A potential function must strictly decrease each iteration and have a lower bound

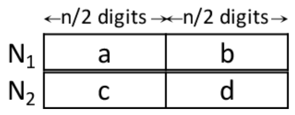
Euclid's Algorithm (finds the greatest common divisor of x and y):

```
Euclid($x$, $y$):  // for $x \geq y > 0$
if $y$ divides $x$: return $y$
return Euclid($y$, $x \bmod y$)
```

Big O = an upper bound
Big Ω = a lower bound
Big θ = tightest bound

Types of Algorithms:

- divide and conquer
    - Merge sort, closest-pair, karatsuba multiplication ($O(n^{\log_2 3})$, shown below)



$$m_1 = (a+b) \times (c+d)$$
$$m_2 = a \times c$$
$$m_3 = b \times d$$

$$\begin{aligned} N_1 \times N_2 &= a \times c \cdot 10^n + (a \times d + b \times c) \cdot 10^{n/2} + b \times d \\ &= m_2 \cdot 10^n + (m_1 - m_2 - m_3) \cdot 10^{n/2} + m_3 \end{aligned}$$

- dynamic programming
    - Fibonacci
    - Longest Common Subsequence

$$L(S_1[1..N], S_2[1..M]) = \begin{cases} 0 & \text{if } N = 0 \text{ or } M = 0 \\ L(S_1[1..N-1], S_2[1..M-1]) + 1 & \text{if } S_1[N] = S_2[M] \\ \max(L(S_1[1..N-1], S_2), L(S_1, S_2[1..M-1])) & \text{if } S_1[N] \neq S_2[M] \end{cases}$$

-
    - Longest Increasing Subsequence

$$L(i) = \begin{cases} 1 & \text{if } i = 1 \text{ or } S[j] \geq S[i] \text{ for all } 0 < j < i \\ 1 + \max\{L(j) : 0 < j < i \text{ and } S[j] < S[i]\} & \text{otherwise} \end{cases}$$

-
    - Floyd-Warshall (lowest-cost path between all pairs of vertices, where $d^k(i,j)$ is the minimum cost to travel from i to j going through only vertices 1 through k)

```
Algorithm $Floyd\text{-}Warshall(G = (V, E))$ :
  Let $d^0(i,j) = weight(i,j)$ for all $i, j \in V$
  For $k = 1$ to $|V|$ :
    For all $i, j \in V$ :
      $d^k(i,j) := \min\{d^{k-1}(i,j),\ d^{k-1}(i,k) + d^{k-1}(k,j)\}$
  Return $d^n$
```

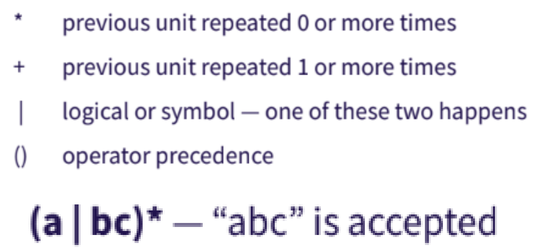
- Greedy
    - Kruskal's MST algorithm (insert cheapest edge that doesn't create cycle)

Master Theorem: if $T(n) = kT(\frac{n}{b}) + O(n^d)$,

$$T(n) = \begin{cases} O(n^d) & \text{if } k/b^d < 1 \\ O(n^d \log n) & \text{if } k/b^d = 1 \\ O(n^{\log_b k}) & \text{if } k/b^d > 1 \end{cases}$$

Decidability: decider halts on all inputs, always accepts if input is in the language or rejects if input is not in the language.

- Closed under complement
- Closed under union and intersection
- Deterministic Finite Automata (DFAs) and Turing Machines can simulate a decider's behavior
    - DFAs only go forward through string, can have multiple accepting states
    - Turing Machines go forward or backward, overwrite cell, has 1 accept and 1 reject state

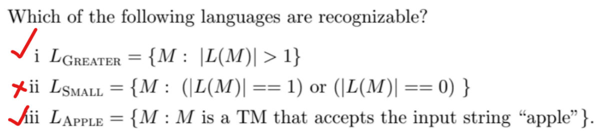


*	previous unit repeated 0 or more times
+	previous unit repeated 1 or more times
|	logical or symbol — one of these two happens
()	operator precedence

(a | bc)* — "abc" is accepted

Recognizability: recognizer accepts on all inputs in the language and either rejects or loops forever on inputs not in the language. All decidable languages are also recognizable.

