Data Structures

September 21

Org Remarks

- Send your Assignments (of course, zipped – since your work will usually consist of several files) to Simon Zaaijer (email: szaaijer@liacs.nl)
- In addition to the above turn in a paper copy of your work in my mail box on the day of the deadline
- Let your zip file have a name which can be traced to your team in the following standard way
  - username1andUsername2No1.zip,
  - where username1 is the liacs username of the first member and username2 is the liacs username of the second member
- Moreover: each file you are turning in should contain the names of the team members, date turned in, due date, Assignment No; all this in a conspicuous manner at the top of the files
- furthermore a README file should describe which files your turning in and instructions for compilation and execution, if necessary

Sparse Tables

Sparse tables: example student grades

FIGURE 3.21 Arrays and sparse table used for storing student grades.

students

1	Sheaver Geo
2	Weaver Henry
3	Shelton Mary
:	
404	Crawford William
405	Lawson Earl
:	
5206	Fulton Jenny
5207	Craft Donald
5208	Oates Key
:	

(a)

classes

1	Anatomy/Physiology
2	Introduction to Microbiology
:	
30	Advanced Writing
31	Chaucer
:	
115	Data Structures
116	Cryptology
117	Computer Ethics
:	

(b)

gradeCodes

a	A
b	A–
c	B+
d	B
e	B–
f	C+
g	C
h	C–
i	D
j	F

(c)

