Final Exam Revision

Exam Information

- Open Book Exam

Tips for Preparing the Exam

- Stay loose and calm down!
- Sleep well, at least for the few days before the exam!
- Higher priority on the second half of the course (after midterm):
  - DP, flow/matching, LP, NP-hardness/NP-completeness, approximation algorithms
- Familiarity with chapter content
  - So you can quickly know which slide to look up to in the exam
- Review homework assignments and the midterm exam

Tips During the Exam

- Focus only on the questions! Don’t think about anything else!
- For the algorithm design questions, focus primarily on “design”.
- You can use any theorems/results in the lecture slides.
- Time management tip 1: Do not write your solutions in too details. Try to solve all the problems first. You can come back to perfect your solutions later.
- Time management tip 2: If you spend more than 20 minutes on thinking about a problem, you should skip it at the moment and come back later.

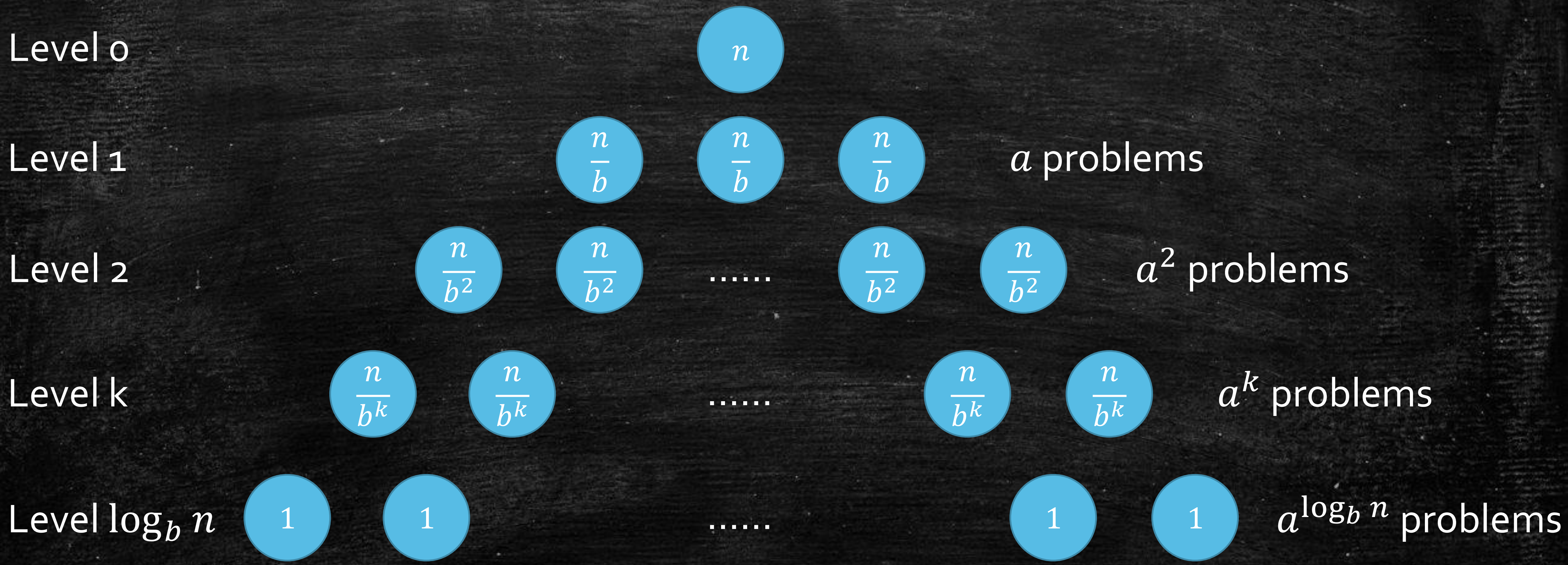
Course Content Overview

1. Divide and Conquer
2. Graph Algorithms
3. Greedy
4. DP
5. Flow/Matching
6. LP
7. Hardness and Approximation Algorithms

Divide and Conquer

- Karatsuba
- Strassen
- Sorting (Insertion, Merge)
- Counting Inversions
- “Median-of-the-median”
- Closest pair
- Fast Fourier Transform

Master Theorem



Master Theorem

- Master Theorem
  - If $T(n) = \boldsymbol{a}T\left(\frac{n}{\boldsymbol{b}}\right) + O(n^{\boldsymbol{d}})$

Combining cost: $O(n^d)$

Divide into a subproblems

Subproblem size: n/b

  - $T(n) = \begin{cases} O(n^d) & a < b^d \\ O(n^{\log_b a}) & a > b^d \\ O(n^d \log n) & a = b^d \end{cases}$

Correctness Analysis

- Induction

Time Complexity Analysis

- Master Theorem
  - for most problems
- Guess and Induction
  - “Median-of-the-median”, Midterm Q1

Graph Algorithms

- DFS
- BFS
- Dijkstra
- Bellman-Ford
- Floyd-Warshall (Dynamic Programming)
- Kruskal and Prim (Greedy)

DFS vs BFS

	DFS	BFS
Detecting Cycles	YES	NO
Topological Ordering	YES	NO
Finding CCs	YES	YES
Finding SCCs	YES	NO
Shortest Path	NO	YES

- Hard to distinguish cross edge and back edges in BFS
- Finish time is meaningful in BFS

Shortest Path Algorithms

Algorithm	Single Source?	Graph	Complexity
BFS	Single Source	Unweighted	$O(\|V\| + \|E\|)$
Dijkstra	Single Source	Positively Weighted	$O(\|V\|^2 + \|E\|)$ (Fibonacci Heap)
Bellman-Ford	Single Source	General weighted	$O(\|V\| \cdot \|E\|)$
Floyd-Warshall	All Pairs	General weighted	$O(\|V\|^3)$

Greedy Algorithms

Exact Algorithms:

- Minimum Spanning Tree
  - Prim
  - Kruskal (Union-Find Set, Path Compression)
- Task Schedule (Earliest Deadline First)
- Huffman Coding

Approximation Algorithms:

- Makespan Minimizing
- Set Cover/Max-k-Coverage

Greedy Algorithms

- Easier to design
- Harder to analyze

Analyzing Greedy Algorithms

Exact Algorithms:

- Induction
- Show that the solution at the current iteration is still a part of an optimal solution.

Approximation Algorithms:

- Require adequate understanding on the problem's nature
- Find a "reference" that your solution can compare with
- Reference: OPT, or lower bound (upper bound) to OPT
- and other tricks….

Kruskal Algorithm

- Initialize $S = \emptyset \subseteq E$.
- Sort E in weight-ascending order.
- For each $e \in E$, if $S \cup \{e\}$ does not contain a cycle, update $S \leftarrow S \cup \{e\}$.

Kruskal Algorithm – Correctness

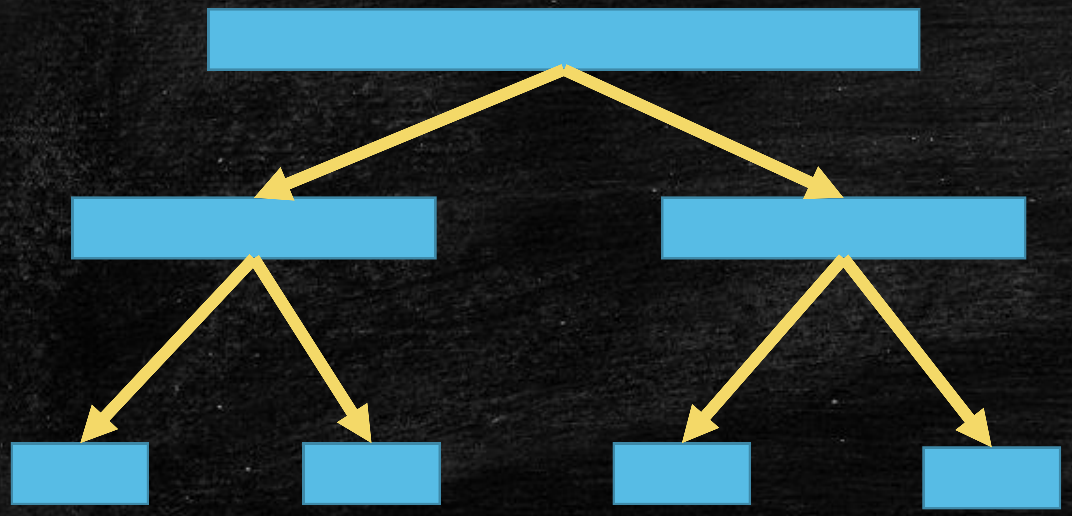
- Let S_i be the tree at i-th iteration. We will prove by induction that S_i is a part of an optimal solution for all i.
- Base Step: $S_0 = \emptyset$ is clearly a part of an optimal solution.
- Inductive Step: Suppose $S_i \subseteq S^*$ for some MST S^*.
- Let $S_{i+1} = S_i \cup \{e\}$. If $S_{i+1} \subseteq S^*$, we are done.
- Suppose $S_{i+1} \nsubseteq S^*$. Then $S^* \cup \{e\}$ contains a cycle C.
- Case 1: $\exists e' \in C \setminus S_{i+1}$ with $w(e') \geq w(e)$.
- Then, $S^{**} = S^* \cup \{e\} \setminus \{e'\}$ is a spanning tree with $w(S^{**}) \leq w(S^*)$. S^{**} is also optimal, and $S_{i+1} = S_i \cup \{e\} \subseteq S^{**}$.

