NP-Hard Graph Problem

สมชาย ประสิทธิ์จูตระกูล

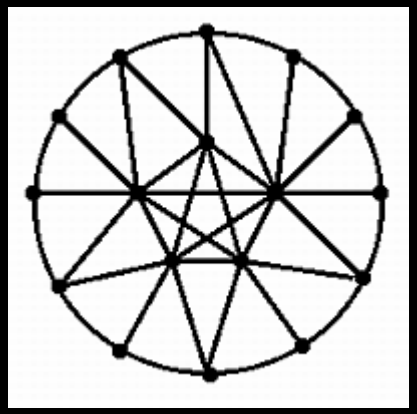
ภาควิชาวิศวกรรมคอมพิวเตอร์

จุฬาลงกรณ์มหาวิทยาลัย

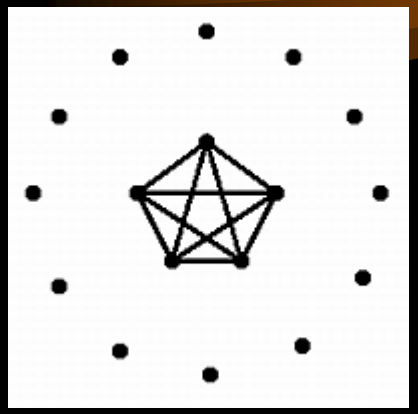
Outline

- NP-hard graph problems
- Decision problems
- Reduction
- Completeness
- NP-completeness and NP-hard
- Solving NP-hard problems

