

Analysis and Design of Algorithms



CS3230
C23530

Week 10

Reductions & Intractability

Diptarka Chakraborty

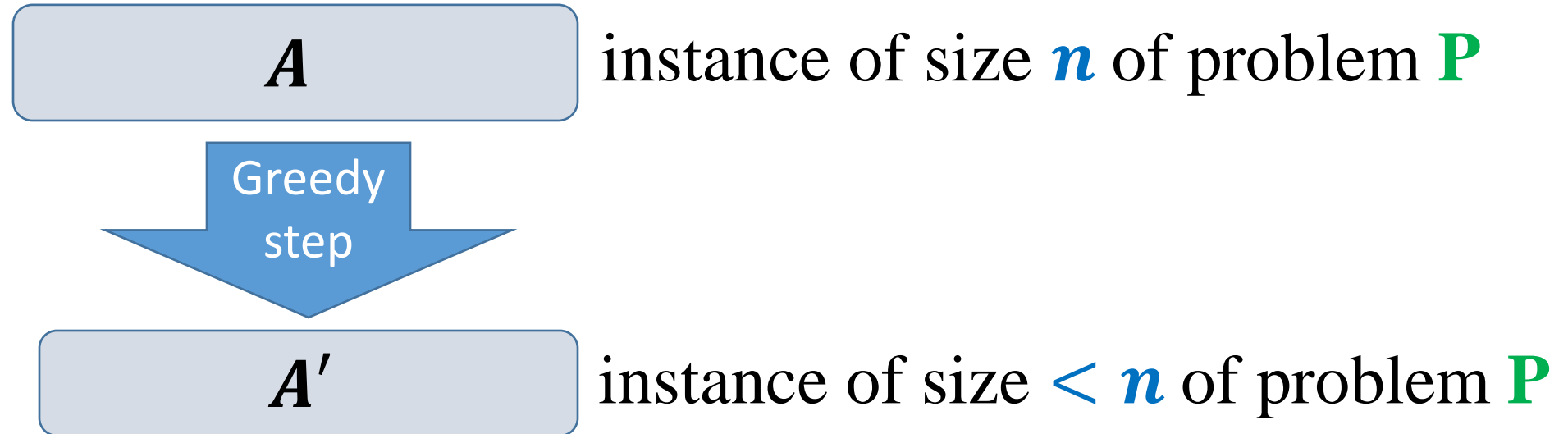
Ken Sung

Paradigm for greedy algorithms (Recap)

1. Cast the problem where we have to **make a choice and are left with one subproblem** to solve.
2. Prove that there is always an **optimal solution to the original problem that makes the greedy choice**, so the greedy choice is safe.
3. Use **optimal substructure** to show that we can combine an optimal solution to the subproblem with the greedy choice to get an optimal solution to the original problem.

To **prove** that a **greedy strategy** works (Recap)

P: A given optimization problem



1. **Try to establish** a relation between **OPT(A)** and **OPT(A')**;
2. **Try to prove** the relation formally by
 - ❑ deriving a (not necessarily optimal) solution of **A** from **OPT(A')**
 - ❑ deriving a (not necessarily optimal) solution of **A'** from **OPT(A)**
3. **If you succeed**, this would give you an algorithm.

Fractional Knapsack

Input:

$(w_1, v_1), (w_2, v_2), \dots, (w_n, v_n)$ and W

Output:

Weights x_1, \dots, x_n that maximize $\sum_i v_i \cdot \frac{x_i}{w_i}$ subject to:

$$\sum_i x_i \leq W \text{ and } 0 \leq x_j \leq w_j \text{ for all } j \in [n].$$

Question 5 (last lecture)

Prove the following: Let j^* be the item with the maximum value/kg, v_j/w_j . Then, there exists an optimal knapsack containing $\min(w_{j^*}, W)$ kgs of item j^* .

Use the above as greedy strategy to design an algorithm for the fractional knapsack problem.

Optimal Substructure

If we remove w kgs of one item j from the optimal knapsack, then the remaining load must be the optimal knapsack weighing at most $W - w$ kgs that one can take from the $n - 1$ original items and $w_j - w$ kgs of item j .

Greedy-choice Property: Why?

Claim: Let j^* be the item with the maximum value/kg, v_j/w_j . Then, there exists an optimal knapsack containing $\min(w_{j^*}, W)$ kgs of item j^* .

Greedy-choice Property: Why?

Claim: Let j^* be the item with the maximum value/kg, v_j/w_j . Then, there exists an optimal knapsack containing $\min(w_{j^*}, W)$ kgs of item j^* .

- Suppose an optimal knapsack contains x_1 kgs of item 1, x_2 kgs of item 2, ..., x_n kgs of item n such that:
$$x_1 + x_2 + \cdots + x_n = \min(w_{j^*}, W)$$

Greedy-choice Property: Why?

Claim: Let j^* be the item with the maximum value/kg, v_j/w_j . Then, there exists an optimal knapsack containing $\min(w_{j^*}, W)$ kgs of item j^* .

- Suppose an optimal knapsack contains x_1 kgs of item 1, x_2 kgs of item 2, ..., x_n kgs of item n such that:
$$x_1 + x_2 + \cdots + x_n = \min(w_{j^*}, W)$$
- Replace this weight by $\min(w_{j^*}, W)$ kgs of item j^* .

Greedy-choice Property: Why?

Claim: Let j^* be the item with the maximum value/kg, v_j/w_j . Then, there exists an optimal knapsack containing $\min(w_{j^*}, W)$ kgs of item j^* .

- Suppose an optimal knapsack contains x_1 kgs of item 1, x_2 kgs of item 2, ..., x_n kgs of item n such that:
$$x_1 + x_2 + \cdots + x_n = \min(w_{j^*}, W)$$
- Replace this weight by $\min(w_{j^*}, W)$ kgs of item j^* .
- Total weight does not change. Total value does not decrease because value/kg of j^* is maximum. So, knapsack stays optimal.

Strategy for Greedy Algorithm

- Use greedy-choice property to put $\min(w_{j^*}, W)$ kgs of item j^* in knapsack.
- If knapsack weighs W kgs, we are done.
- Otherwise, use optimal substructure to solve subproblem where all of item j^* is removed and knapsack weight limit is $W - w_{j^*}$.

