AAI2007 Introduction to Algorithms

Week 13: Greedy Algorithms

Instructor: Jinyoung Han (jinyounghan@skku.edu)

Schedule

Tentative Schedule

수업일	내용
9/4	Course Introduction, Algorithm Basic, Level Test
9/11	Order of Complexity, List
9/18	Stack, Queue
9/25	건학 기념일
10/2	Tree, Binary Search Tree (BST)
10/9	Priority Queue, Heap, Heap Sort 한글날
10/16	Hash Table, Searching Revisited
10/23	Graph Basic
10/30	Midterm Exam
11/6	Graph Algorithms
11/13	Sorting
11/20	Dynamic Programming (1)
11/27	Dynamic Programming (2)
12/4	Greedy Algorithms
12/11	Algorithm Practice (Google software engineer), "구글 소프트웨어 엔지니어가 설명해주는 알고리즘 공부 팁", 국제관 9B217
12/18	Final Exam

Schedule

Final Notice

- Time: 12/18, Wednesday 6pm~9pm
  - Allow to leave from around 7pm (after 1 hour from starting)
- Location: same with midterm
- Scope: All
- Midterm environment: Python idle
  - Will be explained by TA
- Closed book, no Internet, no cell phone

Schedule

Final Notice

교육학과	2013312973	김도균
한문교육과	2015311588	김결
컴퓨터교육과	2019319220	ALI RAZA MALIK
컴퓨터교육과	2017310669	윤혜정
국어국문학과	2015312762	최예은
국어국문학과	2016311484	장윤예
독어독문학과	2015313707	박채원
러시아어문학과	2012313961	윤승록
한문학과	2013310866	이성현
사학과	2013310406	안세운
문헌정보학과	2016313545	최지연
기계공학부	2014312298	윤관식
건축학과	2014311748	이석현
화학공학/고분자공학부	2016312825	강규란
심리학과	2013312615	우도균
심리학과	2016310768	오소영
글로벌리더학부	2013312097	김도영
글로벌리더학부	2014314865	곽준원
경제학과	2016310051	한지선
경제학과	2015311039	임지윤
통계학과	2014311688	한대룡
통계학과	2016310142	이정의
통계학과	2016311924	양지연
통계학과	2016313680	김수진

[수선관 6층 PC실]

경영학과	2017313398	서가영
경영학과	2014312601	김태경
글로벌경영학과	2013310419	이동은
글로벌경영학과	2014314081	김주한
디자인학과	2016314216	이상아
영상학과	2016311814	남정원
의상학과	2015312542	이은혜
의상학과	2016312126	안리아
수학과	2014311729	이대희
전자전기공학부	2014312256	정우석
융합생명공학과	2015312815	김혜빈
데이터사이언스융합전공	2018312024	김다솔
인공지능융합전공	2018310412	고준서
인공지능융합전공	2018311375	김규리
인공지능융합전공	2018311886	주소미
인공지능융합전공	2018312827	김민지
인공지능융합전공	2018312986	임준우
인공지능융합전공	2018313960	김근석
인공지능융합전공	2018314025	권혜현
인공지능융합전공	2018314374	고귀환
인공지능융합전공	2018314529	이상은
인공지능융합전공	2018314848	한승희
미디어커뮤니케이션학과	2014314487	이교영

[호암관 3층 PC실 50313]

Schedule

Check Your Scores

- Time: 12/20, Friday 1pm~4pm
  - Last chance
- Location: 국제관 9B303

In This Lecture

Outline

1. Greedy Approach
2. Coin Change Problem
3. Activity Selection Problem
4. Minimum Spanning Tree (MST) Problem
5. Single-Source Shortest Path Problem

Optimization Problem

- An optimization problem
  - Given a problem instance, a set of **constraints** and an **objective function**
  - Find a **feasible** solution for the given instance for which the objective function has an optimal value
  - Either maximum or minimum depending on the problem being solved
- A feasible solution satisfies the problem's constraints
- The constraints specify the limitations on the required solutions.
  - Example (knapsack problem)
    - Require that the items in the knapsack will not exceed a given weight