Clique

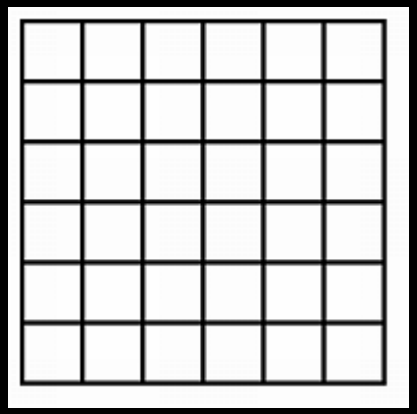


input

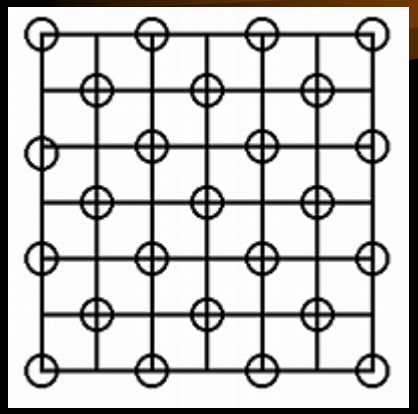


output

Independent Set

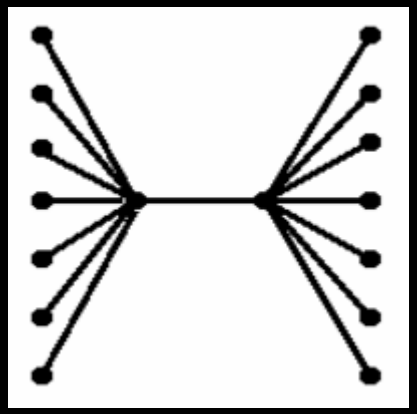


input

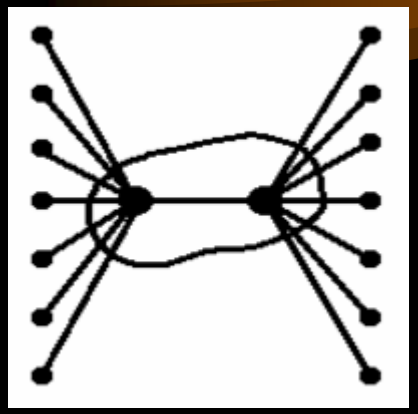


output

Vertex Cover

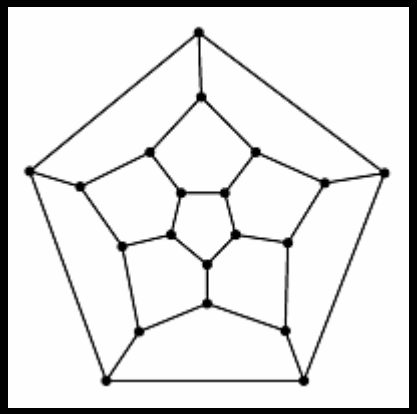


input

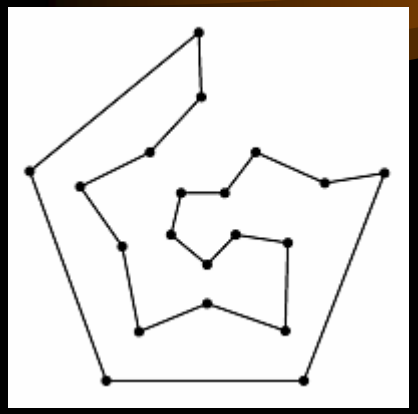


output

Hamiltonian Cycle

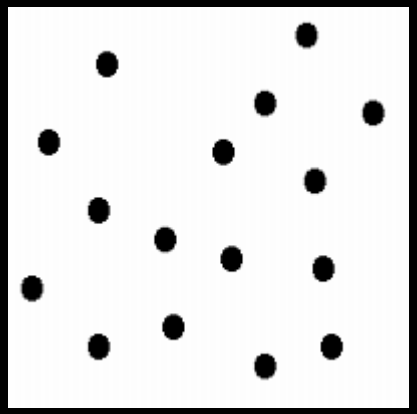


input

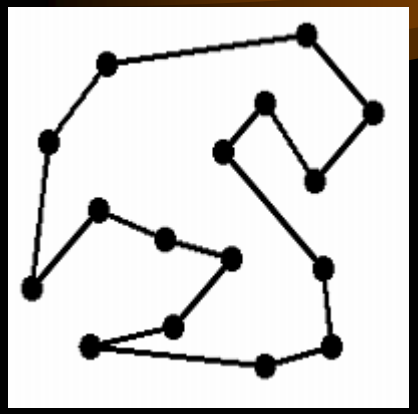


output

Traveling Salesperson Problem

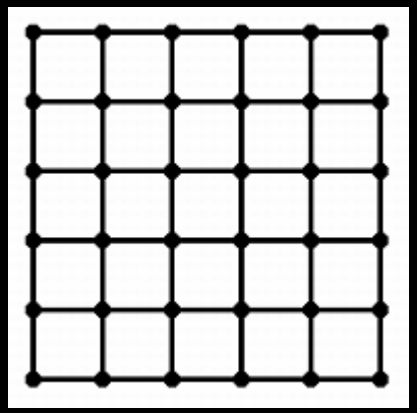


input

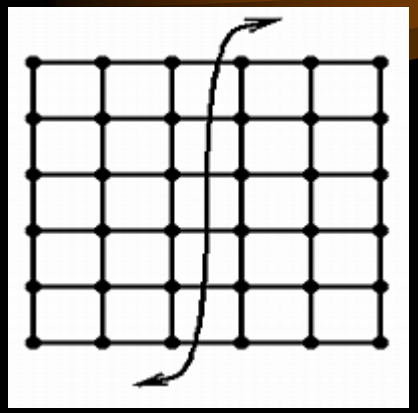


output

Graph Partition

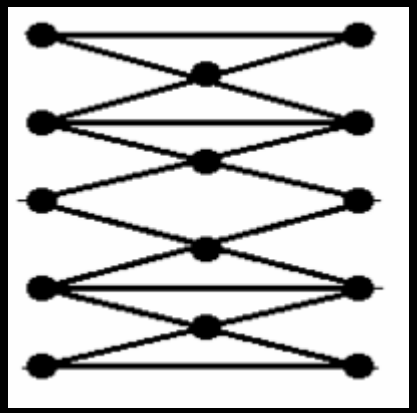


input

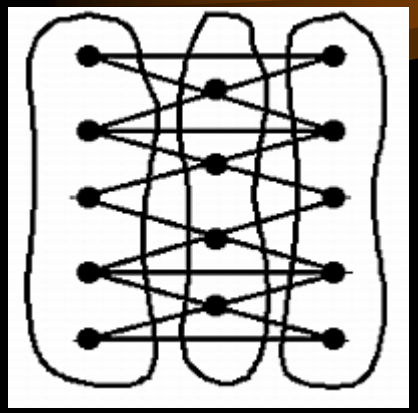


output

Vertex Coloring

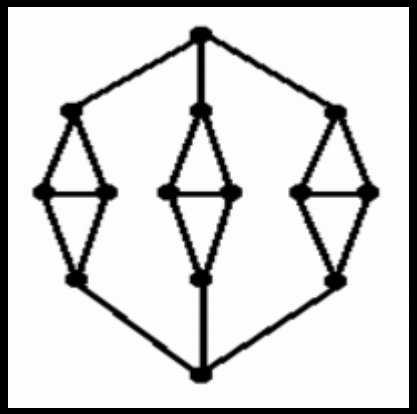


input

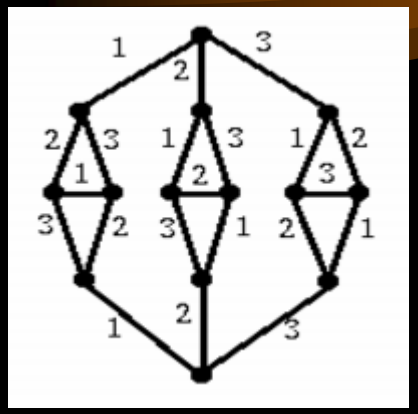


output