- Not necessarily closed under complement- if it is, it's decidable
- Closed under union and intersection



Which of the following languages are recognizable?

✓ i $L_{\text{GREATER}} = \{M : |L(M)| > 1\}$

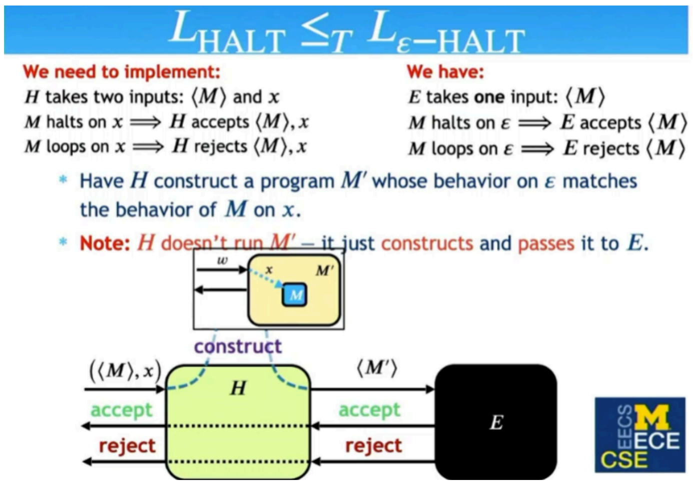
✗ ii $L_{\text{SMALL}} = \{M : (|L(M)| == 1) \text{ or } (|L(M)| == 0)\}$

✓ iii $L_{\text{APPLE}} = \{M : M \text{ is a TM that accepts the input string "apple"}\}$.

Turing Reduction: $A \le_T B$ means if B has a membership oracle (black box), then A is decidable

If $A \le_T B$, and A is undecidable then B must also be undecidable. If B is decidable, A is decidable

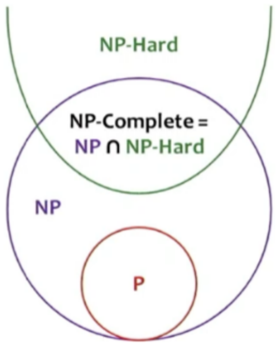
L_{ACC} and L_{HALT} are two undecidable languages that can be used in Turing reduction proofs. Example:



$L_{\text{HALT}} \le_T L_{\varepsilon\text{-HALT}}$

We need to implement:	**We have:**
H takes two inputs: $\langle M\rangle$ and x	E takes **one** input: $\langle M\rangle$
M halts on $x \Longrightarrow H$ accepts $\langle M\rangle, x$	M halts on $\varepsilon \Longrightarrow E$ accepts $\langle M\rangle$
M loops on $x \Longrightarrow H$ rejects $\langle M\rangle, x$	M loops on $\varepsilon \Longrightarrow E$ rejects $\langle M\rangle$

- Have H construct a program M' whose behavior on ε matches the behavior of M on x.
- **Note:** H doesn't run M' — it just constructs and passes it to E.



$L \in P$ means L has an efficient decider that can determine whether an input *x* is in or not in *L* in polynomial time

$L \in NP$ if for every $x \in L$ there is an efficient (poly-time) verifier V such that

$x \in L \iff V(x, c)$ accepts for some certificate *c* (a proposed solution/proof that *x* is in *L*)
$x \text{ is not in } L \iff V(x, c)$ rejects for all *c*

V(x , c) accepts if c is a valid solution which shows that x is in L
V(x , c) rejects if c is not a valid solution does not show that x is in L

An example of a language in NP is the following (note search problem was recast as a decision problem):

Problem: Given an (integer-)weighted graph G, vertices s and t, and $z \in \mathbb{Z}$, is there an $s \to t$ path of length at most z?

