COMP 251

Algorithms & Data Structures (Winter 2022)

Extras – Exam Review

School of Computer Science
McGill University

Slides of Langer (2014) & Cormen et al., 2009 & Comp251-Fall McGill & Kleinberg & *Tardos*, 2006 & Lin & Devi (UNC)

Announcements

Outline

- Extras.
    - Final Review

Final Exam

- To be released.
  - April 20th 2022 at 9:00 AM.
- Exam due on.
  - April 20th 2022 at 12:00 PM.

Timed exam – 3 hours

- The exam will cover all the lectures given during the term.

Final Exam

- The exam will be in crowdmark.
  - A 3-hour exam.
  - You will have 180 minutes to solve it.
      - Under exceptional circumstances, you will be able to email us your solutions.
  - How will you submit your answers?
    - Answering True/False questions.
    - Answering multiple choice questions.
      - Click the correct response(s).
    - Answering short questions.
      - Type the information in the box.
        - Evaluating (for one question) if it is possible also to upload a file.

Final Exam

- The exam will be in crowdmark.
  - Make sure your answers are clear and readable
  - Make sure you press the submit button!
  - Allowed material.
    - Everything that is posted in mycourses.
    - The two reference books.
  - Not allowed material (examples).
    - google
    - stackoverflow and other Q&A sites
    - chegg and comparable websites
    - talking to your friends about the exam
    - talking with someone during the exam

Final Exam

- Type of questions.
    - True/False Questions.
        - Given a claim, check if the claim is True or False.
        - Miscellaneous of topics.
    - Multiple Choice Questions.
        - Running of algorithms (what will be the next step/output).
        - Which of the following propositions is/are valid.
        - Given a code, select a good tight big-O.
        - Solving recurrences.
        - Miscellaneous of topics.

Final Exam

- Type of questions.
    - Short Answers – Graph Theory (theory-practice).
        - Given a graph.
            - Run an algorithm on that graph.
            - Answer theoretical questions about that graph.
        - Given a claim.
            - Decide if the claim is True or False.
                - If it is True, justify your answer.
                - If it is False, give a counter example.
    - Algorithms and Data Structures (real-world).
        - Given a scenario, propose a DS, algorithm or paradigm to solve it.
            - Justify, analyse, defend your answer.
        - Given a scenario, propose an algorithm, recurrence, sub-problem definition etc.
        - Give the answer for a small instance of the problem.

Final Exam

- Type of questions.
    - Short Answers - Algorithms and Data Structures (real-world).
        - Four friends (I guess you know who they are) meet to study and prepare the final exam. Given “real-world” scenarios.
            - Analyze the complexity of a solution.
            - Analyze/Complete the recurrence of a solution.
            - Analyze/Complete the (pseudo)code of a solution.
            - Propose a solution.
            - Give the answer for a small instance of the problem.

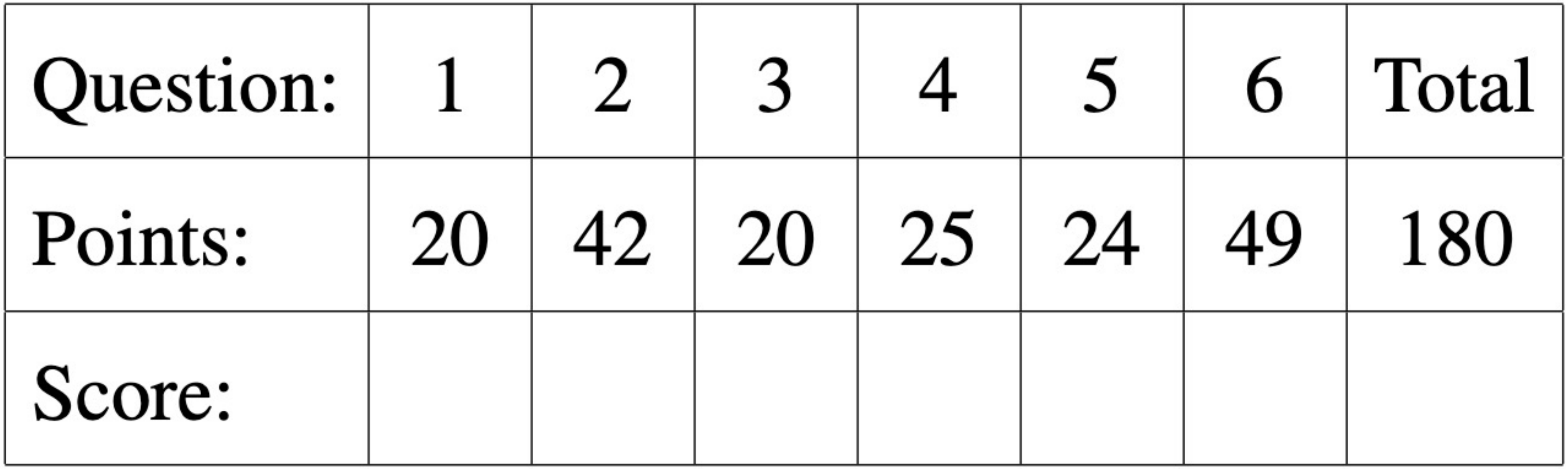
Final Exam

- How can I get help and clarifications during the exam?
  - Please post your questions in the discussion board.
    - Similar dynamic as the one of the midterm.
    - David and one T.A will be there answering clarifications.

Final Exam

- What is in my mind?
  - 3-hour exam.
    - I created the exam thinking on that.
    - T.As answered the exam for feedback.
      - Is it Clear? Fun? Challenging?
      - Taking times (Long? Short?).
  - Each question cannot be a simple look-up for the answer.
  - Questions must be designed to make the student think through a problem.
  - Students must be able to analyze and produce working solutions.
  - Diversify as much as possible the type of questions and topics.
    - Emphasis in Graph Theory.

Final Exam



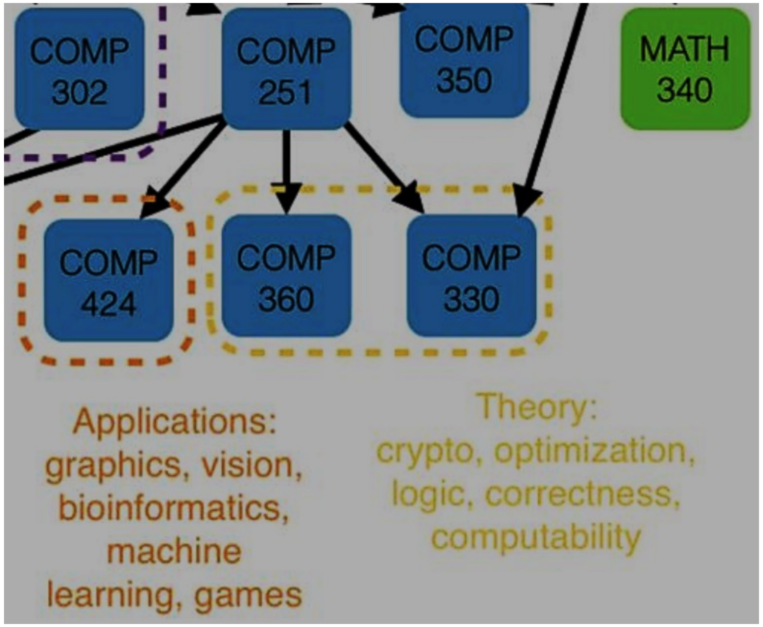
	True – False (10 Q)	Multiple-Choice (8 Q)	Short-Graph (4 Q)	Short-Graph (4 Q)	Short-Claim (3 Q)	Short-Paradigms (7 Q)	
Question:	1	2	3	4	5	6	Total
Points:	20	42	20	25	24	49	180
Score:							

