Computational Models - Lecture 11[1]

Handout Mode

Iftach Haitner and Yishay Mansour.

Tel Aviv University.

May 7/9, 2012

[1] Based on frames by Benny Chor, Tel Aviv University, modifying frames by Maurice Herlihy, Brown University.

Talk Outline

- Reminder – deterministic and nondeterministic time classes
- Additional NP languages
- The class co-$\mathcal{NP}$
- $\mathcal{P}$ Verses $\mathcal{NP}$
- NP-completeness
- Satisfiability
- Reductions

- Sipser's book, 7.4–7.5

Reminder – Deterministic Time

Definition 1 (deterministic Time)

A deterministic TM M runs in time t, where $t : \mathbb{N} \mapsto \mathbb{N}$, if for every input w, the number of steps that $M(w)$ uses is at most $t(|w|)$.

Definition 2 (DTIME)

For $t : \mathbb{N} \mapsto \mathbb{N}$, let $\mathrm{DTIME}(t) =$
$\{L \subseteq \Sigma^* :$ L is decided by an $O(t)$-time single tape TM$\}$

Note that $t(|w|)$ running time, is also required for w's not in L.

Definition 3 ($\mathcal{P}$)

$\mathcal{P} = \bigcup_{c \geq 0} \mathrm{DTIME}(n^c)$

Reminder – NonDeterministic Time

Definition 4 (nondeterministic Time)

A non-deterministic TM N runs in time t, where $t : \mathbb{N} \mapsto \mathbb{N}$, if for every input w, the maximum number of steps that $N(w)$ uses on any branch of its computation tree , is at most $t(|w|)$.

Notice that non-accepting branches must reject within $t(|w|)$ steps.

Definition 5 (NTIME)

For $t \mapsto \mathbb{N} \mapsto \mathbb{N}$ let $\mathrm{NTIME}(t) =$
$\{L \subseteq \Sigma^* : \ L$ is decided by an $O(t)$-time single tape NTM$\}$

Definition 6 ($\mathcal{NP}$)

$\mathcal{NP} = \bigcup_{c \geq 0} \mathrm{NTIME}(n^c)$

Reminder – NonDeterministic Time, cont.

Definition 7 (verifier)

A verifier for a language L, is an algorithm V with
$L = \{w : V \text{ accepts } \langle w, c \rangle \text{ for some string } c\}$.

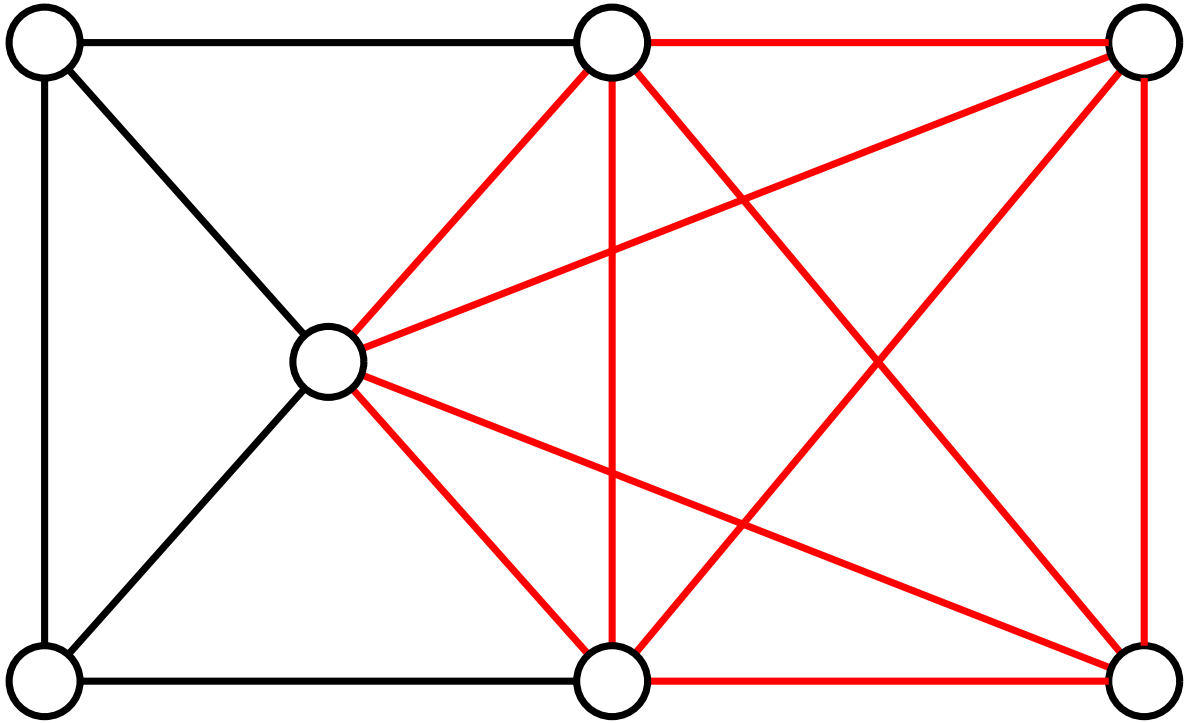
Theorem 8

A language is in $\mathcal{NP}$ iff it has a polynomial time verifier.

Section 1

Additional NP Languages (Monday)

CLIQUE



A clique in a graph is a subgraph where every two nodes are connected by an edge.

A k-clique is a clique of size k.

Question 9

What is the largest k-clique in the figure?

CLIQUE **cont.**

$$\text{CLIQUE} = \{\langle G, k\rangle : G \text{ is an undirected graph with a } k\text{-clique}\}$$

Theorem 10

CLIQUE $\in \mathcal{NP}$

Proof's idea: The clique is the certificate.

Algorithm 11 (V)

On input $(\langle G, k\rangle, c)$
`Accept` if c is a k-clique subgraph of G;
Otherwise `Reject`.

SUBSET-SUM

- A collection of non-negative integers $x_1, \ldots, x_k$
- A target number t

Question: does some subcollection add up to t?

Example 12

- $(\{4, 11, 16, 21, 27\}, 25) \in$ SUBSET-SUM because $4 + 21 = 25$.
- $(\{4, 11, 16, 21, 27\}, 26) \notin$ SUBSET-SUM (why?)

SUBSET-SUM cont.

$$\text{SUBSET-SUM} = \{\langle S = \{x_1, \ldots, x_k\}, t\rangle : \exists\{y_1, \ldots, y_\ell\} \subseteq S : \sum_{j=1}^{\ell} y_j = t\}$$

Collections are sets: repetitions not allowed.