Iterative greedy algorithm

ITER-FRAC-KNAPSACK(v, w, W):

$valperkg \leftarrow [1, 2, \dots, n]$

Sort $valperkg$ using comparison operator \preceq where $i \preceq j$ if $\frac{v[i]}{w[i]} \leq \frac{v[j]}{w[j]}$

for $i = n$ to 1:

if $W == 0$: **break**

$j \leftarrow valperkg[i]$

$w \leftarrow \min(w[j], W)$

print “ w kgs of item j ”

$W \leftarrow W - w$

return

Iterative greedy algorithm

ITER-FRAC-KNAPSACK(v, w, W):

$valperkg \leftarrow [1, 2, \dots, n]$

Sort $valperkg$ using comparison operator \preccurlyeq where $i \preccurlyeq j$ if $\frac{v[i]}{w[i]} \leq \frac{v[j]}{w[j]}$

for $i = n$ to 1:

if $W == 0$: **break**

$j \leftarrow valperkg[i]$

$w \leftarrow \min(w[j], W)$

print “ w kgs of item j ”

$W \leftarrow W - w$

return



$O(n \log n)$

Today: Reductions

- Reductions between computational problems is a fundamental idea in algorithm design
- Viewed another way, reductions also give a way to compare the **hardness** of two problems.

What is a reduction?

Consider two problems A and B . A can be solved as follows:

Another word for “input”

Input: An instance α of A

1. Convert α to an instance β of B
2. Solve β and obtain a solution
3. Based on the solution of β , obtain the solution of α

What is a reduction?

Consider two problems A and B . A can be solved as follows:

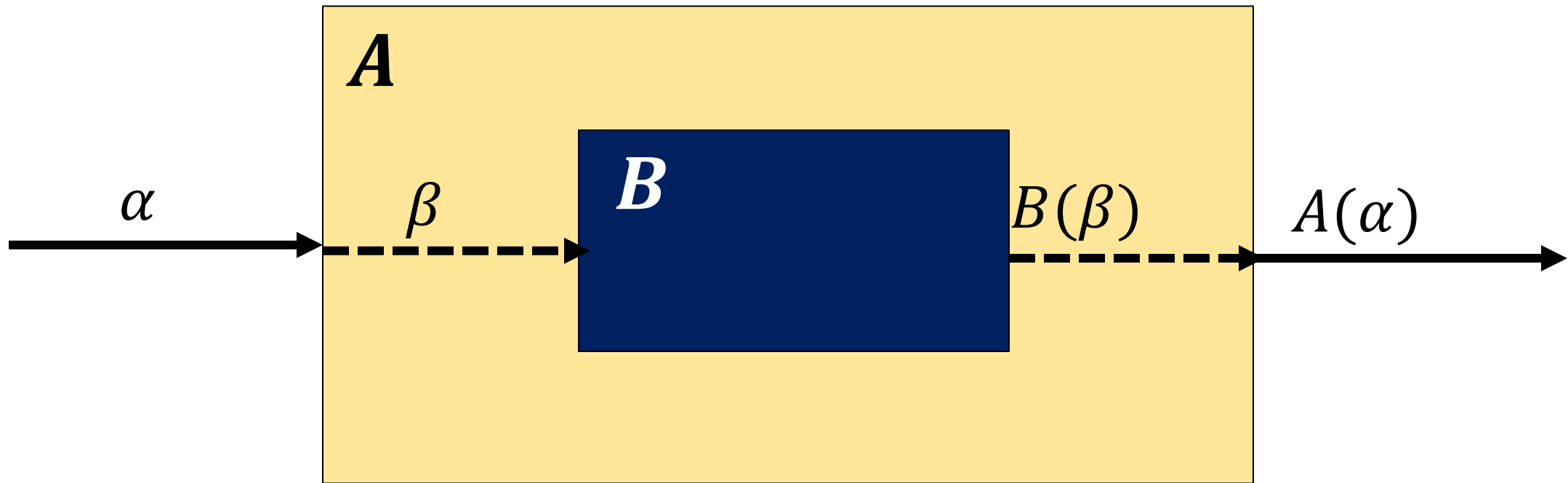
Another word for “input”

Input: An instance α of A

1. Convert α to an instance β of B
2. Solve β and obtain a solution
3. Based on the solution of β , obtain the solution of α

Then, we say A reduces to B .

What is a reduction?



Example

- In the last lecture, we saw how to solve **LIS** using **LCS** algo
 - Sort the input sequence and then find LCS between original and the sorted sequence.
- Two lectures ago, we saw that the problem of finding the **longest palindromic subsequence** in a string reduces to finding the **LCS** of a pair of strings
 - Longest palindromic subsequence of x is an LCS of x and $\text{reverse}(x)$.

Matrix multiplication and squaring

MAT-MULTI

Input:

- ❑ Two $N \times N$ matrices A and B

Output:

- ❑ $A \times B$

MAT-SQR

Input:

- ❑ One $N \times N$ matrix C

Output:

- ❑ C^2

Matrix multiplication and squaring

Claim: MAT-SQR reduces to MAT-MULTI.

Proof: Given input matrix C for MAT-SQR, let $A = C$ and $B = C$ be the inputs for MAT-MULTI. Clearly, $AB = C^2$.

Matrix multiplication and squaring

Claim: MAT-MULTI reduces to MAT-SQR.

Proof: Given input matrices A and B :

Construct:

$$C = \begin{bmatrix} 0 & A \\ B & 0 \end{bmatrix}$$

Call MAT-SQR to get

$$C^2 = \begin{bmatrix} 0 & A \\ B & 0 \end{bmatrix} \begin{bmatrix} 0 & A \\ B & 0 \end{bmatrix} = \begin{bmatrix} AB & 0 \\ 0 & BA \end{bmatrix}$$

Question 1

Consider the following two problems:

0-SUM

Input:

□ An array A of length n

Output:

□ $i, j \in [n]$ such that $A[i] + A[j] = 0$

T-SUM

Input:

□ An array B of length n and number T

Output:

□ $i, j \in [n]$ such that $B[i] + B[j] = T$

Show that T-SUM reduces to 0-SUM.

Question 1: Solution

- **Careful** about which way you do the reduction!!
- Given array B , define array A such that $A[i] = B[i] - T/2$.
- If i, j satisfy $A[i] + A[j] = 0$, then $B[i] + B[j] = T$.