grades

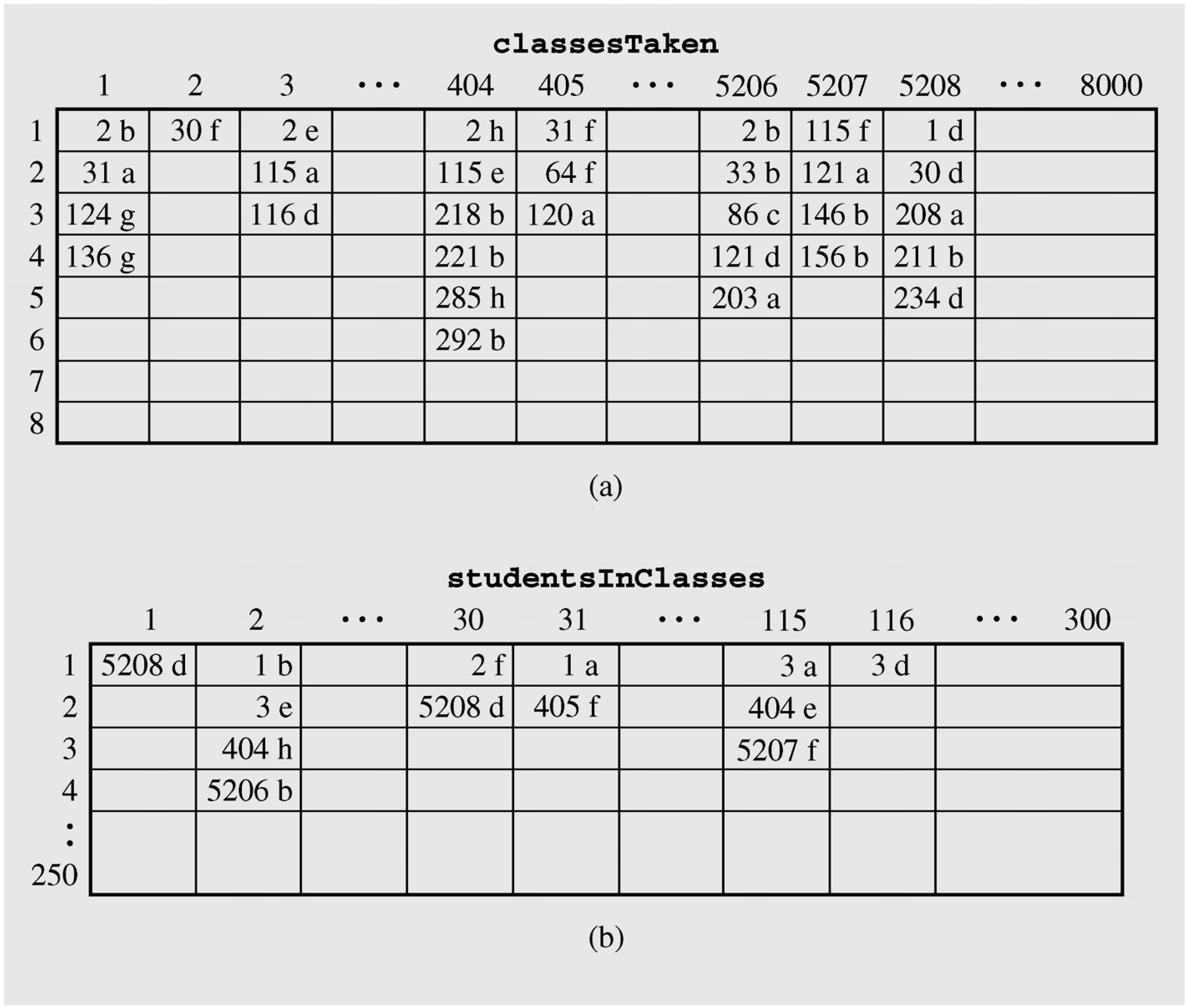
Student

Class	1	2	3	···	404	405	···	5206	5207	5208	···	8000
1										d		
2	b		e		h			b				
:												
30		f								d		
31	a					f						
:												
115			a		e				f			
116			d									
117												
:												
300												

(d)

Sparse tables: example student grades

FIGURE 3.22 Two-dimensional arrays for storing student grades.



classesTaken

	1	2	3	• • •	404	405	• • •	5206	5207	5208	• • • 8000
1	2 b	30 f	2 e		2 h	31 f		2 b	115 f	1 d	
2	31 a		115 a		115 e	64 f		33 b	121 a	30 d	
3	124 g		116 d		218 b	120 a		86 c	146 b	208 a	
4	136 g				221 b			121 d	156 b	211 b	
5					285 h			203 a		234 d	
6					292 b						
7											
8											

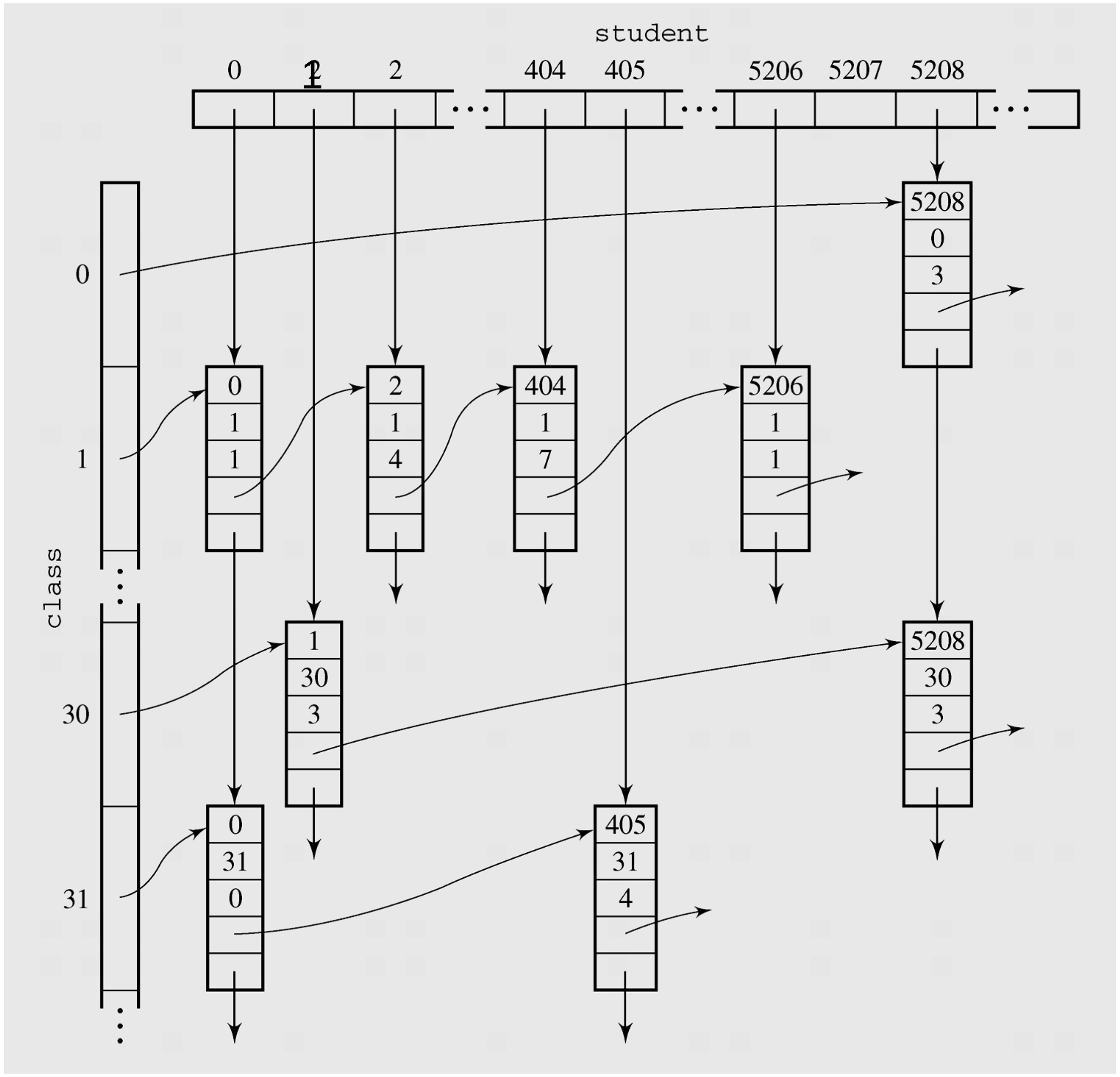
(a)

