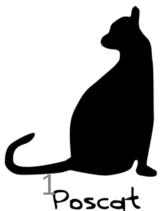
# POSCAT Seminar 2 : Algorithm Verification & Efficiency

yougatup @ POSCAT



# Topic

- Topic today
  - Problem Solving Procedure
    - Algorithm Verification
    - Computational Efficiency
  - Basic arithmetic
    - Multiplication
    - Power
    - Greatest Common Divisor
  - Primarity Testing
    - Naïve approach
    - Eratosthenes' sieve
  - Stable Matching
  - Implementation
    - Basic implementation
    - Covering today's topic

# Problem Solving Procedure

- 문제를 푸는 과정
  - 알고리즘을 개발한 후에 이 **알고리즘이 정확하다는 것**을 **증명**
  - 이 알고리즘이 충분히 빠른 알고리즘인지를 생각
  - 괜찮은 알고리즘이면 위의 알고리즘을 오류 없이 구현
- 정확성의 증명과 효율성
  - 알고리즘의 **정확성**에 대한 **수학적 증명**
  - 알고리즘의 **시간**이 얼마나 걸릴지 **계산**하는 과정

# Basic Arithmetic

- Multiplication

$$x \cdot y = \begin{cases} x & if\ y = 1 \\ x \cdot (y-1) + x & otherwise \end{cases}$$

**Verification** : Is this algorithm **true** ?

**Efficiency** : How long does it **takes** ?

# Basic Arithmetic

- Multiplication

$$x \cdot y = \begin{cases} x & if\ y = 1 \\ x \cdot (y-1) + x & otherwise \end{cases}$$

**Verification** : Is this algorithm **true** ?

# Basic Arithmetic

- Multiplication

$$x \cdot y = \begin{cases} x & if\ y = 1 \\ x \cdot (y-1) + x & otherwise \end{cases}$$

**Verification** : Is this algorithm **true** ?

Proof : use Mathematical Induction

$$x \cdot y = x + x + \cdots + x$$

# Basic Arithmetic

- Multiplication

$$x \cdot y = \begin{cases} x & if\ y = 1 \\ x \cdot (y-1) + x & otherwise \end{cases}$$

**Efficiency** : How long does it **takes** ?

# Basic Arithmetic

- Multiplication

$$x \cdot y = \begin{cases} x & if\ y = 1 \\ x \cdot (y-1) + x & otherwise \end{cases}$$

**Efficiency** : How long does it **takes** ?

**What is the unit ?**

# Computational Efficiency

- 알고리즘의 대략적인 연산횟수를 계산
  - 연산을 오래 하는 알고리즘이면 시간도 더 느릴 것이다
  - 사칙연산, 비교연산 등 연산에 관계없이 모두 1번이라 생각
  - 데이터 크기에 따라 연산 횟수가 달라짐 → **n에 대한 함수** 필요
  - 정확하게는 세기 어려움

```
1 int sum = 0;
2
3 for(int i=1;i<=n;i++)
4       sum = sum + i;
```

몇 번의 연산이 필요한가 ?

# Computational Efficiency

- 알고리즘의 대략적인 연산횟수를 계산
  - 연산을 오래 하는 알고리즘이면 시간도 더 느릴 것이다
  - 사칙연산, 비교연산 등 연산에 관계없이 모두 1번이라 생각
  - 데이터 크기에 따라 연산 횟수가 달라짐 → **n에 대한 함수** 필요
  - 정확하게는 세기 어려움

```
1 int sum = 0;
2
3 for(int i=1;i<=n;i++)
4       sum = sum + i;
```

몇 번의 연산이 필요한가 ? 약 n번

# Computational Efficiency

- 알고리즘의 대략적인 연산횟수를 계산
  - 연산을 오래 하는 알고리즘이면 시간도 더 느릴 것이다
  - 사칙연산, 비교연산 등 연산에 관계없이 모두 1번이라 생각
  - 데이터 크기에 따라 연산 횟수가 달라짐 → **n에 대한 함수** 필요
  - 정확하게는 세기 어려움

```
1 int sum = 0;
2
3 for(int i=1;i<=n;i++)
4       sum = sum + i;
```

