Data Structure and Algorithms Design

Vincent Chau （周）

2021.11.10

Summary

- Divide and Conquer √
- Dynamic programming √
- Greedy Algorithm
- Tree Structure

Greedy Algorithm

Greedy Algorithm

Concept

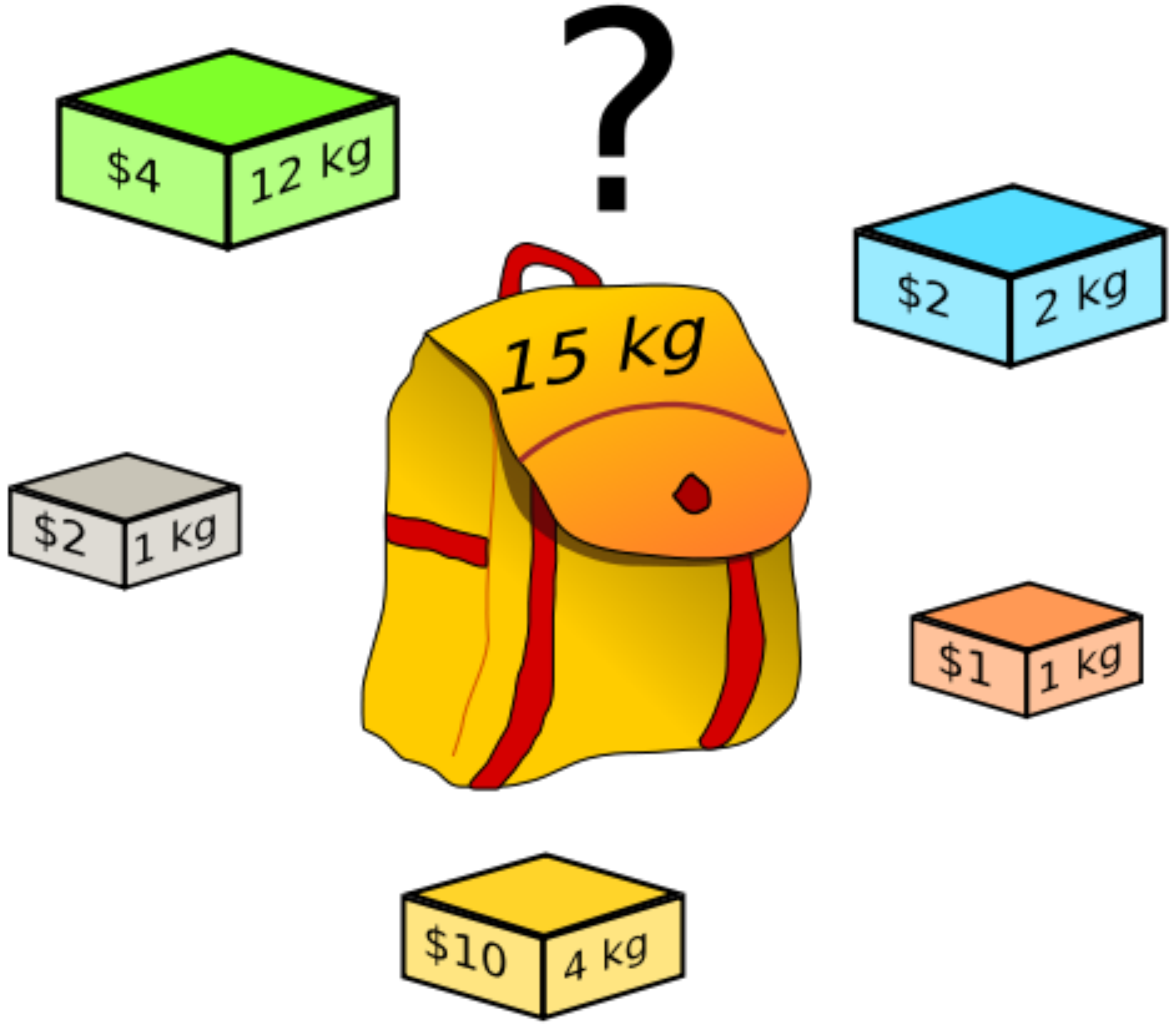
1. A greedy algorithm is a simple, intuitive algorithm that is used in optimization problems
2. The algorithm makes the optimal choice at each step as it attempts to find the overall optimal way to solve the entire problem
3. It usually does not revoke former decisions
4. Fast running time

0-1 Knapsack problem

- Weight capacity K
- Set of n items. Each item i has
  - weight w_i
  - value v_i

Goal: find subset of items such that

  - total weight is less than K
  - total value is maximized



0-1 Knapsack problem

Weight capacity K=25

item i	weight w_i	value v_i
1	22	12
2	10	9
3	9	9
4	7	6

```
S ← ∅;capacity ← 0;
for i = 1,…,n:
  if wi + capacity ≤ K
    S ← S ∪ {i}
    capacity ← wi + capacity
  end if
end for
return S
```

Running time:

Recall that running time of DP is $2 \times K \times n = 2 \times 25 \times 4$

Improvement?

Weight capacity K=25

item i	weight w_i	value v_i
1	9	9
2	10	9
3	7	6
4	22	12

Change order of items: sort by decreasing ratio v_i/w_i

```
S ← ∅;capacity ← 0;
for i = 1,…,n:
  if wi + capacity ≤ K
    S ← S ∪ {i}
    capacity ← wi + capacity
  end if
end for
return S
```

Running time:

How bad it is?

Weight capacity K=25

item i	weight w_i	value v_i
1		
2		
3		
4		

Change order of items: sort by decreasing ratio v_i/w_i

```
S ← ∅;capacity ← 0;
for i = 1,…,n:
  if wi + capacity ≤ K
    S ← S ∪ {i}
    capacity ← wi + capacity
  end if
end for
return S
```

How bad it is?

Weight capacity K=25

item i	weight w_i	value v_i
1	1	1
2	25	24.9999

Change order of items: sort by decreasing ratio v_i/w_i

```
S ← ∅;capacity ← 0;
for i = 1,…,n:
    if wi + capacity ≤ K
        S ← S ∪ {i}
        capacity ← wi + capacity
    end if
end for
return S
```

Fractional Knapsack

Change order of items: sort by decreasing ratio v_i/w_i

Weight capacity K=25

item i	weight w_i	value v_i
1	9	9
2	10	9
3	7	6
4	22	12

```
S ← ∅;capacity ← 0;
for i = 1,…,n:
  if wi + capacity ≤ K
    S ← S ∪ {i}
    capacity ← wi + capacity
  end if
end for
return S
```

```
else
   Take (K-capacity)/wi
   of item i
```

Can choose a fraction of an item

A Special Case of Knapsack

Change order of items: sort by decreasing v_i

Weight capacity K=3

item i	weight w_i	value v_i
1	1	12
2	1	9
3	1	9
4	1	6

```
S ← ∅;capacity ← 0;
for i = 1,…,n:
    if wi + capacity ≤ K
        S ← S ∪ {i}
        capacity ← wi + capacity
    end if
end for
return S
```