Edge Coloring

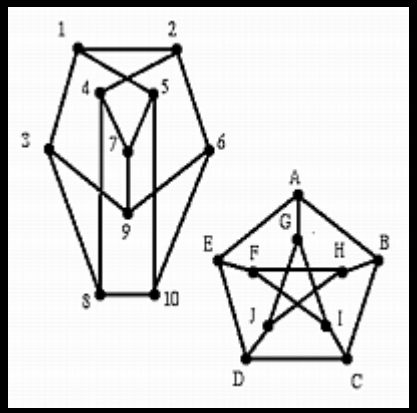


input

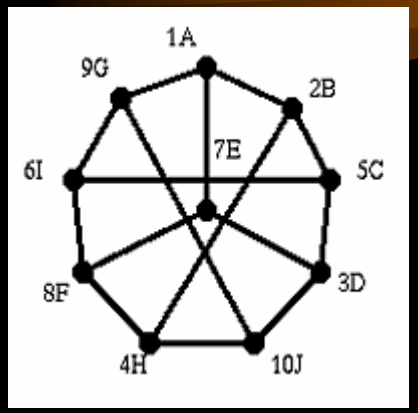


output

Graph Isomorphism

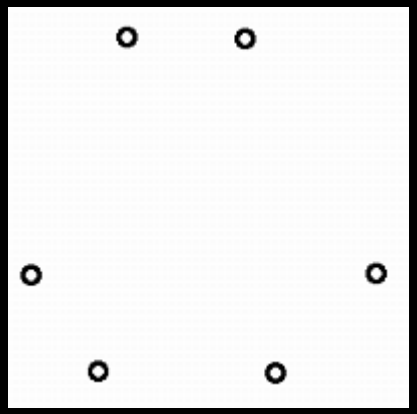


input

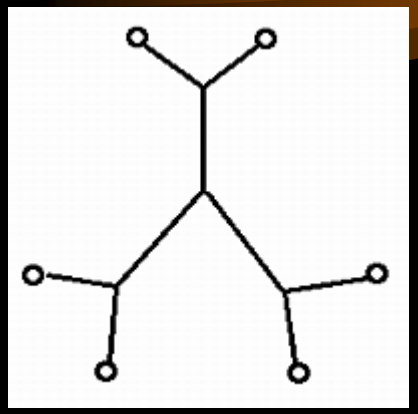


output

Steiner Tree

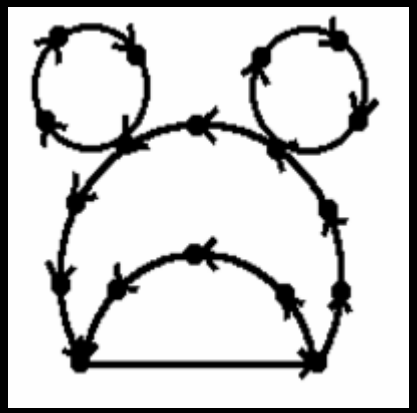


input

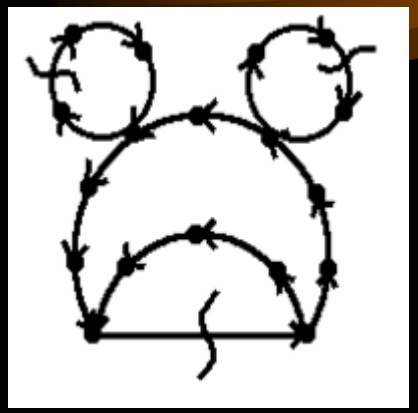


output

Feedback Edge

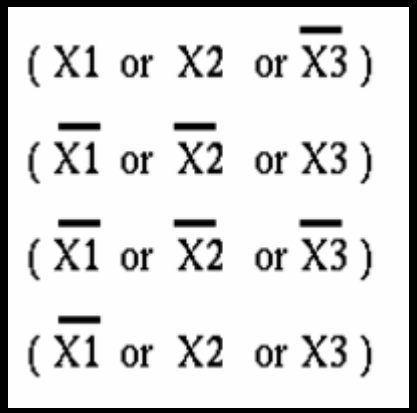


input

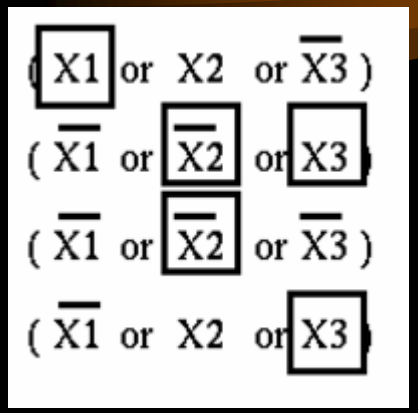


output

Satisfiability



input

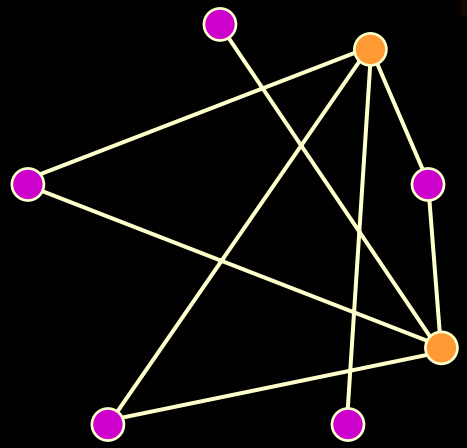


output

Decision Problems

- ปัญหาที่ให้คำตอบสองแบบเท่านั้น คือ “จริง” หรือ “เท็จ”
  - กราฟ G มี vertex cover ขนาดไม่เกิน k หรือไม่ ?
  - กราฟ G มี clique ขนาดอย่างน้อย k หรือไม่ ?
  - กราฟ G มี salesman tour ความยาวไม่เกิน k หรือไม่ ?
  - ใช้สีไม่เกิน k สีในการทาสี vertices ของกราฟ G ได้หรือไม่ ?
  - ...