몇 번의 연산이 필요한가 ? **O($n$)**

# Computational Efficiency

- Example

```
1 for(int i=1;i<=n;i++){
2     for(int j=1;j<=n;j++){
3         A[i][j] = 1;
4     }
5 }
6
7 for(int i=1;i<=n;i++)
8     A[i][i] = 2;
```

# Computational Efficiency

- Example

```
1 for(int i=1;i<=n;i++){
2     for(int j=1;j<=n;j++){
3         A[i][j] = 1;
4     }
5 }
6
7 for(int i=1;i<=n;i++)
8     A[i][i] = 2;
```

$O(n^2 + n)$

# Computational Efficiency

- Example

```
1 for(int i=1;i<=n;i++){
2       for(int j=1;j<=n;j++){
3             A[i][j] = 1;
4       }
5 }
6
7 for(int i=1;i<=n;i++)
8       A[i][i] = 2;
```

O($n^2$)

# Computational Efficiency

- Example

```
1 for(int i=1;i<=n;i++){
2     for(int j=1;j<=n-i;j++){
3         if(Data[j] > Data[j+1]){
4             int temp = Data[j];
5             Data[j] = Data[j+1];
6             Data[j+1] = temp;
7         }
8     }
9 }
```

# Computational Efficiency

- Example

```
1 for(int i=1;i<=n;i++){
2     for(int j=1;j<=n-i;j++){
3         if(Data[j] > Data[j+1]){
4             int temp = Data[j];
5             Data[j] = Data[j+1];
6             Data[j+1] = temp;
7         }
8     }
9 }
```

swap    loop

$$O(3 \cdot \{ (n-1) + (n-2) + \cdots + 1\})$$

# Computational Efficiency

- Example

```
1 for(int i=1;i<=n;i++){
2     for(int j=1;j<=n-i;j++){
3         if(Data[j] > Data[j+1]){
4             int temp = Data[j];
5             Data[j] = Data[j+1];
6             Data[j+1] = temp;
7         }
8     }
9 }
```

$O(3 \cdot \{\frac{n \cdot (n-1)}{2}\})$

# Computational Efficiency

- Example

```
1 for(int i=1;i<=n;i++){
2     for(int j=1;j<=n-i;j++){
3         if(Data[j] > Data[j+1]){
4             int temp = Data[j];
5             Data[j] = Data[j+1];
6             Data[j+1] = temp;
7         }
8     }
9 }
```

$O(3 \cdot \{\frac{n^2}{2}\})$

# Computational Efficiency

- Example

```
1 for(int i=1;i<=n;i++){
2     for(int j=1;j<=n-i;j++){
3         if(Data[j] > Data[j+1]){
4             int temp = Data[j];
5             Data[j] = Data[j+1];
6             Data[j+1] = temp;
7         }
8     }
9 }
```

O$(3n^2)$

# Computational Efficiency

- Example

```
1 for(int i=1;i<=n;i++){
2     for(int j=1;j<=n-i;j++){
3         if(Data[j] > Data[j+1]){
4             int temp = Data[j];
5             Data[j] = Data[j+1];
6             Data[j+1] = temp;
7         }
8     }
9 }
```

O($n^2$)

# Computational Efficiency

- Example

```
1 for(int i=1;i<=n;i++){
2     for(int j=1;j<=n-i;j++){
3         if(Data[j] > Data[j+1]){
4             int temp = Data[j];
5             Data[j] = Data[j+1];
6             Data[j+1] = temp;
7         }
8     }
9 }
```

O($n^2$)

사실 대부분의 경우에는 최고차 항만 고려하면 됨

# Basic Arithmetic

- Multiplication

$$x \cdot y = \begin{cases} x & if\ y = 1 \\ x \cdot (y-1) + x & otherwise \end{cases}$$

**Efficiency** : How long does it **takes** ?

**What is the unit ?**

# Basic Arithmetic

- Multiplication

$$x \cdot y = \begin{cases} x & if\ y = 1 \\ x \cdot (y-1) + x & otherwise \end{cases}$$

**Efficiency** : How long does it **takes** ?

**What is the unit ?**
**→ The number of operations !**

# Basic Arithmetic

- Multiplication

$$x \cdot y = \begin{cases} x & if\ y = 1 \\ x \cdot (y-1) + x & otherwise \end{cases}$$

**Efficiency** : How long does it **takes** ?

So, what is the efficiency ?