Theorem 13

SUBSET-SUM $\in \mathcal{NP}$

Proof's idea: The subset is the certificate.

Algorithm 14 (V)

On input $(\langle S = \{x_1, \ldots, x_k\}, t\rangle, c)$:
`Accept` if the following holds (otherwise `Reject`):

1. c is a collection of numbers summing to t.
2. c is a subset of S

Pseudo-polynomial algorithm for SUBSET-SUM

Theorem 15

There is a pseudo-polynomial algorithm for SUBSET-SUM

- Running time $O(|S| \cdot t)$
- Not $\text{poly}(\log_2 t)$ but $\text{poly}(t)$ (i.e., pseudo-polynomial)

Proof's idea: By a reduction from SUBSET-SUM to PATH.

Given input $\langle S = \{x_1, \ldots, x_k\}, t\rangle$, create an input $\langle G = (V, E), s, t\rangle$ to PATH of the form:

- $V = V_0 \dot\cup V_1 \ldots \dot\cup V_k$
- $V_i = \{v_{i,0}, \ldots, v_{i,t}\}$
- $E = \{(v_{i-1,j}, v_{i,j}), (v_{i,j}, v_{i,j+x_i}) : \ i \in \{1, \ldots, k\}, j \in \{0, \ldots, t\}\}$
- $s = v_{0,0}$ and $t = v_{k,t}$.

Size of G is $O(|S| \cdot t)$, and therefore the resulting algorithm for SUBSET-SUM runs in time $\text{poly}(|S| \cdot t)$.

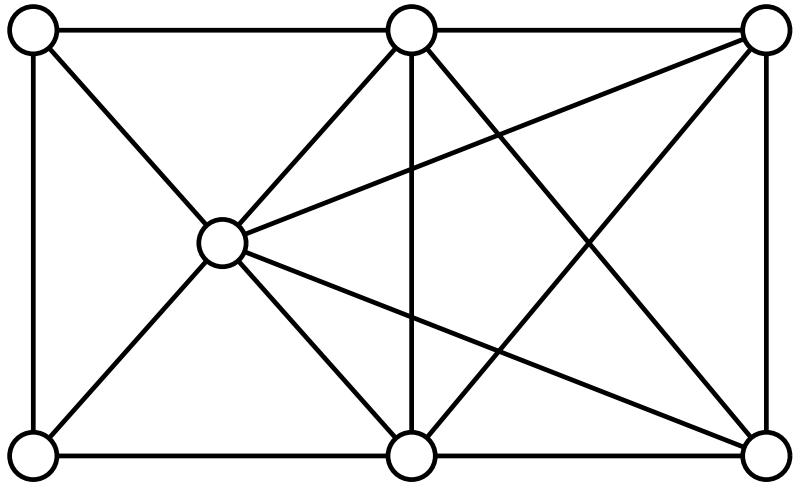
Claim 16

$v_{i,j}$ is reachable from $v_{0,0}$, iff $\langle\{x_1, \ldots, x_i\}, j\rangle \in$ SUBSET-SUM

Section 2

Additional NP Languages (Wednesday)

Independent Set



An independent set in a graph is a set of vertexes, no two of which are linked by an edge.

A k-IS is an independent set of size k.

Question 17

What is the largest k-IS in the figure?

IND-SET cont.

$$\text{IND-SET} = \{\langle G, k\rangle : G \text{ contains an independent set of size } k\}$$

Theorem 18

IND-SET $\in \mathcal{NP}$

Proof's idea: The independent set is the certificate.

Algorithm 19 (V)

On input $(\langle G, k\rangle, c)$
`Accept` if c is a k-IS of G (no edges between nodes in c, and $|c| = k$);
Otherwise `Reject`.

KNAPSACK

- A collection of non-negative integers $x_1, \ldots, x_k$ (size)
- A collection of non-negative integers $y_1, \ldots, y_k$ (benefit)
- A capacity B
- A target number t

Question: does some $S \subset [1, k]$ has $\sum_{i \in S} x_i \leq B$ and $\sum_{i \in S} y_i \geq t$?

Example 20

- $(\{4, 11, 16, 21, 27\}, \{7, 3, 9, 5, 35\}, B = 20, t = 16) \in KNAPSACK$ because $S = \{1, 3\}$.

KNAPSACK cont.

Theorem 21

$KNAPSACK \in \mathcal{NP}$

Proof's idea: The subset is the certificate.

Algorithm 22 (V)

On input $(\langle \{x_1, \ldots, x_k\}, \{y_1, \ldots, y_k\}, B, t \rangle, c)$:
`Accept` if the following holds (otherwise `Reject`):

1. $\sum_{i \in c} x_i \leq B$
2. $\sum_{i \in c} y_i \geq t$.
3. c is a subset of $[1, \ldots, k]$

Pseudo-polynomial algorithm for KNAPSACK

Theorem 23

There is a pseudo-polynomial algorithm for KNAPSACK

- Running time $O(k \cdot t)$
- Not $\text{poly}(\log_2 t)$ but $\text{poly}(t)$ (i.e., pseudo-polynomial)

Algorithm 24 (KNAPSACK)

$A(0,0) = 0$ and $A(0,p) = \infty$ for $p \neq 0$.
For $i = 1$ to k DO
For $p = 0$ to t DO
$A(i,p) = \min\{A(i-1,p), x_i + A(i-1,p-y_i)\}$
END
Test $t \leq \max\{p | A(k,p) \leq B\}$.

Section 3

The Class coNP

The Class co-$\mathcal{NP}$

$\overline{\text{CLIQUE}}$ and $\overline{\text{SUBSET-SUM}}$ seem not to be members of $\mathcal{NP}$.

It is harder to efficiently verify that something does not exist than to efficiently verify that something does exist..

