CMSC 451 - Algorithm Design
Lecture 19 - Subset Sum Approximation

Subset Sum (SS):

- Given a set of positive integers $S = \{x_1, \ldots, x_n\}$ + target value t, what is largest sum $\leq t$?
- Objective:

$$\max_{S' \subseteq S} \left\{ \sum_{x \in S'} x \;:\; \sum_{x \in S'} x \leq t \right\}$$

Example: $S = \{3, 6, 9, 12, 15, 23, 32\}$ $t = 34$

Answer: $S' = \{6, 12, 15\}$ $6+12+15 = \underline{33} \leq t$ (can't make 34)

Application:

- A simplified version of 0-1 knapsack and various packing problems.
- SS is NP-complete (won't prove this) (see optional material at end)

This lecture:

Approximation algorithm

PTAS - ε-error ← User specifies ε

- Returns answer within $(1-\varepsilon) \cdot opt$
- Running time - Roughly $\tilde{O}(n^2/\varepsilon)$

Polynomial-Time Approximation to Subset Sum

- We'll present a poly-time algorithm for SS which given ε, where $0 < \varepsilon < 1$, returns a subset $S' \subseteq S$ whose sum z, satisfies

$$(1-\varepsilon)\, z^* \leq z \leq z^* \leq t$$

where z^* is the optimum sum.

Exact (Exponential time) Algorithm

- For $0 \leq i \leq n$, L_i = set of all possible sums of $\{x_1, \ldots, x_i\}$

E.g. $S = \{1, 4, 6, \ldots\}$

$L_0 = \langle 0 \rangle$

$x_1 = 1 \Rightarrow \quad L_1 = \langle 0, 1 \rangle$

$x_2 = 4 \Rightarrow \quad L_2 = \langle 0, 1, 4, 5 \rangle$

$x_3 = 6 \Rightarrow \quad L_3 = \langle 0, 1, 4, 5, 6, 7, 10, 11 \rangle$

- Observe: $|L_i| \leq 2^i$

- Final answer: Largest in L_n that is $\leq t$.

$$z^* = \max \{ z \in L_n \mid z \leq t \}$$

- Approach:
  - Compute L_i for $i = 0, 1, \ldots, n$

    Define: $L + x$ = add x to all $z \in L$

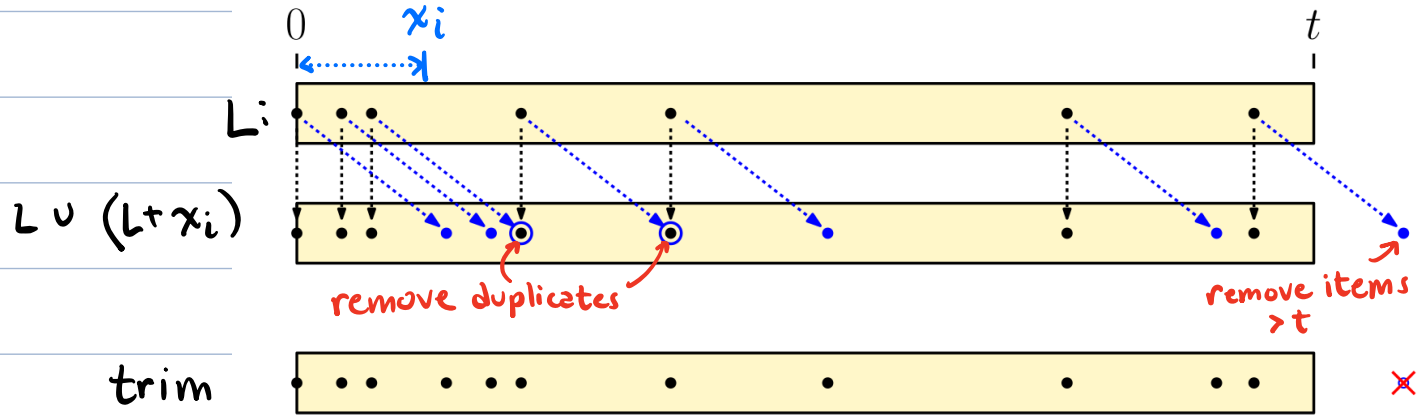
    E.g. $L = \langle 0, 3, 5 \rangle \rightarrow L + 4 = \langle 4, 7, 9 \rangle$