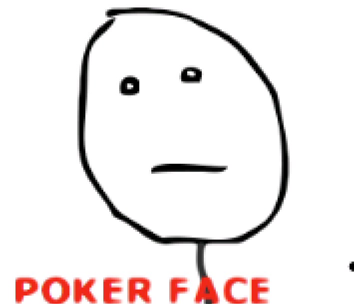
# Basic Arithmetic

- Multiplication

$$x \cdot y = \begin{cases} x & if\ y = 1 \\ x \cdot (y-1) + x & otherwise \end{cases}$$

**Efficiency** : How long does it **takes** ?

So, what is the efficiency ?



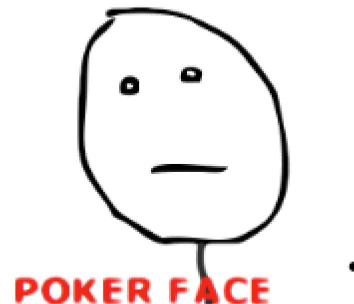
# Basic Arithmetic

- Multiplication

$$x \cdot y = \begin{cases} x & if\ y = 1 \\ x \cdot (y-1) + x & otherwise \end{cases}$$

**Efficiency** : How long does it **takes** ?

So, what is the efficiency ?
→ We need **pseudo-code**



# Basic Arithmetic

- Multiplication

$$x \cdot y = \begin{cases} x & if\ y = 1 \\ x \cdot (y-1) + x & otherwise \end{cases}$$

**Efficiency** : How long does it **takes** ?

```
1 for(int i=1;i<=y;i++)
2       result = result + x;
```

# Basic Arithmetic

- Multiplication

$$x \cdot y = \begin{cases} x & if\ y = 1 \\ x \cdot (y-1) + x & otherwise \end{cases}$$

**Efficiency** : How long does it **takes** ?

```
1 for(int i=1;i<=y;i++)
2       result = result + x;
```

**O(y)**

# Basic Arithmetic

- Multiplication

$$x \cdot y = \begin{cases} x & if\ y = 1 \\ x \cdot (y-1) + x & otherwise \end{cases}$$

**Efficiency** : How long does it **takes** ?

```
1 for(int i=1;i<=y;i++)
2       result = result + x;
```

**O(y)**

Done!

# Basic Arithmetic

- Multiplication

$$x \cdot y = \begin{cases} x & if\ y = 1 \\ x \cdot (y-1) + x & otherwise \end{cases}$$

**Efficiency** : How long does it **takes** ?

```
1 for(int i=1;i<=y;i++)
2       result = result + x;
```

**O(y)**

Done!

Is there another algorithm faster ?

# Basic Arithmetic

- Multiplication

$$x \cdot y = \begin{cases} x + (2 \cdot x \cdot \lfloor \frac{y}{2} \rfloor) & if\ y\ is\ odd \\ 2 \cdot x \cdot \lfloor \frac{y}{2} \rfloor & if\ y\ is\ even \end{cases}$$

**Verification** : Is this algorithm **true** ?

**Efficiency** : How long does it **takes** ?

# Basic Arithmetic

- Multiplication

$$x \cdot y = \begin{cases} x + (2 \cdot x \cdot \lfloor \frac{y}{2} \rfloor) & if\ y\ is\ odd \\ 2 \cdot x \cdot \lfloor \frac{y}{2} \rfloor & if\ y\ is\ even \end{cases}$$

**Verification** : Is this algorithm **true** ? Mathematical Induc.

**Efficiency** : How long does it **takes** ? $O(\log y)$

# Basic Arithmetic

- Multiplication

$$x \cdot y = \begin{cases} x + (2 \cdot x \cdot \lfloor \frac{y}{2} \rfloor) & if\ y\ is\ odd \\ 2 \cdot x \cdot \lfloor \frac{y}{2} \rfloor & if\ y\ is\ even \end{cases}$$

