[CSE3081(2반)] 알고리즘 설계와 분석

2020학년도 2학기

강의자료

(2020.11.03 화요일)

서강대학교 공과대학 컴퓨터공학과
임 인 성 교수

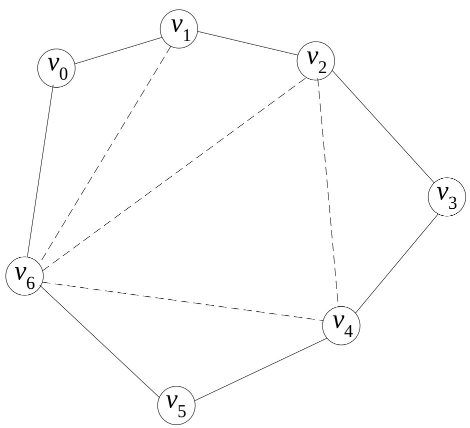
[주제 4]

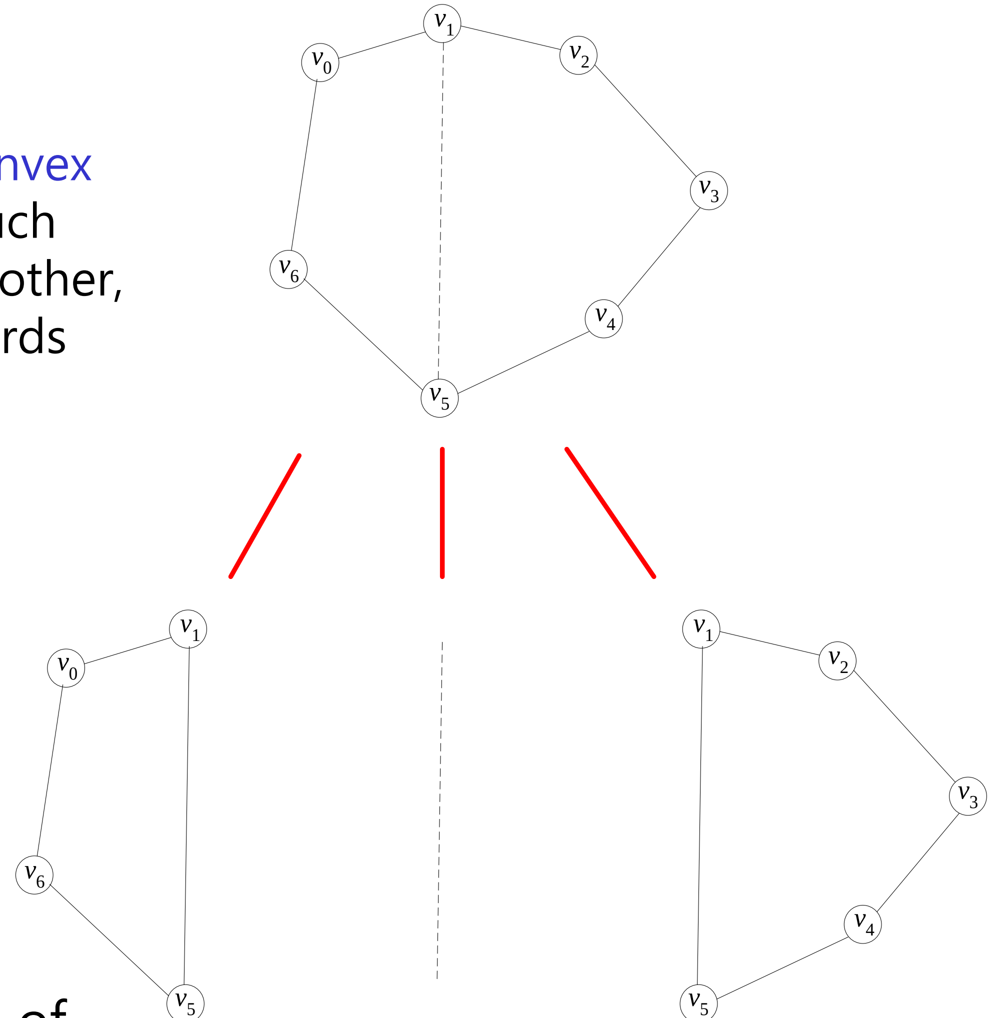
Dynamic Programming

[A. Aho, J. Hopcroft, and J. Ullman, *Data Structures and Algorithms*, Addison-Wesley, 1983. 10.2]

Minimal Triangulation

- Problem
  - Given a set of *n* vertices for convex polygon, find a triangulation such that no two chords cross each other, and the total length of the chords selected is a minimum.





