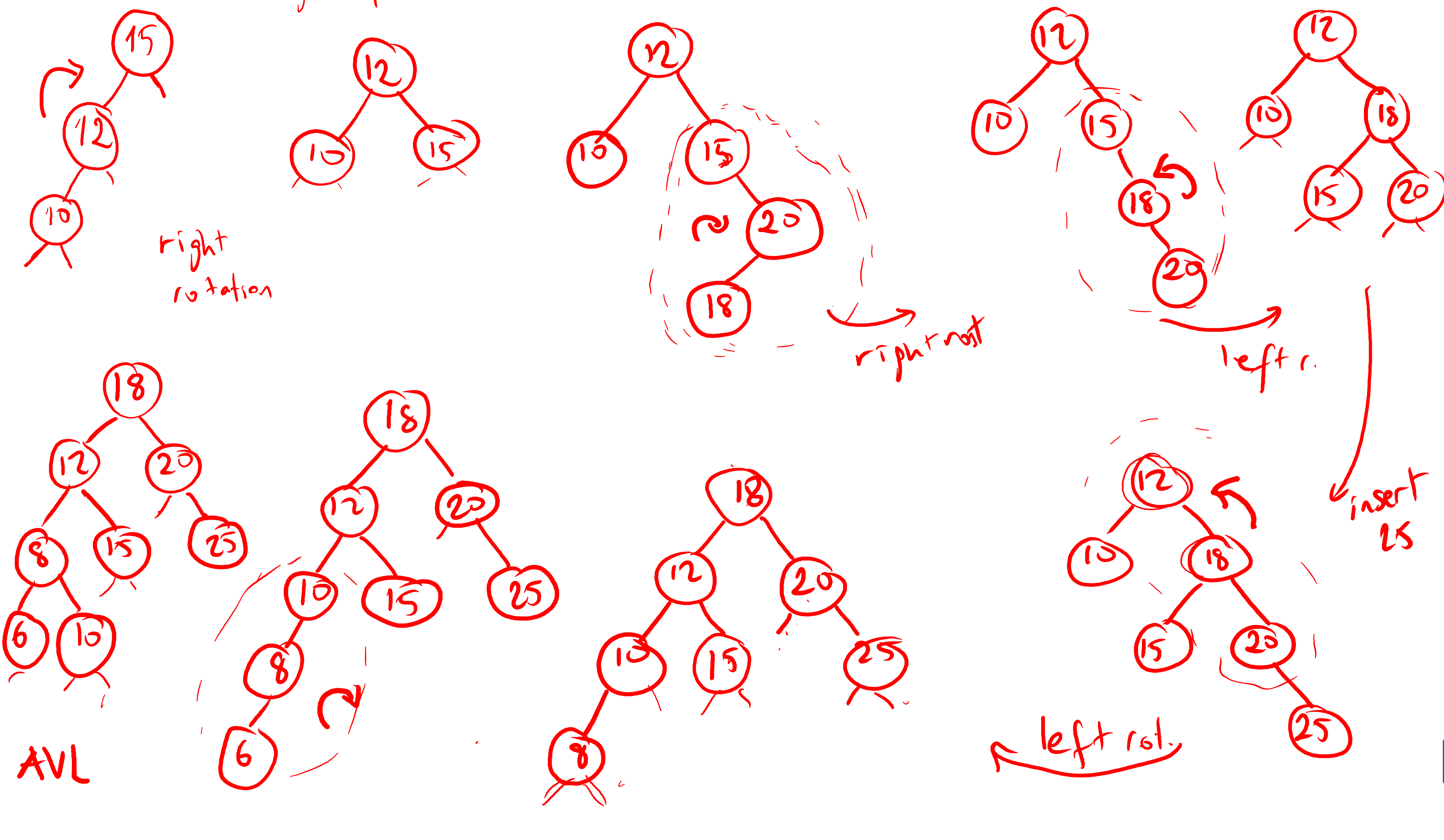
CENG 218 Spring 2023

Recitation 29.05.2023

Balanced Search Trees: AVL Tree

