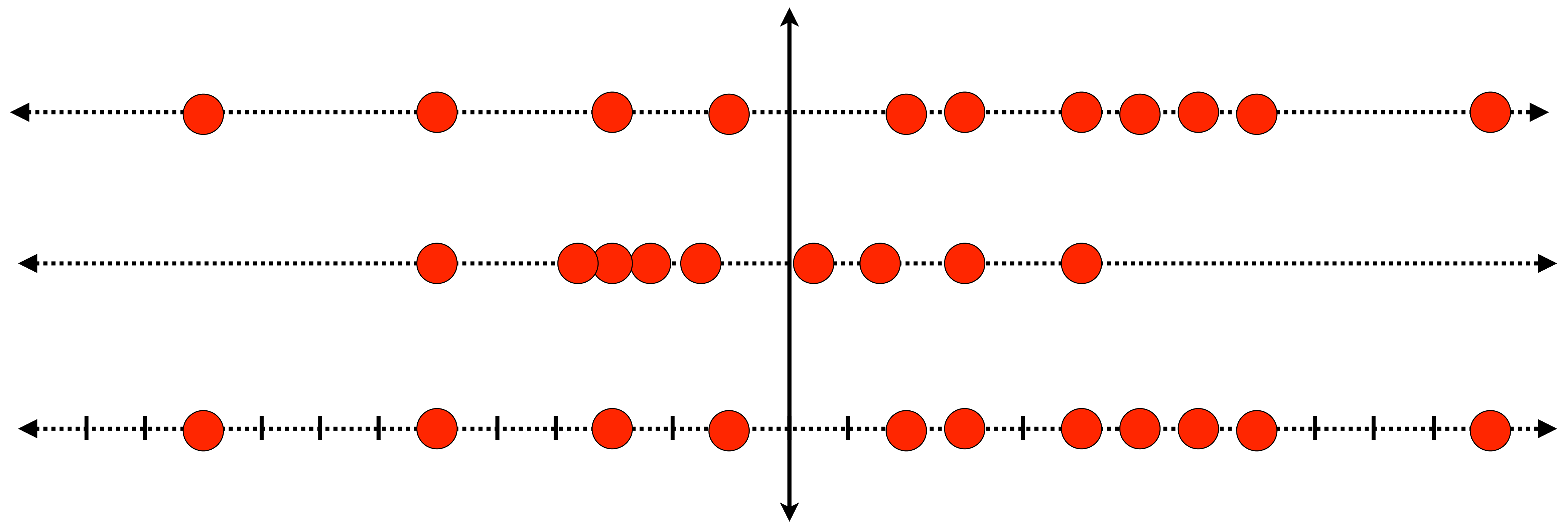
$T = \{$ `3,-6,5,2,6,8,-1,12,7,-10,-3,14` $\}$



P=

T= { 3,-6,5,2,6,8,-1,12,7,-10,-3,14 }

P=

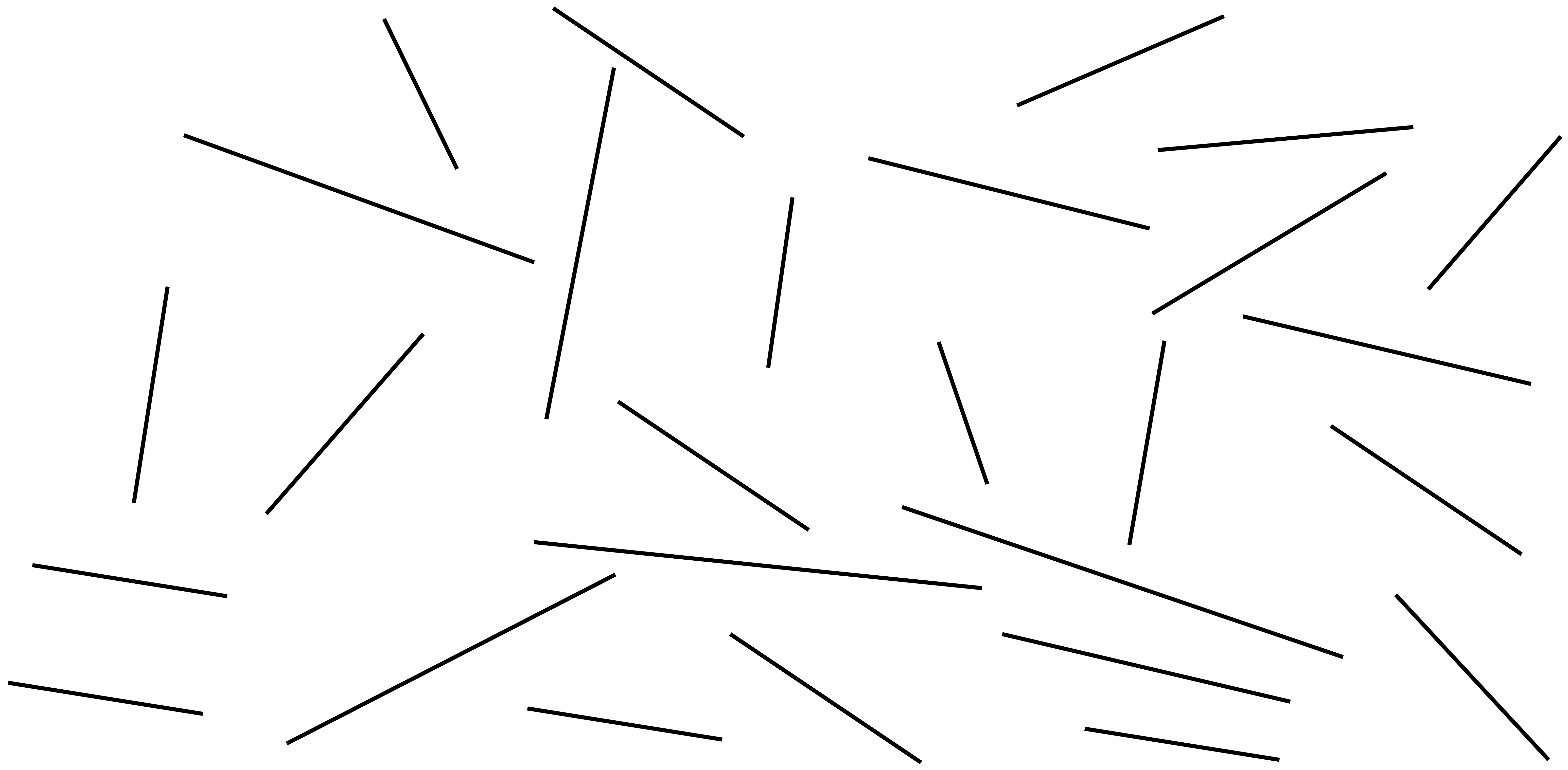


T is a TRIPLET-set if and only if P is a COLINEAR set.

