Edmonds-Karp Algorithm

```
Edmonds-Karp Algorith(G, s, t, c)
Initialize f such that for all e in E, f(e) = 0, initialize G^f = G
while there is an s-t path on G^f
        find such a path p by BFS
        find an edge e in p with minimum capacity b
        update f that pushes b units of flow along p
        update G^f
endwhile
return f
```

O(|V||E|^2)

Dinic's Algorithm

We now seek a way to improve the Edmonds-Karp Algorithm. Note that $\mathrm{dist}[u]$ is non-decreasing but not strictly increasing. If $\mathrm{dist}[t]$ is strictly increasing, then the time complexity can be further optimized to $O(|V|^2|E|)$. To achieve this, we should remove all shortest $s \to t$ path but not only one.

We build a *level* graph, where vertices at level i are at distance i. We only keep the edges from a level to the next level. It can be done in $O(|E|)$ time using a similar idea to BFS.

We intend to find a blocking flow on the level graph. We push flow on multiple $s - t$ paths where each $s - t$ path must contain a critical ledge. To find such a blocking flow, we run a frontward DFS. There are two possibilities

- End up at t, in this case, we update f and remove the critical edge
- End up at a dead-end, a vertex v with no out-going edges in G_L^f: in this case, we remove all the incoming edges of v.

Since at least one edge is removed after each search, and each search takes at most $|V|$ steps, so the time complexity for each iteration is $O(|V||E|)$. For the next part, we need to prove the distance monotonicity. All the original shortest paths are now unavailable (by blocking), so we consider the potential new paths. It might contain a backward edge whose reverse was a critical edge in the previous iteration. In this case, $\mathrm{dist}(t)$ is increased by at least 2. Or it may be an edge in G^{f_i} but not in $G_L^{f_i}$. In this case, $\mathrm{dist}(t)$ is increased by at least 1.

Therefore, we have proved that the total number of iterations is $O(|V|)$.

Hopcroft-Karp-Karzanov Algorithm

It is a special case of Dinic's Algorithm, and can find a maximum bipartite matching in $O(|E| \cdot \sqrt{|V|})$. To prove this, we consider the following two steps.

- Finding a blocking flow in a level graph takes $O(|E|)$ time.

  A key difference is that for each $s - t$ path, all edges along the path must be deleted, in contrast with the previous case where it is possible for only one edge along the path to be deleted. The backtracing procedure also visits an edge only once. So the time complexity is $O(|V| + |E|) \approx O(|E|)$.

- Number of iterations is at most $2\sqrt{|V|}$.

  If the algorithm terminates within $\sqrt{|V|}$ iterations,we are already done. If not, let f be the flow after $\sqrt{|V|}$ iterations. We first claim that $f_{\max} - f \leq \sqrt{V}$, which is equivalent to the claim that the maximum flow of G^f has value at most $\sqrt{V}$. If the claim is true, then by the result of Ford-Fulkerson, the algorithm will terminate within another $\sqrt{|V|}$ iterations (each time 1 is added).

  To prove the claim, we have the following statement: *in each iteration, for each $v \in V \backslash \{s, t\}$, either its in-degree, or its outdegree is 1*. The statement apparently holds for G, and for G^f we can verify it relatively easily. With this statement, we further have the proposition that *there exists a maximum integral flow f' in G^f*. Actually, f' consists of **vertex-disjoint** paths with flow 1. (Proof by contradiction, if not then violates flow conservation)

  So we simply needs to prove that there are at most $\sqrt{|V|}$ vertex-disjoint paths with flow 1 after $\sqrt{|V|}$ iterations. By Dinic Algorithm, at this point $\text{dist}(t) > \sqrt{|V|}$. Therefore, the number of vertex-disjoint path is at most $\frac{|V|}{\sqrt{|V|}} = \sqrt{|V|}$. Hence, we have proved that the number of iterations is at most $2\sqrt{|V|}$

LP Duality and Relaxation

The dual problem for standard form is given by

$$\begin{aligned} &\min \mathbf{b}^T\mathbf{y} \\ \text{subject to } &\begin{cases} \mathbf{y}^T\mathbf{A} \geq \mathbf{c}^T \\ \mathbf{y} \geq 0 \end{cases} \end{aligned}$$

Theorem [Weak Duality Theorem]. If $\hat{x}$ is a feasible solution and $\hat{y}$ is a feasible solution, then $c^T\hat{x} \leq b^T\hat{y}$.