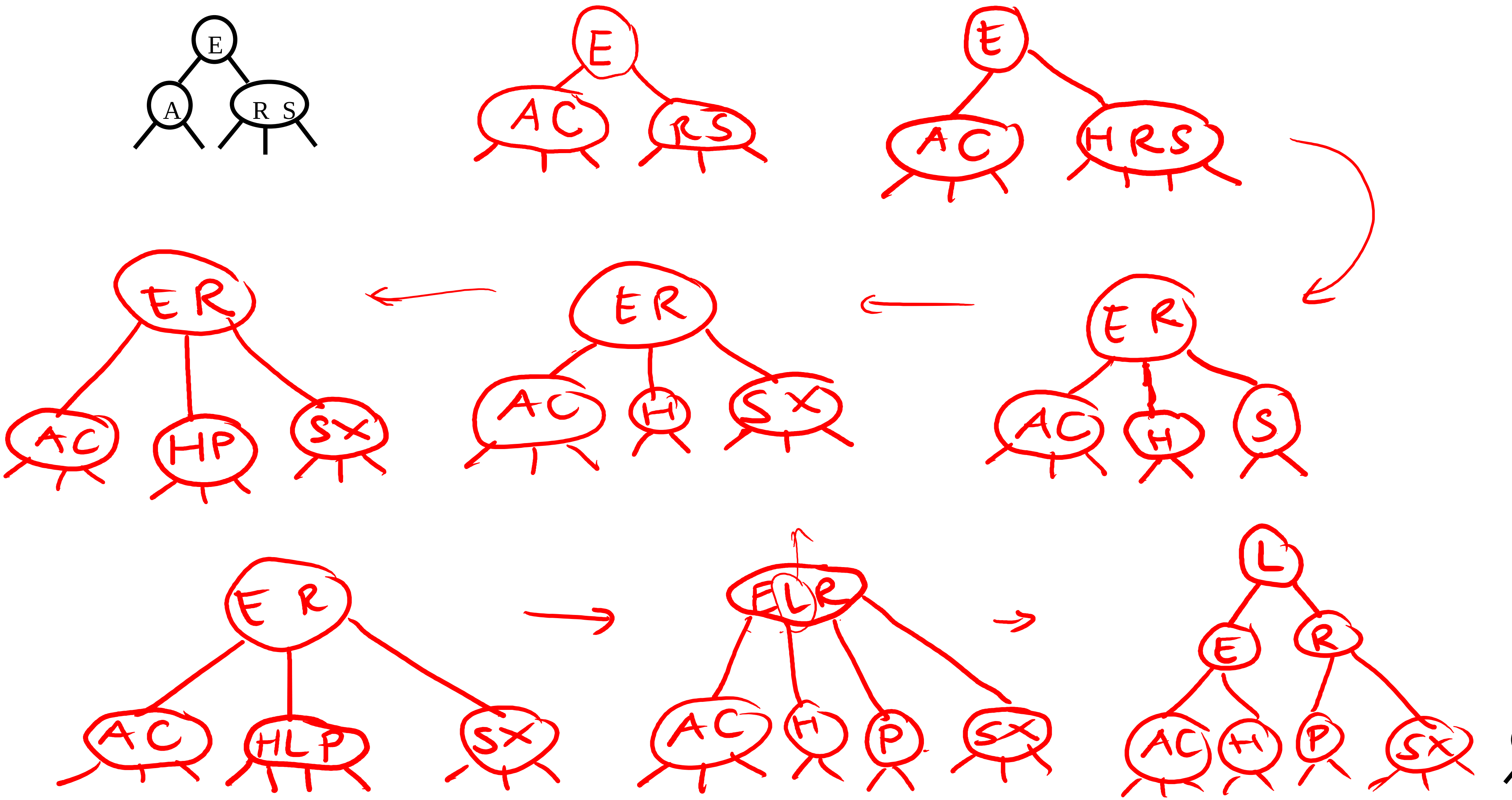
Build an AVL tree after successively inserting the keys 15, 12, 10, 20, 18, 25, 8, 6.

