Design of Algorithms by Induction-1

Design of Algorithms

- Using some well-know strategies
  - greedy method
  - dynamic programming
  - divide and conquer
  - …
- Using induction

  base: solve a small instance of problem

  induction: a solution to every problem
  can be constructed from
  solutions of smaller problems.

Greedy Algorithms

- Greedy algorithm work in phases
- In each phase, a ***local optimum*** decision is made
- At last, ***global optimum*** is achieved.
- If local optimum -> global optimum, optimal solution
  else suboptimal solution
  - If absolute optimal answer not required,
    simple greedy algorithms are sometimes used as
    approximate solution.

Job Scheduling

- Given N jobs $j_1, j_2, \ldots, j_N$, each with $t_1, t_2, \ldots, t_N$ running time
  how to schedule jobs
  in order to minimize average completion time

- Given 4 jobs j_1, j_2, j_3, j_4, each with running time 15, 8, 3, 10
  how to schedule jobs
  in order to minimize average completion time ？

Schedule：8, 10, 3, 15

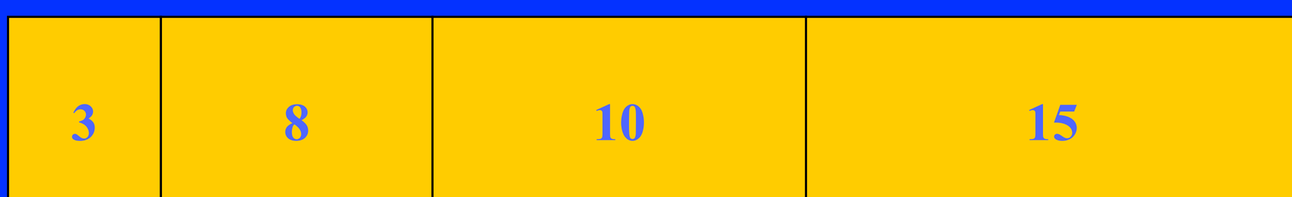
Average completion time = (8+18+21+36)/4=83/4

Job Scheduling

- Given N jobs $j_1, j_2, \ldots, j_N$, each with $t_1, t_2, \ldots, t_N$ running time
  how to schedule jobs
  in order to minimize average completion time

First Come First Serve

[15 + (15+8) + (15+8+3) + (15+8+3+10)]/4 = 25



Shortest Job First

[3 + (3+8) + (3+8+10) + (3+8+10+15)]/4 = 17.75

Divide-and Conquer

- Recursively
  - Divide into two parts
  - Conquer each parts

Merge Sort

- Merge sort
  - merge two sorted lists
  - recursive algorithm
  - Divide-and-conquer strategy

Example of Merge Sort

	A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]
Original	11	8	14	7	6	8	23	4
after pass 1	8	11	7	14	6	8	4	23
afetr pass 2	7	8	11	14	4	6	8	23
after pass 3	4	6	7	8	8	11	14	23

Dynamic Programming

- Using a table, instead of recursion
- Solution depends on many solutions of smaller problems
- A table is used to store previous results
- Less efficient than divide-and-conquer

A Simple Example of Dynamic Programming

- Fibonacci Number F(n) = ?
- F(n) = F(n-1) + F(n-2)   1, 1, 2, 3, 5, 8, 13
- Two approaches
  - Recursive program

```
int fib(int n)
{
    if ( n <= 1)
        return (n);
    else
        return( fib(n-1) + fib(n-2) );
}
```

  - Dynamic Programming

1	1	2	3	5	8	13	21	34	…
1	2	3	4	5	6	7	8	9	…

Design of Algorithms

- Using some well-know strategies
  - greedy method
  - dynamic programming
  - divide and conquer
  - …
- Using induction

  base: solve a small instance of problem

  induction: a solution to every problem

  can be constructed from

  solutions of smaller problems

Mathematical Induction

- Mathematical induction for proof of theorem T(n)

  1. T holds for n=1
  2. $\forall$ n >1, if T holds for n-1 (induction hypothesis)

     then T holds for n

Variations of Mathematical Induction

- 1. T holds for n = 1

  2. $\forall$ n >1, if T holds for all natural numbers < n

  then T holds for n
- 1. T holds for n=1 & n=2

  2. $\forall$ n >2, if T holds for n-2

  then T holds for n
- 1. T holds for $n=2^0$

  2. $\forall$ n >1, n is an integer power of 2,

  if T holds for $n/2$ $(2^{k-1} \Rightarrow 2^k)$

  then T holds for n

Mathematical Induction: Example 1

- ∀ natural number x & n,
  x^n-1 is divisible by (x-1)

<proof>

1. n=1, x-1 is divisible by x-1
2. Assume x^{n-1}-1 is divisible by x-1

$$x^n-1 = x(x^{n-1}-1) + (x-1)$$

Mathematical Induction: Example 2

- The number of regions in the plane formed by n lines in general position is $\frac{n(n+1)}{2}+1$

* A set of lines in the plane is in general position
 if no two lines are parallel and
 no three lines intersect at a common point

<proof>

1. n=1 $\rightarrow$ 2 regions, n=2 $\rightarrow$ 4 regions, n=3 $\rightarrow$ 7regions