SEGMENT PARTITION

given line segments in the plane, determine if there is a line that cleanly partitions the segments into two sets.

SEGMENT PARTITION

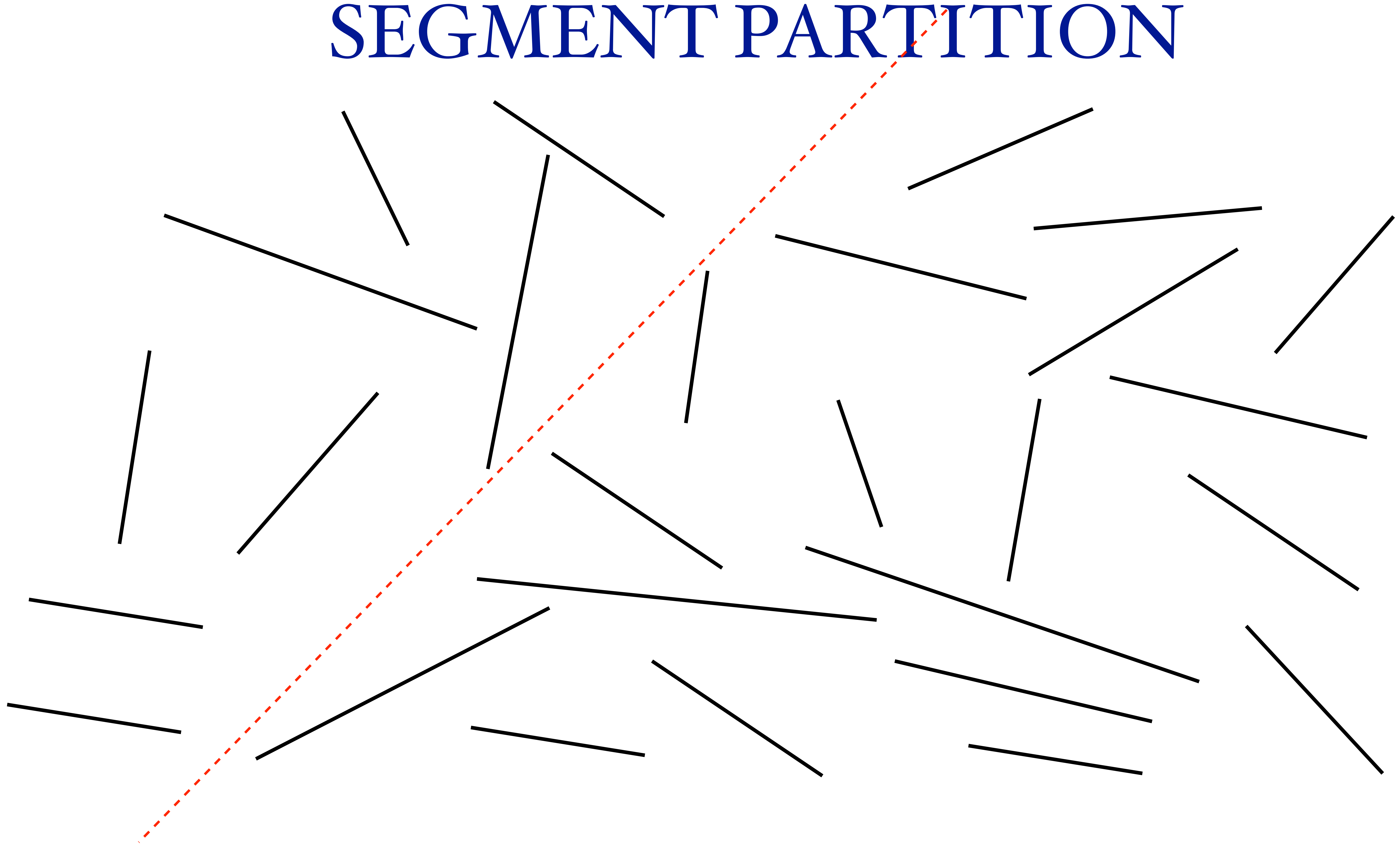


SEGMENT PARTITION

Problem: Given a set of line segments in the plane, determine if there exists a line that partitions the segments into two sets.

this problem is related to motion planning..or avoiding asteroids in star wars, etc... the partition is a way to cleanly avoid the obstacles, etc...

SEGMENT PARTITION



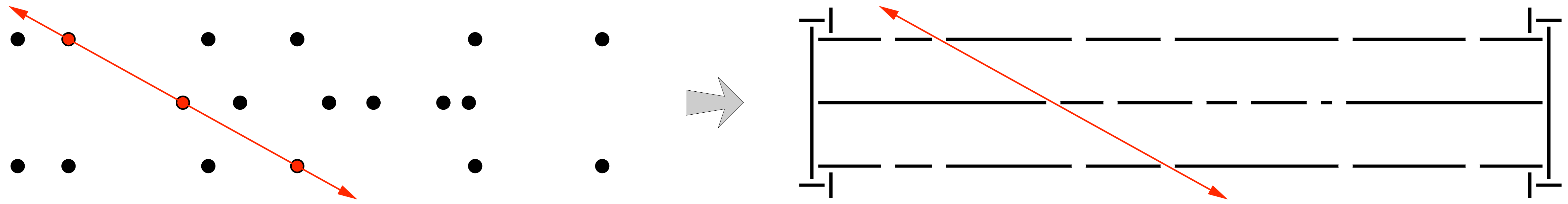
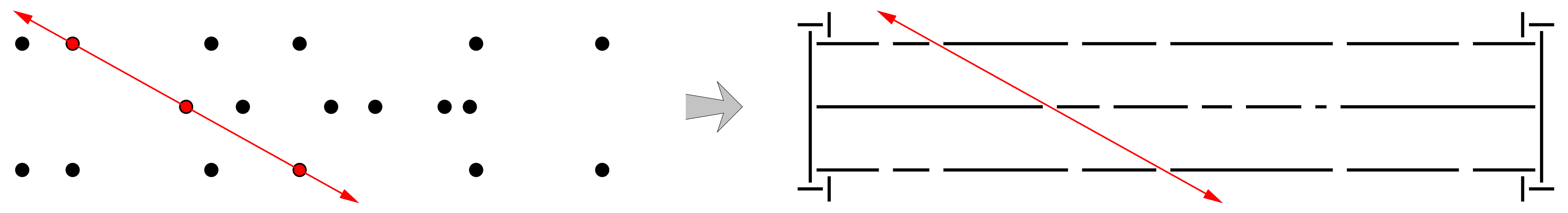


image: erickson

3SUM <= SEGMENTS.
solve an instance of colin
so this problem is as hard as the 3-sum problem

Consider the following *segment splitting problem*: Given a collection of line segments in the plane, is there a line that does not hit any segment and splits the segments into two non-empty subsets?

