Greedy Algorithm

Algorithm

2014 Fall Semester

SUNG KYUN KWAN UNIVERSITY

Optimization problems

- An optimization problem is one in which you want to find, not just *a* solution, but the *best* solution
- A "greedy algorithm" sometimes works well for optimization problems
- A greedy algorithm works in phases. At each phase:
  - You take the best you can get right now, without regard for future consequences
  - You hope that by choosing a *local* optimum at each step, you will end up at a *global* optimum

Example: Counting money

- Suppose you want to count out a certain amount of money, using the fewest possible bills and coins
- A greedy algorithm would do this would be:
  At each step, take the largest possible bill or coin that does not overshoot
  - Example: To make $6.39, you can choose:
    - a $5 bill
    - a $1 bill, to make $6
    - a 25¢ coin, to make $6.25
    - A 10¢ coin, to make $6.35
    - four 1¢ coins, to make $6.39
- For US money, the greedy algorithm always gives the optimum solution

A failure of the greedy algorithm

- In some (fictional) monetary system, "krons" come in 1 kron, 7 kron, and 10 kron coins
- Using a greedy algorithm to count out 15 krons, you would get
  - A 10 kron piece
  - Five 1 kron pieces, for a total of 15 krons
  - This requires six coins
- A better solution would be to use two 7 kron pieces and one 1 kron piece
  - This only requires three coins
- The greedy algorithm results in a solution, but not in an optimal solution

Example : Making changes

The greedy algorithm used to give change.
Amount owed: 41 cents.

Subtract Quarter
41 - 25 = 16

Subtract Dime
16 - 10 = 6

Subtract Nickel
6 - 5 = 1

Subtract Penny
1 - 1 = 0

Huffman codes

- Any binary tree with edges labeled with 0's and 1's yields a prefix-free code of characters assigned to its leaves
- Optimal binary tree minimizing the expected (weighted average) length of a codeword can be constructed as follows

Huffman's algorithm

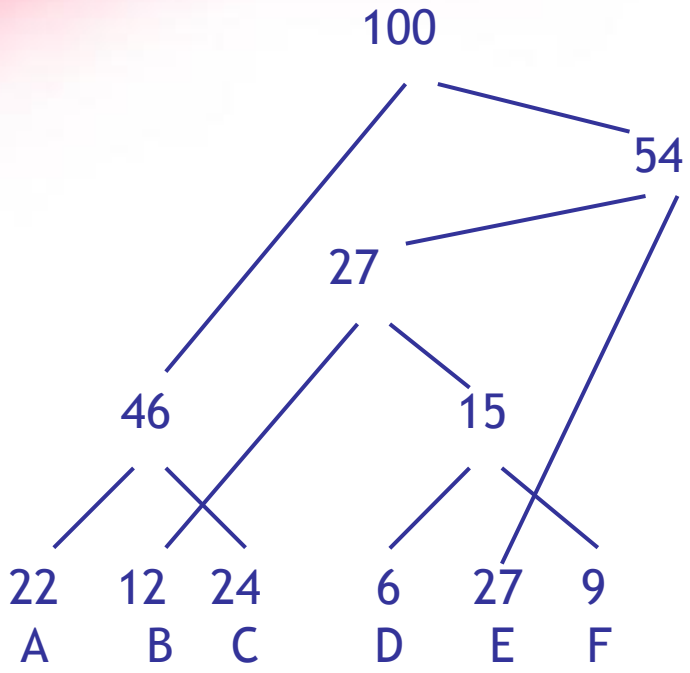
Initialize *n* one-node trees with alphabet characters and the tree weights with their frequencies.

Repeat the following step *n*-1 times: join two binary trees with smallest weights into one (as left and right subtrees) and make its weight equal the sum of the weights of the two trees.

Mark edges leading to left and right subtrees with 0's and 1's, respectively.