2. Hypothesis: adding i-th line increases i regions

3. Proof of hypothesis

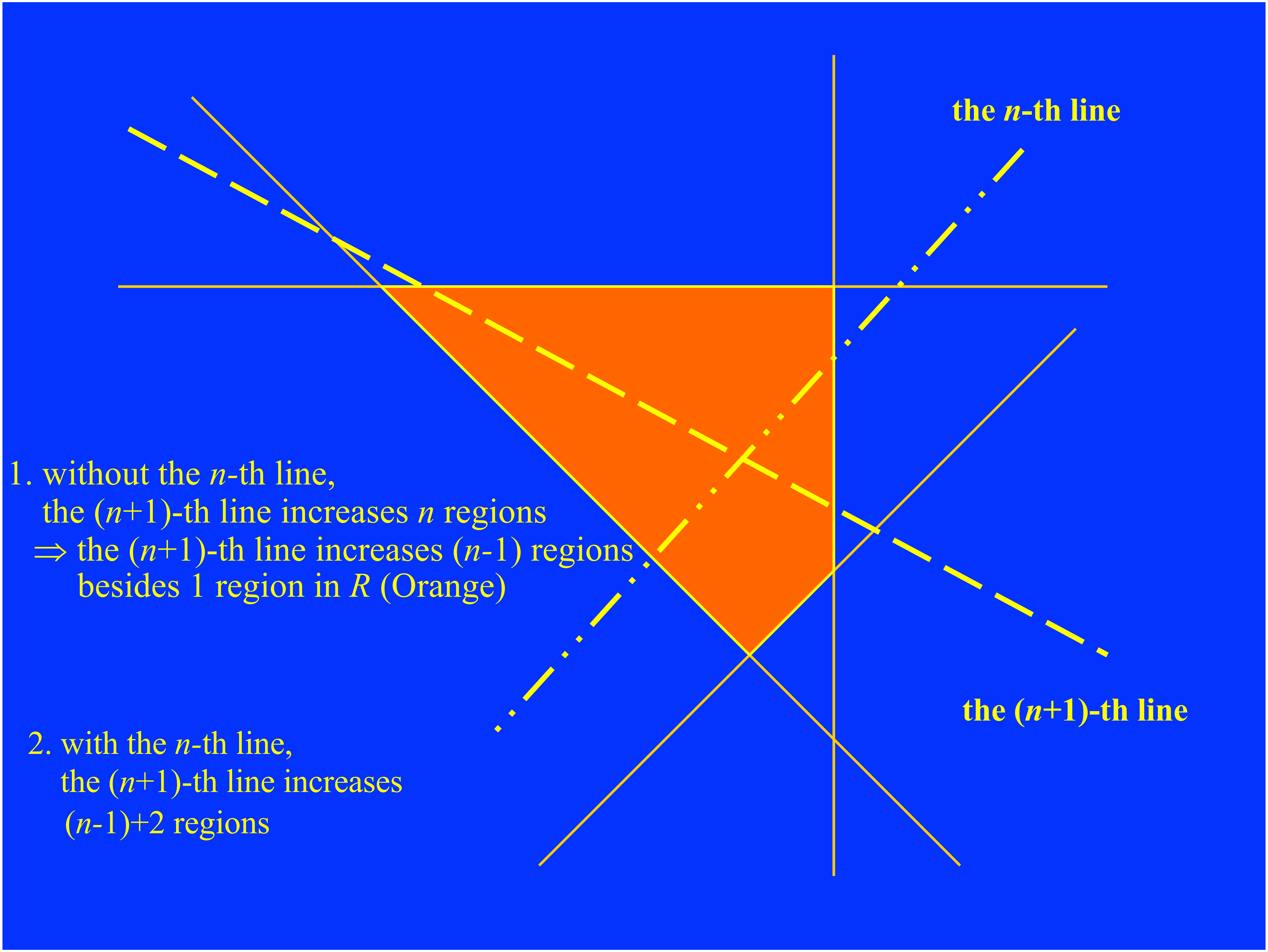
- approach 1: (n+1)-th line intersects n+1 existing regions
- approach 2:

  1. Without n-th line, (n+1)-th line increases n regions

     $\Rightarrow$ (n+1)-th line increases (n-1) regions besides 1 region in R

  2. R is the only affected region & R is cut from 2 to 4

     $\therefore$ With nth line, (n+1)-th line increases (n-1)+2 regions


the *n*-th line
1. without the *n*-th line,
the (*n*+1)-th line increases *n* regions
⇒ the (*n*+1)-th line increases (*n*-1) regions
besides 1 region in *R* (Orange)
the (*n*+1)-th line
2. with the *n*-th line,
the (*n*+1)-th line increases
(*n*-1)+2 regions

<proof>

1. n=1 → 2 regions, n=2 → 4 regions, n=3 → 7regions
2. Hypothesis: adding *i*-th line increases *i* regions
3. Proof of hypothesis
   - approach 1: (n+1)th line intersects n+1 existing regions
   - approach 2:
     1. Without nth line, (n+1)th line increases n regions

        ⇒ (n+1)-th line increases (n-1) regions

        besides one region in R
     2. R is the only affected region & R is cut from 2 to 4

        ∴ With n-th line, (n+1)-th line increases (n-1)+2 regions
4. n lines increases 2+2+3+4+…+n = n(n+1)/2+1 regions

- Comments
  - The hypothesis deal with the growth of the function
  - The hypothesis was used twice (nth line, (n+1)th)

Mathematical Induction: Example 3

$$
\begin{array}{ccccccccccc}
 & & & & 1 & & & & & & = 1 \\
 & & & 3 & + & 5 & & & & & = 8 \\
 & & 7 & + & 9 & + & 11 & & & & = 27 \\
 & 13 & + & 15 & + & 17 & + & 19 & & & = 64 \\
21 & + & 23 & + & 25 & + & 27 & + & 29 & & = 125
\end{array}
$$

Hypothesis: the sum of row *i* in the triangle is i^3

Mathematical Induction: Example 3 (cont.)