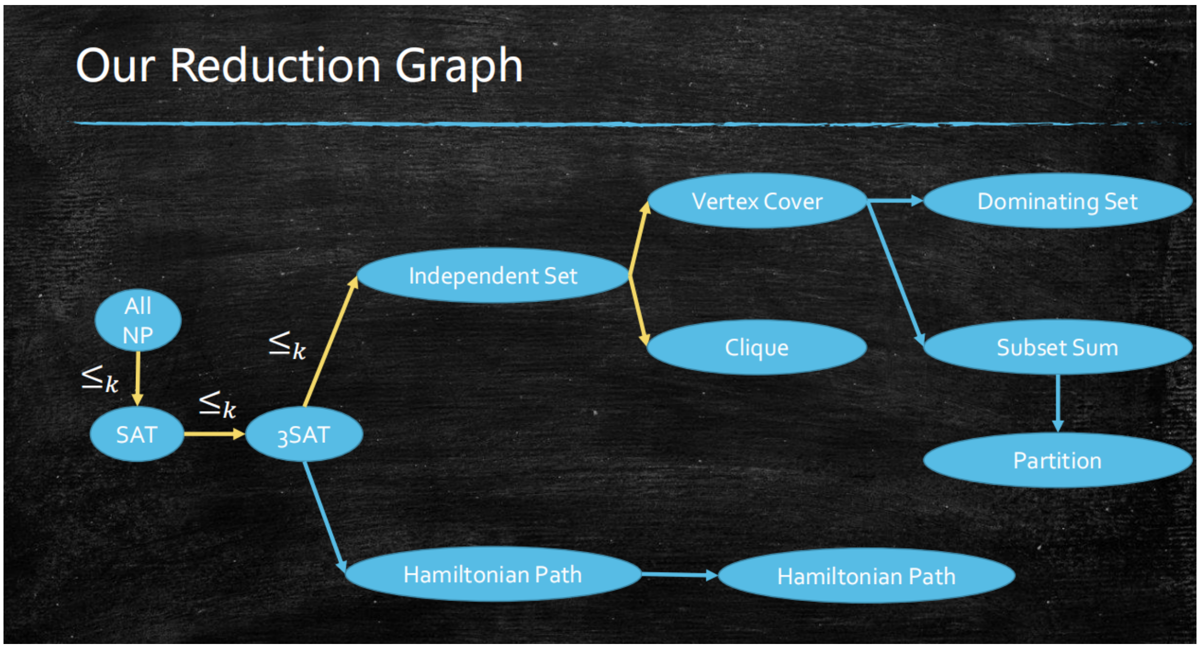
If we require each variable in a linear program is an integer, we obtain an integer problem. We can see that many problem can be formulated as IP, with standard form

$$\begin{aligned} &\max \mathbf{c}^T\mathbf{x} \\ \text{subject to } &\begin{cases} A\mathbf{x} \leq \mathbf{b} \\ \mathbf{x} \geq 0 \\ \mathbf{x} \in \mathbb{Z}^n \end{cases} \end{aligned}$$

Unfortunately, integer programming is NP-complete, even for the zero-one special case $\forall i,\, x_i \in \{0, 1\}$. We will use an example of **vertex cover** to illustrate the problem. Given an undirected graph $G = (V, E)$, a subset of vertices $S \subseteq V$ is a vertex cover if S contains at least one endpoint of every vertex. It can be formulated as follows

$$\begin{aligned} &\min \sum_{v \in V} x_v \\ \text{subject to } &\begin{cases} x_u + x_v \geq 1, & \forall (u, v) \in E \\ x_v \in \{0, 1\} \end{cases} \end{aligned}$$

We now consider a relaxation from $x_v \in \{0, 1\}$ to $0 \leq x_v \leq 1$. Then we introduce the method of LP Rounding. Specifically, we have $x_v^* = \text{sgn}(x_v - \frac{1}{2})$, and the correctness is relatively straightforward.



We then consider another example, i.e. try to reduce 3SAT problem to Independent Set.

Proof: We need to transfer the Boolean expression ϕ to a graph G and a parameter k. For each clause, construct a triangle where three vertices represent three literals. Connect two vertices if one represents the negation of the other We set k to be the number of clauses. If $f(\phi) = 1$, then for each triangle in G, we can pick exactly one vertex representing a true literal in S. Then we can verify easily that S is an independent set and $|S| = k$.