$p(n)$ -time Reduction

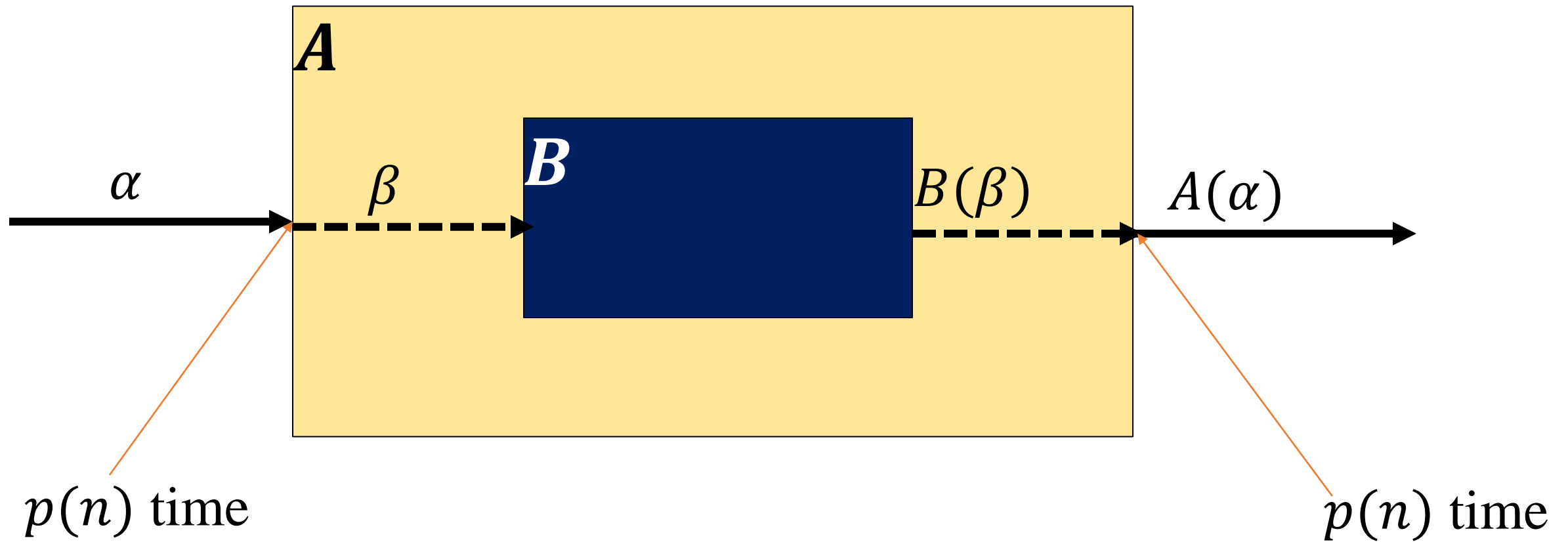
Consider two problems A and B .

If for any instance α of problem A **of size n** :



- An instance β for problem B can be constructed in $p(n)$ time
- A solution to problem A for input α can be recovered from a solution to problem B for input β in time $p(n)$

we say that there is a **$p(n)$ -time reduction from A to B** .

$p(n)$ -time Reduction



Example

- In the last lecture, we saw how to solve **LIS** using **LCS** algo
 - Sort the input sequence and then find LCS between original and the sorted sequence.  **$O(n \log n)$ time reduction**
 - Two lectures ago, we saw that the problem of finding the **longest palindromic subsequence** in a string reduces to finding the **LCS** of a pair of strings
 - Longest palindromic subsequence of x is an LCS of x and $\text{reverse}(x)$.  **$O(n)$ time reduction**

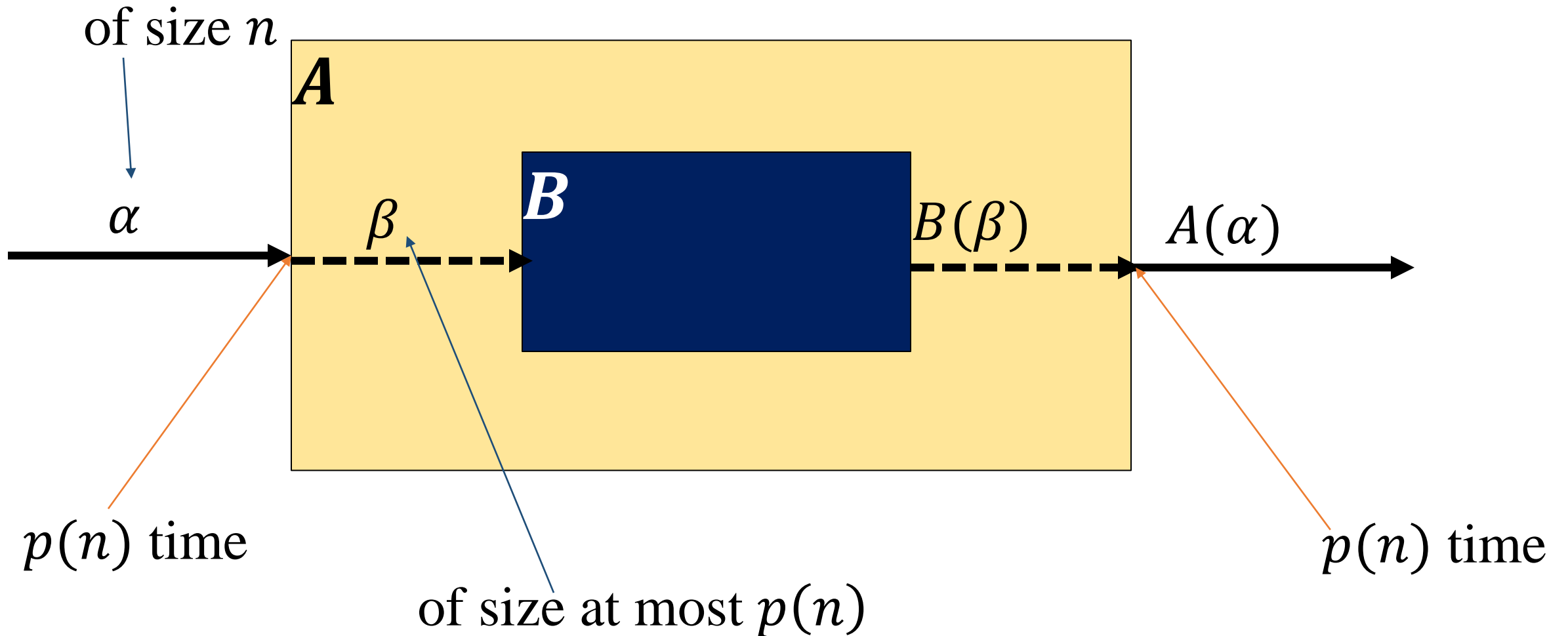
Running Time Composition

Claim: If there is a $p(n)$ -time reduction from problem A to problem B , and there exists a $T(n)$ -time algorithm to solve problem B on instances of size n , then there is a