    $L_i \leftarrow L_{i-1} \cup (L_{i-1} + x_i)$ } merge$(L, L + x_i)$

  - Remove duplicates
  - Remove items $> t$

    } trim(L, t)

- Pictorially:

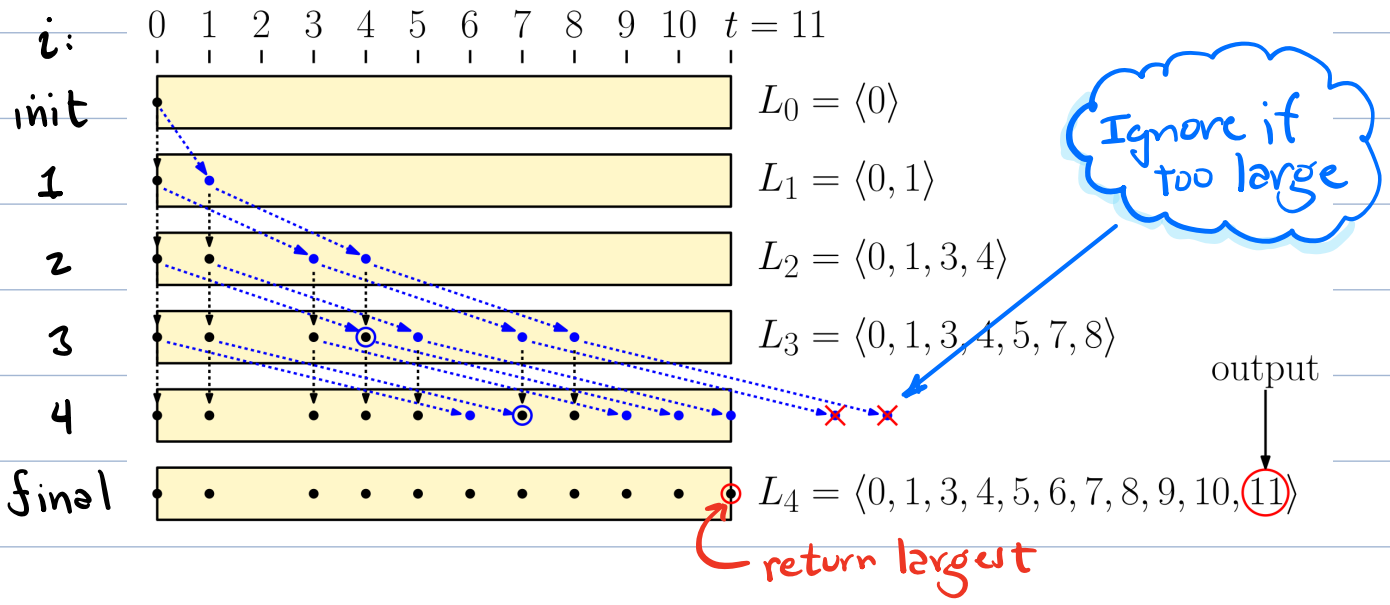



```
exact-subset-sum (x[1..n], t)      // exact subset sum
    L ← ⟨0⟩                        // basis
    for (i ← 1 to n)
        L ← merge(L, L + x[i])     // L ← L ∪ (L + x_i)
        L ← trim(L, t)             // rem. dups & > t
    return largest in L
```

Correctness – Easy

Running Time = $O(2^n)$ exponential!

Example:

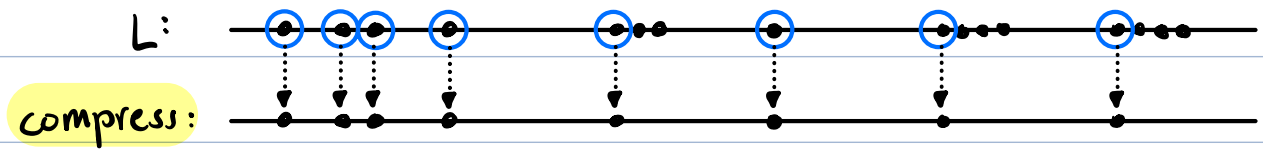


Approximation Algorithm

- Too many distinct sums (potentially 2^n)
- Idea: compress nearly equal sums

$L = \langle \ldots, 9851, 9852, 9860, 9950, 9952, 10{,}301, \ldots \rangle$

compress to 9851 compress to 9950



How large a difference can we tolerate?

