9/8 지난시간 : 마지막 Instance를 제한하면 쉬워지는 경우.

Problem 의 solution에 제한을 가하면 어려워지는 경우.

Knapsack Problem
배낭.

어떤 집합 $Y = \{y_1, y_2, ---, y_n\}$

$C = \{c_1, c_2, ---, c_n\}$

y_i : 물건 i의 부피 (무게도 가능)

c_i : 물건 i를 배낭에 넣었을때 얻는 이득.

W : 가방의 부피 (운반가능한수치)

질문 : 이득이 최대가 되도록 가방에 물건을 선택하여 넣으시오.

조건 : 가방에 넣는 물건의 부피 $\leq W$

① 물건을 넣는데 있어서 아무 제한이 없는 경우. (floating knapsack ?)

⇒ 배낭에 여분이 있으면 물건의 일부도 넣을 수 있다.

⇒ easy to solve an optimal sol. by greedy algorithm.

② 물건을 모두 넣던가 포기하던가 선택하라.

⇒ very very difficult problem to get an optimal sol.

(0/1 Knapsack problem)

0/1 Knapsack

여러 물건 $Y = \{y_1, y_2, \cdots, y_n\}$

$C = \{c_1, c_2, \cdots, c_n\}$

y_i : 물건 i의 부피 (쪼개질 수 없음)

c_i : 물건 i를 배낭에 넣었을 때 얻는 이득.

W : 가방의 부피 (담을 수 있는 최대)

목표 : 이득이 최대가 되도록 가방에 물건을 선택하여 넣으시오.

조건 : ① 가방에 넣는 물건의 부피 $\leq W$

② 넣을 수 있는가.

이런 문제는 combinational optimization problem 이라고 한다.

Q1. solution space의 크기는? (모든 경우의 수는?)

Boolean Var x_1 x_2 ---- x_n

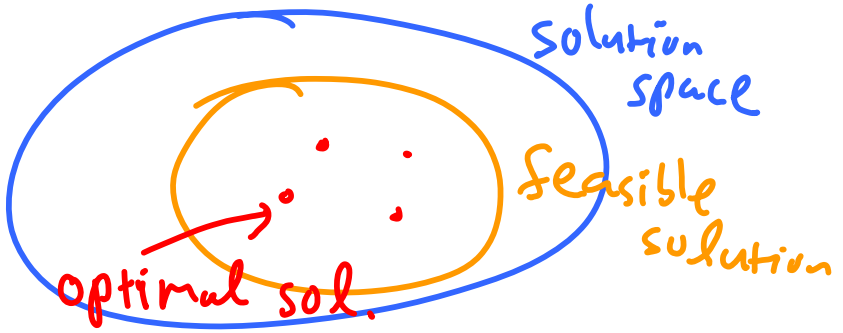
$x_i = 1$: 넣는다 / 0 : 포기하는 경우. → 2^n

Q2 : feasible solution은?

부피의 합 $\leq W$ 인 solution

Q3 : optimal solution?

feasible이고 이득이 가장 큰 sol.



Data Structure의 역할.

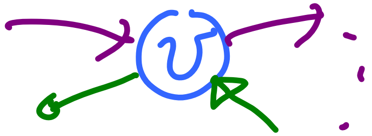
예: Euler Path or Cycle 구하기.

Path: 딱 2개만 odd.
Cycle: 모든 vertex degree = even

Euler cycle이 존재하면
아무 vertex에서 시작
Euler path "
홀수 degree vertex에서 시작.

관찰

짝수 degree vertex인 경우 v
v 진입하면 진출할 수 있는 edge가 있다.



홀수 degree vertex이면

- 진입 진출이 반복되면 1개의 unused edge가 존재

⇒ 시작은 홀수 degree vertex에서
종료도 " "

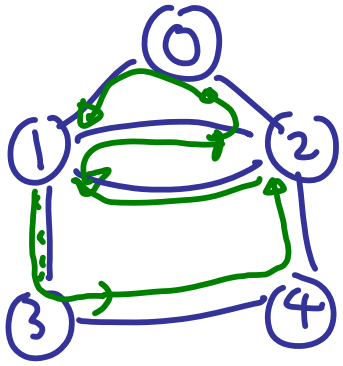
더이상 선택할 edge가 없는 때까지 계속 진행.