To show that this problem is 3SUM-hard, we start with the collection of points produced by our last reduction. Replace each point by a 'hole' between two horizontal line segments. To make sure that the only way to split the segments is by passing through three colinear holes, we build two 'gadgets', each consisting of five segments, to cap off the left and right ends as shown in the figure below.



Top: $3n$ points, three on a non-horizontal line.
Bottom: $3n + 13$ segments separated by a line through three colinear holes.

This reduction could be performed in linear time if we could make the holes infinitely small, but computers can't really deal with infinitesimal numbers. On the other hand, if we make the holes too big, we might be able to thread a line through three holes that don't quite line up. I won't go into details, but it is possible to compute a working hole size in $O(n \log n)$ time by first computing the distance between the closest pair of points.

Thus, we have a valid reduction from 3SUM to segment splitting (by way of colinearity):

$$\begin{tikzcd}
\boxed{\text{set of } n \text{ numbers}} \arrow[r, "O(n)"] & \boxed{\text{set of } 3n \text{ points}} \arrow[r, dashed, "O(n\log n)"] & \boxed{\text{set of } 3n+13 \text{ segments}} \arrow[d, Rightarrow, "\textsc{Splittable?}"] \\
\boxed{\textsc{True} \text{ or } \textsc{False}} & \boxed{\textsc{True} \text{ or } \textsc{False}} \arrow[l, "\text{trivial}"'] & \boxed{\textsc{True} \text{ or } \textsc{False}} \arrow[l, dashed, "\text{trivial}"']
\end{tikzcd}$$

$$T_{3\text{SUM}}(n) \le T_{\text{split}}(3n+13) + O(n \log n) \quad \Longrightarrow \quad T_{\text{split}}(n) \ge T_{3\text{SUM}}\left(\frac{n-13}{3}\right) - O(n \log n).$$

hidden slide...for your own understanding of hte problem

WHY DO WE CARE?

because now we can study the 3sum problem---a simply stated one, and see if we can find a lower bound on its complexity.
if we can, then we know a lot about other problems...in particular, motion planning for a robot may also require n^2 time...
currently, we only know very weak lower-bounds on 3sum. NLOGN in a standard model,...n^2 in a specific, limited model of computation....we also only know an n^2 algorithm. so progress in either direction is important: either faster algorithm or better lower bound.

Anka Gajentaan and Mark Overmars[5] defined a whole class of computational geometry problems that are harder than 3SUM; they called these problems 3SUM-*hard*. A sub-quadratic algorithm for any 3SUM-hard problem would imply a subquadratic algorithm for 3SUM. I'll finish the lecture with two more examples of 3SUM-hard problems.

[4]The $\Omega(n^2)$ lower bound holds in a decision tree model where every query asks for the sign of a linear combination of three of the input numbers. For example, 'Is $5x_1 + x_{42} - 17x_5$ positive, negative, or zero?' See my paper 'Lower bounds for linear satisfiability problems' (http://www.uiuc.edu/~jeffe/pubs/linsat.html) for the gory(!) details.

[5]A. Gajentaan and M. Overmars, On a class of $O(n^2)$ problems in computational geometry, *Comput. Geom. Theory Appl.* 5:165–185, 1995. ftp://ftp.cs.ruu.nl/pub/RUU/CS/techreps/CS-1993/1993-15.ps.gz

3sum hard

ANOTHER EXAMPLE

3SAT PROBLEM

input:

output: “

input: boolean formula phi in cnf 3 vars per clause
output: YES OR NO. Is there a way to assign t/f to variables that makes every clause satisfiable??