Huffman encoding

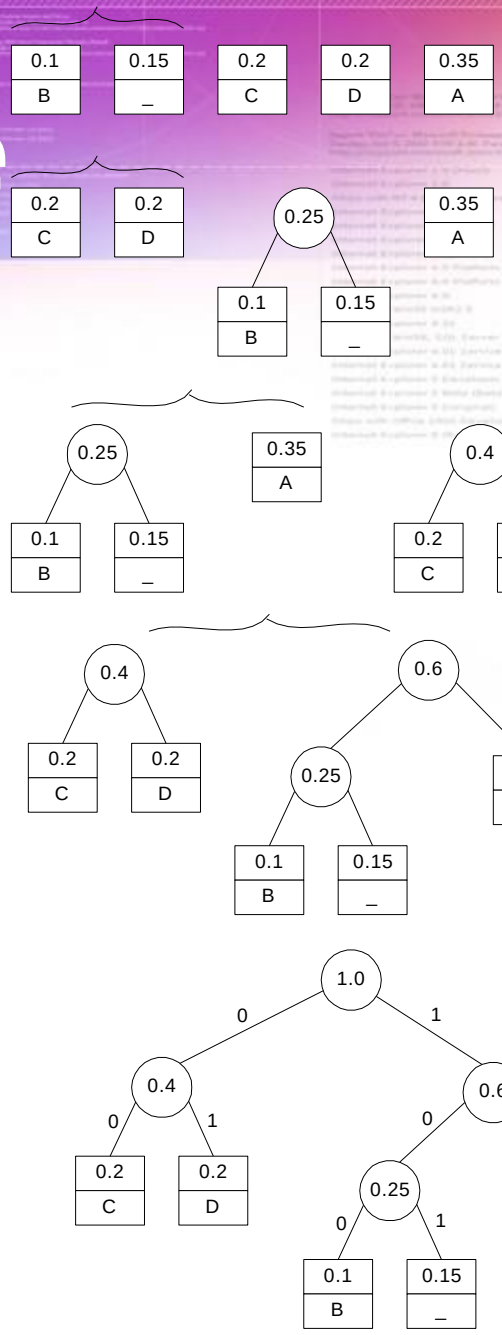
- The Huffman encoding algorithm is a greedy algorithm
- You always pick the two smallest numbers to combine



A=00
B=100
C=01
D=1010
E=11
F=1011

Example

character	A	B	C	D	_
frequency	0.35	0.1	0.2	0.2	0.15



codeword 11 100 00 01 101

Encode the text ADCB :

110100100

Decode the text whose encoding is

0001100101110001 :

CDB_ACD

In-Class Exercise

- Build a huffman tree for the following:

  A_x={ a , b , c , d , e }

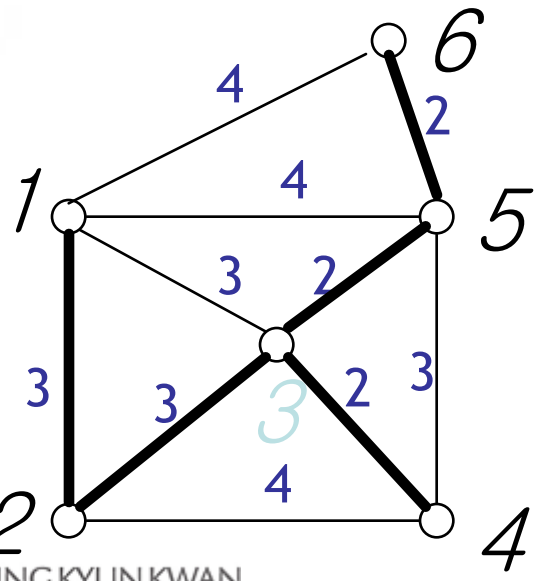
  P_x={0.25, 0.25, 0.2, 0.15, 0.15}

- Define the codeword for each alphabet.

- Encode the text - bbcddeabc

Minimum spanning tree

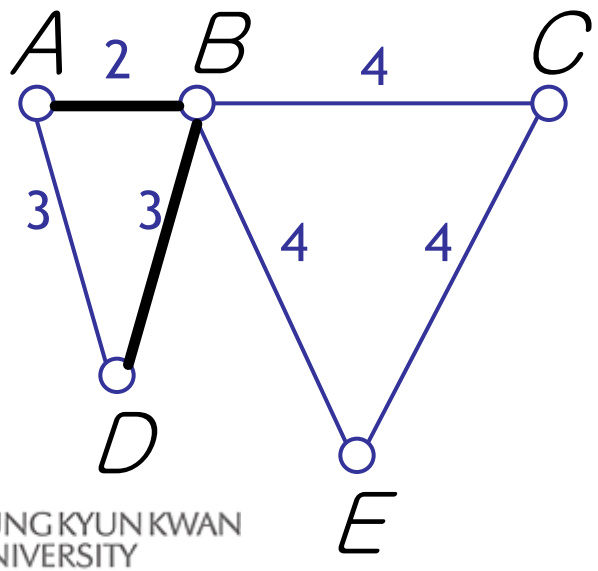
- A minimum spanning tree is a least-cost subset of the edges of a graph that connects all the nodes
  - Start by picking any node and adding it to the tree
  - Repeatedly: Pick any *least-cost* edge from a node in the tree to a node not in the tree, and add the edge and new node to the tree
  - Stop when all nodes have been added to the tree



