파이썬으로 배우는 알고리즘 기초

Chap 4. 탐욕 알고리즘

# 4.1 탐욕법과 최소비용신장트리

# 탐욕 알고리즘


- **탐욕 알고리즘**: The **Greedy** Approach
  - 최종 해답을 찾기 위해서 각 단계마다 하나의 답을 고름
  - 각 단계에서 답을 고를 때 가장 좋아 보이는 답을 선택
  - **최적화 문제**에서,
    - 선택할 당시에는 최적의 답을 고르지만 (locally optimal),
    - 최종 해답이 반드시 최적임을 보장하지 않음(globally optimal not guaranteed)

# 탐욕 알고리즘


- 동전의 거스름돈 문제
  - 거스름돈이 870원인 경우,
  - 동전의 개수가 최소가 되도록 동전을 선택하는 방법은?

# 탐욕 알고리즘


- **거스름돈 문제: 탐욕법으로 풀기**

```
while (동전이 남아있고 문제가 미해결):
  가장 가치가 높은 동전을 선택한다.
  if (동전을 더하여 거스름돈의 총액이 거슬러주어야 할 액수를 초과):
    동전을 도로 집어넣는다.
  else:
    거스름돈에 동전을 포함시킨다.
  if (거스름돈의 총액이 거슬러주어야 할 액수와 같다)
    문제 해결.
```

# 탐욕 알고리즘


- **거스름돈 알고리즘은 항상 최적해를 찾는가?**
  - **동전의 구성이** [500, 100, 50, 10, 5, 1]**일 경우**: Yes!
  - **동전의 구성이** [500, 100, 80, 50, 10, 5, 1]**일 경우**: No!

  - **예) 거스름돈** 360**원의 최적해는?**
    - **탐욕 알고리즘의 해**: [100, 100, 100, 50, 10]
    - **최적해**: [100, 100, 80, 80]

# 탐욕 알고리즘


- 탐욕 알고리즘의 이모저모
  - 탐욕 알고리즘의 설계 전략: 공집합에서 시작
    - 선택 과정: 집합에 추가할 다음 최적의 원소를 고른다.
    - 적절성 검사: 새로운 집합이 해답으로 적절한지 검사한다.
    - 해답 점검: 새로운 집합이 문제의 해답인지 판단한다.
  - 탐욕 알고리즘의 장단점
    - 장점: 상대적으로 설계하기가 매우 쉽다.
    - 단점: 최적화 문제에서 반드시 정확성을 증명해야 한다.

# 4.1 최소비용 신장트리



## 최소비용 신장트리 문제

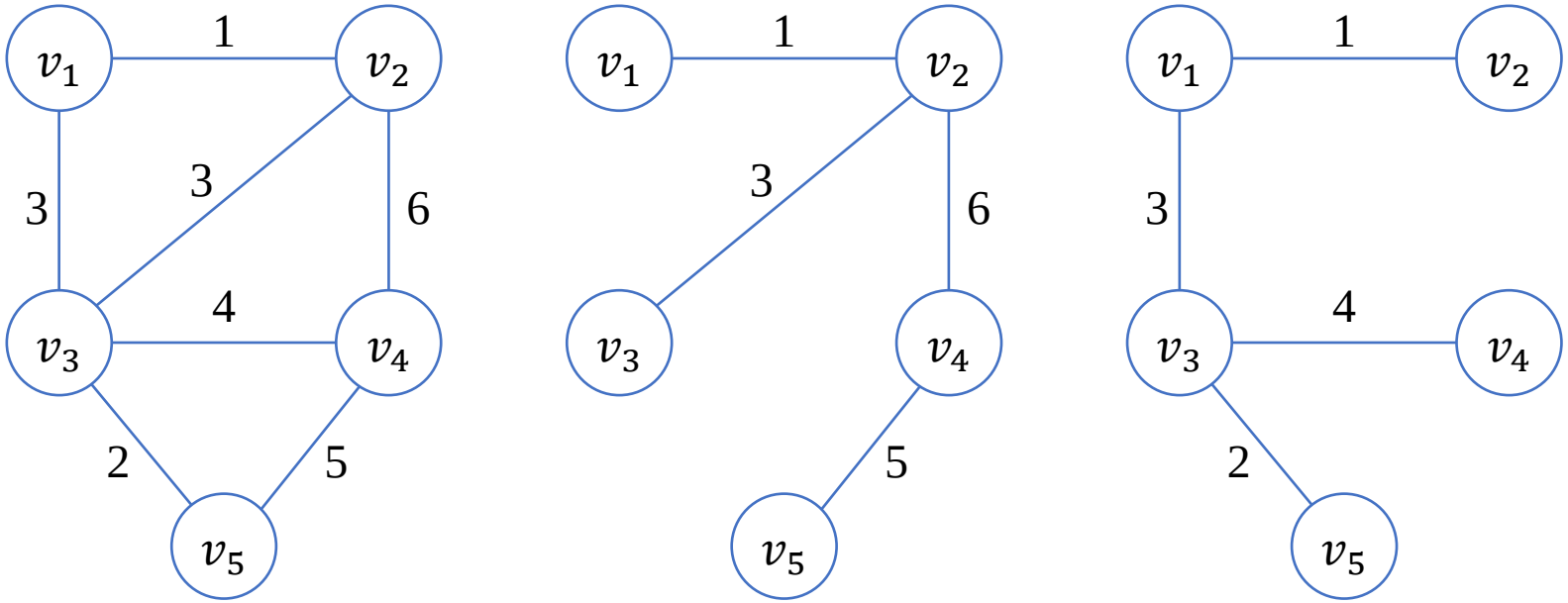
- 문제: 주어진 그래프에서 최소비용 신장트리를 구하시오.

