[CSE3081(2반)] 알고리즘 설계와 분석

2020학년도 2학기

강의자료

(2020.11.17 화요일)

서강대학교 공과대학 컴퓨터공학과
임 인 성 교수

본 강의에서 제작하여 제공하는 PDF 파일, 동영상, 그리고 예제 코드 등의 강의 자료의 저작권은 특별히 명기되어 있지 않은 한 서강대학교에 있습니다.

본인의 학습 목적 외에 공개된 장소에 올리거나 타인에게 배포하는 등의 행위를 금합니다. 협조 부탁합니다.

[주제 5]
Greedy Methods

Scheduling with Deadlines

- **Problem**
  - Let $J = \{1, 2, \ldots, n\}$ be a set of jobs to be served.
  - Each job takes one unit of time to finish.
  - Each job has a **deadline** and a **profit**.
    - If the job starts before or at its deadline, the profit is obtained.
  - ➢ Schedule the jobs so as to **maximize** the total profit (not all jobs have to be scheduled).

- **Example:**

Job	Deadline	Profit
1	2	30
2	1	35
3	2	25
4	1	40

→

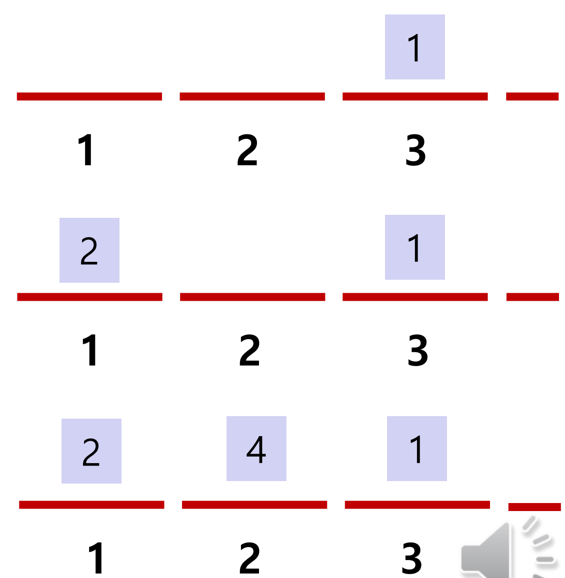
Schedule	Total Profit
[1, 3]	30 + 25 = 55
[2, 1]	35 + 30 = 65
[2, 3]	35 + 25 = 60
[3, 1]	25 + 30 = 55
[4, 1]	☞ 40 + 30 = 70
[4, 3]	40 + 25 = 65

- **A greedy approach**
  - Sort the jobs **in non-increasing order by profit**.
  - Scan each job in the sorted list, adding it to the schedule if possible.