01. Greedy Approach

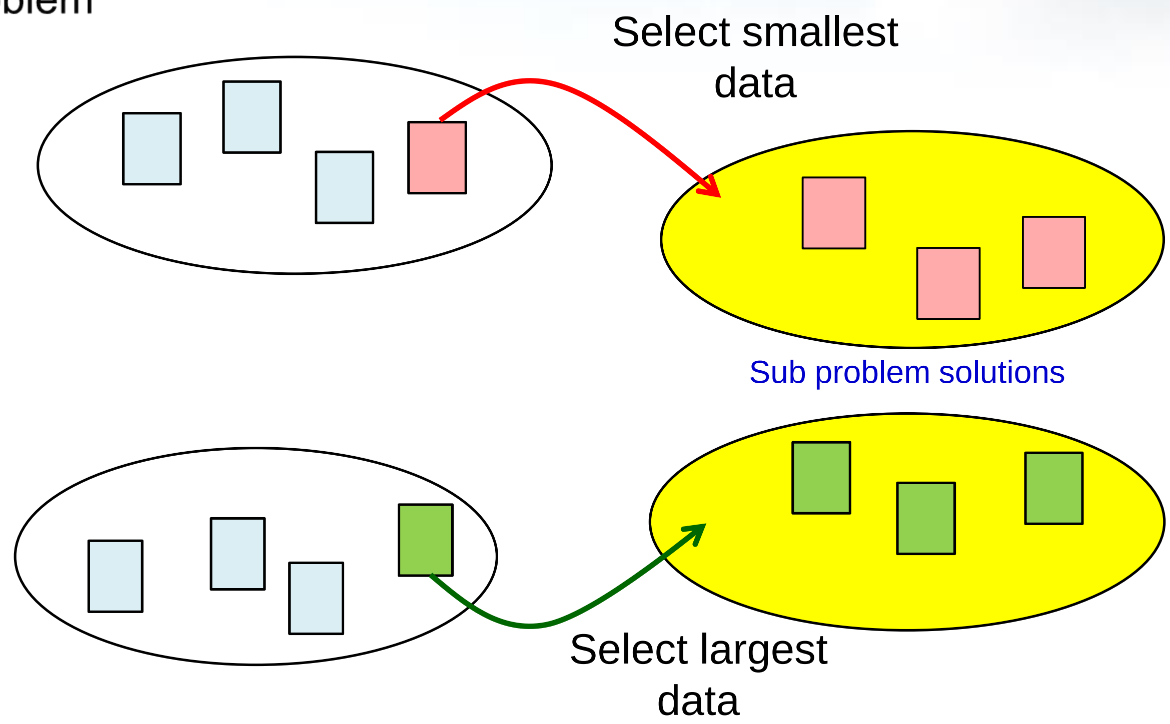
Greedy

- Greedy algorithms make **good local choices** in the hope that they result in an optimal solution
  - They result in feasible solutions
    - satisfying the constraints
  - Not necessarily an optimal solution
- A proof is needed to show that the algorithm finds an optimal solution
- A counter example shows that the greedy algorithm does not provide an optimal solution

01. Greedy Approach

Greedy

- A short-sighted selection
  - Finds a optimal solution for a sub-problem
  - And then such sub-problems result in the final optimal solution for the original problem



01. Greedy Approach

A Pseudo Code

```
Greedy (Candidate){
   solution= new Set( );
   while (Candidate.isNotEmpty()) {
                next = Candidate.select(); //use selection criteria,
               //remove from Candidate and return value
                if (solution.isFeasible(next)) //constraints satisfied
                        solution.union(next);
                if (solution.solves()) return solution
   }
   //No more candidates and no solution
   return null
}
```

- select() chooses a candidate based on a local selection criteria, removes it from Candidate, and returns its value
- isFeasible() checks whether adding the selected value to the current solution can result in a feasible solution (no constraints are violated)
- solves() checks whether the problem is solved

Coin Change Problem

02. Coin Change Problem

Problem

- Problem
  - Return correct change using a minimum number of coins
  - a.k.a cashier’s algorithm
- Greedy choice
  - Coin with highest coin value
- Example of American money
  - (quarter: 25 cent, dime: 10 cent, nickel: 5 cent, penny: 1 cent)
  - The amount owed = 37 cent
  - The change is: 1 quarter -> 1 dime -> 2 pennies

02. Coin Change Problem

Algorithm

```
Input: Set of coins of different denominations, amount-owed
change = {}
while (more coin-sizes && valueof(change)<amount-owed)
    Choose the largest remaining coin-size // Selection
    // feasibility check
       while (adding the coin does not make the valueof(change) exceed the amount-owed) then
                add coin to change
   //check if solved
    if (valueof(change) equals amount-owed)
       return change
    else delete coin-size
return "failed to compute change"
```

02. Coin Change Problem

Pseudo Code

```
CoinChange for Korean won (500 won, 100 won, 50 won, 10 won, 1 won)

1. change=W, n500=n100=n50=n10=n1=0
2. while ( change ≥ 500 )
       change = change-500, n500++      // increase the count of 500 won
3. while ( change ≥ 100 )
       change = change-100, n100++      // increase the count of 100 won
4. while ( change ≥ 50 )
       change = change-50, n50++        // increase the count of 50 won
5. while ( change ≥ 10 )
       change = change-10, n10++        // increase the count of 10 won
6. while ( change ≥ 1 )
       change = change-1, n1++          // increase the count of 1 won
7. return (n500+n100+n50+n10+n1)
```