- 엄밀한 문제 정의
  - 주어진 그래프: $G = (V, E)$: *connected*, *weighted*, and *undirected*
    - 그래프 $G$는 모든 정점이 연결된 가중치가 있는 무방향 그래프
  - 신장트리(spanning tree): $G$의 부분 그래프 $T = (V, F)$, $F \subseteq E$
    - 그래프 $G$의 모든 정점을 연결하는 트리: 간선의 개수는 $n - 1$
  - 최소비용 신장트리(MST: Minimum cost Spanning Tree)
    - 모든 신장트리 $T$ 중에서 가중치의 합이 최소가 되는 신장트리

# 4.1 최소비용 신장트리



- 최소비용 신장트리 문제의 이해
  - 단순무식하게 풀기(Brute-Force)
    - 모든 신장트리를 찾아서 가중치의 합이 가장 작은 것을 선택
  - 신장트리를 찾는 법
    - 간선의 개수가 $n-1$인 연결된 트리(*acyclic*)가 될 때까지 간선을 하나씩 제거



# 4.1 최소비용 신장트리



- 최소비용 신장트리: 욕심쟁이 방법(Greedy Approach)
  - 1단계(초기화): 해답의 집합을 공집합으로 둔다.
    - 간선 집합 $E$의 부분 집합 $F$를 공집합으로 둔다.
  - 2단계(선택): 최적의 원소 하나를 해답의 집합에 포함시킨다.
    - $E$에서 최적의 간선 하나를 추출해서 $F$에 포함시킨다.
    - 최적을 선택하는 방법? 프림 .vs. 크루스칼
  - 3단계(검사): 해답의 집합이 최종이면 종료, 아니면 2단계를 반복한다.
    - 간선의 부분 집합 $F$의 원소 개수가 $n-1$이면 최종 해답

# 4.1 최소비용 신장트리



- 최소비용 신장트리: 프림 알고리즘(Prim's Algorithm)
  - 1단계(초기화): 해답의 집합을 공집합으로 둔다.
    - $F = \phi$, $Y = \{v_1\}$: $Y$는 정점의 집합 $V$의 부분 집합
  - 2단계(선택): 최적의 원소 하나를 해답의 집합에 포함시킨다.
    - $V - Y$ 집합에서 $Y$ 집합에서 가장 가까운 정점 $vnear$를 선택
    - $Y$ 집합에 $vnear$를 추가, $F$ 집합에 $(nearest(vnear), vnear)$를 추가
  - 3단계(검사): 해답의 집합이 최종이면 종료, 아니면 2단계를 반복한다.
    - $Y = V$: $Y$ 집합이 $V$ 집합의 모든 원소를 포함하면 종료