height dif <2



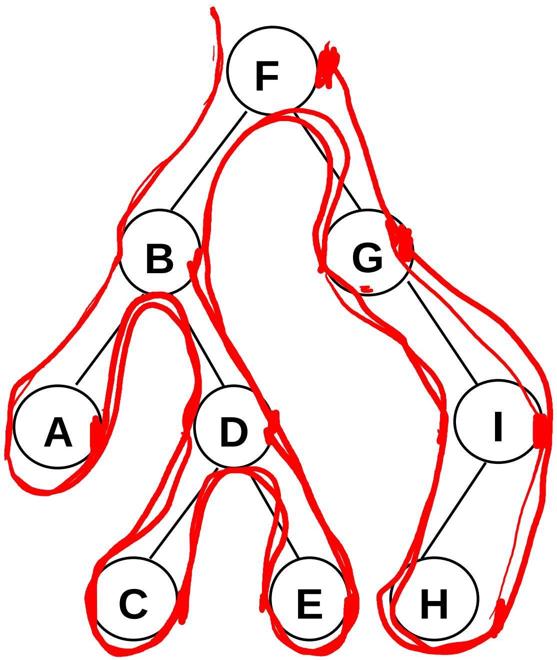
HW2.1 2-3 Tree

Below is a 2-3 tree, into which 4 elements were inserted so far. Please insert C,H,X,P and L in this tree in the given order.



Tree traversals

What are the in-order, pre-order and post-order tree traversals for the following tree:



In order: A B C D E F G H I

Preorder: F B A D C E G I H

Postorder: A C E D B H I G F

LCS

|LCS| = 6

Determine an LCS of x = (1; 0; 0; 1; 0; 1; 0; 1) and y = (0; 1; 0; 1; 1; 0; 1; 1; 0).

y \ x		1	0	0	1	0	1	0	1
	0	0	0	0	0	0	0	0	0
0	0	0	1	1	1	1	1	1	1
1	0	1	1	1	2	2	2	2	2
0	0	1	2	2	2	3	3	3	3
1	0	1	2	2	3	3	4	4	4
1	0	1	2	2	3	3	4	4	5
0	0	1	2	3	3	4	4	5	5
1	0	1	2	3	4	4	5	5	6
1	0	1	2	3	4	4	5	5	6
0	0	1	2	3	4	5	5	6	6

LCS = 01 01 01

0 0 1 0 1 0

HW2.2 Matrix-chain multiplication problem

We are given a matrix sequence (chain) $A_1, A_2, \ldots, A_n$ and we wish to compute the product $A_1A_2\ldots A_n$. Any order gives the same product but the order changes the amount of scalar multiplications we do.

Illustration: Consider the chain $\langle A_1, A_2, A_3 \rangle$. Dimensions of A_1, A_2 and A_3 are 10x100, 100x5, and 5x50 respectively. Multiplication of $((A_1A_2)A_3)$ takes 7500 scalar multiplications, whereas $(A_1(A_2A_3))$ takes 75000. The problem is finding the multiplication order that takes minimum amount of scalar multiplications.

a) What is the complexity of the brute-force algorithm? I.e. How much time does it take to try out all alternatives to find the best order?

b) Please refer to Section 15.2 in your textbook and shortly explain how this problem is solved with DP? Give the formulation.

c) Solve the example below using DP. Do not only write the answer but also show the table constructed with bottom-up approach.

matrix	A_1	A_2	A_3	A_4	A_5	A_6
dimension	30×35	35×15	15×5	5×10	10×20	20×25

d) What is the complexity of bottom-up dynamic programming algorithm?

HW2.2 continued

3 → 2
4 → 3!
5 → 4!

$\Omega(2^n)$

a) Brute-force algorithm:

$A_1 \; A_2 \; A_3$ 2!

Try out all parenthesis alternatives for n matrices is at least exponential time: $\Omega(2^n)$
Actually it has been shown that the number growing proportional to Catalan numbers which is also exponential but grows faster than 2^n.

$A_1 \; A_2 \; A_3 \; A_4$ 6

$A_1 \; A_2 \; A_3 \; A_4 \; A_5$ 24

Catalan sayıları

$A_1 \dots | \quad | \dots A_n$

b) $A_i \, A_{i+1} \, \dots \, A_j$ for $1 \le i \le j \le n$.

Let $m[i, j]$ be the minimum number of scalar multiplications to compute $A_{i\dots j}$

We can define $m[i, j]$ recursively as follows:

If $i=j$, the problem is trivial, no scalar multiplications.