$w_i = 1 \ \forall i = 1,\ldots,n$

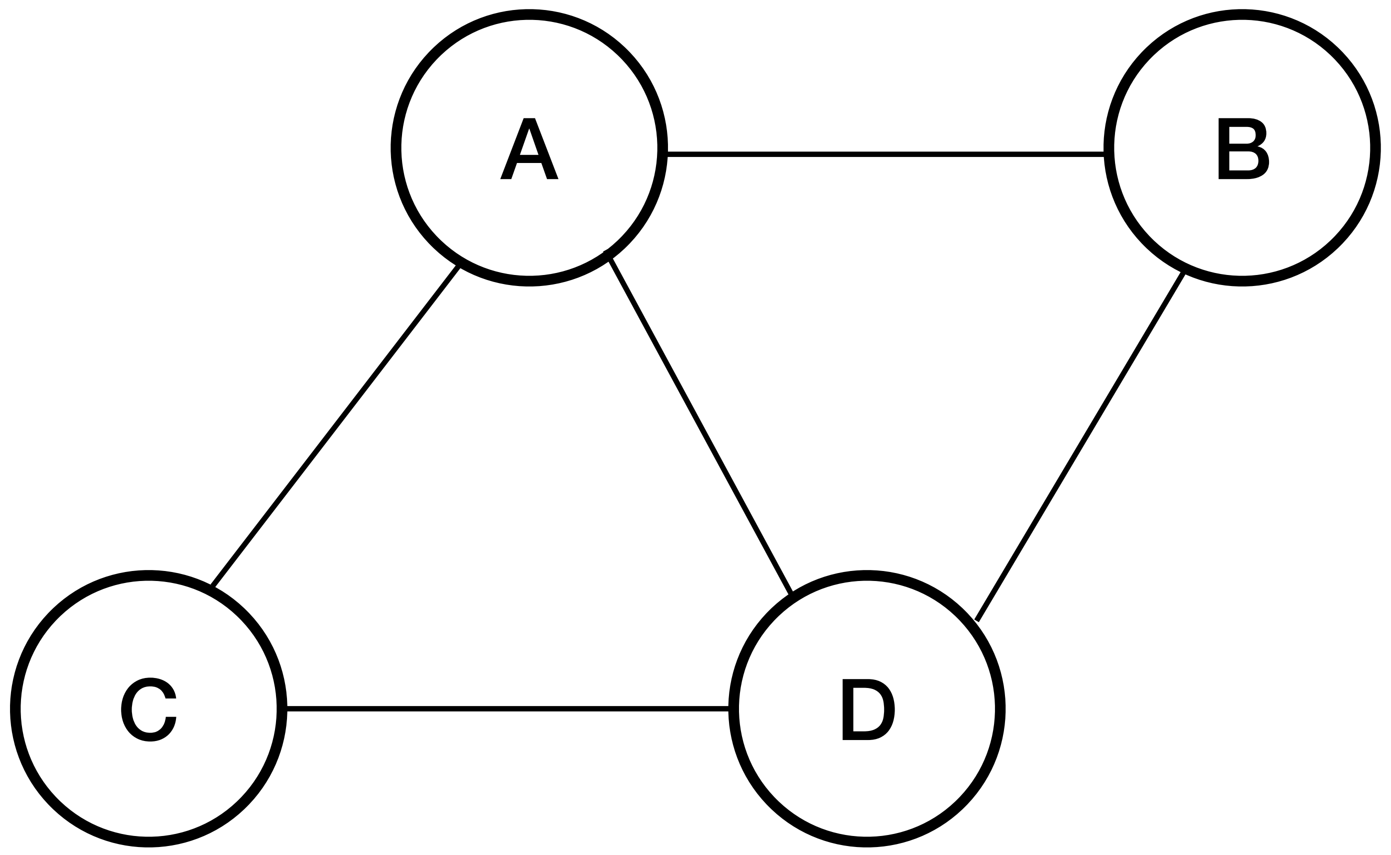
This greedy algorithm is optimal for this special case

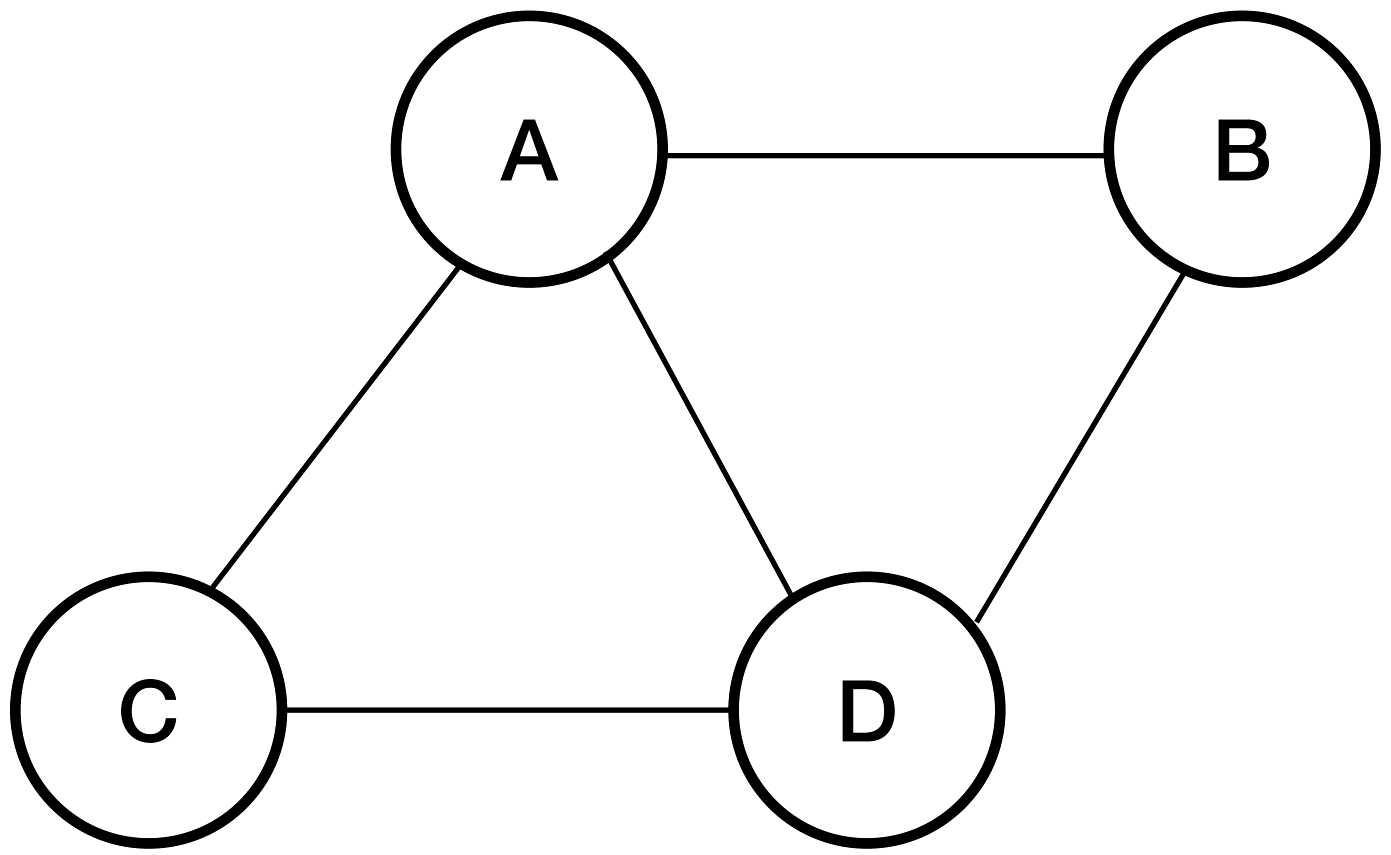
Vertex Cover

- Given a graph G=(V,E), find a subset S of vertices such that every edge in E has one endpoint in S
- and such that |S| is minimal

This problem is NP-hard, it is unlikely to get a polynomial time algorithm

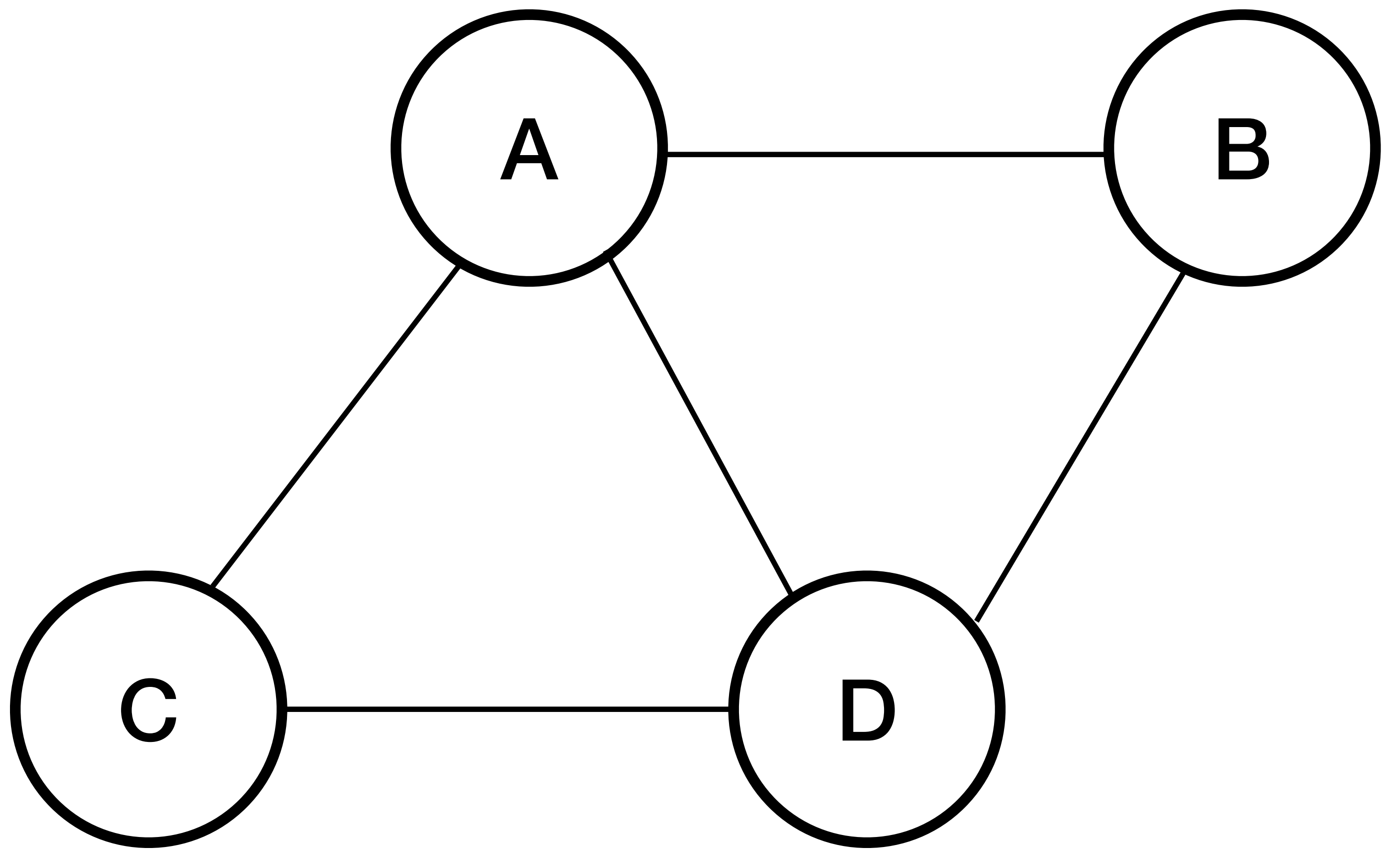
- Application: camera placement to cover all edges