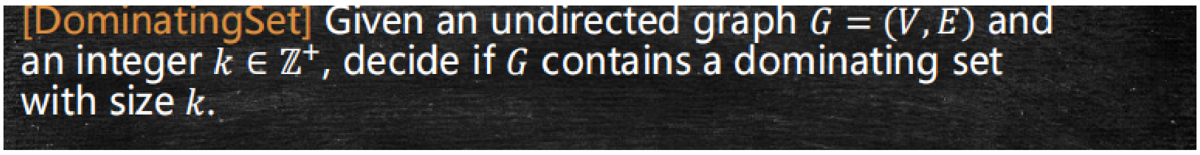
Then we consider the other case. Suppose if G has an independent set of size k, then we can simply assign true to the literals representing the chosen vertices. Note that we will not assign both true and false to the same literal, as x_i and $\neg x_i$ is connected. For those that are not assigned values, we can assign values to them arbitrarily.

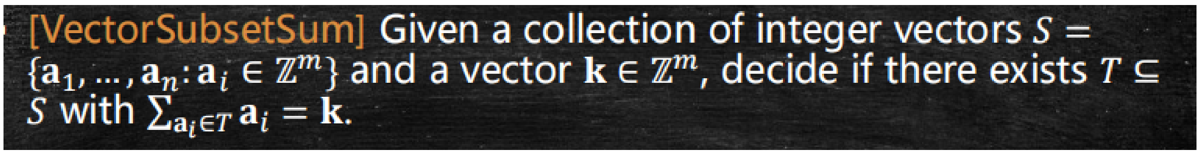
Dominating Set Problem

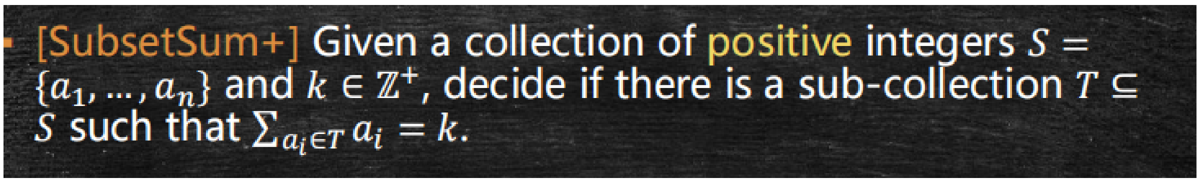
Given an undirected graph $G = (V, E)$, a **dominating set** is a subset of vertices S such that for any $v \in V \backslash S$, there is a vertex $u \in S$ that is adjacent to v. Now we try to prove that the dominating set problem is NP-complete.

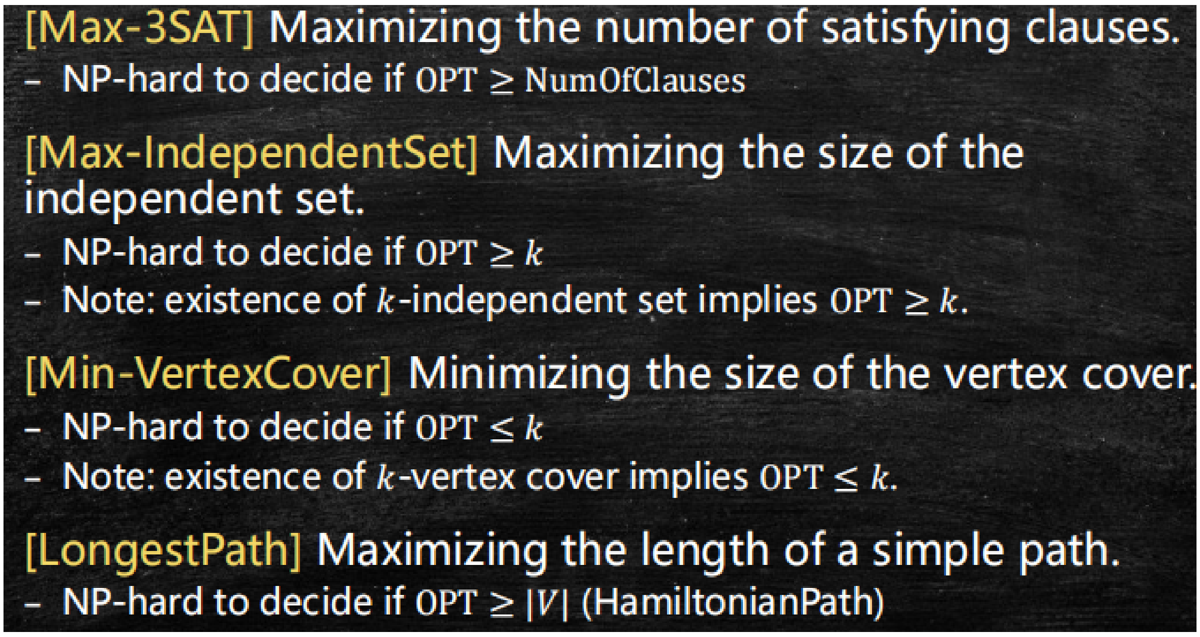
Subset Sum

> Given a collection of integers $S = \{a_1, \ldots, a_n\}$ and $k \in \mathbb{Z}$. Determine whether we can find a subset $A \subseteq S$ such that $\sum A = k$.