*** Known as optimal under a certain condition**

Activity Selection Problem

03. Activity Selection Problem

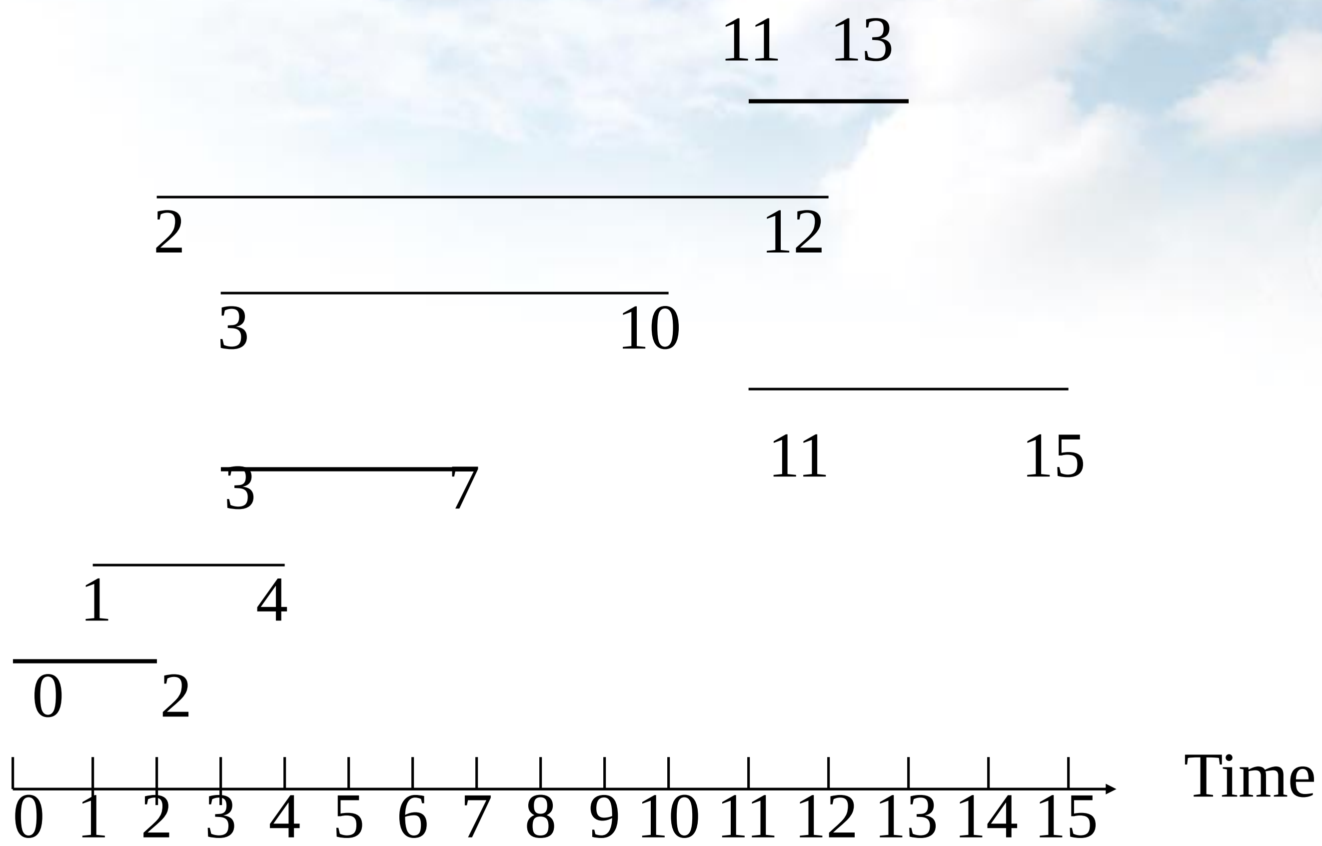
Problem

- Given a set S of n activities with start time s_i and finish time f_i of activity i
- Find a maximum size subset A of compatible activities (maximum number of activities)
  - Activities are compatible if they do not overlap
- Can you suggest a greedy choice?