prev = previous unpruned value

y = element under consideration

Cannot prune y if relative difference $> \delta$:

$$\frac{y - prev}{y} > \delta \iff y > \frac{prev}{1-\delta}$$

```
compress(L, δ, t)   // compress sorted list L
    L' ← ∅
    prev ← 0
    for each y ∈ L
        if (y > t) break          // ignore values > t
        if (y > prev/(1-δ))       // different enough?
            append y to L'        // add to L
            prev ← y
    return L'                     // final compressed list
```

Example: $L = \langle 10, 11, 12, 15, 20, 21, 22, 51, 53, 54, 56, 80 \rangle$

$\delta = 0.1 \quad t = 60$

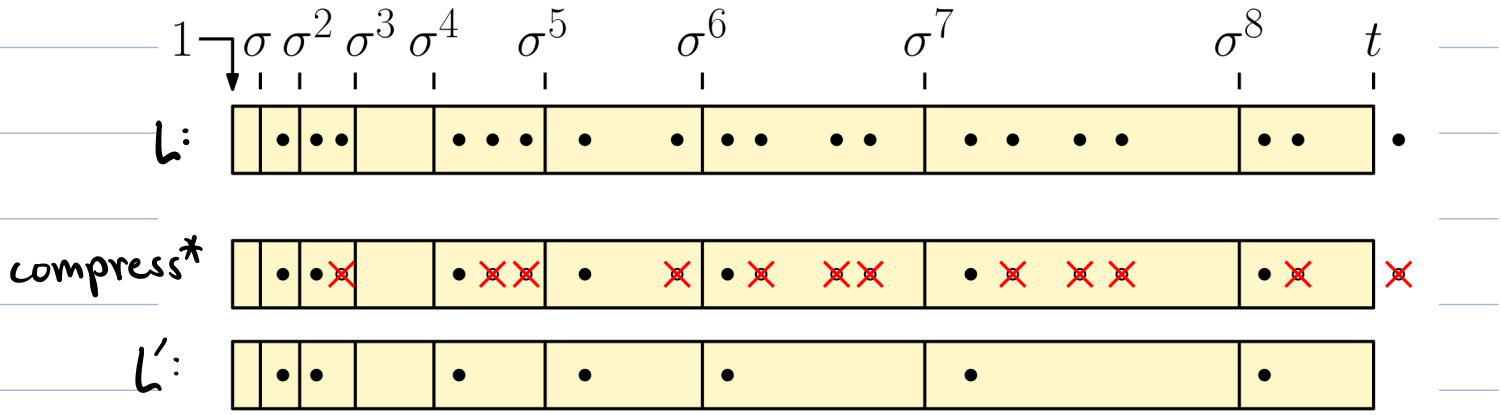
$L' = \langle 10, 12, 15, 20, 51 \rangle$ (>t)

return $\langle 10, 12, 15, 20, 51 \rangle$

"Bucket" Perspective:

- A bit easier to visualize in terms of buckets of same relative size
- Let $\sigma = 1/(1-\delta)$ $(\sigma > 1)$
- Breakpoints: $1, \sigma, \sigma^2, \sigma^3, \ldots, \sigma^i, \ldots$
- Bucket j: $[\sigma^{j-1}, \sigma^j]$

compress*(L, δ, t) - keep smallest in each bucket up to t



- Not exactly same as compress, but same properties. If y, y' in same bucket then $\sigma^{i-1} \le y < y' \le \sigma^i$

$$\frac{y' - y}{y'} = 1 - \frac{y}{y'} \le 1 - \frac{\sigma^{i-1}}{\sigma^i} = 1 - \frac{1}{\sigma} = \delta$$

$\sigma = 1/(1-\delta)$

$\Rightarrow 1 - \frac{1}{\sigma} = \delta$

$\Rightarrow$ in compress, y would prune y'

Value of compression – Number of surviving values is polynomial, not exponential!