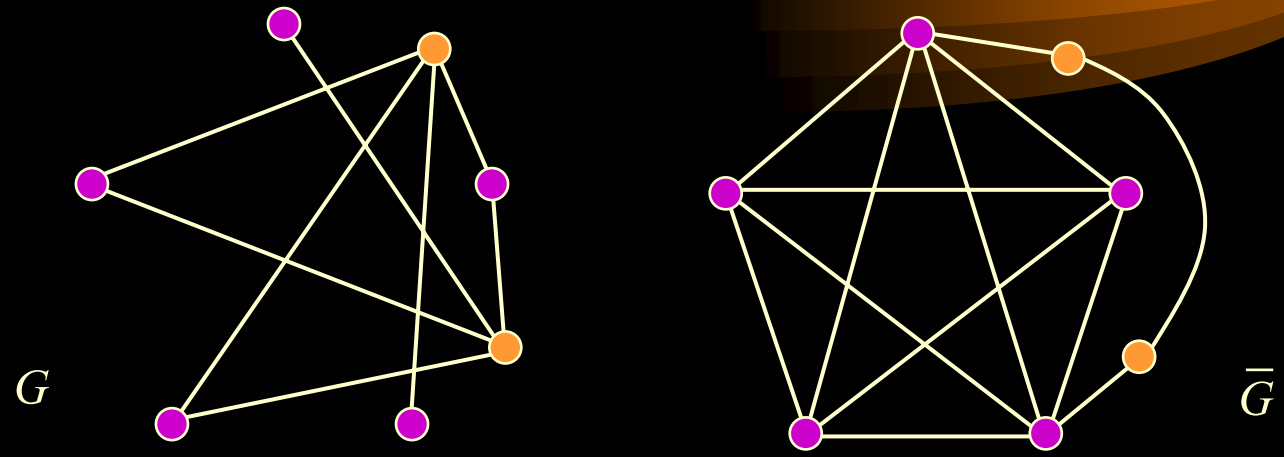
Vertext Cover → *Independent Set*



```
boolean VC( Graph g, int k ) {
  return IS( g, g.numNodes - k );
}
```

Vetex cover ไม่ยากกว่า Independent set

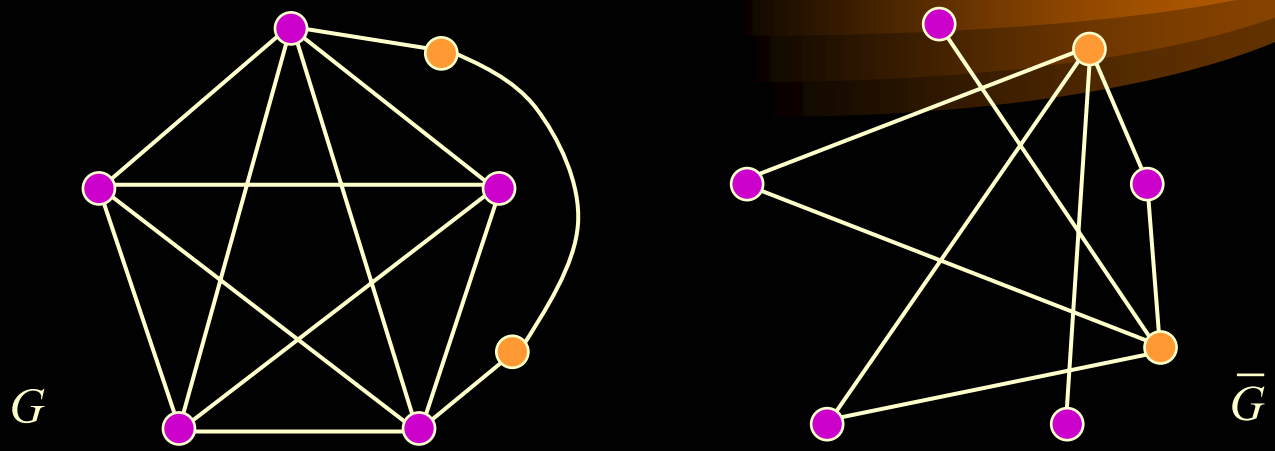
Independent Set ⇒ *Clique*




```
boolean IS( Graph g, int k ) {
  return Clique( complement(g), k );
}
```

Independent set ไม่ยากกว่า Clique

Clique ⟹ *Vertex Cover*




```
boolean Clique( Graph g, int k ) {
  return VC( complement(g), g.numNodes - k );
}
```

Clique ไม่ยากกว่า Vertex cover

Completeness

- Vetex cover ไม่ยากกว่า Independent set
- Independent set ไม่ยากกว่า Clique
- Clique ไม่ยากกว่า Vertex cover
- ทั้งสามปัญหามีความยาก (ง่าย) เทียบเท่ากัน