03. Activity Selection Problem

Example

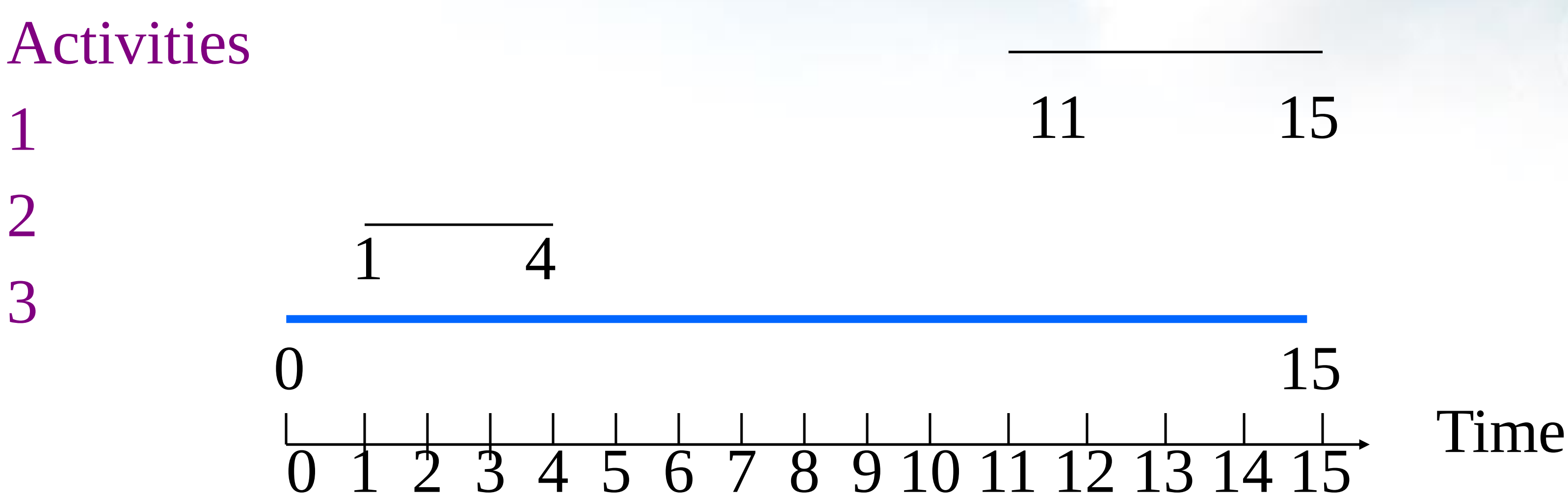
Activities
1
2
3
4
5
6
7



03. Activity Selection Problem

Example

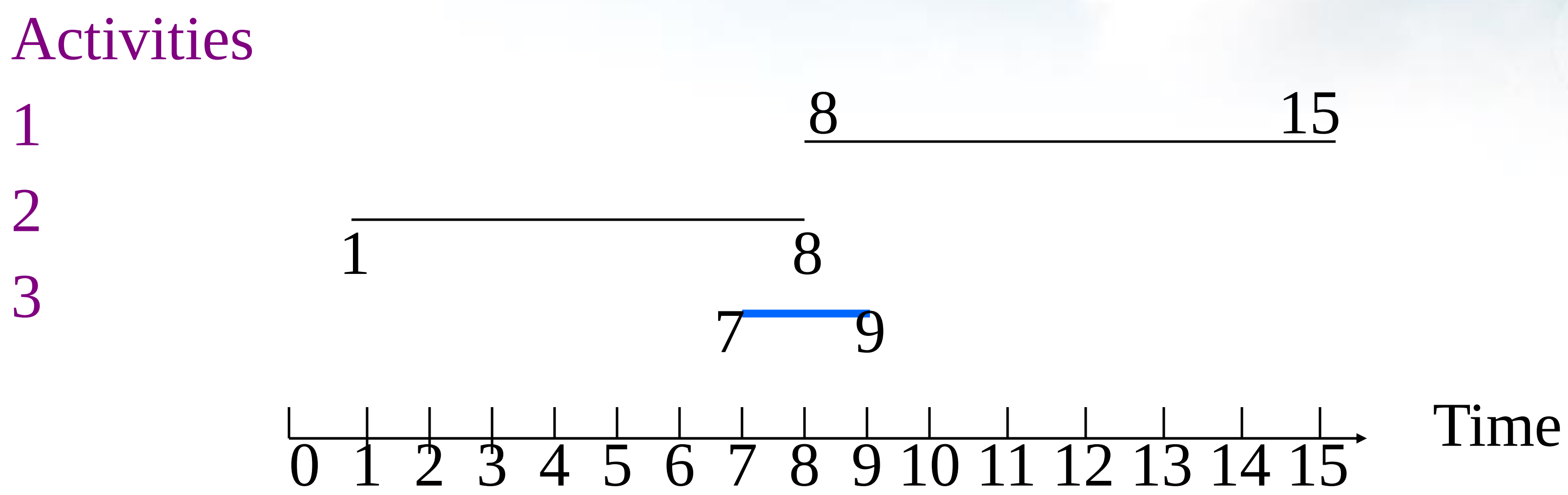
- Counter example, select by start time



03. Activity Selection Problem

Example

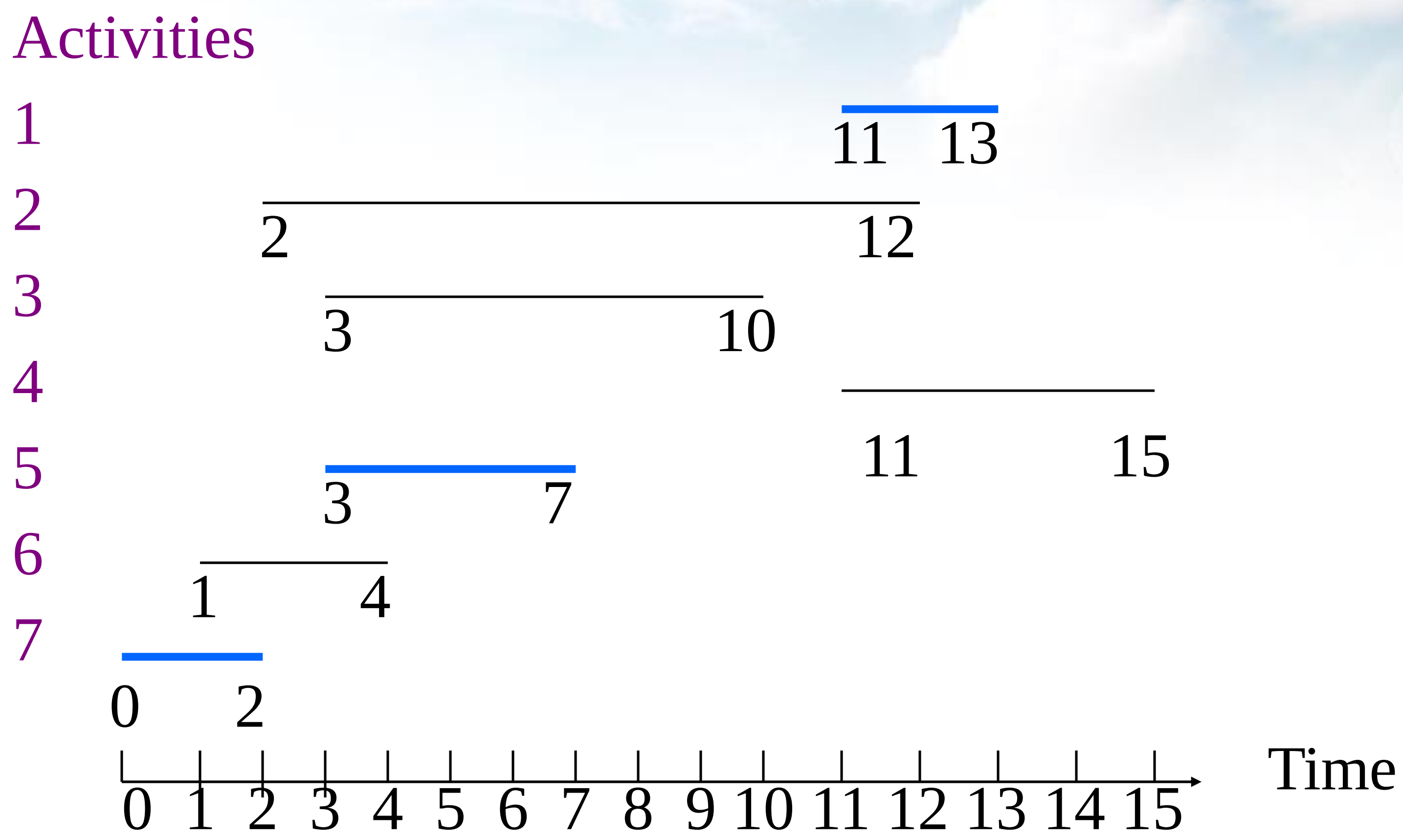
- Counter example, select by minimum duration



03. Activity Selection Problem

Example

- Select by finishing time



03. Activity Selection Problem

Algorithm

- Assume without loss of generality that we number the intervals in order of finish time
  - So f1<=...<=fn
- Greedy choice: choose activity with minimum finish time
- The following greedy algorithm starts with A={1} and then adds all compatible jobs (O(n))
  - O(nlogn) when including sort

```
n <- length[s] // number of activities
A <- {1}
j <- 1 // last activity added
for i <- 2 to n // select
    if si >= fj then // compatible (feasible)
        add {i} to A
        j <- i // save new last activity
return A
```