- The result is a least-cost (3+3+2+2+2=12) spanning tree
- If you think some other edge should be in the spanning tree:
  - Try adding that edge
  - Note that the edge is part of a cycle
  - To break the cycle, you must remove the edge with the greatest cost
    - This will be the edge you just added

Traveling salesman

- A salesman must visit every city (starting from city *A*), and wants to cover the least possible distance
  - He can revisit a city (and reuse a road) if necessary
- He does this by using a greedy algorithm: He goes to the next nearest city from wherever he is

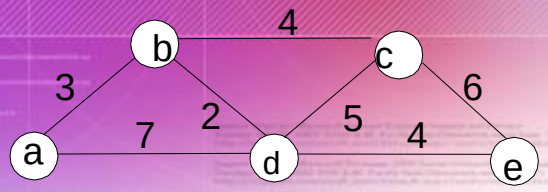


- From *A* he goes to *B*
- From *B* he goes to *D*
- This is *not* going to result in a shortest path!
- The best result he can get now will be *ABDBCE*, at a cost of 16
- An actual least-cost path from *A* is *ADBCE*, at a cost of 14

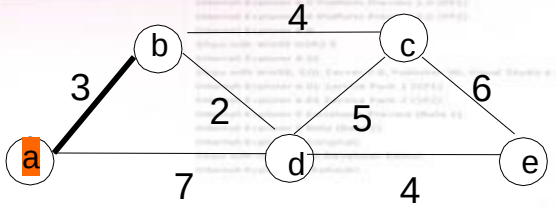
Other greedy algorithms

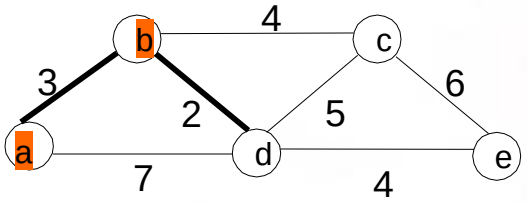
- Dijkstra's algorithm for finding **the shortest path** in a graph
  - Always takes the *shortest* edge connecting a known node to an unknown node
- Prim's algorithm for finding a **minimum-cost spanning tree**
  - Always takes the *lowest-cost* edge between nodes in the spanning tree and nodes not yet in the spanning tree
- Kruskal's algorithm for finding a **minimum-cost spanning tree**
  - Always tries the *lowest-cost* remaining edge

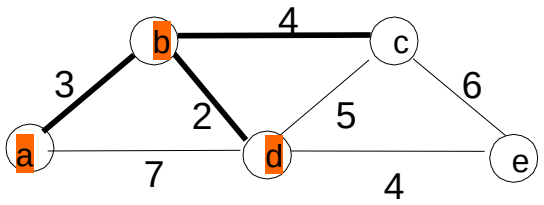
Dijkstra : Example

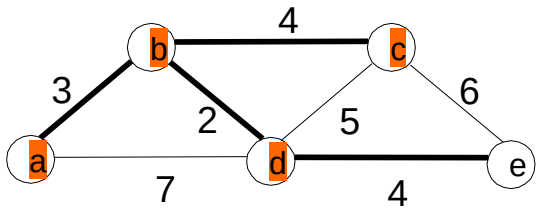


Tree vertices	Remaining vertices
a(−,0)	<u>b(a,3)</u> c(−,∞) d(a,7) e(−,∞)
b(a,3)	c(b,3+4) <u>d(b,3+2)</u> e(-,∞)
d(b,5)	<u>c(b,7)</u> e(d,5+4)
c(b,7)	<u>e(d,9)</u>
e(d,9)	