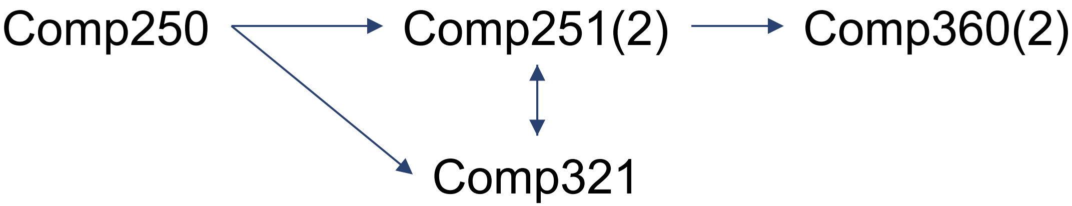
Final Exam - Review

L0 - Goals

The description of various computational problems and the algorithms that can be used to solve them, along with their associated data structures. Proving the correctness of algorithms and determining their computational complexity.

- By the end of this course, students will be able to:
  - Analyze the correctness of an algorithm.
  - Analyze the efficiency of an algorithm.
  - Know algorithms in the state of the art.
  - Design correct and efficient algorithms.
  - Implement the designed algorithms.


COMP 302
COMP 251
COMP 350
MATH 340
COMP 424
COMP 360
COMP 330
Applications: graphics, vision, bioinformatics, machine learning, games
Theory: crypto, optimization, logic, correctness, computability



Comp250

Comp251(2)

Comp360(2)

Comp321

L0 – Course Outline

- Data Structures
  - Hashing
  - Disjoint Sets
  - Heaps
  - Binary Search Trees
  - Balanced Trees => AVL and Red Black trees
- Design & Analysis
  - Complete Search
  - Divide and Conquer
  - Dynamic Programming
  - Greedy Algorithms
- Graph Algorithms
  - Traversal (BFS & DFS)
  - Network Flow
  - Connected Components
  - Minimum Spanning Tree
  - Single Source Shortest Path
  - Bipartite Graphs
- Extras
  - Amortized Analysis
  - Randomized Algorithms
  - Probabilistic Analysis

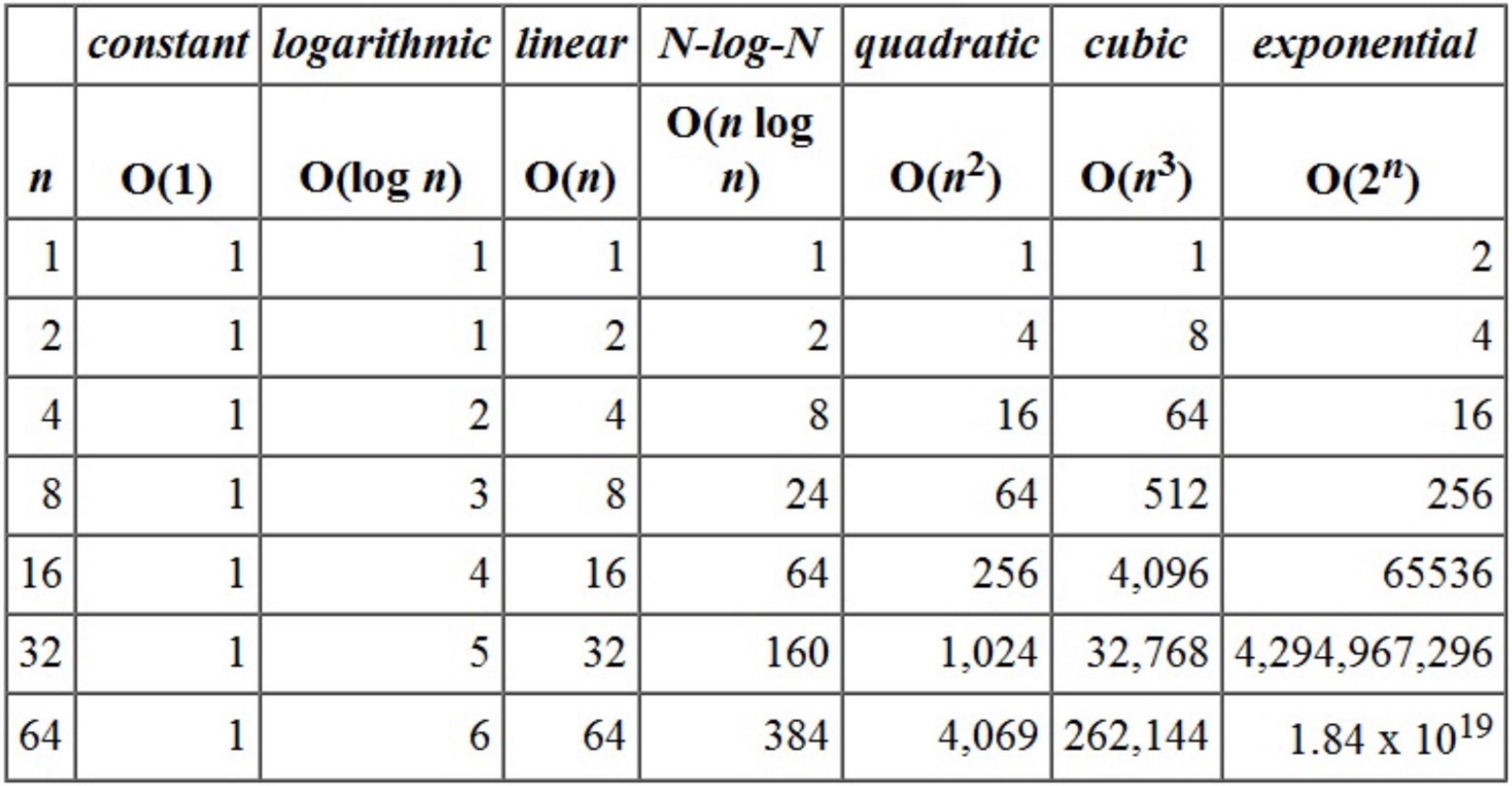
L1 – What should you already know?

- Linear Data Structures (ordering the elements sequentially).
  - Array lists, singly and doubly linked lists, stacks, queues.
- Induction and Recursion
- Tools for analysis of algorithms.
  - Recurrences.
  - Asymptotic notation (big O, big Omega, big Theta)
- Basics in non-linear data structures.
  - Trees, heaps, maps, graphs.

