CS 341: ALGORITHMS

Lecture 19: intractability I

Readings: see website

Trevor Brown

https://student.cs.uwaterloo.ca/~cs341

trevor.brown@uwaterloo.ca

THIS TIME

- Intractability (hardness of problems)
  - Decision problems
  - Complexity class P
  - Polynomial-time **Turing** reductions
  - Introductory reductions
    - Three flavours of the **traveling salesman** problem

INTRACTABILITY

Studying the **hardness** of problems

Decision Problems

Decision Problem: Given a problem instance I, answer a certain question "yes" or "no".

Problem Instance: Input for the specified problem.

Problem Solution: Correct answer ("yes" or "no") for the specified problem instance. I is a **yes-instance** if the correct answer for the instance I is "yes". I is a **no-instance** if the correct answer for the instance I is "no".

Size of a problem instance: $\mathit{Size}(I)$ is the number of bits required to specify (or encode) the instance I.

The Complexity Class P

Algorithm Solving a Decision Problem: An algorithm A is said to **solve** a decision problem Π provided that A finds the correct answer ("yes" or "no") for every instance I of Π in finite time.

Polynomial-time Algorithm: An algorithm A for a decision problem Π is said to be a **polynomial-time algorithm** provided that the complexity of A is $O(n^k)$, where k is a positive integer and $n = \mathit{Size}(I)$.

The Complexity Class P denotes the set of all decision problems that have polynomial-time algorithms solving them. We write $\Pi \in \mathbf{P}$ if the decision problem Π is in the complexity class **P**.

Knapsack Problems

Relative problem hardness?

Problem 7.3

0-1 Knapsack-Dec

Instance: *a list of* **profits**, $P = [p_1, \ldots, p_n]$; *a list of* **weights**, $W = [w_1, \ldots, w_n]$; *a* **capacity**, M; *and a* **target profit**, T.

Question: *Is there an n-tuple* $[x_1, x_2, \ldots, x_n] \in \{0,1\}^n$ *such that* $\sum w_i x_i \leq M$ *and* $\sum p_i x_i \geq T$?

Problem 7.4

Rational Knapsack-Dec

Instance: *a list of* **profits**, $P = [p_1, \ldots, p_n]$; *a list of* **weights**, $W = [w_1, \ldots, w_n]$; *a* **capacity**, M; *and a* **target profit**, T.

Question: *Is there an n-tuple* $[x_1, x_2, \ldots, x_n] \in [0,1]^n$ *such that* $\sum w_i x_i \leq M$ *and* $\sum p_i x_i \geq T$?

Cycles in Graphs

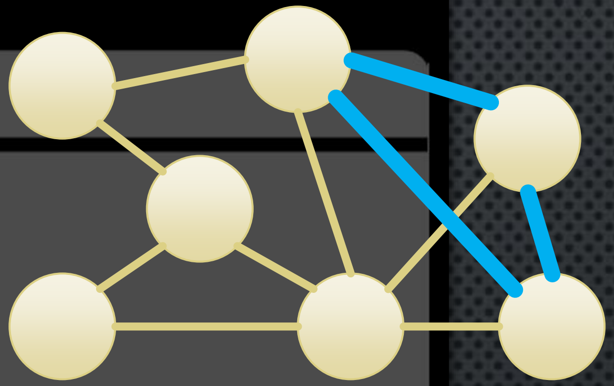
Relative hardness?

Problem 7.1

Cycle

Instance: *An undirected graph* $G = (V, E)$.

Question: *Does* G *contain a cycle?*

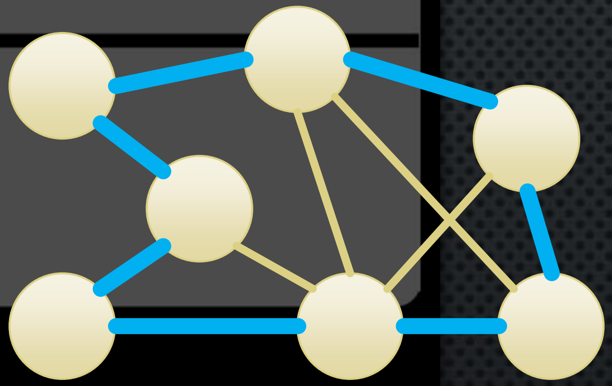


Problem 7.2

Hamiltonian Cycle

Instance: *An undirected graph* $G = (V, E)$.