* $L_{path} = \{\langle G, s, t, z\rangle : G \text{ has an } s \to t \text{ path of length } \leq z\}$

A certificate for this would be a set of vertices which results in a path from s to t that is less than z in length

Poly-time reduction (NP-completeness proof is likely to be on the exam)
$A \leq_p B$ reads A is polytime reducible to B. This means if you have a poly-time algorithm for *B*, then you have a polytime algorithm for *A* as well.
if $A \leq_p B$ and $B \leq_p C$, then $A \leq_p C$
if $A \leq_p B$ and $B \in P$, then $A \in P$
A language *A* is NP-hard if for every language $L \in NP$, $L \leq_p A$ (*L* is "at least as hard" as every language in NP; every language in NP reduces to it)
If $L \in NP - hard$, and $L \leq_p A$, then *A* is NP-hard (in words: a language *A* can be shown to be NP-hard if you can show that a language *L* already known to be NP-hard reduces to it).
Some NP-hard languages include:

- SAT = {b: b is a satisfiable boolean formula}. Cook-Levin theorem is that all languages in NP polytime reduce to this.
- 3SAT (each clause is a set of three literals or'd with each other, and each clause is and'd with each other)
- CLIQUE = {(G, k): G is an undirected graph that has a clique of size k}, where a clique is a set of vertices that has an edge between every other vertex in the clique
- Vertex Cover = {(G,k): G is an undirected graph that has a vertex cover of size k}, where a vertex cover is a set of vertices such that every edge in the graph is connected to at least one vertex in the cover. 3SAT polytime reduces to this.
- Set Cover = {$(S, S_1, S_2, ..., S_n, k)$: $S_i \subseteq S$ for each i, and there is a collection of *k* sets S_i that cover S}, where S_i is the set of skills that worker i has. Vertex Cover polytime reduces to this.
- Hamiltonian Cycle (set of all graphs *G* with a Hamiltonian cycle). HAMPATH polytime reduces to HAMCYCLE.
- Subset-sum (set *S* is a set that has a subset whose elements add up to exactly *k*). 3SAT polytime reduces to this.

A language is NP-complete if it is both NP-hard and in NP itself.

Some randomized algorithms include quicksort (random pivot) and skip lists (including element in next level is random decision)
Expected value:

$$\mathbb{E}[X] = \sum_i a_i \cdot \Pr[X = a_i]$$

Linearity of expectation:

$$\mathbb{E}[\sum_i N_i] = \sum_i \mathbb{E}[N_i].$$

Averaging argument: if the average outcome is X, at least one outcome must be greater than or equal to X.

Chernoff-Hoeffding bound (a bound on the probability that the sum of indicator variables X divided by the number of indicator variables n was ε greater than the expected value of X \ n, *p*

$$\Pr\left[\frac{1}{n}X \geq p + \varepsilon\right] \leq e^{-2\varepsilon^2 n}$$

(10 points) Suppose you have $n \geq 2$ bins. You independently toss n^2 balls at the bins, uniformly at random. You may find formulas on the first page to be helpful.

(a) For each $i \in \{1, \ldots, n\}$, what is the expected number of balls in bin i?

(b) Use Markov's inequality to bound the probability that bin i has at least $2n$ balls.

(c) Use Chernoff-Hoeffding to bound the probability that bin i has at least $2n$ balls.

Solution: (a) Let X be the number of balls in bin i and let X_j be the indicator random variable for whether ball $j \in \{1, \ldots, n^2\}$ lands in bin i. Since each $E[X_j] = 1/n$, by linearity of expectation, $E[X] = \sum_j E[X_j] = n$.

(b) Markov's inequality says that $\Pr(X \geq 2n) \leq E[X]/2n = 1/2$.

(c) Chernoff-Hoeffding says that $\Pr(X \geq 2n) \leq e^{-2n^2/n^2} = e^{-2}$.

Markov's inequality (X is a nonnegative random variable):

$$\Pr[X \geq v] \leq \frac{\mathbb{E}[X]}{v}$$

$a \equiv b^{-1} mod(n)$

means $a * b^{-1} \equiv 1\, mod(n)$ or $a * b - 1$ is a multiple of n. a is the inverse of b in mod n. b only has an inverse in mod n if b is coprime to n, meaning b and n have no common factors (besides 1).