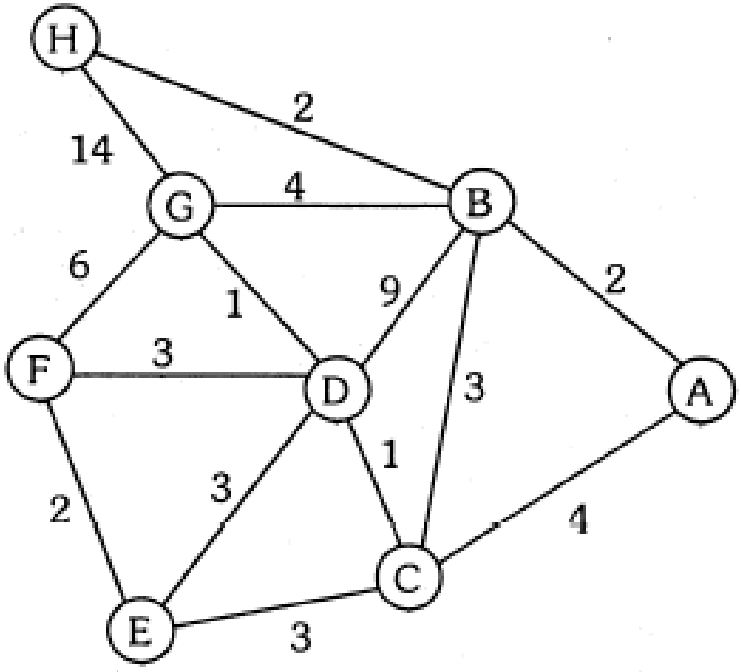




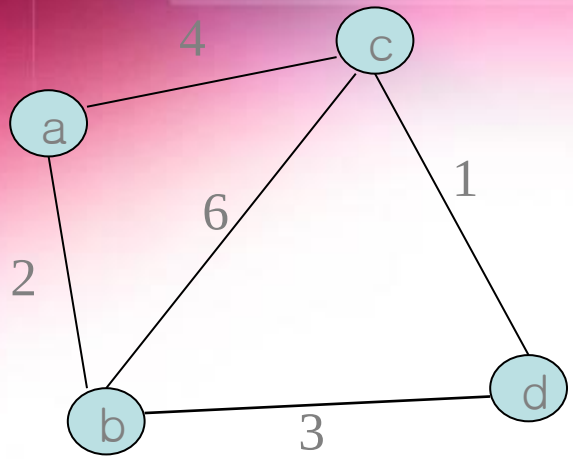


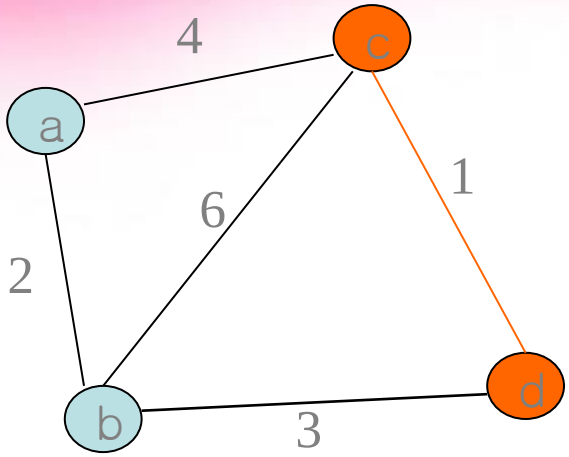
In-Class Exercise

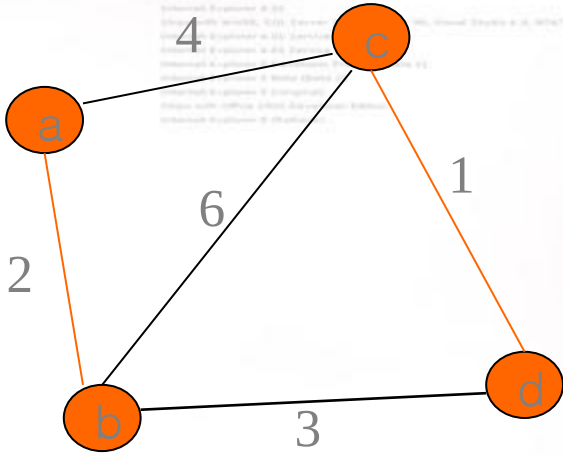
- Apply Dijkstra's algorithm to find the shortest path in the following graph.