Definition 25 (co-$\mathcal{NP}$)

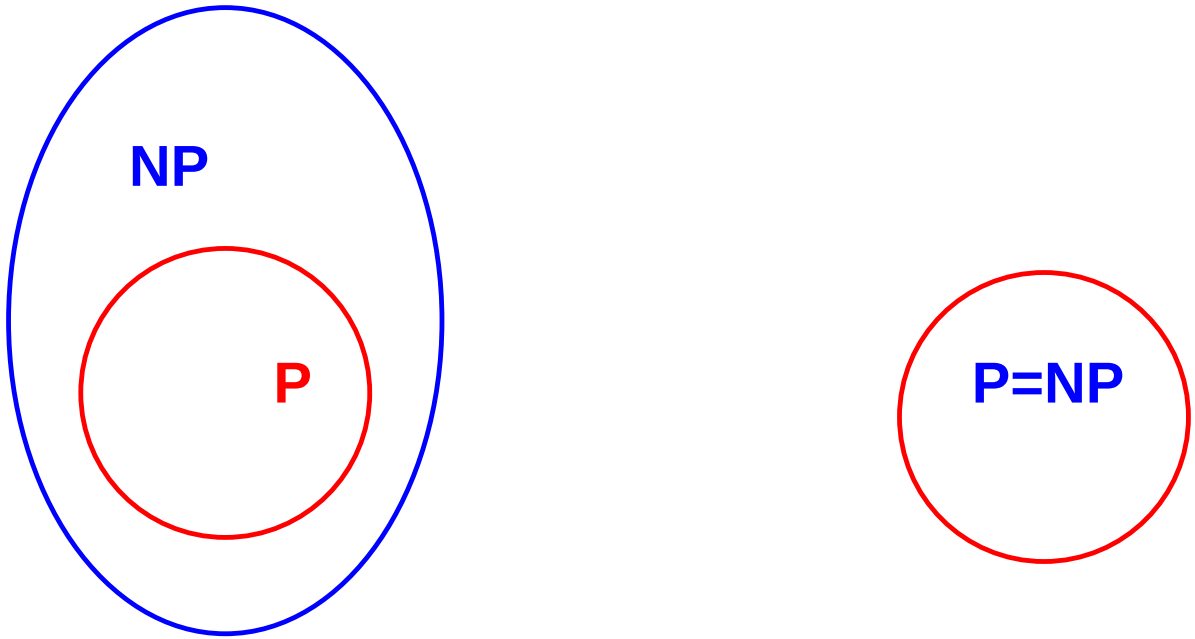
co-$\mathcal{NP} = \{L : \overline{L} \in \mathcal{NP}\}$.

So far, no one knows if co-$\mathcal{NP}$ is distinct from $\mathcal{NP}$.

Section 4

P vs. NP

$\mathcal{P}$ Vs. $\mathcal{NP}$



The question $\mathcal{P} \stackrel{?}{=} \mathcal{NP}$ is one of the great unsolved mysteries in contemporary mathematics.

- Most computer scientists believe the two classes are not equal
- Most bogus proofs show them equal (why?)

Observations

If $\mathcal{P}$ differs from $\mathcal{NP}$, then the distinction between $\mathcal{P}$ and $\mathcal{NP} \setminus \mathcal{P}$ is meaningful and important.

- languages in $\mathcal{P}$ are tractable
- languages in $\mathcal{NP} \setminus \mathcal{P}$ are intractable

Until we can prove that $\mathcal{P} \neq \mathcal{NP}$, there is no hope of proving that a specific language lies in $\mathcal{NP} \setminus \mathcal{P}$.

Nevertheless, we can prove statements of the form

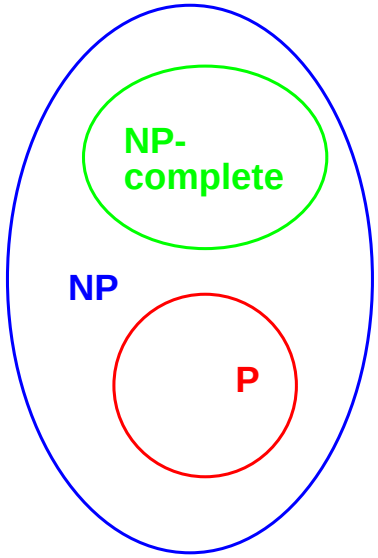
If A $\neq \mathcal{NP}$ *then* B $\in \mathcal{NP} \setminus \mathcal{P}$.

If co-$\mathcal{NP} \neq \mathcal{NP}$ *then* $\mathcal{P} \neq \mathcal{NP}$.

Section 5

NP Completeness

NP Completeness



The class of NP-complete languages are

- "hardest" languages in $\mathcal{NP}$
- "least likely" to be in $\mathcal{P}$
- If any NP-complete L $\in P$, then $\mathcal{NP} = \mathcal{P}$.

Such languages, "carry on their backs" the burden of all of $\mathcal{NP}$.

Question 26

Are there NP-complete languages?

Polynomial-Time Computable Functions

Definition 27 (poly-time computable functions)

A function $f : \Sigma^* \mapsto \Sigma^*$ is polynomial-time computable, if there is a poly-time deterministic TM that

- starts with input w, and
- halts with $f(w)$ on tape.

Polynomial-Time Reducibility

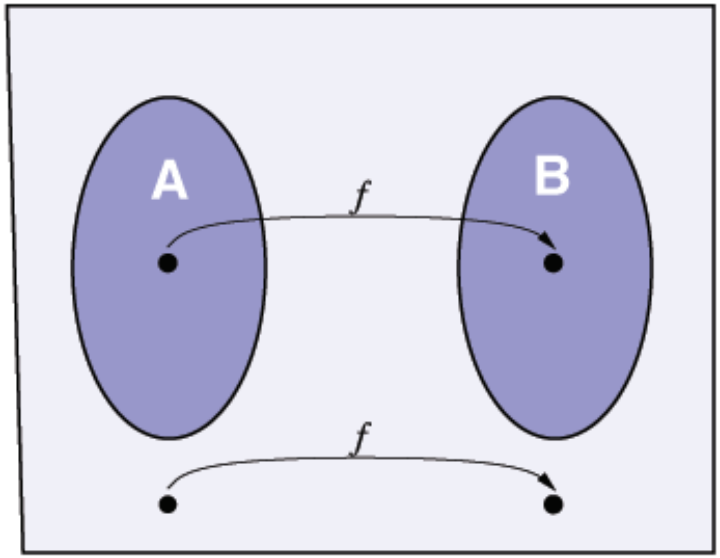
Definition 28

A language A is polynomial time mapping reducible to B, denoted $A \leq_P B$, if exists poly-time computable f such that

$$w \in A \iff f(w) \in B.$$

for every $w \in \Sigma^*$.
The function f is called a polynomial-time reduction from A to B.

The mapping f efficiently converts questions about membership in A to membership in B.



Reductions to $\mathcal{P}$

Theorem 29

If $A \leq_P B$ *and* $B \in \mathcal{P}$ *then* $A \in \mathcal{P}$.

Proof:

- Let f the reduction from A to B, computed by TM M_f. On input x, the TM M_f makes at most $c_f \cdot |x|^{a_f}$ steps.
- Let M_B be the poly-time decider for B. On input y, the TM M_B makes at most $c_B \cdot |y|^{a_B}$ steps.