Fermat's little theorem: if you have a prime number *p* and an integer *a* such that $1 \leq a \leq p - 1$, then $a^{p-1} \equiv 1\ (mod\ p)$

Also know:

$a^k\ (mod\ p)\ =\ (a\ (mod\ p))^k$

Solution: Since 13 is prime we know that $a^{12} \equiv 1 \pmod{13}$ by Fermat's little theorem. Therefore $5^{146} \pmod{13} \equiv 5^{144} * 5^2 \pmod{13} \equiv (5^{12})^{12} * 5^2 \pmod{13} \equiv 5^2 \pmod{13} \equiv 25 \pmod{13} \equiv 12$

$Z_p = \{0, 1, \ldots, p - 1\}$

A generator *g* of Z_p where *p* is prime is an element of Z_p such that for every nonzero element *x* of Z_p (aka Z_p^+) there exists a number *i* such that $g^i \equiv x\ (mod\ p)$

Diffe-Helman protocol:

Public values are p (preferably a large prime number), g (a generator for Z_p), A, and B

Private values are a, b, and k

- Alice picks a random $a \in \mathbb{Z}_p^+$ as her private key, computes $A \equiv g^a \pmod{p}$ as her public key, and sends A t Bob over the insecure channel.
- Bob picks a random $b \in \mathbb{Z}_p^+$ as his private key, computes $B \equiv g^b \pmod{p}$ as his public key, and sends B to Alice over the insecure channel.
- Alice computes

$$k \equiv B^a \equiv (g^b)^a \equiv g^{ab} \pmod{p}$$

  as the shared secret key.

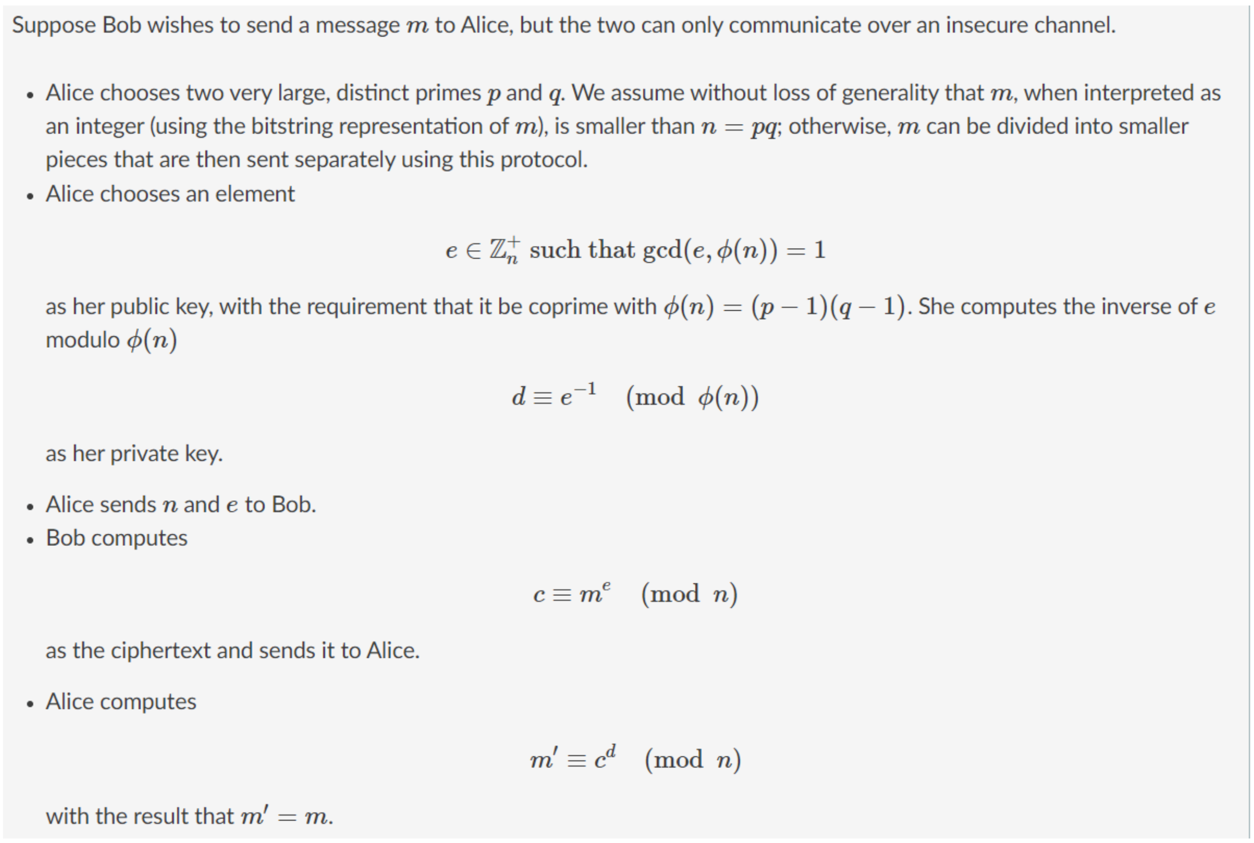
- Bob computes

$$k \equiv A^b \equiv (g^a)^b \equiv g^{ab} \pmod{p}$$

  as the shared secret key.