Lemma: After compress(L, δ, t) where $\delta = \varepsilon/n$, number of nonzero items in L is $O((n \log t)/\varepsilon)$

Proof: Each consecutive pair of survivors is separated by factor of $\geq \sigma$, where $\sigma = 1/1-\delta$

- Let k = number of nonzero survivors
- $L_{final} = \langle 0, y_1, y_2, \ldots, y_k \rangle$ s.t. $y_i \geq \sigma \cdot y_{i-1}$

- $y_1 \geq 1 \Rightarrow y_i \geq \sigma^{i-1} \cdot y_1 \geq \sigma^{i-1}$

$\Rightarrow y_k \geq \sigma^{k-1}$ but $y_k \leq t$

$\Rightarrow \sigma^{k-1} \leq t$

$\Rightarrow (k-1) \ln \sigma \leq \ln t$

$\Rightarrow k \leq 1 + \ln t / \ln \sigma$

$= 1 + \ln t / \ln 1/(1-\delta)$

$= 1 + \ln t / (-\ln(1-\delta))$

Standard inequality: $-\ln(1-x) \geq x$

$\Rightarrow k \leq 1 + (\ln t)/\delta$

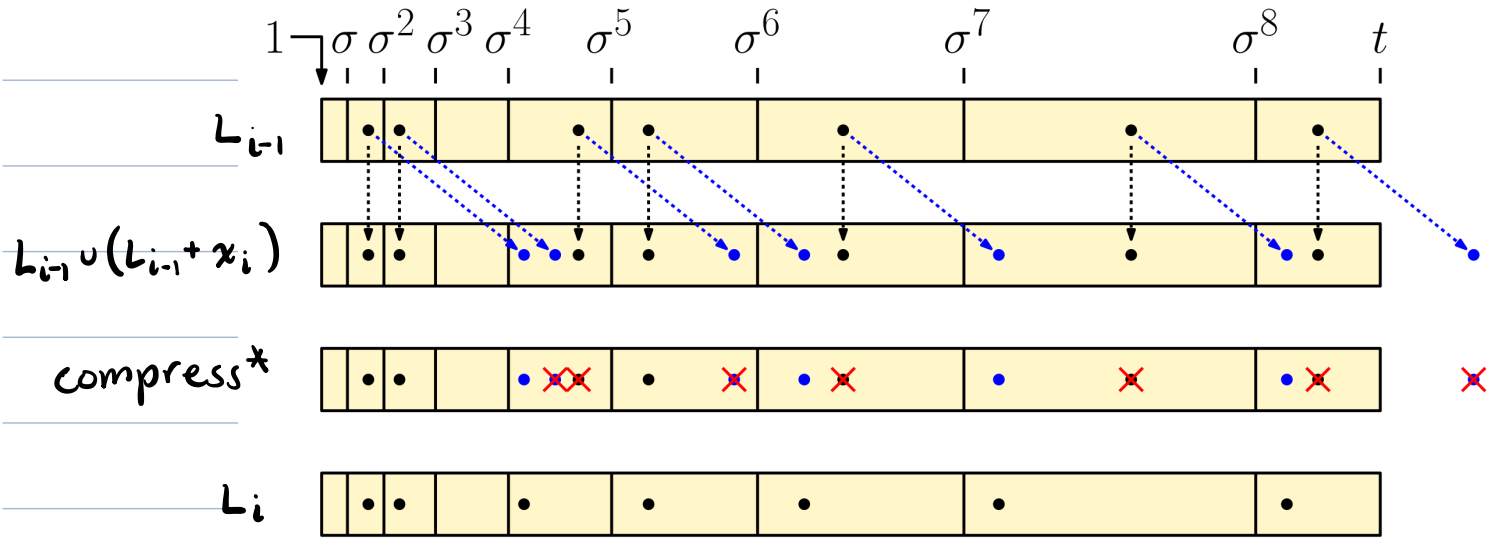
$\delta = \varepsilon/n \Rightarrow k \leq 1 + (n \cdot \ln t)/\varepsilon = O\left(\frac{n \cdot \log t}{\varepsilon}\right)$ □

Assuming $x_i \le t$, each number needs $O(\log t)$ bits

- Total input size = $O(n \log t)$
- List size $O((n \log t)/\varepsilon) = \frac{1}{\varepsilon} \cdot O(\text{input size})$
- Polynomial!

```
approx-subset-sum(x[1..n], t, ε)
    δ ← ε/n                          // pruning resolution
    L ← ⟨0⟩                          // basis case
    for (i ← 1 to n)
        L ← merge(L, L + x[i])       // add in x_i
        L ← compress(L, δ, t)        // + compress
    return largest item in L         // largest survivor
```

Example: (using compress* rather than compress)



Approximation Analysis:

Let Y^* = optimum subset sum value

Y_ε = algorithm output

→ Although we only output sum, there is a subset summing to Y_ε.

we know: $Y_\varepsilon \leq Y^* \leq t$

want to show $Y_\varepsilon \geq (1-\varepsilon) Y^*$

Intuition:

- Algorithm has n stages
- We allowed an error of $\delta = \varepsilon/n$ per stage
- Total error $\leq n \cdot \delta = \varepsilon$ ☺
- But these are multiplicative errors not additive errors ☹
- We'll show the intuition still works

Define:

L_i = algorithm's list after i^{th} iteration

L_i^* = exact list after i^{th} iteration (no compression)

Claim that for $y \in L_i^*$ there is "close" $z \in L_i$

Lemma: For $0 \le i \le n$, for all $y \in L_i^*$, there exists $z \in L_i$, s.t.

$$(1-\delta)^i y \le z \le y,$$

where $\delta = \varepsilon/n$.

z called y's representative

Proof: We'll just prove lower bound:

$$(1-\delta)^i y \le z$$

(Upper bound is an easy extension)

By induction on i. Basis: $L_0 = L_0^* = \langle 0 \rangle$ ✓

Let's assume lemma holds for $i-1$ & we'll show i

Every element of L_i^* arises in two ways:

$y \in L_{i-1}^*$

or $y + x_i$, where $y \in L_{i-1}^*$

We'll show both y & $y + x_i$ have representatives

By ind. hypothesis, $\exists\, z \in L_{i-1}$ s.t.

$$(1-\delta)^{i-1} y \le z \quad (a)$$

By adding x_i to both, we have

$$(1-\delta)^{i-1} y + x_i \leq z + x_i$$

$$\Rightarrow (1-\delta)^{i-1}(y+x_i) \leq z + x_i \quad (b) \quad (\text{since } 1-\delta < 1)$$

By compression, neither z nor $z+x_i$ in L_i^*

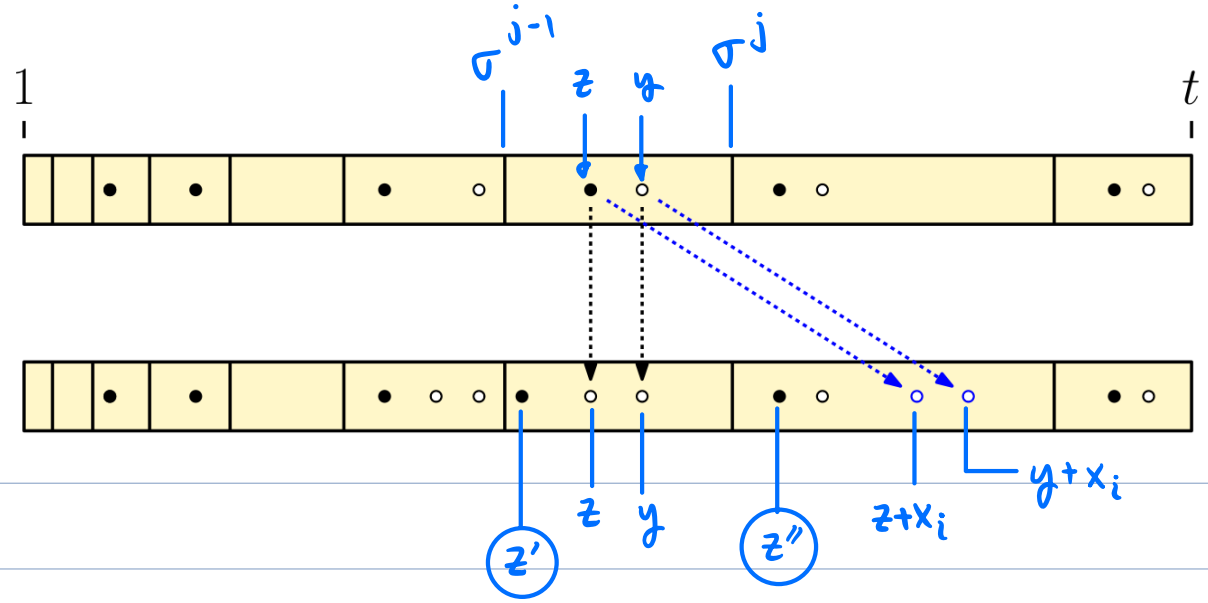
Let $z' + z''$ be elements that pruned z & $z+x_i$

By rules of compression: (c)

$$(1-\delta) z \leq z'$$

$$(1-\delta)(z+x_i) \leq z''$$

Illustration in compress*:



Combining (a), (b), (c):

$$(1-\delta)^{i-1}(1-\delta)y \le z' \Rightarrow (1-\delta)^{i} y \le z'$$

$$(1-\delta)^{i-1}(1-\delta)(y+x_i) \le z'' \Rightarrow (1-\delta)^{i}(y+x_i) \le z'$$

Since $z' + z'' \in L_i$, this completes proof □

Applying lemma with: $i = n$, $y = Y^*$ (opt s.s)
$z = Y$ (approx s.s.)

we have:

$$\left(1-\frac{\varepsilon}{n}\right)^n Y^* \le Y \le Y^*$$

Standard lemma: $\forall\, n > 0 \;\&\; \forall\, a \in \mathbb{R}$

$$(1+a) \le \left(1+\frac{a}{n}\right)^n$$

Set $a = -\varepsilon$

$$(1-\varepsilon) \le \left(1-\frac{\varepsilon}{n}\right)^n$$

$$\Rightarrow \quad (1-\varepsilon) Y^* \le Y \le Y^*$$

$\Rightarrow$ approx-subset-sum yields an ε-approximation ✓

Summary:

- Subset sum problem - SS
- SS is NP-complete (see below)
- PTAS for SS
  - User parameter ε
  - We return a sum Y s.t.
    $(1-\varepsilon)Y^* \leq Y \leq Y^*$
    where Y^* is optimum
  - Running time:
    - n phases
    - Each updates list of size
      $O((n \log t)/\varepsilon)$
    $\Rightarrow O((n^2 \log t)/\varepsilon)$

Weakly polynomial for fixed ε

SS is NP-Complete:

OPTIONAL (Not on quiz or final)

(i) SS ∈ NP (easy)

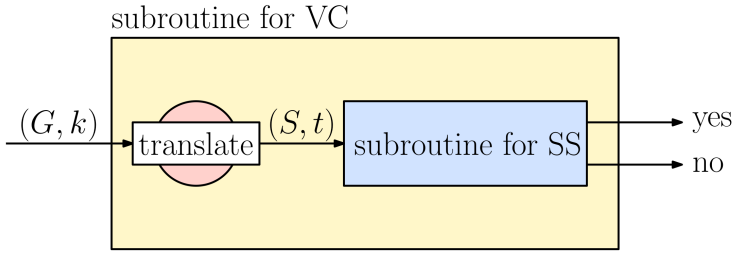
- certificate: indices of elements in S'
- verification: add up and check = t

(ii) SS is NP-hard

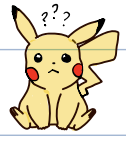
- We'll show VC $\leq_p$ SS
- Give poly-time function f:

$f(G, k) \longrightarrow (S, t)$, s.t.

G has vc of size k iff S has subset sums to t



Intuition:



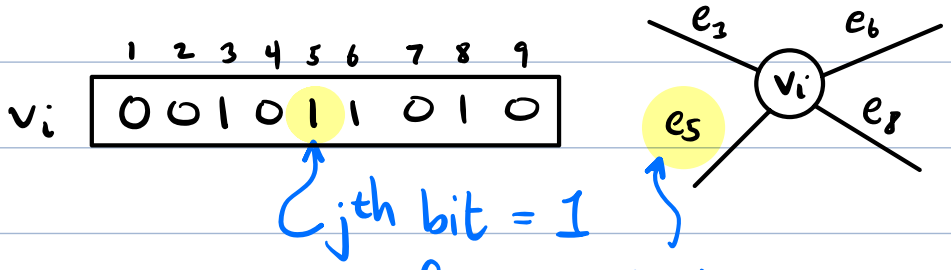
- Why is computing sums like building a cover?
- Numbers
  - ≡ bit strings
  - ≡ sets (bit vector)
- Addition
  - ≈ boolean-or
  - ≈ set union

First Attempt: Given graph $G = (V, E)$ and k, does G have a vertex cover of size k?