**Verification** : Is this algorithm **true** ? Mathematical Induc.

**Efficiency** : How long does it **takes** ? $O(\log y)$

그냥 x*y 하면 될걸 무슨 이런 헛짓을…

# Basic Arithmetic

- Power
  - Derive the efficient algorithm to calculate $x^y$
  - Find correct algorithm which takes $O(\log y)$
  - Also, give me the pseudo-code

# Basic Arithmetic

- Euclid's Rule

$$\gcd(x, y) = \gcd(x \ mod \ y, y)$$

Claim. G = $\gcd(x, y) = \gcd(x - y, y) \quad if \ x \geq y$

Proof.

# Basic Arithmetic

- Euclid's Rule

$$\gcd(x, y) = \gcd(x \; mod \; y, y)$$

Claim. G = $\gcd(x, y) = \gcd(x - y, y) \quad if \; x \geq y$

Proof. Any divisor of both x and y also divides x-y

Therefore, $\gcd(x, y)$ is the divisor or $\gcd(x - y, y)$

$\therefore \; \gcd(x, y) \leq \gcd(x - y, y)$

# Basic Arithmetic

- Euclid's Rule

$$\gcd(x, y) = \gcd(x \ mod \ y, y)$$

Claim. G = $\gcd(x, y) = \gcd(x - y, y) \quad if \ x \geq y$

Proof. Also, any divisor of both $x - y$ and $y$ also divides x

Therefore, $\gcd(x, y) \geq \gcd(x - y, y)$

$\therefore \gcd(x, y) = \gcd(x - y, y)$

# Basic Arithmetic

- Euclid's Rule

$$\gcd(x, y) = \gcd(x \; mod \; y, y)$$

Then G = $\gcd(x, y) = \gcd(x \; mod \; y \;, y) \quad if \; x \geq y$

# Basic Arithmetic

- Euclid's Rule

$$\gcd(x, y) = \gcd(x \ mod \ y, y)$$

$$\gcd(248, 32)$$

# Basic Arithmetic

- Euclid's Rule

$$\gcd(x, y) = \gcd(x \ mod \ y, y)$$

$$\gcd(248, 32)$$
$$\gcd(24, 32)$$

# Basic Arithmetic

- Euclid's Rule

$$\gcd(x, y) = \gcd(x \; mod \; y, y)$$

$$\gcd(248, 32)$$
$$\gcd(24, 32)$$
$$\gcd(32, 24)$$

# Basic Arithmetic

- Euclid's Rule

$$\gcd(x, y) = \gcd(x \; mod \; y, y)$$

$$\gcd(248, 32)$$
$$\gcd(24, 32)$$
$$\gcd(32, 24)$$
$$\gcd(8, 24)$$

# Basic Arithmetic

- Euclid's Rule

$$\gcd(x, y) = \gcd(x \ mod \ y, y)$$

$$\gcd(248, 32)$$
$$\gcd(24, 32)$$
$$\gcd(32, 24)$$
$$\gcd(8, 24)$$
$$\gcd(24, 8)$$

# Basic Arithmetic

- Euclid's Rule

$$\gcd(x, y) = \gcd(x \bmod y, y)$$

$$\gcd(248, 32)$$
$$\gcd(24, 32)$$
$$\gcd(32, 24)$$
$$\gcd(8, 24)$$
$$\gcd(24, 8)$$

8 divides 24 ! Therefore, $\gcd(24, 8) = 8$

# Basic Arithmetic

- Euclid's Rule

$$\gcd(x, y) = \gcd(x \; mod \; y, y)$$

How long does it takes ?

# Basic Arithmetic

- Euclid's Rule

$$\gcd(x, y) = \gcd(x \ mod \ y, y)$$

How long does it **takes** ?

Claim. If $a \geq b$ then $a \ mod \ b < a/2$

Consider the cases $b \leq a/2$ and $b > a/2$

# Basic Arithmetic

- Euclid's Rule

$$\gcd(x, y) = \gcd(x \; mod \; y, y)$$

How long does it **takes** ?

