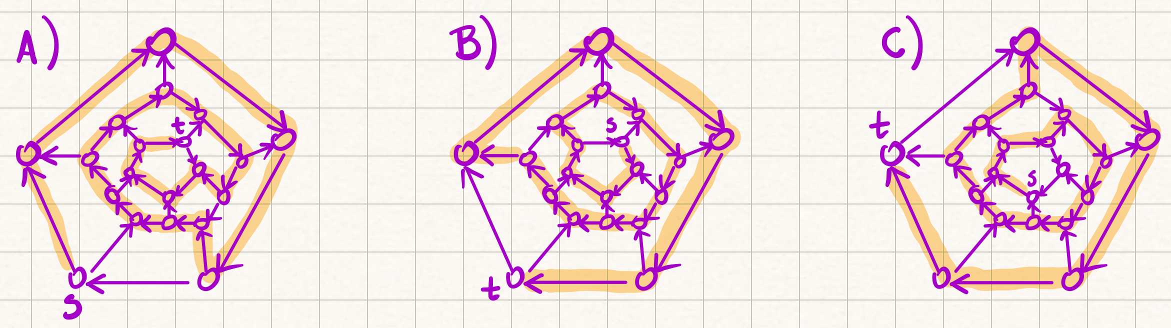
Lecture 7

The Hamiltonian path problem asks whether the input directed graph has a path from s to t that goes through every node exactly once.

HAMPATH = { <G, s, t> | G is a directed graph with a Hamiltonian path from s to t }

Quiz 7.1: Which of the following paths is not a Hamiltonian path?



[Theorem 7.46: HAMPATH is NP-complete.

Proof: To show that HAMPATH is NP-complete, we first have to show that HAMPATH ∈ NP and then find an NP-complete problem Z and construct a Karp reduction so that $Z \leq_p$ HAMPATH.

① HAMPATH ∈ NP

A nondeterministic polynomial time TM can "guess" a path from s to t and then check in polynomial time if ① there are no repetitions in the path, and ② every vertex is visited in the path. If both conditions hold then the NTM accepts, otherwise it rejects.

② Show that 3SAT $\leq_p$ HAMPATH

The Karp reduction should take as an input ANY 3CNF formula ϕ (that may or may not be satisfiable) and construct a directed graph $G=(V,E)$ so that for $s,t \in V$:

ϕ is satisfiable if and only if there exists a Hamiltonian path from s to t in G.

$\phi \in$ 3SAT $\iff$ $f(\phi) \in$ HAMPATH

or $\phi \in$ 3SAT $\iff$ $(G, s, t) \in$ HAMPATH

* f is the function that computes the Karp reduction. Input is ϕ and output is (G, s, t).

this can be ANY 3CNF formula. No control over the input

This directed graph G is constructed by us! We have control over which graph we generate through the reduction (also we choose s and t).

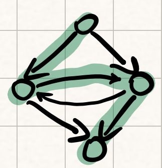
$$\phi \overset{\Delta}{=} (C_1[1] \vee \underbrace{C_1[2]}_{\substack{\text{second position}\\ \text{of first clause}}} \vee C_1[3]) \wedge \quad \dots \quad \wedge (C_k[1] \vee C_k[2] \vee C_k[3])$$

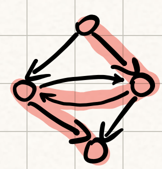
defined

- k clauses
- each clause has 3 literals
- ℓ variables, $x_1, \dots, x_\ell$

- The main "knob" in ϕ is the truth assignment of a variable x_i.
- We have to somehow signal whether $x_i = \text{TRUE}$ or $x_i = \text{FALSE}$ through a path on a directed graph

> The direction of a subgraph traversal can act as a signal about whether $x_i = \text{TRUE}$ or $x_i = \text{FALSE}$

x_i  left-to-right traversal of subgraph $\rightsquigarrow x_i = \text{TRUE}$

x_i  right-to-left traversal of subgraph $\rightsquigarrow x_i = \text{FALSE}$

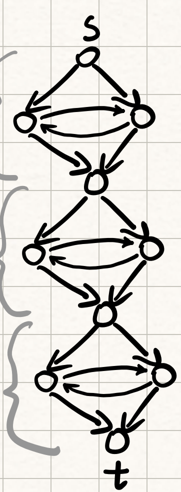
- Suppose we have the following concrete input $\phi' = \underbrace{(x_1 \vee x_2 \vee x_3)}_{C_1} \wedge \underbrace{(\bar{x}_1 \vee x_2 \vee x_3)}_{C_2} \wedge \underbrace{(\bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_3)}_{C_3}$