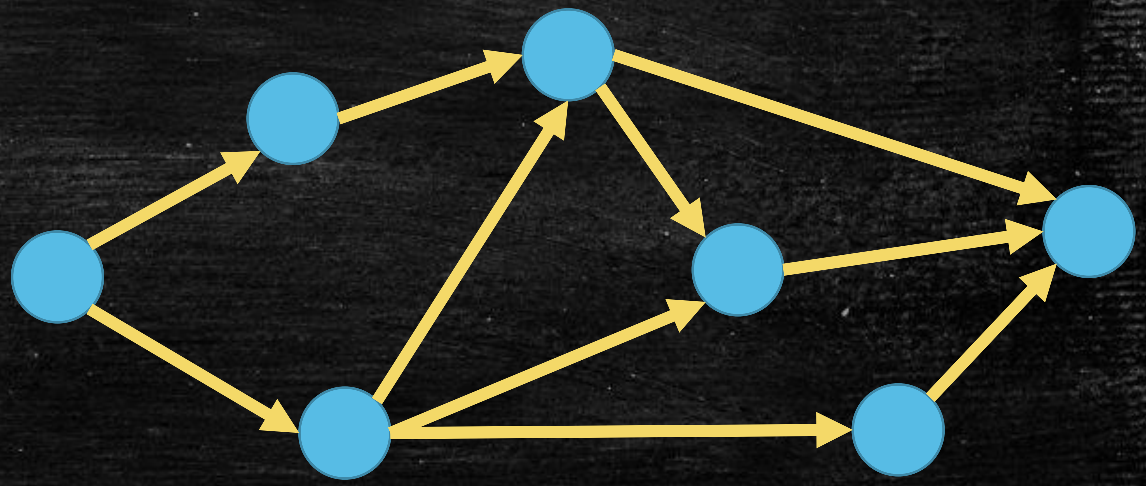
Kruskal Algorithm – Correctness (cont'd)

- Case 2: $w(e') < w(e)$ for $\forall e' \in C \setminus S_{i+1}$.
- For any $e' \in C \setminus S_{i+1}$, we have $e' \in C \subseteq S^*$ and $S_i \subseteq S^*$ (induction hypothesis), so $S_i \cup \{e'\} \subseteq S^*$.
- This implies $S_i \cup \{e'\}$ contains no cycle.
- Then the algorithm should not choose e at the $(i+1)$-th iteration, as choosing e' with smaller weight does not create a cycle.
- We have a contradiction.

Dynamic Programming

- Break problems into subproblems
- Divide and Conquer: subproblems form a tree
- Dynamic Programming: subproblems form a directed acyclic graph





Dynamic Programming

- Longest Increasing Sequence
- Edit Distance
- Knapsack
- Floyd-Warshall
- Independent Set on Trees (also a greedy algorithm)

Correctness of DP

- Induction…
- Validity of recurrence relation is just the validity of inductive step!

Dynamic Programming Exercise

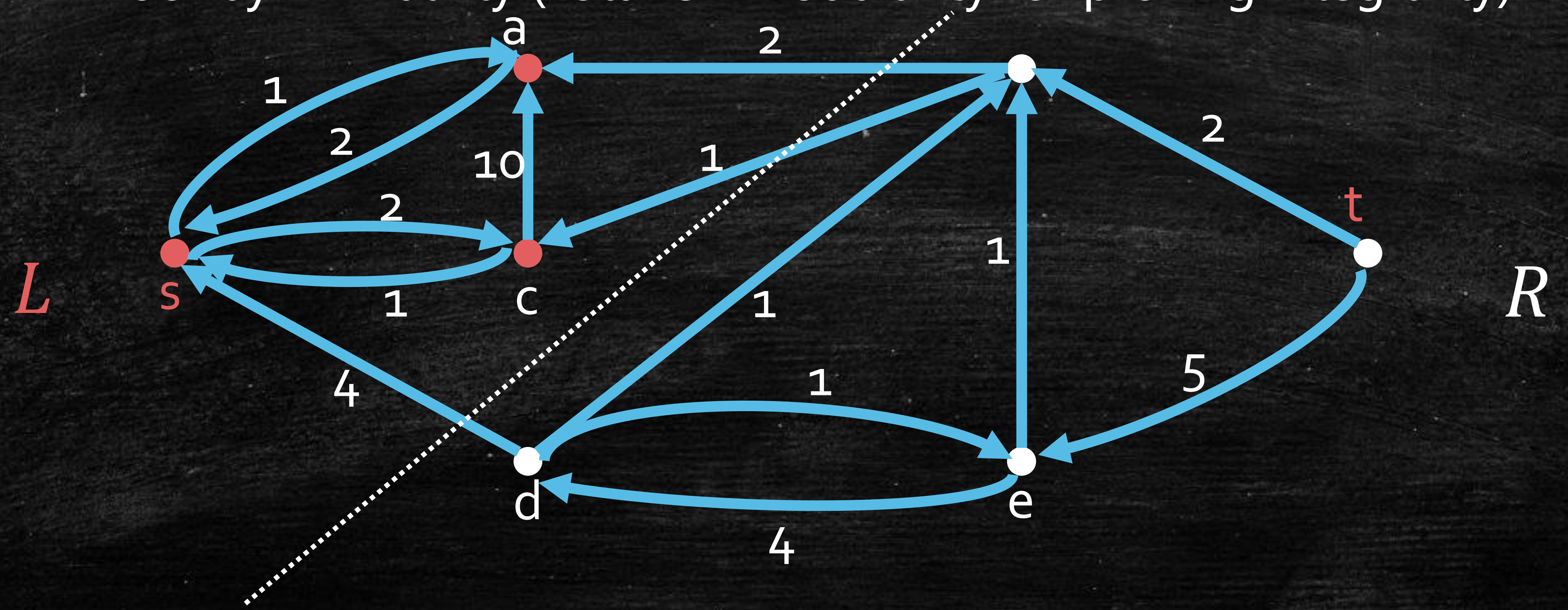
- Given sequence $h_1, h_2, \dots, h_n \in \mathbb{Z}^+$ indicating the height of each piece of land along a line. The goal is to come up with a non-decreasing sequence $h'_1, h'_2, \dots, h'_n$ such that the "absolute change" $\sum_{i=1}^{n} |h_i - h'_i|$ is minimized.
- Note: All of the heights are integers, and $h_i \in [0, H]$.

Max-Flow

- Ford-Fulkerson Method: $O(|E| \cdot f_{max})$
- Edmonds-Karp Algorithm: $O(|V| \cdot |E|^2)$
- Dinic's Algorithm: $O(|V|^2 \cdot |E|)$

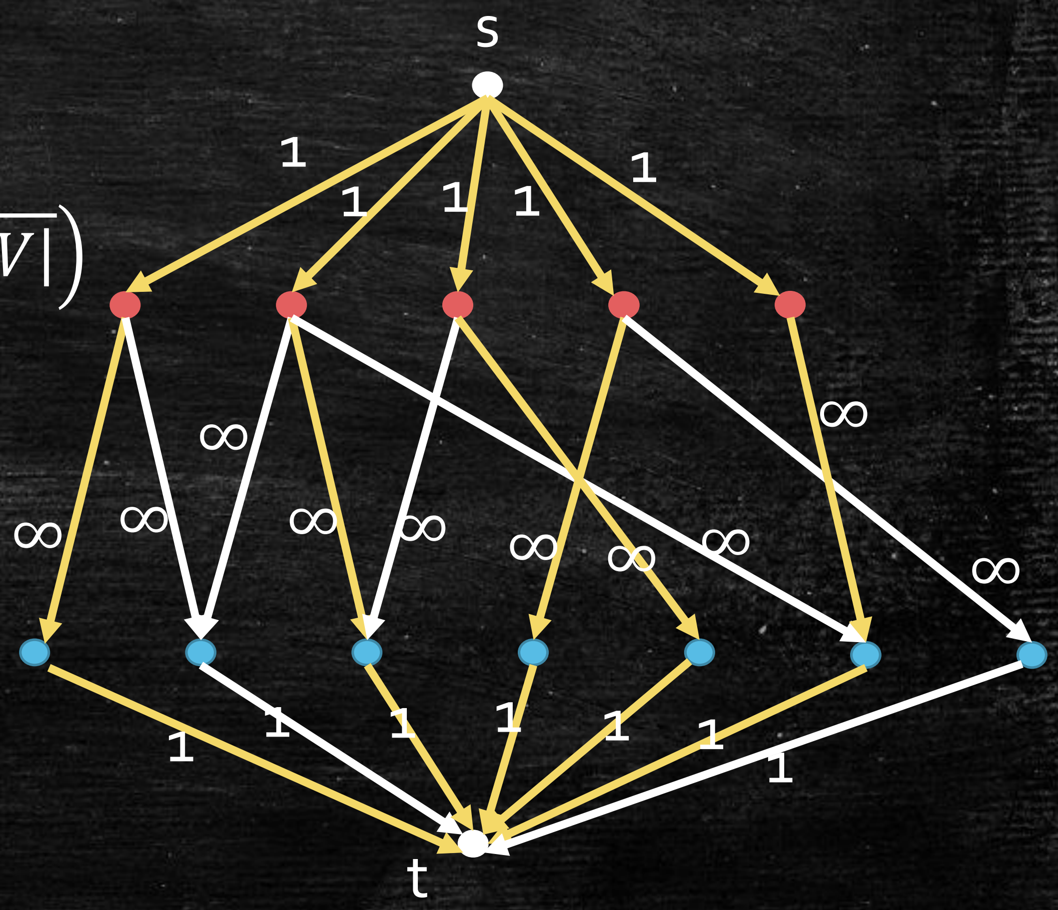
Max-Flow Related Problems

- Min-Cut: Max-Flow-Min-Cut Theorem
- Two proofs:
  - Proof by analyzing G^f
  - Proof by LP-Duality (Total Unimodularity for proving integrality)



Max-Flow Related Problems

- Maximum Cardinality Matching
- Flow integrality:
  - Integer capacities $\Longrightarrow$ Integral flow
- Hopcroft-Karp Algorithm: $O\left(|E|\sqrt{|V|}\right)$