Claim. If $a \geq b$ then $a \; mod \; b \leq a/2$

Consider the cases $b \leq a/2$ and $b > a/2$

If $b \leq a/2$ then $a \; mod \; b < b \leq a/2$

$b > a/2$ then $a \; mod \; b = a - b < a/2$

# Basic Arithmetic

- Euclid's Rule

$$\gcd(x, y) = \gcd(x \ mod \ y, y)$$

How long does it **takes** ?

After two consecutive round, a and b are at least half

→ $\mathbf{O(\log n)}$

# Primarity Testing

- Problem Definition

  Given p, determine whether p is prime number or not

  Given a interval [s, e], show all of the prime numbers in the interval

- Approach
  - Naïve approach
  - Eratosthenes' seive

# Primarity Testing

Given p, determine whether p is prime number or not

- Naïve approach

  For all $2 \leq i \leq p-1$, test whether $i$ divides p

  If there is such $i$, then p is NOT prime number

  Otherwise, p is prime number

**Verification** : Is this algorithm **true** ?

**Efficiency** : How long does it **takes** ?

# Primarity Testing

Given p, determine whether p is prime number or not

- Naïve approach

  For all $2 \leq i \leq p-1$, test whether $i$ divides p

  If there is such $i$, then p is NOT prime number

  Otherwise, p is prime number

**Verification** : Is this algorithm **true** ? By definition of prime number

**Efficiency** : How long does it **takes** ? $O(p)$

# Primarity Testing

Given p, determine whether p is prime number or not

- Naïve approach

For all $2 \leq i \leq p-1$, test whether $i$ divides p

If there is such $i$, then p is NOT prime number

Otherwise, p is prime number

**Verification** : Is this algorithm **true** ? By definition of prime number

**Efficiency** : How long does it **takes** ? O($p$)

Is there another algorithm faster ?

# Primarity Testing

Given p, determine whether p is prime number or not

- We don't have to consider all such $i$ !

  It is sufficient to consider $2 \leq i \leq \sqrt{p}$, **Why ?**

**Verification** : Is this algorithm **true** ? By definition of prime number

**Efficiency** : How long does it **takes** ? O($p$)

# Primarity Testing

Given p, determine whether p is prime number or not

- We don't have to consider all such $i$ !

  It is sufficient to consider $2 \leq i \leq \sqrt{p}$, **Why ?**

  If p is tested as prime number when $i > \sqrt{p}$, It must be tested before !

**Verification** : Is this algorithm **true** ? By definition of prime number

**Efficiency** : How long does it **takes** ? O($p$)

# Primarity Testing

Given p, determine whether p is prime number or not

- We don't have to consider all such $i$ !

  It is sufficient to consider $2 \leq i \leq \sqrt{p}$, **Why ?**

  If p is tested as prime number when $i > \sqrt{p}$, It must be tested before !

**Verification** : Is this algorithm **true** ? By definition of prime number

**Efficiency** : How long does it **takes** ? **O($\sqrt{p}$)**

# Primarity Testing

Given a interval [s, e], show all of the prime numbers in the interval

- Naïve approach

  For all numbers $x$ in the interval, test it

**Verification** : Is this algorithm **true** ? Trivial

**Efficiency** : How long does it **takes** ? **O($n\sqrt{n}$)**

# Primarity Testing

Given a interval [s, e], show all of the prime numbers in the interval

- Naïve approach

  For all numbers $x$ in the interval, test it

**Verification** : Is this algorithm **true** ?   Trivial

**Efficiency** : How long does it **takes** ?  **$O(n\sqrt{n})$**

Is there another algorithm faster ?

# Eratosthenes' seive

- Simple Algorithm

1. '1' is not a prime number by definition
2. Pick the smallest value among we have, which is 2
3. Erase all the numbers divided by 2 ( except 2 )
4. Pick the smallest value among we have, which is 3
5. Erase all the numbers divided by 3 ( except 3 )
6. Repeat this procedure

# Eratosthenes' seive

- Simple Algorithm

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

# Eratosthenes' seive

- Simple Algorithm

'1' is not a prime number by definition

| 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

# Eratosthenes' seive

- Simple Algorithm

Pick the smallest value among we have, which is 2

| 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

# Eratosthenes' seive

- Simple Algorithm

Erase all the numbers divided by 2 ( except 2 )