1 = 1
3 + 5 = 8
7 + 9 + 11 = 27
13 + 15 + 17 + 19 = 64
21 + 23 + 25 + 27 + 29 = 125

Hypothesis: the sum of row i in the triangle is i^3

<proof> Difference between row $i+1$ & i is $(i+1)^3-i^3$

1. Difference between corresponding elements: $2i$

13 + 15 + 17 + 19 (row 4)
8 8 8 8
21 + 23 + 25 + 27 + 29 (row 5)

2. totally, there are i pairs, each with difference $2i$

3. last element? $(i+1)^3-i^3-2i*i=i^3+3i^2+3i+1-i^3-2i^2= i^2+3i+1$

Mathematical Induction: Example 3 (cont.)

$$
\begin{array}{ccccccccccc}
 & & & & 1 & & & & & & = 1 \\
 & & & 3 & + & 5 & & & & & = 8 \\
 & & 7 & + & 9 & + & 11 & & & & = 27 \\
 & 13 & + & 15 & + & 17 & + & 19 & & & = 64 \\
21 & + & 23 & + & 25 & + & 27 & + & 29 & & = 125
\end{array}
$$

Nested Hypothesis: the last number in row i+1 is i^2+3i+1

<nested proof>

1. i=1, the 2nd row , last element=5
2. i=n-1, the n-th row , last element= $(n-1)^2+3(n-1)+1$

=> i=n, the (n+1)-th row, last element

$$= (n-1)^2+3(n-1)+1+2n+2=n^2+3n+1$$

- Comments
  - Should not always try to achieve whole proof in one step
  - Going backward: start with the final problem & reducing to a simpler problem

Reversed Induction Principle

- If a statement *P* is true

  for an infinite subset of the natural numbers

and

  if its truth for *n* implies its truth for *n*-1,

  then *P* is true for all natural numbers

Mathematical Induction: Example 4

- If $x_1, x_2, \ldots x_n$ are all positive numbers, then

$$(x_1 x_2 \ldots x_n)^{1/n} <= (x_1 + x_2 + \ldots + x_n)/n$$

- Proof

case 1: n is a power of 2

n=1: trivial

n=2: $(x_1 x_2)^{1/2} <= (x_1 + x_2)/2$

assume $n=2^k$ is true

consider $2n=2^{k+1}$

$$(x_1 x_2 \ldots x_{2n})^{1/2n} = [(x_1 x_2 \ldots x_n)^{1/n} (x_{n+1} x_{n+2} \ldots x_{2n})^{1/n}]^{1/2}$$

$$= (y_1 y_2)^{1/2} <= (y_1 + y_2)/2$$

$$<= [(x_1 + x_2 + \ldots + x_n)/n + (x_{n+1} + x_{n+2} + \ldots + x_{2n})/n]/2$$

Mathematical Induction: Example 4 (cont.)

- If $x_1, x_2, \ldots x_n$ are all positive numbers, then

  $(x_1 x_2 \ldots x_n)^{1/n} <= (x_1+x_2+\ldots+x_n)/n$

- Proof (using reversed induction principle)

case 2: n is not a power of 2

assume n is true and consider n-1

define $z = (x_1+x_2+\ldots+x_{n-1})/(n-1)$

$(x_1 x_2 \ldots x_{n-1} z)^{1/n} <= (x_1+x_2+\ldots+x_{n-1}+z)/n = [(n-1)z+z]/n = z$

$\Rightarrow (x_1 x_2 \ldots x_{n-1} z)^{1/n} <= z$

$\Rightarrow (x_1 x_2 \ldots x_{n-1} z) <= z^n$

$\Rightarrow (x_1 x_2 \ldots x_{n-1}) <= z^{(n-1)}$

$\Rightarrow (x_1 x_2 \ldots x_{n-1})^{1/(n-1)} <= z = (x_1+x_2+\ldots+x_{n-1})/(n-1)$

Evaluating Polynomials

Evaluating Polynomials

- Given a sequence of real no. $a_n, a_{n-1}, \ldots a_1, a_0$

  and a real number x

  Compute value of polynomial

  $P_n(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_0$

- Given 10, 5, 8, 2, 6, (i.e., $P_4(x) = 10x^4+5x^3+8x^2+2x+6$)

  and 2, (i.e., x=2)

  Compute $P_4(2) = 10*2^4+5*2^3+8*2^2+2*2+6$

Evaluating Polynomials

- Approach 1
  - Hypothesis: we know how to compute $P_{n-1}(x)$
  - Base: a_0
  - Induction: $P_n(x) = a_n x^n + P_{n-1}(x)$
  - Complexity

Evaluating Polynomials

- Approach 1
  - $P_4(x) = 10x^4+5x^3+8x^2+2x+6=$
    $10*x*x*x*x+5*x*x*x+8*x*x+2*x+6$
  - Hypothesis: we know how to compute $P_{n-1}(x)$
  - Base: a_0
  - Induction: $P_n(x) = a_n x^n + P_{n-1}(x)$
  - Complexity: $(n+n-1+n-2+\ldots+1)$ multiplications & n additions

有可能次數更少嗎？

Evaluating Polynomials (cont.)

- Approach 2
  - $10x^4+5x^3+8x^2+2x+6=$

    $6+ 2*x+ 8*x*x+5*x*x^2+10*x*x^3$
  - Hypothesis: we know how to compute $P_{n-1}(x)$ & x^{n-1}
  - Induction: $P_n(x) = P_{n-1}(x) + a^n \cdot x \cdot x^{n-1}$
  - Complexity ?

Evaluating Polynomials (cont.)