NP-Complete Problems

- กลุ่มปัญหา (decision problem) ที่มีความยาก (ง่าย) เทียบเท่ากันหมด
- ทุกๆ ปัญหาในกลุ่มเป็นแบบ NP
  - เป็นปัญหาที่มีวิธีทวนสอบคำตอบได้เร็ว (polynomial time)

“vertices เหล่านี้ เป็น vertex cover ของกราฟ G ที่มีขนาดอย่างมาก k หรือไม่ ? ”

“vertices เหล่านี้ เป็น clique ของกราฟ G ที่มีขนาดอย่างน้อย k หรือไม่ ? ”

“เส้นทางนี้ เป็น salesman tour ของกราฟ G ที่มีความยาวอย่างมาก k หรือไม่ ? ”

“เส้นทางนี้ เป็น Hamiltonian cycle ของกราฟ G หรือไม่ ? ”

ปัจจุบัน

- มี NP-complete problems เป็นพัน และพบเพิ่มขึ้นเรื่อยๆ
- พบ NP-complete problem ในโลกแห่งความเป็นจริง
- เป็นปัญหาที่ใครๆ ก็บอกว่า “ยาก” แต่พิสูจน์ไม่ได้

NP-Hard

- Q เป็นปัญหาแบบ NP-hard
  ถ้า Q เป็นปัญหาที่ไม่ง่ายกว่าปัญหาใน NP-complete problems
- Q ไม่จำเป็นต้องเป็น decision problem
- ปัญหาสงสัยว่ายากทั้งหลายที่พบมักเป็น NP-hard problem

แล้วจะทำอย่างไร ?

- พบว่า Q เป็น NP-hard
- สบายใจ : ไม่มีใครๆ ในโลกแก้ Q ได้ดีที่สุดสำหรับทุกๆ input ได้เร็วๆ
- หนักใจ : จะแก้ Q ได้ “ดีๆ” สำหรับ input ส่วนใหญ่ ได้เร็วๆ ได้อย่างไร
  - greedy algorithms
  - backtracking, branch and bound
  - local search
  - approximation algorithms

Vertex Cover

- Exhaustive search
- Backtracking
- Greedy heuristics
- Approximation algorithm

Enumerate All Subsets

```
01: enumSubsets( x[0..n-1], k ) {
02:   if ( k == n ) check( x )
03:   else {
04:     x[k] = 1
05:     enumSubsets( x, k+1 )
06:     x[k] = 0
07:     enumSubsets( x, k+1 )
08:   }
09: }
```

Vertex Cover : Exhaustive Search

```
01: getVertexCover( G, vc[0..n-1], x[0..n-1], k ) {
02:   if ( k == n )
03:     if ( isVertexCover( G, x ) &&
04:          sizeofVC(x) < sizeofVC(vc) ) {
05:       vc[0..n-1] = x[0..n-1]
06:     }
07:   } else {
08:     x[k] = 1
09:     vc = getVertexCover( G, vc, x, k+1 )
10:     x[k] = 0
11:     vc = getVertexCover( G, vc, x, k+1 )
12:   }
13:   return vc;
14: }
```

Vertex Cover : Backtracking

```
01: getVertexCover( G, vc[0..n-1], x[0..n-1], k )
02:   if ( k == n )
03:     if ( isVertexCover( G, x ) &&
04:          sizeofVC(x) < sizeofVC(vc) ) {
05:       vc[0..n-1] = x[0..n-1]
06:     }
07:   } else {
08:     if (size(x) < size(vc)-1) {
09:       x[k] = 1
10:       vc = getVertexCover( G, vc, x, k+1 )
11:     }
12:     x[k] = 0
13:     vc = getVertexCover( G, vc, x, k+1 )
14:   }
15:   return vc;
16: }
```

Vertex Cover : Backtracking

```
01: getVertexCover( G, vc[0..n-1], x[0..n-1], k )
02:   if ( k == n )
03:     if ( isVertexCover( G, x ) &&
04:          sizeofVC(x) < sizeofVC(vc) ) {
05:       vc[0..n-1] = x[0..n-1]
06:     }
07:   } else {
08:     x[k] = 0
09:     vc = getVertexCover( G, vc, x, k+1 )
10:     if (size(x) < size(vc)-1) {
11:       x[k] = 1
12:       vc = getVertexCover( G, vc, x, k+1 )
13:     }
14:   }
15:   return vc;
16: }
```

Enumerate All Subsets

```
01: enumSubsets( x[0..n-1], k ) {
03:   check( x, k )
04:   j = (k == 0 ? -1, x[k-1])
04:   for ( i=j+1 to n-1 ) {
05:     x[k] = i
06:     enumSubsets( x, k+1 )
07:   }
08: }
```

Vertex Cover : Exhaustive Search

```
01: getVertexCover( G, vc[], x[0..n-1], k )
02:   if ( isVertexCover( G, x ) && k < size(vc) ) {
03:     vc[] = x[0..k-1]
04:   }
05:   j = (k == 0 ? -1, x[k-1])
06:   for ( i=j+1 to n-1 ) {
07:     x[k] = i
08:     vc = getVertexCover( G, vc, x, k+1 )
09:   }
10:   return vc;
11: }
```