- Let's see how we can construct a (G, s, t) output for Karp reduction by combining a subgraph per variable of ϕ'.

x_1

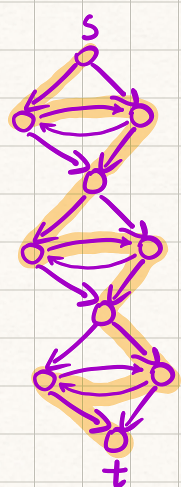
x_2

x_3

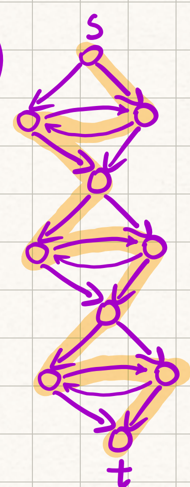


Quiz 7.2: Which of the following Hamiltonian paths does not give a satisfying truth assignment to $\phi' = (x_1 \vee x_2 \vee x_3) \wedge (\bar{x}_1 \vee x_2 \vee x_3) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_3)$

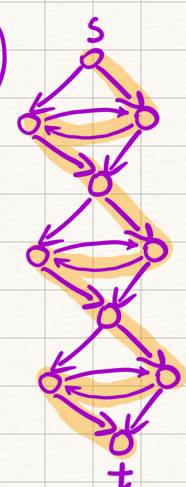
A)



B)



C)



What is the problem with the above approach?

Even though we can translate a satisfying truth assignment to a Hamiltonian path, we cannot translate every Hamiltonian path to a satisfying truth assignment. That is:

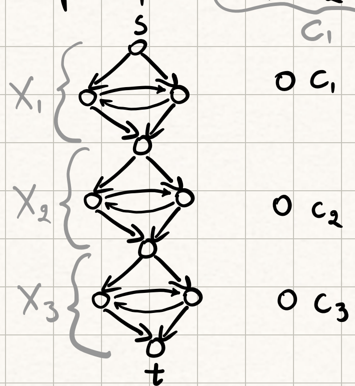
$\phi \in 3SAT \Rightarrow (G, s, t) \in HAMPATH$ ✓ $\qquad \phi \in 3SAT \Leftarrow (G, s, t) \in HAMPATH$ ✗

The main problem is that we are not keeping track of whether every clause is satisfied by the chosen Hamiltonian path. In fact we have not encoded clauses at all in the (G, s, t) we have constructed so far.

Somehow we have to keep track of

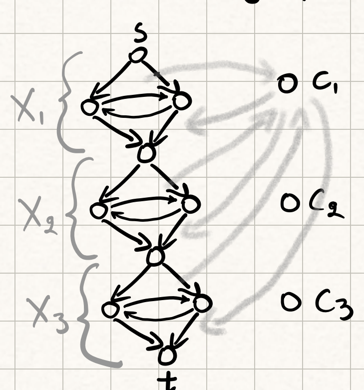
1. How many clauses do we have
   ↳ Insert one vertex per clause
2. Which variables are associated with each clause
   ↳ Connect at most three subgraphs/variables with each clause
3. which truth assignment of a variable would satisfy each clause
   ↳ Pick the appropriate direction for the edges that connect a subgraph with a clause

Resolve ① in our running example $\phi' = \underbrace{(x_1 \vee x_2 \vee x_3)}_{C_1} \wedge \underbrace{(\bar{x}_1 \vee x_2 \vee x_3)}_{C_2} \wedge \underbrace{(\bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_3)}_{C_3}$



Resolve ② in our running example $\phi' = \underbrace{(x_1 \vee x_2 \vee x_3)}_{C_1} \wedge \underbrace{(\bar{x}_1 \vee x_2 \vee x_3)}_{C_2} \wedge \underbrace{(\bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_3)}_{C_3}$

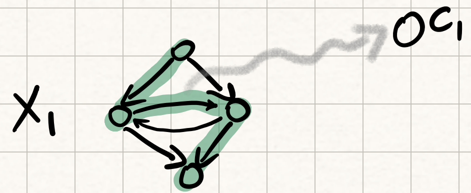
Notice that C_1 is associated with variables x_1, x_2, and x_3. Thus, we need to add edges from subgraphs of x_1, x_2, and x_3 to C_1.