Algorithm 30 (Decider M_A for A)

On input x, return $M_B(f(x))$

- M_B decides B
- Running time of $M_B(x)$, is at most
  $c_B \cdot (c_f \cdot |x|^{a_f})^{a_B} = (c_B \cdot c_f^{a_B}) \cdot |x|^{a_f \cdot a_B} \in \text{poly}(|x|)$
  Hence, $A \in \mathcal{P}$

Remarks

Question 31

Assume that $\{0^n1^n : n \geq 0\} \leq_P \mathsf{L}$. Does it yield that $\mathsf{L} \in \mathcal{P}$?

$\mathsf{L} = \mathsf{H}_{\mathsf{TM}}$
If $x = 0^n1^n$ then $f(x) = M_{stop}$
Otherwise $f(x) = M_{run}$.

NP Completeness, Formal Definition

Definition 32 ($\mathcal{NP}$-complete)

A language B is NP-complete, if

- B $\in \mathcal{NP}$, and
- Every A $\in \mathcal{NP}$ is poly-time reducible to B

We let $\mathcal{NPC}$ denote the class of all NP-complete languages.

Compare to

Definition 33 (RE-Complete)

A language B is RE-complete, if

- B $\in \mathcal{RE}$, and
- Every A $\in \mathcal{RE}$ is mapping reducible to B.

Why NP Completeness?

Theorem 34

If B $\in \mathcal{NPC}$ *and* B $\in \mathcal{P}$, *then* $\mathcal{P} = \mathcal{NP}$.

Proof: Immediately follows by Thm 29. ♣

To show $\mathcal{P} = \mathcal{NP}$ (and make an instant fortune, see www.claymath.org/millennium/P_vs_NP/), suffices to find a polynomial-time algorithm for any NP-complete problem.

Question 35

Why is $\mathcal{NPC}$Not Empty?

$\mathcal{NPC}$ Is Not Empty

$A_{NP} = \{\langle M, x, 1^n\rangle : M \text{ is a TM} \wedge \exists c \in \Sigma^* \text{ s.t. } M(x,c) \text{ accepts within } n \text{ steps}\}$

Theorem 36

$A_{NP} \in \mathcal{NPC}$

Proof:

- Clearly $A_{NP} \in \mathcal{NP}$
- Let $L \in \mathcal{NP}$, let V be a verifier for L and let $p \in poly$ be a bound on the running time of V (i.e., $V(w,\cdot)$ halts within $p(|w|)$ steps, for every $w \in \Sigma^*$)).
- Define $f(x) = \langle V, x, 1^{p(|x|)}\rangle$.
- Clearly f is poly-time computable and $x \in L \iff f(x) \in A_{NP}$.

♣

Finding Additional NP-complete Languages

Theorem 37

Assume that

1. $B \in \mathcal{NP}$
2. $A \in \mathcal{NPC}$ *and* $A \leq_P B$

then $B \in \mathcal{NPC}$.

Proof: Home exercise . . . ♣

We would like to find $L \in \mathcal{NPC}$ that is “natural" and “easy" to reduce to.

The most useful such language is the language of satisfied formulas.

Section 6

Satisfiability

Boolean Variables

- A Boolean variable assumes values
  - TRUE (written 1), and FALSE (written 0).
- Boolean operations:
  - and: $\wedge$
  - or: $\vee$
  - not: $\neg$
- Examples:

$$\begin{aligned} 0 \wedge 1 &= 0 \\ 0 \vee 1 &= 1 \\ \overline{0} &= 1 \end{aligned}$$

Boolean Formulas and SAT

A Boolean formula is an expression involving Boolean variables and operations.

$$\phi = (\overline{x} \wedge y) \vee (x \wedge \overline{z})$$

Definition 38 (satisfiable formula)

A formula is satisfiable, if some assignment of 0s and 1s to the variables makes the formula evaluate to 1.

The formula $\phi = (\overline{x} \wedge y) \vee (x \wedge \overline{z})$ is satisfiable by the assignment

$$\begin{aligned} x &= 0 \\ y &= 1 \\ z &= 0 \end{aligned}$$

The language of satisfied formulas:

$$\text{SAT} = \{\langle\phi\rangle : \ \phi \text{ is a satisfiable Boolean formula}\}$$

SAT $\in \mathcal{NPC}$

$$\text{SAT} = \{\langle\phi\rangle : \ \phi \text{ is satisfiable Boolean formula}\}$$

Theorem 39 (Cook-Levin (early 70s))

SAT $\in \mathcal{NPC}$.

- The “most important" $\mathcal{NP}$-complete language.
- It is easy to see that SAT $\in \mathcal{NP}$
- For the proof of other part wait for next week . . .

The Language 3SAT

It is useful to consider a special version of SAT

- A literal is a variable or negated variable: x or $\overline{x}$.
- A clause is several literals joined by $\vee$s: $(x_1 \vee \overline{x_2} \vee \overline{x_3})$
- A Boolean formula is in conjunctive normal form (CNF) if it consists of clauses, connected with $\wedge$s.
- For example: $(x_1 \vee \overline{x_2} \vee \overline{x_3} \vee x_4) \wedge (x_3 \vee \overline{x_5} \vee x_6) \wedge (x_3 \vee \overline{x_6})$

Definition 40

A Boolean formula is in k-CNF form, if it is a CNF formula, and all clauses have k literals.

- Example of 3CNF: $(x_1 \vee \overline{x_2} \vee \overline{x_3}) \wedge (x_3 \vee \overline{x_5} \vee x_6) \wedge (x_3 \vee \overline{x_6} \vee x_4)$

The language of satisfied 3CNF formulas:

$$3SAT = \{\langle \phi \rangle : \ \phi \text{ is satisfiable 3CNF formula}\}$$

3SAT $\in \mathcal{NPC}$

- It is trivial that 3SAT $\leq_P$ SAT
- Clearly 3SAT $\in \mathcal{NP}$
- We later show that SAT $\leq_P$ 3SAT, yielding that 3SAT $\in \mathcal{NPC}$.

- Since 3SAT has more structure than SAT, it is simpler to reduce it to other languages
- In the following we prove that several $\mathcal{NP}$ languages are $\mathcal{NPC}$, by showing a reduction from 3SAT to them

1SAT, 2SAT $\in \mathcal{P}$

Question 41

Is 1SAT $\in \mathcal{P}$?

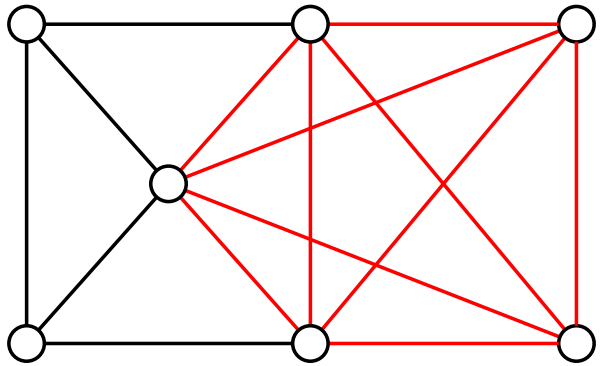
Question 42

Is 2SAT $\in \mathcal{P}$?