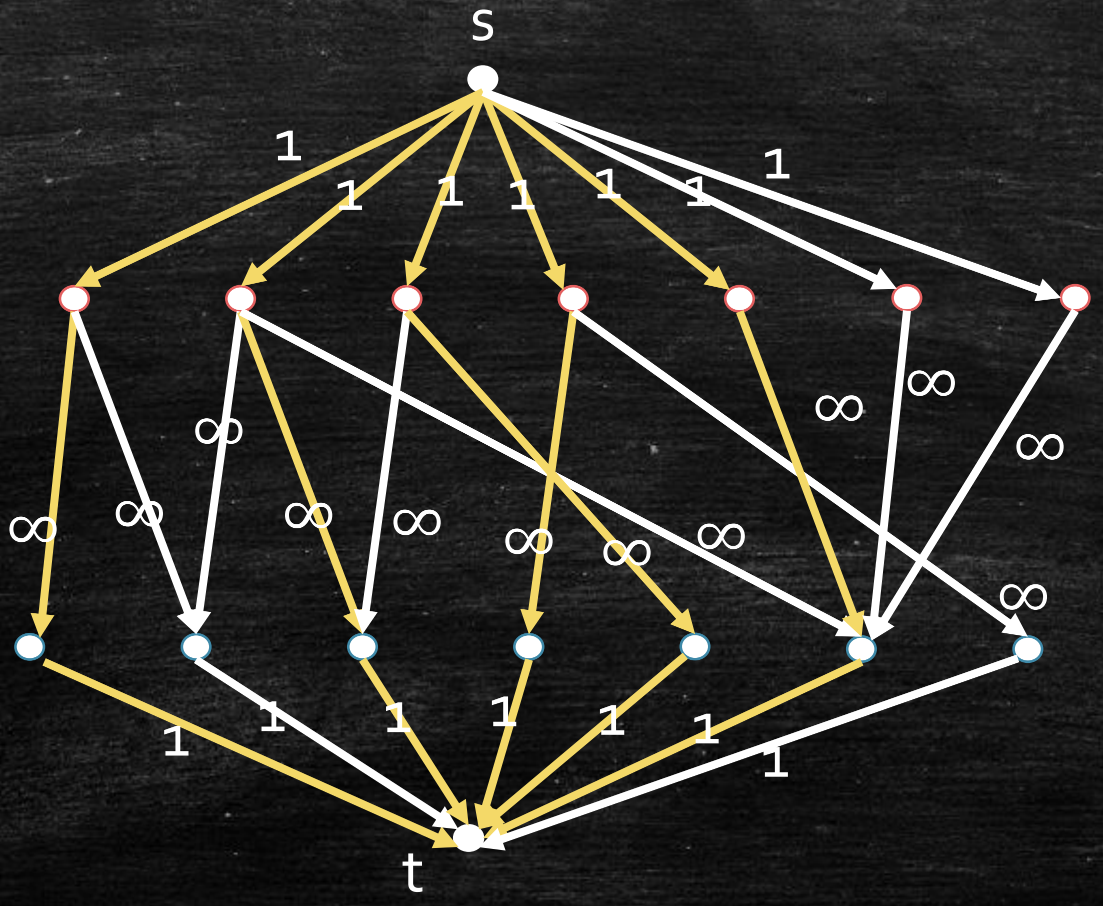
Problems on Bipartite Graphs

- Maximum Cardinality Matching
- Maximum Independent Set
- Minimum Vertex Cover

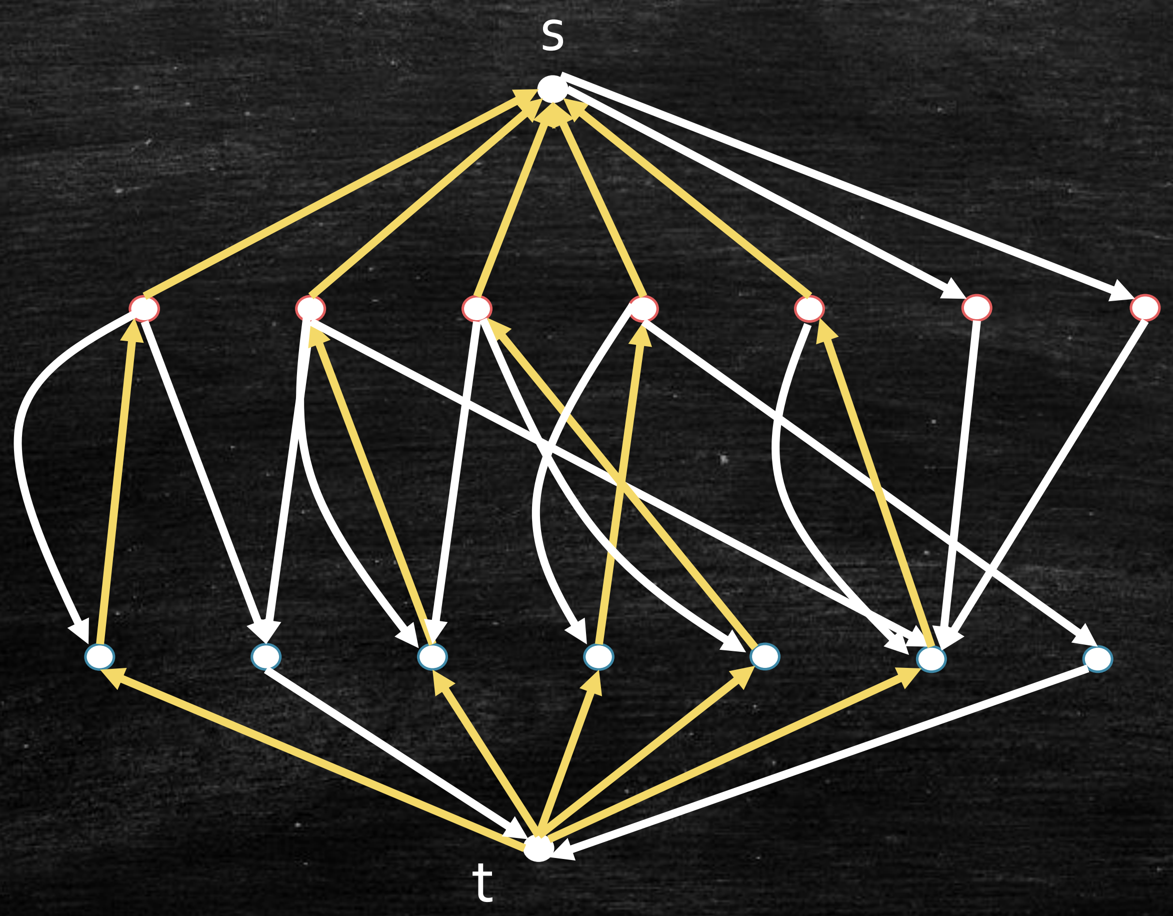
[Advanced] Hungarian Algorithm:

- Maximum Weight Perfect Matching
- Minimum Weight Perfect Matching
- Maximum Weight Matching