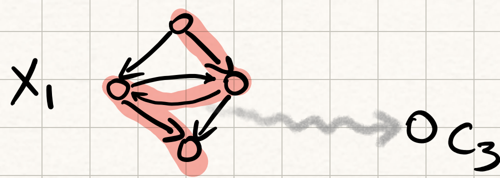


* Let's not commit on the specifics of these edges just yet.

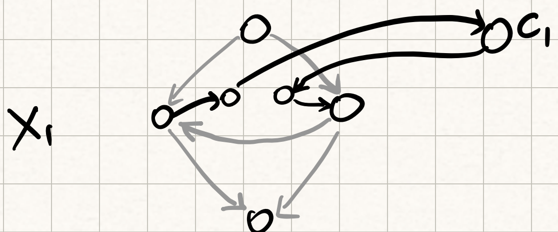
Resolve ③ in our running example $\phi' = \underbrace{(x_1 \vee x_2 \vee x_3)}_{C_1} \wedge \underbrace{(\bar{x}_1 \vee x_2 \vee x_3)}_{C_2} \wedge \underbrace{(\bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_3)}_{C_3}$

For example, we want to make sure that x_1 subgraph can access c_1 vertex during a left-to-right traversal (because c_1 has the literal x_1). On the other hand, we want to make sure that x_1 subgraph can access c_3 vertex during a right-to-left traversal (because c_3 has the literal $\bar{x}_1$).

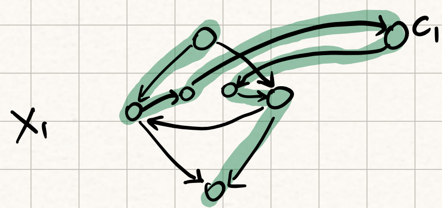




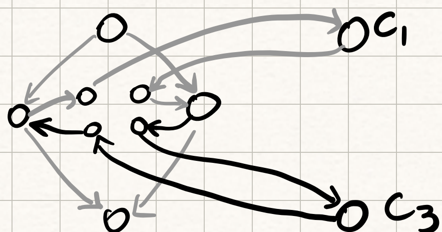
(Failed) Attempt-1: Remove the "horizontal" edge and add two vertices that can go to c_1 and return back



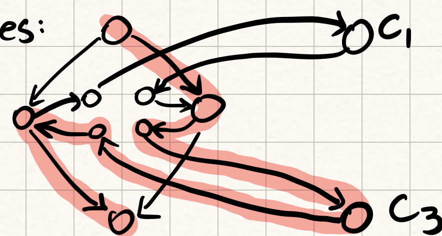
The X_1=TRUE becomes:



Similarly, change the horizontal edge so that the path can reach c_3 and return back

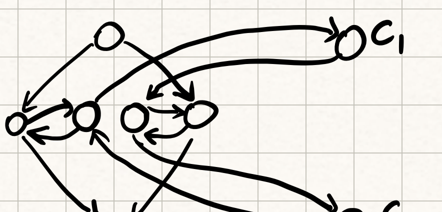


The X_1=FALSE becomes:

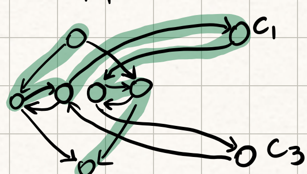


⚠ The problem now is that we added two vertices for the left-to-right and two vertices for the right-to-left. Thus, regardless of which way we choose, the resulting path is not a Hamiltonian path because we do not pass through at least two vertices.

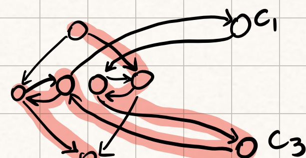
(Correct) Attempt-2: Instead of four new vertices, add two that are "shared".