Section 7

Clique

3SAT $\leq_P$ CLIQUE



$$\text{CLIQUE} = \{\langle G, k\rangle : G \text{ is an undirected graph with a } k\text{-clique}\}$$

Claim 43

3SAT $\leq_P$ CLIQUE

Since CLIQUE $\in \mathcal{NP}$, it follows that CLIQUE $\in \mathcal{NPC}$

Proof's idea: We'll construct a poly-time reduction f that maps 3CNF formulae ϕ to graphs and numbers $\langle G, k\rangle$.

The function f will have the property that ϕ is satisfiable, iff G has a clique of size k.

Proving 3SAT $\leq_P$ CLIQUE

Let ϕ be a 3CNF formula with k clauses. On going example:

$$(x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_3} \vee \overline{x_5} \vee x_6) \wedge (x_3 \vee \overline{x_6} \vee x_4)$$

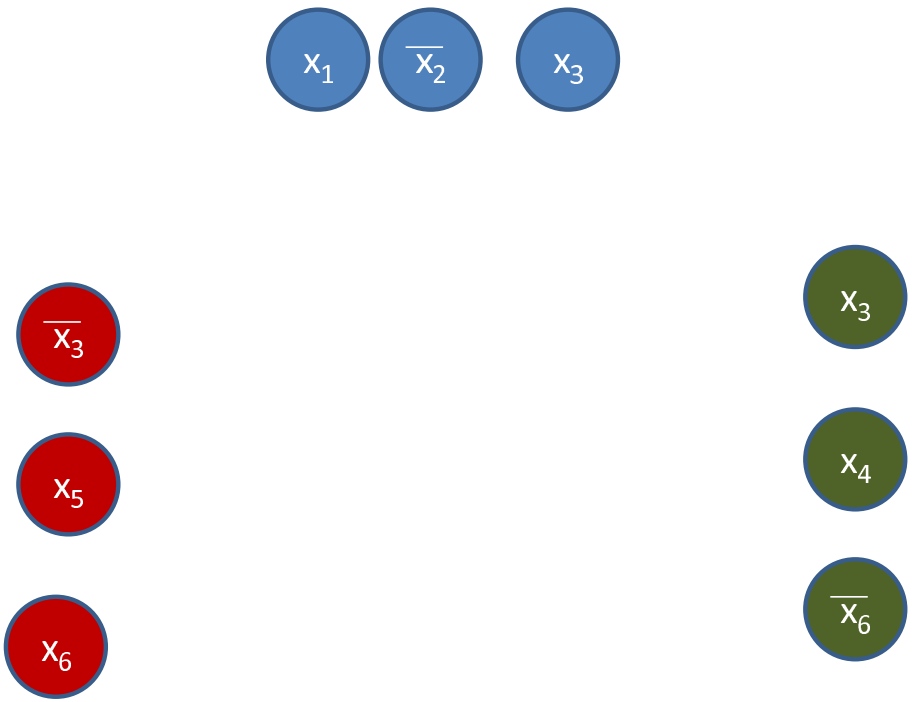
We define a graph $G = G(\phi)$ as follows:

- Nodes in G are organized into triples $t_1, \ldots, t_k$.
- Each triple corresponds to a clause of ϕ
- Each node in a triple corresponds to a literal in corresponding clause.

Nodes of G

$$\phi = (x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_3} \vee \overline{x_5} \vee x_6) \wedge (x_3 \vee \overline{x_6} \vee x_4)$$

Add a node per literal

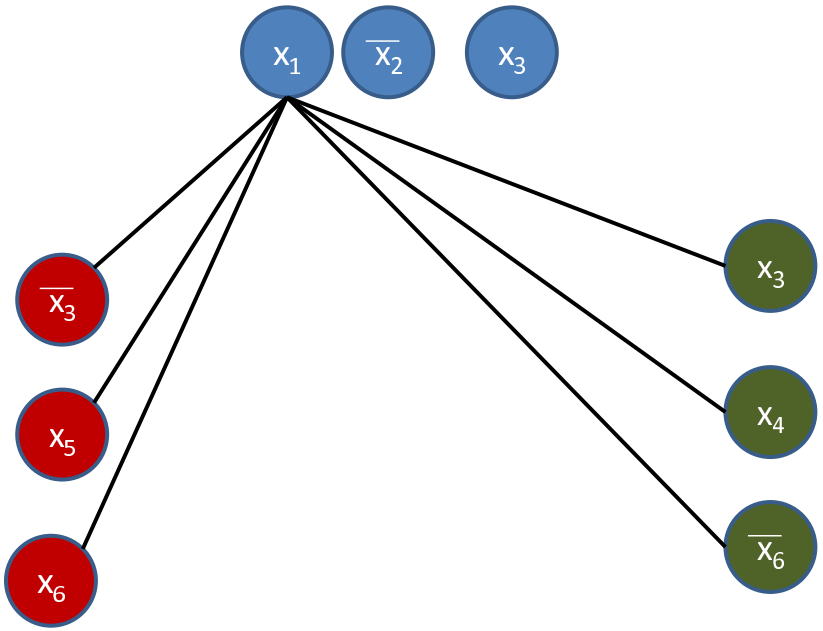


Edges of G

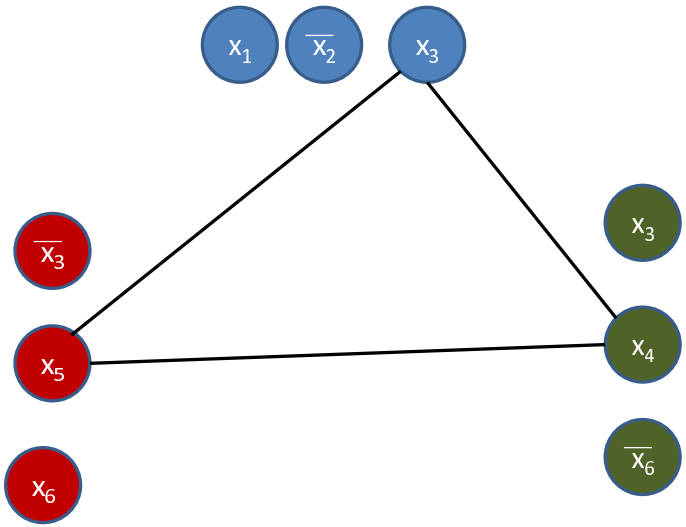
$$\phi = (x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_3} \vee \overline{x_5} \vee x_6) \wedge (x_3 \vee \overline{x_6} \vee x_4)$$