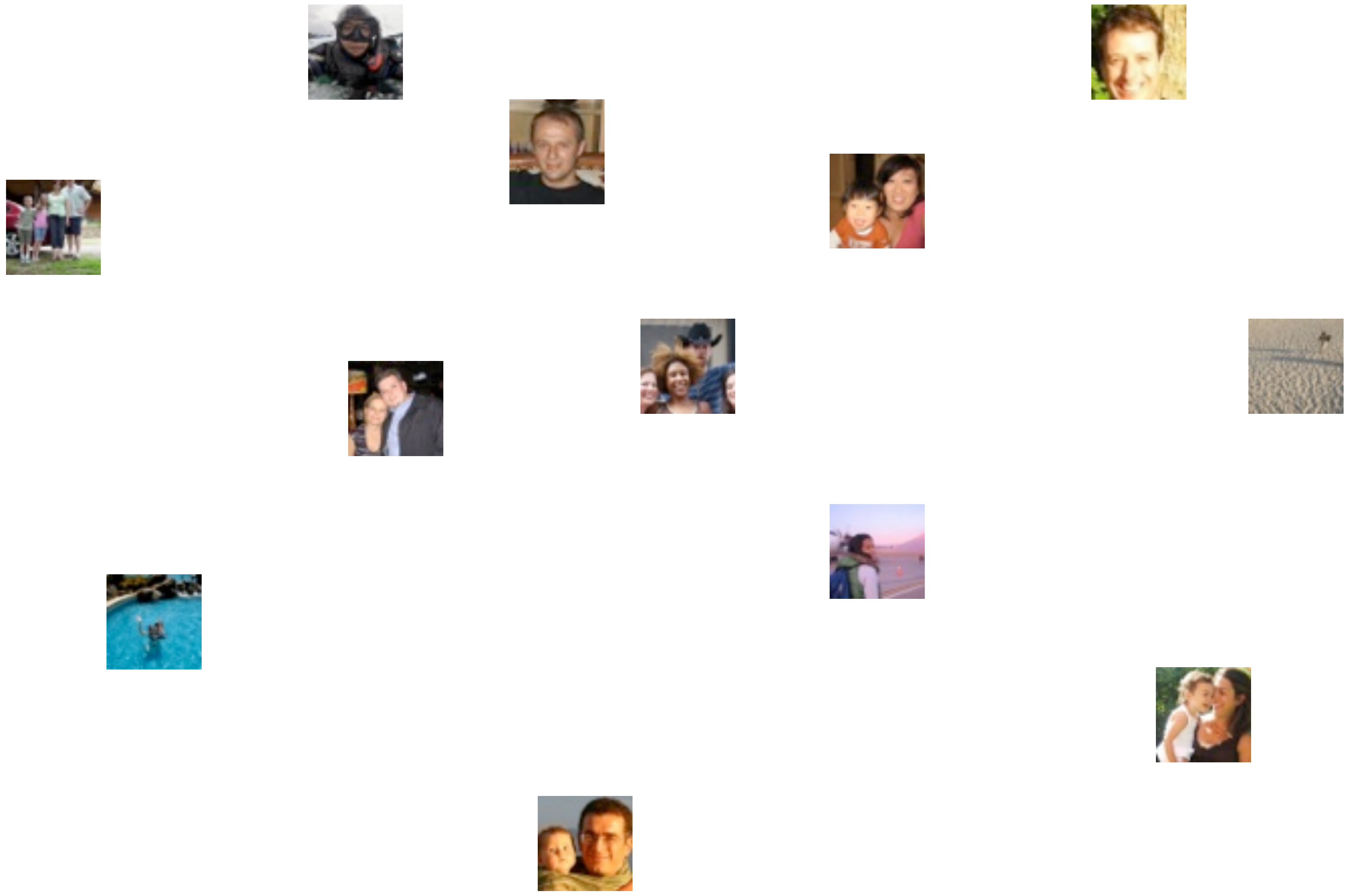
3SAT EXAMPLE

$$(x \vee y \vee z) \wedge (x \vee \overline{y} \vee y) \wedge (u \vee y \vee \overline{z}) \wedge (z \vee \overline{x} \vee u) \wedge (\overline{x} \vee \overline{y} \vee \overline{z})$$

x,y,u true, z false.

this seems like a total nerd problem. who cares? but it turns out that this simple combinatorial logic problem is related to thousands of important practical problems...we shall see it is the hardest problem in the class “NP”

PARTY PROBLEM



for example...3sat is related to the following problem. You are having a party with all of your friends...
but not all of your friends are friends of each other.

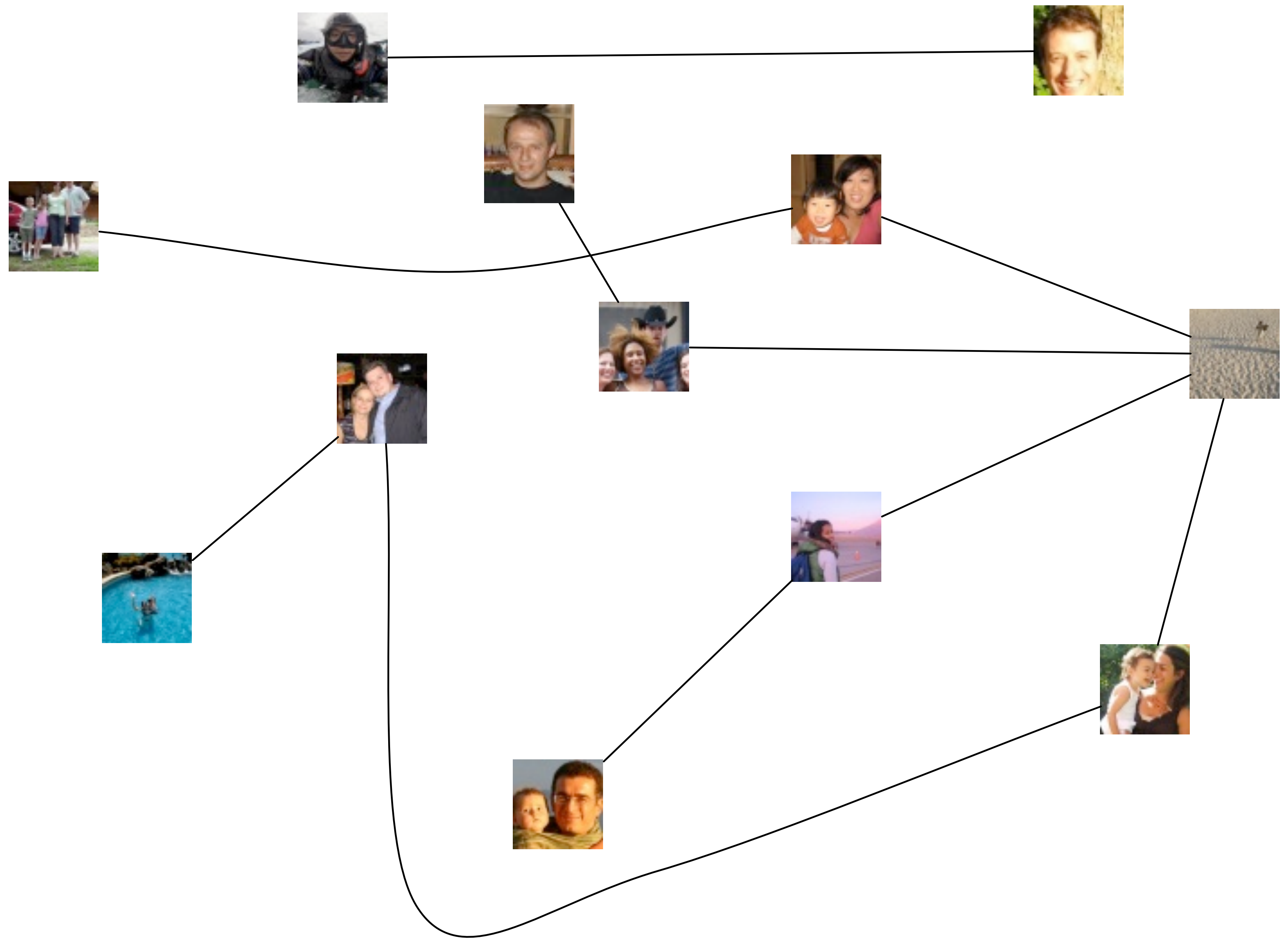
INDEPENDENT SET

write out def of the problem

INDEPENDENT SET

a set $S \subseteq V$ is an independent set if no two nodes in S are joined by an edge.

EXAMPLE



find an ind set on this graph...

GOAL: given a graph G,

find the largest independent set

actually...we are going to modify the goal slightly...and make hte problem (G,k): YES if G has an independent set of size k.
we just want to know the SIZE of the largest party we can have.

3SAT $\leq_p$ INDSET

$$(x \vee y \vee z) \wedge (x \vee \overline{y} \vee y) \wedge (u \vee y \vee \overline{z}) \wedge (z \vee \overline{x} \vee u) \wedge (\overline{x} \vee \overline{y} \vee \overline{z})$$

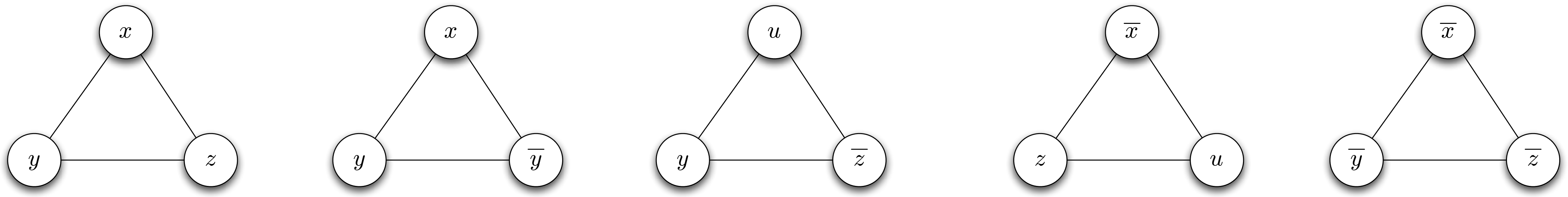
what must we do to?