For X_1=TRUE



For X_1= FALSE

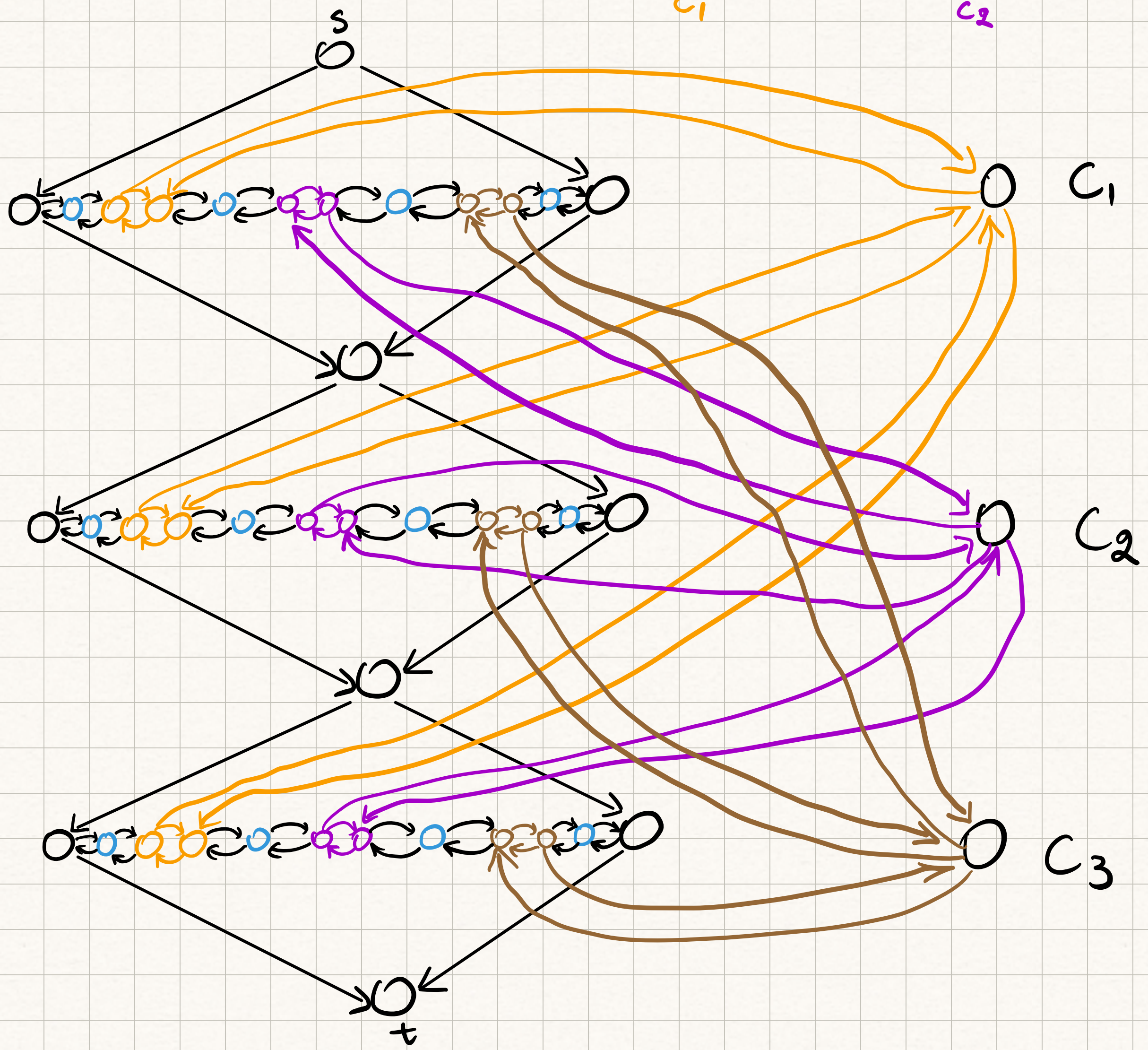


The resulting paths go through all vertices of subgraph X_1.