- Counting all possible selections of chords in an inefficient way results in an exponential algorithm.

The 0-1 Knapsack Problem

- **Problem**

  Given two sets of positive integers $\{w_1, w_2, \cdots, w_n\}$ and $\{p_1, p_2, \cdots, p_n\}$ of size n and a positive integer W, find a subset A of $\{1, 2, \cdots, n\}$ that maximizes $\sum_{i \in A} p_i$ subject to $\sum_{i \in A} w_i \leq W$.

- **Example**

  $$\{w_1, w_2, \cdots, w_5\} = \{6, 5, 10, 3, 4\},\ \{p_1, p_2, \cdots, p_5\} = \{9, 7, 11, 6, 8\},\ W = 15$$
  $$\longrightarrow \{1, 2, 5\}$$

- **An intuitive interpretation**
  - There are n items in a store.
  - The i th item weighs w_i kilograms and is worth p_i wons, where w_i and p_i are positive integers.
  - A thief has a knapsack that can carry at most W kilograms, where W is a positive integer.
  - What items should the thief take to maximize his "profit"?

http://standardwisdom.com/softwarejournal/2010/03/bang-for-the-buck-knapsacks-in-real-life/

A 0-1 Knapsack Problem in Real Life

- **Problem**

  Given two sets of positive integers $\{w_1, w_2, \cdots, w_n\}$ and $\{p_1, p_2, \cdots, p_n\}$ of size n and a positive integer W, find a subset A of $\{1, 2, \cdots, n\}$ that maximizes $\sum_{i \in A} p_i$ subject to $\sum_{i \in A} w_i \leq W$.

  – You have a marketing budget of 5 million dollars.

  – You have the following marketing options and their paybacks in new potential customers:

Option	Cost (dollars)	Expected reach (people)
Super bowl	3M	80M
Radio ad campaign for 40 metro areas	800K	20M
TV non peak hour campaign	500K	22M
City top paper network	2M	75M
Viral marketing campaign	50K	4M
Web advertising	600K	10M

  – Which marketing campaigns would you choose to **maximize the total expected reach** under the condition that, for each of these marketing campaigns, you either select it or you don't?

How to Solve the 0-1 Knapsack Problem

- **Naïve approach**
  - There are 2^n subsets of $\{1, 2, \ldots, n\}$!

Given two sets of positive integers $\{w_1, w_2, \cdots, w_n\}$ and $\{p_1, p_2, \cdots, p_n\}$ of size n and a positive integer W, find a subset A of $\{1, 2, \cdots, n\}$ that maximizes $\sum_{i \in A} p_i$ subject to $\sum_{i \in A} w_i \leq W$.

- **Dynamic programming approach**
  - Let $P(i, w)$ be **the maximized profit** obtained when choosing items **only from the first *i* items** under the restriction that **the total weight cannot exceed *w***.
  - If we let A^* be an optimal subset of $\{1, 2, \ldots, n\}$,

$$
\begin{aligned}
&1.\ n \in A^* : \ P(n, W) = p_n + P(n-1, W - w_n) \\
&2.\ n \notin A^* : \ P(n, W) = P(n-1, W)
\end{aligned}
$$

  - **Optimal substructure**

$$
P(i, w) = \begin{cases} 0, & \text{if } i = 0 \text{ or } w = 0 \\ P(i-1, w), & \text{if } i > 0 \text{ and } w_i > w \\ \max\{\, P(i-1, w),\ p_i + P(i-1, w - w_i) \,\}, & \text{if } i > 0 \text{ and } w_i \leq w \end{cases}
$$

Example

$$P(i,w) = \begin{cases} 0, & \text{if } i = 0 \text{ or } w = 0 \\ P(i-1,w), & \text{if } i > 0 \text{ and } w_i > w \\ \max\{\, P(i-1,w),\ p_i + P(i-1, w-w_i)\,\}, & \text{if } i > 0 \text{ and } w_i \le w \end{cases}$$

$\{w_1, w_2, w_3, w_4\} = \{4, 3, 2, 3\}$, $\{p_1, p_2, p_3, p_4\} = \{3, 2, 4, 4\}$, $W = 6$

w

i	0	1	2	3	4	5	6
0	0	**0**	0	0	0	0	0
1	0	**0**	0	0	3	3	3
2	0	**0**	0	2	3	3	3
3	0	0	4	**4**	4	6	7
4	0	0	4	4	4	8	**8**

$P(2,4) = \max\{P(1,4),\ \underset{2}{p_2} + P(1, 4 - \underset{3}{w_2})\} = 3$ ② X