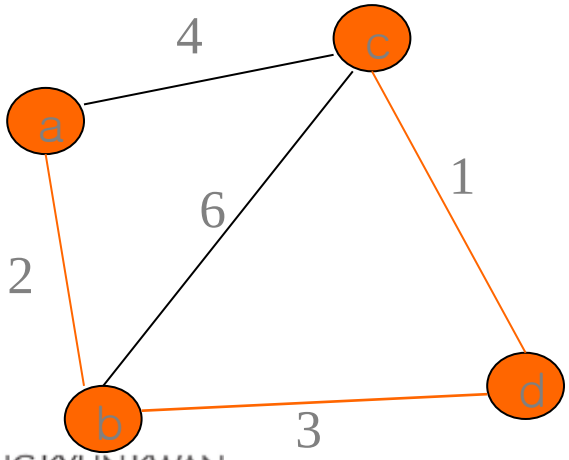


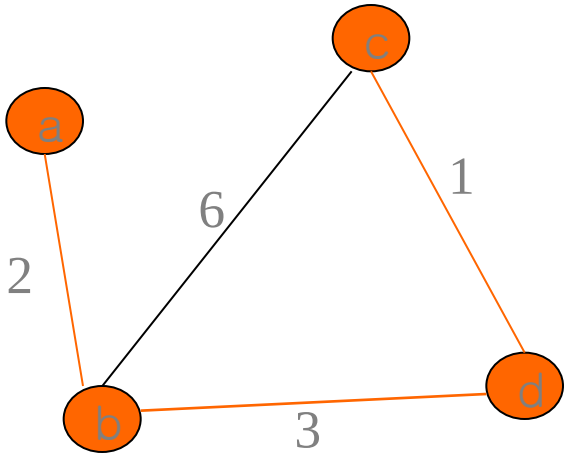
Kruskal's MST - Example

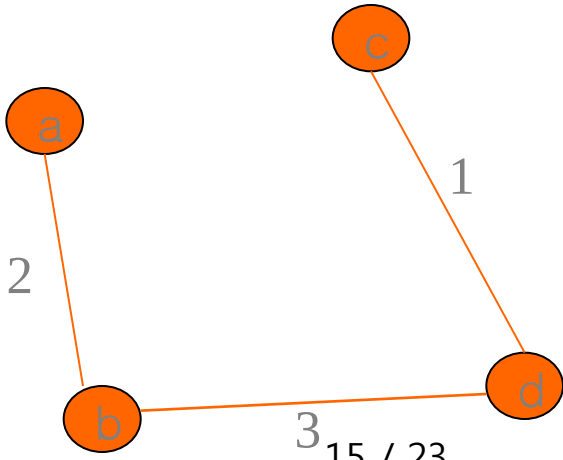






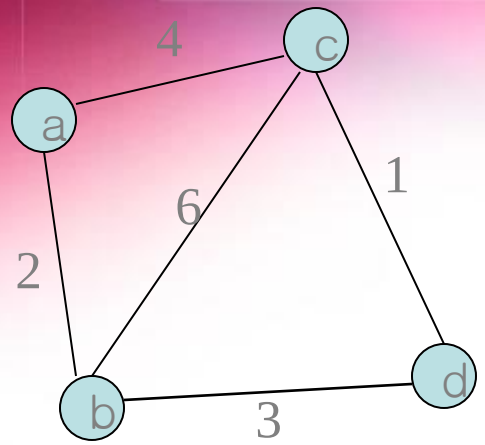


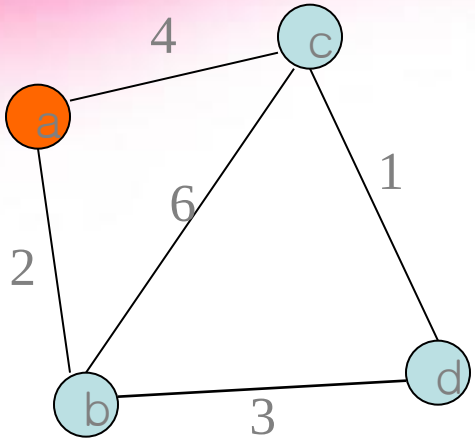


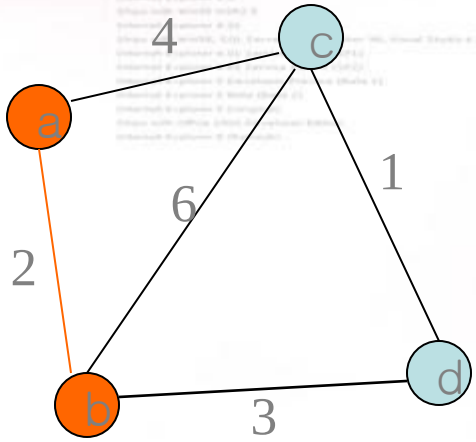


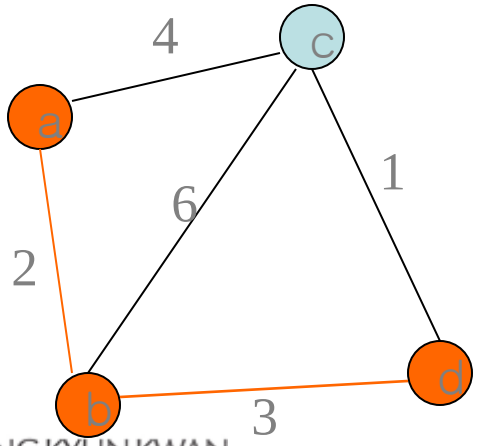
SUNG KYUN KWAN UNIVERSITY

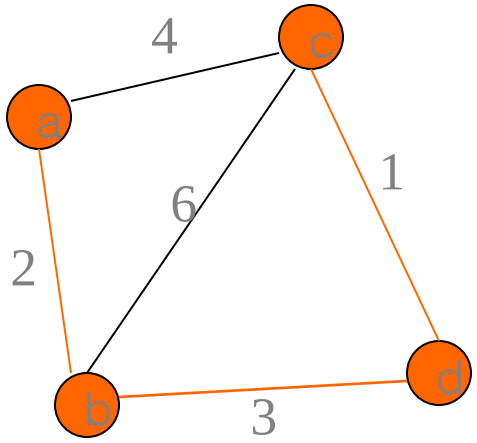
Prim's MST - Example





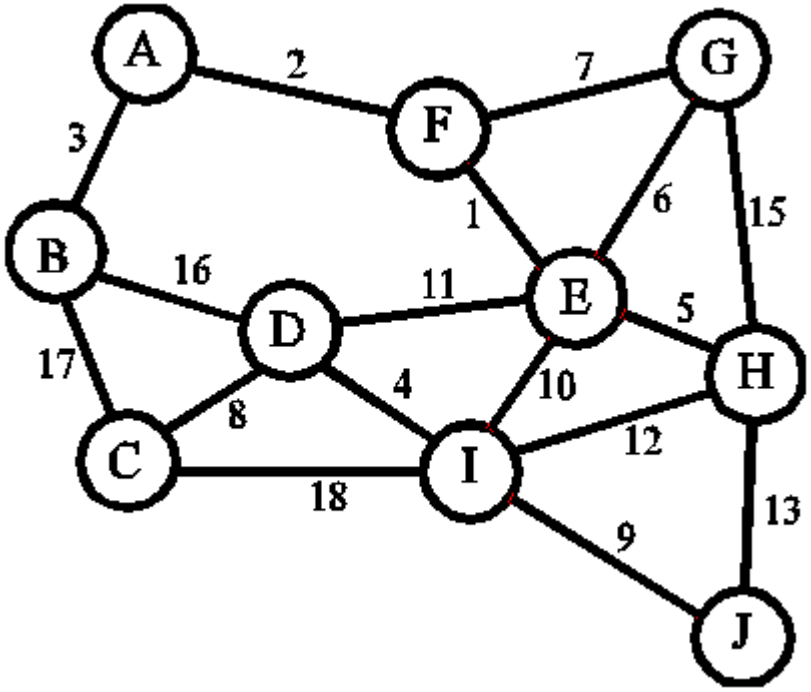