Question: *Does* G *contain a hamiltonian cycle?*



A **hamiltonian cycle** is a cycle that passes through every vertex in V exactly once.

Polynomial-time Turing Reductions

Suppose Π_1 and Π_2 are problems (not necessarily decision problems). A (hypothetical) algorithm B to solve Π_2 is called an **oracle** for Π_2.

Suppose that A is an algorithm that solves Π_1, assuming the existence of an oracle B for Π_2. (B is used as a subroutine within the algorithm A.)

Then we say that A is a **Turing reduction** from Π_1 to Π_2, denoted $\Pi_1 \leq^T \Pi_2$.

A Turing reduction A is a **polynomial-time Turing reduction** if the running time of A is polynomial, under the assumption that the oracle B has **unit cost** running time.

If there is a polynomial-time Turing reduction from Π_1 to Π_2, we write $\Pi_1 \leq_P^T \Pi_2$.

Informally: Existence of a polynomial-time Turing reduction means that if we can solve Π_2 in polynomial time, then we can solve Π_1 in polynomial time.

Example: all-pairs-shortest-paths easily reduces to single-source-shortest-path

A reduction typically:
1. transforms the larger problem's input so it can be fed to the oracle, and
2. transforms the oracle's output into a solution to the larger problem.

Travelling Salesperson Problems

Problem 7.5

TSP-Optimization

Instance: *A graph G and edge weights $w : E \to \mathbb{Z}^+$.*

Find: *A hamiltonian cycle H in G such that $w(H) = \sum_{e \in H} w(e)$ is minimized.*

Problem 7.6

TSP-Optimal Value

Instance: *A graph G and edge weights $w : E \to \mathbb{Z}^+$.*

Find: *The minimum T such that there exists a hamiltonian cycle H in G with $w(H) = T$.*

Problem 7.7

TSP-Decision

Instance: *A graph G, edge weights $w : E \to \mathbb{Z}^+$, and a target T.*

Question: *Does there exist a hamiltonian cycle H in G with $w(H) \leq T$?*

Positive edge weights

Return type "a path/cycle H"

Is TSP-Dec $\leq_P^T$ TSP-Optimal Value?

Return type "a positive integer T"

Is TSP-Dec $\leq_P^T$ TSP-Optimization?

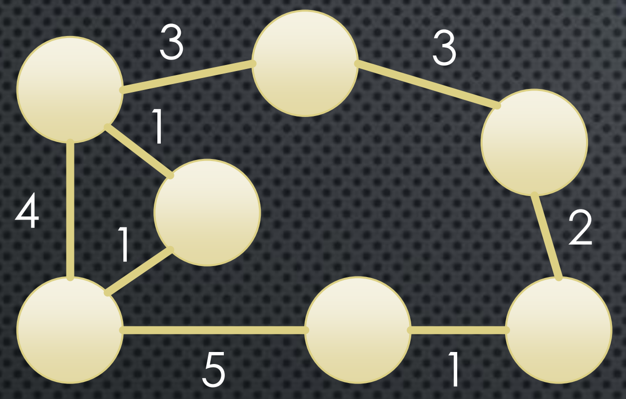
Return type "yes/no"

We will use polynomial-time Turing reductions to show that different versions of the **TSP** are polynomially equivalent: if one of them can be solved in polynomial time, then all of them can be solved in polynomial time. (However, it is believed that none of them can be solved in polynomial time.)

- We already know
  - TSP-Dec $\leq_P^T$ TSP-Optimal Value
  - TSP-Dec $\leq_P^T$ TSP-Optimization
- We show
  - TSP-Optimal Value $\leq_P^T$ TSP-Dec
  - TSP-Optimization $\leq_P^T$ TSP-Dec

TSP-Optimal Value $\leq_P^T$ TSP-Dec

TSP-Optimal Value input: **G, w**



TSP-Dec() **also** needs a **target T**

What if we try **TSP-Dec(G, w, 100)**?