Vertex Cover : Backtracking

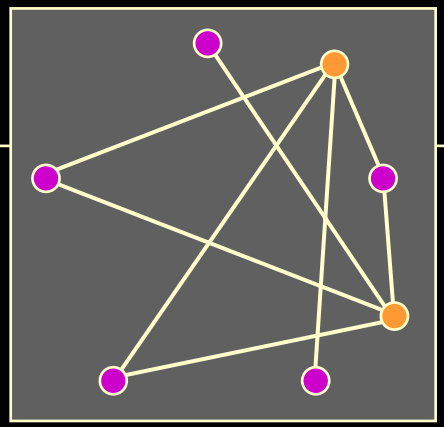
```
01: getVertexCover( G, vc[], x[0..n-1], k )
02:   if ( isVertexCover( G, x ) && k < size(vc) ) {
03:     vc[] = x[0..k-1]
04:   } else {
05:     j = (k == 0 ? -1, x[k-1])
06:     for ( i=j+1 to n-1 ) {
07:       x[k] = i
08:       vc = getVertexCover( G, vc, x, k+1 )
09:     }
10:   }
11:   return vc;
12: }
```

Vertex Cover : Greedy

```
01: vertexCover_greedy( G=(V,E) )
02: {
03:   Vc = empty set
04:   while ( E is not empty ) {
05:     choose v in V whose degree is maximum
07:     Vc = Vc U {v}
08:     V  = V - {v}
09:     remove from E every edge incident on v
10:   }
11:   return Vc
12: }
```

อาจได้ผลลัพธ์ที่มีขนาดเป็น

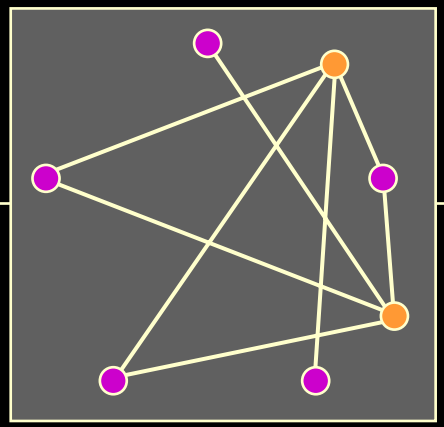
O(lg n) เท่าของชุดเล็กสุด



Vertex Cover : Approximation

```
01: VertexCover_approx( G=(V,E) )
02: {
03:   Vc = empty set
04:   E' = E
05:   while ( E' is not empty ) {
06:     choose (u,v) an arbitary edge of E'
07:     VC = Vc U {u, v}
08:     remove from E' every edge incident on u or v
09:   }
10:   return Vc
11: }
```

รับประกันได้ว่าผลลัพธ์มีขนาดไม่เกิน 2 เท่าของชุดเล็กสุด



Traveling Salesperson Problem

- Exhaustive search
- Greedy heuristics
- Approximation algorithm
- Local search
  - simulated annealing

Enumerate All Permutations

```
01: EnumPermutations( x[0..n-1], k ) {
02:   if ( k == 0 ) for(i=0 to n-1) x[i] = i
03:   if ( k == n ) Check( x[0..n-1] )
04:   else {
05:     for (i=k to n) {
06:       Swap( x[k], x[i] );
07:       EnumPermutations( x[0..n-1], k+1 )
08:       Swap( x[i], x[k] );
09:     }
10: }
```

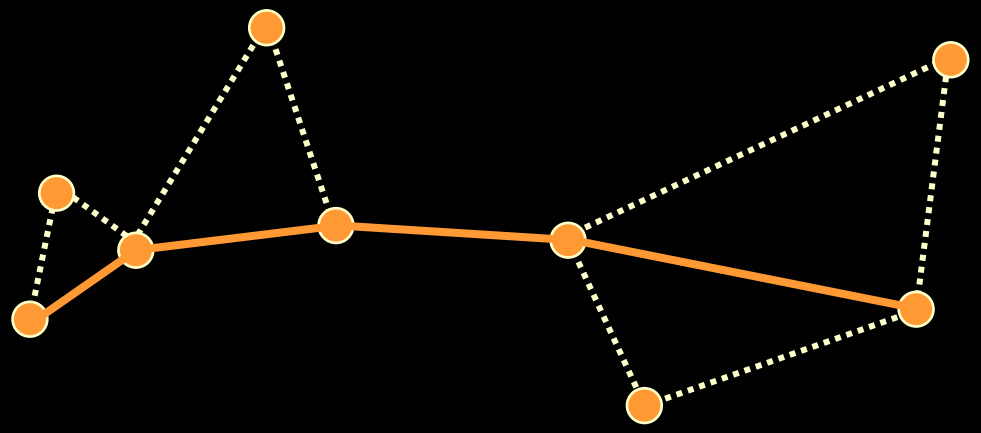
TSP : Exhaustive Search

```
01: TSP( G, tour, x[0..n-1], k ) {
02:   if ( k == 0 ) for(i=0 to n-1) x[i] = i
03:   if ( k == n ) {
04:     if length(G,x) < length(G,tour) {
05:       tour = x
06:     }
07:   } else {
08:     for (i=k to n) {
09:       Swap( x[k], x[i] );
10:       EnumPermutations( x[0..n-1], k+1 )
11:       Swap( x[i], x[k] );
12:     }
13:   }
14:   return tour
15: }
```