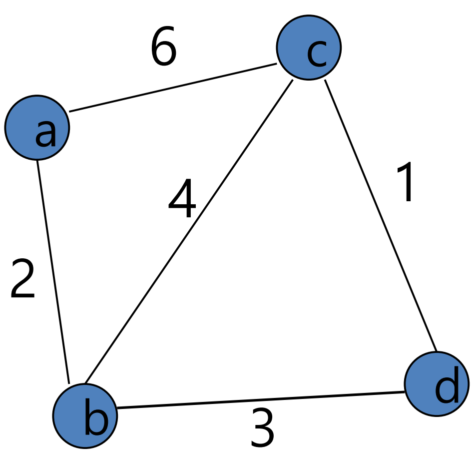
*** Known as optimal.**

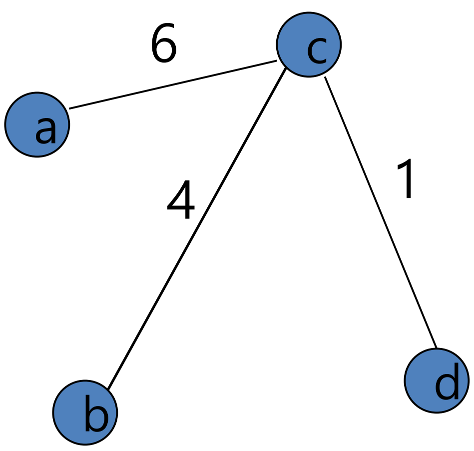
Minimum Spanning Tree (MST)

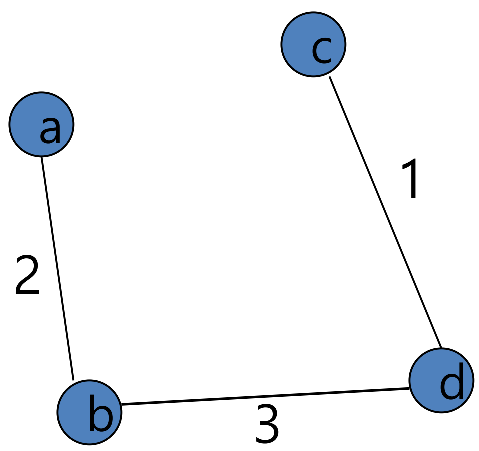
04. Minimum Spanning Tree (MST)

MST

- Spanning tree of a connected graph G
  - A connected acyclic subgraph of G that includes all of G’s vertices
- Minimum spanning tree of a weighted, connected graph G
  - A spanning tree of G of the minimum total weight
- Example







04. Minimum Spanning Tree (MST)

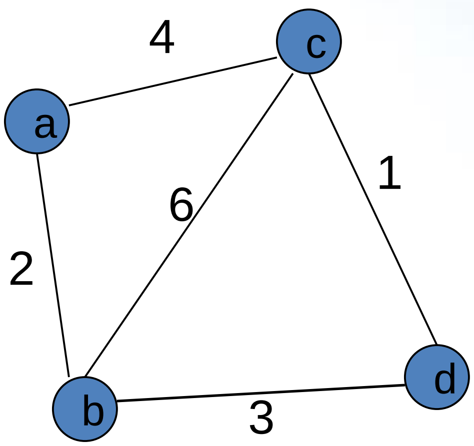
Prim's MST

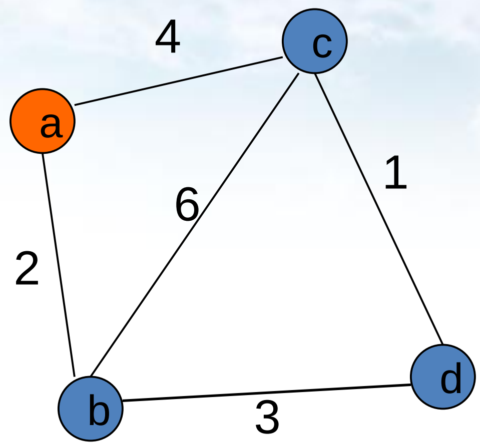
- Algorithm
  - Start with tree T_1 consisting of one (any) vertex and "grow" tree
    - One vertex at a time to produce MST through a series of expanding subtrees $T_1, T_2, \ldots, T_n$
  - On each iteration, construct T_{i+1} from T_i by adding vertex not in T_i that is closest (or lightest) to those already in T_i
    - this is a "greedy" step!
  - Stop when all vertices are included

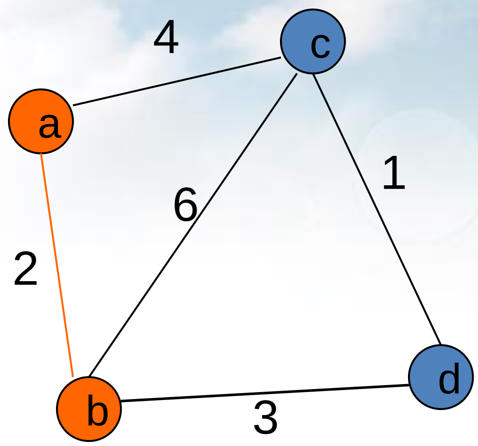
04. Minimum Spanning Tree (MST)