It returns true. But we don't learn optimal value… just that it's ≤100

Problem 7.6

TSP-Optimal Value

Instance: *A graph G and edge weights $w : E \to \mathbb{Z}^+$.*

Find: *The minimum T such that there exists a hamiltonian cycle H in G with $w(H) = T$.*

Problem 7.7

TSP-Decision

Instance: *A graph G, edge weights $w : E \to \mathbb{Z}^+$, and a target T.*

Question: *Does there exist a hamiltonian cycle H in G with $w(H) \leq T$?*

How can we learn the **exact** optimal value by making such calls?

TSP-Optimal Value $\leq_P^T$ TSP-Dec

Use **binary search**! How to define the ***starting range (lo, hi)*** to search?

Algorithm: *TSP-OptimalValue-Solver*(G, w)

external *TSP-Dec-Solver*

$hi \leftarrow \sum_{e \in E} w(e)$

Largest possible cycle could include **every** edge

$lo \leftarrow 0$

0 is smallest possible weight for any cycle

if not *TSP-Dec-Solver*(G, w, hi) **then return** (∞)

Maybe there is no Hamiltonian cycle, at all

while $hi > lo$

do $\begin{cases} mid \leftarrow \lfloor \frac{hi+lo}{2} \rfloor \\ \textbf{if } \textit{TSP-Dec-Solver}(G, w, mid) \\ \quad \textbf{then } hi \leftarrow mid \\ \quad \textbf{else } lo \leftarrow mid + 1 \end{cases}$

return (hi)

This is a standard binary search technique.

Is this a **"poly-time reduction?"**

I.e., if we assume TSP-Dec-Solver runs in O(1) time, is the runtime a **polynomial in the input size**?

Questions: (1) What's the input size? (2) What's the runtime?

What's the size of the input $I = (G, w)$?

$$Size(I) = Size(G) + Size(w)$$

But wait… G and w could be represented in many different ways.
Could the choice of representation affect our complexity result?

Only for very inefficient representations (that are **exponentially larger than optimal**).

For example if we store weights in **unary**

We **rule out such inefficient representations** for the purpose of proving polynomial runtime

Polynomial differences in size do not matter.
Exercise: **if** $T \in poly(Size(I)^{40})$ **then** $T \in poly(Size(I))$

What's the size of the input $I = (G, w)$?

$$Size(I) = Size(G) + Size(w)$$

So, suppose G is represented as an **array of adjacency lists** (one list for each vertex), with each list containing **edges** to neighbouring vertices, and an edge is represented by a **weight** and the **name** of the target vertex

Bits to store **weight** of the edge (storing $w(e)$ takes $\log w(e) + 1$ bits)

Bits to store the **name** of the target vertex (in 1..|V|)

$$Size(I) = |V| + \underbrace{\sum_{e \in E} (\log w(e) + 1 + \log|V| + 1)}_{\textbf{For all edges}}$$

Array of empty lists for all vertices v

Let's relate this to runtime… what's the runtime?

TSP-Optimal Value $\leq_P^T$ TSP-Dec

Let's assume $O(1)$ time for operations on weights

Later we'll see this isn't needed to show polytime

Algorithm: *TSP-OptimalValue-Solver*(G, w)

external *TSP-Dec-Solver*

$hi \leftarrow \sum_{e \in E} w(e)$ — $O(|E|)$

$lo \leftarrow 0$ — $O(1)$

if not *TSP-Dec-Solver*(G, w, hi) **then return** (∞) — $O(1)$ for the oracle

while $hi > lo$ — # iterations: $O(\log(hi - lo))$ $= \log \sum_{e \in E} w(e)$

do $\left\{ \begin{array}{l} mid \leftarrow \left\lfloor \frac{hi+lo}{2} \right\rfloor \\ \textbf{if } \textit{TSP-Dec-Solver}(G, w, mid) \\ \quad \textbf{then } hi \leftarrow mid \\ \quad \textbf{else } lo \leftarrow mid + 1 \end{array} \right.$ — $O(1)$

return (hi)