Greedy Algorithm

- Incrementally insert the point v into partial tour T such that

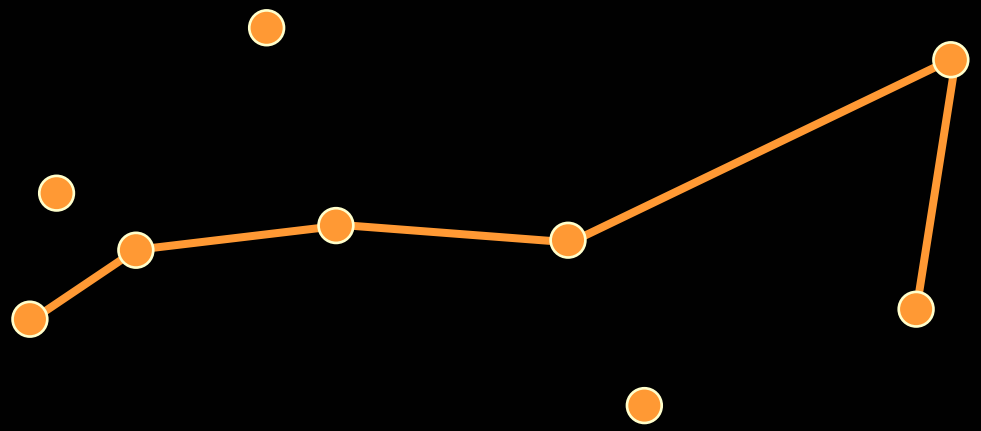
$$\max_{v \in V} \left\{ \min_{1 \le i \le |T|} \left(d(v, v_i) + d(v, v_{i+1}) \right) \right\}$$



Greedy Algorithm

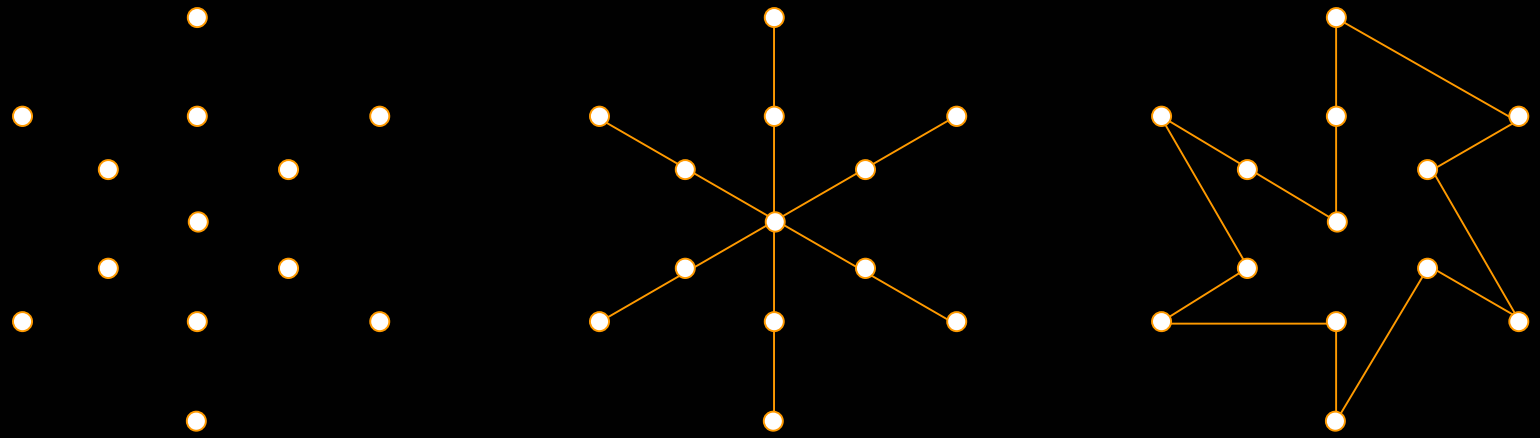
- Incrementally insert the point v into partial tour T such that

$$\max_{v \in V} \left\{ \min_{1 \le i \le |T|} \left(d(v, v_i) + d(v, v_{i+1}) \right) \right\}$$