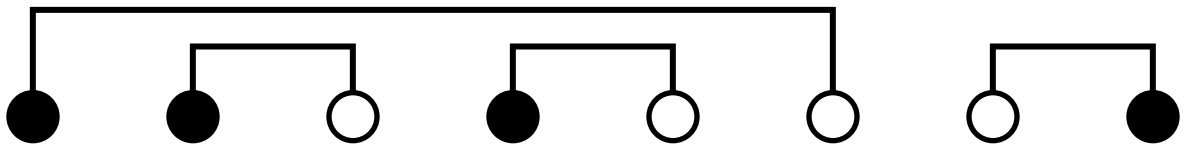
In-Class Exercise

- Find a minimum-cost spanning tree
  ① Kruskal
  ② Prim



Connecting wires

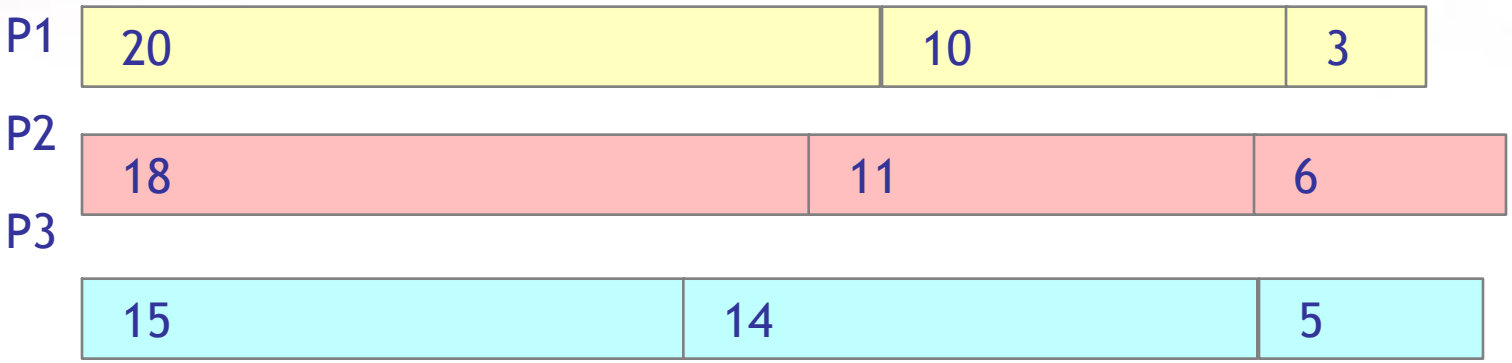
- There are n white dots and n black dots, equally spaced, in a line
- You want to connect each white dot with some one black dot, with a minimum total length of "wire"
- Example:



- Total wire length above is 1 + 1 + 1 + 5 = 8
- Do you see a greedy algorithm for doing this?
- Does the algorithm guarantee an optimal solution?