$3\textsc{Sat} \le_p \textsc{IndSet}$

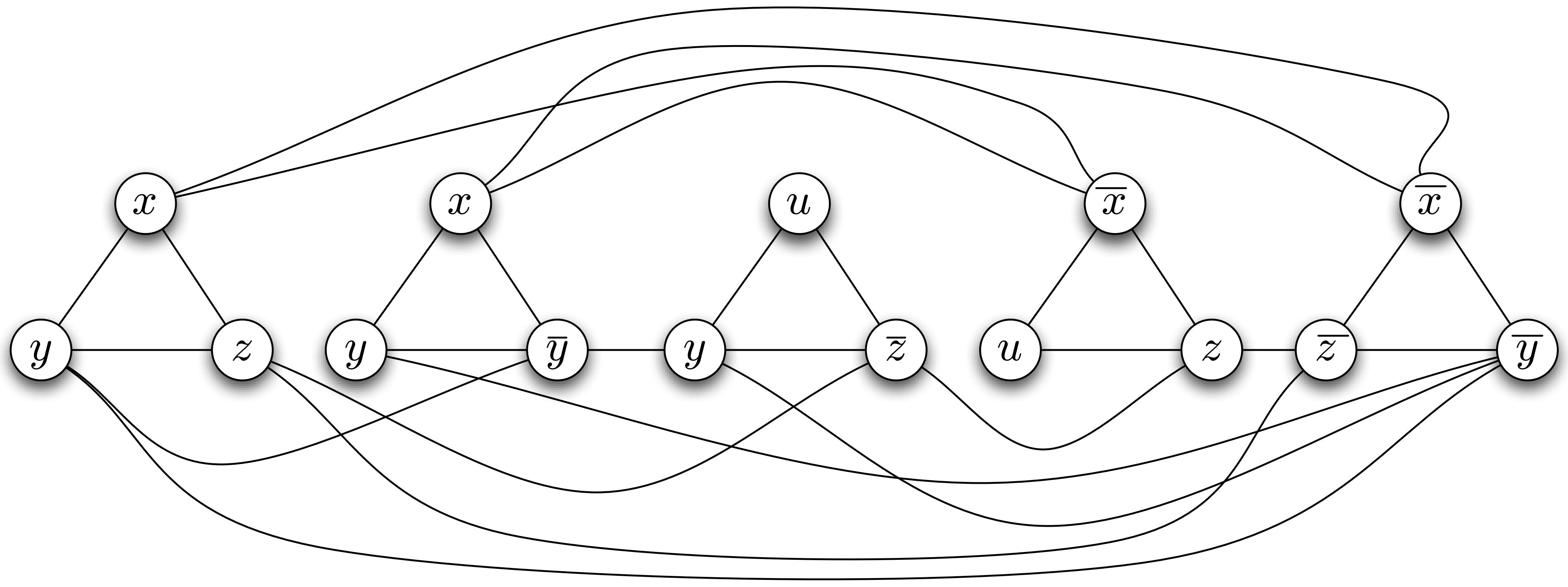
$$(x \vee y \vee z) \wedge (x \vee \overline{y} \vee y) \wedge (u \vee y \vee \overline{z}) \wedge (z \vee \overline{x} \vee u) \wedge (\overline{x} \vee \overline{y} \vee \overline{z})$$

3SAT $\leq_p$ INDSET

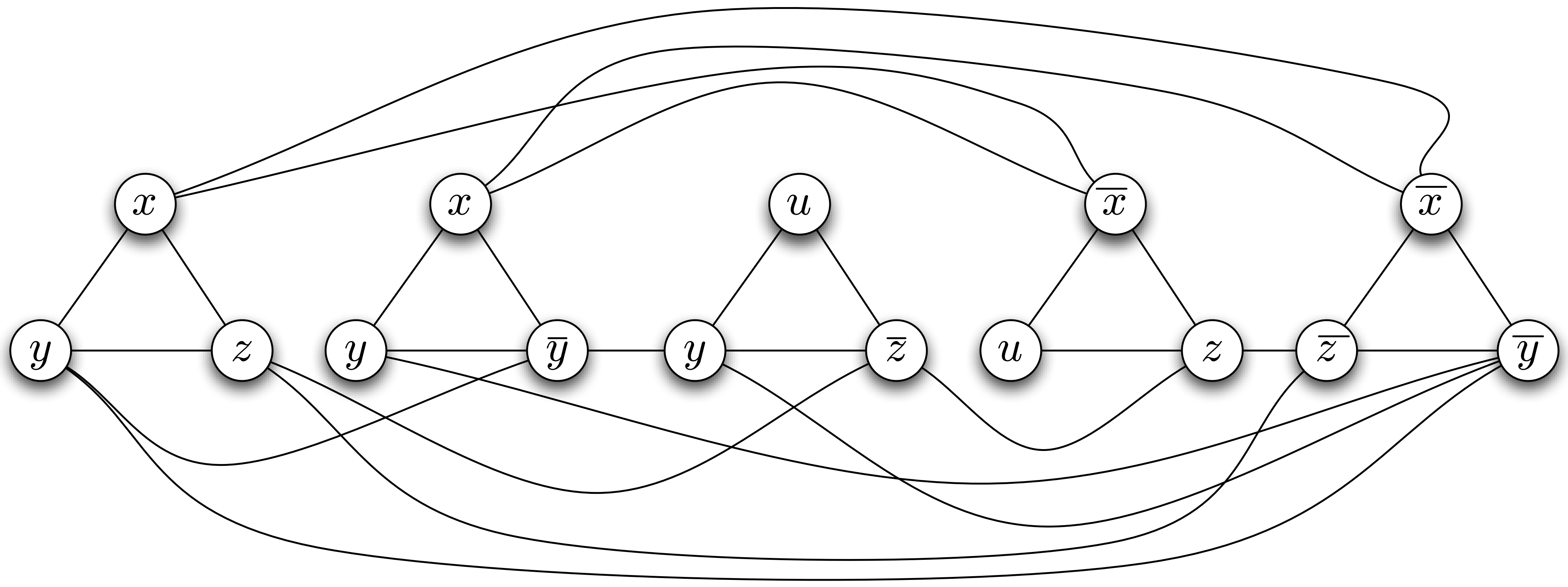
$$(x \vee y \vee z) \wedge (x \vee \overline{y} \vee y) \wedge (u \vee y \vee \overline{z}) \wedge (z \vee \overline{x} \vee u) \wedge (\overline{x} \vee \overline{y} \vee \overline{z})$$



$$(x \vee y \vee z) \wedge (x \vee \overline{y} \vee y) \wedge (u \vee y \vee \overline{z}) \wedge (z \vee \overline{x} \vee u) \wedge (\overline{x} \vee \overline{y} \vee \overline{z})$$