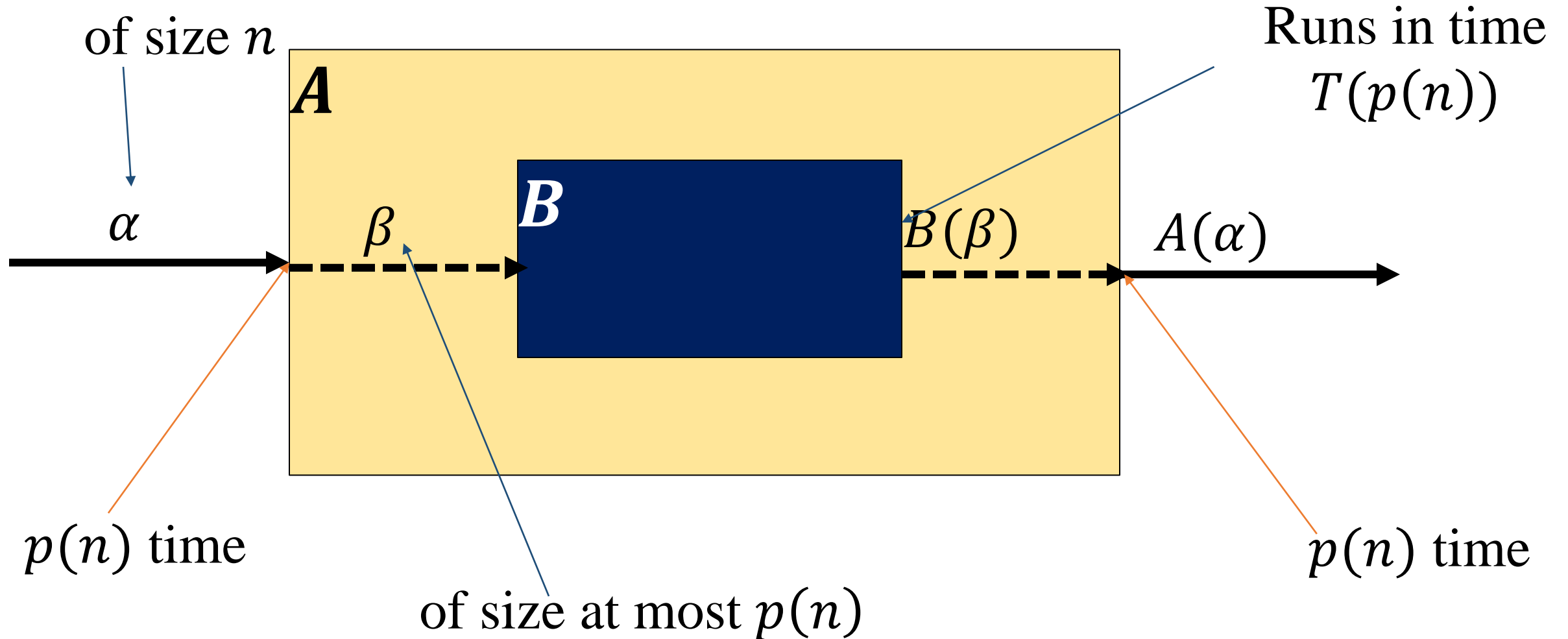
$$T(p(n)) + O(p(n))$$

time algorithm to solve problem A on instances of size n .

Running Time Composition



Running Time Composition



Polynomial-Time Reduction

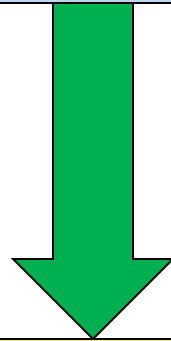
Definition:

$$A \leq_P B$$

If there is a $p(n)$ -time reduction from A to B for some polynomial function $p(n) = O(n^c)$ for some constant c .

Poly-time Reduction

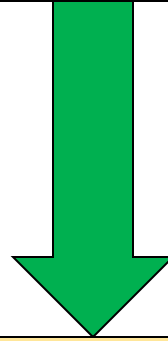
$$A \leq_P B$$



If B has a polynomial time algorithm, then so does A !

Poly-time Reduction

$$A \leq_p B$$



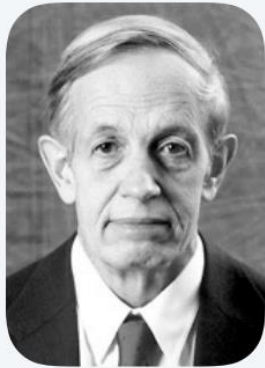
If B is “easily solvable”, then so is A !

Why Poly-Time?

A working definition of problems solvable in practice is that they have polynomial time algorithms using “standard” computing hardware.



von Neumann
(1953)



Nash
(1955)



Gödel
(1956)



Cobham
(1964)



Edmonds
(1965)



Rabin
(1966)

Why Poly-Time?

- Notion is broad and robust even if computing model/hardware is changed “reasonably”
- Usually, poly-time algorithms for real-life problems have runtime $O(n)$ or $O(n^2)$ or $O(n^3)$, not $O(n^{100})$.

A note on encoding

- For polynomial time, we mean that the runtime is polynomial in the **length of the encoding of the problem instance**.
- For many problems, can use a “standard” encoding.
 - Binary encoding of integers
 - For mathematical objects (graphs, matrices, etc.): list of parameters enclosed in braces, separated by commas

Analyze iterative algorithm for $F(n, 3)$

```
IFIB( $n, 3$ ) {  
     $F[0] \leftarrow 0;$   
     $F[1] \leftarrow 1;$   
    For( $i=2$  to  $n$ ) do  
         $F[i] \leftarrow F[i-1] + F[i-2];$   
    return  $F[n] \pmod{3};$   
}
```

2 instructions

$n - 1$ iterations

1 instruction

1 instruction

$$\text{No. of instructions} = 2 + (n - 1) \times 1 + 1 = n + 2$$

Analyze iterative algorithm for $F(n, 3)$

Not a
polytime
algo

```
{  
  F[0] ← 0;  
  F[1] ← 1;
```

```
  For(i=2 to n) do
```

```
    F[i] ← F[i-1] + F[i-2];
```

```
  return F[n] (mod 3);  
}
```

There is a polytime
algo for Fibonacci
number with
runtime $O(\log n)$

2 instructions

$n - 1$ iterations

1 instruction

1 instruction

$$\text{No. of instructions} = 2 + (n - 1) \times 1 + 1 = n + 2$$

Pseudo-polynomial algorithms

An algorithm that runs in time polynomial in the numeric value of the input but is exponential in the length of the input is called a **pseudo-polynomial** time algorithm.

Question 2

Are the algorithms we saw for KNAPSACK and FRACTIONAL KNAPSACK polynomial time?

- Yes for both
- Yes for KNAPSACK, no for FRACTIONAL KNAPSACK
- No for KNAPSACK, yes for FRACTIONAL KNAPSACK
- No for both

Question 2: Solution

No for KNAPSACK, yes for FRACTIONAL KNAPSACK

The input for both problems is a list $(v_1, w_1), \dots, (v_n, w_n), W$. The input size is $O(n \log M + \log W)$ where M is an upperbound on the v_i 's and w_i 's.

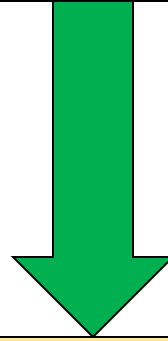
Running time for KNAPSACK is $O(nW \log M)$. Running time for FRACTIONAL KNAPSACK is $O(n \log W \log M)$.