# Eratosthenes' seive

- Simple Algorithm

Pick the smallest value among we have, which is 3

# Eratosthenes' seive

- Simple Algorithm

Erase all the numbers divided by 3 ( except 3 )

# Eratosthenes' seive

- Simple Algorithm

Pick the smallest value among we have, which is 5

# Eratosthenes' seive

- Simple Algorithm

Erase all the numbers divided by 5 ( except 5 )

# Eratosthenes' seive

- Simple Algorithm

Repeat this procedure

# Eratosthenes' seive

- Analysis

**Verification** : Is this algorithm **true** ?      It looks like …

**Efficiency** : How long does it takes ?

# Eratosthenes' seive

- Analysis

**Verification** : Is this algorithm **true** ? It looks like …

**Efficiency** : How long does it **takes** ?

Suppose that we want to find all the prime numbers in [1, n]

Let's focus on a integer $x$ in the interval

Then $x$ is "clicked" as many as the number of divisor of $x$

It must be bounded by O($\log x$) for each $x$

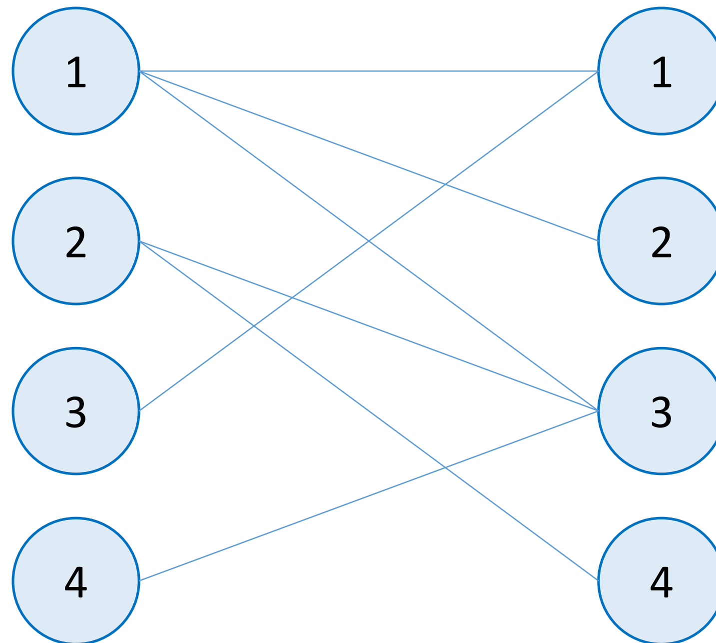
Therefore, It takes **O($n \log n$)**

Much better than O($n\sqrt{n}$) !

# Bipartite Matching

- Problem Definition (weak)

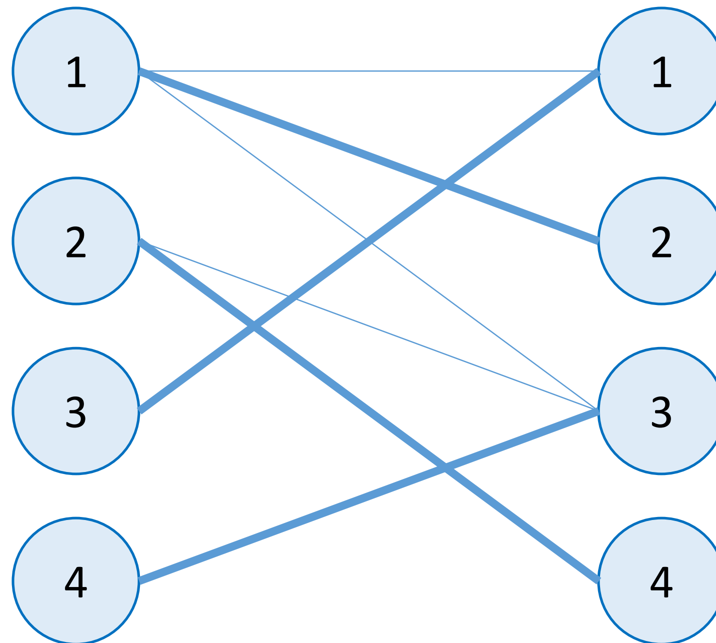
Maximize the number of matched pair on bipartite graph