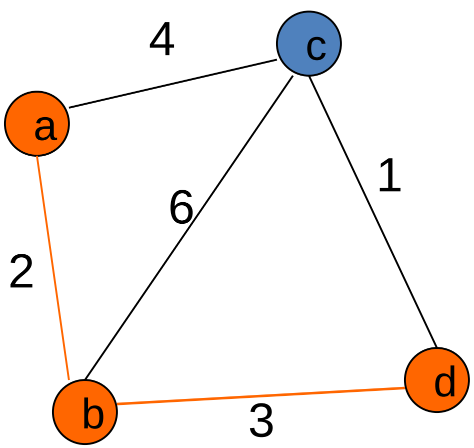
Prim's MST

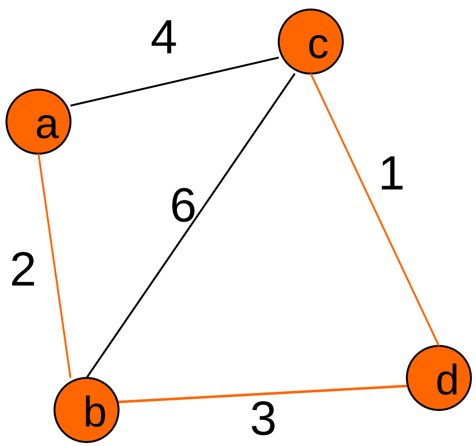
- Example











04. Minimum Spanning Tree (MST)

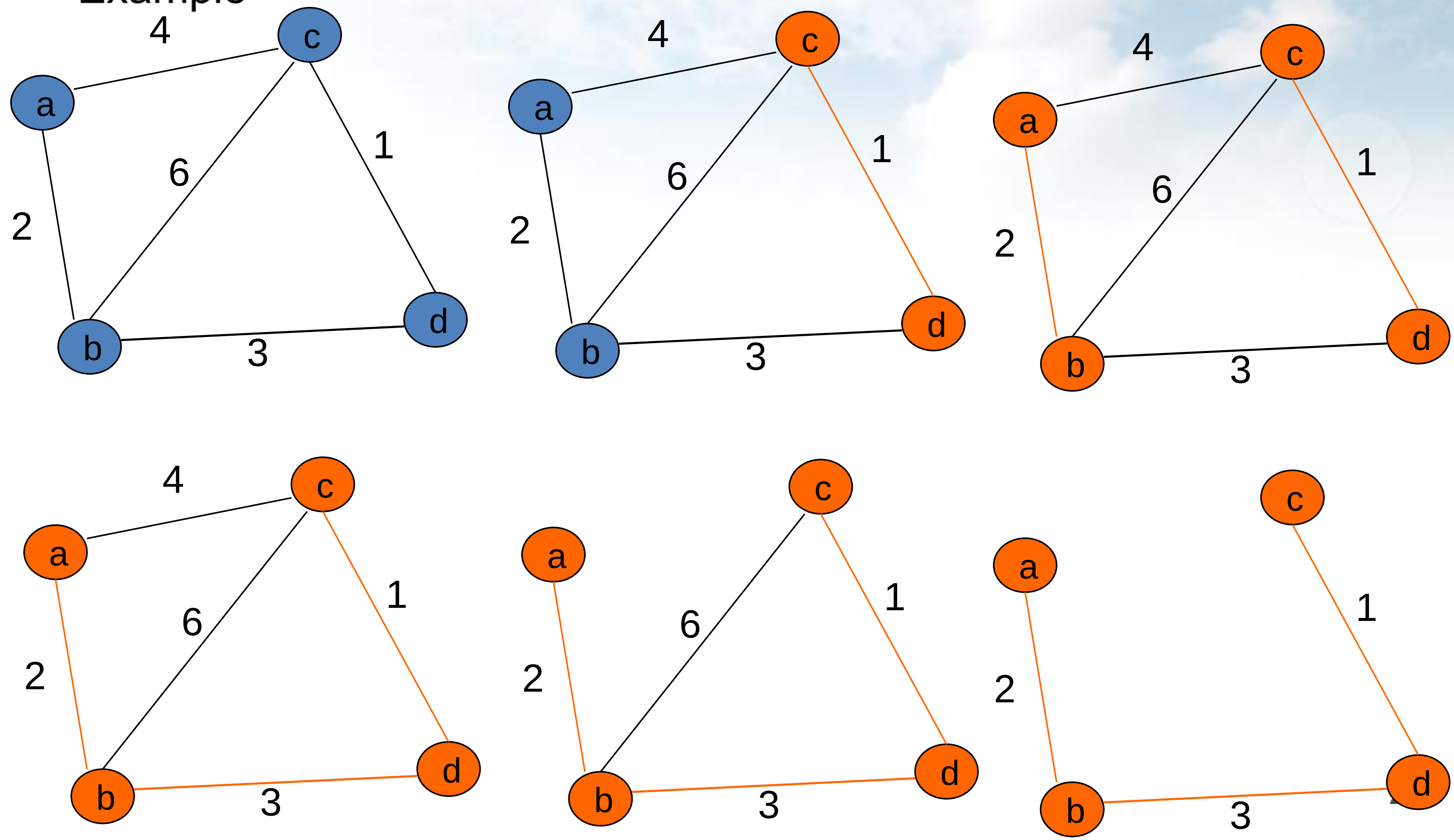
Kruskal's MST

- Algorithm
  - Sort the edges in non-decreasing order of lengths
  - "Grow" tree one edge at a time to produce MST through a series of expanding forests $F_1, F_2, \ldots, F_{n-1}$
  - On each iteration, add the next edge on the sorted list unless this would create a cycle
    - If it would, skip the edge

04. Minimum Spanning Tree (MST)

Kruskal's MST

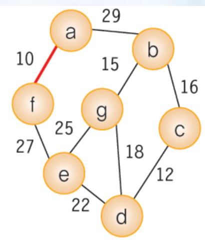
- Example



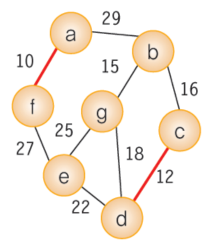
Kruskal's MST

- Example

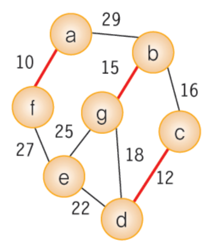
af	cd	bg	bc	dg	de	eg	ef	ab
10	12	15	16	18	22	25	27	29



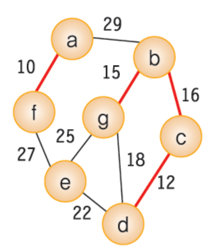
af	**cd**	bg	bc	dg	de	eg	ef	ab
10	12	15	16	18	22	25	27	29



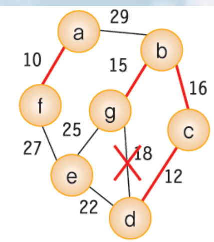
af	cd	**bg**	bc	dg	de	eg	ef	ab
10	12	15	16	18	22	25	27	29



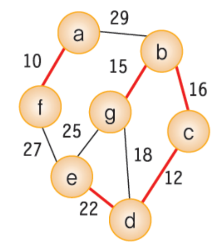
af	cd	bg	**bc**	dg	de	eg	ef	ab
10	12	15	16	18	22	25	27	29



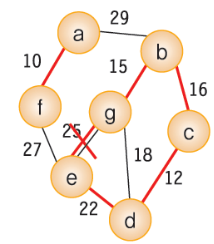
af	cd	bg	bc	**dg**	de	eg	ef	ab
10	12	15	16	18	22	25	27	29