studentsInClasses

	1	2	• • •	30	31	• • •	115	116	• • • 300
1	5208 d	1 b		2 f	1 a		3 a	3 d	
2		3 e		5208 d	405 f		404 e		
3		404 h					5207 f		
4		5206 b							
: 250									

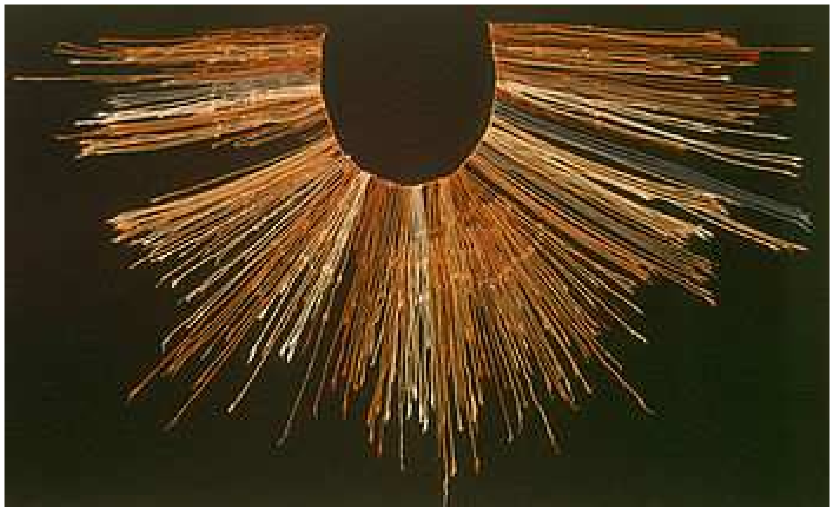
(b)

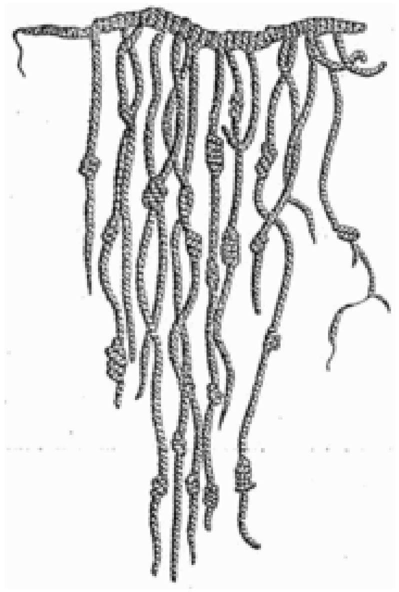
Sparse tables: example student grades



Recap Linear Structures

(see commons.wikimedia.org/wiki/Image:Sumerian_26th_c_Adab.jpg)