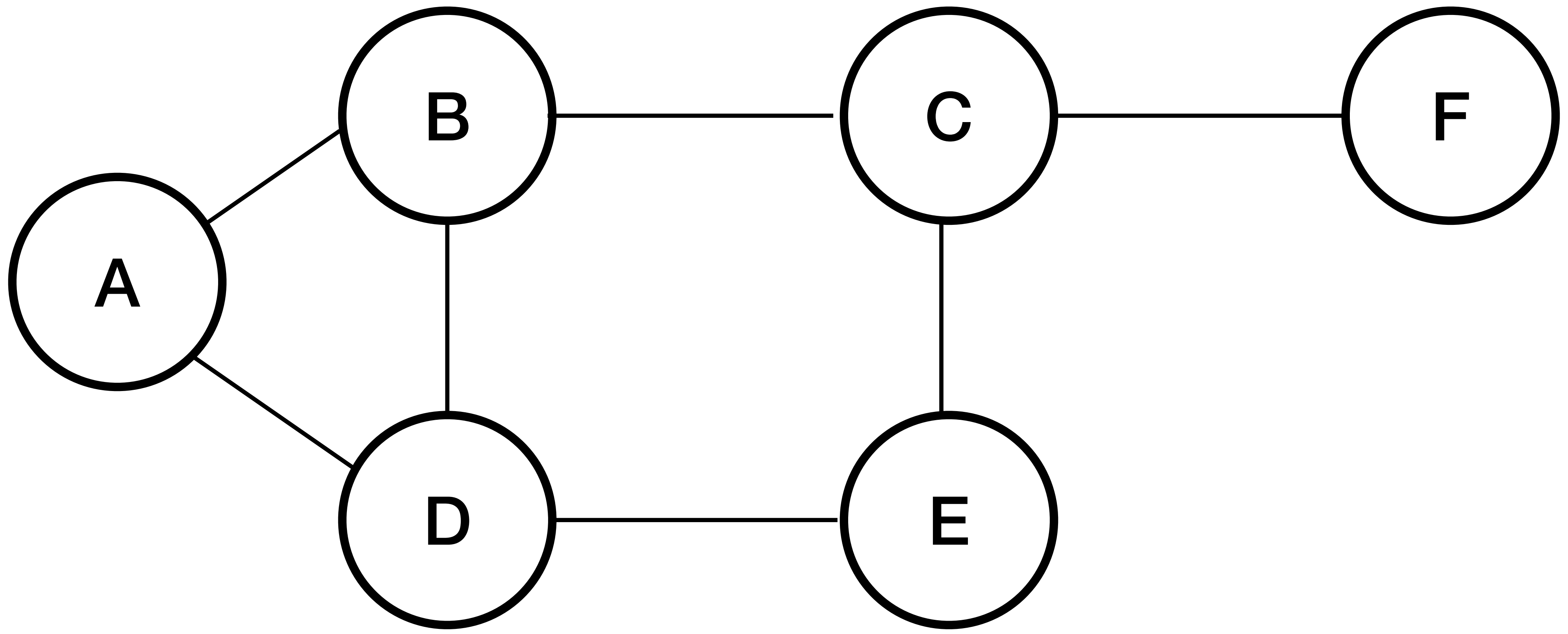




Vertex Cover

Input: $G = (V, E)$
$C \leftarrow \varnothing$
$E' \leftarrow G.E$
while $E' \neq \varnothing$:
    let (u, v) be an arbitrary edge of E'
    $C = C \cup \{u, v\}$
    remove from E' every edge incident on either u or v
end while
return C





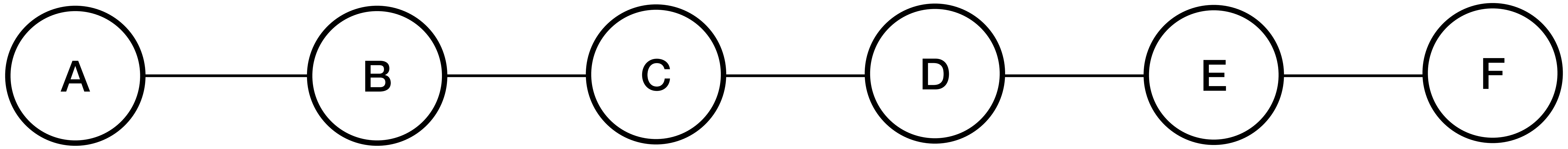
How bad is this algorithm?

Input: $G = (V, E)$
$C \leftarrow \varnothing$
$E' \leftarrow G.E$
while $E' \neq \varnothing$:
    let (u, v) be an arbitrary edge of E'
    $C = C \cup \{u, v\}$
    remove from E' every edge incident on either u or v
end while
return C

Each edge has one of its endpoint in C

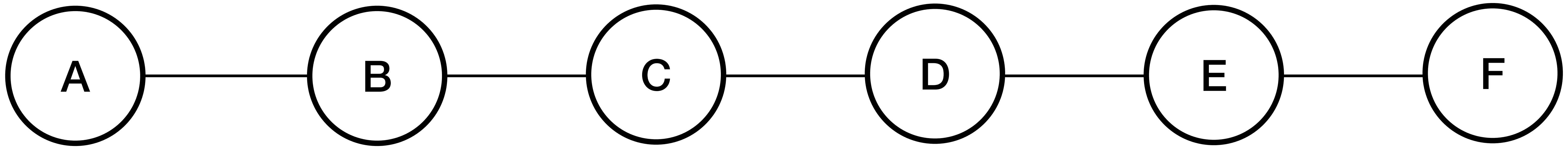
The algorithm may choose both

C is at most 2 times as large as the optimal vertex cover



How about special cases of graph?

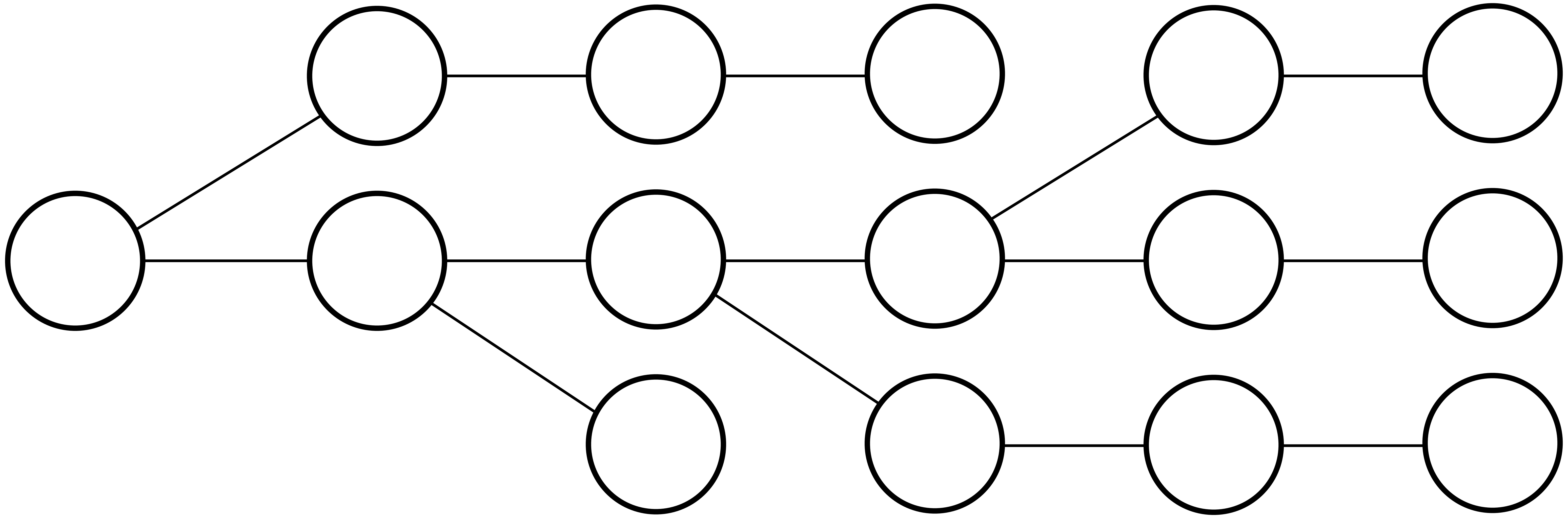
- Path



Input: $G = (V, E)$
$C \leftarrow \emptyset$
$E' \leftarrow G.E$
while $E' \neq \emptyset$:
    let (u, v) be an arbitrary edge of E' such that u is a vertex of degree 1
    $C = C \cup \{v\}$
    remove from E' every edge incident on vertex v
end while
return C

Other special cases of graph?

