Fundamental Algorithms
lecture #14

Cluj-Napoca, 15.01.20

TECHNICAL UNIVERSITY OF CLUJ-NAPOCA

Agenda

- B-trees - operations
- P vs NP
- Why to avoid exponential complexity?
- Why it cannot be avoided?
- Is here a paradox?
- What (not) to do

TECHNICAL UNIVERSITY OF CLUJ-NAPOCA

B-trees

- Previous DS reside in the primary memory
- **Trees on secondary storage devices (disk)**
- A node may have many children
- Goal: **decrease the number of pages accessed** when search for a node
- Store a very large number of keys
- Maintain the height of the tree under control (h very small)

B-trees – contd.

- **Typical pattern while working with B-trees:**

```
x<-pointer to some object
Disk-Read(x)
Operations that access/modify some fields of x
Disk-Write(x)
```

- **Once in memory, operations are performed fast**
- **Objective: as few pages read/write operations**

B-trees – contd.

Generalization of BST (with ordered lists)

- P1: n[x] keys in node x
- P2: keys are ordered

$key_1[x] <= key_2[x] <= \ldots <= key_{n[x]}[x]$

- P3: An internal node contains n[x]+1 pointers to the children $c_i[x]$
- P4: is a search tree:

key[c_1[x]]<= key_1[x]<=keyc[$_2$[x]] <=

- P5: All leaves are at the same level = height of the tree = h
- P6: t = degree of the tree; min(t)=2. Every node (except for the root) has at **least *t-1 keys***, and ***t children***
- P7: Every node (except for the root) has at **most *2t-1 keys***, and ***2t children***

B-trees – contd.

- **One pass procedures (top-down ONLY, NO back up; as opposed to PBT, AVL, RB trees)**
- **For ALL operations, the process is JUST top->down!!!**
- **Search**
  - Straightforward generalization of BST search
  - Combined with ordered list search
  - #pages accessed (worst) O(h)
  - Access time in a page (worst) O(t)
  - **Overall O(th)**

B-trees – contd.

- **One pass procedures (top-down ONLY,** NO back up rebalance; as opposed to PBT, AVL, RB trees**)**
- **Insert**
  - **Search** for a LEAF position to insert
  - Insertion is performed in an **EXISTING** leaf
  - Along the path while searching (top-down), ensure **there is space for a safe insert** (*split full nodes* on the path down to avoid overflowing, so that the insertion is successful in an existing node!!!)
  - A key added to a **full** leaf will induce:
    - the ***migration of the median key to the parent node,***
    - and the **split** of the given leaf into 2 leaves
  - **Time: O(th)**
    - **O(h) disk accesses,**
    - **O(t) CPU time in one page**

B-trees – contd.

- **One pass procedures** (**top-down**, NO back up; as opposed to AVL, RB trees)
- **Delete**
  - **Search** for the node to contain the key to be removed and identify its type (leaf/no leaf – all nodes are either internal, or leaves; the tree is complete) (z from BST)
  - Physical ***removal*** of one object which *belongs to a leaf* (y in BST)
    - Q: Why y, a node with one successor only in a BST is in a leaf in a B-tree?
  - Along the path while searching (top-down), **ensure the constraints for a safe delete are met** (*merge nodes with degree t* on the path down to avoid underflowing, so that the deletion is possible)
  - **Time: O(th)**
    - **O(h) disk accesses**
    - **O(t) CPU time in one page =>**

B Trees - insert

- Like in any BST tree insert in a leaf (in a leaf, NOT as leaf; the node is NOT now created!)
- Stages:
  - **Search** the path for the position (leaf) to insert
  - Ensure the search **path is safe**
  - **Insert** the key in the corresponding **leaf**
- Types of nodes to store a key:
  - Leaf (key to be inserted)
  - Non-leaf (the "safe path" step = in the attempt to make room = in the split stage with median migration up)
- Possible issues
  - Attempt to store in a full node, with (2t-1) keys – issue – node overflows! Not allowed.
- Cases to analyze
  - Not overflowing node – no issue
  - Overflowing node – issue – needs a strategy to handle it

B Trees -insert

- **Strategy**
  - Along the searched path, ANY full node along the path (with already (2t-1) keys) is "fixed" (allowing for a potential full node to accept a new key to be added):
    - **Divide the full node** in 2 nodes with (t-1) keys
    - The **median key** in the full node is **promoted to the parent** node (there *is room*, as we proceed top-down, and an upper node was "fixed", is not overflowing)
    - if ***root is full*, increase the height** (by adding 1 more node = new root); the ONLY case of *height increase*.
- Insert procedure
  - Top-down approach (descendent)
  - There is NO operation performed on return (bottom up)