# Bipartite Matching

- Problem Definition (weak)

Maximize the number of matched pair on bipartite graph



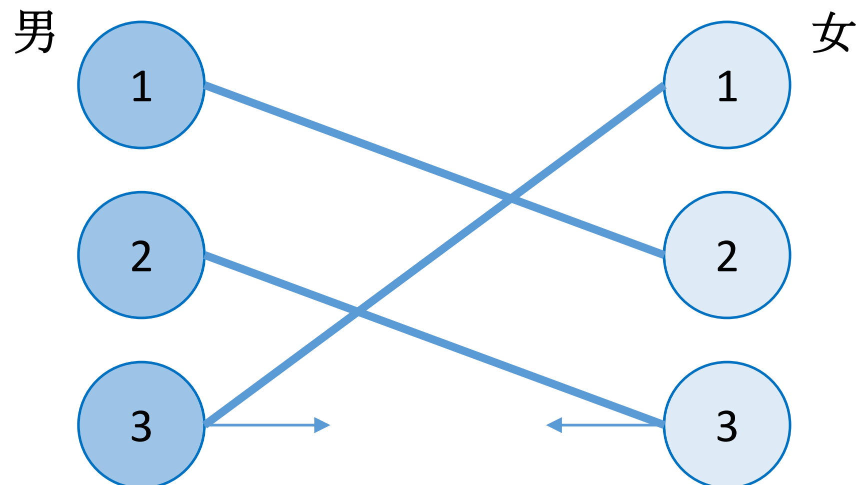
# Stable Matching

- Problem Definition (weak)

  N명의 남자와 여자가 미팅을 하기 위해서 만남

  남자와 여자 모두 원하는 상대방의 우선순위를 정함

  **서로 짝이 아닌 두 남녀**가 **자신의 짝보다** **상대방을 더** 선호하면 ?




난 1번보다 3번이 더 좋은데…

난 1번보다 3번이 더 좋은데…

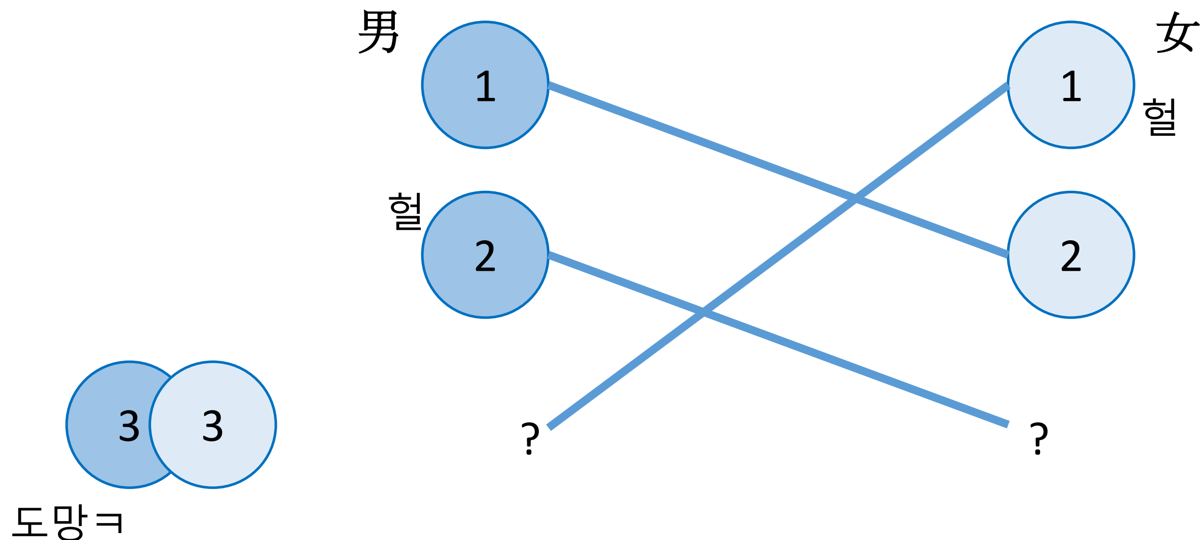
# Stable Matching

- Problem Definition (weak)