Recap

- Reductions are a basic tool in algorithm design: using an algorithm for one problem to solve another.
- If you have a poly time reduction from A to B and you also have a poly time algorithm for B , then you get a poly time algorithm for A .

Poly-time Reduction

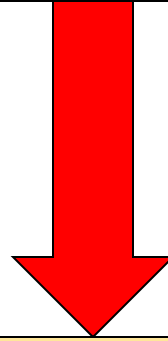
$$A \leq_p B$$



If B is “easily solvable”, then so is A !

Poly-time Reduction

$$A \leq_P B$$



If A is “hard”, then so is B !

Poly-time Reduction

$$A \leq_P B$$

*Intuition: A
is a special
case of B .*

If A is “hard”, then so is B !

Question 3

Suppose that $A \leq_P B$. Which of the following can we infer?

- a) If A can be solved in poly time, so can B .
- b) A can be solved in poly time iff B can be solved in poly time.
- c) If A cannot be solved in poly time, then neither can B .
- d) If B cannot be solved in poly time, then neither can A .

Question 3: Solution

Only (C)

Note that if A can be solved in poly time, then there is a reduction from A to **any** problem B whatsoever. The reason is that the reduction can just solve the instance for A by itself in polynomial time.

Intractability

Goal in this lecture and the next will be to **compare** the hardness of basic computational problems.

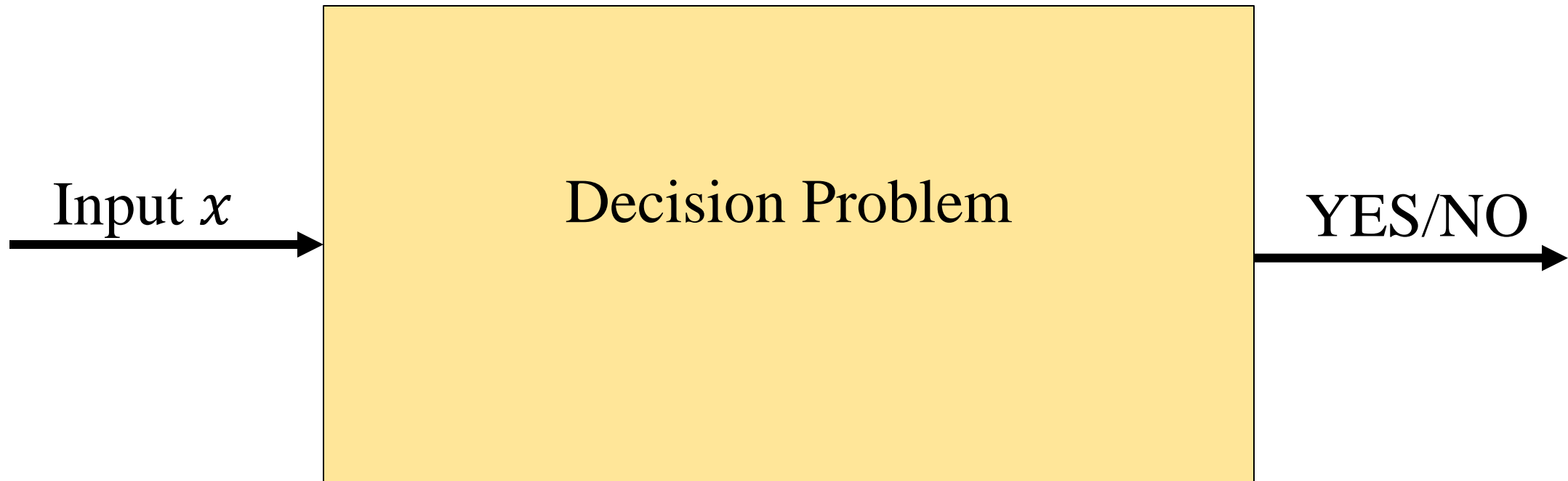
Intractability

Goal in this lecture and the next will be to **compare** the hardness of basic computational problems.

- Need a framework to talk about all problems using the same language!

Decision Problems

A **decision problem** is a function that maps an instance space I to the solution set $\{\text{YES}, \text{NO}\}$.



Decision vs Optimization

- **Decision Problem:** Given a directed graph G with two given vertices u and v , is there a path from u to v of length $\leq k$?
- **Optimization Problem:** Given a directed graph G with two given vertices u and v , what is the length of the shortest path from u to v ?

Decision vs Optimization

Depends on minimization
or maximization

Given an optimization problem, we can convert it into a decision problem:

Given an instance of the optimization problem and a number k , ask for a solution with value at \leq (**or** $>$) k ?

Examples: a path of length $\leq k$, LCS with length $> k$, a knapsack solution with value $> k$, etc.

Decision reduces to optimization

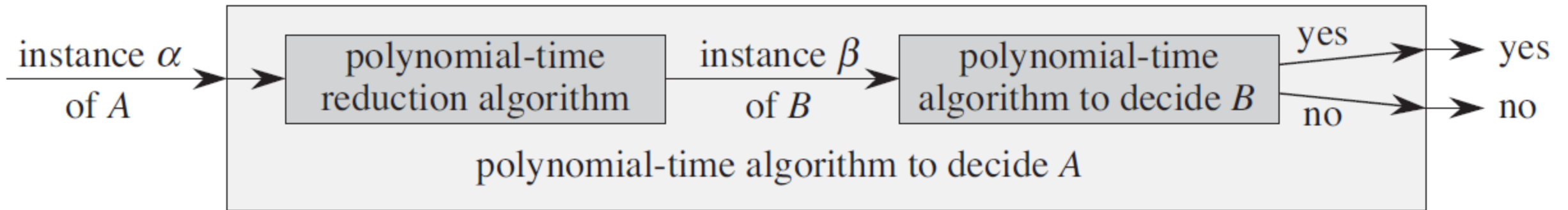
- The decision problem is no harder than the optimization problem
 - Given the value of the optimal solution, simply check whether it is $\leq k$.
- So, if we cannot solve the decision problem quickly, we cannot solve the optimization problem quickly! For hardness, enough to study decision problems.

Reductions between Decision Problems

Given two decision problems A and B , a **polynomial time reduction** from A to B , denoted $A \leq_P B$, is a transformation from instances α of A to instances β of B such that:

1. α is a YES-instance for A if and only if β is a YES-instance for B .
2. The transformation takes polynomial time in the size of α .