The assumption is there is no efficient algorithm to calculate the shared key *k* without knowing either *a* or *b* in addition to the public information.

RSA protocol:
Public values are e (public exponent that has no common factors with (p - 1)(q - 1)), n (product of primes), m^e(mod n)
Private values are d (private exponent), p, and q (secret primes)



Suppose Bob wishes to send a message m to Alice, but the two can only communicate over an insecure channel.

- Alice chooses two very large, distinct primes p and q. We assume without loss of generality that m, when interpreted as an integer (using the bitstring representation of m), is smaller than $n = pq$; otherwise, m can be divided into smaller pieces that are then sent separately using this protocol.
- Alice chooses an element

$$e \in \mathbb{Z}_n^+ \text{ such that } \gcd(e, \phi(n)) = 1$$

as her public key, with the requirement that it be coprime with $\phi(n) = (p-1)(q-1)$. She computes the inverse of e modulo $\phi(n)$

$$d \equiv e^{-1} \pmod{\phi(n)}$$

as her private key.

- Alice sends n and e to Bob.
- Bob computes

$$c \equiv m^e \pmod{n}$$

as the ciphertext and sends it to Alice.

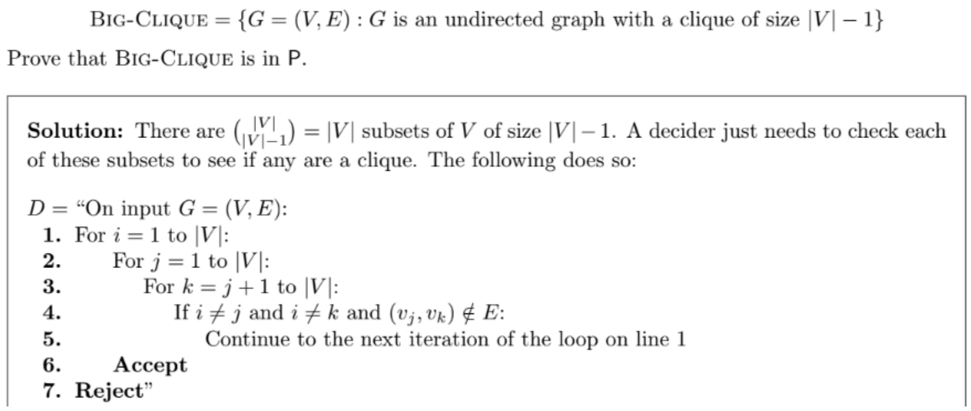
- Alice computes

$$m' \equiv c^d \pmod{n}$$

with the result that $m' = m$.

The assumption is there is no efficient algorithm to factor *n* into the product of two prime numbers.
d can be efficiently computed using Euclid's algorithm if *e* is known.

RSA signature $s = m^d$; the signature can be verified by showing that $s^e \equiv m \ (mod\ n)$



$$\text{BIG-CLIQUE} = \{G = (V, E) : G \text{ is an undirected graph with a clique of size } |V| - 1\}$$

Prove that BIG-CLIQUE is in P.

Solution: There are $\binom{|V|}{|V|-1} = |V|$ subsets of V of size $|V| - 1$. A decider just needs to check each of these subsets to see if any are a clique. The following does so:

```
D = "On input $G = (V, E)$:
 1. For $i = 1$ to $|V|$:
 2.     For $j = 1$ to $|V|$:
 3.         For $k = j + 1$ to $|V|$:
 4.             If $i \neq j$ and $i \neq k$ and $(v_j, v_k) \notin E$:
 5.                 Continue to the next iteration of the loop on line 1
 6.     Accept
 7. Reject"
```

Give correctness and runtime analysis (show it is O(polynomial)) to show it is a decider and it is efficient

. You are planning to drive from Michigan to Southern California for the CFP National Championship game, and there are a number of sights you'd like to see along the way (e.g. Yellowstone, The Grand Canyon, Death Valley). You'd like to see as many of these places as possible, without having to backtrack (traverse a cycle) during your trip.