L1 – Recursion

- Every recursive algorithm involves at least 2 cases:
  - → **base case**: A simple occurrence that can be answered directly. 1 or more
  - → **recursive case**: A more complex occurrence of the problem that cannot be directly answered but can instead be described in terms of smaller occurrences of the same problem.
- Some recursive algorithms have more than one base or recursive case, but all have at least one of each.
- **A crucial part of recursive programming is identifying these cases.**

L1 – Growth of functions



	constant	*logarithmic*	*linear*	*N-log-N*	*quadratic*	*cubic*	*exponential*
n	O(1)	O(log n)	O(n)	O(n log n)	O(n^2)	O(n^3)	O(2^n)
1	1	1	1	1	1	1	2
2	1	1	2	2	4	8	4
4	1	2	4	8	16	64	16
8	1	3	8	24	64	512	256
16	1	4	16	64	256	4,096	65536
32	1	5	32	160	1,024	32,768	4,294,967,296
64	1	6	64	384	4,069	262,144	1.84×10^{19}

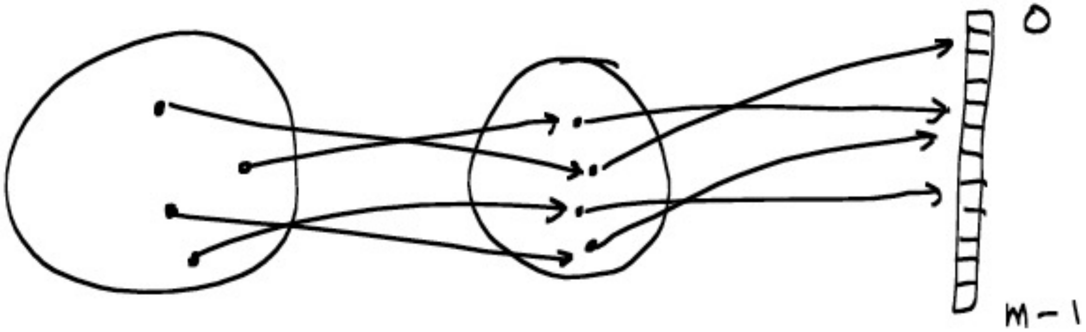
If the unit is in seconds, this would make ~10^{11} years…

L2 – Hashing

hash coding   Compression

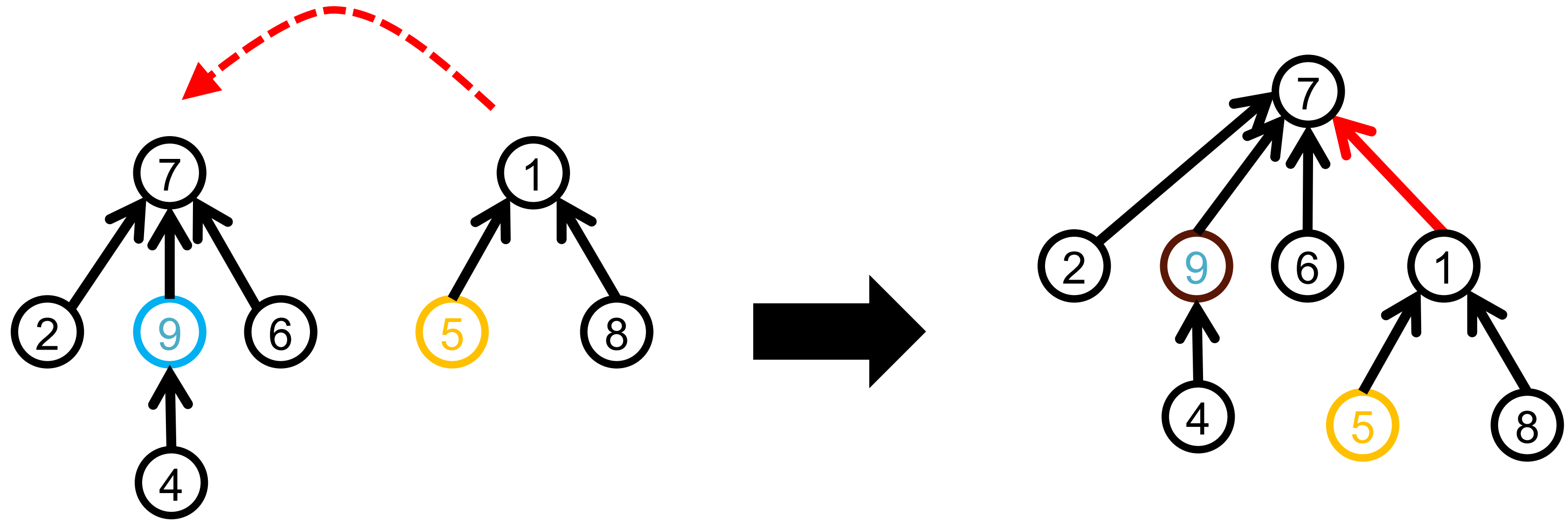
$$h : \mathcal{U} \to \{\text{integers}\} \to \{0, 1, \ldots m-1\}$$

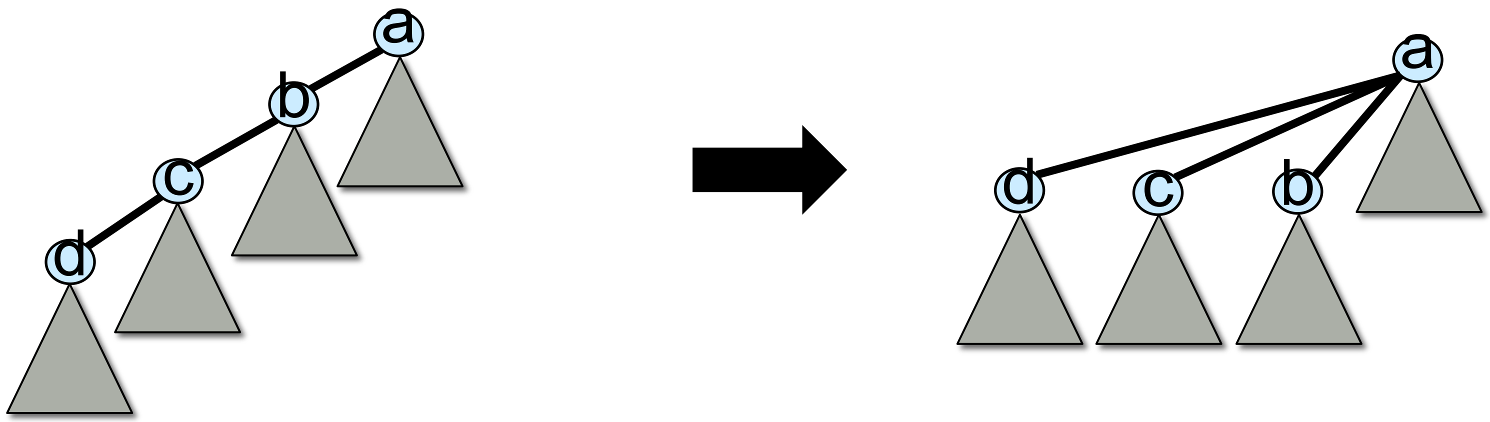
universe of keys   "hash codes"   "hash values"



- Hash functions.
  - Division method.
  - Multiplication method.
- Collision resolution.
  - Chaining method.
  - Open addressing.
    - Linear
    - Quadratic
    - Double Hashing