- Thus, we need two extra vertices for each clause at each subgraph

* We will also add separator vertices in-between.

The end result for input $\phi = \underbrace{(x_1 \vee x_2 \vee x_3)}_{c_1} \wedge \underbrace{(\bar{x}_1 \vee x_2 \vee x_3)}_{c_2} \wedge \underbrace{(\bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_3)}_{c_3}$ is:



A more formal treatment:

↳ The reduction should work for ALL possible inputs

Given any arbitrary 3-SAT input, with n Boolean variables and m clauses, the TM that computes the Karp reduction function outputs a graph $G=(V,E)$ and the nodes $s,t \in V$.

① Describe the reduction

② Show that it takes polynomial time

③ Show that $\phi \in 3SAT \Rightarrow f(\phi) \in HAMPATH$

Suppose ϕ is satisfiable, for each variable $x_1, \ldots, x_\ell$ traverse "horizontal" nodes from left-to-right if x_i = TRUE according to the satisfying truth assignment, and right-to-left if x_i = FALSE according to the satisfying truth assignment. For each clause $c_1, \ldots, c_k$ choose one literal that is true (it is possible to have more than one true literals) and detour to pass from c_j.

④ Show that $f(\phi) \in HAMPATH \Rightarrow \phi \in 3SAT$

Suppose G has a Hamiltonian path from s to t. We need to translate this Hamiltonian path in G to a satisfying assignment in ϕ.

Case Analysis:

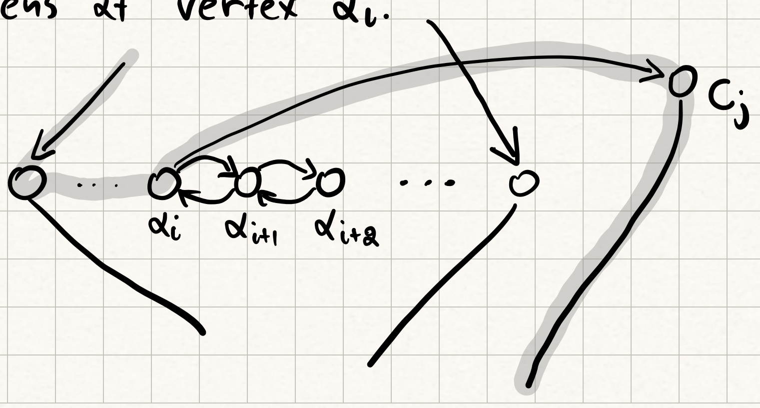
Case-A) The Hamiltonian path passes through each subgraph $x_1, \ldots, x_\ell$ in order,

or

Case-B) The Hamiltonian path jumps between subgraphs $x_1, \ldots, x_\ell$.

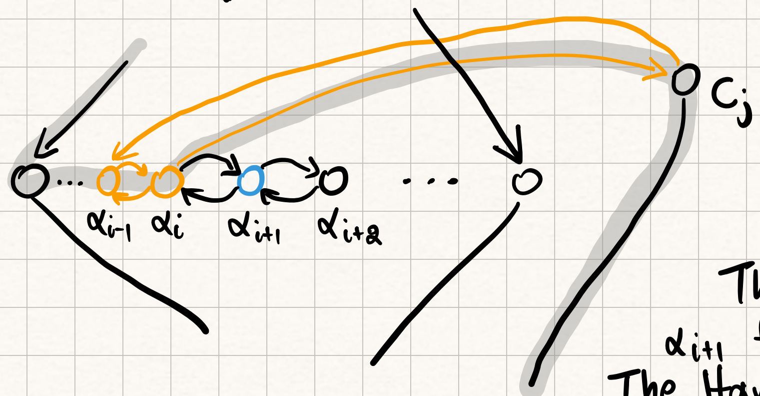
We will prove that Case-B is impossible

For the sake of contradiction, suppose that the Hamiltonian path jumps between subgraphs and that the first (from top-to-bottom) jump happens at vertex α_i.

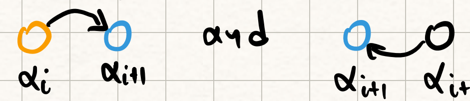


- Observe that either α_{i+1} or α_{i+2} is a separator.