Otherwise, $m[i, j]$ equals the minimum cost for computing subproducts $A_{i\dots k}$ and $A_{k+1\dots j}$, plus the cost of multiplying these two matrices.

$$m[i,j] = \begin{cases} 0 & \text{if } i = j \\ \min\limits_{i \le k < j} \{m[i,k] + m[k+1,j] + \underline{p_{i-1} p_k p_j}\} & \text{if } i < j \end{cases}$$

where matrix A_i is represented as a p_{i-1} x p_i size matrix.
Product $A_{i\dots k} \, A_{k+1\dots j}$ takes $p_{i-1} \, p_k \, p_j$ scalar multiplications.

A_1

5×10 $p_i = 10$

HW2.2 continued

$\Omega(2^n)$

c) How to compute optimal solution in bottom-up fashion?

Construct a table for:

matrix	A_1	A_2	A_3	A_4	A_5	A_6
dimension	30×35	35×15	15×5	5×10	10×20	20×25

E.g:

$$m[2,5] = \min \begin{cases} m[2,2] + m[3,5] + p_1 p_2 p_5 = 0 + 2500 + 35 \cdot 15 \cdot 20 = 13{,}000 , \\ m[2,3] + m[4,5] + p_1 p_3 p_5 = 2625 + 1000 + 35 \cdot 5 \cdot 20 = 7125 , \\ m[2,4] + m[5,5] + p_1 p_4 p_5 = 4375 + 0 + 35 \cdot 10 \cdot 20 = 11{,}375 \end{cases}$$
$$= 7125 .$$

m[1,70]

$O(n^3) = O(n^2) \cdot O(n)$

$O(n^2)$

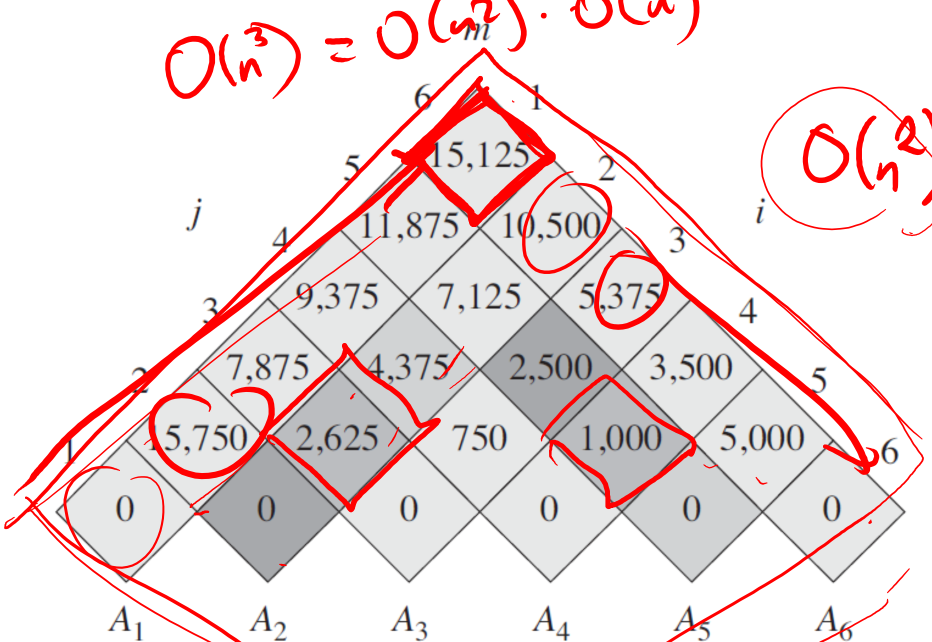
m[1,2] + m[3,3] + 30·15·5

15750   0   2250 = 18000

OR

m[1,1] + m[2,3] + 30·35·5

0   2625   5250 = 7875



Solution is the box of the largest interval, i.e. *m*[1,6]=15,125 scalar multiplications.

HW2.2 continued

d) What is the complexity of bottom-up dynamic programming algorithm?

Complexity is equal to *size of the table* x *time spent to fill each box*.