Runtime $T(I) \in O(|E| + \log \sum_{e \in E} w(e))$

COMPARING $T(I)$ AND $Size(I)$

- $T(I) \quad \in O(|E| + \log \sum_{e \in E} w(e))$
- $Size(I) \quad = |V| + \sum_{e \in E}(\log w(e) + 1 + \log|V| + 1)$

$$= |V| + \Sigma_{e \in E}(\log w(e) + 1) + \boldsymbol{\Sigma_{e \in E}}(\log|V| + 1)$$

$$= |V| + \Sigma_{e \in E}(\log w(e) + 1) + \Sigma_{e \in E}(\log|V|) + |\boldsymbol{E}|$$

- Want to show $T(I) \in O(Size(I)^c)$ for some constant c (we show c=1)

$$O(|\boldsymbol{E}| + \log \textstyle\sum_{e \in E} w(e)) \subseteq^{?} O(|V| + \Sigma_{e \in E}(\log w(e) + 1) + \Sigma_{e \in E} \log|V| + |\boldsymbol{E}|)$$

$$\Leftrightarrow O(\log \textstyle\sum_{e \in E} w(e)) \subseteq^{?} O(|V| + \Sigma_{e \in E}(\log w(e) + 1) + \Sigma_{e \in E} \log|V|)$$

How to compare $\log \sum_{e \in E} w(e)$ and $\Sigma_{e \in E}(\log w(e) + 1)$?

COMPARING $T(I)$ AND $Size(I)$

- **How to compare $\log \sum_{e \in E} w(e)$ and $\Sigma_{e \in E}(\log w(e) + 1)$?**
- $\Sigma_{e \in E}(\log w(e) + 1) = (\log w(e_1) + 1) + (\log w(e_2) + 1) + \cdots + \left(\log\left(w\left(e_{|E|}\right)\right) + 1\right)$
- Can we combine these terms into one log using $\log x + \log y = \log xy$?
- $\Sigma_{e \in E}(\log w(e) + 1) \; = (\log w(e_1) + \log 2) + + \cdots + \left(\log\left(w\left(e_{|E|}\right)\right) + \log 2\right)$
- $\Sigma_{e \in E}(\log w(e) + 1) \; = \log 2w(e_1)\, 2w(e_2) \; \ldots \; 2w\left(e_{|E|}\right) = \log \prod_{e \in E} 2w(e)$
- So how to compare $\log \prod_{e \in E} 2w(e)$ and $\log \sum_{e \in E} w(e)$?
  - **All** $w(e)$ are positive integers, so $\prod_{e \in E} 2w(e) \geq \sum_{e \in E} w(e)$
  - Since log is increasing on $\mathbb{Z}^+$, $\log \prod_{e \in E} 2w(e) \geq \log \sum_{e \in E} w(e)$

COMPARING $T(I)$ AND $Size(I)$

- We in fact show $T(I) \in O(Size(I))$

$$O(\log \textstyle\sum_{e \in E} w(e)) \subseteq^{?} O(|V| + \Sigma_{e \in E}(\log w(e) + 1) + \Sigma_{e \in E} \log|V|)$$

How to compare $\log \sum_{e \in E} w(e)$ and $\Sigma_{e \in E}(\log w(e) + 1)$?

We just saw $\Sigma_{e \in E}(\log w(e) + 1) = \log \prod_{e \in E} 2w(e) \geq \log \sum_{e \in E} w(e)$

So $T(I) \in O(Size(I)^c)$ where $c = 1$

So this reduction has runtime that is polynomial in the input size!