[DominatingSet] Given an undirected graph $G = (V, E)$ and an integer $k \in \mathbb{Z}^+$, decide if G contains a dominating set with size k.


[VectorSubsetSum] Given a collection of integer vectors $S = \{\mathbf{a}_1, \ldots, \mathbf{a}_n : \mathbf{a}_i \in \mathbb{Z}^m\}$ and a vector $\mathbf{k} \in \mathbb{Z}^m$, decide if there exists $T \subseteq S$ with $\sum_{\mathbf{a}_i \in T} \mathbf{a}_i = \mathbf{k}$.


[SubsetSum+] Given a collection of positive integers $S = \{a_1, \ldots, a_n\}$ and $k \in \mathbb{Z}^+$, decide if there is a sub-collection $T \subseteq S$ such that $\sum_{a_i \in T} a_i = k$.



[Max-3SAT] Maximizing the number of satisfying clauses.

- NP-hard to decide if $\mathrm{OPT} \geq \mathrm{NumOfClauses}$

[Max-IndependentSet] Maximizing the size of the independent set.

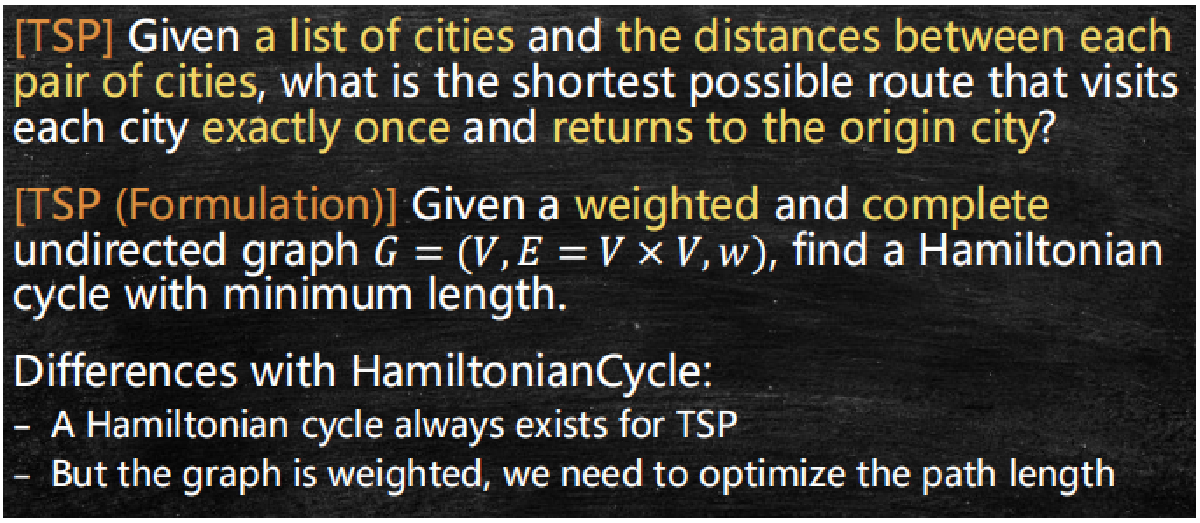
- NP-hard to decide if $\mathrm{OPT} \geq k$
- Note: existence of k-independent set implies $\mathrm{OPT} \geq k$.

[Min-VertexCover] Minimizing the size of the vertex cover.

- NP-hard to decide if $\mathrm{OPT} \leq k$
- Note: existence of k-vertex cover implies $\mathrm{OPT} \leq k$.

[LongestPath] Maximizing the length of a simple path.

- NP-hard to decide if $\mathrm{OPT} \geq |V|$ (HamiltonianPath)



[TSP] Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city?

[TSP (Formulation)] Given a weighted and complete undirected graph $G = (V, E = V \times V, w)$, find a Hamiltonian cycle with minimum length.

Differences with HamiltonianCycle:

- A Hamiltonian cycle always exists for TSP
- But the graph is weighted, we need to optimize the path length