Tree

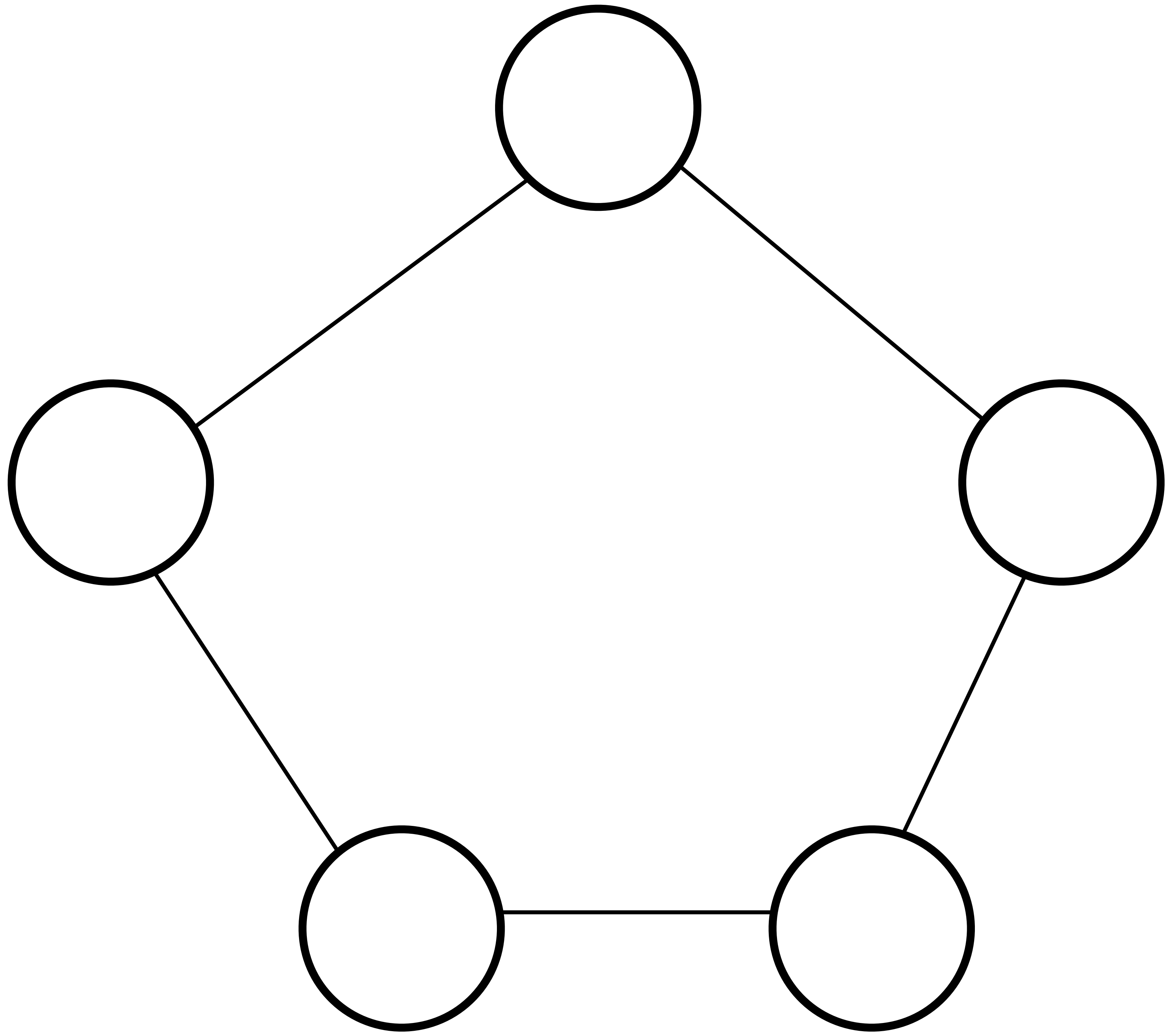


```
Input: G = (V, E)
C ← ∅
E′ ← G.E
while E′ ≠ ∅:
    let (u, v) be an arbitrary edge of E′ such that u is a vertex of degree 1
    C = C ∪ {v}
    remove from E′ every edge incident on vertex v
end while
return C
```