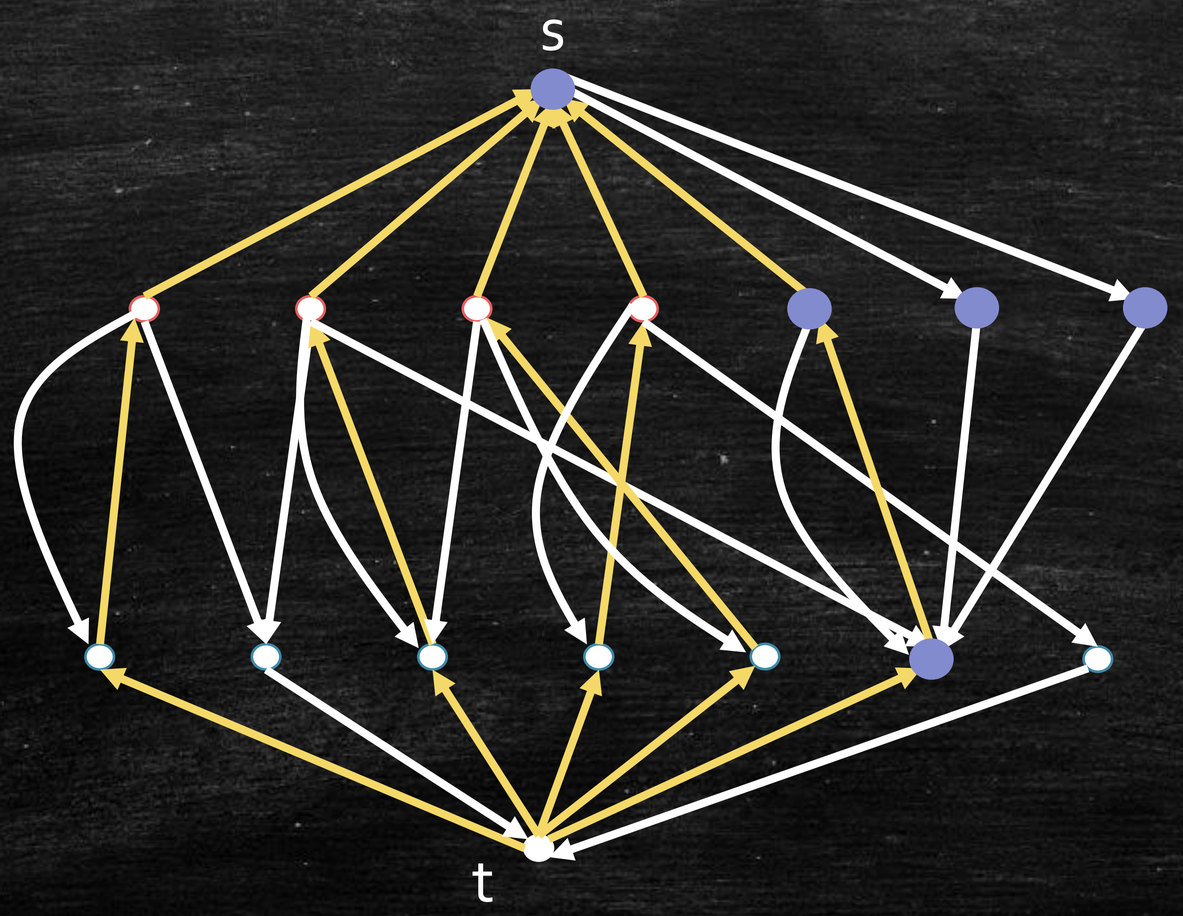
Max-Flow = 5



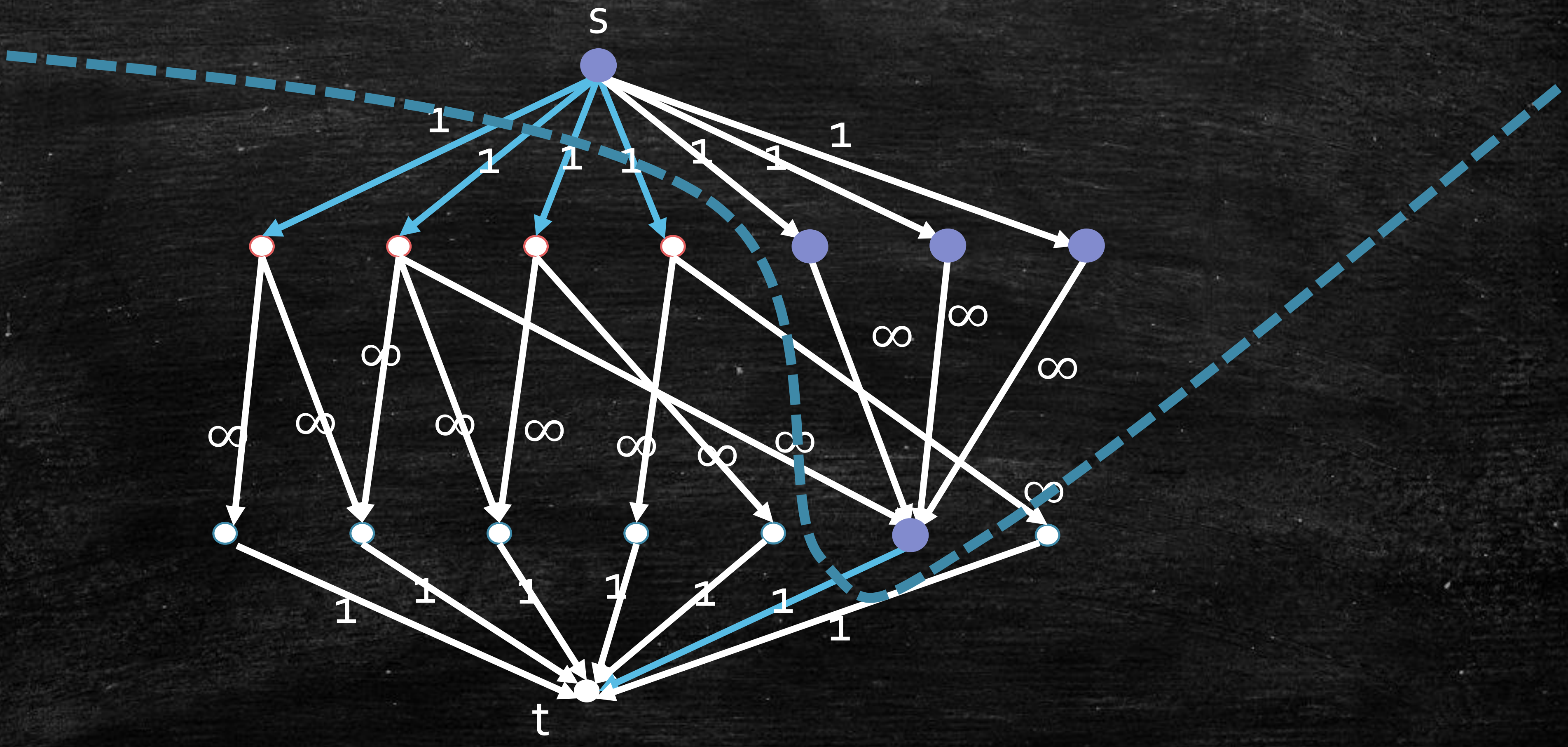
Residual Graph G^f



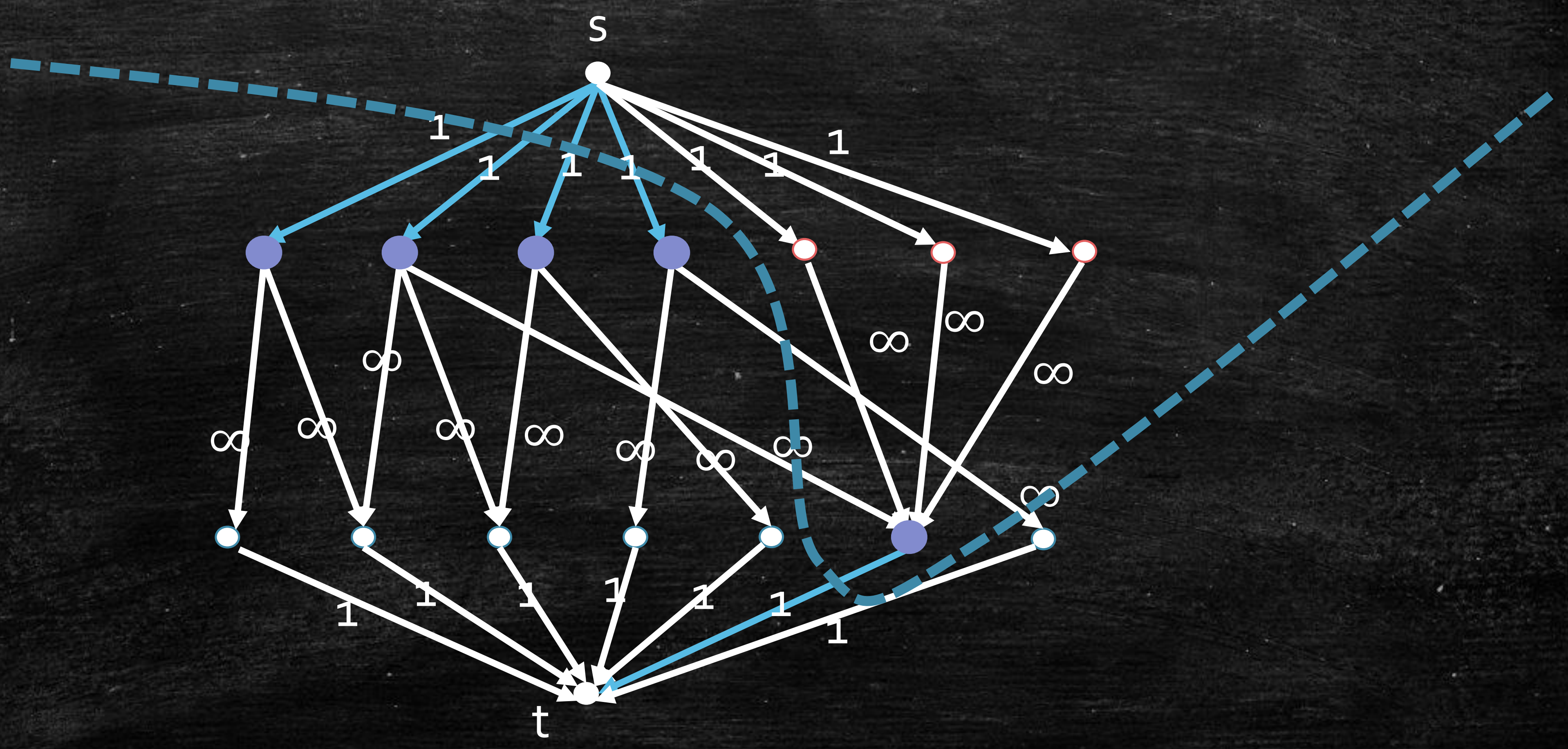
Vertices Reachable from s in G^f



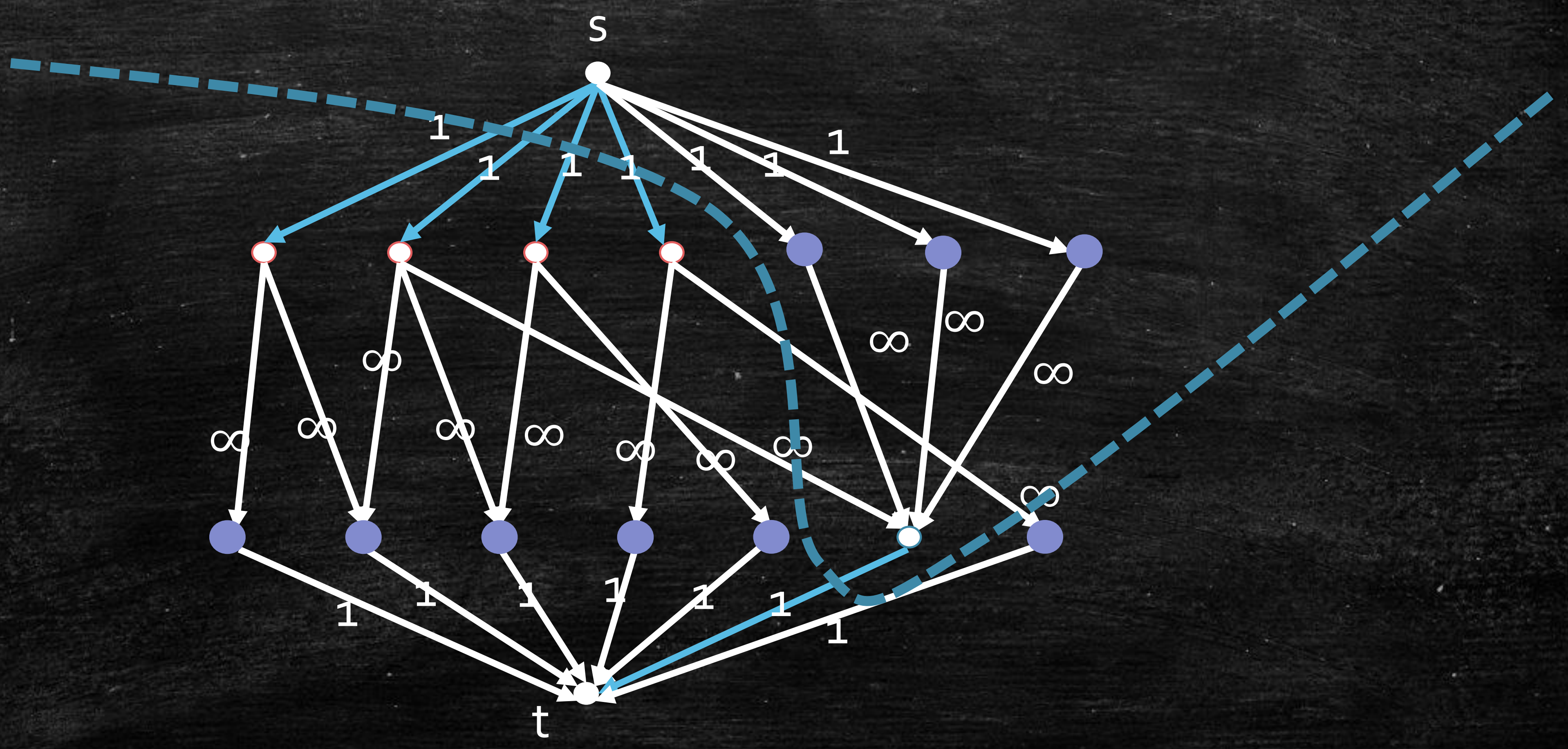
Min-Cut = 5



Min Vertex Cover = 5

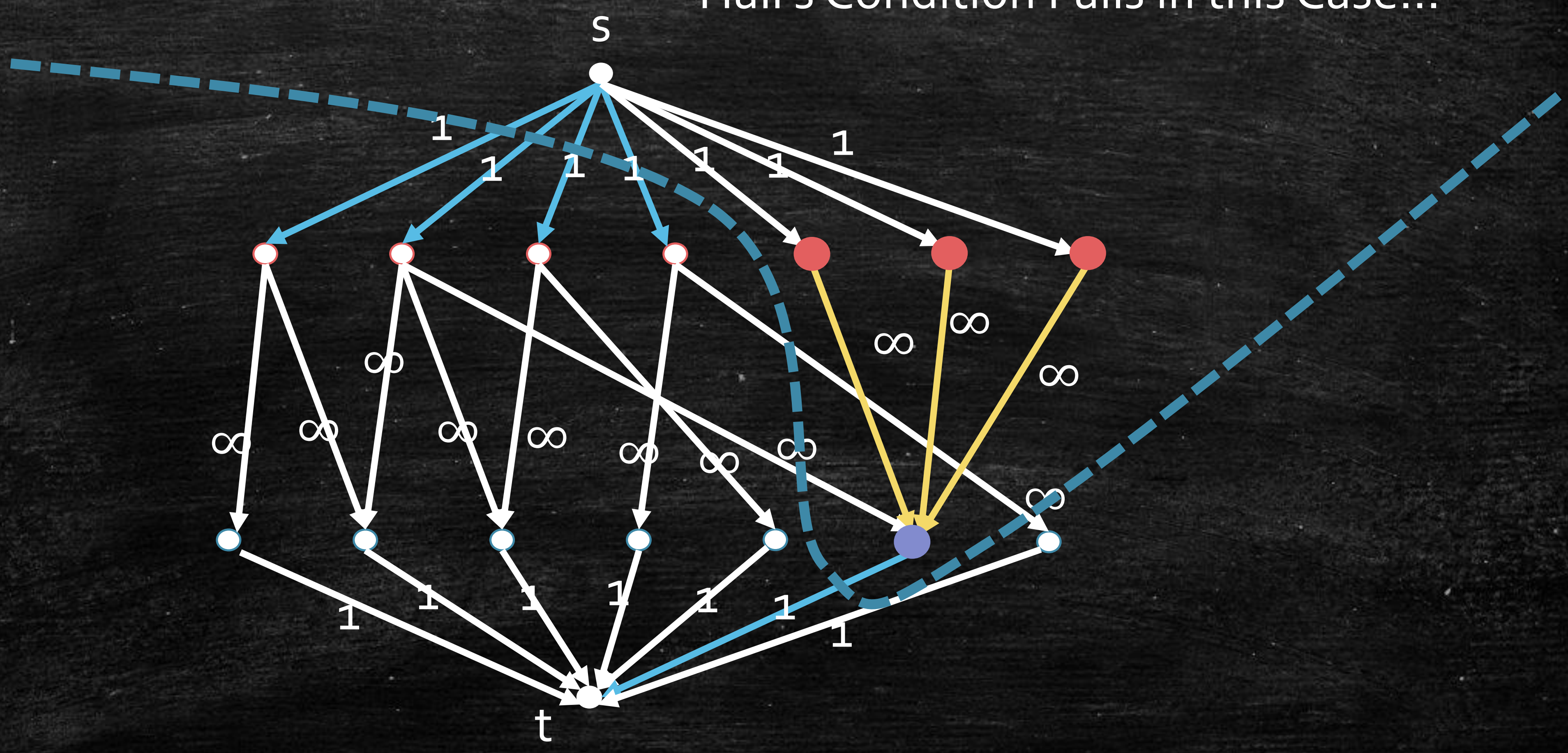


Max Independent Set = 9   (14 – 5 = 9)



Hall's Marriage Theorem

Hall's Condition Fails in this Case...



Proof Techniques

- Establish one-to-one correspondence
  - Cut ⟺ Vertex Cover
  - Matching ⟺ Flow
  - Flow ⟺ Cut
  - Vertex Cover ⟺ Independent Set
  - Etc.
- Argue that optimizing one optimizes the other.
- Prove by contradiction:
  - Given a min-cut, we construct a vertex cover by XXX
  - This is a minimum vertex cover
  - Suppose there is an even smaller vertex cover, we can build a cut smaller than the min-cut by do the followings…

Max-Flow Applications

- Min-Cut
- Matching
- Bipartite Vertex Cover/Independent Set
- Tournament
- Other problems that look similar to any of the followings:
  - Max-Flow, Min-Cut, Matching, etc.

Max-Flow Exercises

1. Solve Assignment 5 Q1