$P(4,2) = P(3,2) = 4$ ④ X

$P(3,5) = \max\{P(2,5),\ \underset{4}{p_3} + P(2, 5 - \underset{2}{w_3})\} = 6$ ③ √

How to Reconstruct the Solution

$$\{w_1, w_2, w_3, w_4\} = \{4,3,2,3\},\ \{p_1, p_2, p_3, p_4\} = \{3,2,4,4\},\ W = 6$$

w

i	0	1	2	3	4	5	6
0	0	**0**	0	0	0	0	0
1	0	**0**	0	0	3	3	3
2	0	**0**	0	2	3	3	3
3	0	0	4	**4**	4	6	7
4	0	0	4	4	4	8	**8**

$$\begin{cases} P(i-1, w), \\ \max\{\, P(i-1, w),\ p_i + P(i-1, w - w_i)\,\}, \end{cases}$$

$P(4,6) = \max\{P(3,6),\ \underset{4}{p_4} + P(3, \underset{3}{6 - w_4})\} = 8$ ④ ✓

$P(3,3) = \max\{P(2,3),\ \underset{4}{p_3} + P(2, \underset{2}{3 - w_3})\} = 4$ ③ ✓

$P(2,1) = P(1,1) = 0$ ② ✗

$P(1,1) = P(0,1) = 0$ ① ✗

Implementation and Time Complexity

```
int zero_one_knapsack(int *p, int *w, int n, int W) {
  int i, ww, tmp;
    …
  for (ww = 0; ww <= W; ww++) P[0][ww] = 0;
  for (i = 1; i <= n; i++) {
    P[i][0] = 0;
    for (ww = 1; ww <= W; ww++) {
      if (w[i] <= ww) {
        if ((tmp = p[i] + P[i-1][ww-w[i]]) > P[i-1][ww])
          P[i][ww] = tmp;
        else
          P[i][ww] = P[i-1][ww];
      }
      else P[i][ww] = P[i-1][ww];
    }
  }
  return P[n][W];
}
```

→ $O(nW)$ time

$$P(i,w) = \begin{cases} 0, & \text{if } i = 0 \text{ or } w = 0 \\ P(i-1,w), & \text{if } i > 0 \text{ and } w_i > w \\ \max\{\, P(i-1,w),\ p_i + P(i-1,w-w_i) \,\}, & \text{if } i > 0 \text{ and } w_i \le w \end{cases}$$

0-1 Knapsack Example 1: n = 6, W = 10

	1	2	3	4	5	6
p_i	6	4	5	3	9	7
w_i	4	2	3	1	6	4

$$P(i,w) = \begin{cases} 0, & \text{if } i = 0 \text{ or } w = 0 \\ P(i-1,w), & \text{if } i > 0 \text{ and } w_i > w \\ \max\{\, P(i-1,w),\ p_i + P(i-1,w-w_i) \,\}, & \text{if } i > 0 \text{ and } w_i \le w \end{cases}$$

P	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	6	6	6	6	6	6	6
2	0	0	4	4	6	6	10	10	10	10	10
3	0	0	4	5	6	9	10	11	11	15	15
4	0	3	4	7	8	9	12	13	14	15	18
5	0	3	4	7	8	9	12	13	14	16	18
6	0	3	4	7	8	10	12	14	15	16	**19**

Q	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	**0**	0	0	0	1	1	1	1	1	1	1
2	0	0	**1**	1	0	0	1	1	1	1	1
3	0	0	0	1	0	**1**	0	1	1	1	1
4	0	1	0	1	1	0	**1**	1	1	0	1
5	0	0	0	0	0	0	**0**	0	0	1	0
6	0	0	0	0	0	1	0	1	1	0	**1**

?	0	1	2	3	4	5	6	7	8	9	10
0											
1	**0**										
2			**2**								
3						**5**					
4							**6**				
5							**6**				
6											**10**

Selected items: i = 2, 3, 4, 6
Obtained profit: 19

- **Is the time-complexity $O(nW)$ an efficient one?**
  - This is not a linear-time algorithm!
    - A problem is that W is not bounded with respect to n.
    - What if n = 20 and W = 20!? → $O(n*n!)$
    - When W is extremely large in comparison with n, this algorithm is worse than the brute-force algorithm that simply considers all subsets.
  - This algorithm can be improved so that the worst-case number of entries computed is $O(2^n)$.

✓ No one has ever found an algorithm for the 0-1 Knapsack problem whose worst-case time complexity is better than exponential, yet no one has proven that such an algorithm is not possible!