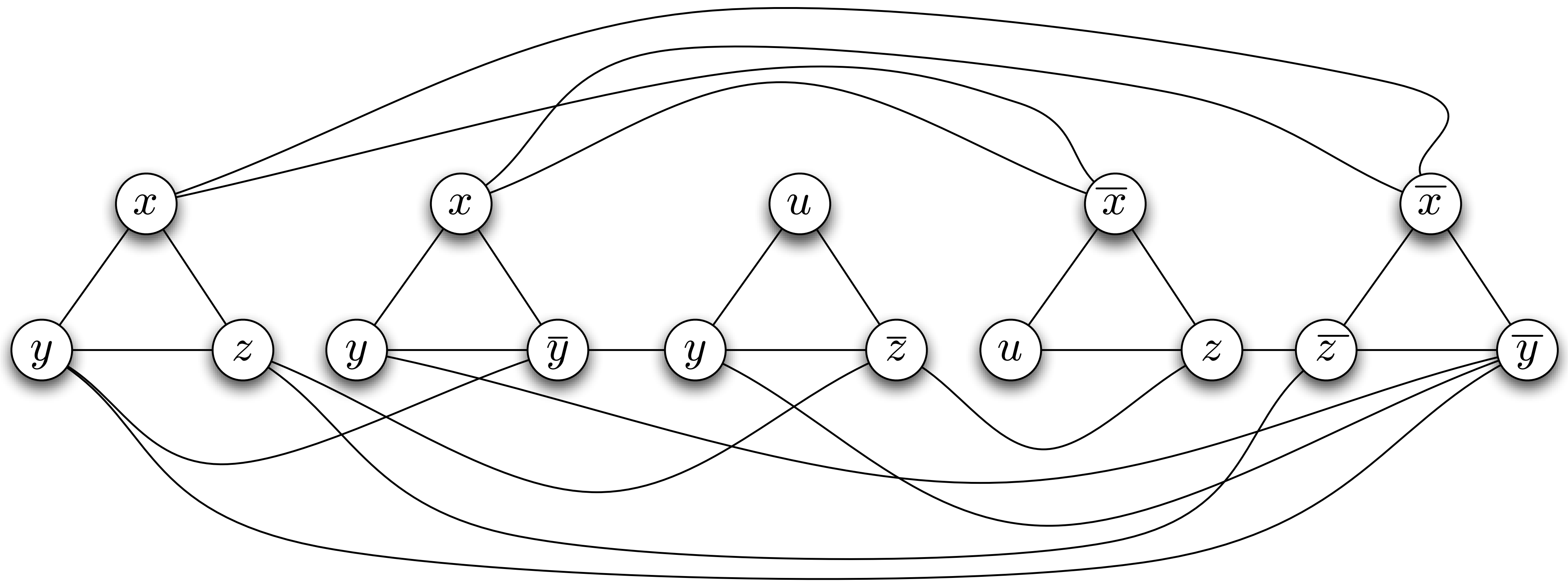


make a triangle for each clause. connect each node to its opposites. make sure that the graph has an independent set of size # of clauses.



$\phi \in \text{SAT} \implies$

each clause is satisfied. pick one variable that makes each clause true and add to INDSET. its negation will never be selected, and therefore we have an INDSET Of size k.



$(G, k) \in$ INDSET $\implies$

each triangle has one node selected only. if k can be selected, each clause has one. make this variable true.

why do we care? because now we know something about hardness of this problem...we shall see it is as hard as any other important problem in the class NP

COMPLEXITY THEORY

one will spend roughly 1/2 of 3102 on these types of reductions...
the key idea for complexity theory is a tool from algorithms however...the reduction.

Theory of NP

DEFINITION OF NP

A language L

belongs to the class NP if and only if

DEFINITION OF NP

a language L belongs to the class NP iff

$\exists A, c$ such that

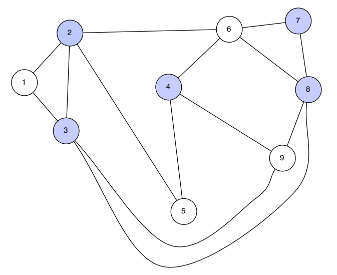
$$L = \{x \in \{0,1\}^* \mid \exists y \in \{0,1\}^{|x|^c} \text{ s.t. } A(x,y) = 1\}$$

means that we can "check" whether the instance satisfies some property in poly time.

WHY IS TRIPLETS IN NP?

$$(x_1, x_2, \ldots, x_n)$$

WHY IS INDSET IN NP?