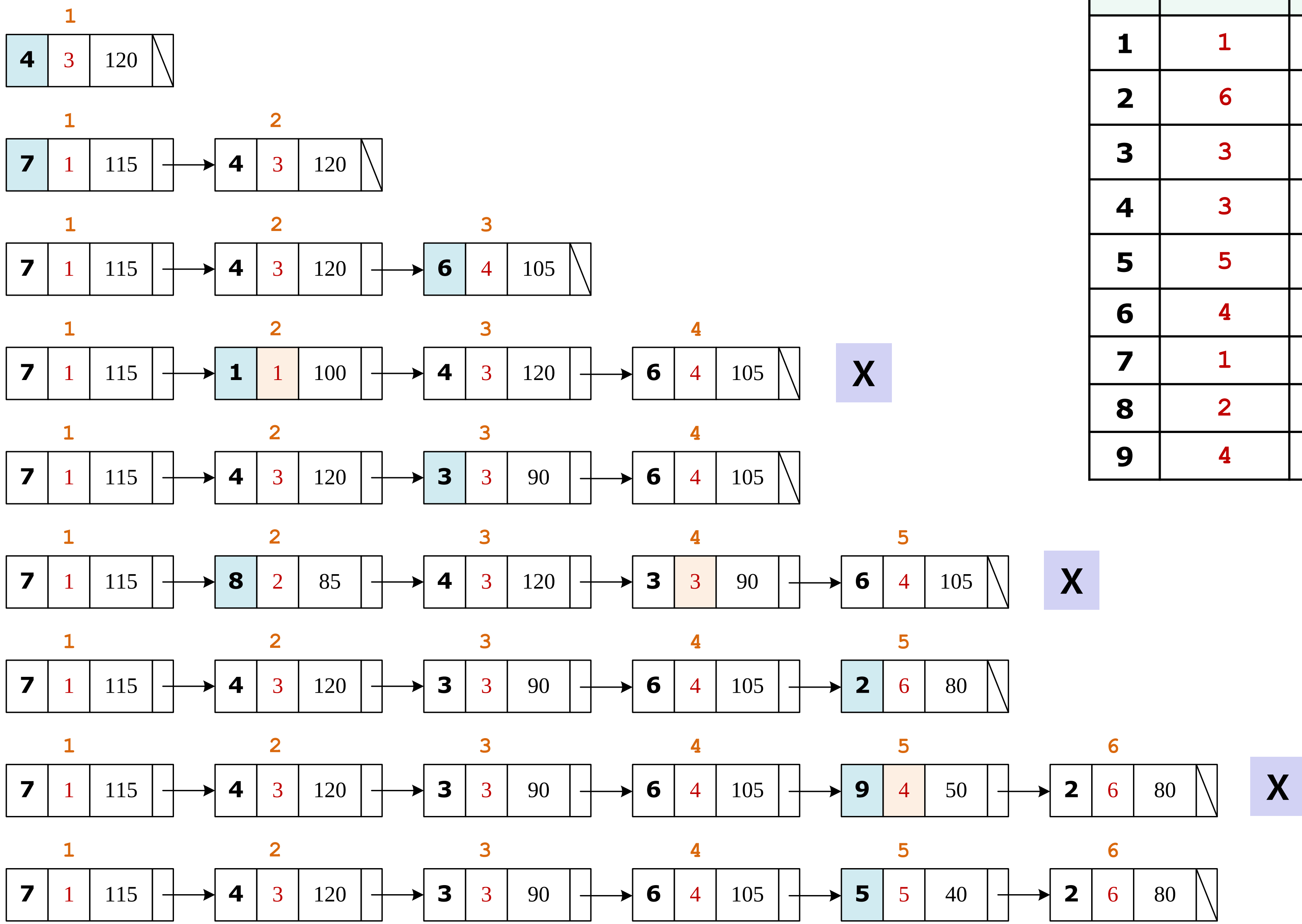
- **Example**
  - S = EMPTY
  - Is S = {1} OK?
    - Yes: $S \leftarrow$ {1} ([1])
  - Is S = {1, 2} OK?
    - Yes: $S \leftarrow$ {1, 2} ([2, 1])
  - Is S = {1, 2, 3} OK?
    - No.
  - Is S = {1, 2, 4} OK?
    - Yes: $S \leftarrow$ {1, 2, 4} ([2, 1, 4] or [2, 4, 1])
  - Is S = {1, 2, 4, 5} OK?
    - No.
  - Is S = {1, 2, 4, 6} OK?
    - No.
  - Is S = {1, 2, 4, 7} OK?
    - No.

<After sorting by profit>

Job	Deadline	Profit
1	3	40
2	1	35
3	1	30
4	3	25
5	1	20
6	3	15
7	2	10



- **Example**



Job	Deadline	Profit
1	1	100
2	6	80
3	3	90
4	3	120
5	5	40
6	4	105
7	1	115
8	2	85
9	4	50