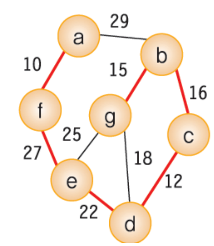
af	cd	bg	bc	dg	**de**	eg	ef	ab
10	12	15	16	18	22	25	27	29



af	cd	bg	bc	dg	de	**eg**	ef	ab
10	12	15	16	18	22	25	27	29



af	cd	bg	bc	dg	de	eg	**ef**	ab
10	12	15	16	18	22	25	27	29



04. Minimum Spanning Tree (MST)

Comparisons

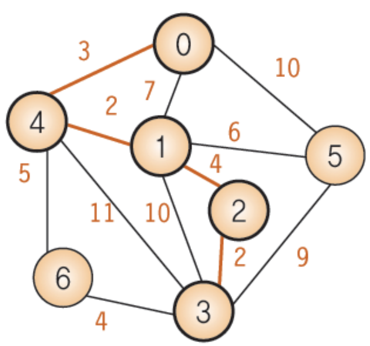
- Kruskal's algorithm looks easier than Prim's but is harder to implement (checking for cycles!)
  - Cycle checking: a cycle is created iff added edge connects vertices in the same connected component
    - Union-find algorithms
- Kruskal vs. Prim
  - Kruskal: from n trees to 1 mst
  - Prim: from a tree to 1 mst

Single-Source Shortest Path Problem

05. Shortest Path Problem

Shortest Path

- Among the paths that connect node u and node v in the given network, the path where sum of weights on the edges is minimum
  - Weight can be cost, distance, time, etc.
- A problem: to find the shortest path from node 0 to node 3
  - Adjacent matrix
    - If there is no direct edge, its weight is ∞
  - 0, 4, 1, 2, 3 is the shortest path
    - Length of the shortest path = 3 + 2 + 4 + 2 = 11



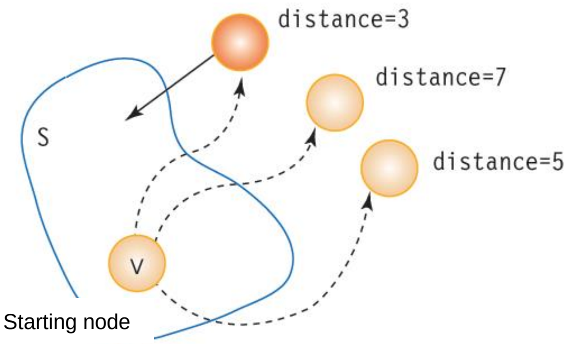
	0	1	2	3	4	5	6
0	0	7	∞	∞	3	10	∞
1	7	0	4	10	2	6	∞
2	∞	4	0	2	∞	∞	∞
3	∞	10	2	0	11	9	4
4	3	2	∞	11	0	∞	5
5	10	6	∞	9	∞	0	∞
6	∞	∞	∞	4	5	∞	0

05. Shortest Path Problem

Dijkstra

- Algorithm
  - Search the shortest paths from “a starting node” to all the other nodes
  - Set S
    - A set of nodes (already) included in the shortest path starting from node v
  - Distance variables
    - The shortest paths from node v to other nodes
    - Initialization (starting node v)
      - distance[v] = 0
      - distance[n] = w(v, w) if an edge exists,

        ∞ otherwise

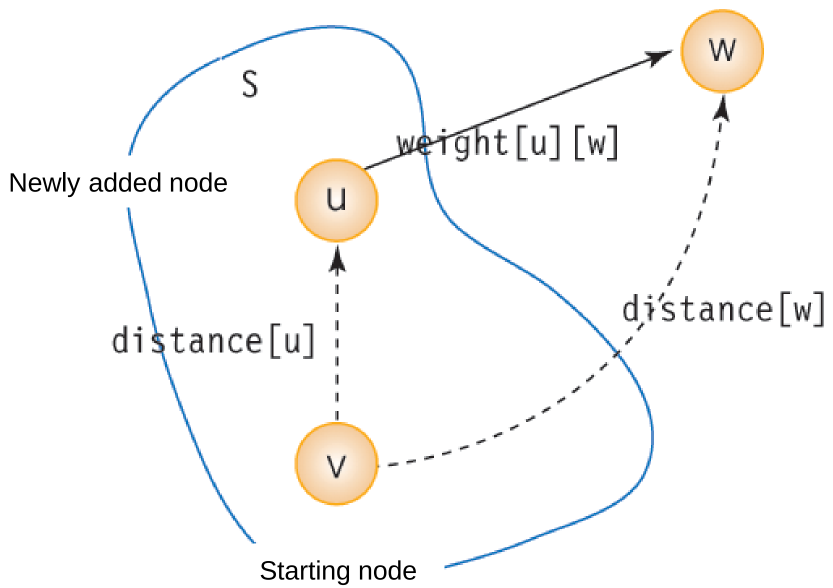


  - For each step, a node with the smallest distance is added to S

Dijkstra

- Algorithm
  - Update distance variables if a node is added to S

```
distance[w]=min(distance[w], distance[u]+weight[u][w])
```



05. Shortest Path Problem

Dijkstra

- Pseudo code

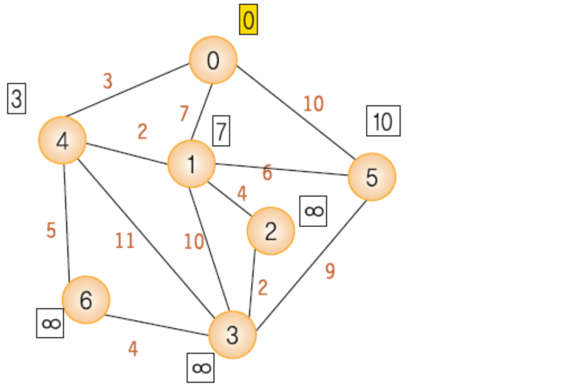
```
// input: a (non-negative) weighted graph G
// output: distance array, distance[u] is the shortest path from v to u

shortest_path(G, v)

S←{v}
for each node w∈G do
        distance[w]←weight[v][w];
while all the nodes are not contained in S do
        u← node with smallest distance, which are not included in S;
        S←S∪{u}
        for u's adjacent and member of S "z" do
                if distance[u]+weight[u][z] < distance[z]
                        then distance[z]←distance[u]+weight[u][z];
```