B Trees – delete

- **Like in ANY other BST tree**
  - **Search for the key to remove** (pointed by **z**)
  - **If in leaf, delete it**
  - **If not in the leaf, remove (physically) a node** (pointed by **y**; its content is moved in the node pointed by z) **with one-single child** (pointed by **x**) **- the predecessor/successor – (in B-trees y is in a leaf, hence x points to nil)**
- **Types of nodes containing the key to be deleted**
  - **Leaf**
  - **Non-leaf (i.e. internal)**

Node type	Capacity
Leaf	Does not underflow
Non-leaf	Underflows

- **Possible issues**
  - **Attempt to delete from a node with only (t-1) keys – issue – node underflows! Not allowed**
- **Cases to analyze**

B Trees – delete

- **Cases to analyze**

Issue: attempt to delete from a node with only (t-1) keys –
node underflows! Not allowed

Node type	Capacity
Leaf	Does not underflow
Non-leaf	Underflows

- **Solution**
  - Similar strategy as in case of insert: **prevent, rather than repair**
  - In the **search stage (for the key to delete**), ensures that **each** node (along the searched path) has the ability to allow for delete (does not underflow)
  - **Along the searched path, any node with only (t-1) keys is "fixed"**
    - If any of the **sibling** nodes has **at least t keys, "borrow"** a key from it (promote the last/first key from the left/right brother node to the parent node, and move down the appropriate key from parent to the almost underflowing node). (see examples for case 3.1 – next slide)
    - If **both sibling** neighbors have **only (t-1) keys, merge** the underflowing node with one of the 2 siblings, and **put the key from the parent node in between them in the new generated (by merge) node.** (see examples for case 3.2.1 next slide). Maybe a height shrink occurs (see examples for case 3.2.2 next slide).
- **Delete procedure**
  - Top-down approach (descendent)
  - There is no operation performed on return (bottom up)

B Trees –delete contd.

Node type	Capacity
Leaf	Does not underflow
Non-leaf	Underflows

- Cases – only 3 to be considered (not 4): non-leaf/underflow is not considered (since along the path, the underflow situation is solved, anyway).
- Cases: (follow the examples – blackboard)
  - Case1 – key in leaf, node does not underflow
    - Simply delete the key
  - Case2 – key not in leaf, node does not underflow
    - Remove pred/succ (from a leaf for a BT) like in BST. The case reduces to case 1 or 3.
  - Case3 – key in leaf, node underflows
    - 3.1 sibling consistent (at least one sibling neighbor has at least t keys) sol: "borrow" from sibling
      - Promote the last/first key in consistent neighbor to parent
      - Move down the key in parent node in between the leaf and consistent sibling to the underflowing leaf
    - 3.2 neighbor siblings both with just (t-1) keys
      - Merge the underflowing leaf with one sibling neighbor by adding in the middle the key in the parent in between the leaf and sibling
        - 3.2.1 keep the height of the tree
        - 3.2.2 *decrease the height* of the tree (if the *parent* is the *root* and has just one single key)

TECHNICAL UNIVERSITY OF CLUJ-NAPOCA

P vs NP

- Why to avoid exponential complexity?
- Why it cannot be avoided?
- Is here a paradox?
- What (not) to do

TECHNICAL UNIVERSITY OF CLUJ-NAPOCA

Complexity

- Ω characterizes the problem, lower bound
- O characterizes the alg that solves that problem, upper bound

- An algorithm is optimal if the running time of the algorithm to solve the problem in the worst case scenario equals the lower bound of the given problem and NO additional space is used:

$$O = \Omega$$

Complexity – cont.

- leading (lim)

f1(n)	leads	f2(n)
n^n		$n!$
$n!$		a^n, $a>1$
a^n		b^n, $a>b$
a^n		n^b, $a>1$
$\log_a n$		$\log_b n$, $b>a>1$
$\log_a n$		1, $a>1$

We ONLY discussed problems with solutions in the blue range.
WHY?

TECHNICAL UNIVERSITY OF CLUJ-NAPOCA

Complexity – cont.

- Q: How max dim (of the problem that can be solved on a computer) grows in case we increase the speed of the computer?
- How different classes of algs affect performance?

Complexity – cont.

- Remember the experiment of 2 classes of algs:
  - Alg1: polynomial
  - Alg2: exponential
- Assumption of a new hardware system, and its speed increases **V** times (compared to a former system)
- **Q?** How does this affect (increase) the max dim of the problem to be solved on the new system?
- That is: $n_2=f(V,n)$
  - V=increase of speed of the new machine
  - n=max dim on the former (let's call it old) machine

TECHNICAL UNIVERSITY OF CLUJ-NAPOCA

Complexity – cont.