COMPLEXITY CLASSES

NP

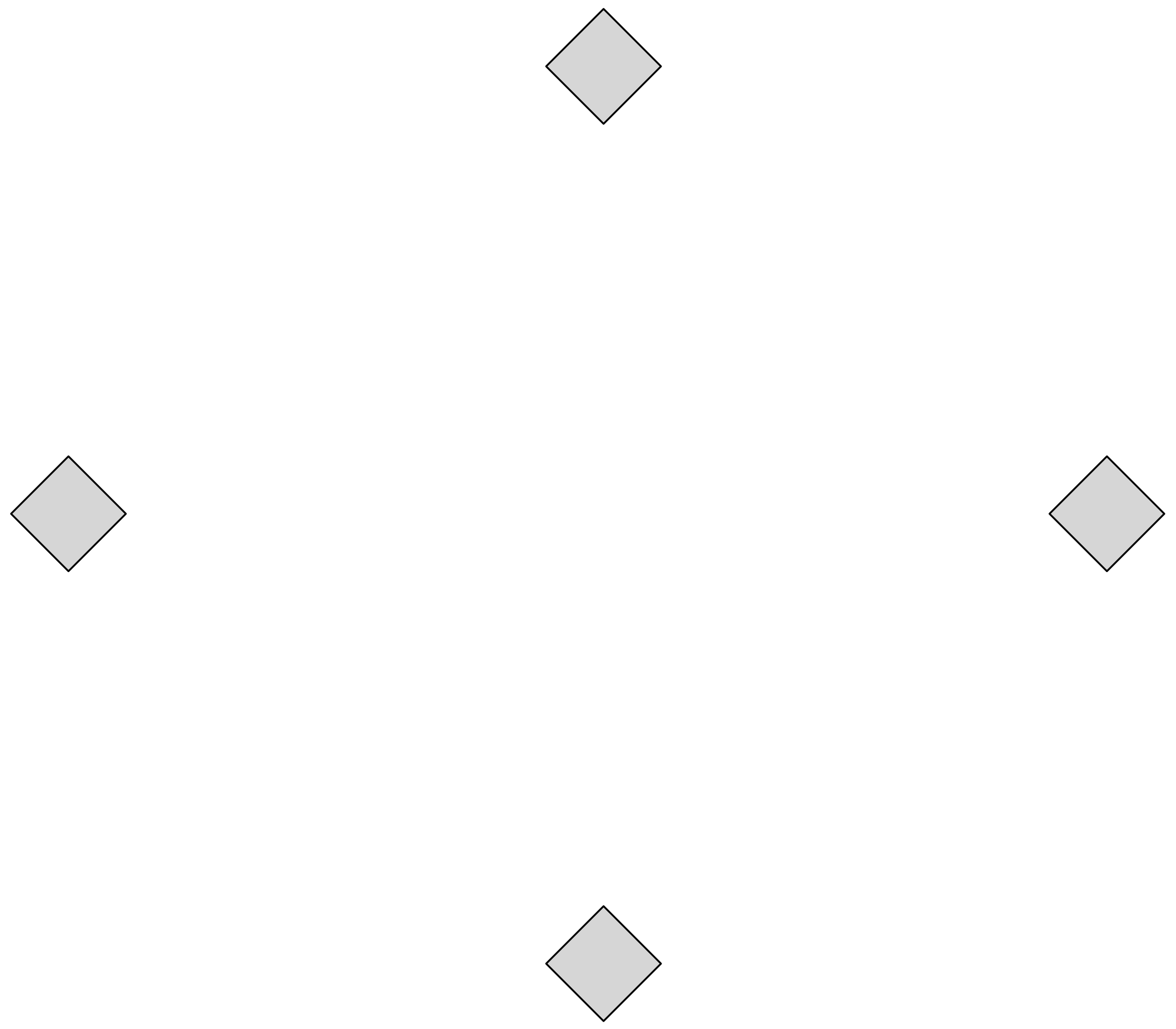
P

COOK-LEVIN THEOREM

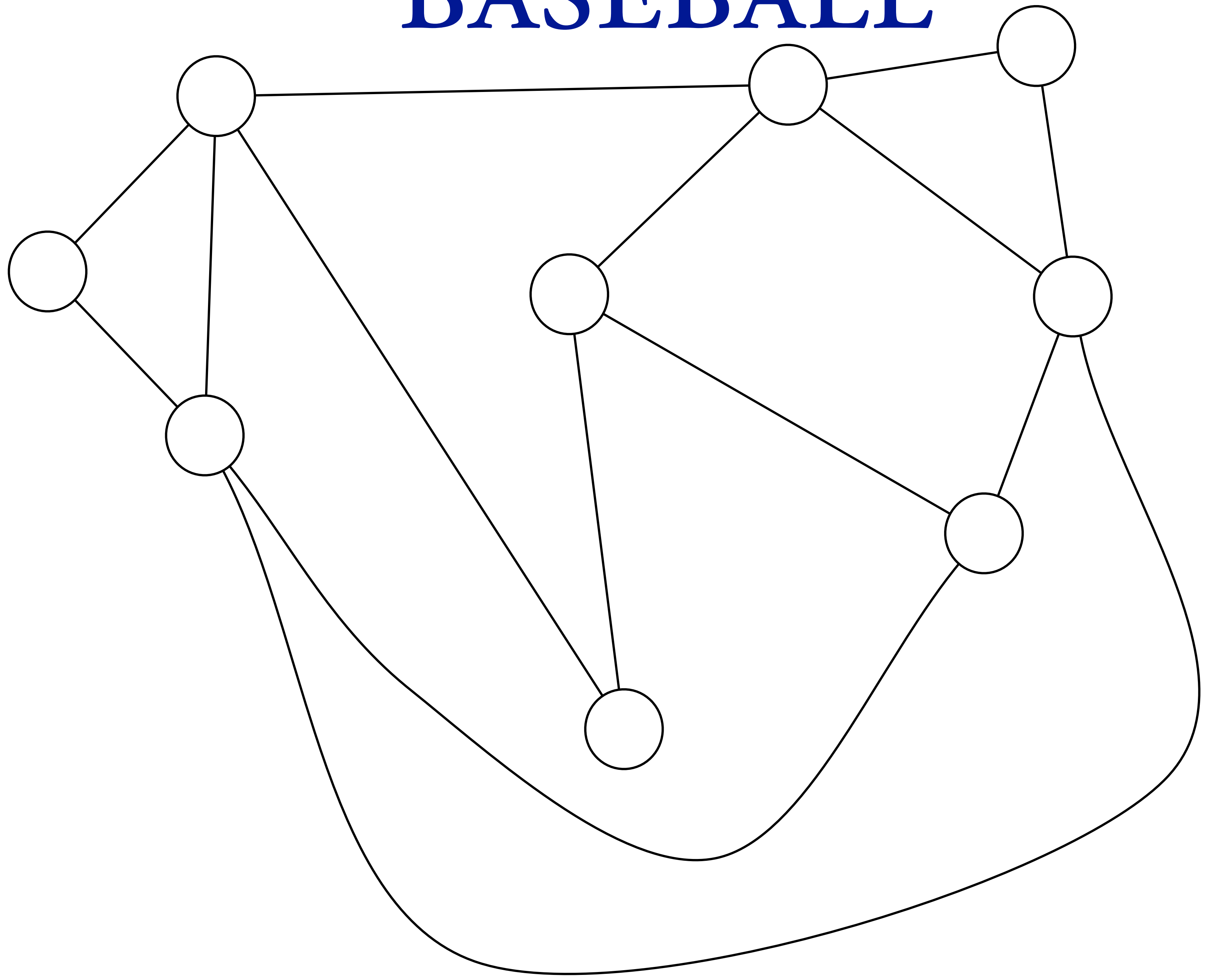
for all L in NP, L <= SAT

The implication of this

BASEBALL



BASEBALL



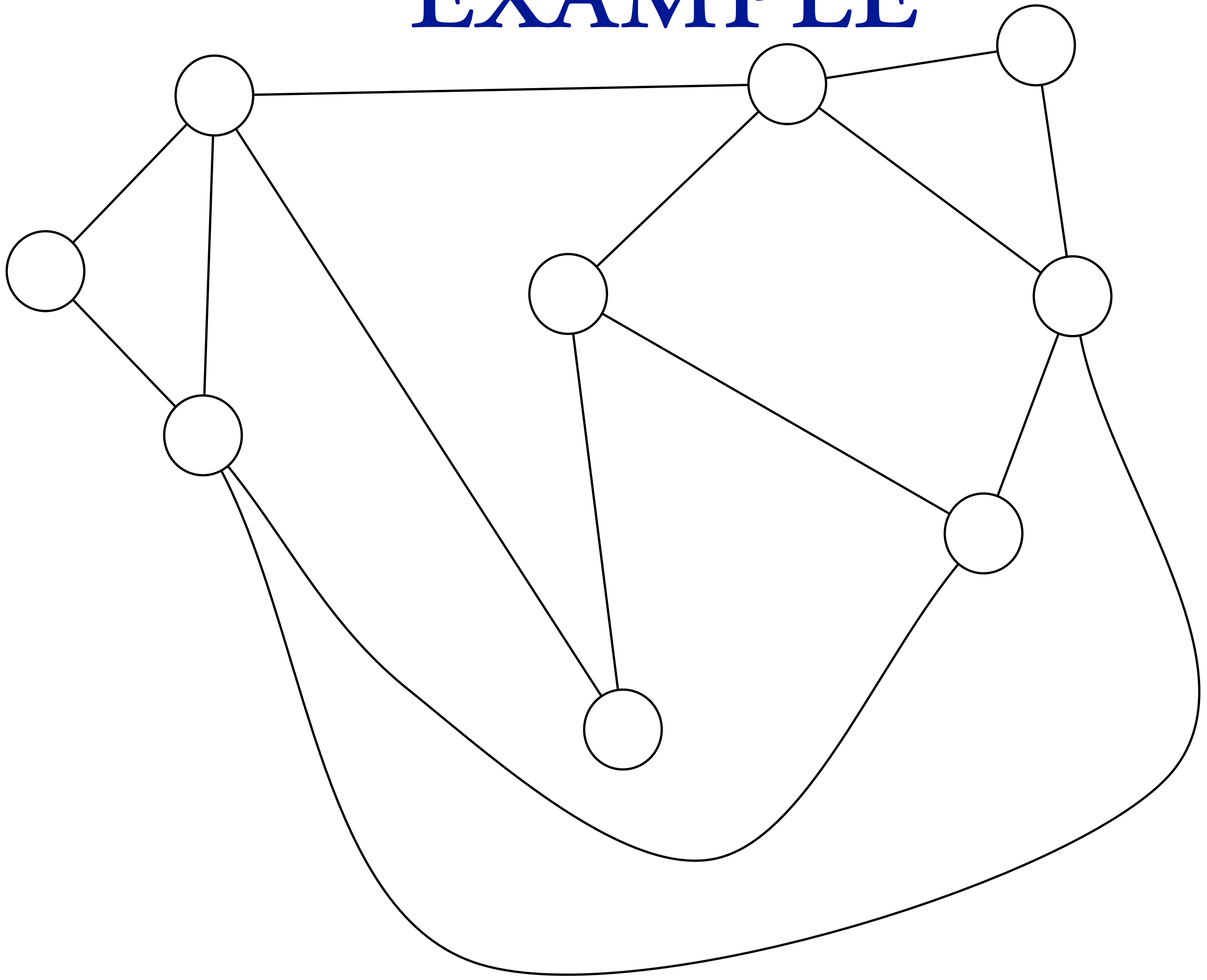
A VERTEX COVER OF A GRAPH IS A