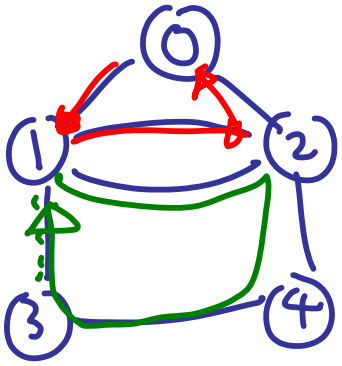
찾은 cycle 또는 path는 Euler cycle or path가 아닐수도 있다.

관찰

- Euler cycle인 경우 v에서 시작하여 임의로 edge를 선택해서 계속 진행하면 v로 돌아온다.
- Euler path인 경우 홀수 degree에서 시작해서 임의로 edge를 선택해서 진행하면 또다른 홀수 degree에서 끝난다.

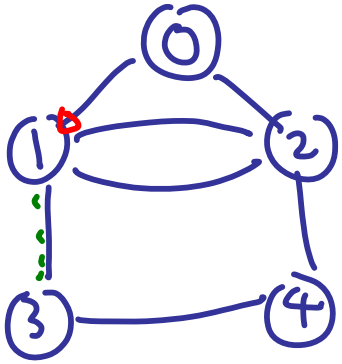
예





1 → 3 → 4 → 2 → 1 → 2 → 0 → 1

Euler cycle (한 개!)



0 → 1 → 2 → 0 (not a Euler cycle)

현재 path의 시작부터 scan.

1 (edges exist)

1부터 다시 경로를 이어갈 수 있을 때까지.

1 → 2 → 4 → 3 → 1

이 경로는 본래 경로에 있음

new cycle

0 → 1 → 2 → 4 → 3 → 1 → 2 → 0

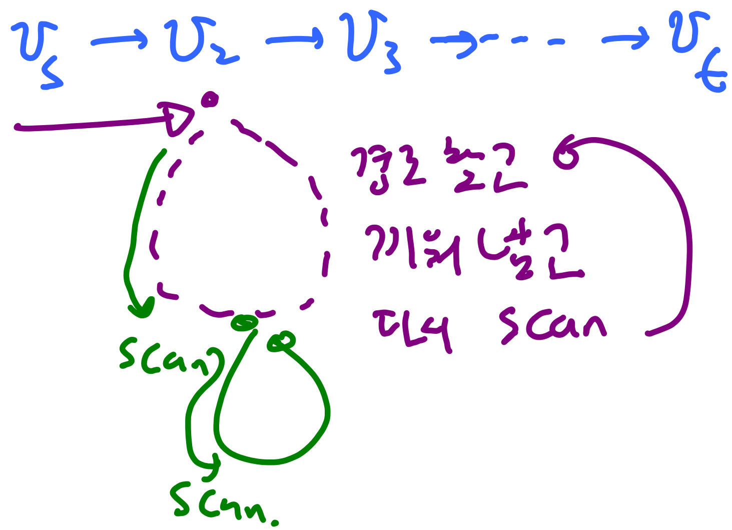
Euler Cycle ← ↑ 다시 검사 (미사용 edge가 있는 경우)

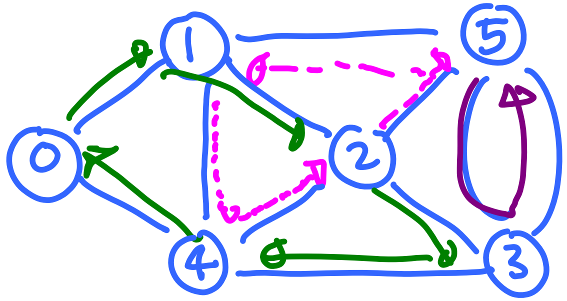
Euler Path 인 경우

① 홀수 degree 부터 시작해서 임의의 경로를 찾는다 (더이상 찾을 수 없는 때 까지)
또는 (홀수 degree vertex 에 도착)

② 그다음 위 사용한 edge 는 빼면 모든 vertex degree 는 짝수.