- Approach 2
  - Hypothesis: we know how to compute $P_{n-1}(x)$ & x^{n-1}
  - Induction: $P_n(x) = P_{n-1}(x) + a^n \cdot x \cdot x^{n-1}$

    e.g. $10x^4+5x^3+8x^2+2x+6=$

    $6+ 2*x+ 8*x*x+5*x*x^2+10*x*x^3$
  - Complexity: 2n multiplication & n addition

有可能只需要 n 次乘法及 n 次加法嗎？

Evaluating Polynomials (cont.)

- Approach 3
  - $10x^4 + 5x^3 + 8x^2 + 2x + 6 =$

    $\{[(10x + 5)x + 8]x + 2\}x + 6$
  - Hypothesis: we know how to compute $P'_{n-1}(x)$

    $P'_{n-1}(x) = a_n x^{n-1} + a_{n-1} x^{n-2} + \ldots + a_1$
  - Induction: $P_n(x) = x \bullet P'_{n-1}(x) + a_0$

    $a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_0 =$

    $((\ldots(a_n x + a_{n-1})x + a_{n-2})\ldots)x + a_1)x + a_0$
  - Complexity: *n* multiplication & *n* addition

Algorithm of Polynomial Evaluation

Algorithm Polynomial_Evaluation (A, x)

Input: $A=\{a_n, a_{n-1}, \ldots a_1, a_0\}$ and x

Output: P

Begin

    $P = a_n$;

    for i =1 to n do

        $P = x*P + a_{n-i}$

End

Comments

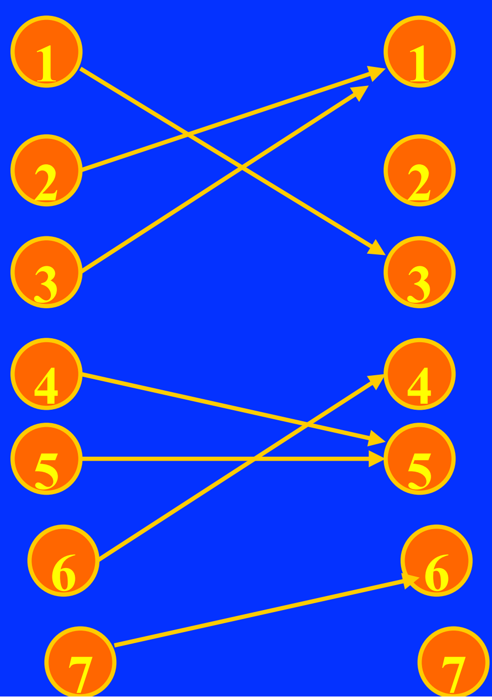
- Induction: extending solutions of smaller subproblems to those of larger problems
- Usual: $p(n-1) \rightarrow p(n)$
- Other possible induction
  - considering input from left to right
  - comparing top down vs. bottom up
  - go in increments of 2 rather than 1
  - ….