A VERTEX COVER OF A GRAPH IS A SET $C \subseteq V$

SUCH THAT $\forall\ (x, y) \in E$

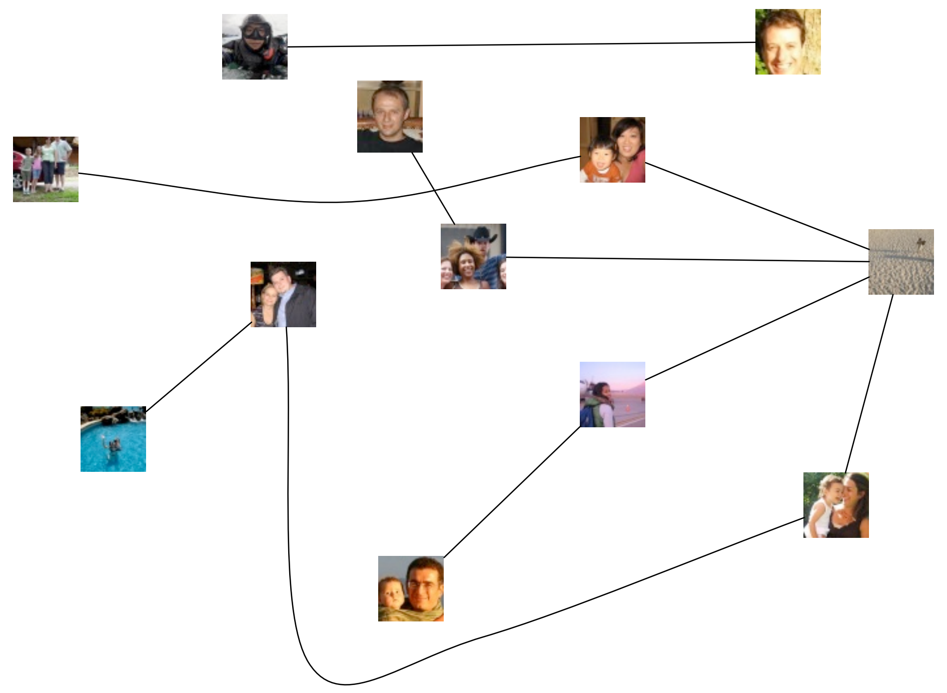
EITHER $x \in C$ OR $y \in C$

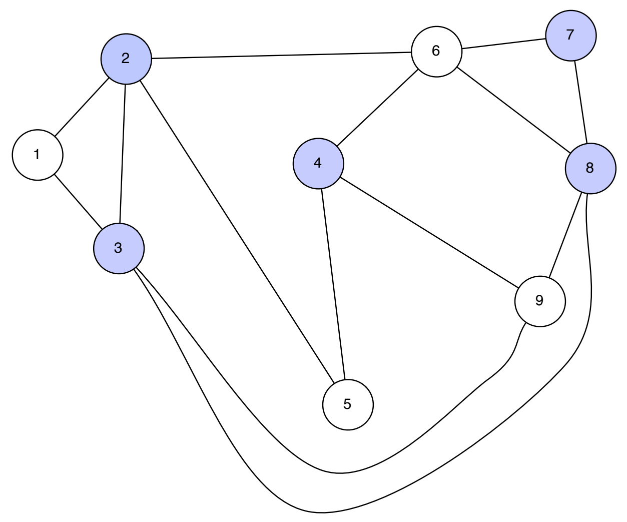
EXAMPLE



GOAL:

GIVEN A GRAPH G,





$$\text{MAXINDSET} \le_{O(V)} \text{MINVERTEXCOVER}$$