Possibly the world’s oldest data
Structures: tablets in cuneiform
Script used more than 5000 years
Ago by custodians in Sumerian temples

Shown: the 4600 year old tablet at the top left
Is a list of gifts to the high priestess of Adab

Peruvian quipu

- Remarks on the ADT we tried to specify last time: “ADT List With Insertion Influenced Implicitly by Search”
  - Implementations with Self-Organization: move-to-front, transpose, count-method (excluding ordered implementation)

Summary

- A linked structure is a collection of nodes storing data and links to other nodes.
- A linked list is a data structure composed of nodes, each node holding some information and a reference to another node in the list.
- We viewed each of the linked structures in Ch3 as a different implementation of **ADT ListDrozdek or ADT ListDrozdek+op whoIsNext .**
- A singly linked list is a node that has a link only to its successor in this sequence.
- A circular list is when nodes form a ring: The list is finite and each node has a successor.

Summary (continued)

- A skip list is a variant of the ordered linked list that makes a non sequential search possible.
- There are four methods for organizing lists: move-to-front method, transpose method, count method, and ordering method.
- Optimal static ordering - all the data are already ordered by the frequency of their occurrence in the body of data so that the list is used only for searching, not for inserting new items.

Summary (continued)

- A sparse table refers to a table that is populated sparsely by data and most of its cells are empty.
- Linked lists allow easy insertion and deletion of information because such operations have a local impact on the list.
- The advantage of arrays over linked lists is that they allow random accessing.

Hierachical structures: Trees

•

Objectives

Discuss the following topics:

- Trees, Binary Trees, and Binary Search Trees
- Implementing Binary Trees
- Tree Traversal
- Searching a Binary Search Tree
- Insertion
- Deletion

Objectives (continued)

Discuss the following topics:

- Heaps
- Balancing a Tree
- Self-Adjusting Trees

Trees, Binary Trees, and Binary Search Trees

- A **tree** is a data type that consists of **nodes** and **arcs**
- These trees are depicted upside down with the root at the top and the **leaves** (**terminal nodes**) at the bottom
- The **root** is a node that has no parent; it can have only child nodes
- Leaves have no children (their children are null)

Trees, Binary Trees, and Binary Search Trees (continued)

- Each node has to be reachable from the root through a unique sequence of arcs, called a **path**
- The number of arcs in a path is called the **length** of the path
- The **level** of a node is the length of the path from the root to the node plus 1, which is the number of nodes in the path
- The **height** of a nonempty tree is the maximum level of a node in the tree

Trees, Binary Trees, and Binary Search Trees (continued)