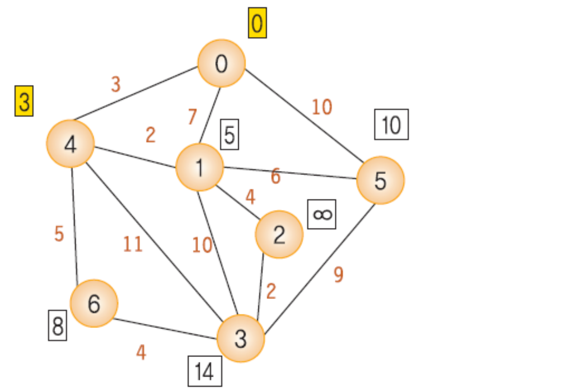
Dijkstra

- Example



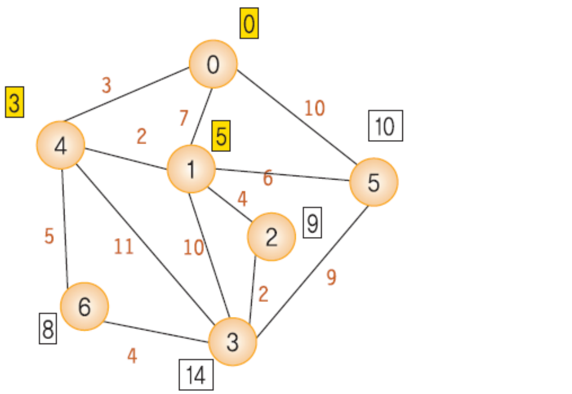
S={0}

	0	1	2	3	4	5	6
distance[]=	0	7	∞	∞	3	10	∞



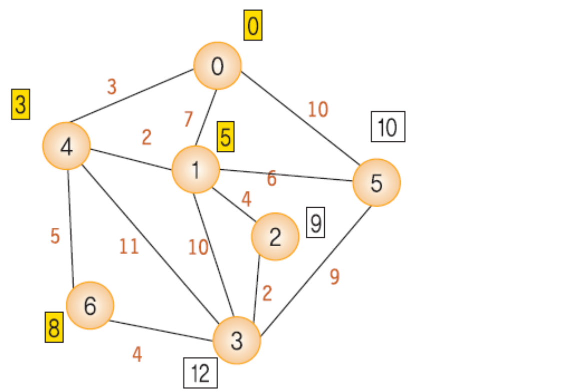
S={0, 4}

	0	1	2	3	4	5	6
distance[]=	0	5	∞	14	3	10	8



S={0, 4, 1}

	0	1	2	3	4	5	6
distance[]=	0	5	9	14	3	10	8

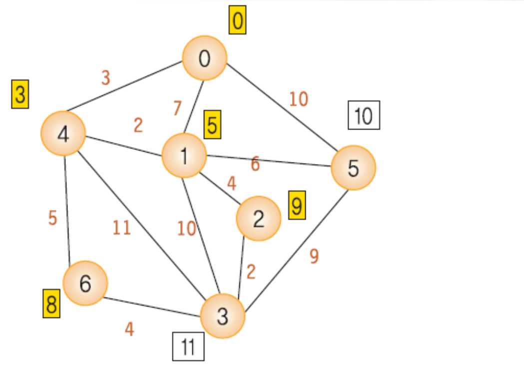


S={0, 4, 1, 6}

	0	1	2	3	4	5	6
distance[]=	0	5	9	12	3	10	8

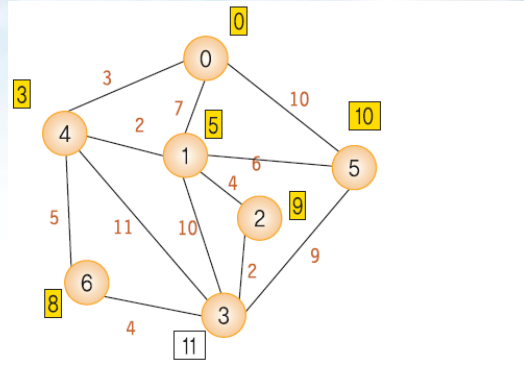
Dijkstra

- Example (cont'd)



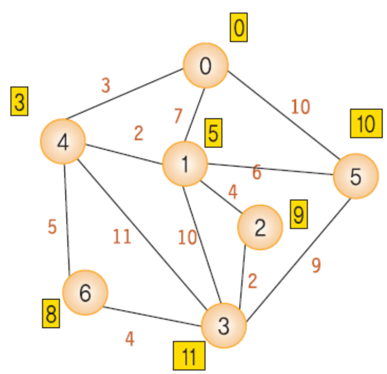
S={0, 4, 1, 6, 2}

	0	1	2	3	4	5	6
distance[]=	0	5	9	11	3	10	8



S={0, 4, 1, 6, 2, 5}

	0	1	2	3	4	5	6
distance[]=	0	5	9	11	3	10	8



S={0, 4, 1, 6, 2, 5, 3}

	0	1	2	3	4	5	6
distance[]=	0	5	9	11	3	10	8

What You Need to Know

Summary

- Greedy approach
  - Greedy algorithms make good local choices in the hope that they result in an optimal solution
- Greedy algorithms
  - Coin Change Problem
  - Activity Selection Problem
  - Minimum Spanning Tree (MST) Problems
    - Prim
    - Kruskal
  - Single-Source Shortest Path Problem

Thanks

Week 13: Greedy Algorithms
Instructor: Jinyoung Han (jinyounghan@skku.edu)