TSP-Optimal Value $\leq_P^T$ TSP-Dec

Algorithm: *TSP-OptimalValue-Solver*(G, w)

external *TSP-Dec-Solver*

$hi \leftarrow \sum_{e \in E} w(e)$

$lo \leftarrow 0$

if not *TSP-Dec-Solver*(G, w, hi) **then return** (∞)

while $hi > lo$

do $\begin{cases} mid \leftarrow \left\lfloor \frac{hi+lo}{2} \right\rfloor \\ \textbf{if } \textit{TSP-Dec-Solver}(G, w, mid) \\ \quad \textbf{then } hi \leftarrow mid \\ \quad \textbf{else } lo \leftarrow mid + 1 \end{cases}$

return (hi)

Exercise: show the variant of this reduction where **linear search** is used instead of binary search is **not** $\boldsymbol{poly(Size(I))}$

REACHED THIS POINT

(but will recap the comparison of T(I) and Size(I) next time)

TSP-Optimal Value $\leq_P^T$ TSP-Dec

Algorithm: *TSP-OptimalValue-Solver*(G, w)
external *TSP-Dec-Solver*
$hi \leftarrow \sum_{e \in E} w(e)$
$lo \leftarrow 0$
if not *TSP-Dec-Solver*(G, w, hi) **then return** (∞)
while $hi > lo$
do $\begin{cases} mid \leftarrow \lfloor \frac{hi+lo}{2} \rfloor \\ \textbf{if } \textit{TSP-Dec-Solver}(G, w, mid) \\ \quad \textbf{then } hi \leftarrow mid \\ \quad \textbf{else } lo \leftarrow mid + 1 \end{cases}$
return (hi)

So TSP-OptimalValue-Solver is polytime… But is it a **correct reduction from TSP-Optimal Value to TSP-Dec?**

Need to prove:
TSP-OptimalValue-Solver(G,w)
returns the weight $\boldsymbol{W}$
of the **shortest Hamiltonian Cycle (HC) in G**

Sketch: We return ∞ iff there is **no HC**.
Loop invariant: $W \in [lo, hi]$.
So, at termination when $hi = lo$,
we return exactly $hi = W$.

TSP-Optimal Value $\leq_P^T$ TSP-Dec

Algorithm: *TSP-OptimalValue-Solver*(G, w)

```
external TSP-Dec-Solver
hi ← Σ_{e∈E} w(e)
lo ← 0
if not TSP-Dec-Solver(G, w, hi) then return (∞)
while hi > lo
  do { mid ← ⌊(hi+lo)/2⌋
       if TSP-Dec-Solver(G, w, mid)
         then hi ← mid
         else lo ← mid + 1
return (hi)
```

So, TSP-OptimalValue-Solver is **polytime**, and is a **correct** reduction.

We have therefore shown:
TSP-Optimal Value is polytime reducible to TSP-Dec

So, if an $\boldsymbol{O(1)}$ implementation of TSP-Dec-Solver exists, then we have a **polytime** implementation of TSP-Optimal-Value-Solver!

In fact, TSP-OptimalValue-Solver remains **polytime** even if the implementation of the **oracle runs in polytime** instead of O(1)!

TSP-Optimal Value $\leq_P^T$ TSP-Dec

Algorithm: *TSP-OptimalValue-Solver*(G, w)
external *TSP-Dec-Solver*
$hi \leftarrow \sum_{e \in E} w(e)$
$lo \leftarrow 0$
if not *TSP-Dec-Solver*(G, w, hi) **then return** (∞)
while $hi > lo$
do
  $mid \leftarrow \lfloor \frac{hi+lo}{2} \rfloor$
  if *TSP-Dec-Solver*(G, w, mid)
    then $hi \leftarrow mid$
    else $lo \leftarrow mid + 1$
return (hi)

TSP-OptimalValue-Solver remains **polytime** even if the **oracle runs in polytime** instead of O(1)!

The key idea is: Consider polynomials $P_R(s)$ and $P_O(s)$ representing the runtime of a reduction and its oracle, respectively, on an input of size s.
Worst possible runtime happens if **every step** in the reduction is a call to the oracle.
This is $P_R(s)P_O(s)$ --- multiplication of polynomials.