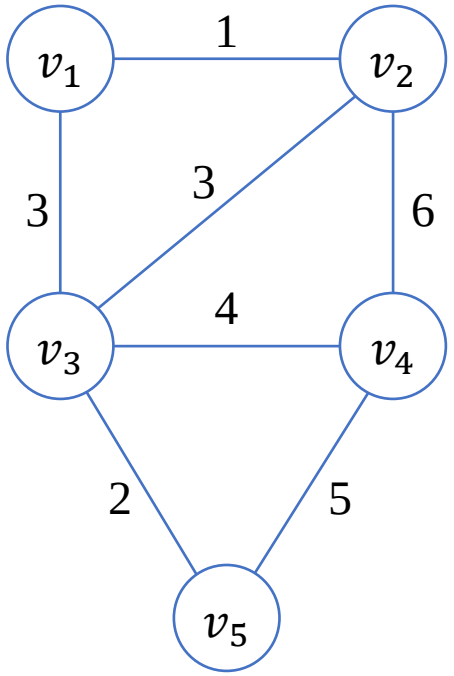
# 4.1 최소비용 신장트리

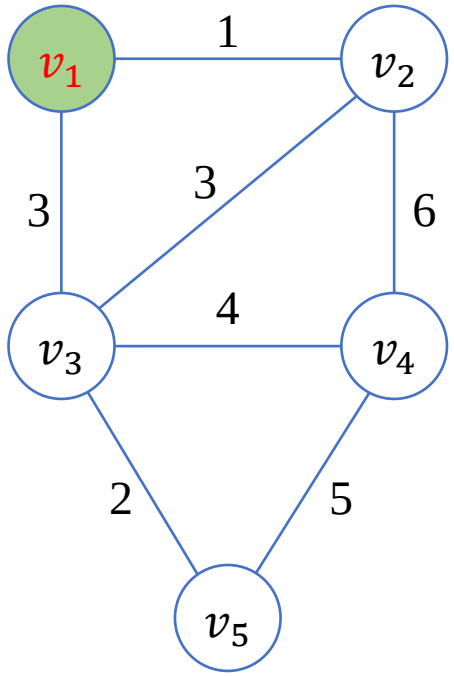


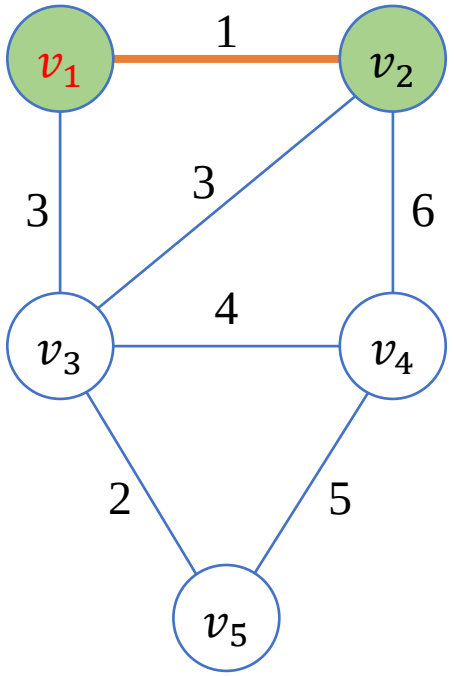
- Determine a minimum spanning tree

1. Vertex $v_1$ is selected first

2. Vertex $v_2$ is selected because it is nearest to $\{v_1\}$



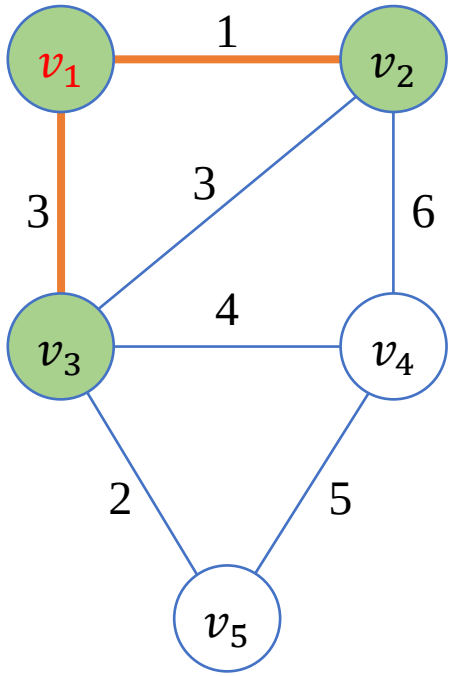



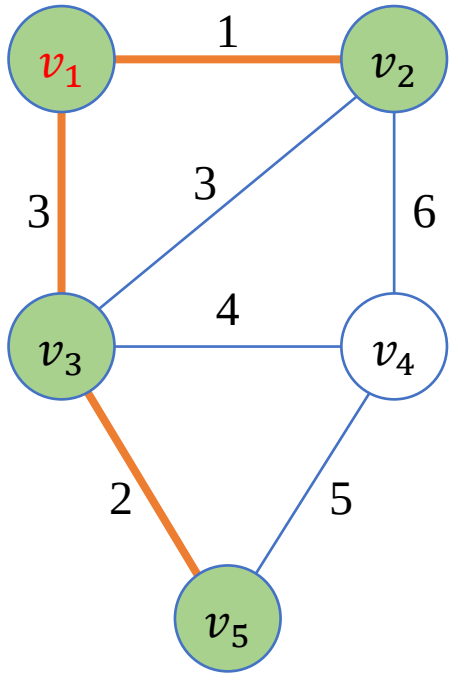
# 4.1 최소비용 신장트리



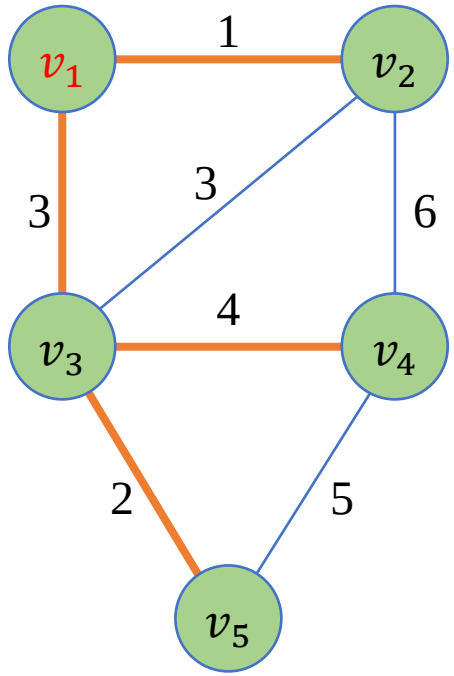
3. Vertex $v_3$ is selected because it is nearest to $\{v_1, v_2\}$




4. Vertex $v_5$ is selected because it is nearest to $\{v_1, v_2, v_3\}$




5. Vertex $v_4$ is selected because it is nearest to $\{v_1, v_2, v_3, v_5\}$



# 4.1 최소비용 신장트리



- W[i][j]: 인접행렬 (간선의 가중치)
- nearest[i]: $Y$ 집합에서 $v_i$에 가장 가까운 정점의 인덱스
- distance[i]: $v_i$와 $nearest[i]$의 정점을 연결하는 간선의 가중치

| W | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 0 | 1 | 3 | ∞ | ∞ |
| 2 | 1 | 0 | 3 | 6 | ∞ |
| 3 | 3 | 3 | 0 | 4 | 2 |
| 4 | ∞ | 6 | 4 | 0 | 5 |
| 5 | ∞ | ∞ | 2 | 5 | 0 |

| i | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| nearest[i] | -1 | 1 | 1 | 1 | 1 |
| distance[i] | -1 | 1 | 3 | ∞ | ∞ |
| nearest[i] | -1 | 1 | 1 | 2 | 1 |
| distance[i] | -1 | -1 | 3 | 6 | ∞ |
| nearest[i] | -1 | | | | |
| distance[i] | -1 | | | | |
| nearest[i] | -1 | | | | |
| distance[i] | -1 | | | | |
| nearest[i] | -1 | | | | |
| distance[i] | -1 | | | | |

## Algorithm 4.1: Prim's Algorithm

```
def prim (W):
    n = len(W) - 1
    F = []
    nearest = [-1] * (n + 1)