Finding One-To-One Mapping

Finding One to One Mapping

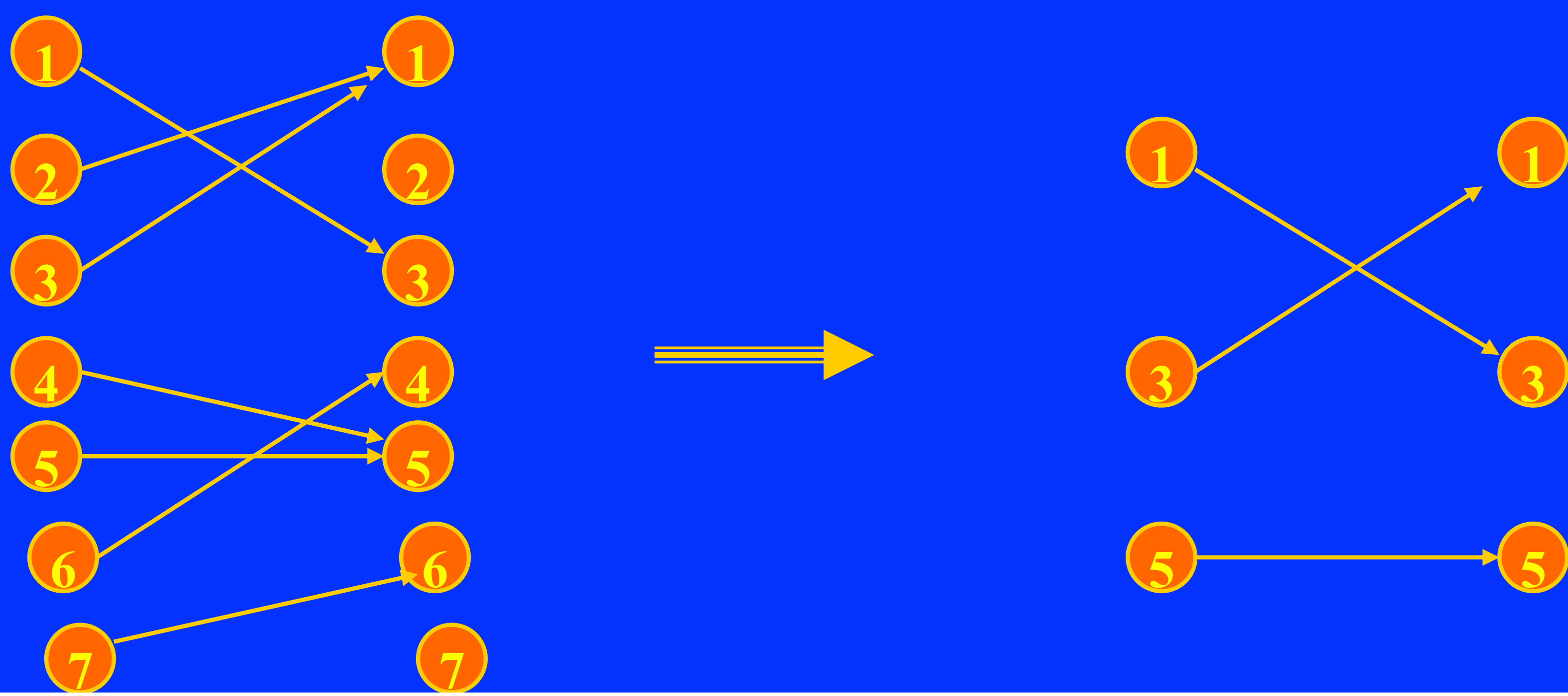
Mary 想邀請朋友舉行個 Party，並互相交換禮物。

每位朋友事先必須準備一份禮物，且每位朋友只會得到一份禮物。Mary 計畫事先瞭解每位朋友想將禮物送給誰 (也可以送給自己)。但有可能有的朋友因此會收到多個禮物，也有可能有的朋友因此沒收到任何禮物。為了避免這尷尬的情形，因此 Mary 只邀請部分朋友交換禮物。請問 Mary 應邀請哪些朋友？



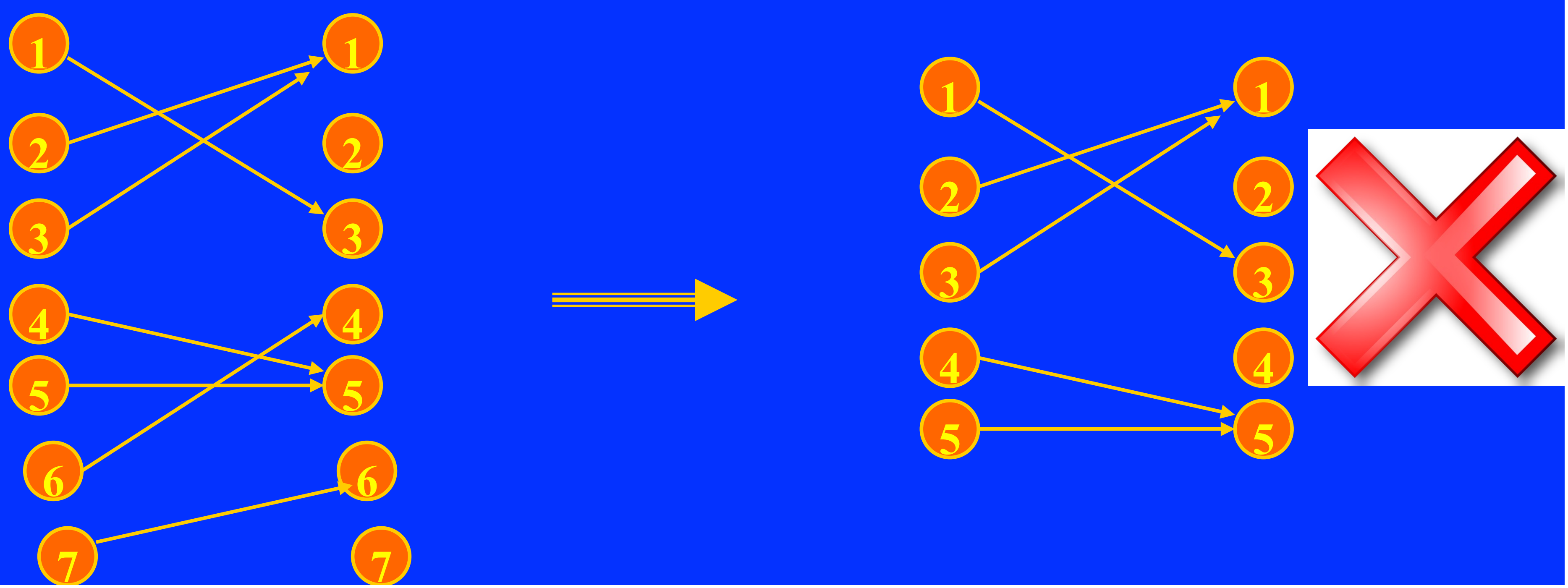
Finding One to One Mapping

- **Given** a finite set A & a function *f* from A to itself
  Find a subset S of A
  with maximum number of elements
  such that
  (1) *f* maps S to itself
  (2) *f* is one to one when restricted to S