Add edges between all vertex pairs, except

- within same triple
- between contradictory literals (e.g., x_3 and $\overline{x_3}$)



$\phi \in 3SAT \implies \langle G, k \rangle \in CLIQUE$

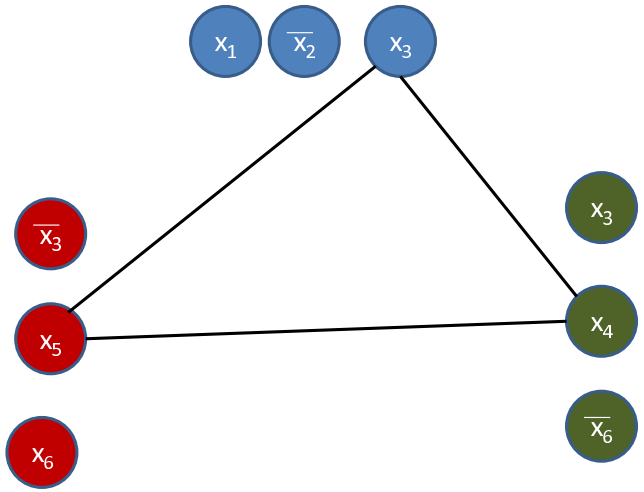


Proof: Suppose ϕ is satisfiable by an assignment ψ.
With respect to ψ:

- At least one literal is assigned to 1 in every clause (why?)
- Select a 1-literal in every tuple;
  These literals can be joined by edges (why?)
  Yielding a k-clique in G. ♣

♣

$\langle G, k \rangle \in$ CLIQUE $\Longrightarrow$ $\phi \in$ 3SAT



Proof: Suppose G has a k-clique.

- No two of the cliques nodes are in the same triple. (why?)
- G has k vertexes and k clauses, so each triple has exactly one clique node.
- Assign 1 to each node in clique;
  Assignment has no contradictions (why?)
  Yielding a satisfying assignment to ϕ.

Recap

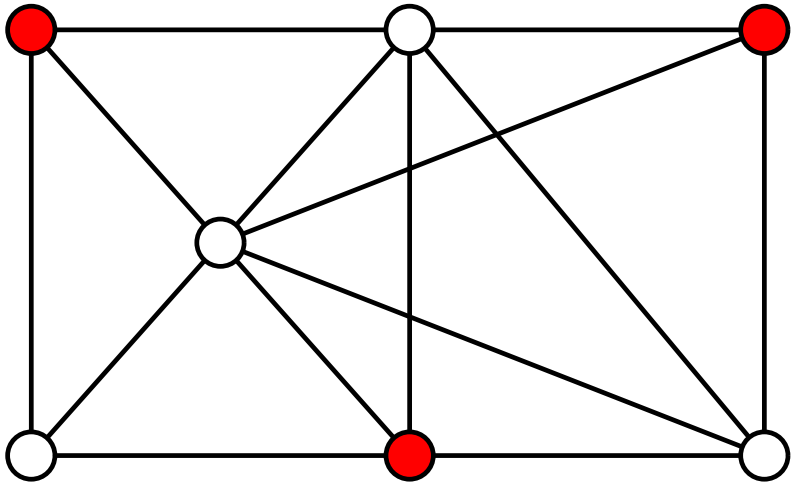
- We've constructed a poly-time computable function f.
- We saw that f has the property that $\phi \in$ 3SAT iff $f(\phi) \in$ CLIQUE.
- Therefore, f is a poly-time reduction from 3SAT to CLIQUE

  $\Longrightarrow$ 3SAT $\leq_P$ CLIQUE. ♣

Section 8

Independent Set

Independent Set



Definition 44

An independent set in a graph is a set of vertexes, no two of which are linked by an edge.

The language of independent sets:

$$\text{IND-SET} = \{\langle G, k\rangle : G \text{ contains an independent set of size } k\}$$

Clearly IND-SET $\in \mathcal{NP}$. We show that CLIQUE $\leq_P$ IND-SET and deduce that IND-SET $\in \mathcal{NPC}$

CLIQUE $\leq_P$ IND-SET

Proof:

Definition 45

The complement of a graph $G = (V, E)$ is a graph $G^c = (V, E^c)$, where $E^c = \{(v_1, v_2) | v_1, v_2 \in V \text{ and } (v_1, v_2) \notin E\}$.

Claim 46

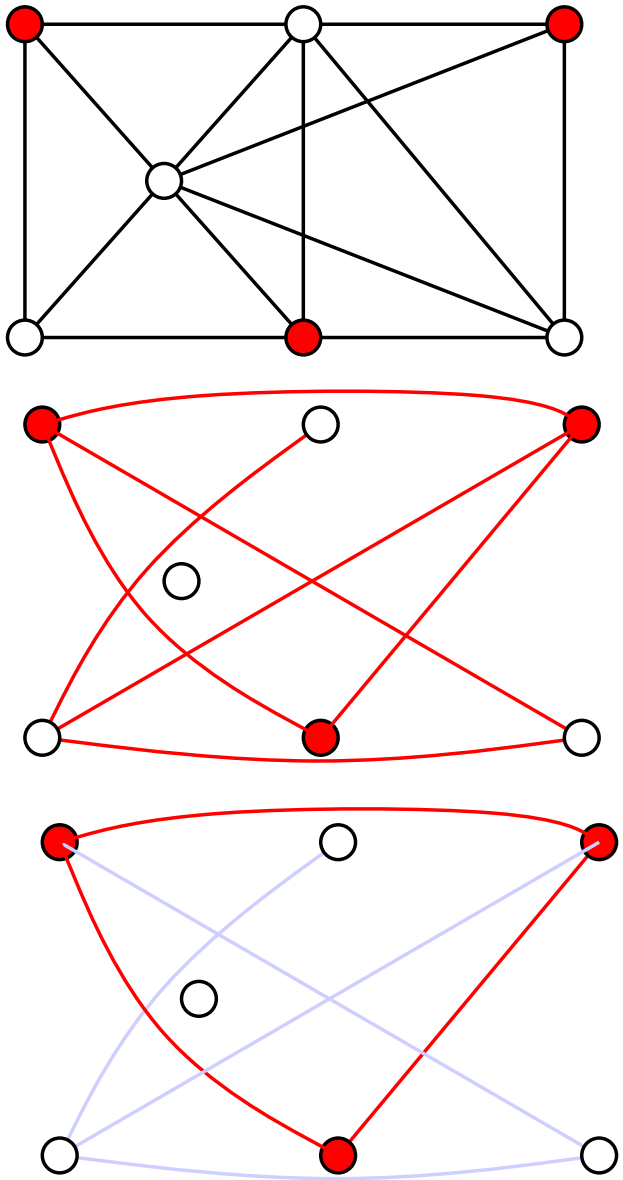
If U is an independent set in G, then U is a clique in G^c.