L3 – Union & Find

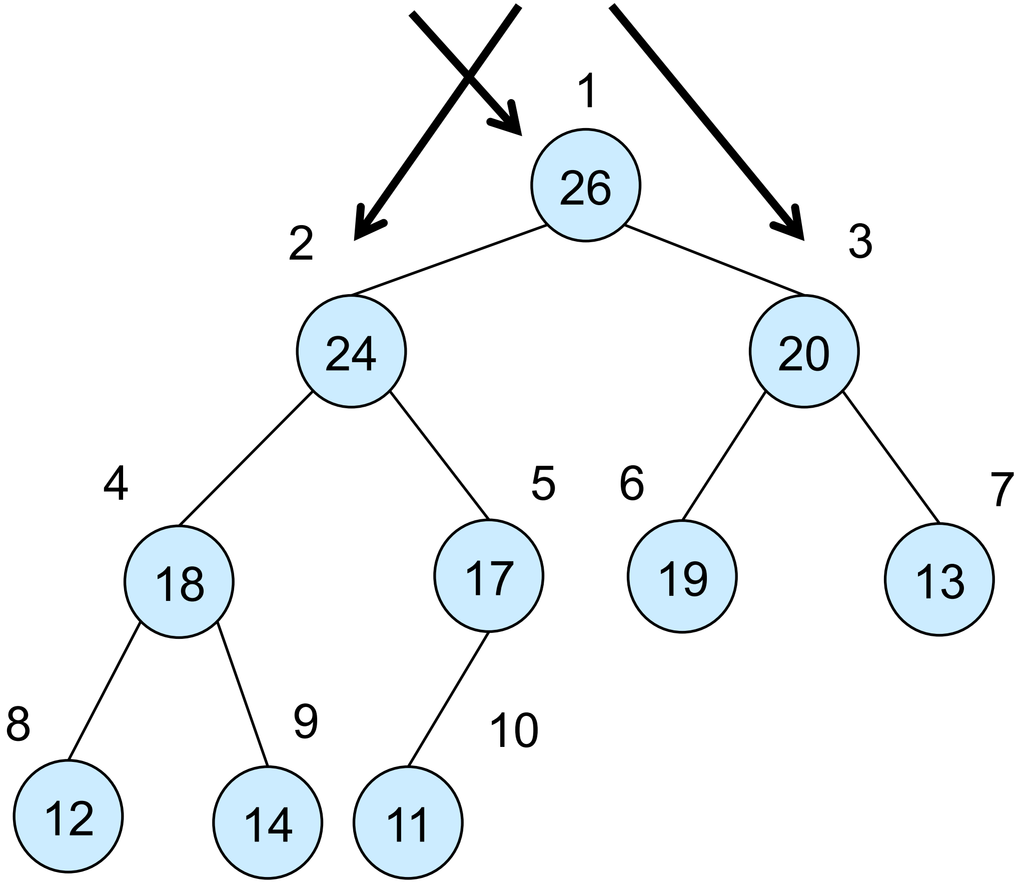




L4 - Heaps

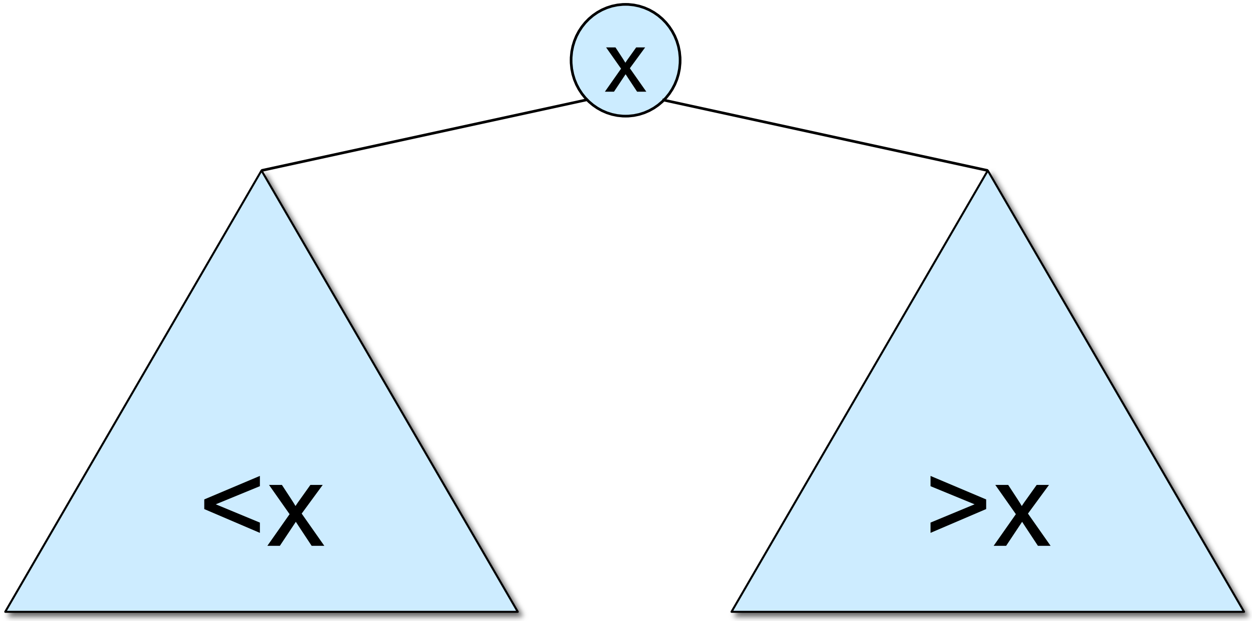
26	24	20	18	17	19	13	12	14	11
1	2	3	4	5	6	7	8	9	10

Max-heap as an array.



- Largest element is stored at the root.
- for all nodes *i*, excluding the root, *A*[PARENT(*i*)] ≥ *A*[*i*].
- Last row filled from left to right.
- Operations & Applications.
    - MaxHeapify
    - BuildMaxHeap
    - HeapSort

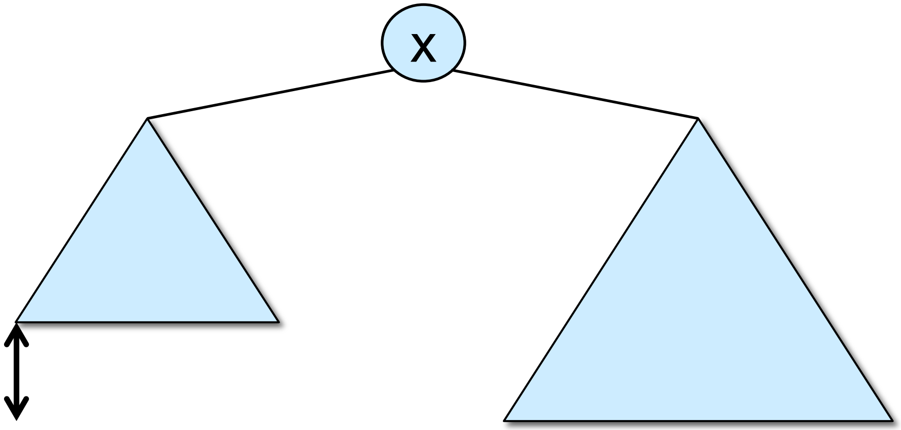
L5 – Binary Search Trees



- T is a rooted binary tree
- Key of a node x > keys in its left subtree.
- Key of a node x < keys in its right subtree.

L5 - AVL

Definition: BST such that the heights of the two child subtrees of any node differ by at most one.