③





Ⓐ 5부터 시작해서 cycle 찾기

5 → 3 → 5

끼워 넣기

0 → 1 → 4 → 2 → 5 → 3 → 5 → 1 → 2 → 3 → 4 → 0

scan: 통과 통과 통과 통과 … 통과 → 끝

cycle 출력.

① 판정: degree check ⇒ Euler cycle 존재.

② 초기 cycle 찾기

0 → 1 → 2 → 3 → 4 → 0

반복 ③ scan

0 : 통과, 1 : unused incident edges exist

1 → 4 → 2 → 5 → 1

끼워 넣기

0 → 1 → 4 → 2 → 5 → 1 → 2 → 3 → 4 → 0

scan: 1 — 통과, 4 — 통과, 2 — 통과, 5 — Ⓐ

Programming 하려면

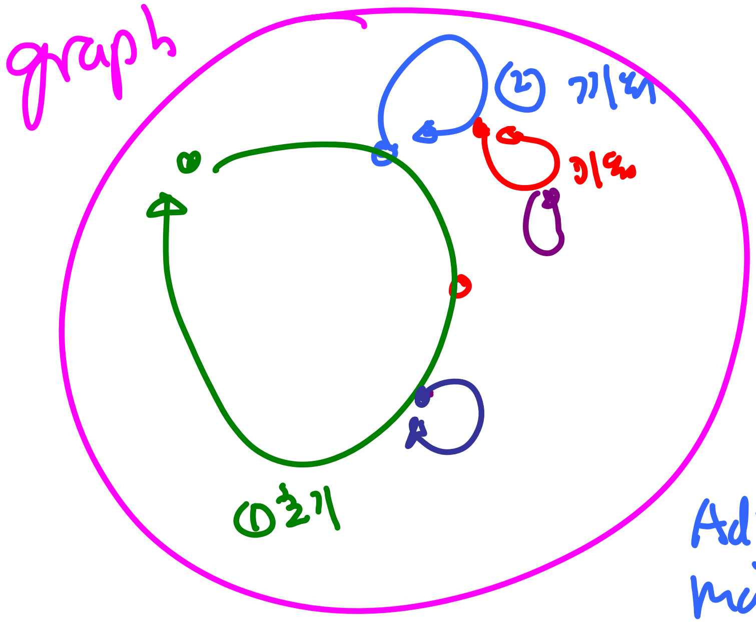
① Array 사용 ⇒ 끼워넣기가 비효율적.

② 가장 좋은 방법은? ⇒ linked list

Q: single linked / double linked

시간복잡도는 O( )?

(vertex의 개수 n
  edge " m



Adj matrix에선 또 다르다 아니도 $O(n^2)$

이미 사용된 edge는 mark해서 둔다.

각 vertex에서 해야 할 일은?

연결된 edge가 있는가? → 개수는 degree(v_i)

check (사용여부 check)

만일 사용된 edge는 지우면

⇒ 전체적으로 ~~edge 개수 만큼 걸림~~

끼워넣기 = O(1)

경로 만들기 = O(m)

Adj list는 사용할경우.

⇒ O(m) time

- Review Assignments
  - (Doubly) Linked List Operations.
  - Terminology in Trees and Graph (영어로!!)
  - Tree Traversal Methods: Preorder, Inorder, Postorder.
  - Binary Tree Representation.
  - Heap Operations.) complete binary tree, full binary tree
    array로 구현하는.
  - Union/Find Operations.
  - Definitions and Notations on Graphs. (adjacent, incident, etc) → 의미
  - Data Structures for Graphs.
  - DFS(Depth First Search) and BFS(Breath First Search).
  - Spanning Trees.