Formally, we define the following language:

$$\text{Road-Trip} = \left\{ (G = (V, E), s, t, S, k) : \begin{array}{l} G \text{ has a simple path from } s \text{ to } t \text{ that} \\ \text{visits at least } k \text{ of the vertices in } S \end{array} \right\}$$

(A *simple path* is a path without a cycle.)

Given an efficient decider D for Road-Trip, design and analyze (both correctness and runtime) an efficient algorithm that given $(G = (V, E), s, t, S)$, finds an actual simple path (i.e. a path without a cycle) from s to t that goes through as many vertices of S as possible. The path should be returned as a set of edges (the edges do **not** have to be ordered). You may assume that $\{s, t\} \subseteq V$ and $S \subseteq V$.

Solution: We follow the standard process of first finding the size of an optimal object, then finding the object itself. On input (G, s, t, S), we first run the decider on (G, s, t, S, k) for each k between 0 and $|S|$ to find the largest k for which the decider accepts.

Second, we determine which edges are essential for the maximal path. For each edge e, we use the decider to determine whether removing e from the graph causes the graph to no longer have a path that goes through the maximal number of vertices of S – if so, the edge is part of that path. Otherwise, it is not, and we can safely remove e and move on to the next edge.

The full search algorithm is as follows:

```
$\mathcal{A}$ = "On input $(G = (V, E), s, t, S)$:
1. $k \leftarrow 0$
2. While $D(G, s, t, S, k + 1)$ accepts:
3.     $k \leftarrow k + 1$
4. For each edge $e \in E$:
5.     $E' \leftarrow E \setminus \{e\}$
6.     If $D(G' = (V, E'), s, t, S, k)$ accepts:
7.         $E \leftarrow E'$
8. Return $E$"
```

(a) Provide an efficient verifier for Road-Trip. You do **not** need to provide analysis for this part.

(b) Prove that Road-Trip is NP-Hard.

Solution:

(a) The following is an efficient verifier V for Road-Trip.

```
$V$ = "On input $((G = (V, E), s, t, S, k), (v_1, \ldots, v_m))$:
1. If $v_1 \neq s$ or $v_m \neq t$ or $\{v_1, \ldots, v_m\} \not\subseteq V$ or $k \notin [0, |V|]$, **reject**
2. If there are any duplicates in $(v_1, \ldots, v_m)$, **reject**
3. $count \leftarrow 0$
4. For $i = 1$ to $m - 1$:
5.     If $(v_i, v_{i+1}) \notin E$, **reject**
6.     If $v_i \in S$, $count \leftarrow count + 1$
7. If $v_m \in S$, $count \leftarrow count + 1$
8. If $count \geq k$, **accept**
9. Else **reject**"
```

(b) Next, we show Road-Trip is NP-Hard. To do this, we will show Hamiltonian-Path $\leq_p$ Road-Trip. Recall that

$$\text{Hamiltonian-Path} = \{(G, s, t) : G \text{ is a graph with a Hamiltonian path from } s \text{ to } t\}$$

and that Hamiltonian-Path is NP-Complete. We construct the following mapping function.

```
$f$ = "On input $(G = (V, E), s, t)$:
1. Return $(G, s, t, V, |V|)$."
```

We now show f is a valid polynomial mapping function from Hamiltonian-Path to Road-Trip.

f runs in polynomial time, as it just copies the input to the output, with the addition of $|V|$ which can be computed efficiently.

($\Rightarrow$) Suppose $(G = (V, E), s, t) \in$ Hamiltonian-Path. Then by definition, there is a simple path $(s, v_2, \ldots, v_{n-1}, t)$ that goes through all $|V|$ vertices in G. Thus, G has a simple path from s to t that goes through $|V|$ vertices of V, so $(G, s, t, V, |V|) \in$ Road-Trip.

($\Leftarrow$) Suppose $(G = (V, E), s, t, V, |V|) \in$ Road-Trip. Then by definition, there is a simple path $(s, v_2, \ldots, v_{n-1}, t)$ that goes through $|V|$ vertices of V. This path goes through all of the vertices of G, so it is a Hamiltonian path from s to t. Therefore $(G, s, t) \in$ Hamiltonian-Path.

f is a valid, efficient mapping function, so Road-Trip is NP-Hard