  N명의 남자와 여자가 미팅을 하기 위해서 만남

  남자와 여자 모두 원하는 상대방의 우선순위를 정함

  **서로 짝이 아닌 두 남녀**가 **자신의 짝보다 상대방을 더** 선호하면 ?



# Stable Matching

- Problem Definition (weak)

  N명의 남자와 여자가 미팅을 하기 위해서 만남

  남자와 여자 모두 원하는 상대방의 우선순위를 정함

  **서로 짝이 아닌 두 남녀**가 **자신의 짝보다** **상대방을 더** 선호하면 ?

  이런 불상사가 나지 않도록 짝을 짓는 것이 가능한가?

# Stable Matching

- Problem Definition (weak)

N명의 남자와 여자가 미팅을 하기 위해서 만남
남자와 여자 모두 원하는 상대방의 우선순위를 정함
**서로 짝이 아닌 두 남녀**가 **자신의 짝보다** **상대방을 더** 선호하면 ?

이런 불상사가 나지 않도록 짝을 짓는 것이 가능한가?

→ Yes ! 가능하다는 것을 해를 구하는 알고리즘으로 증명
2012년 노벨 경제학상을 받게 한 알고리즘 (Gale-Shapely Algorithm)

# Stable Matching

- Problem Definition (weak)
  1. 처음에 남성이 모두 가장 선호하는 여성에게 프로포즈를 함
  2. 여성이 그 중 가장 마음에 드는 남성을 고르고 나머지는 퇴짜
  3. 퇴짜맞은 남성은 그 다음으로 선호하는 여성이 파트너가 있던 말던 프로포즈를 함
  4. 여성은 현재 자신의 파트너보다 프로포즈 한 남성이 더 마음에 든다면 자신의 파트너에게 퇴짜를 놓음
  5. 이 과정을 계속해서 반복 !

# Stable Matching

- Correctness
  - Is it terminate ? // 언젠간 **종료**하는가 ?
  - Is there any lonely man or woman ? // **짝 없는 사람**이 존재하는가 ?
  - Is it stable ? // **불상사**가 없는가 ?

# Stable Matching

- Correctness
  - Is it terminate ? // 언젠간 **종료**하는가 ?

# Stable Matching

- Correctness
  - Is it terminate ? // 언젠간 **종료**하는가 ?

각 남성은 많아야 n명에게 프로포즈를 하고, 같은 사람에게 두 번 하지 않는다.
따라서 이 과정은 언젠간 종료된다.

# Stable Matching

- Correctness
  - Is there any lonely man or woman ? // **짝 없는 사람**이 존재하는가 ?

# Stable Matching

- Correctness
  - Is there any lonely man or woman ? // **짝 없는 사람**이 존재하는가 ?

여성이 짝이 없다고 가정하자. 그렇다면 한 명도 프로포즈한 남성이 없다.
그러면 모든 남성이 n-1명의 여성과 짝을 이룬다는 것이므로 모순.

남성이 짝이 없다고 가정하자. 그렇다면 모두 퇴짜를 맞았다는 것이다.
그러면 모든 여성이 n-1명의 남성과 짝을 이룬다는 것이므로 모순.

# Stable Matching

- Correctness
  - Is it stable ?    // **불상사**가 없는가 ?

# Stable Matching

- Correctness
  - Is it stable ? // **불상사**가 없는가 ?

불상사가 생긴 남성과 여성이 존재한다고 가정하자.
편의상 현재 파트너를 X, 불상사가 생기는 파트너를 Y 이라고 하자.
남성은 Y를 X보다 더 좋아한다. 그러면 X보다 Y에게 더 먼저 프로포즈 했을 것.
여성은 현재 이 남성과 파트너가 아니므로 퇴짜를 놓았다는 뜻이다.
하지만 여성 역시 현재 자신의 파트너보다 이 남성이 더 좋은 상황이다.
더 좋은 사람을 퇴짜 놓은 상황이므로 모순.

# Question ?

- To-do List
  - 코딩 합시다 ☺