    distance = [-1] * (n + 1)
    for i in range(2, n + 1):
        nearest[i] = 1
        distance[i] = W[1][i]
```

# 4.1 최소비용 신장트리



**Algorithm 4.1**: Prim's Algorithm

```
    for _ in range(n - 1):
        minValue = INF
        for i in range(2, n + 1):
            if (0 <= distance[i] and distance[i] < minValue):
                minValue = distance[i]
                vnear = i
        edge = (nearest[vnear], vnear, W[nearest[vnear]][vnear])
        F.append(edge) # add edge to F
        distance[vnear] = -1
        for i in range(2, n + 1):
            if (distance[i] > W[i][vnear]):
                distance[i] = W[i][vnear]
                nearest[i] = vnear
    return F
```

## Algorithm 4.1: Prim's Algorithm

```
def cost (F):
    total = 0
    for e in F:
        total += e[2]
    return total

def print_nd (F, nearest, distance):
    print('F = ', end = '')
    print(F)
    print('   nearest: ', end = '')
    print(nearest)
    print('  distance: ', end = '')
    print(distance)
```

# 4.1 최소비용 신장트리


```
INF = 999
W = [
    [-1, -1, -1, -1, -1],
    [-1, 0, 1, 3, INF, INF],
    [-1, 1, 0, 3, 6, INF],
    [-1, 3, 3, 0, 4, 2],
    [-1, INF, 6, 4, 0, 5],
    [-1, INF, INF, 2, 5, 0],
]

F = prim(W)
for i in range(len(F)):
    print(F[i])

print("Minimum Cost is ", cost(F, W))
```

# 주니온TV@Youtube

자세히 보면 유익한 코딩 채널

https://bit.ly/2JXXGqz


- 여러분의 **구독**과 **좋아요**는 강의제작에 큰 힘이 됩니다.
- 강의자료 및 소스코드: **구글 드라이브**에서 다운로드
  (다운로드 주소는 영상 하단 설명란 참고)

  https://bit.ly/3fN0q8t