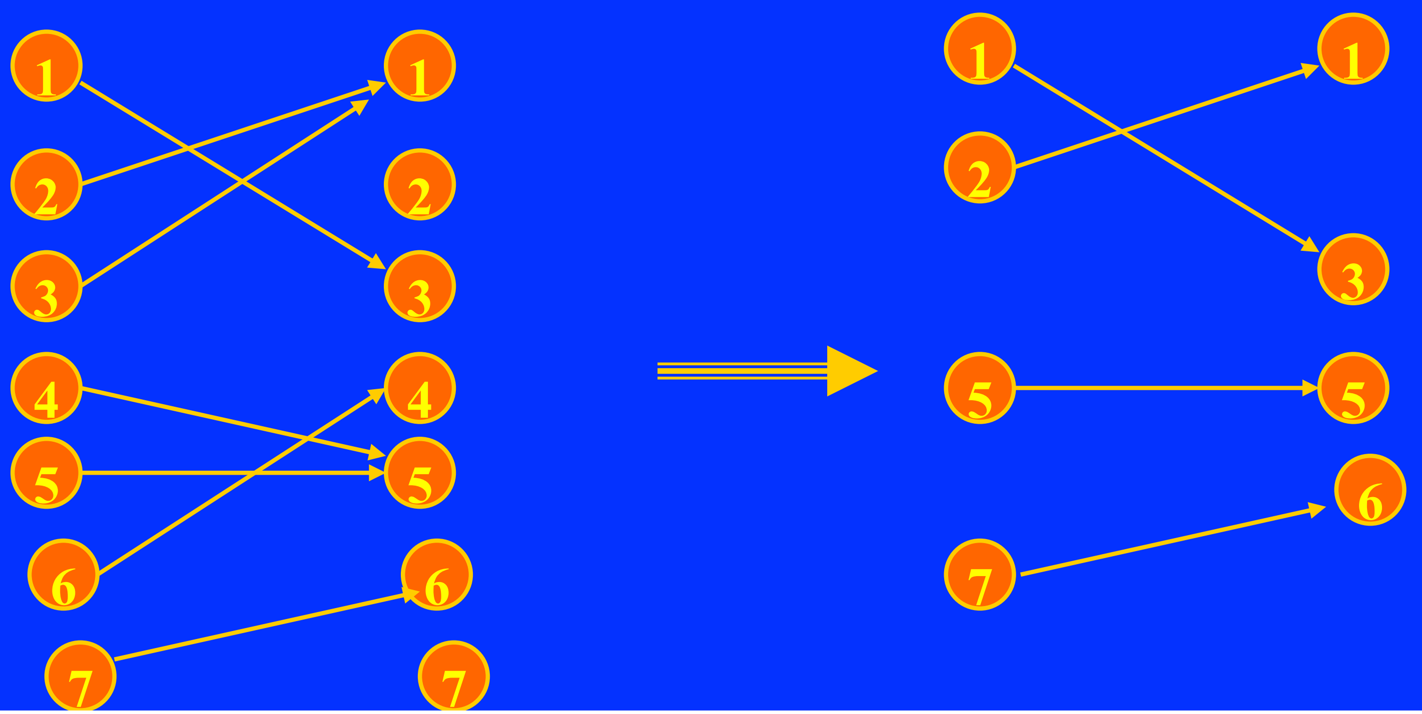
Finding One to One Mapping

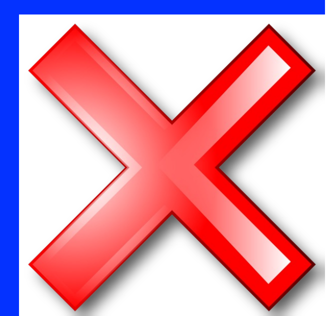
- **Given** a finite set A & a function *f* from A to itself
  Find a subset S of A
  with maximum number of elements
  such that
  (1) *f* maps S to itself
  (2) *f* is one to one when restricted to S



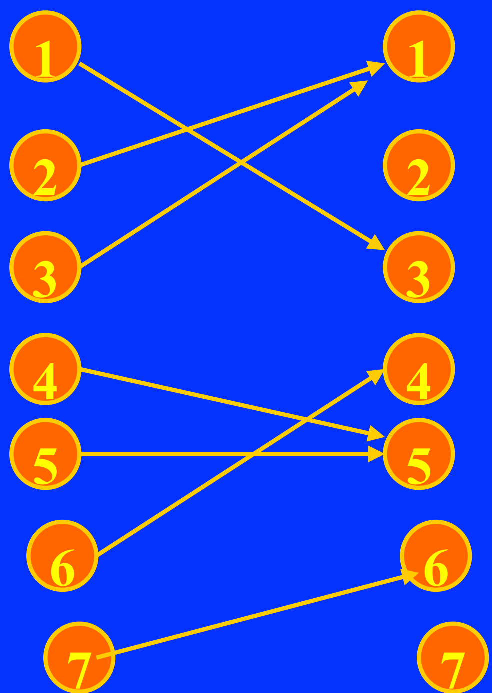
Finding One to One Mapping

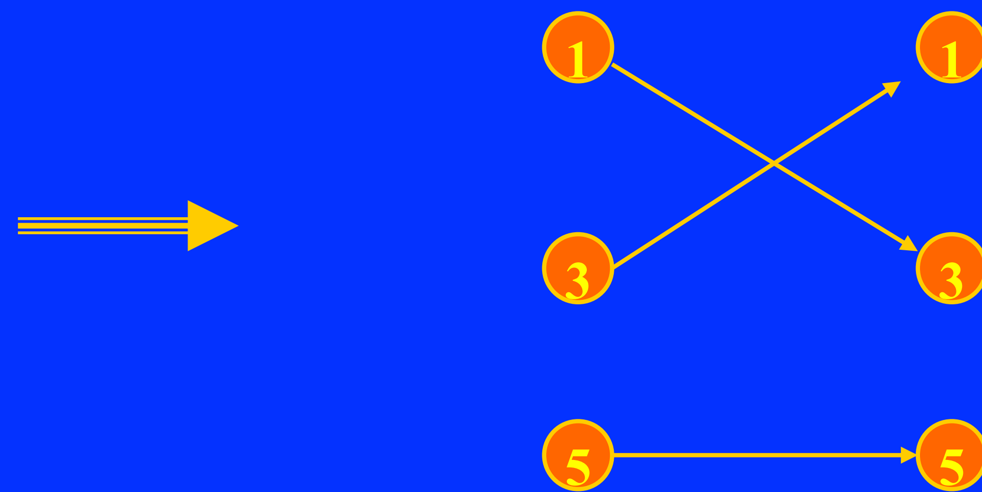
- **Given** a finite set A & a function *f* from A to itself
  Find a subset S of A
  with maximum number of elements
  such that
  (1) *f* maps S to itself
  (2) *f* is one to one when restricted to S