2. Suppose students from m different universities participate in a conference. Each university i has r_i students. Suppose the conference has n tables, and each table can be shared by at most c_i students. Decide if it is possible to make an arrangement such that each table is shared by students from different universities.
3. Prove that a bipartite regular graph contains a perfect matching.

Linear Programming

- Polynomial-Time Solvable (Interior-Point Method)
- Exist optimum at a vertex
- Standard Form
- Dual LP
- LP-Duality Theorem: Primal Maximum = Dual Minimum
- LP-Duality Theorem Applications (Total Unimodularity for proving integrality)
  - Max-Flow-Min-Cut Theorem
  - Von Neumann's Minimax Theorem
  - Kőnig-Egerváry Theorem

Linear Programming [Advanced]

- LP-relaxation for approximation algorithm.
  - Vertex Cover
  - Metric Facility Location
- Primal-Dual Method
  - Hungarian Algorithm

LP Exercise

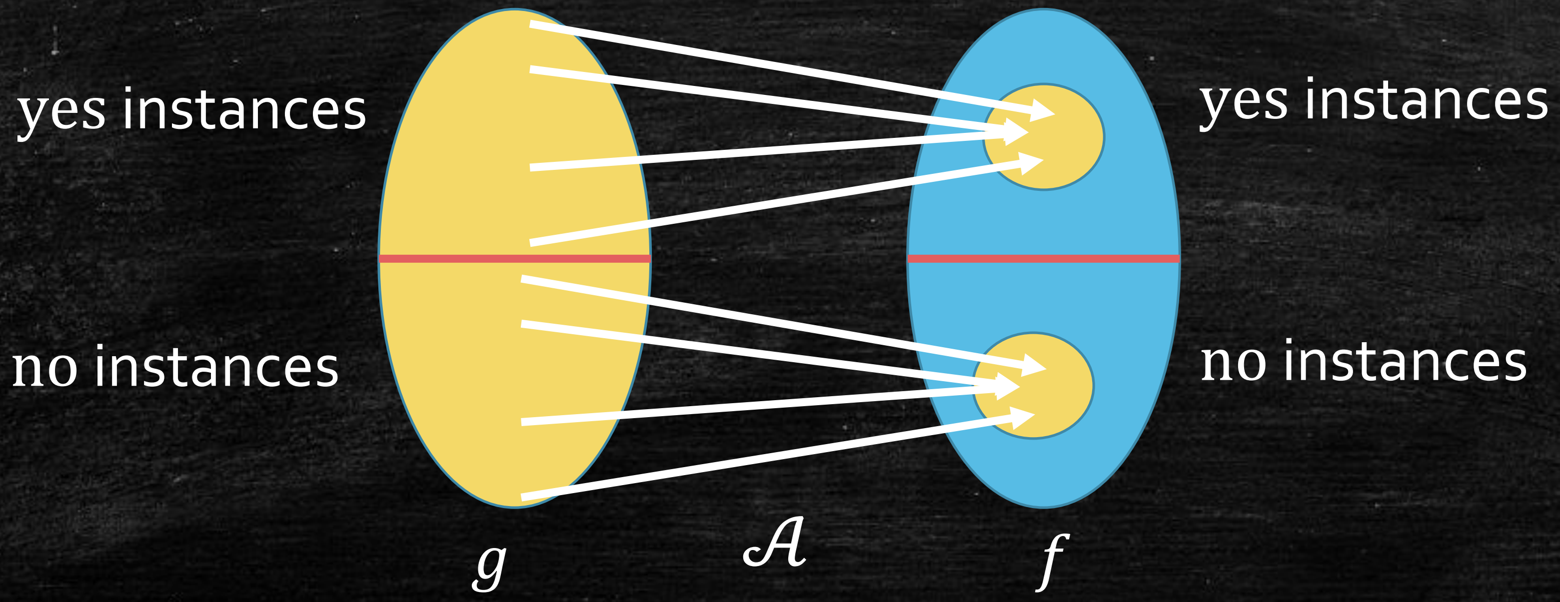
- For the linear program
- maximize $\quad x_1 - 2x_3$
- subject to $\quad x_1 - x_2 \leq 1$
- $\qquad 2x_2 - x_3 \leq 1$
- $\qquad x_1, x_2, x_3 \geq 0$
- prove that the solution $(x_1, x_2, x_3) = \left(\frac{3}{2}, \frac{1}{2}, 0\right)$ is optimal.