Cycle

Cycle of size n: P_n

- all vertices have degree 2
- has n edges
- is connected

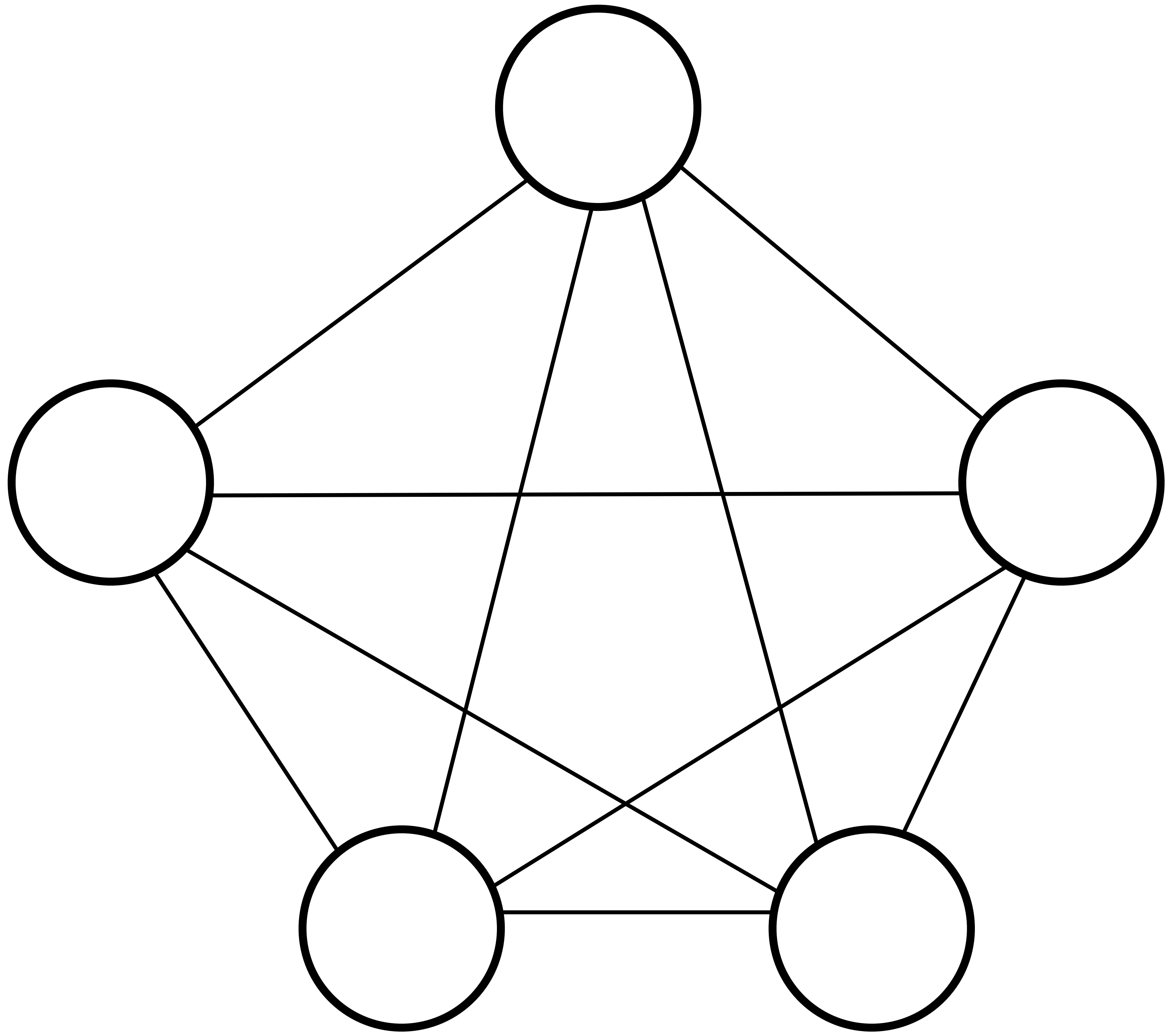


Optimal solution of the Vertex Cover has $\lceil n/2 \rceil$ vertices

Complete graph

Complete graph of size n: K_n

- has edge between every pair of vertices
- has $\dfrac{n(n-1)}{2}$ edges



Optimal solution of the Vertex Cover has $n - 1$ vertices

Vertex Cover

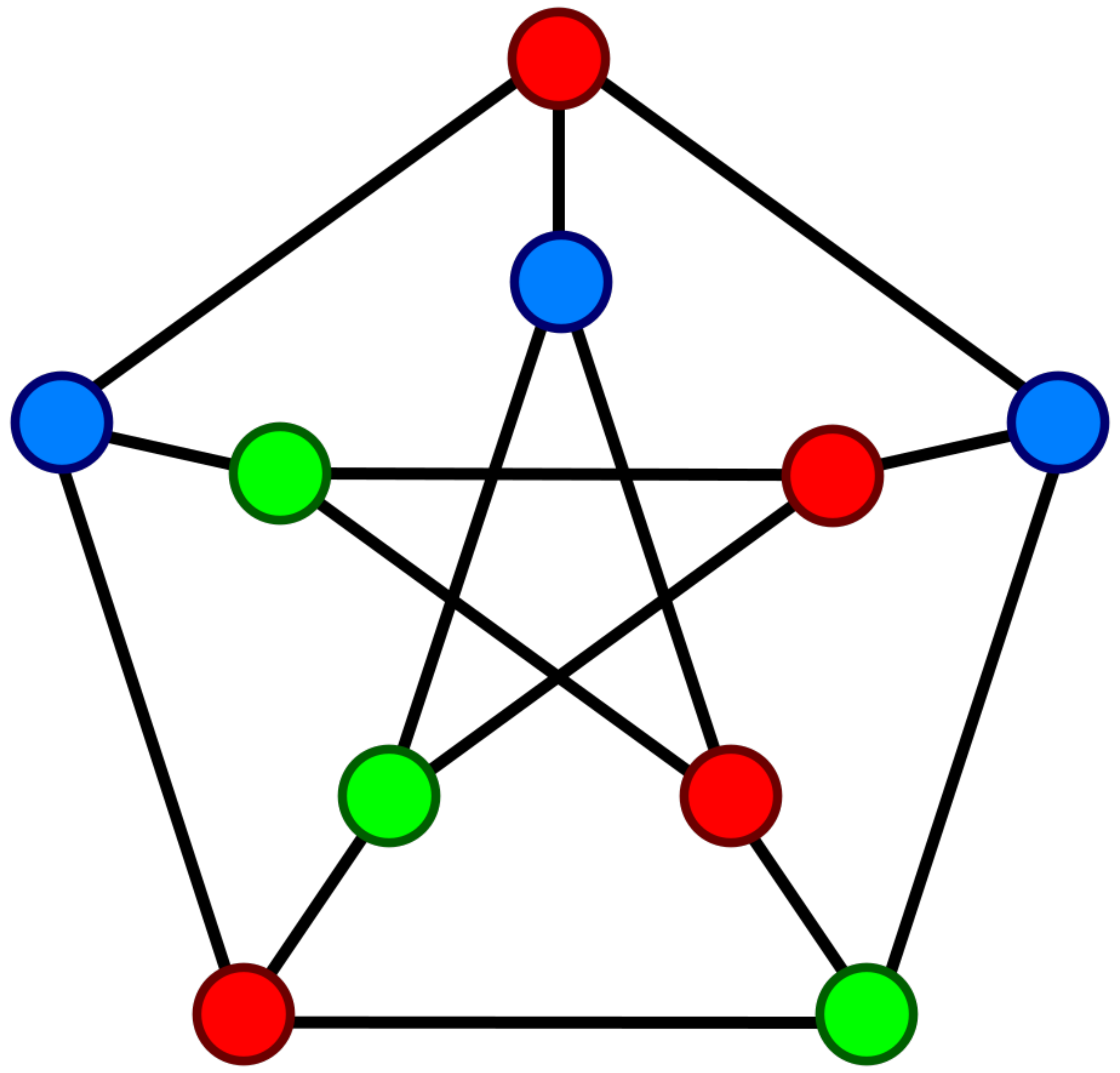
Summary

- NP-hard problem for general case
- 2-approximation greedy algorithm
- optimal greedy algorithm for special class of graphs:
    - Tree (Path)
    - Cycle
    - Complete graph

Graph Coloring Problem

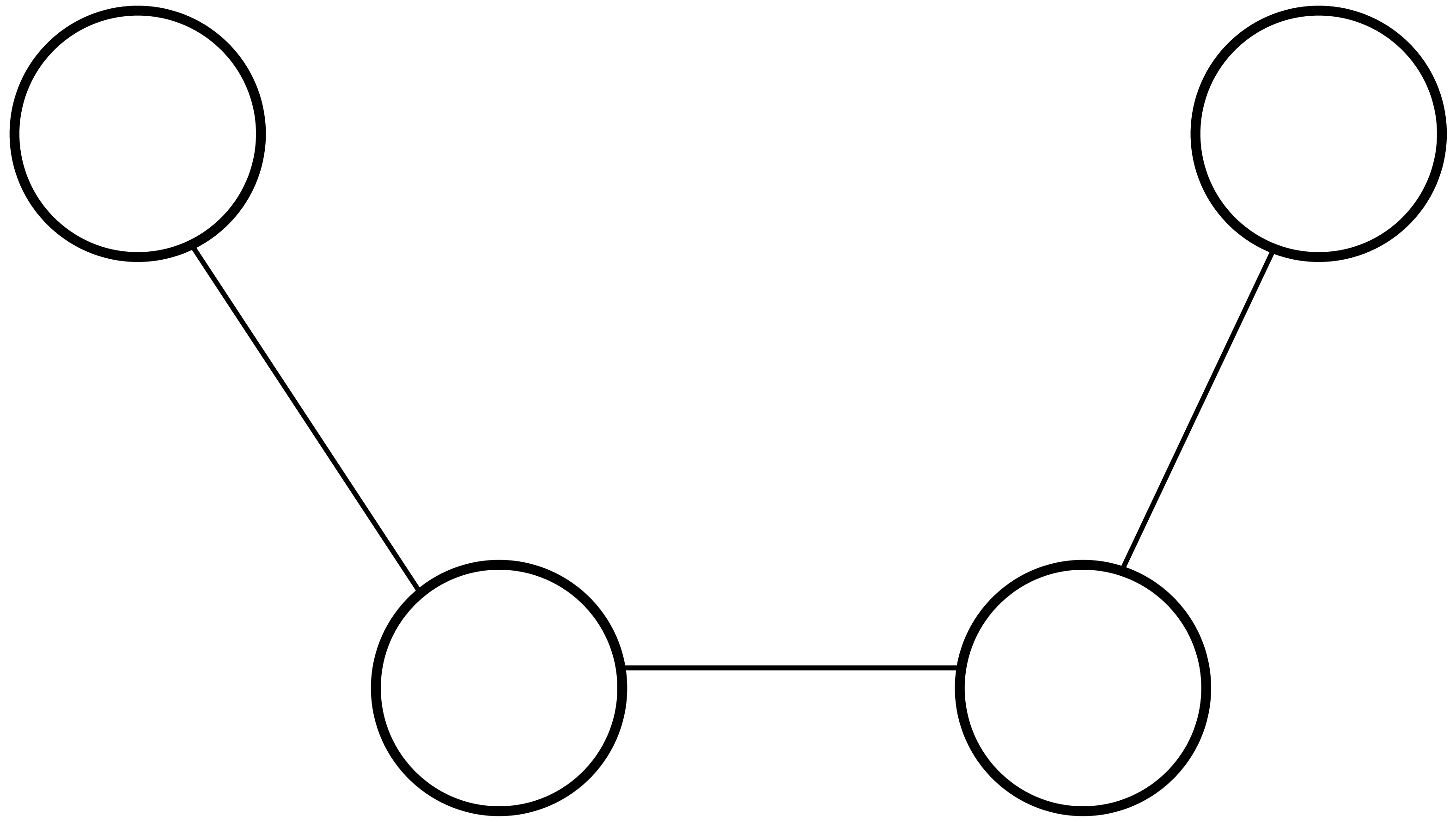
- Given a graph G(V,E)
- Color (Label) the vertices such that:
    - two adjacent vertices does not have the same color
    - use the minimum number of colors

NP-hard problem



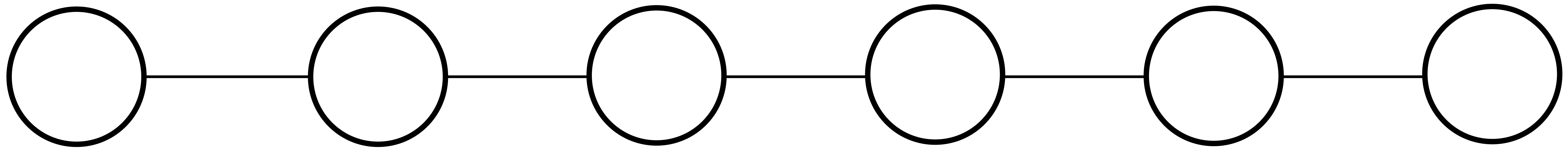
Greedy Algorithm for Graph Coloring

```
Input: G = (V,E)
color[v]←0 for v in V
for each vertex v in V:
  | color[v]←next_color_available(v,color,E)
end for
return color
```