The reduction from CLIQUE to IND-SET simply compute the complement of the graph. ♣

Remark 47

Same reduction shows that IND-SET $\leq_P$ CLIQUE

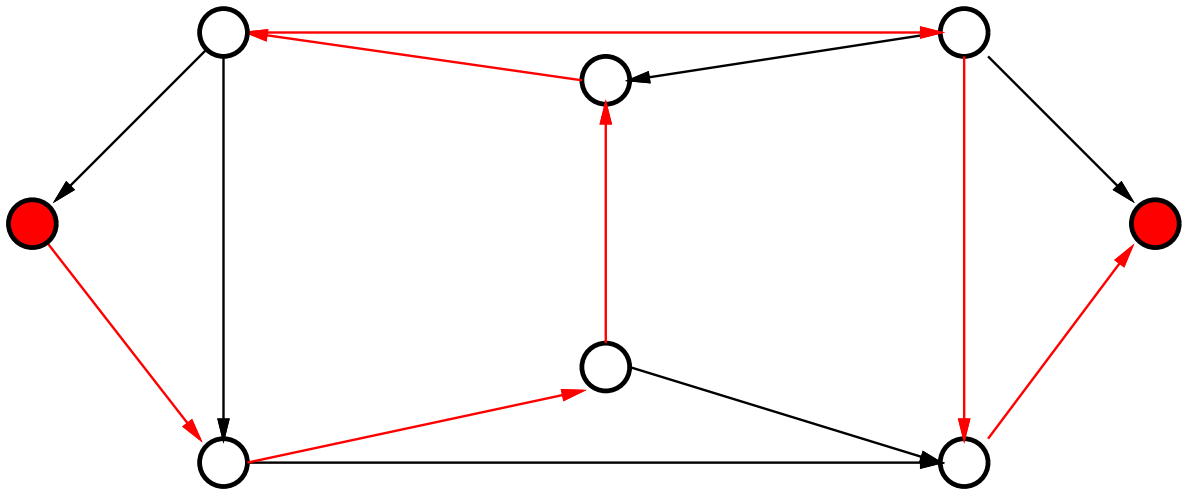
Independent Set $\mapsto$ Clique



Section 9

Hamiltonian Paths and Cycles

Reminder – Hamiltonian Path



A Hamiltonian path in a directed G visits each node exactly once.

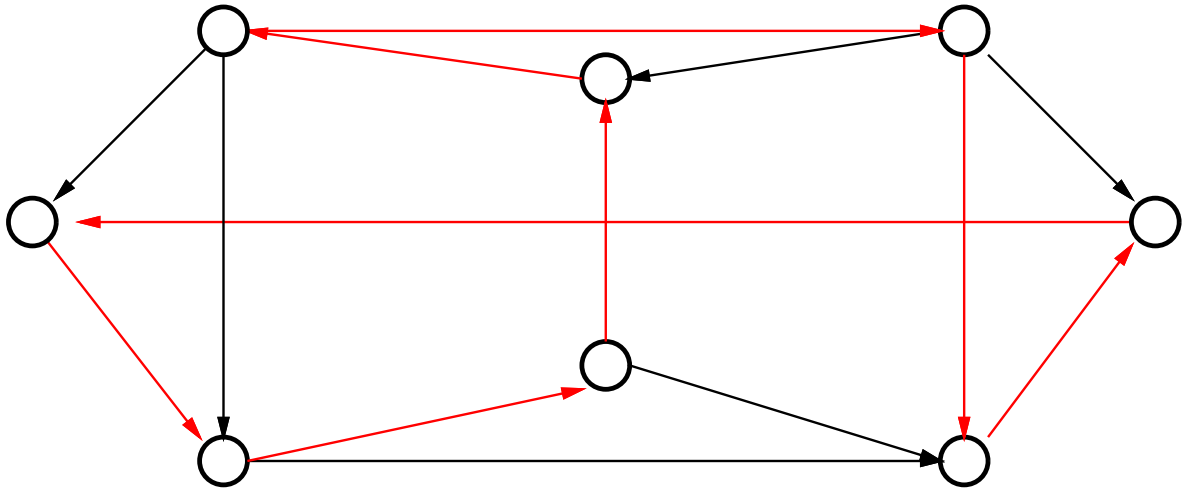
$$\text{HAMPATH} = \{\langle G, s, t\rangle : G \text{ has Hamiltonian path from } s \text{ to } t\}$$

Theorem 48

HAMPATH $\in \mathcal{NPC}$.

Not today . . .

Hamiltonian Circuit



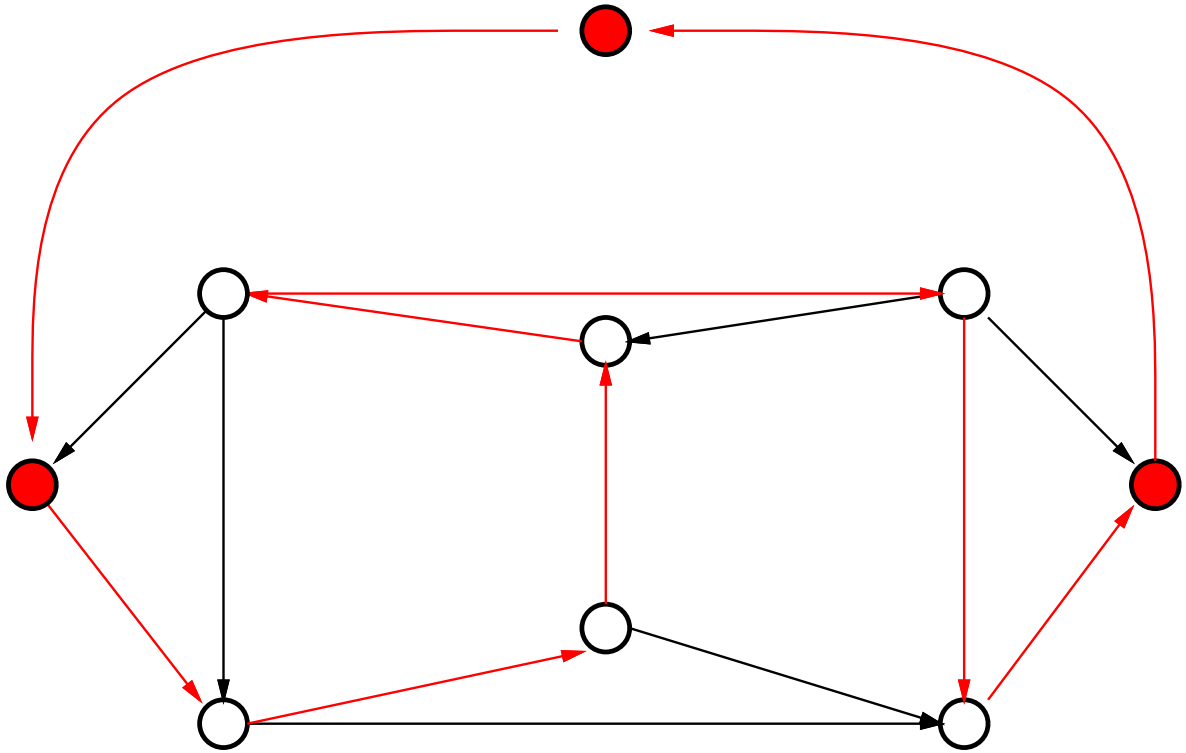
A Hamiltonian Circuit in a directed G visits each node exactly once, and end uo where it started.