NP-Hardness/NP-Completeness

- **P**: decision problems that can be decided efficiently
- **NP**: decision problems that can be verified efficiently
- Reduction is an effective tool to show one problem is "weakly harder" than another.
- NP-Completeness describes the hardest problems in **NP**.
- Cook-Levin Theorem. SAT is NP-complete.

Reduction: $\mathcal{A}$ computes $g \leq_k f$

- $x \mapsto y$ under poly-time TM $\mathcal{A}$
- x is yes $\Longrightarrow$ y is yes
- x is no $\Longrightarrow$ y is no



Proving a decision problem f is NP-Complete

- Show that $f \in$ **NP**
- Find an NP-complete problem g and prove $g \leq_k f$

Four Steps:

1. Prove that f is in **NP**
2. Present the reduction $g \leq_k f$
3. Show that yes instances of g are mapped to yes instances of f
4. Show that no instances of g are mapped to no instances of f
   - Most of the time, it is easier to prove its contrapositive

Difference Between NP-Complete and NP-Hard

- NP-complete = NP-hard + (in NP)
- Difference 1: NP-hard problem needs not to be in NP
- Difference 2: NP-hardness can describe non-decision problems
  - We have seen many NP-hard optimization problems

Proving an optimization problem is NP-hard

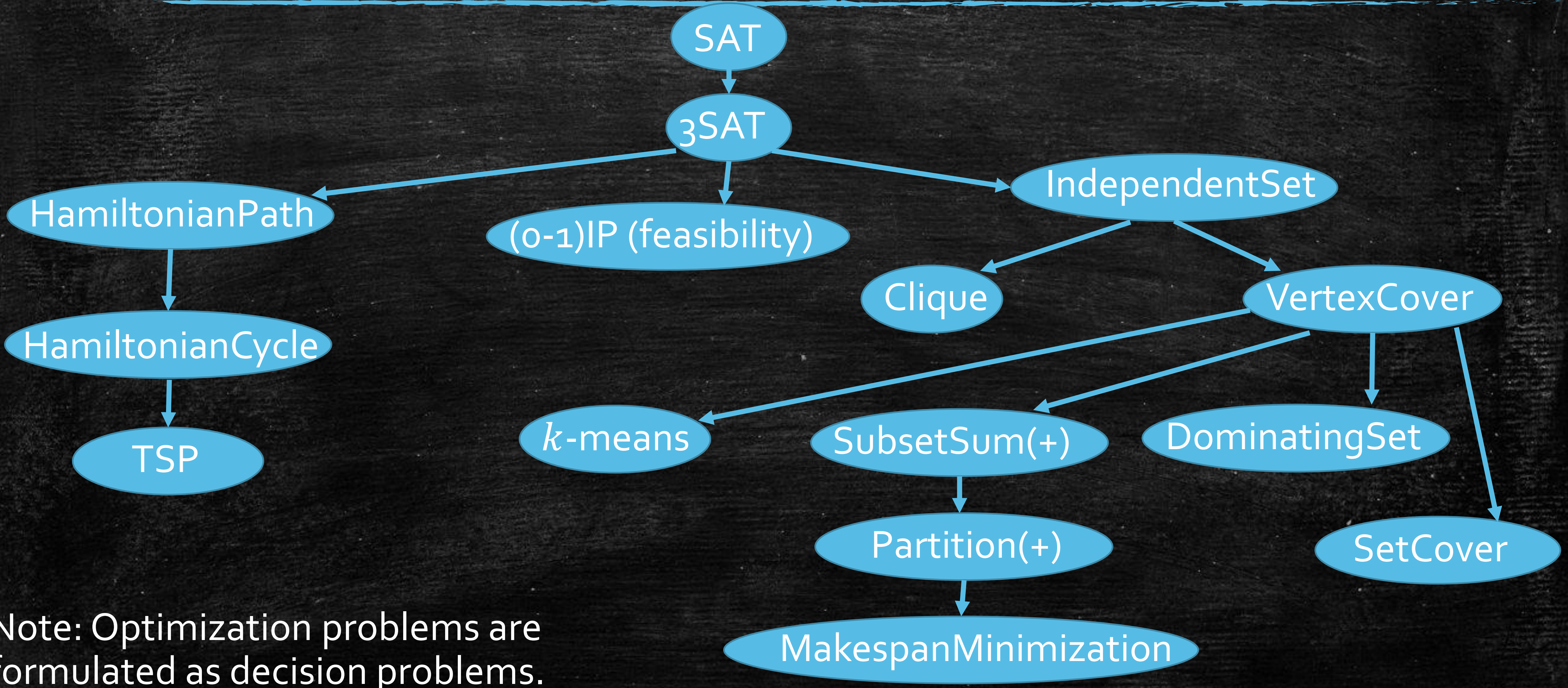
To prove an optimization problem is NP-hard, prove that its "decision version" is NP-hard.

- A maximization problem is NP-hard if there exists $k \in \mathbb{R}$ such that deciding whether $\mathrm{OPT} \geq k$ is NP-hard.
- A minimization problem is NP-hard if there exists $k \in \mathbb{R}$ such that deciding whether $\mathrm{OPT} \leq k$ is NP-hard.

Techniques

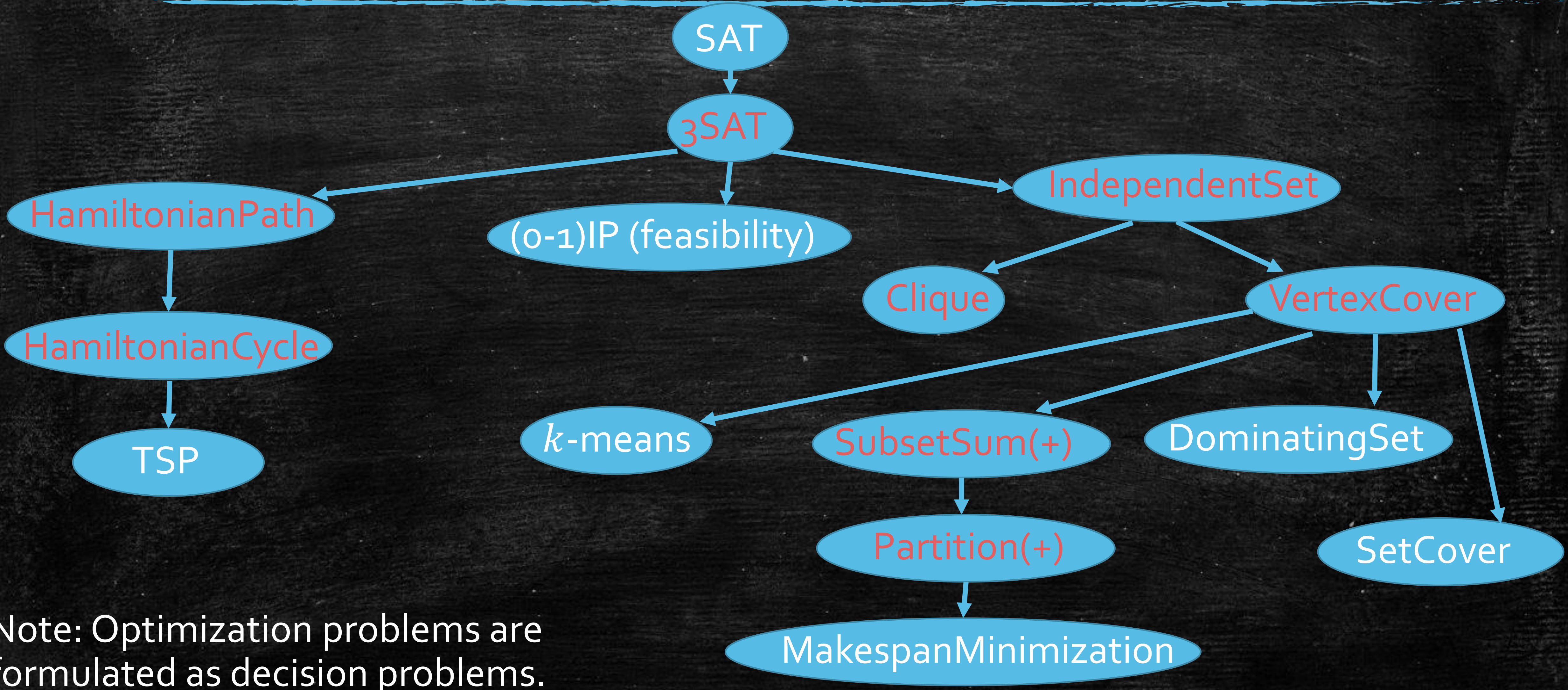
- Choose a suitable problem
- Introducing intermediate problems
- Gadget Construction

You can use any problem below for reduction!



Note: Optimization problems are formulated as decision problems.

I recommend these ones!