Reductions



Suffices to show:

- Reduction runs in polynomial time
- If α is a YES-instance of A , β is a YES-instance of B
- If β is a YES-instance of B , α is a YES-instance of A

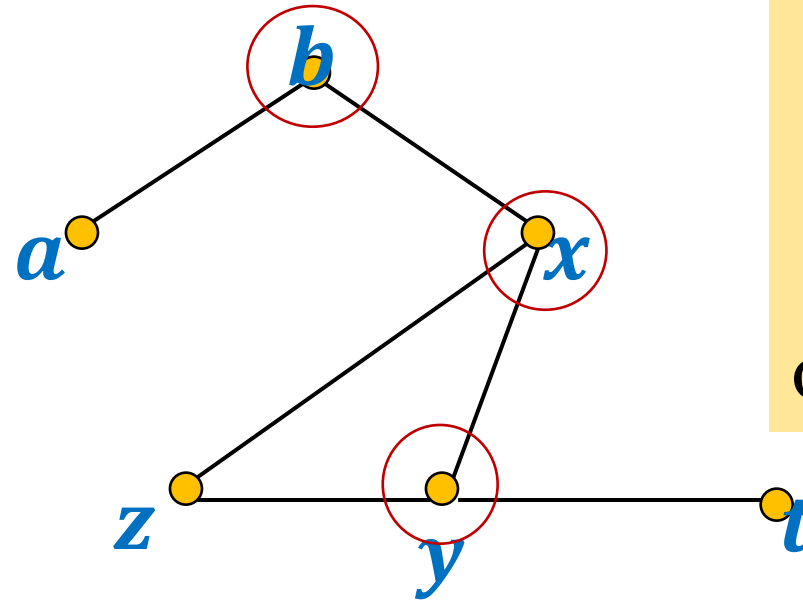
Example 1

Vertex Cover and Independent Set

Vertex Cover

Definition: Given an undirected graph $G = (V, E)$, a subset $X \subseteq V$ is said to be a **vertex cover** if

For each edge $(u, v) \in E$,
either $u \in X$ or $v \in X$



Is it a vertex cover now ?

YES

Reason:

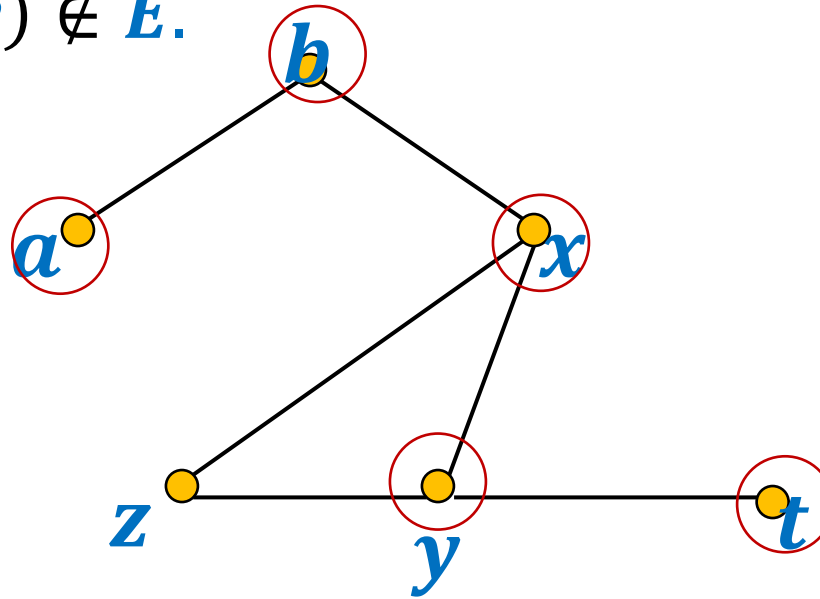
None of (y, z)
or (y, t) is covered

Optimization version: compute vertex cover of smallest size.

Decision version: Does there exist a vertex cover of size $\leq k$?

Independent Set

Definition: Given an undirected graph $G = (V, E)$, a subset $X \subseteq V$ is said to be an **independent set** if for each $u, v \in X$, $(u, v) \notin E$.



Is it the largest independent set ?

YES

NO

Optimization version: compute Independent set of Largest size.

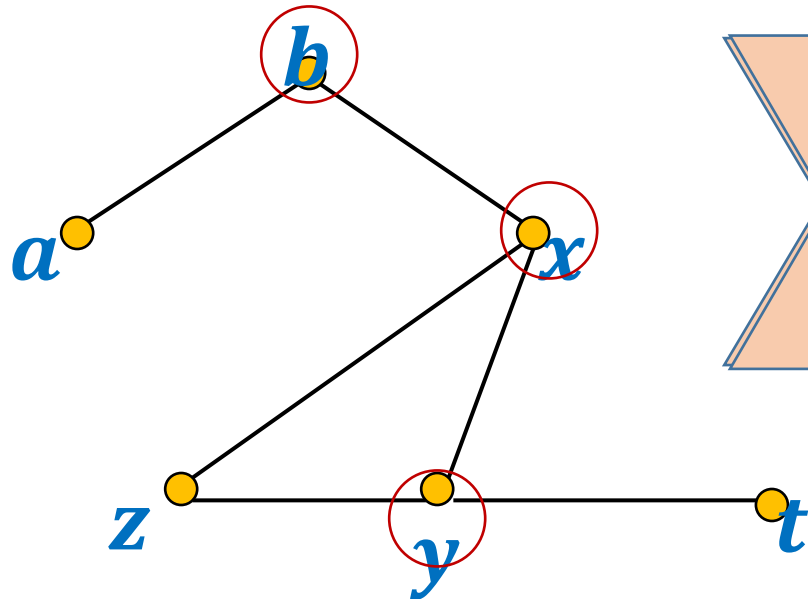
Decision version: Does there exist an independent set of size $> k$?

$$VC \leq_P IS$$

VC: Vertex Cover

Input: A graph $G = (V, E)$ and $k \in \mathbb{Z}^+$

Problem: Does there exist a vertex cover of size $\leq k$?

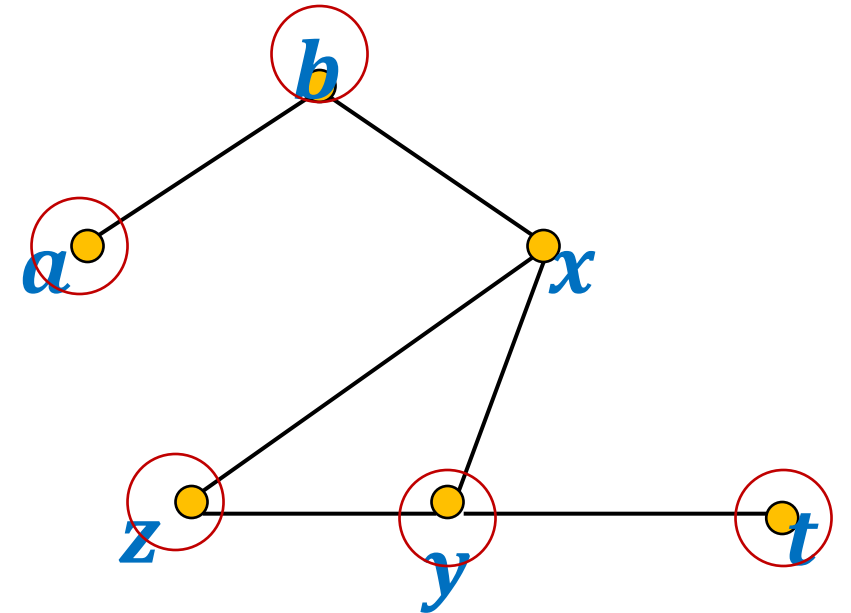


IDEA: Can you see the relation now ?