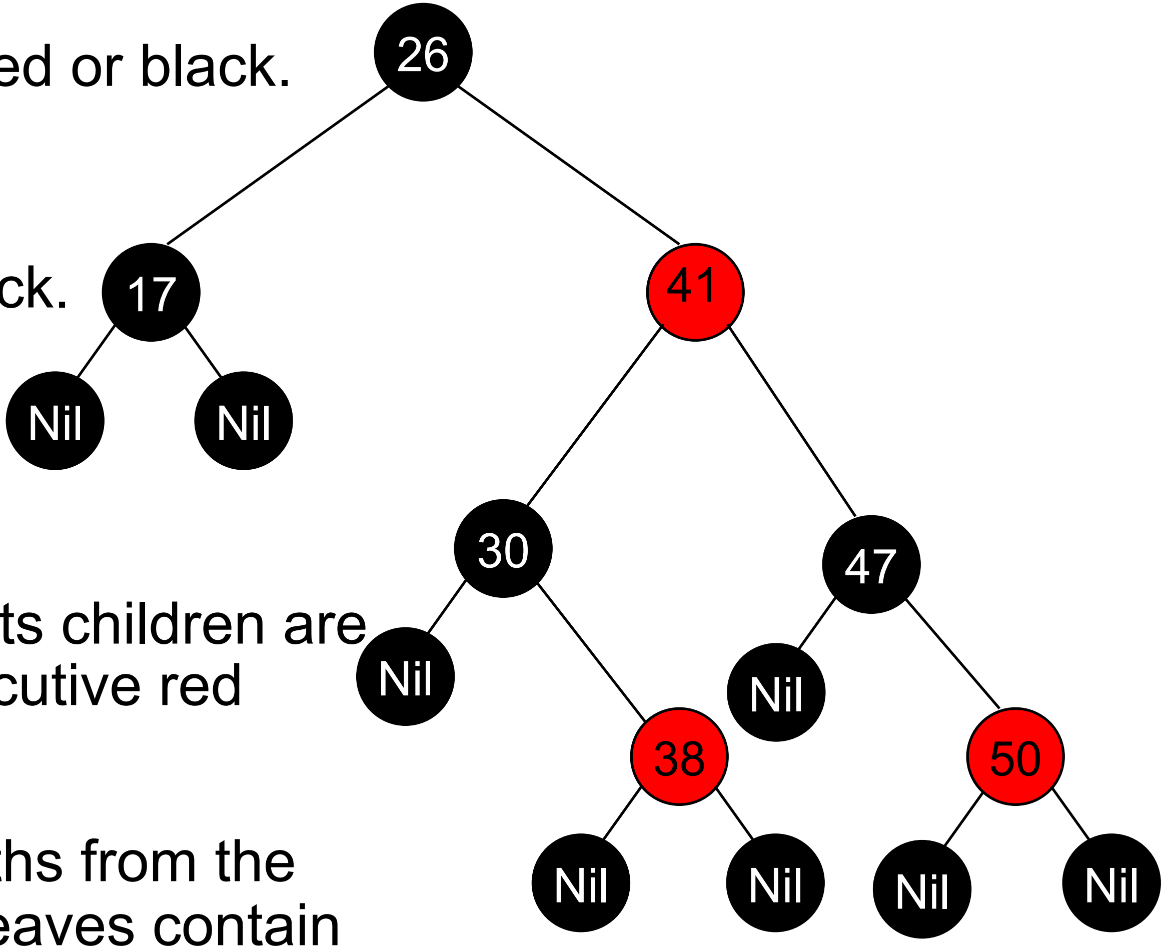


$|h_{left} - h_{right}| \leq 1$

- AVL trees are self-balanced binary search trees.
- Insert, Delete & Search take O(log n) in average and worst cases.
  - Rotations.

L6 – Red-Black trees

1. Every node is either red or black.
2. The root is black.
3. All leaves (*nil*) are black.



4. If a node is red, then its children are black (i.e. no 2 consecutive red nodes).
5. For each node, all paths from the node to descendant leaves contain the same number of black nodes (i.e. same black height).

L7 – Complete Search

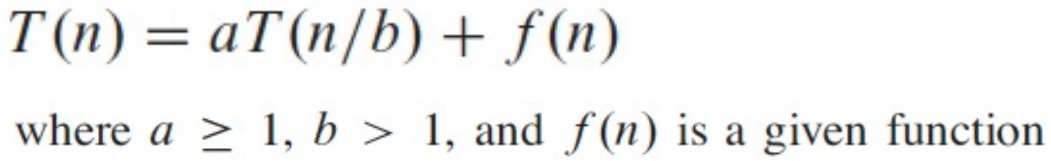
- Also know (related) as recursive backtracking or brute force.
  - Backtracking is a sort of refined brute force
- It is a method for solving a problem by searching (up to) the entire search space to obtain the required solution.
  - The search space can be large but finite.
  - Does not exist an algorithm that uses a method other than exhaustive search.
    - Need for developing techniques of searching, with the hope of cutting down the search space to possibly a much smaller space. Backtracking can be described as an organized exhaustive search which often avoids searching all possibilities.

L8-L9 – Divide and Conquer

- It is a problem solving paradigm where we try to make a problem simpler by ‘dividing’ it into smaller parts and ‘conquering’ them.
- Recursive in structure
  - ***Divide*** the problem into sub-problems that are similar to the original but smaller in size
    - Usually by half or nearly half.
  - ***Conquer*** the sub-problems by solving them recursively. If they are small enough, just solve them in a straightforward manner.
  - ***Combine*** the solutions to create a solution to the original problem

L8-L9 – Solving Recurrences

- ***Substitution method***: we guess a bound and then use mathematical induction to prove that our guess is correct.
- ***Recursion-tree method:*** converts the recurrence into a tree whose nodes represent the costs incurred at various levels of the recursion. We use techniques for bounding summations to solve the recurrence.
- ***Master method***: provides bounds for recurrences of the form.

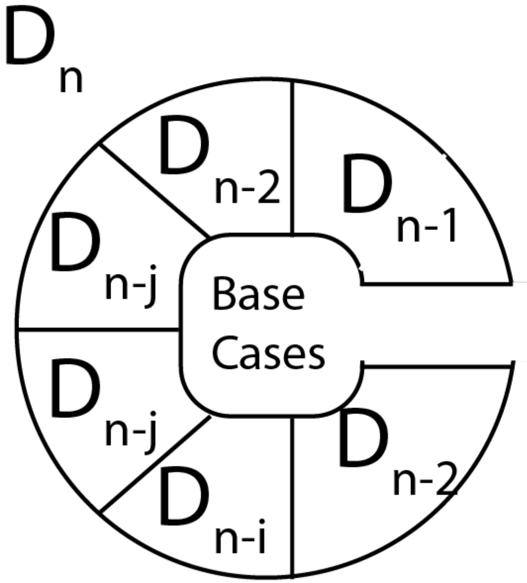


$$T(n) = aT(n/b) + f(n)$$

where $a \geq 1$, $b > 1$, and $f(n)$ is a given function

L10-L11 – Dynamic Programming

Paradigm



Overlapping subproblems

AND

Optimal substructure

+

Solution

↓ Memoization
Top-Down Approach

OR

↑ Tabulation
Bottom-up Approach

Dynamic programming is *not* about filling in tables.
It's about smart recursion!

L12-L13 - Greedy

No general way to tell if a greedy algorithm is optimal, but two key ingredients are:

- Greedy-choice Property.
  - We can build a globally optimal solution by making a locally optimal (greedy) choice.
- Optimal Substructure.
- Hackerearth "..If we make a choice that seems best at the moment and solve the remaining subproblems later, we still reach optimal solution. We never have to reconsider our previous choices".
- Data Compression application.