Case B-1 Vertex α_{i+1} is a separator if α_{i-1} and α_i concern clause c_j



If α_{i+1} is a separator, then the only edges entering α_{i+1} are

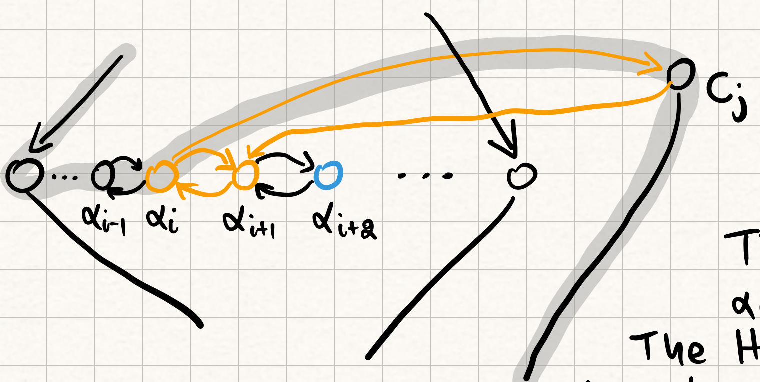


The Hamiltonian path cannot enter α_{i+1} from α_i because α_i is already traversed.
The Hamiltonian path cannot enter α_{i+1} from α_{i+2} because if it does, the path would be stuck in α_{i+1} since both of α_{i+1} neighbors are visited

$\Downarrow$

Cannot terminate in t $\Rightarrow$ Contradiction

Case B-2 Vertex α_{i+2} is a separator if α_i and α_{i+1} concern clause c_j



If α_{i+2} is a separator, then the only edge entering α_{i+1} are

The Hamiltonian path cannot enter α_{i+1} from α_i or c_j because they are already traversed.
The Hamiltonian path cannot enter α_{i+1} from α_{i+2} because if it does, the path would be stuck in α_{i+1} since both of α_{i+1} neighbors are visited

$\Downarrow$

Cannot terminate in t $\Rightarrow$ Contradiction

$\blacksquare$

Recall:
A graph $G=(V,E)$ has an independent set of size k, if there exists a set S of size k vertices such that for any pair of vertices $u, v \in S$, there is no edge (u,v).

We define the language for the problem Subset Sum

$$\text{SUBSET-SUM} = \left\{ (\alpha_1, \alpha_2, \ldots, \alpha_n, t) \;\middle|\; \begin{array}{l} \text{All } \alpha_i \text{ are positive integers and there exists} \\ \text{a subset of } \alpha_i \text{ integers that sums to } t \end{array} \right\}$$

Not in Sipser

Theorem: SUBSET-SUM is NP-complete

Proof: To show that SUBSET-SUM is NP-complete, we first have to show that SUBSET-SUM $\in$ NP and then find an NP-complete problem Z and construct a Karp reduction so that $Z \leq_p$ SUBSET-SUM.

① SUBSET-SUM $\in$ NP

A nondeterministic polynomial time TM can "guess" a subset of numbers from $\alpha_1, \dots, \alpha_n$ and checks in polynomial time if this subset sums to t. If it does then the NTM accepts, otherwise it rejects.

② Show that IND $\leq_p$ SUBSET

The Karp reduction should take as an input ANY input (G,k) for IND (that may or may not have an ind. set) and construct a set of numbers $\{\alpha_1, \dots, \alpha_n\}$ and a target t such that:

G admits an ind. set of size k if and only if there is a subset in $\{\alpha_1, \dots, \alpha_n\}$ that sums to t

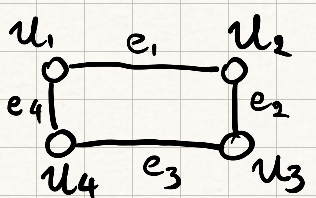
$(G,k) \in \text{IND} \iff f((G,k)) \in \text{SUBSET-SUM}$

or $(G,k) \in \text{IND} \iff (\alpha_1, \dots, \alpha_n, t) \in \text{SUBSET-SUM}$

* f is the function that computes the Karp reduction.