Speed of the new computer in terms of the old one: $V_2 = V \cdot V_1$

Alg1: **$O(n^k)$:** $n_2 = v^{1/k} \cdot n$

Alg2: **$O(2^n)$:** $n_2 = n + \lg V$

CL: For exp algs, no matters how many times we increase the speed of the system, the dim increases with an additive constant!!!

Sol:

avoid implementing exponential algorithms!

are there any **problems** with no known poly alg?

P=NP ? 1 million USD problem

TECHNICAL UNIVERSITY OF CLUJ-NAPOCA

Birth of NP-comp

- Cook, Stephen (1971). "The complexity of theorem proving procedures". Proceedings of the Third Annual ACM Symposium on Theory of Computing. pp. 151–158.
  - any problem in NP can be reduced in polynomial time by a deterministic Turing machine to the problem of determining whether a Boolean formula is satisfiable.
- Karp, Richard M. (1972). "Reducibility Among Combinatorial Problems". In Raymond E. Miller and James W. Thatcher (editors). Complexity of Computer Computations. New York pp. 85–103.
  - The list of 21 NP-complete problems
- BOTH – Turing award

TECHNICAL UNIVERSITY OF CLUJ-NAPOCA

Stephen Cook

- Education
  - received his Bachelor's degree in 1961 from the University of Michigan
  - received his Master's degree and Ph.D. from Harvard University, respectively in 1962 and 1966.
- Career
  - 1966-1970 Assistant Professor - joined the University of California, Berkeley, mathematics department
  - 1970 when he was denied reappointment!!!
  - 1970 Associate Professor- joined the University of Toronto, Computer Science and Mathematics Departments
  - 1975 Professor - University of Toronto
  - 1985 University Professor - University of Toronto
- !!!
  - In a speech celebrating the 30th anniversary of the Berkeley EECS department, fellow Turing Award winner and Berkeley professor Richard Karp said that, "It is to our everlasting shame that we were unable to persuade the math department to give him tenure."

TECHNICAL UNIVERSITY OF CLUJ-NAPOCA

Karp's 21 NPC problems ('72)