$$\text{HAMCIRCUIT} = \{\langle G \rangle : G \text{ has Hamiltonian circuit } \}$$

Clearly HAMCIRCUIT $\in \mathcal{NP}$. We will show that HAMPATH $\leq_P$ HAMCIRCUIT, and deduce that HAMCIRCUIT $\in \mathcal{NPC}$

HAMPATH $\leq_P$ HAMCIRCUIT.

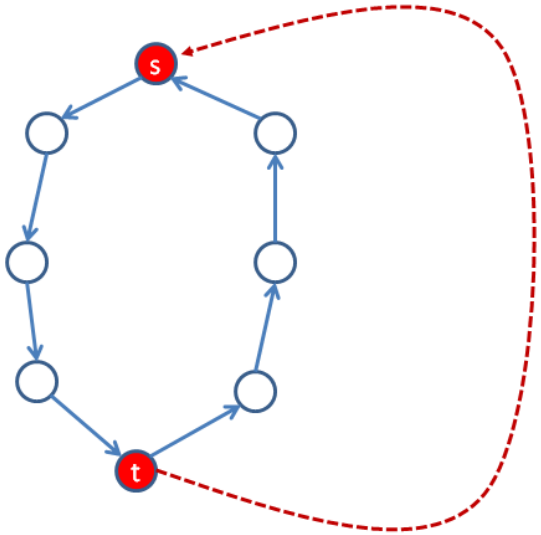
Proof's idea:



Hey, is the new vertex really needed? Why not just add an edge from t to s?

HAMPATH $\leq_P$ HAMCIRCUIT.

Why the new vertex really needed? Why not just add an edge from t to s?



HAMCIRCUIT $\leq_P$ HAMPATH.

Claim 49

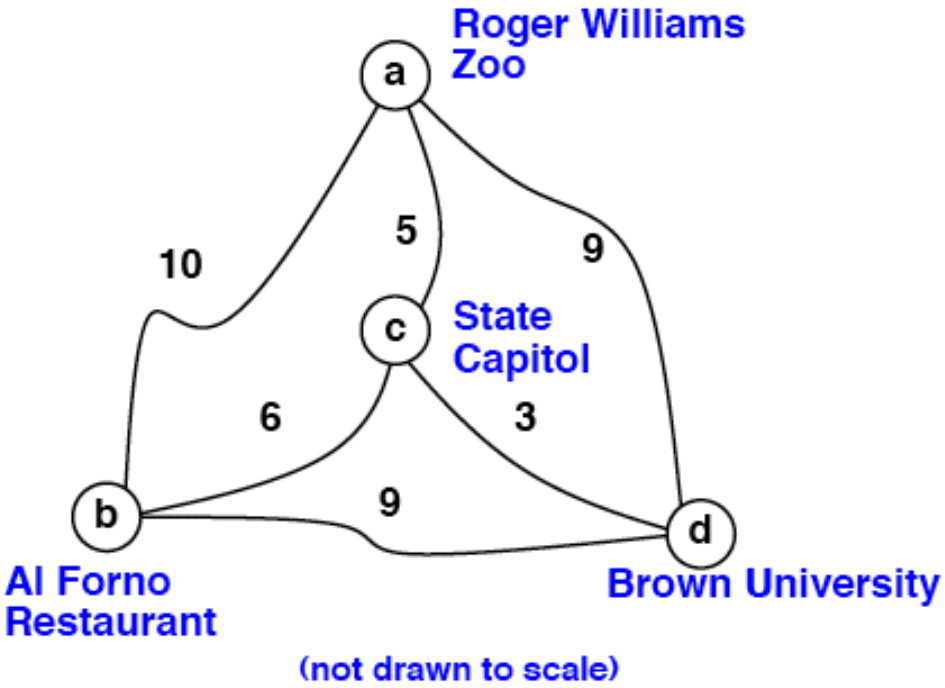
HAMCIRCUIT $\leq_P$ HAMPATH.

Left as an easy (recommended) exercise.

Section 10

Traveling Salesman

Traveling Salesman



Input: set of cities C and set of inter-city distances D

Goal: find a an hamiltonian circuit (TS tour) of total distance at most k

TRAVELING-SALESMAN $=$
$\{\langle C, D, k\rangle : (C, D)$ has a TS tourof total distance $\leq k\}$

HAMCIRCUIT $\leq_P$ TRAVELING-SALESMAN

Proof: Given a directed graph $G = (V, E)$, construct traveling salesman instance.

- The cities are identical to the nodes of the original graph, $C = V$.
- The distance of going from v_1 to v_2 is 1 if $(v_1, v_2) \in E$, and 2 otherwise.
- The bound on the total distance of a tour is $k = |V|$.
- **correctness:**

  $\Longrightarrow$ Suppose G has a Hamiltonian circuit.
  The distance assigned by the reduction to all edges in this circuit is 1. Thus in (C, D) there is a traveling salesman tour of total distance $|V| = k$, namely $(C, D, k) \in$ TRAVELING-SALESMAN.
  $\Longleftarrow$ Suppose (C, D) has a traveling salesman tour of total distance $|V| = k$.
  Tour cannot contain any edge of distance 2. Therefore it gives a Hamiltonian circuit in G.
- **Efficiency:** reduction done in quadratic time (filling up distances for all edges of the complete graph).

Chains of Reductions: NPC Problems