But multiplying polynomials of degrees d_1, d_2 results in a polynomial of degree $\leq d_1 + d_2$. **Example:**

$$P_1(x) = 5x^2 + 10x + 100$$
$$P_2(x) = 20x^3 + 20$$
$$P_1(x)P_2(x) = (5x^2 + 10x + 100)(20x^3 + 20)$$
$$= 100x^5 + 200x^4 + 2000x^3 + 100x^2 + 200x + 2000$$

PROVING REDUCTIONS CORRECT

- **In more complex reductions** where we **transform the input** before calling the oracle, we will need a **more complex proof**:
- (A) If there is a(n optimal) solution in the input, our transformation will preserve that solution so the oracle can find it, and
- (B) Our transformation doesn't introduce new solutions that are ***not*** present in the original input
  - *(i.e., if we find a solution in the transformed input, there was a corresponding solution in the original input)*

More on this later…

INPUT SIZE **CHEAT SHEET**

Input I	**Perfectly fine** choices of $Size(I)$
int x	1 or $\lfloor \log(x) \rfloor + 1$ **(can simplify to** $\log(x) + 1$ **or** $\log x$**)**
Graph (V, E) with weights W:	$\|V\|$ or $\|E\|$ or $\|V\|^2$ or $\|V\| + \|E\|$ or $\sum_{e \in E}(\log(w(e)) + 1)$ or $\sum_{u,v \in V}(\log(w(u,v)) + 1)$ or **any sum of terms above**
$A[1..n]$ of int	n or $\sum_i(\log(A[i]) + 1)$
n x n matrix m	n^2 or $\sum_{i,j}(\log(m_{ij}) + 1)$

To write down x=1, need log(1)+1=1 bit. For x=2 this is 2 bits. For x=4, 3 bits.

Pick any expression that makes your analysis easy

Exponentially larger than optimal representation!

Input I	**Examples of** ***BAD*** choices of $Size(I)$
int x	x
Graph (V, E)	$2^{\|V\|}$ or $\|V\|^{\|E\|}$ or $\sum_{e \in E} w(e)$
$A[1..n]$ of int	2^n or $\sum_i A[i]$

Pseudo-polynomial ~= no exponentiation of non-constant terms

Technically any **pseudo-polynomial combination** of these terms is fine. For example, the following is fine: $(|E|^{100} + |V|^2) \cdot \sum_{e \in E}(\log(w(e)) + 1)$

BONUS SLIDES

efficient vs inefficient input representations

What's the size of the input I?

$$Size(I) = Size(G) + Size(w)$$

But wait… G and w could be represented in many different ways.
Could the choice of representation affect our complexity result?

Representation 1: What if the entire graph is simply represented as a **weight matrix** $\boldsymbol{W}$ which contains a weight w_{uv} for each $u, v \in V$ (∞ if an edge does not exist)

Consider weight w_{uv}. **It takes $\boldsymbol{\Theta(\log w_{uv})}$ bits** ($\log(w_{uv}) + 1$) to store this weight.

We would then have:

$$Size(R_1) = \sum_{u \in V} \sum_{v \in V} \log(w_{uv}) + 1$$

What would it mean to have a **runtime $\boldsymbol{T}$** that is **polynomial in $\boldsymbol{Size(R_1)}$**?

We say **$\boldsymbol{T}$ is polynomial in $\boldsymbol{Size(R_1)}$** (denoted $\boldsymbol{T \in poly(Size(R_1))}$) iff:

$\exists$ **constant** c s.t. **for all $\boldsymbol{I}$**, we have $T \in O(Size(R_1)^c)$

Representation 2: What if the graph were represented as an **array of adjacency lists** (one list for each vertex), with each list containing **edges** to neighbouring vertices, where an edge is represented by a **weight** and the **name** of the target vertex?

We would then have:

$$Size(R_2) = |V| + \sum_{(u,v)\in E} (\log(w_{uv}) + 1 + \log|V| + 1)$$

Array with one list per vertex v

Weight of the edge

Name of the target vertex

Compare with **representation 1**:

$$Size(R_1) = \sum_{u\in V}\sum_{v\in V} \log(w_{uv}) + 1$$

Representation 3: What if we were to represent the graph as a weight matrix W but write all weights in ***unary***, instead of binary **(so it takes w_{uv} bits to store weight w_{uv})**.

For this (very stupid) representation, we would then have:

$$Size(R_3) = \sum_{u \in V} \sum_{v \in V} (w_{uv})$$

This can be **exponentially larger** than $Size(R_1)$!