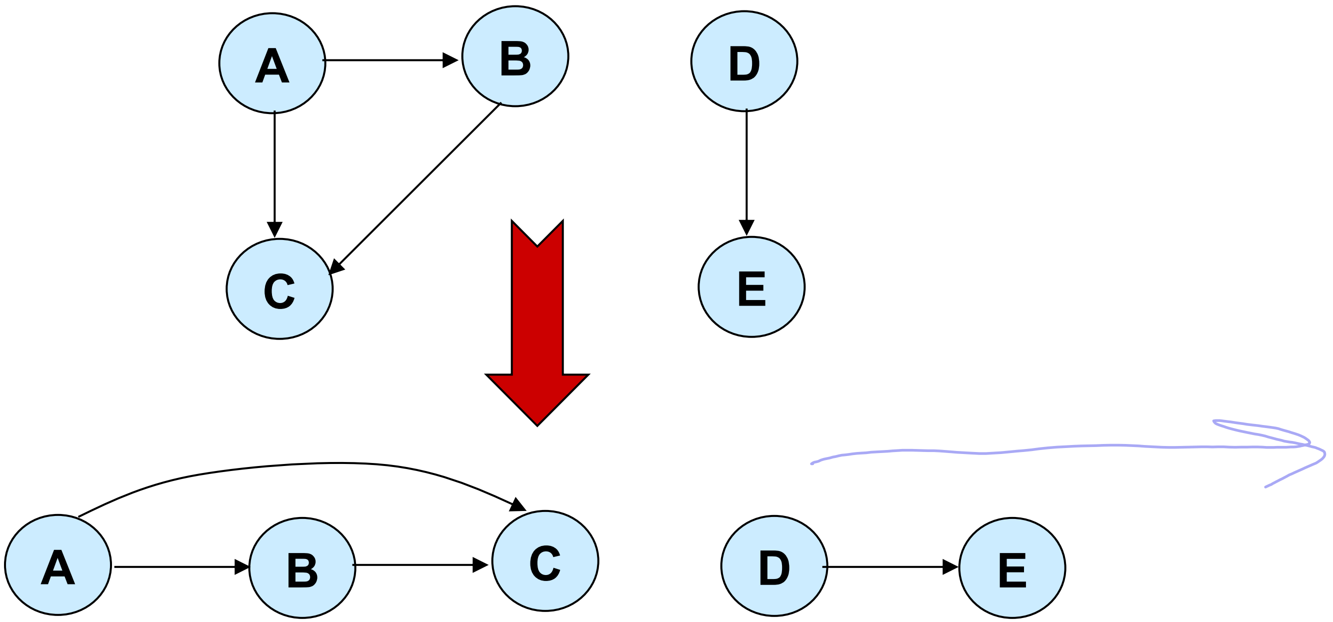
HUFFMAN: Merge the two least frequent letters and recurse.

L14 - Graphs

- Is there a path from s to v in G?
- Searching a graph:
  - Systematically follow the edges of a graph to visit the vertices of the graph.
- Used to discover the structure of a graph.
- Standard graph-searching algorithms.
  - Breadth-first Search (BFS).
    - Expands the frontier between discovered and undiscovered vertices uniformly across the breadth of the frontier O(V+E).
  - Depth-first Search (DFS).
    - DFS yields valuable information about the structure of a graph.
    - Discovery and finishing times have **parenthesis structure**.

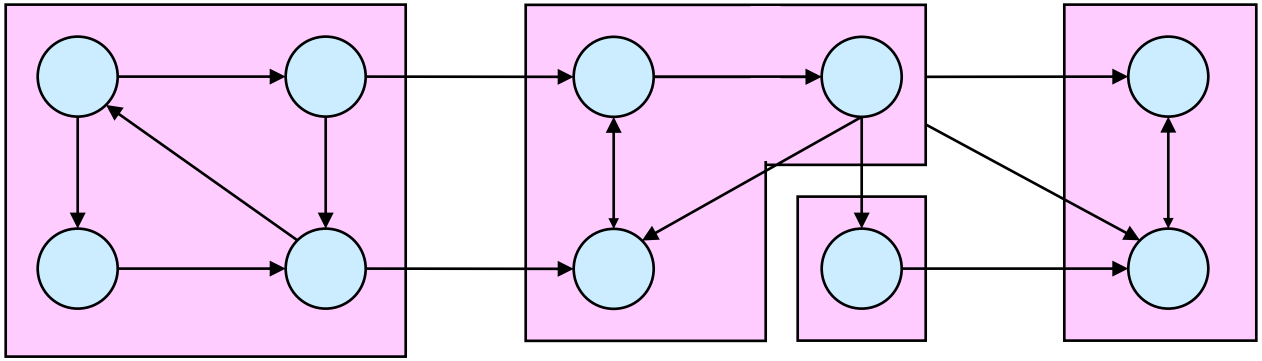
L15 – Topological Sort

- Want to ‘sort’ a DAG.
  - Linear ordering of the vertices of G such that if $(u, v) \in E$, then u appears somewhere before v.
  - Less formally, a topological ordering arranges the vertices along a horizontal line so that all edges point from left to right.
  - We studied two algorithms: **Time:** $\Theta(V + E)$.



L15 – Strongly Connected Components

- *G* is strongly connected if every pair (*u*, *v*) of vertices in *G* is reachable from one another.
- A **strongly connected component** (***SCC***) of *G* is a maximal set of vertices $C \subseteq V$ such that for all $u, v \in C$, both $u \leadsto v$ and $v \leadsto u$ exist.
- **Time:** $\Theta(V + E)$.



L16 – Flow Network

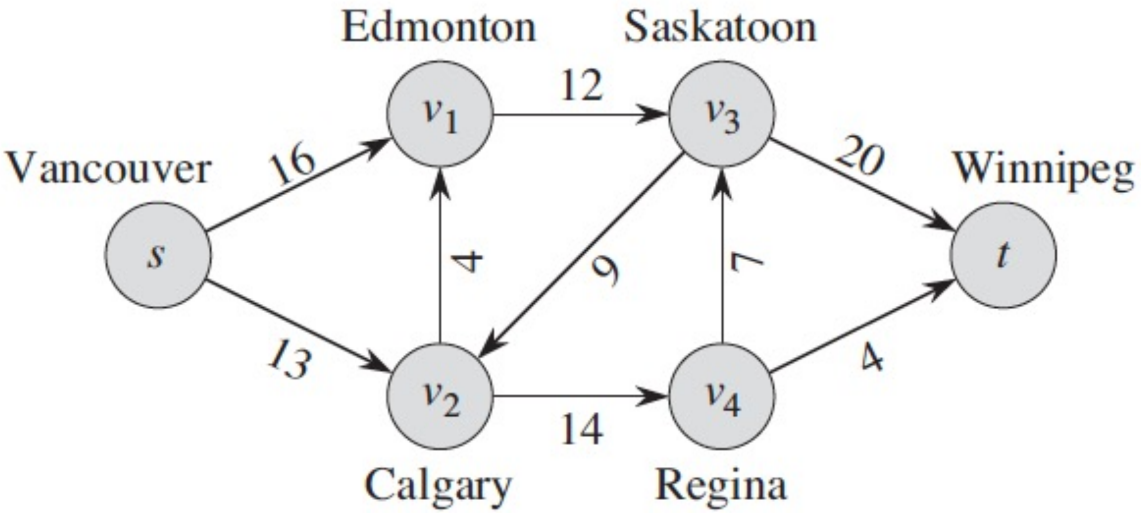
$G = (V, E)$ directed.