TSP : Approximaton Algorithm

- Euclidean TSP : $d(i, j) \le d(i,k) + d(k, j)$

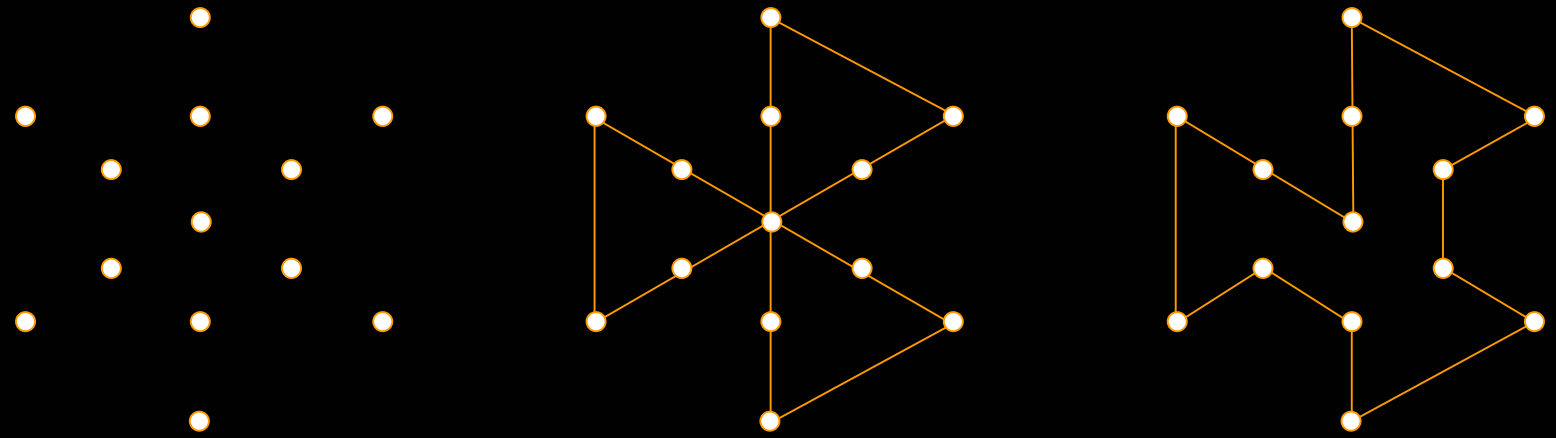


```
01: EuclideanTSPapprox( G=(V,E) ) {
02:    Tmst = MST( G )
03:    Tour = PreorderTraversal( Tmst )
04: }
```

รับประกันได้ว่าผลลัพธ์ยาวไม่เกิน 2 เท่าของเส้นทางสั้นสุด

TSP : Approximaton Algorithm

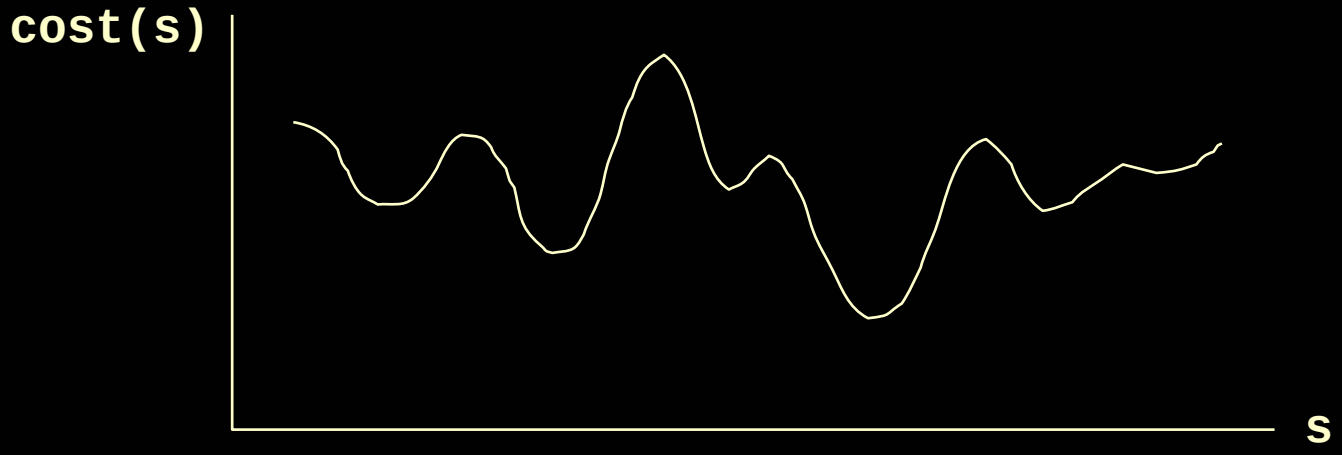
- Euclidean TSP : $d(i, j) \le d(i,k) + d(k, j)$



รับประกันได้ว่าผลลัพธ์ยาวไม่เกิน 1.5 เท่าของเส้นทางสั้นสุด

Local Search

```
01: localSearch( ) {
02:   s = initialSolution()
03:   while not terminate(s) {
04:     s' = select( next(s) )
05:     if ( accept(s,s') ) then s = s'
06:   }
07: }
```



TSP : Next Solution

0, 2, 4, 5, 6, 7, 1, 3, 9, 8, 10, 11

0, 2, 4, 8, 9, 3, 1, 7, 6, 5, 10, 11

Simulated Annealing

```
01: simulatedAnnealing( ) {
02:   s = initialSolution()
03:   t = initialTemperature()
04:   while ( isNotFreezing(t) ) {
05:     s' = select( next(s) )
06:     d = cost(s') - cost(s)
07:     if ( d < 0 )
08:       s = s'
09:     else
10:       if ( e^-d/kt > random(0,1) ) then s = s'
11:     t = decreaseTemperature(t)
12:   }
13: }
```