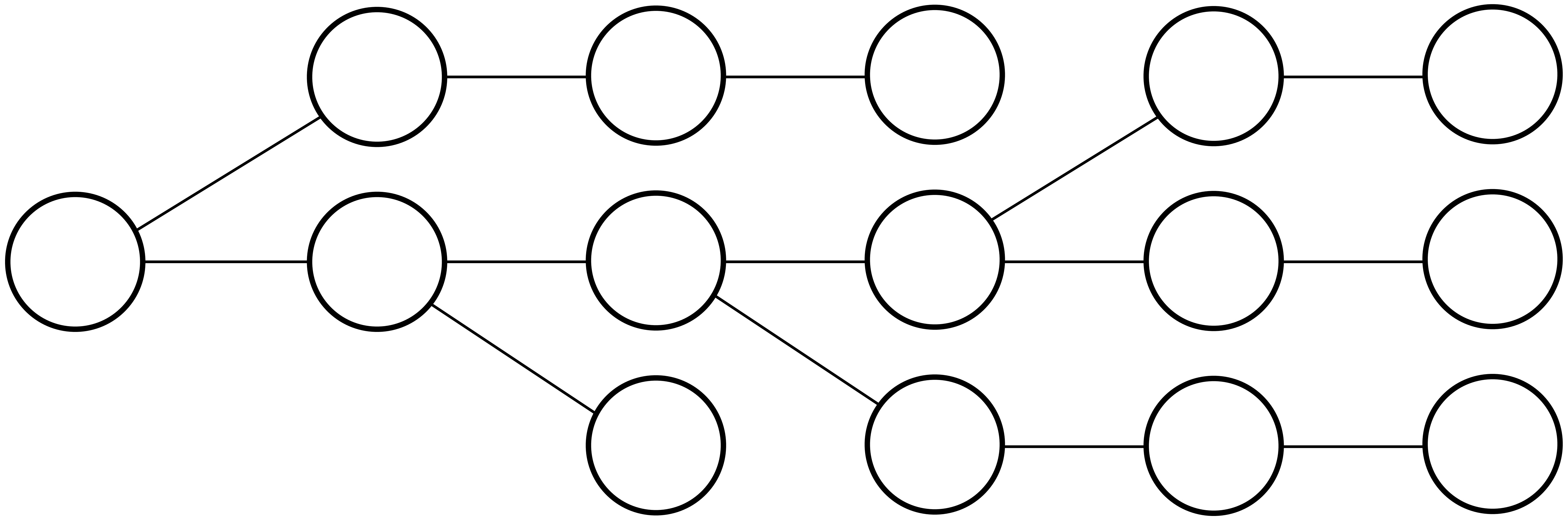
```
next_color_available(v,color,E):
S ← ∅, i ← 1
for each edge (u,v) in E:
  | S←S∪{color[u]}
end for
while not found:
  | if i ∉ S
  |   | return i
  | end if
  | i ← i + 1
end while
return i
```

Path



- 2 colors are enough for this graph
- Worst case of greedy algorithm uses 3 colors

Tree

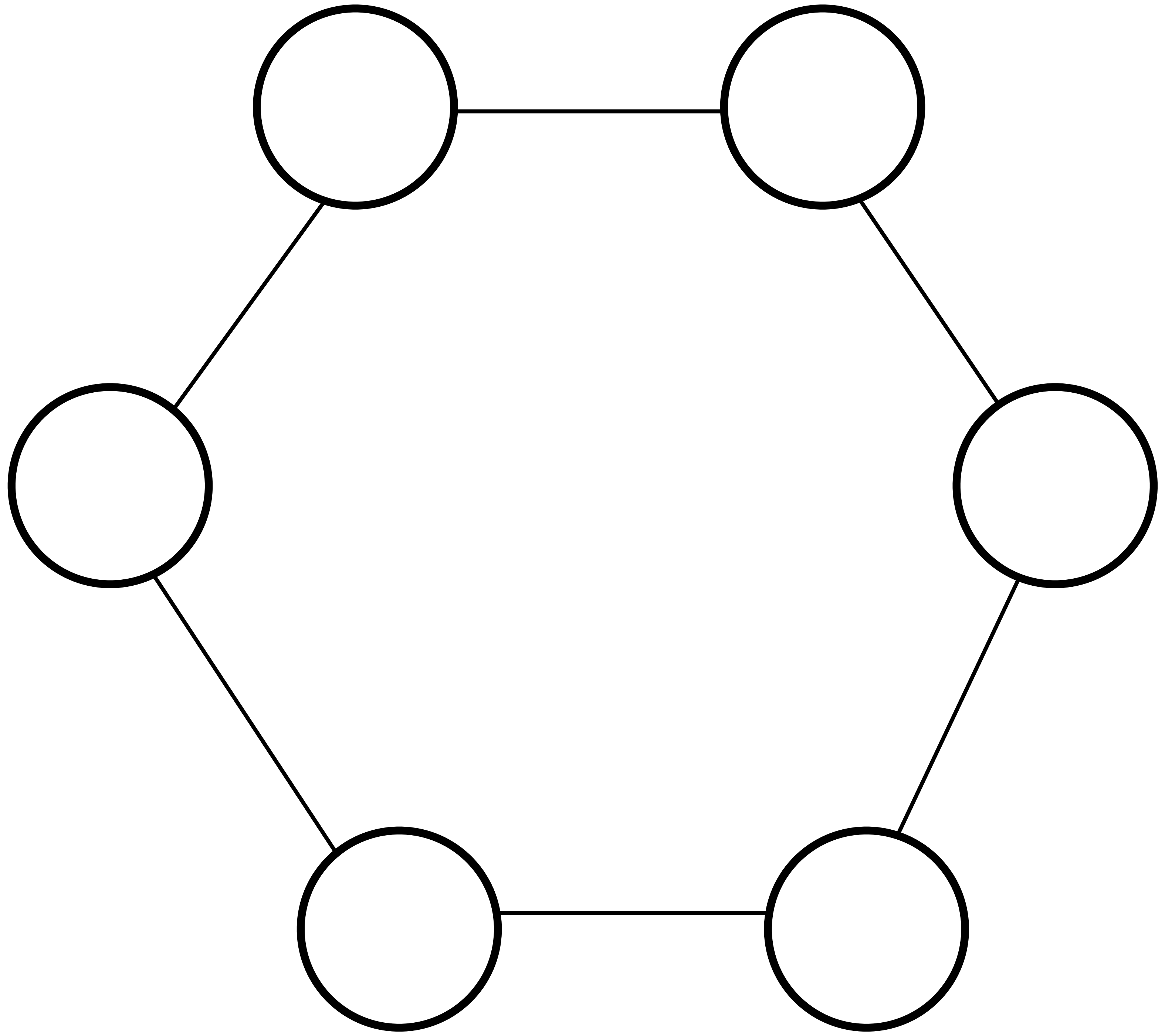


- 2 colors are enough for this graph
- Worst case of greedy algorithm in this graph uses 4 colors