Each edge (u, v) has a ***capacity*** $c(u, \mathrm{v}) \geq 0$.

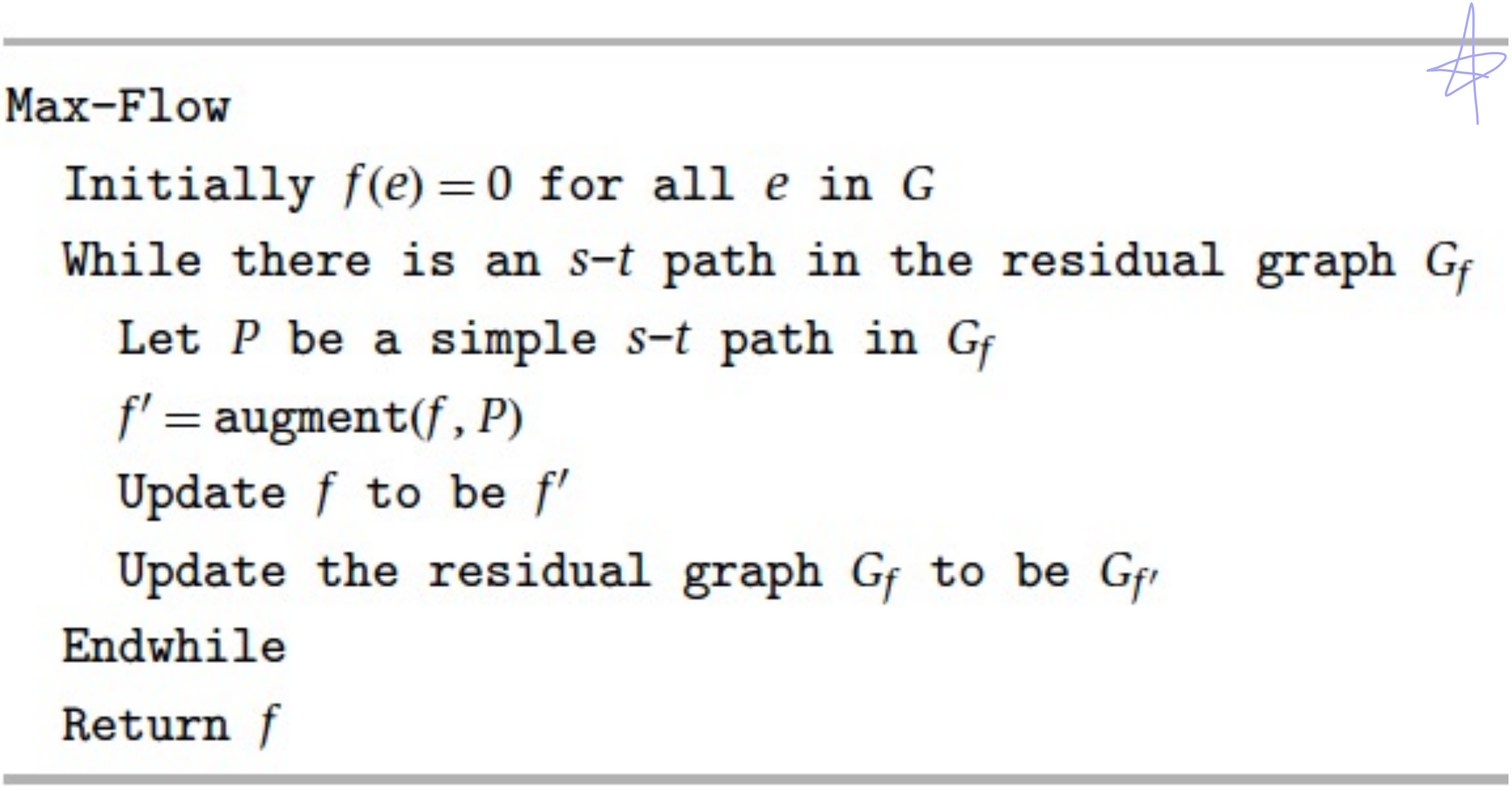
If $(u,\mathrm{v}) \notin E$, then $c(u,\mathrm{v}) = 0$.

Source vertex s, ***sink*** vertex t, assume $s \rightsquigarrow \mathrm{v} \rightsquigarrow t$ for all $\mathrm{v} \in V$.

c(u,v) is a non-negative integer.
If $(u,\mathrm{v}) \in$ E, then $(v,\mathrm{u}) \notin$ E.
No incoming edges in source.
No outcoming edges in sink.



L16 – L17 – Flow Network



```
Max-Flow
  Initially $f(e)=0$ for all $e$ in $G$
  While there is an $s$-$t$ path in the residual graph $G_f$
    Let $P$ be a simple $s$-$t$ path in $G_f$
    $f' = \text{augment}(f, P)$
    Update $f$ to be $f'$
    Update the residual graph $G_f$ to be $G_{f'}$
  Endwhile
  Return $f$
```

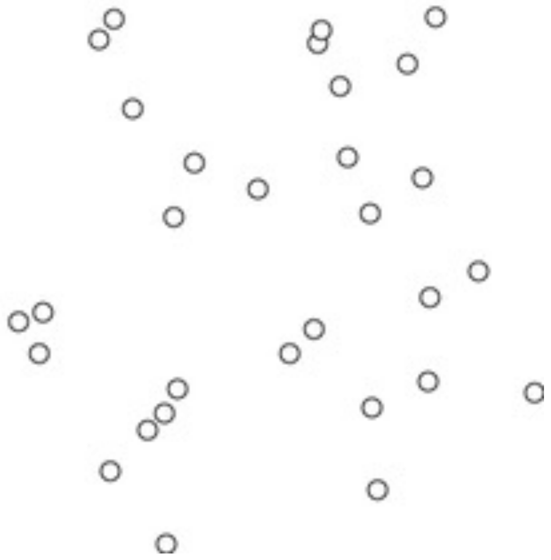
Max flow = Min cut

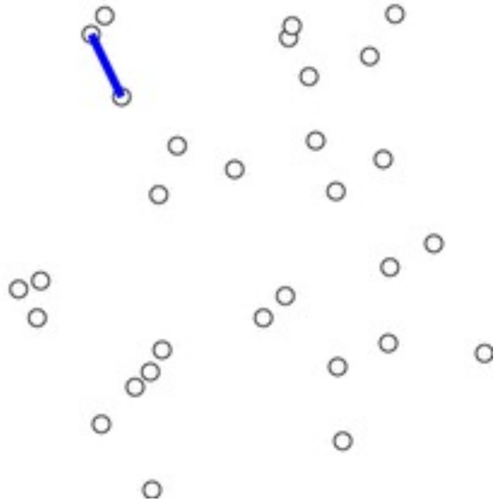
L18 – Single Source Shortest Path

- No negative-weight edges.
- Weighted version of BFS:
  - **Instead of a FIFO queue, uses a priority queue**.
  - Keys are shortest-path weights (*d*[v]).
- Have two sets of vertices:
  - *S* = vertices whose final shortest-path weights are determined,
  - *Q* = priority queue = *V* − *S*.
- Greedy choice: At each step we choose the light edge.

```
DIJKSTRA($V$,$E$,w,$s$)      $O(E \lg V)$.
$Q \leftarrow V$
while $Q \neq \emptyset$ do
    $u \leftarrow$ EXTRACT-MIN($Q$)
    $S \leftarrow S \cup \{u\}$
    for each vertex v $\in Adj[u]$ do
        RELAX($u$,v,w)
```