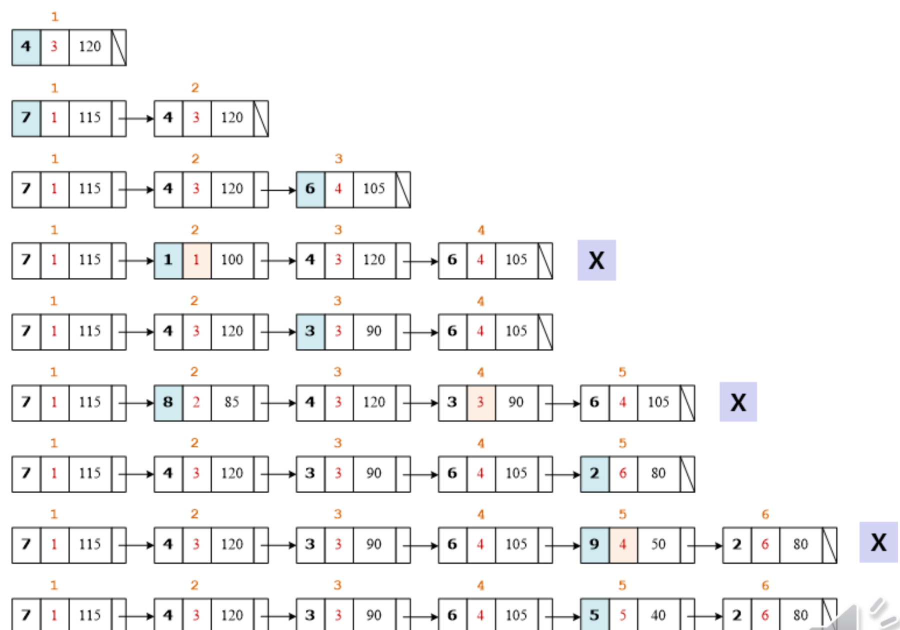
Implementation Issues

- **A key operation in the greedy approach**
  - Determine if a set of jobs S is **feasible**.
  - ✓ Fact: S is feasible if and only if the sequence obtained by ordering the jobs in S according to nondecreasing deadlines is feasible.
  - Example
    - Is $S = \{1, 2, 4\}$ OK? → [2(1), 1(3), 4(3)] → Yes!
    - Is $S = \{1, 2, 4, 7\}$ OK? → [2(1), 7(2), 1(3), 4(3)] → No

- **An $O(n^2)$ implementation**
  - Sort the jobs in non-increasing order by profit.
  - For each job in the sorted order,
    - See if the current job can be scheduled together with the previously selected jobs, using a *linked list* data structure.
      - If yes, add it to the list of feasible sequence.
      - Otherwise, reject it.
  - ✓ **Time complexity** $$O(n \log n) + \sum_{i=2}^{n} \{ (i-1) + i \} = O(n^2)$$
    - When there are i-1 jobs in the sequence,
      - at most i-1 comparisons are needed to add a new job in the sequence, and
      - at most i comparisons are needed to check if the new sequence is feasible.

- Is the time complexity always $O(n^2)$?
  - What if $n >> d_{max}$?
    - $O(n \log n + n\, d_{max})$
  - What if $n >> d_{max}$ and $n >> k_{scanned}$?
    - $O(n + k_{scanned} \log n + k_{scanned}\, d_{max}) = O(n)$ ← Is this complexity achievable when a max heap data structure is employed?

Job	Deadline	Profit
1	1	100
2	6	80
3	3	90
4	3	120
5	5	40
6	4	105
7	1	115
8	2	85
9	4	50



Correctness of the Greedy Method

- Left as an exercise.

Data Structures for Disjoint Sets

- **Partition**
  - A partition of a set X is a set of non-empty subsets of X such that every element x in X is in exactly one of these subsets.
  - **Example: X = {1, 2, 3, 4, 5, 6} → { {1, 3, 5}, {2}, {4, 6} }**

- **Disjoint-set data structure** (**union-find data structure**)
  - Used to effectively manage a collection of subsets that partition a given set of elements.

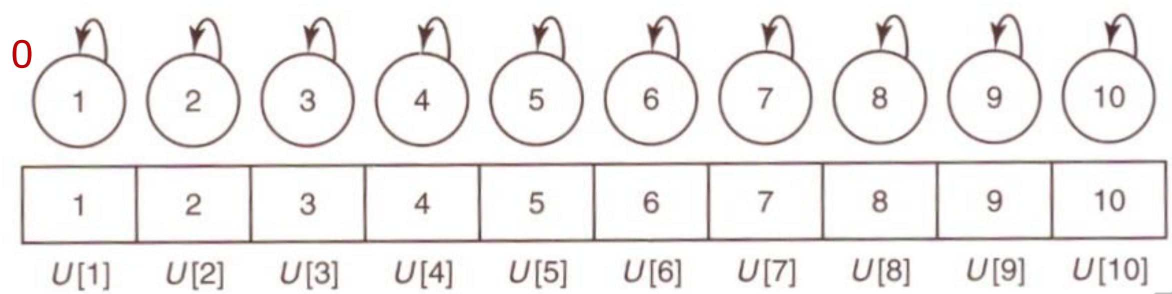
- **Basic operations on disjoint sets**
  - **Makeset(*x*)**
    - Make a set containing only the given element *x*.
  - ***S* = Find(*x*)**
    - Determine which set the particular element *x* is in.
      - Typically return an element that serves as the subset's representative.
      - May be used to determine if two elements are in the same subset.
  - **Union(x, y)** (or Merge(*x*, *y*))
    - Merge two subsets into a single subset.