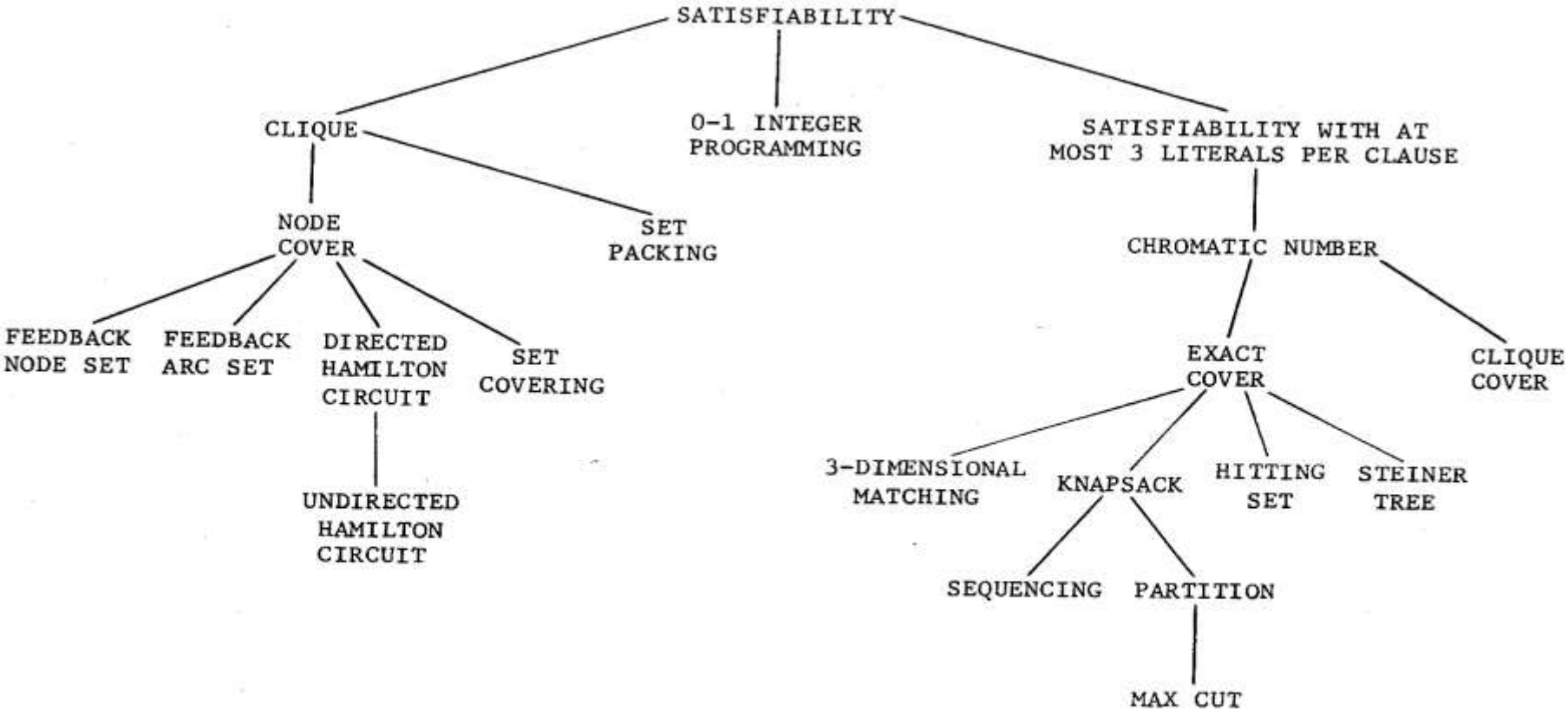


FIGURE 1 - Complete Problems

TECHNICAL UNIVERSITY OF CLUJ-NAPOCA

P vs NP

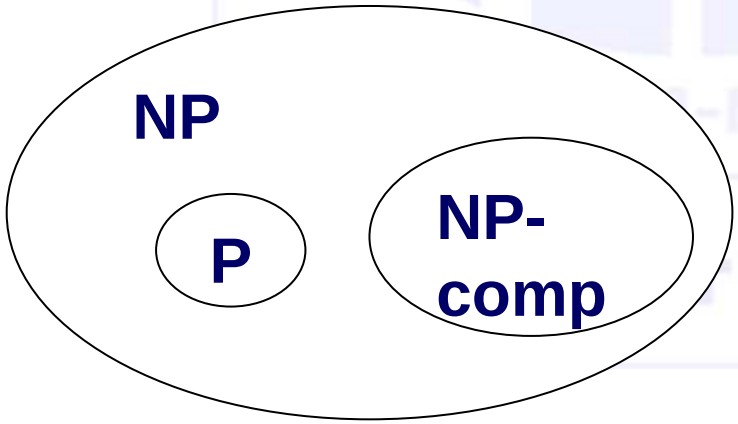
- **NP=** nondeterministic polynomial
- **NP-complete** = most complex in NP

- **Complexity class NP** = the set of **DECISION** problems which can be **VERIFIED** in poly time
- **Decision** problem = a problem which is ONLY asking whether there exists a solution; it only answers by yes/no
- This is NOT a limitation
- ANY **optimization** problem can be transformed into a decision problem (by adding a threshold which we ask to reach)
- In NP-comp we only discuss about decision problems

NP class

?

$P = NP$



NP-complete Problems

(as in Cormen, Leiserson, Rivest – first edition)

Circuit-SAT

SAT

3-FNC SAT

Clique **Ham-Cycle**

Vertex Cover **TSP**

Sum

TECHNICAL UNIVERSITY OF CLUJ-NAPOCA

Hard/easy problems

(from Vazirani @ Berkeley)

	Hard problems (NP-complete)	Easy problems (in P)
1	3SAT	2SAT, HORN SAT
2	TRAVELING SALESMAN PROBLEM	MINIMUM SPANNING TREE
3	LONGEST PATH	SHORTEST PATH
4	3D MATCHING	BIPARTITE MATCHING
5	KNAPSACK	UNARY KNAPSACK
6	INTEGER LINEAR PROGRAMMING	LINEAR PROGRAMMING
7	RUDRATA PATH	EULER PATH
8	SUM	2-SUM
9	INDEPENDENT SET	INDEPENDENT SET on trees

Easy/Hard

- We have seen (seminars) some pairs:
  - Subset Sum: Size_2_Subset/Unknown_Size_Subset
  - Bipartite Graph/Multipartite Graph
  - EulerTour/Hamiltonian Cycle
- What's the issue here?
  - Identify the membership of the complexity class
  - P= problems that can be decided in poly time
  - NP = problems that can be verified in poly time
  - NP-comp = most complex problems in NP (any problem in NP-comp can be transformed in any other problem in NP-comp + can verified in polynomial time)
  - NP-hard = even more complex ☹; we cannot verify in poly time
- Approach
  - Cleary identify the membership
  - Do NOT attempt to solve NP-comp (or higher) problems
  - Most real-world problems belong to this category
  - Context analysis, approximation solutions, … (master course)