Thinking





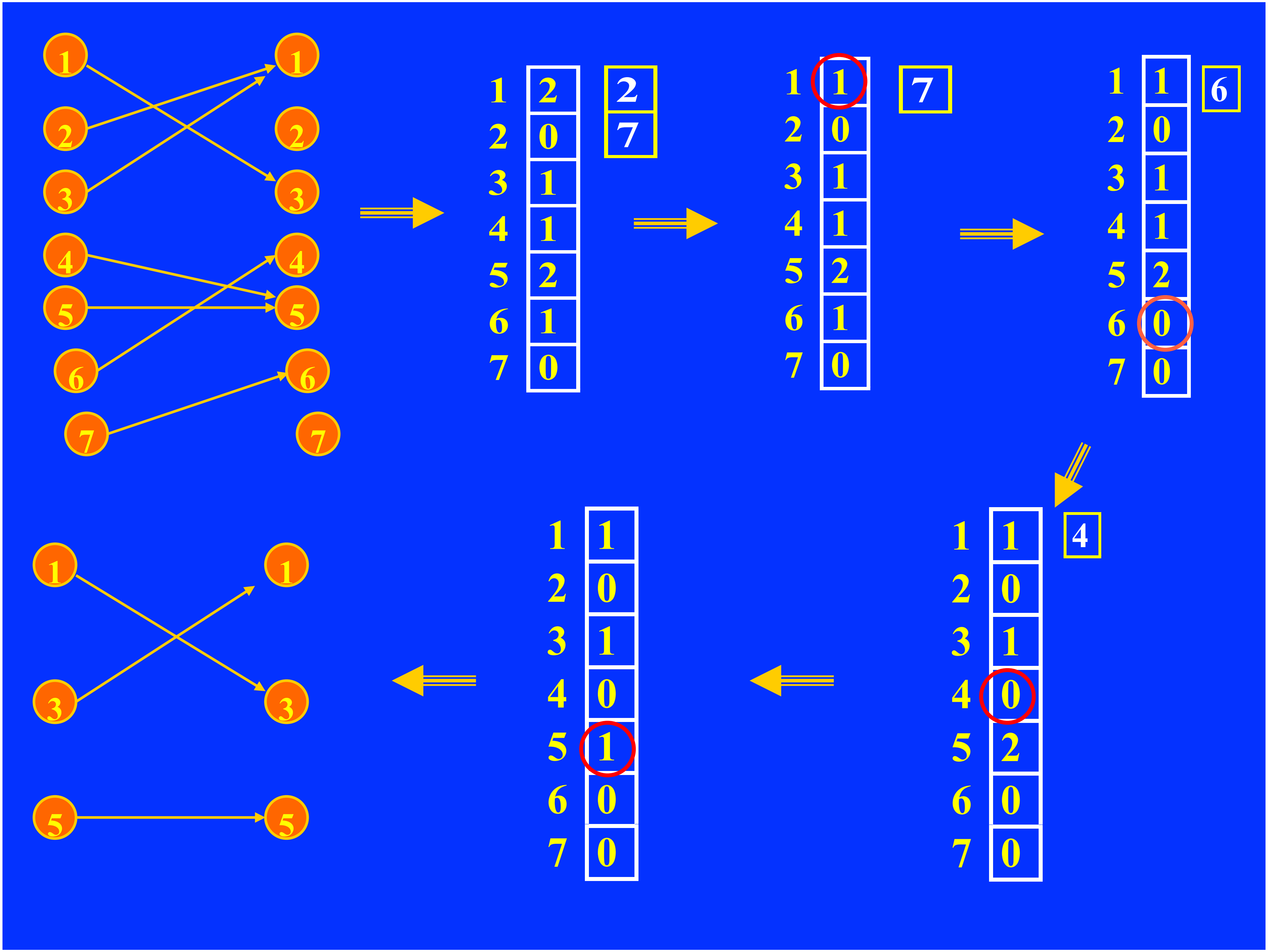
- Hypothesis: solve problem for set of (n-1) elements
- Induction on n elements
  - by finding an element belong to S
  - by finding an element that does not belong to S
    - Elimination
    - Any element i that has no other element mapped to it cannot belong to S

如何找出朋友使得參與交換禮物的人數最多，且避免尷尬的情形？

Induction of One to One Mapping

- Hypothesis: solve problem for set of (n-1) elements
- Base:
- Induction:
  - any element *i* that has no other element mapped to it cannot belong to S
  - remove *i*, A'=A-{*i*}, A' has (n-1) elements

  * condition in A (n element) is the same as that in A'=A-{*i*} (n-1 element), except size
- Complexity: O(n)



1 2 3 4 5 6 7
1 2 3 4 5 6 7
1 2 3 4 5 6 7
2 0 1 1 2 1 0
2
7
1 2 3 4 5 6 7
1 0 1 1 2 1 0
7
1 2 3 4 5 6 7
1 0 1 1 2 0 0
6
1 2 3 4 5 6 7
1 0 1 0 2 0 0
4
1 2 3 4 5 6 7
1 0 1 0 1 0 0
1 3 5
1 3 5

Algorithm of Mapping

```
Algorithm mapping (f, n)
Input: f (array of integer)
Output: S
Begin
    S:=A;
    for j:=1 to n do c[j]:=0;
    for j:=1 to n do increment c[f[j]];
    for j:=1 to n do
        if c[j]=0 then put  j in queue;
    while Queue is not empty do
        remove i from the top of queue;
        S:=S-{i};
        decrement c[f[i]];
        if c[f[i]] = 0 then put f[i] in queue;
End
```

The Celebrity Problem

The Celebrity Problem

- Celebrity: someone who is known by everyone
  but does not know anyone
- Celebrity problem
  Identify the celebrity by asking questions
  "Do you know the person over there?"
  Goal: minimize the number of questions
- In graph theory: celebrity = sink
  sink: vertex with indegree (n-1) & outdegree 0

The Celebrity Problem (cont.)