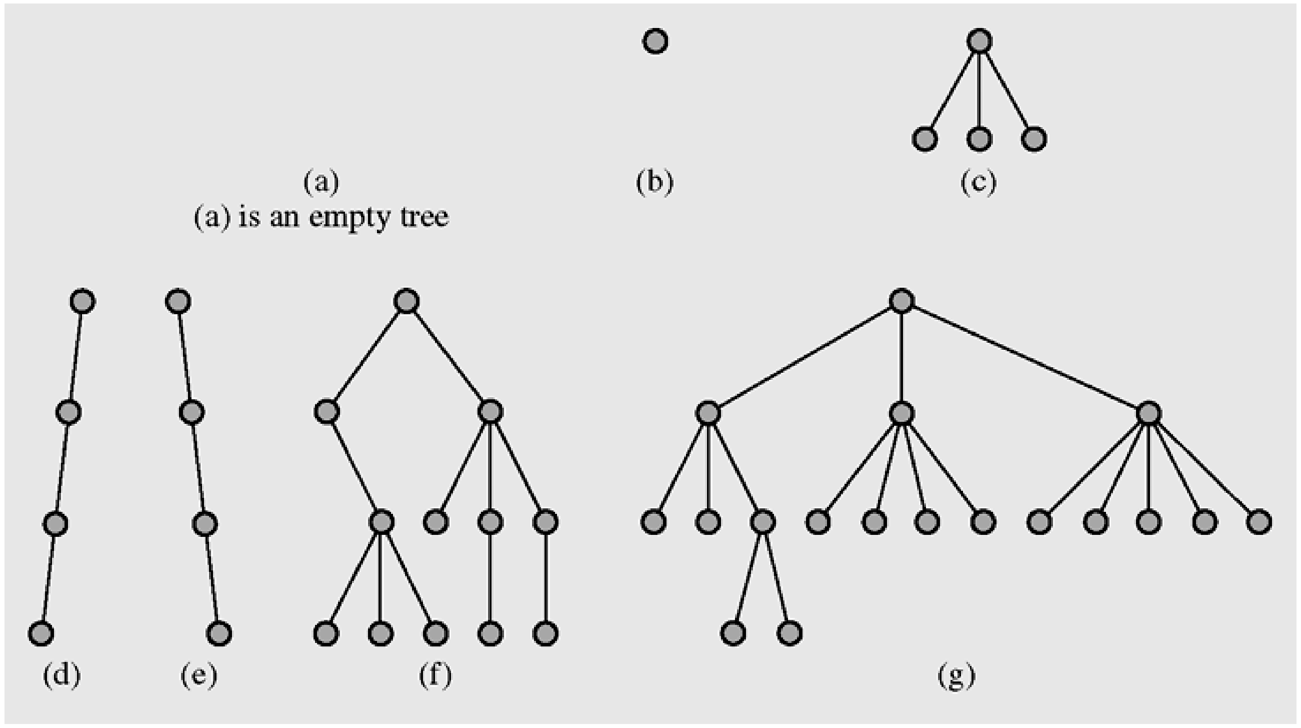


Figure 6-1 Examples of trees

Trees, Binary Trees, and Binary Search Trees (continued)

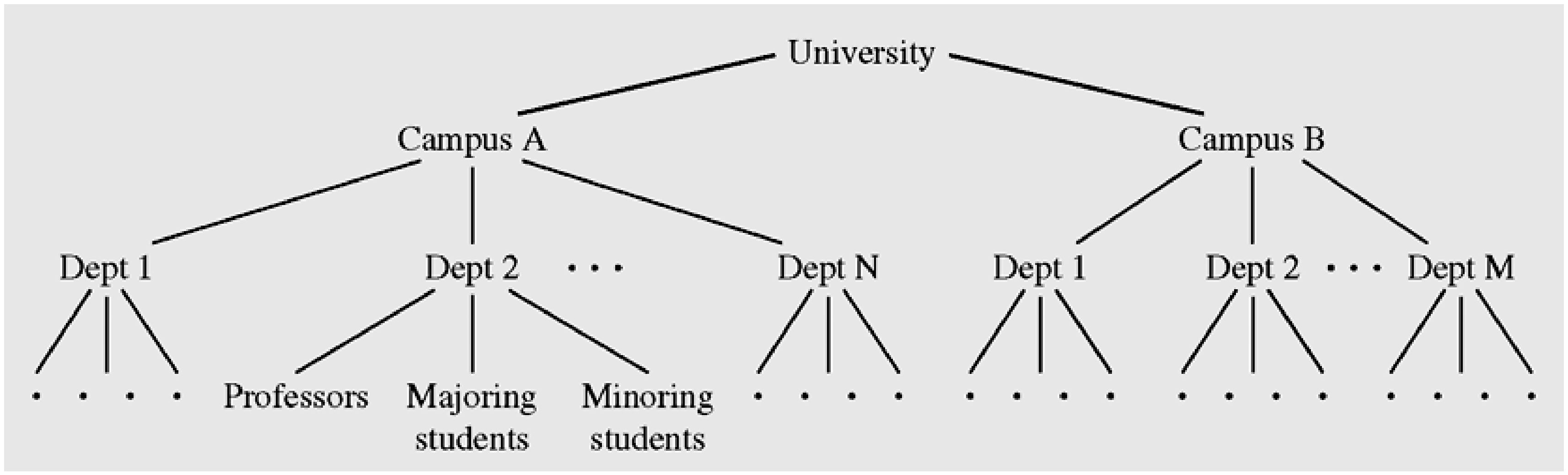


Figure 6-2 Hierarchical structure of a university shown as a tree

Trees: abstract/mathematical

important, great number of varieties

- terminology

  (knoop, wortel, vader, kind)
  node/vertex, root, father/parent, child
  (non) directed
  (non) orderly
  binary trees (left ≠ right)
  full (sometimes called decision trees, see Drozdek), complete (all levels are filled, except the last one)