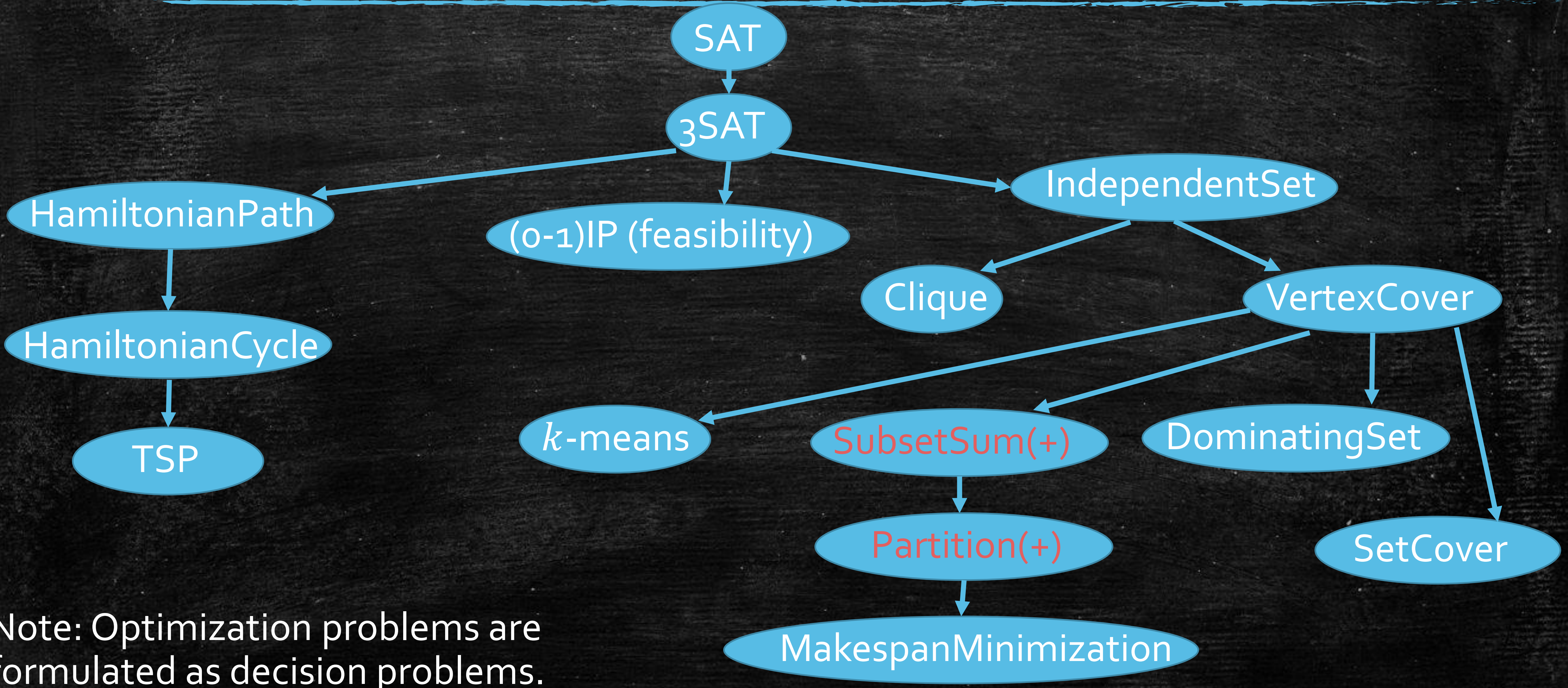


Note: Optimization problems are formulated as decision problems.

Choose a Suitable Problem

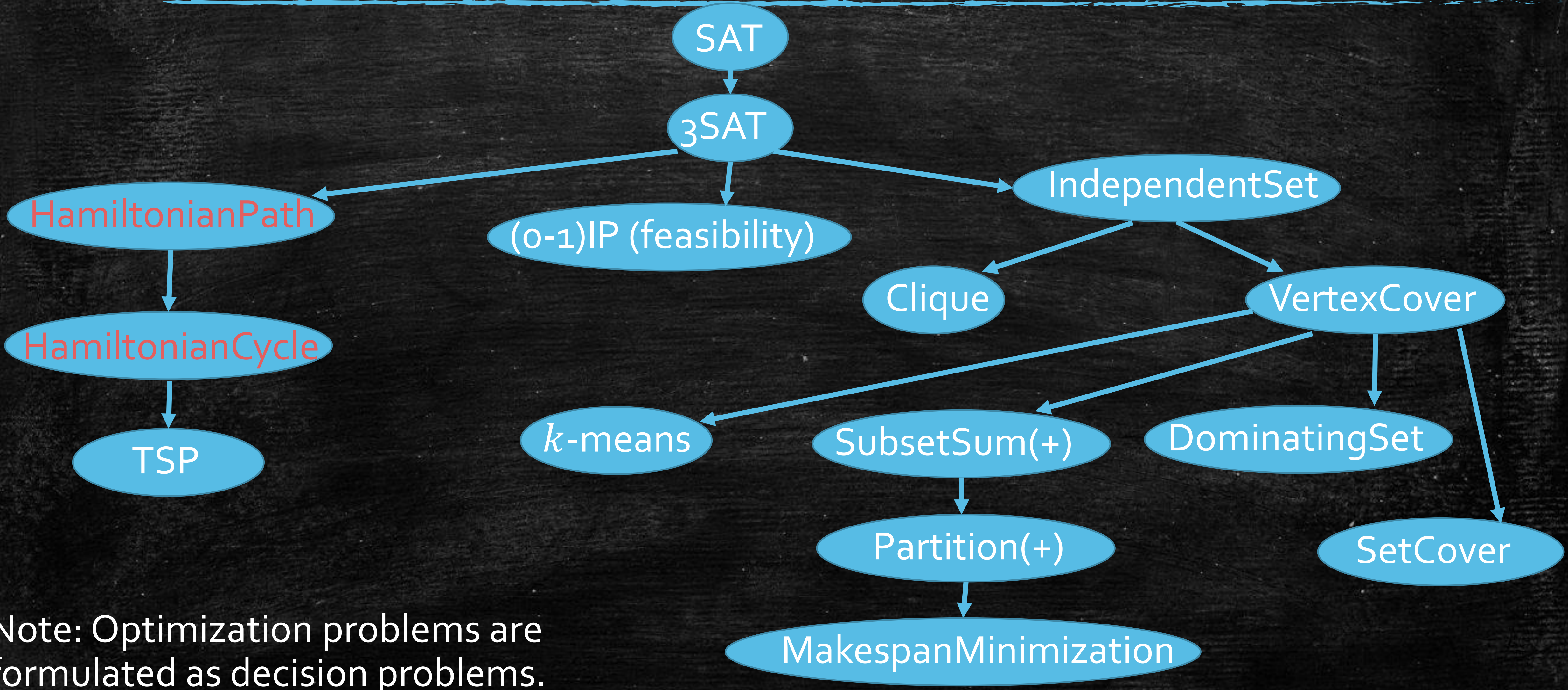
- See which kind of problems it belongs to…

If you see a lot of numbers…



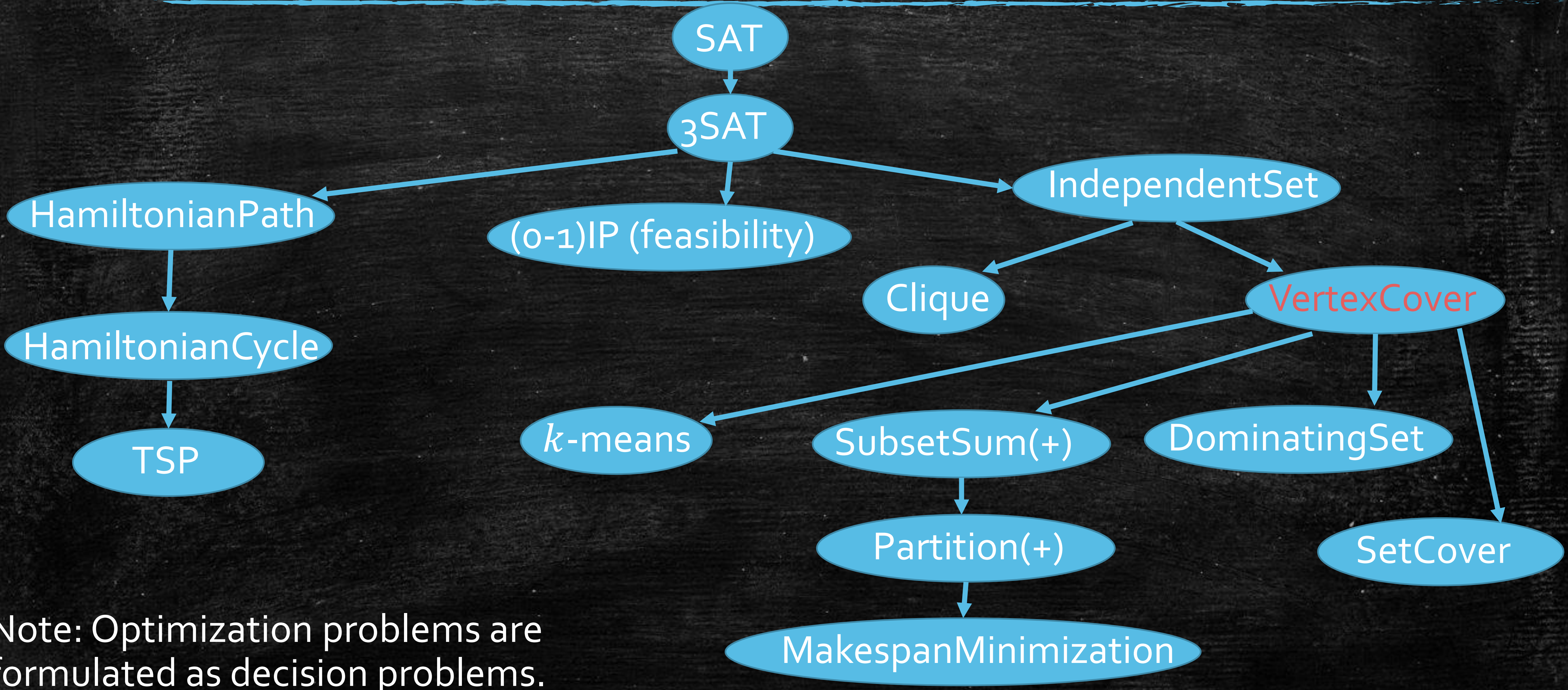
Note: Optimization problems are formulated as decision problems.