- **Celebrity problem expressed in adjacency matrix**

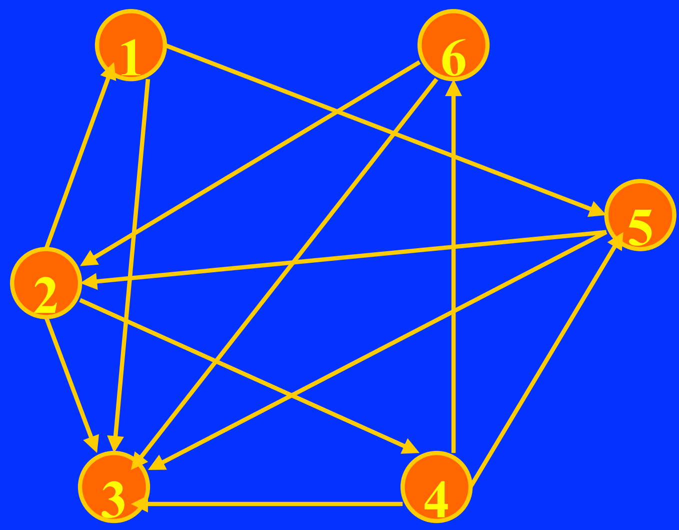
  Given an *n* x *n* adjacency matrix

  determine whether there exists an *i*

  such that

  (1) all the entries in the *i*-th column (except *ii*) are 1

  (2) all the entries in the *i*-th row (except *ii*) are 0



	1	2	3	4	5	6
1	0	0	1	0	1	0
2	1	0	1	1	0	0
3	0	0	0	0	0	0
4	0	0	1	0	1	1
5	0	1	1	0	0	0
6	0	1	1	0	0	0

如何以Brute-Force找出Celebrity?
時間複雜度是多少？

Thinking

- Brute force: $n(n-1)/2$ pairs * 2 = $O(n^2)$ question asking
- Induction from $(n-1)$ to n
  - assume we can find the celebrity among n-1 persons
  - 3 possibilities for $(n-1) \rightarrow n$

    Case 1: celebrity is among the first (n-1)

    check the n-th person

    Case 2: celebrity is the n-th person

    2(n-1) question asking is required

    Case 3: there is no celebrity
  - Complexity: $O(n \times 2(n-1))=O(n^2)$

以Induction的角度，求解O(n)的演算法來找出Celebrity?

Induction

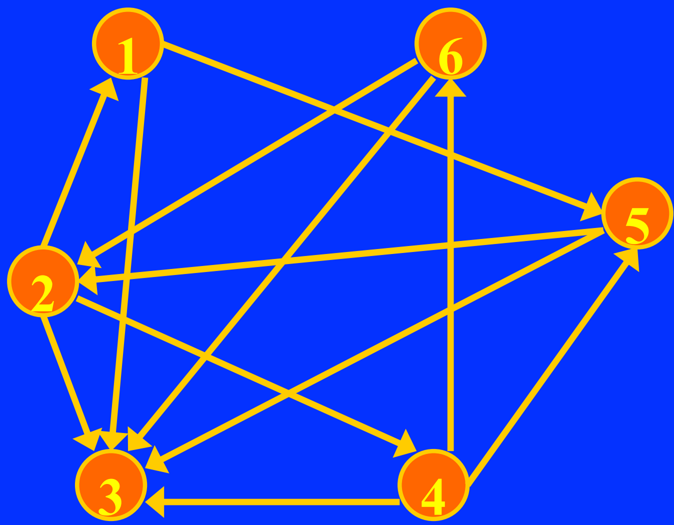
- Hypothesis: we know how to find celebrity among n-1 persons (if there exists, celebrity is among the (n-1) person)
- Induction:
  - eliminate someone who is non-celebrity $n \rightarrow (n\text{-}1)$
    - ask someone X whether he/she knows Y
    - if X knows Y, X is not celebrity, eliminate X
    - if X does not know Y, Y is not celebrity, eliminate Y
  - 3 possibilities
    - Case 3: no celebrity among (n-1) persons
      $\rightarrow$ no celebrity among n persons
    - Case 2: not exist (since celebrity is not the n-th person)
    - Case 1: two more questions to verify the celebrity among (n-1)

Algorithm

- Algorithm
  - Phase 1: eliminate all but one candidate
  - Phase 2: verify candidate
- Implementation
  - Using stack to store candidates
  - Pop 2 candidate & ask question, eliminate one

```
While stack is not empty
        pop one candidate & ask question
        eliminate one
Verify
```

- Complexity: 3(n-1) questions, O(n)



1	2	3	4	5	6	7
i	j	next				
i		j	next			
		j	i	next		
		j		i	next	
		j			i	next
						i

```
Algorithm celebrity (Know);
Input: Know (an n x n Boolean matrix)
Output: celebrity
Begin
    i:=1; j:=2; next:=3;
    while next <= n+1 do
        if know[i, j] then i := next
                      else   j := next;
        next:=next+1;
    if i=n+1 then  candidate := j;
             else    candidate := i;
    wrong:= false;      k:=1;     know[candidate,
  candidate]:=false;
    while not wrong and k <= n do
        if know[candidate, k] then wrong := true;
        if not know[k, candidate] then
                if candidate <> k then wrong := true;
        k:=k+1;
    if not wrong then celebrity := candidate
                   else celebrity :=0
End
```