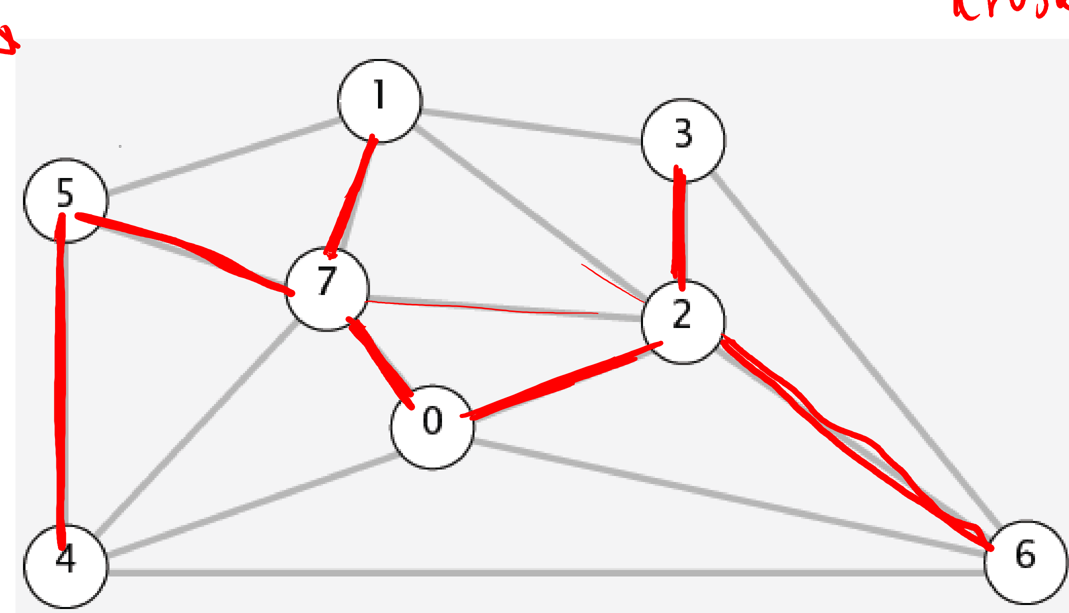
Table size is $n^2/2 = O(n^2)$. Each box is filled by trying out alternatives of where to cut the matrix chain. For boxes below it is $<n$, for the largest case it is n, so there are $O(n)$ alternatives. Total complexity = $O(n^2) \times O(n) = O(n^3)$.

MST

Choose an algorithm to find the MST in the following graph. Do not just tell the result, but also show the steps while applying the algorithm. If the algorithm you chose is ‘greedy’, then please indicate what is the ‘greedy choice’ in this algorithm

Kruskal

edge	weight	
✓ 0-7	0.16	1
✓ 0-2	0.26	4
~~0-4~~	0.38	12
~~1-2~~	0.36	10
✓ 1-7	0.19	3
~~1-3~~	0.29	6
~~1-5~~	0.32	7
✓ 2-3	0.17	2
~~2-7~~	0.34	8
~~3-6~~	0.52	14
✓ 4-5	0.35	9
~~4-7~~	0.37	11
✓ 5-7	→ 0.28	5
✓ 6-2	0.40	13
~~6-0~~	0.58	15
~~6-4~~	0.93	16



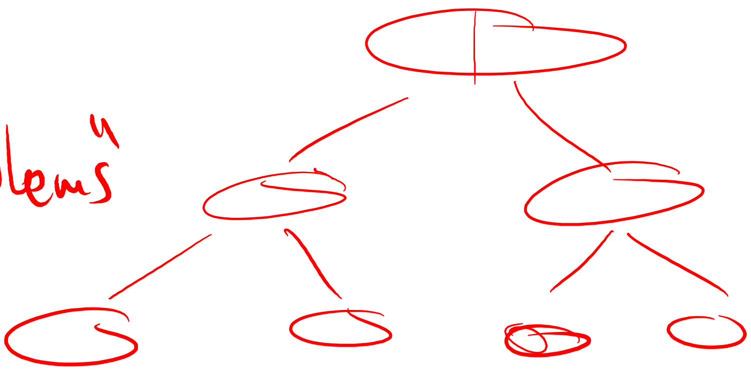
"MST"

HW2.3 Why not DP for Merge-sort?