Compare with **representation 1**:

$$Size(R_1) = \sum_{u \in V} \sum_{v \in V} (\log w_{uv}) + 1$$

For example, in a graph where there are $O(1)$ nodes and all edges have weight w: $Size(R_1) = \Theta(\log_2 w)$ and $Size(R_3) = \Theta(w)$.

In this case, $\boldsymbol{Size(R_3) \in \Theta\left(2^{Size(R_1)}\right)}$

So, some algorithms could be **polynomial** in $Size(R_3)$ **but exponential** in $Size(R_1)$

We should **rule out this highly inefficient representation** for the purpose of proving polynomial runtime

Idea: determine whether runtime is polynomial in the size of the **optimal representation of the input**

Problem: it's not clear what the **optimal** representation is…

What if we can argue the runtime is polynomial in some **lower bound** on the size of the input?

LOWER BOUNDING $Size(I)$

- To prove that a reduction's runtime $T(I)$ on **input I** is **polynomial in the size of I**:
  - Define a **lower bound $L(I)$** on the size of I
  - **For every possible representation I_R of I**, $L(I) \leq Size(I_R)$ should hold
    - Can be proved with information theory, or ad-hoc; **outside the scope of the course**
    - In this course, we can be a bit **sloppy**, and just use the **table of valid choices** here to obtain a term for each variable in I
  - **Then, if we can show** $T(I) \leq poly(L(I))$, we have **actually** shown $T(I) \leq poly(size(I))$

The following are **valid** choices of $L(I)$ for various input types:

Input I	$L(I)$
int x	1 or $\log(x) + 1$
Graph (V, E) possibly with weights W	1 or $\|V\|$ or $\|E\|$ or $\|V\| + \|E\|$ or $\sum_{e \in E}(\log(w(e)) + 1)$
$A[1..n]$ of int	n or $\sum_i (\log(A[i]) + 1)$
n x n matrix m	n^2 or $\sum_{i,j} (\log(m_{ij}) + 1)$

Justifying **sloppy** analysis:

Polynomial differences in choices of $L(I)$, such as $|V|$ vs $|V|^2$ vs $(|E| + |V|)^{40}$ **don't matter.**

Such differences **cannot change** whether a runtime $T(I)$ is in $poly(L(I))$ or not

Exercise: $T(I) \in poly(L(I)^{40})$ **iff** $T(I) \in poly(L(I))$

TSP-Optimal Value $\leq_P^T$ TSP-Dec

Algorithm: *TSP-OptimalValue-Solver*(G, w)

external *TSP-Dec-Solver*

$hi \leftarrow \sum_{e \in E} w(e)$

$lo \leftarrow 0$

if not *TSP-Dec-Solver*(G, w, hi) **then return** (∞)

while $hi > lo$

do $\begin{cases} mid \leftarrow \left\lfloor \frac{hi+lo}{2} \right\rfloor \\ \textbf{if } \textit{TSP-Dec-Solver}(G, w, mid) \\ \quad \textbf{then } hi \leftarrow mid \\ \quad \textbf{else } lo \leftarrow mid + 1 \end{cases}$

return (hi)

This is a standard binary search technique.

$O(|E|)$

$O(1)$

$O(1)$ for the oracle

iterations: $O(\log(hi - lo))$

$= \log \sum_{e \in E} w(e)$

Loop body: $O(1)$

So what's a valid $L(I)$ for an input I to TSP-OptimalValue-Solver?

Input is a graph G with weight matrix w.
From the table of valid $L(I)$ choices, we **let** $\boldsymbol{L(I) = |E| + \sum_{e \in E}(\log(w(e)) + 1)}$.

What's the relationship between the reduction's runtime $T(I)$ and $L(I)$?

$T(I) = O(|E| + \log \sum_{e \in E} w(e))$

and $\boldsymbol{L(I) = O\left(|E| + \sum_{e \in E}(\log(w(e)) + 1)\right)}$

As we argued earlier, $T(I) \in poly(\boldsymbol{L(I)})$

And thus $T(I) \in poly(Size(I))$

So this reduction has runtime that is polynomial in the input size!