- **Applications**
  - **Tracking the connected components of an undirected graph**
    - Decide if two vertices belong to the same component, or if adding an edge between them would result in a cycle.
    - Useful for implementing **the Kruskal's algorithm for finding minimum spanning tree**
  - **Scheduling with deadlines**
  - Computing shorelines of a terrain
  - Classifying a set of atoms into molecules or fragments.
  - Connected component labeling in image analysis

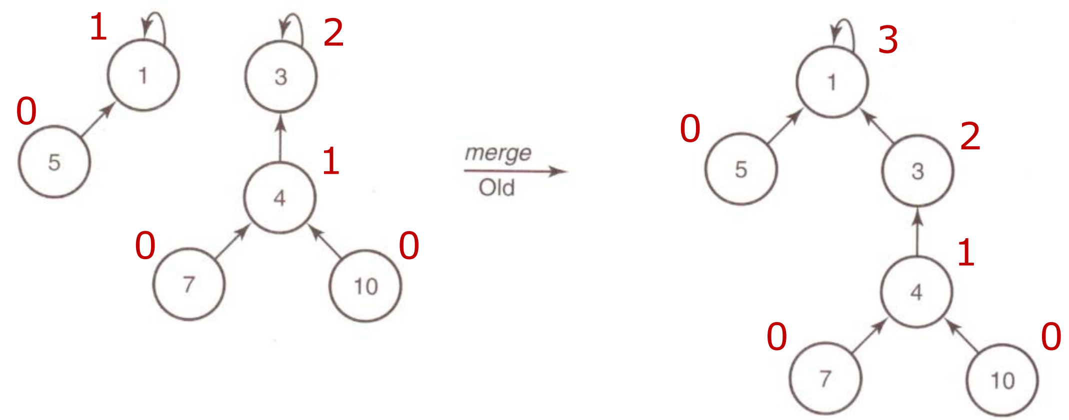
- **Example**
  - U = {a, b, c, d, e}
  - For (each x in U) **Makeset(*x*);** → {a}, {b}, {c}, {d}, {e}
  - **Union(*a*, *c*);** → {a, c}, {b}, {d}, {e}
  - **{a, c} = Find(a);**
  - **Union(*c*, *e*);** → {a, c, e}, {b}, {d}
- **Implementation of disjoint sets using *reversed trees***
  - **Makeset(*x*)**

```
Makeset(x) {
  parent(x) := x
  rank(x) := 0
}
```

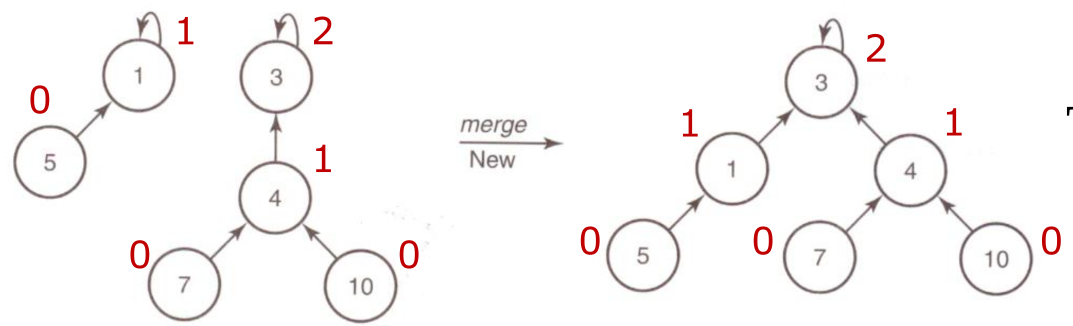
Time complexity: $O(1)$



- **Two ways of implementing the Union operation**



Time complexity: $O(n)$



Time complexity: $O(\log n)$

Union by rank

- Always attach the smaller tree to the root of the larger tree.
- The **rank** increases by one only if two trees of the same rank are merged.
  - ✓ The rank of a one-element tree is zero.
- The Union and Find operations can be done in O(log n) in the worst case.
  - ✓ The number of elements in a tree of rank r is at least 2^r. (Proof by induction)
  - ✓ The maximum possible rank of a tree with n elements is O(log n).

- ***S* = Find(*x*)**

Time complexity: O(depth of x in the tree)

```
Find(x) {
  while (x != parent(x))
    x := parent(x)
  return x
}
```

```
Find(x) {
  if (x == parent(x))
    return x
  else
    return Find(parent(x))
}
```

- **Union(*x*, *y*)**

```
Union(x, y) {
  x0 := Find(x)
  y0 := Find(y)
  if (x0 == y0)
    return
  if (rank(x0) > rank(y0))
    parent(y0) := x0
  else
    parent(x0) := y0
    if (rank(x0) == rank(y0))
      rank(y0) := rank(y0)+1
}
```

Time complexity:

2 Find op's + O (1)