In Merge sort we are recursively dividing the problem into two subproblems and solve them. Explain why a dynamic programming algorithm, using memoization for example, does not speed up a good divide-and-conquer approach such as merge-sort?

sublists are different

"No overlapping subproblems"



HW2.4 Greedy choice

Remember the activity selection problem: We have a set $S = \{a_1, a_2, \ldots, a_n\}$ of n proposed ***activities*** that wish to use a resource. Each activity a_i has a ***start time*** s_i and a ***finish time*** f_i .

Activities a_i and a_j are ***compatible*** if the intervals $[s_i, f_i)$ and $[s_j, f_j)$ do not overlap. We wish to select a maximum-size subset of compatible activities.

a_2 a_3 a_8

i	~~1~~	2	3	~~4~~	~~5~~	~~6~~	~~7~~	8
s_i	3	1	4	3	5	6	7	8
f_i	5	4	7	9	9	10	11	12

~~earliest finish time~~

a) Consider the following greedy approach: "Selecting the activity of least duration" from those that are compatible with previously selected activities". Does this approach provide an optimal solution? YES NO

$\{a_1, a_3\}$

b) Consider the following greedy approach: "Selecting the last activity to start". Does this approach provide an optimal solution? YES

Rod cutting with ~~greedy~~ algorithm?

DP

Remember the rod cutting problem, the following is an instance of it:

n=4

length i	1	2	3	4	5	6	7	8	9	10
price p_i	1	4	8	9	10	17	18	22	22	30
density	1	2	2.66	2.25	2	2.83	2.57	2.75	2.44	3

inch

dollars

Define the ***density*** of a rod of length *i* to be p_i/i, that is, value per inch. A greedy strategy for cutting a rod of length *n* cuts first the piece with the highest density. (first)

Show by a counterexample (at least for one *n*) that this greedy strategy does not always determine an optimal way to cut rods.

i=4 → i=3 → 8\$
i=4 → i=1 → 1\$
9\$

i=4 → i=4 → 9\$

i=4 → i=2 → 4\$
i=4 → i=2 → 4\$
8\$

greedy choice

counter example

n=8

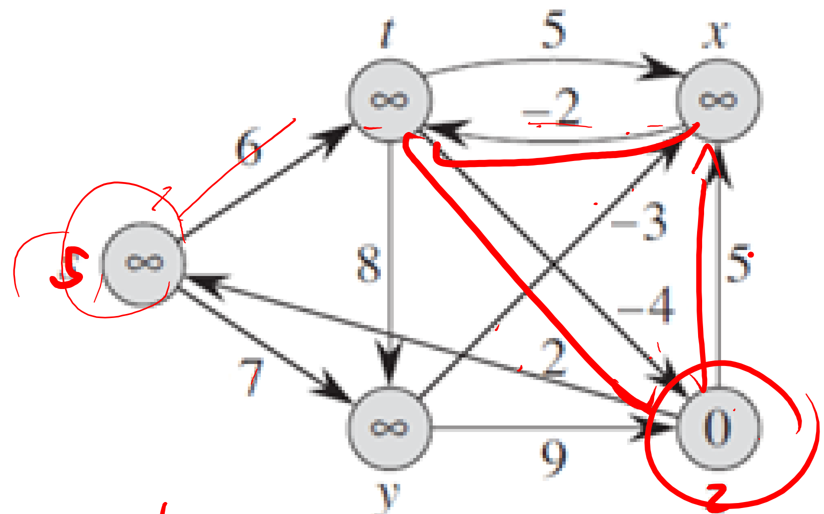
i=8 → i=6 → 17\$
i=8 → i=2 → 4\$
21\$

i=8 → i=8 → 22\$
22\$

Single-source shortest paths / Bellmann-Ford

Run the Bellman-Ford algorithm on the directed graph given below. Use vertex z as the source. Use the given table to write the d values after updates.

	z	x	y	s	t
	0	∞	∞	∞	∞
after round #1	-1	5	8	1	3
after round #2	-6	0	3	-4	-2
after round #3	-11	-5 -2	-2	-9	-7
after round #4	-16	-10	-7	-14	-12
#5	-21	-15	-12	-19	-17



directed

-2-4+5=-1

V. round

|V|-1=4 rounds have passed and there is still relaxations occur. Therefore, Bellmann-Ford reports that there is a negative cycle in this digraph.