Cycle

Cycle of size n: P_n

- all vertices have degree 2
- has n edges
- is connected

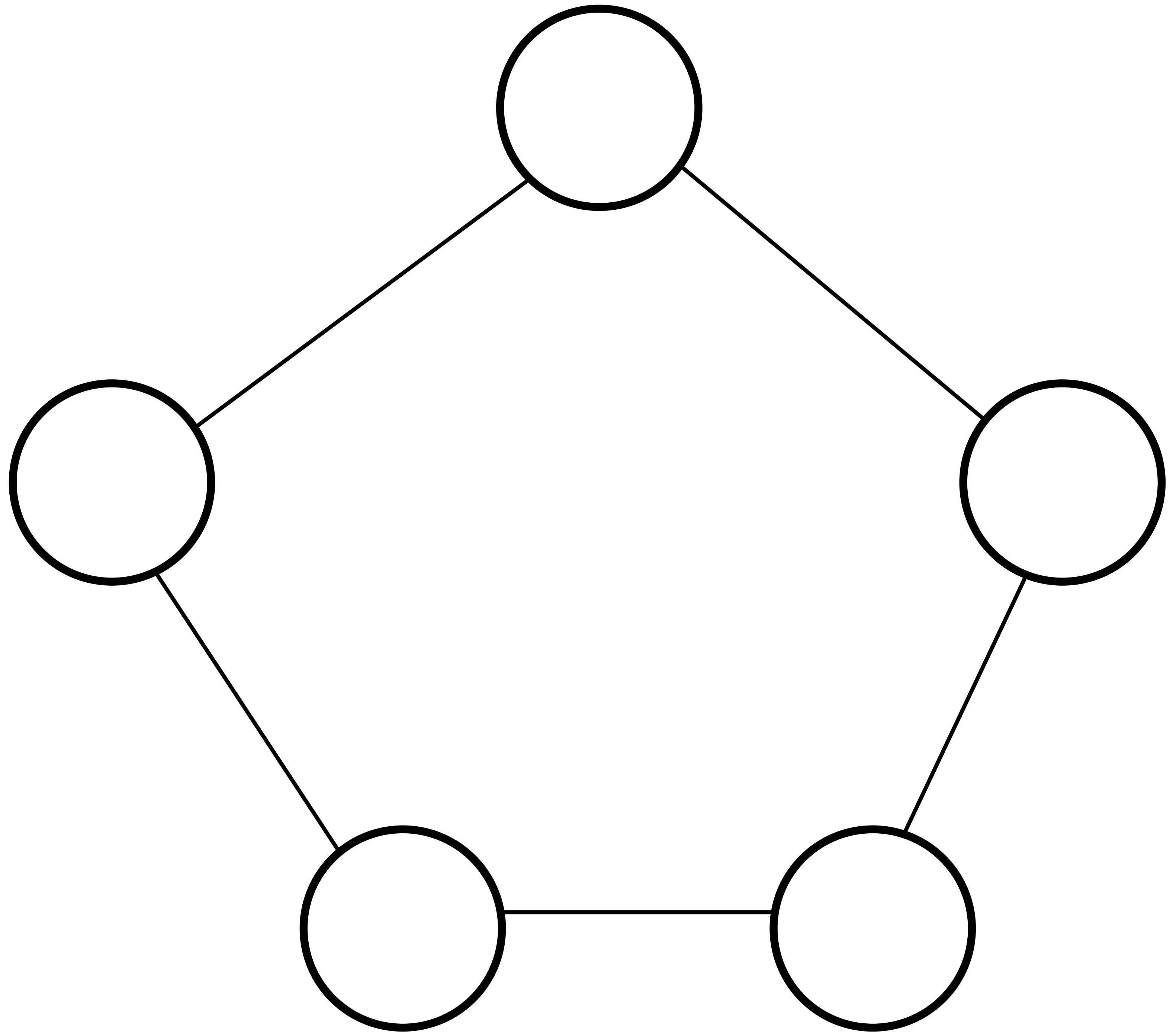


- 2 colors are enough for cycle of even length
- Worst case of greedy algorithm in this graph uses 3 colors

Cycle

Cycle of size n: P_n

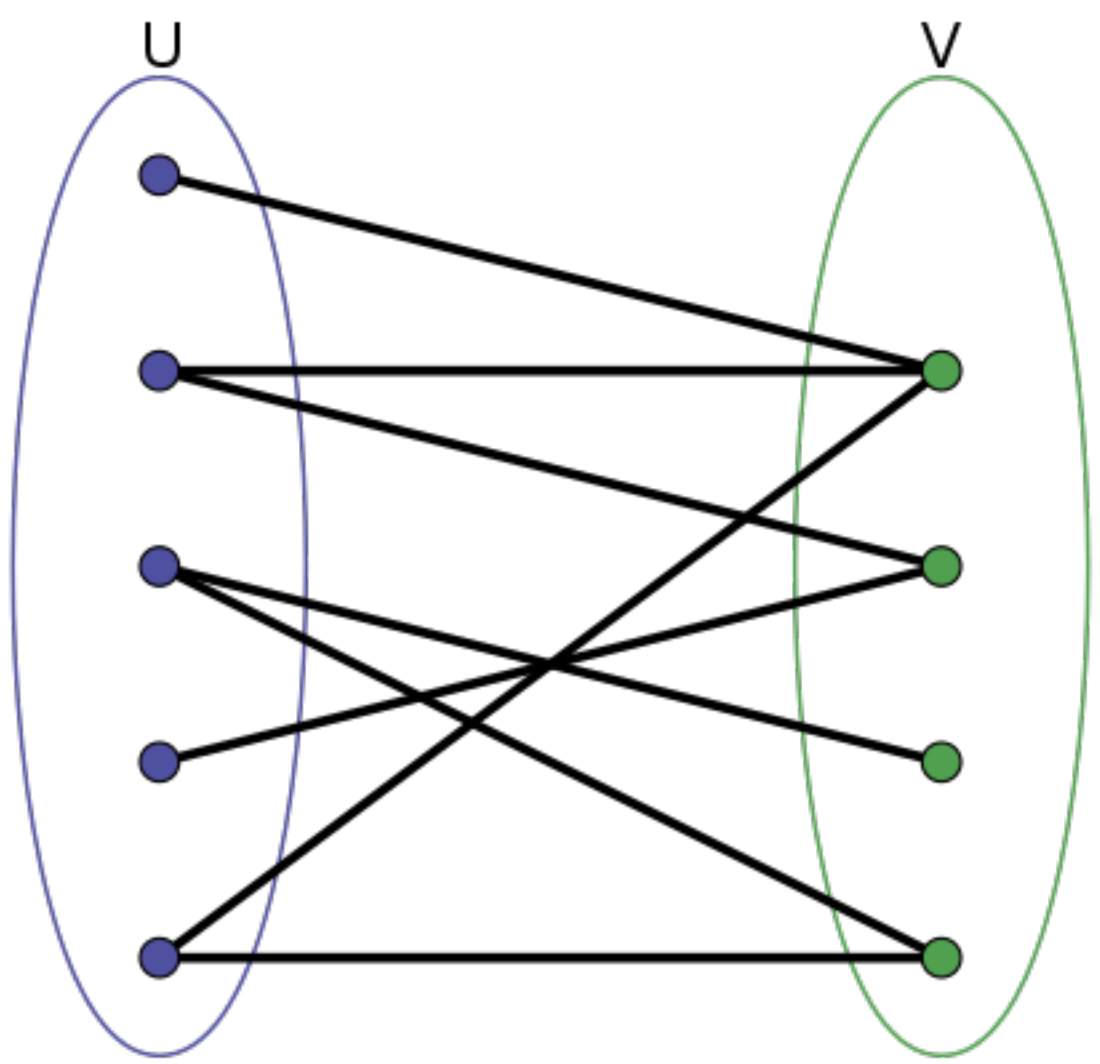
- all vertices have degree 2
- has n edges
- is connected



- 3 colors are enough for cycle of odd length

Bipartite Graph

- Two sets of vertices U and V
- No edges between vertices in U (resp. V)



Properties

- Does not contain odd cycle
- 2 colors are enough for bipartite graphs

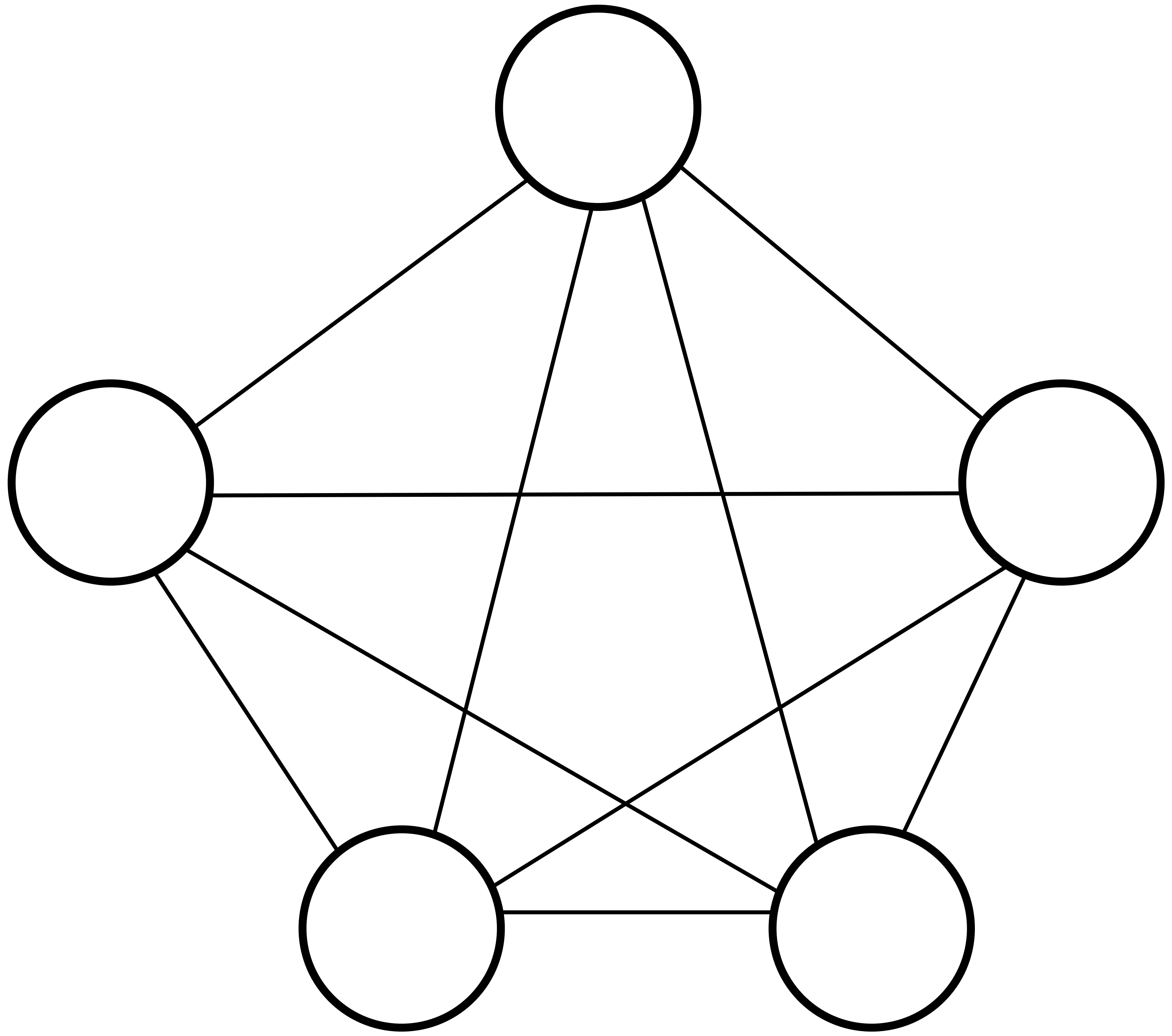
Example of bipartite graphs: Path, Even Cycle, Tree