IS: Independent Set

Input: A graph $G = (V, E)$ and $t \in \mathbb{Z}^+$

Problem: Does there exist an independent set of size $> t$?



$VC \leq_P IS$

Theorem: $X \subseteq V$ is a vertex cover of G if and only if $V \setminus X$ is an independent set of G .

Proof:

- \rightarrow
- \leftarrow

$VC \leq_P IS$

Theorem (\Rightarrow): If $X \subseteq V$ is a vertex cover of G then $V \setminus X$ is an independent set of G .

Proof: Let $Y = V \setminus X$

Consider any two vertices $u, v \in Y$.

Is it possible that $(u, v) \in E$? **NO**

Reason: $u \notin X$ and $v \notin X$

so if $(u, v) \in E$, then X is not a vertex cover.

Hence for each $u, v \in Y$, $(u, v) \notin E$.

$\Rightarrow Y$ is an independent set of G

$$VC \leq_P IS$$

Theorem (\Leftarrow): If X is an independent set of G , then $V \setminus X$ is a vertex cover of G .

Proof: Let $Y = V \setminus X$.

Consider any edge $(u, v) \in E$

Since X is an independent set, so at most one of $\{u, v\}$ can be in X .

So **at least** one of $\{u, v\}$ must be in Y .

So edge (u, v) is covered by Y .

$\rightarrow Y$ is a vertex cover of G

$$\text{VC} \leq_P \text{IS}$$

Theorem: $X \subseteq V$ is a vertex cover of G if and only if $V \setminus X$ is an independent set of G .

Question: With this theorem what is the corresponding f to establish $\text{VC} \leq_P \text{IS}$?

Answer: On input (G, k) , f outputs $(G, n - k)$.

Clearly f takes polynomial time.

Hence $\text{VC} \leq_P \text{IS}$.

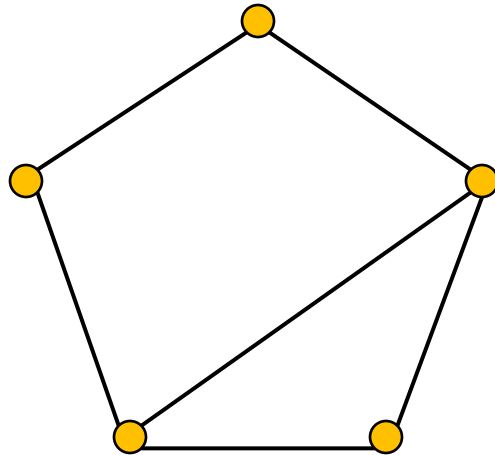
Exercise: Prove that $\text{IS} \leq_P \text{VC}$

Example 2

Hamiltonian cycle Problem **and** **Traveling Sales person Problem**

Hamiltonian Cycle Problem

Definition: Given an undirected graph $G = (V, E)$, a cycle is said to be Hamiltonian if it passes through each vertex.



Does it have a Hamiltonian cycle now?

YES

NO

Optimization version: compute cycle of maximum length.

Decision version: Does there exist a cycle of size n ?

Traveling Sales Person Problem

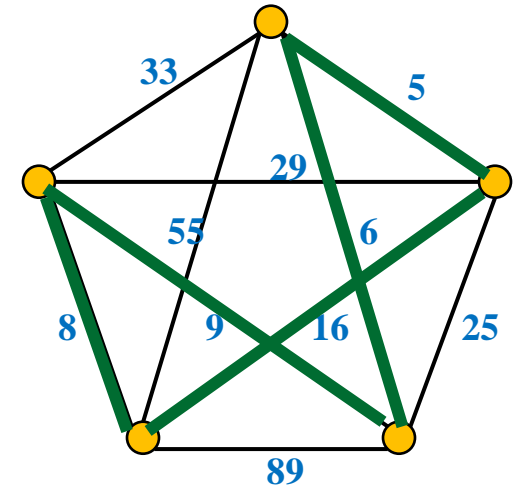
Definition: Given an undirected complete graph $G = (V, E)$ with nonnegative cost on edges, a tour is a sequence of vertices such that

- It originates and terminates at the same vertex
- There is an edge between every consecutive pair of vertices in the sequence
- Each vertex is visited exactly once.

Cost of tour: sum of cost of edges traversed in the tour.

Optimization version: compute TSP tour of minimum cost.

Decision version: Does there exist a TSP tour of cost at most b ?

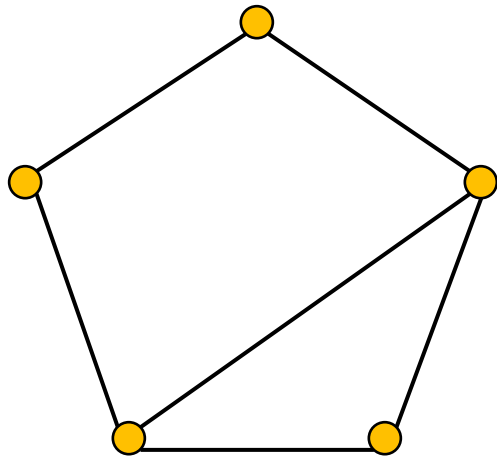


HC \leq_P TSP

HC: Hamiltonian Cycle

Input: An undirected graph

Problem: Does there exist a cycle passing through all vertices ?

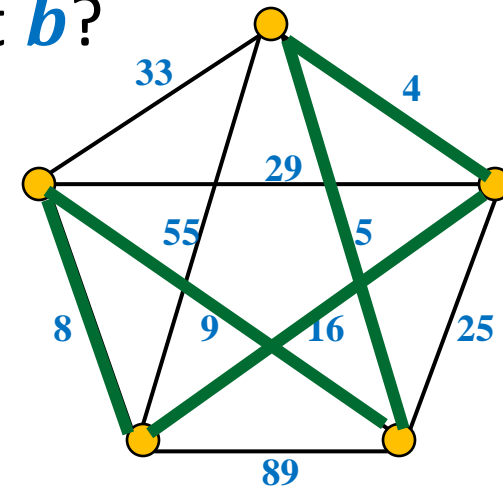


IDEA: Build a weighted complete graph G' s.t. the Hamiltonian cycle in G turns out to be the least cost tour in G' .