  - Can you prove it?
  - Can you find a counterexample?

A scheduling problem

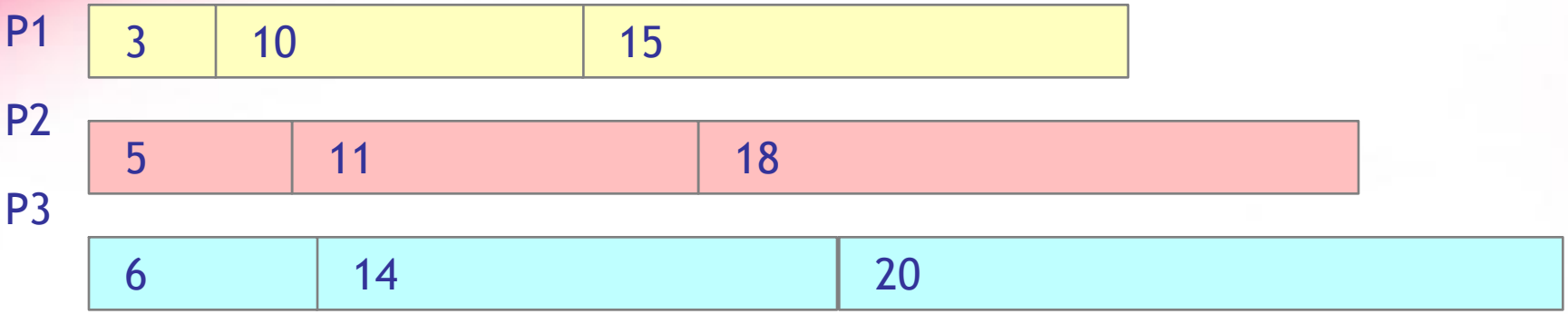
- You have to run nine jobs, with running times of 3, 5, 6, 10, 11, 14, 15, 18, and 20 minutes
- You have three processors on which you can run these jobs
- You decide to do the longest-running jobs first, on whatever processor is available



- Time to completion: 18 + 11 + 6 = 35 minutes
- This solution isn't bad, but we might be able to do better