Complete graph

Complete graph of size n: K_n

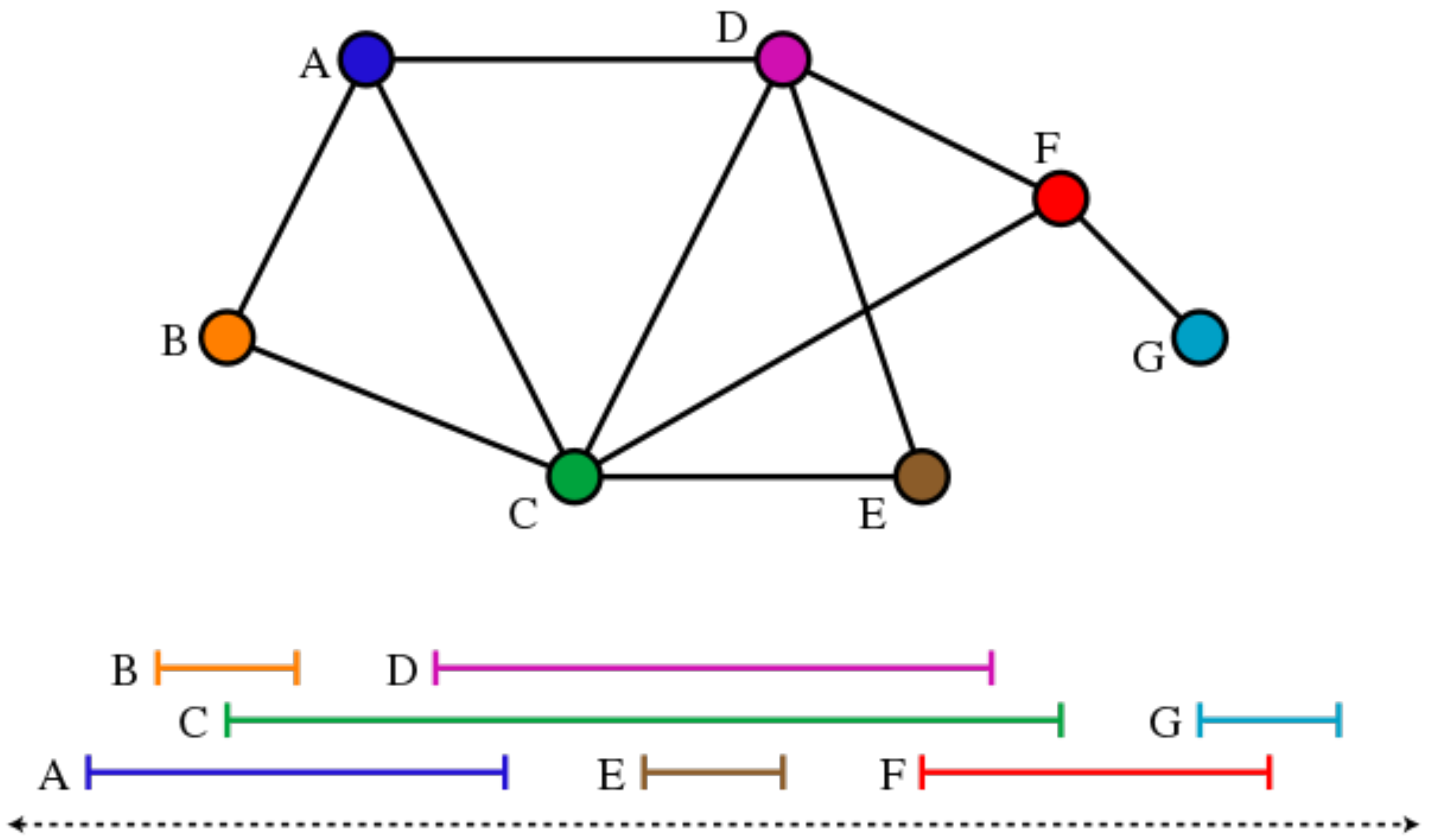
- has edge between every pair of vertices
- has $\dfrac{n(n-1)}{2}$ edges

- Complete graph needs n colors



Interval Graph

- One interval ⇔ One vertex
- If two intervals overlap, add an edge between the corresponding vertices
- Remarks: cycle with length ≥4 is not an interval graph



Summary

- Greedy algorithms are fast and simple
- Do not always return the optimal solution
- But can perform well in some case

Exercises

Scheduling problem (1)

Let $E = \{\sigma_1, \ldots, \sigma_n\}$ be a set of intervals such that for each interval σ_i is defined as follows $\sigma_i = [a_i, b_i)$ with $a_i \leq b_i$. A schedule is feasible if for any two intervals σ_a, σ_b we have $\sigma_a \cap \sigma_b = \emptyset$. We consider the profit of a solution as the total length of the interval. If S is feasible, then the profit is $p(S) = \sum_{i \in S} (b_i - a_i)$.The goal is to find a feasible solution S such that $p(S)$ is maximized.

For example, $E = \{[0,3),\ [1,4),\ [3,7),\ [8,10)\}$
$S = \{[0,3),\ [3,7)\}$ is a feasible schedule with $p(S) = (3-0) + (7-3) = 7$
$S = \{[0,3),\ [1,4)\}$ is not a feasible schedule.

1. Design a greedy algorithm to select the intervals according to the input
2. Is this algorithm optimal? If it is not, find a counter example (bonus: find the worst case of the algorithm)
3. Try different order of the inputs and run the greedy algorithm again
4. Design a dynamic programming algorithm to solve this problem

Scheduling problem (1)

Data set

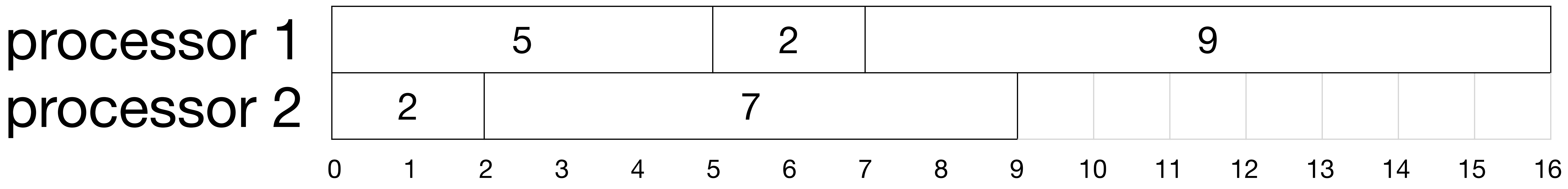
- $E = \{[0,3), [1,4), [3,7), [8,10)\}$
- $E = \{[8,10), [1,3), [4,5), [6,7), [9,15), [1,10), [5,8)\}$
- $E = \{[10,19), [13,23), [0,9), [4,19), [6,9), [7,12), [1,14), [15,35)\}$

Scheduling problem (2)

We consider the following scheduling problem. Given a set of n jobs. Each job i has processing time p_i, all available at the beginning.

The goal is to assign jobs on two processors such that the maximum load is minimized.

Example of 5 jobs with processing times $\{5,2,7,2,9\}$



maximum load=16

Scheduling problem (2)

1. Design a greedy algorithm to assign jobs to processors
2. Is this algorithm optimal? If it is not, find a counter example (bonus: find the worst case of the algorithm)
3. Try different order of the inputs and run the greedy algorithm again
4. Design a dynamic programming algorithm to solve this problem
5. Suppose now we have 3 processors. Answer questions 1-4 again

Scheduling problem (2)

Data set

- $E = \{1,1,2,3,7,10,12\}$
- $E = \{5,8,3,1,8,20\}$
- $E = \{2,3,2,3,2\}$