- Let: $n = |V|$ $m = |E|$

  $V = \{v_1, \ldots, v_n\}$ $E = \{e_1, \ldots, e_m\}$

- For each $v_i \in V$ create bit vector:

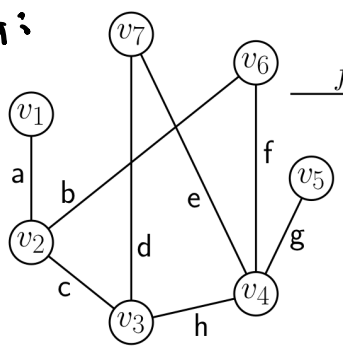


j^{th} bit = 1 if e_j incident to v_i

Example:

Vertex cover $\{v_2, v_3, v_4\}$

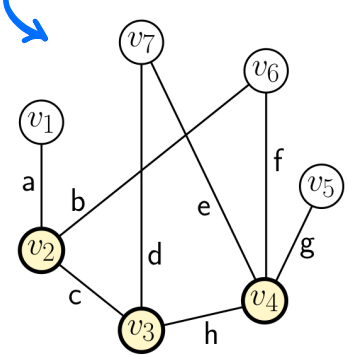
G:



f

	a	b	c	d	e	f	g	h
v_1	1	0	0	0	0	0	0	0
v_2	1	1	1	0	0	0	0	0
v_3	0	0	1	1	0	0	0	1
v_4	0	0	0	0	1	1	1	1
v_5	0	0	0	0	0	0	1	0
v_6	0	1	0	0	0	1	0	0
v_7	0	0	0	1	1	0	0	0
$t =$	1	1	1	1	1	1	1	1

$= v_2 \vee v_3 \vee v_4$



K=3

Every edge must be covered

⇒ logical-or has 1 in every position

Observe:

G has vertex cover of size k

iff

There are k bit vectors whose "or" is 1111...1

Not what we want:

① logical-or ≠ sum

② subset sum has no limit on number of items in sum

Second Attempt: (Correct)

- Addition has carries

$$\begin{array}{r} 1101 \\ \vee\ 0011 \\ \hline 1111 \end{array} \neq \begin{array}{r} 1101_2 \\ +\ 0011_2 \\ \hline 10000_2 \end{array}$$

Solution - Use larger base to avoid carries

$$\begin{array}{r} 1101 \\ \vee\ 0011 \\ \hline 1111 \end{array} \longleftrightarrow \begin{array}{r} 1101_4 \\ +\ 0011_4 \\ \hline 1112_4 \end{array}$$

Base 4 {0,1,2,3}

(closer)

- Each edge may covered once or twice

⇒ final sum has 1's + 2's: 121122_4

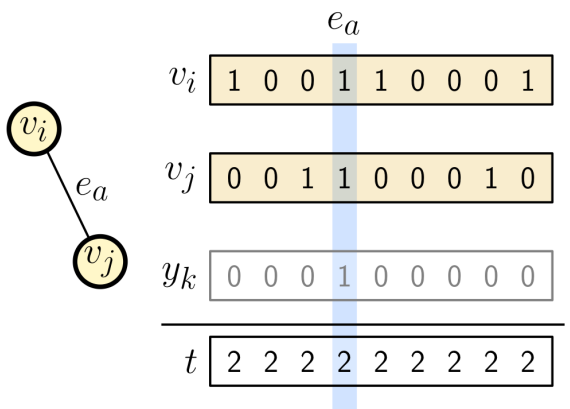
Solution: Create a set of **slack values** with **1 digit** in each. Use with **discretion** to make sum = $222\ldots2_4$

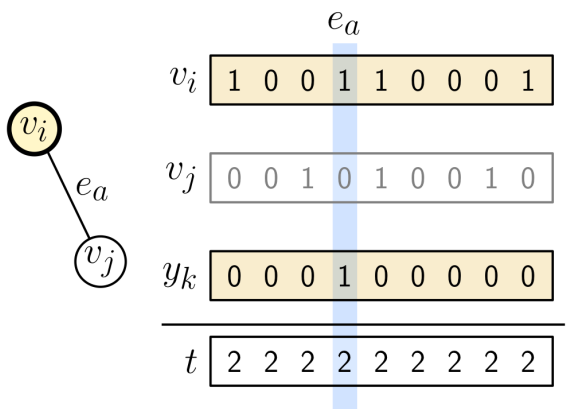
vertex sum: 121122_4

slacks:

y_1	1	0	0	0	0	0	← Select to bump 1-digits up to 2
y_2	0	1	0	0	0	0	
y_3	0	0	1	0	0	0	
y_4	0	0	0	1	0	0	
y_5	0	0	0	0	1	0	
	0	0	0	0	0	1	
t =	2	2	2	2	2	2	$_4$

Example:





- **How to control size k of vertex cover?**
  - Add **column of 1's**
  - Adjust **t** so **this column sums to k**