Another approach

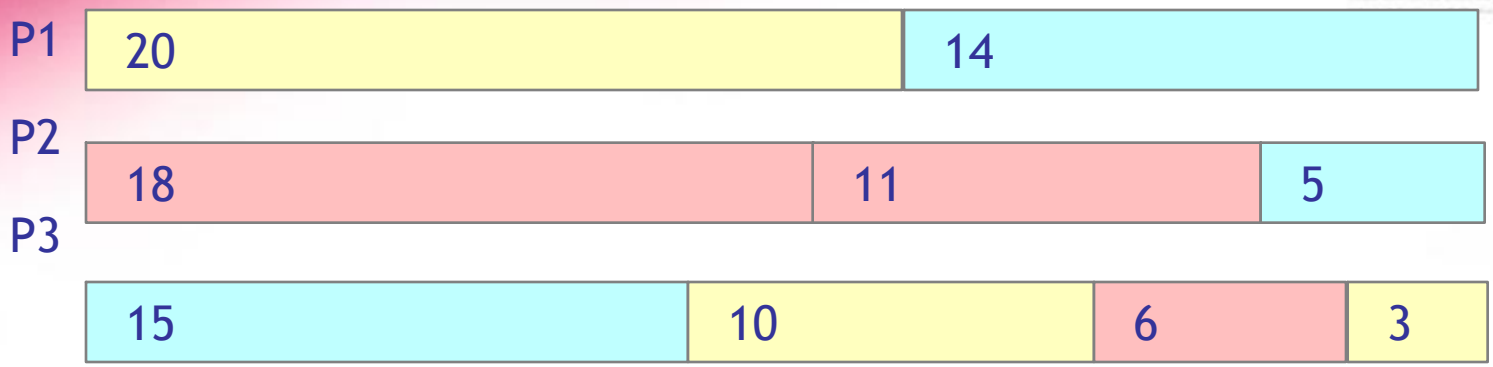
- What would be the result if you ran the *shortest* job first?
- Again, the running times are 3, 5, 6, 10, 11, 14, 15, 18, and 20 minutes



- That wasn't such a good idea; time to completion is now 6 + 14 + 20 = 40 minutes
- Note, however, that the greedy algorithm itself is fast
  - All we had to do at each stage was pick the minimum or maximum

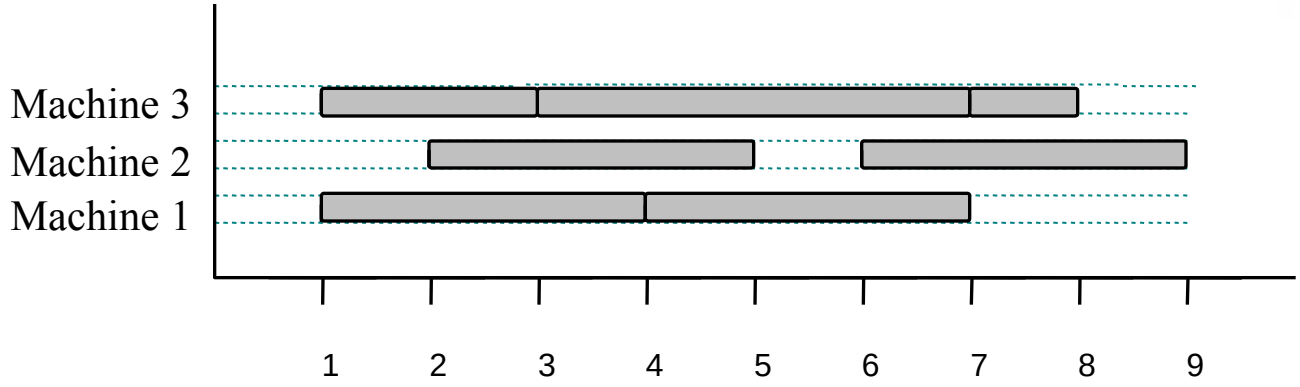
An optimum solution

- Better solutions do exist:



Task Scheduling

- Given: a set T of n tasks, each having:
  - A start time, s_i
  - A finish time, f_i (where $s_i < f_i$)
- Goal: Perform all the tasks using a minimum number of "machines."



Example

- Given: a set T of n tasks, each having:
  - A start time, s_i
  - A finish time, f_i (where $s_i < f_i$)
  - [1,4], [1,3], [2,5], [3,7], [4,7], [6,9], [7,8] (ordered by start)
- Goal: Perform all tasks on min. number of machines

Q & A

SUNG KYUN KWAN UNIVERSITY