If the problem is related to path or tree…



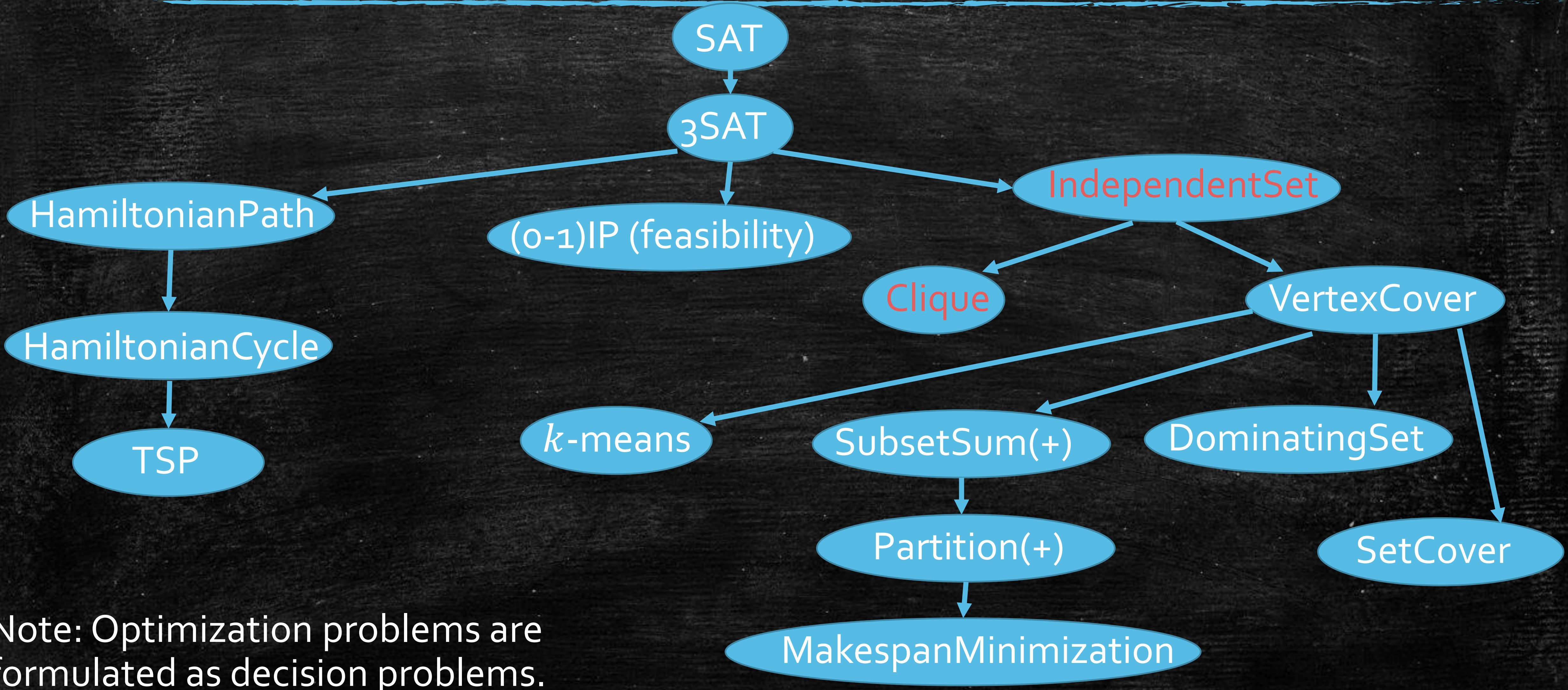
Note: Optimization problems are formulated as decision problems.

If it is some kind of coverage problem…



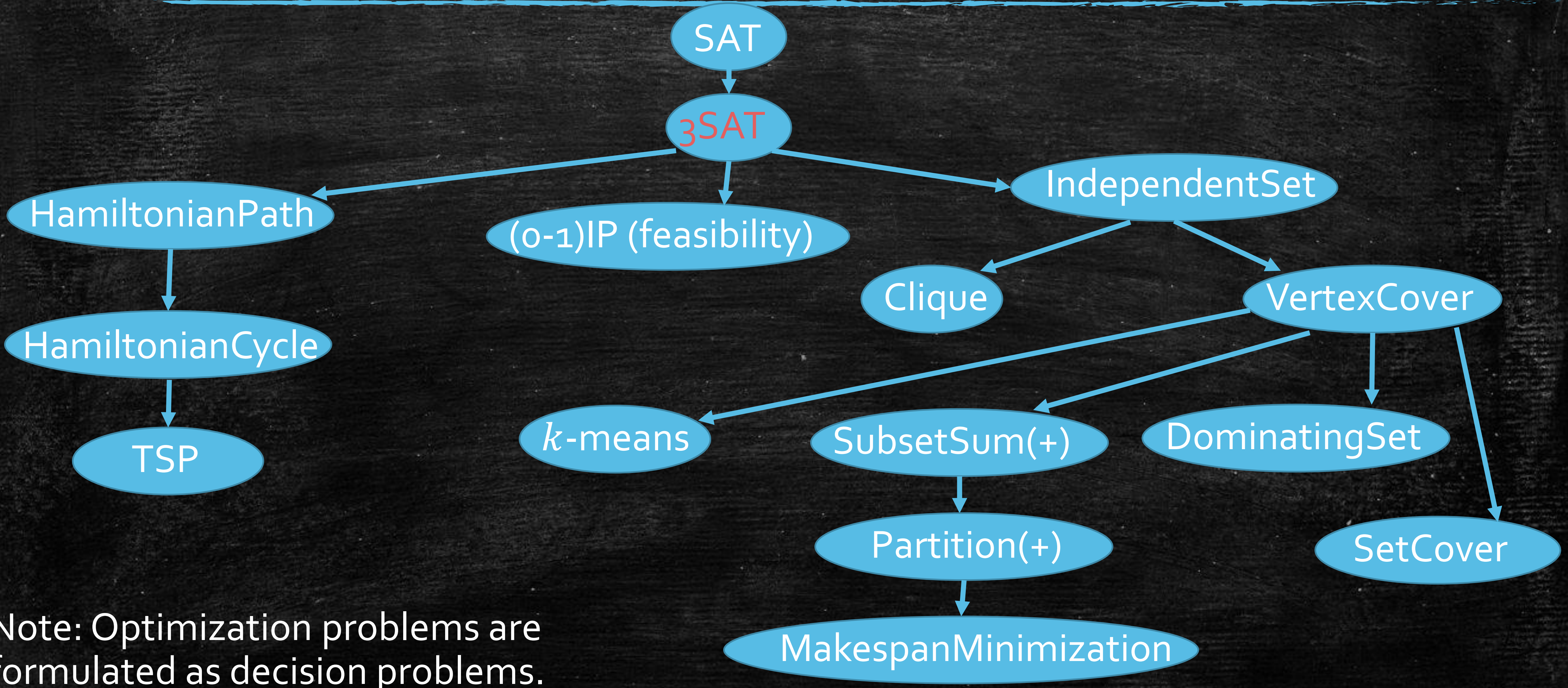
Note: Optimization problems are formulated as decision problems.

Problems about relations between vertices or subgraph structures…



Note: Optimization problems are formulated as decision problems.

Other problems, problems with a number 3...



Note: Optimization problems are formulated as decision problems.

A Common Mistake

Mess up the direction or confuse about special-general case

- 3SAT $\leq_k$ SAT is trivial
- SAT $\leq_k$ 3SAT requires techniques like clause-breaking

A Common Mistake

Mess up the direction or confuse about special-general case

Assignment 6 Problem 2(b)

- Half-Clique: Decide if an undirected graph $G = (V, E)$ contains a $\frac{n}{2}$-clique.
- Clique: Instance $(G = (V, E), k \in \mathbb{Z}^+)$, decide if G contains a k-clique.
- Half-Clique $\leq_k$ Clique is trivial
- Clique $\leq_k$ Half-Clique is not! You need to play some tricks…

A Common Mistake

Mess up the direction or confuse about special-general case

"Prove" that maximum matching is NP-hard by formulating it to an Integer Program...

- Tips for proving $g \leq_k f$: Always start from an arbitrary instance of the problem g... Do not "simultaneously" construct both instances for g and f

Exercise 1

- k-partite Clique: Given a k-partite graph, decide if the graph has a k-clique.

Which of the following is trivial?

- k-partite clique $\leq_k$ clique
- Clique $\leq_k$ k-partite clique

Exercise 2

Which of the following is trivial?

- Partition(+) $\leq_k$ Partition
- Partition $\leq_k$ Partition(+)

Exercise 3

Which of the following is trivial?

- s-t-HamiltonianPath $\leq_k$ HamiltonianPath
- HamiltonianPath $\leq_k$ s-t-HamiltonianPath

Reduction to "Special Cases"

- The reduction $g \le_k f$ do not need to be onto.
- It's OK to reduce g to special cases of f.
- Suppose h is a special case of f.
  - $h \le_k f$ holds trivially.
- It suffices to prove $g \le_k h$.
- Example: Partition(+) $\le_k$ MakespanMinimization

Reduction for Algorithm Design

- Suppose we want to solve g.
- If we can solve f and we can show $g \leq_k f$, we have an algorithm for g.
- We have already seen many examples:
  - Midterm Q2
  - LP $\Longrightarrow$ standard-form LP
  - Min-cut, Matching $\Longrightarrow$ Max-Flow
  - Bipartite Independent Set, Vertex Cover $\Longrightarrow$ Min-Cut
  - Assignment 5 Q1 $\Longrightarrow$ Matching

Approximation Algorithm

- Vertex Cover (2-approximation)
- Metric TSP (1.5-approximation)
- Max-3SAT (7/8-approximation)
- Set Cover ($\ln n$-approximation)
- Max-k-Coverage ($\left(1-\frac{1}{e}\right)$-approximation)
- Max-Cut (0.5-approximation)

Common Techniques for Approximation Algorithms

- Greedy (makespan minimization, set cover, max-k-coverage)
- Local Search (max-cut, Assignment 6 Q1)
- LP-relaxation (vertex cover, metric facility location)
- Conditional Expectation and Derandomization (max-3SAT)
- And many more… (vertex cover, metric TSP, etc.)

Analyzing Approximation Ratio

- Understand problem’s nature!

Good Luck to Your Final Exam!