⇒ **k values selected** ⇒ **k vertices selected**

Here's the whole reduction: Given $G=(V,E)$, k

- Let $n=|V|$ + $m=|E|$
- Create n vertex values in base 4, $\{v_1,\dots,v_n\}$
  - For v_i: $m+1$ digits
    - Leading digit = 1
    - If v_i incident to edge e_j set j^{th} digit to 1 (else 0)

- Create m slack values in base 4, $\{y_1,\dots,y_m\}$
  - For y_j: $m+1$ digits
    - Leading digit = 0
    - Set j^{th} digit to 1 + all others = 0

- Set t to $m+1$ digits in base 4
  - Leading digit = k
  - Remaining m digits = 2

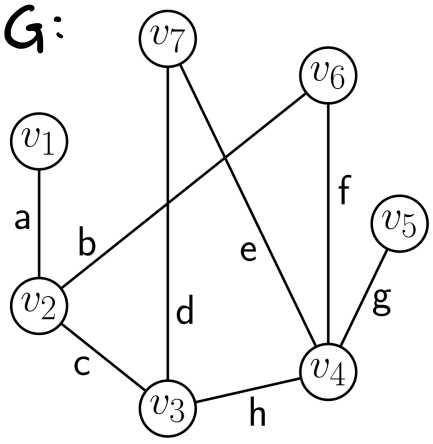
- Convert numbers from base 4 to base 10 (or whatever subset sum expects) $\rightarrow (S,t)$

Time - Total of $n+m$ numbers, each with $m+1$ digits

$\Rightarrow O(m(n+m))$ - polynomial

Example:

G:



$k = 3$

f

Vertex cover $= \{v_2, v_3, v_4\}$

		a	b	c	d	e	f	g	h
v_1	1	1	0	0	0	0	0	0	0
v_2	1	1	1	1	0	0	0	0	0
v_3	1	0	0	1	1	0	0	0	1
v_4	1	0	0	0	0	1	1	1	1
v_5	1	0	0	0	0	0	0	1	0
v_6	1	0	1	0	0	0	1	0	0
v_7	1	0	0	0	1	1	0	0	0
y_a	0	1	0	0	0	0	0	0	0
y_b	0	0	1	0	0	0	0	0	0
y_c	0	0	0	1	0	0	0	0	0
y_d	0	0	0	0	1	0	0	0	0
y_e	0	0	0	0	0	1	0	0	0
y_f	0	0	0	0	0	0	1	0	0
y_g	0	0	0	0	0	0	0	1	0
y_h	0	0	0	0	0	0	0	0	1
$t =$	3	2	2	2	2	2	2	2	2

vertex values

slack values

k

All numbers in base 4

		a	b	c	d	e	f	g	h
v_1	1	1	0	0	0	0	0	0	0
v_2	1	1	1	1	0	0	0	0	0
v_3	1	0	0	1	1	0	0	0	1
v_4	1	0	0	0	0	1	1	1	1
v_5	1	0	0	0	0	0	0	1	0
v_6	1	0	1	0	0	0	1	0	0
v_7	1	0	0	0	1	1	0	0	0
y_a	0	1	0	0	0	0	0	0	0
y_b	0	0	1	0	0	0	0	0	0
y_c	0	0	0	1	0	0	0	0	0
y_d	0	0	0	0	1	0	0	0	0
y_e	0	0	0	0	0	1	0	0	0
y_f	0	0	0	0	0	0	1	0	0
y_g	0	0	0	0	0	0	0	1	0
y_h	0	0	0	0	0	0	0	0	1
$t =$	3	2	2	2	2	2	2	2	2

take as needed to make 2's

Counts number of vertices in cover

Correctness:

Lemma: G has a vertex cover of size k iff S has a subset that sums to t.

Proof: (⇒) Suppose G has vertex cover V', $|V'| = k$

To construct subset sum S':

- for each $v_i \in V'$:
  - add vertex value v_i to S'
- for each edge $e_j \in E$:
  - if only one endpt in V', add y_j to S'

- Claim: S' sums to $t = [k\,2\,2\,2 \ldots 2_4]$
  - Since $|V'| = k$, k values contribute 1 to leading digit position
  - Every edge e_j is covered:
    - If once - take vertex + y_j = 2 ✓
    - If twice - take both vertices = 2 ✓

(⇐) Suppose S has subset S' sums to t.

- To match leading digit, there are k vertex values → V'
- The only way to get 222…2 is for each column to contribute one or two vertices to V'. Why?
  - slacks can only increase counts by +1
  - no carries possible because only two 1-digits per column.

□