We have to encode the graph structure of G with a set of numbers. Let's see a concrete example

G is



What if we define a number with $|E|+1$ digits for each vertex. The $(i+1)$-th digit of u_j is 1 if the i-th edge has u_j as an endpoint, otherwise it is 0.

	e_1	e_2	e_3	e_4
Vertex $u_1 \rightsquigarrow$ integer $\alpha_1 = 1$	1	0	0	1
Vertex $u_2 \rightsquigarrow$ integer $\alpha_2 = 1$	1	1	0	0
Vertex $u_3 \rightsquigarrow$ integer $\alpha_3 = 1$	0	1	1	0
Vertex $u_4 \rightsquigarrow$ integer $\alpha_4 = 1$	0	0	1	1

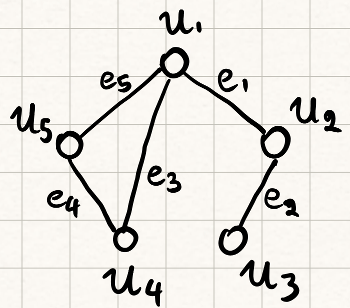
Let's see now the sum of independent sets

$S = \{u_1, u_3\} \rightsquigarrow \alpha_1 + \alpha_3 = 21111$

$S = \{u_2, u_4\} \rightsquigarrow \alpha_2 + \alpha_4 = 21111$

If we pick $t = 21111$ then for this example the independent sets correspond to subsets that sum to t.

Let's see another graph example



$e_1\ e_2\ e_3\ e_4\ e_5$

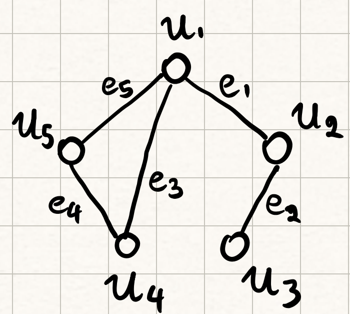
Vertex $u_1 \rightsquigarrow$ integer $\alpha_1 = 110101$
Vertex $u_2 \rightsquigarrow$ integer $\alpha_2 = 111000$
Vertex $u_3 \rightsquigarrow$ integer $\alpha_3 = 101000$
Vertex $u_4 \rightsquigarrow$ integer $\alpha_4 = 100110$
Vertex $u_5 \rightsquigarrow$ integer $\alpha_5 = 100011$

Let's see now the sum of independent sets

$S = \{u_1, u_3\} \rightsquigarrow \alpha_1 + \alpha_3 = 211101$
$S = \{u_2, u_4\} \rightsquigarrow \alpha_2 + \alpha_4 = 211110$

⚠ There is no single t value that works for all independent sets of size two.

💡 Patch the discrepancy by introducing one new number per edge.



$e_1\ e_2\ e_3\ e_4\ e_5$

Vertex $u_1 \rightsquigarrow$ integer $\alpha_1 = 110101$
Vertex $u_2 \rightsquigarrow$ integer $\alpha_2 = 111000$
Vertex $u_3 \rightsquigarrow$ integer $\alpha_3 = 101000$
Vertex $u_4 \rightsquigarrow$ integer $\alpha_4 = 100110$
Vertex $u_5 \rightsquigarrow$ integer $\alpha_5 = 100011$

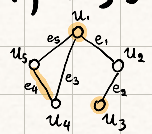
$e_1\ e_2\ e_3\ e_4\ e_5$

Edge $e_1 \rightsquigarrow$ integer $b_1 = 010000$
Edge $e_2 \rightsquigarrow$ integer $b_2 = 001000$
Edge $e_3 \rightsquigarrow$ integer $b_3 = 000100$
Edge $e_4 \rightsquigarrow$ integer $b_4 = 000010$
Edge $e_5 \rightsquigarrow$ integer $b_5 = 000001$

* The idea is that the inclusion of a vertex will flip the digit of the incident edges (by adding α_i to the sum). The rest of the edges will be added manually. In the end all edges must have digit 1 in their corresponding position.