L19 – Minimum Spanning Tree





cross cut + primms

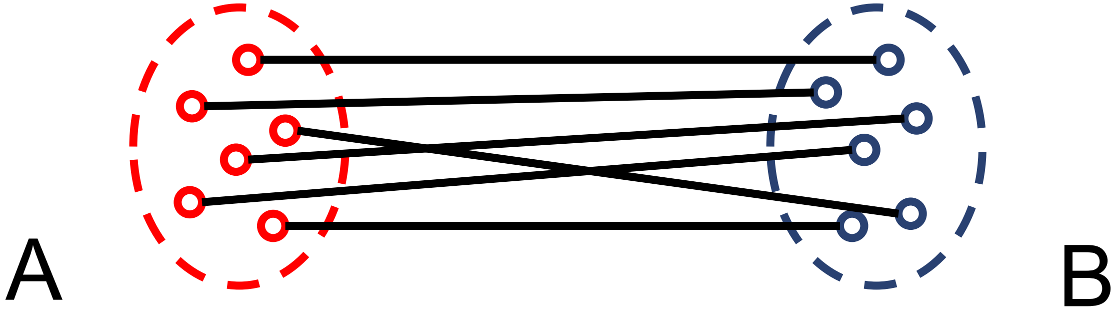
- $O(E \cdot \log V)$

1. everyone stays connected:
2. total repair cost is minimum.

```
A ← ∅;
while A is not a spanning tree do
        find a edge (u, v) that is safe for A;
        A ← A ∪ {(u, v)}
return A
```

L20 – Matching Problem

- **Goal:** Given a set of preferences among hospitals and medical school students, design a self-reinforcing admissions process.
- **Unstable pair:** applicant **x** and hospital **y** are unstable if:
  - **x** prefers **y** to their assigned hospital.
  - **y** prefers **x** to one of its admitted students.
- **Stable assignment:** Assignment with no unstable pairs.
  - Natural and desirable condition.
  - Individual self-interest will prevent any applicant/hospital deal from being made behind the scenes.



L21 – Amortized Analysis

- Analyze a sequence of operations on a data structure.

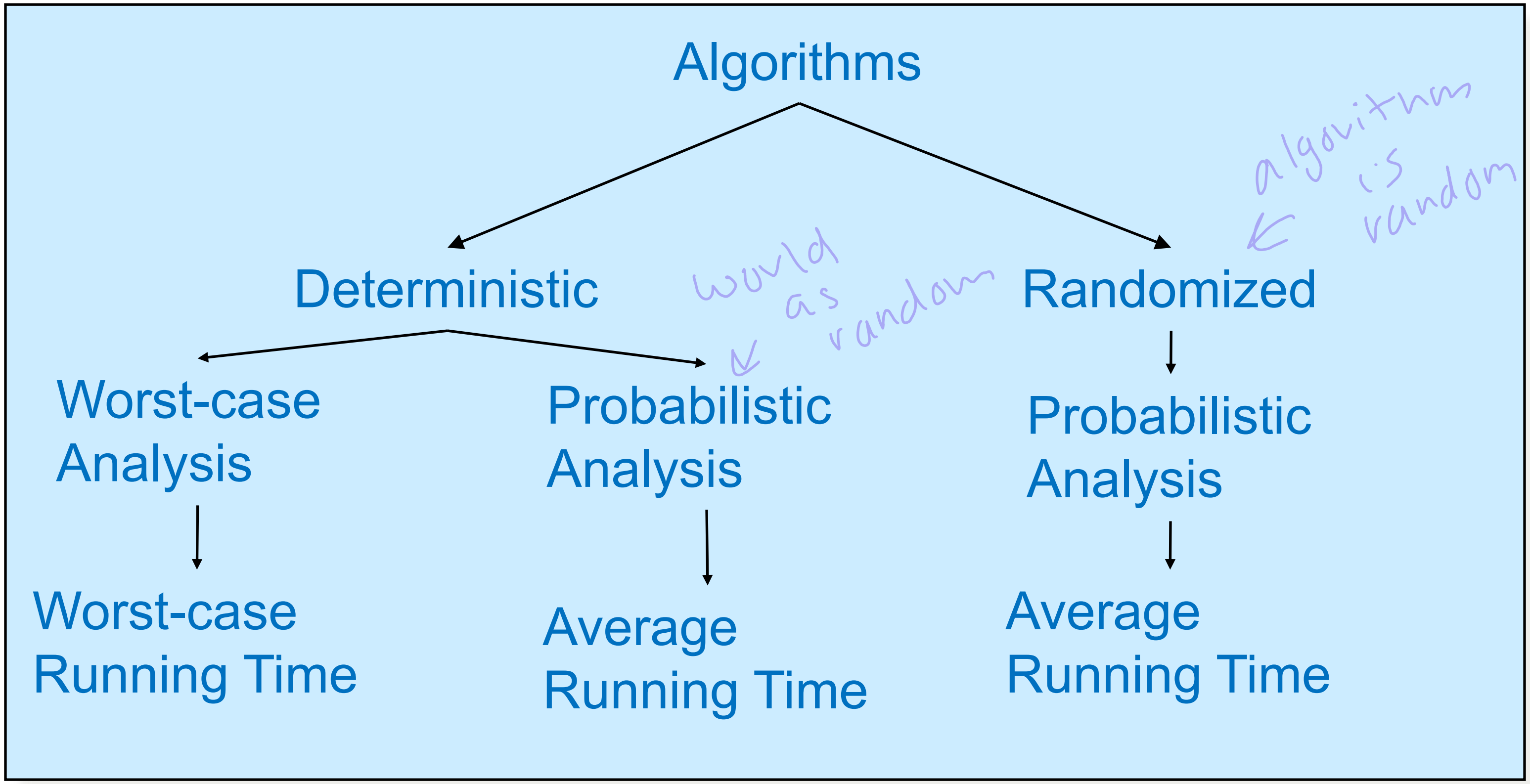
- We will talk about average performance of each operation in the worst case (i.e. not averaging over a distribution of inputs. No probability!)

- **Goal:** Show that although some individual operations may be expensive, on average the cost per operation is small.

- 3 methods:
  1. aggregate analysis: A sequence of n operations takes worst-case time T(n) in total. In the worst case, the average cost, or amortized cost, per operation is therefore T(n)/n.
  2. accounting method: Assign different charges to different operations.
  3. potential method (See textbook for more details)

L22-L23 – Randomization and Probability

- Deterministic Algorithm : Identical behavior for different runs for a given input.
- Randomized Algorithm : Behavior is generally different for different runs for a given input.



Algorithms

Deterministic

Randomized

Worst-case Analysis

Probabilistic Analysis

Probabilistic Analysis

Worst-case Running Time

Average Running Time

Average Running Time

would as random

algorithms is random

time for questions