TSP: Traveling Sales Person

Input: An undirected complete graph with non-negative **cost** on edges and $b \in \mathbb{R}^+$.

Problem: Does there exist a tour of cost at most b ?



HC \leq_P TSP

Let $G = (V, E)$ be an instance of **HC**

Build a complete graph G' on V vertices.

For each pair (u, v) : If $(u, v) \in E$, $c(u, v) \leftarrow 1$
else $c(u, v) \leftarrow 2$

(Any cost greater than 1 will work here)

Theorem: G has a **Hamiltonian cycle** if and only if G' has a **TSP** tour of cost at most n .

Proof:

- \rightarrow
- \leftarrow

HC \leq_P TSP

Theorem (\Rightarrow): If G has a **Hamiltonian cycle** then G' has a **TSP** tour of cost at most n .

Proof: Let C be a **Hamiltonian cycle** in G .

G is a subgraph of G' .

So C must be present in G' as well.

C is a tour since each vertex appears exactly once in C .

Cost of each edge of C is 1 since each edge of C is present in G as well.

So the cost of tour $C = n$

Hence G' has a tour of cost at most n .

HC \leq_P TSP

Theorem (\Leftarrow): If G' has a **TSP** tour of cost at most n , then G has a **Hamiltonian cycle**.

Proof: Let C be a **TSP** tour of cost at most n in G' .

Cost of each edge in G' is at least 1 .

There are n edges in C .

So each edge of C must have weight exactly 1 .

Therefore, each edge of C is present in G as well.

So C is present in G as well.

Since each vertex appears exactly once on C , so C is **Hamiltonian** as well.

Hence G has a Hamiltonian cycle.

HC \leq_P TSP

Theorem: G has a **Hamiltonian cycle** if and only if G' has a **TSP** tour of cost at most n .

Question: With this theorem what is the corresponding f to establish $VC \leq_P IS$?

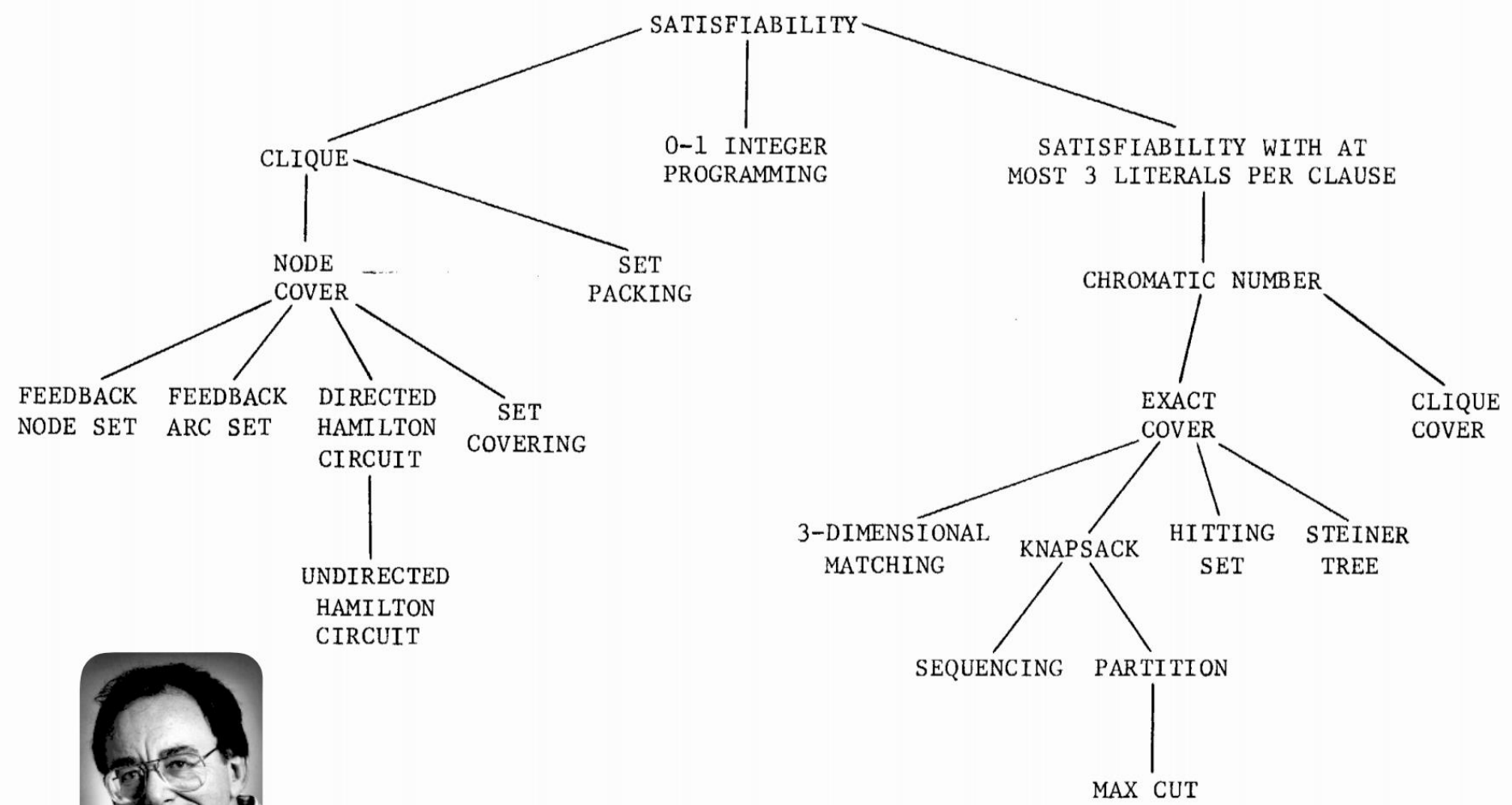
Answer: On input (G) , f constructs G' as described above and outputs (G', n) .

Clearly f takes polynomial time. Hence **HC \leq_P TSP**

Exercise: A path in undirected graph is said to be **Hamiltonian** path if it passes through each vertex exactly once.

HP - Hamiltonian path problem : “Does a given undirected graph G have a Hamiltonian path ?”

Prove that **HC \leq_P HP**



Dick Karp (1972)
1985 Turing Award

FIGURE 1 - Complete Problems

Looking Forward

- In the next lecture, we will see that there are reductions between any pair of these problems.
- Actually, there are hundreds of problems (**NP-complete**) that have reductions to and from the above problems. Said another way, if we have a poly time algorithm for any one of these NP-complete problems, we have poly time algorithms for all of them!

Acknowledgement

- The slides are modified from
 - The slides from Prof. Kevin Wayne
 - The slides from Prof. Surender Baswana
 - The slides from Prof. Erik D. Demaine and Prof. Charles E. Leiserson
 - The slides from Prof. Arnab Bhattacharya and Prof. Wing-Kin Sung