Let's revisit the independent sets

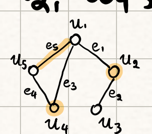
- $S = \{u_1, u_3\} \rightsquigarrow \alpha_1 + \alpha_3 + b_4 = 211111$



edges e_5 e_1, e_3 are incident on u_1 | edge e_2 is incident on u_3 | manually add the last remaining edge

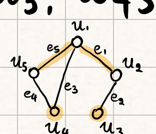
$$\begin{array}{rl} & \alpha_1 = 110101 \\ & \alpha_3 = 101000 \\ + & b_4 = 000010 \\ \hline & 211111 \end{array}$$

- $S = \{u_2, u_4\} \rightsquigarrow \alpha_2 + \alpha_4 + b_5 = 211111$



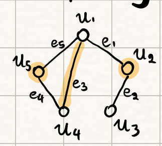
edges e_1 e_2 are incident on u_2 | edges e_3, e_4 is incident on u_4 | manually add the last remaining edge

- $S = \{u_3, u_4\} \rightsquigarrow \alpha_3 + \alpha_4 + b_1 + b_5 = 211111$



edge e_2 is incident on u_3 | edges e_3, e_4 is incident on u_4 | manually add the last remaining edges

•) $S=\{u_2, u_5\} \rightsquigarrow \alpha_2+\alpha_5+b_3 = 211111$



edges e_1 e_2 are incident on u_2 — edges e_4, e_5 is incident on u_5 — manually add the last remaining edge

More formally:

① Describe the reduction

Given an undirected graph $G=(V,E)$ with vertex set $V=\{u_1,u_2,\dots,u_n\}$, edge set $E=\{e_1,\dots,e_m\}$, and a target size k for the independent set, the reduction algorithm constructs $n+m+1$ positive integers that define an instance of the subset sum problem.

Ⓐ For each vertex u_i define the number $\alpha_i = \underbrace{10^m}_{\text{Put 1 at leftmost}} + \underbrace{\sum_{e_j\in A_i} 10^{m-j}}_{\text{Put 1 on each incident edge position}}$, where A_i denotes the set of edges incident to u_i

Ⓑ For each edge e_j, define the number $b_j = 10^{m-j}$

Ⓒ Define the target sum $t = k\cdot 10^m + \sum_{j=1}^{m} 10^{m-j}$

② Polynomial Time

To create each of the α_i numbers, we have iterate through the edges of the corresponding vertex. Since each edge is considered twice (once for each endpoint) the total time for this generation is $O(n+m)$. Generating the b_j numbers takes $O(m)$ time.

③ $(G,k)\in IND \Rightarrow f((G,k)) \in$ SUBSET-SUM

If (G,k) is a member of language IND, then there exists a set of vertices S of size k that is an independent set. By construction, for every vertex in S, there exists an associated number in f's output. Let A' be the set of numbers that correspond to the vertices in S. The fact that there is no edge between any pair of vertices in S means that the sum of A' will not have any number with digit larger than 1 (except the m-th digit). If there was a digit 2 then the collection of vertices S would contain both endpoints of the same edge. Since set S has size k, the sum A' will have digit k in the m-th position (from right-to-left). If we add to sum(A') all the edge-numbers b_j that have no endpoints in S, then we will get a subset of integers that sums to t. Thus, the output of reduction f is a member of the language SUBSET-SUM.

④ $f((G,k)) \in$ SUBSET-SUM $\Rightarrow$ $(G,k) \in$ IND

If $f((G,k))$ is a member of language SUBSET-SUM, then there exists a subset of numbers that sums to $t = k \cdot 10^m + \sum_{j=1}^{m} 10^{m-j}$. Since every digit of t is one (except the m-th), we know that every edge is considered exactly one time; either because the summation includes the corresponding b_j or because the summation includes exactly one of the edge's endpoints. Next, we argue that the vertices associated with the numbers α_i that sum to t, can not share an edge in between the corresponding vertices of the graph. If they did share an edge, say e_j, then the j-th digit of the summation would have been 2, contradiction. Thus, the α_i included in the summation comprise an independent set. Finally, we have exactly k terms from $\{\alpha_1, \alpha_2, \dots, \alpha_n\}$ in the summation because the m-th digit of the summation is k. Therefore, the instance (G,k) is a member of the language IND.

$\blacksquare$