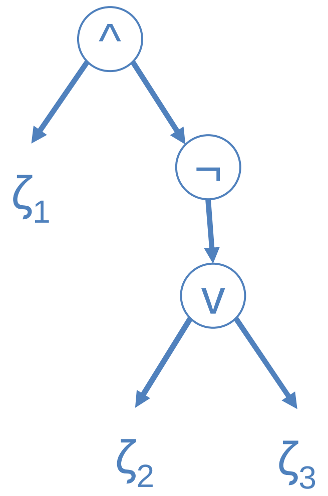
- categorization

structure

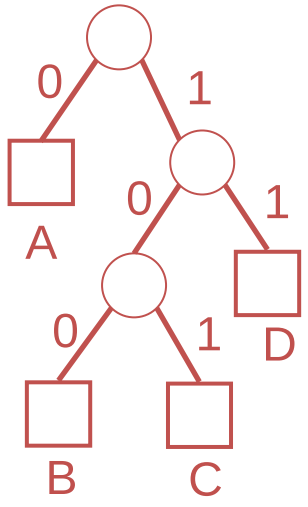
  number of children (binary, B-boom)
  height of subtrees (AVL-, B-trees)
  compleet (heap)

Location of keys

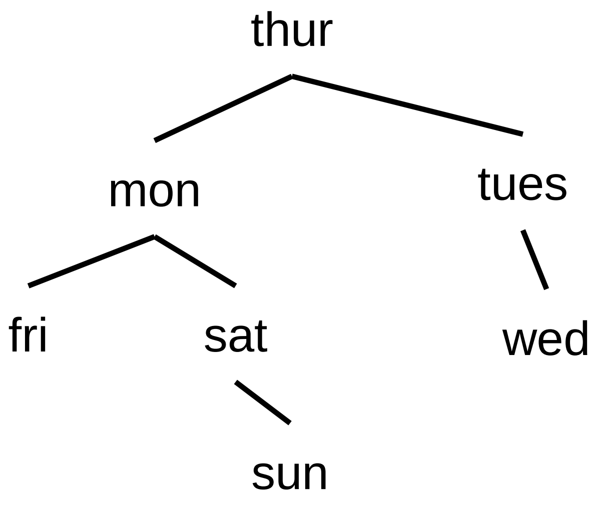
  search tree, heap



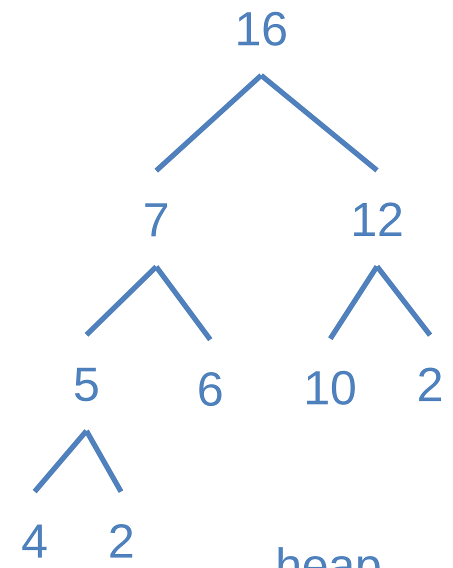
expression



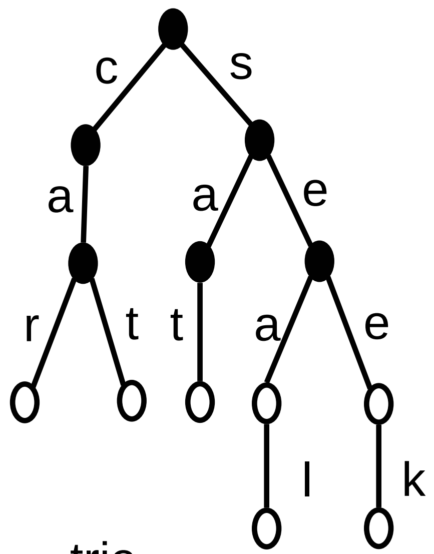
code



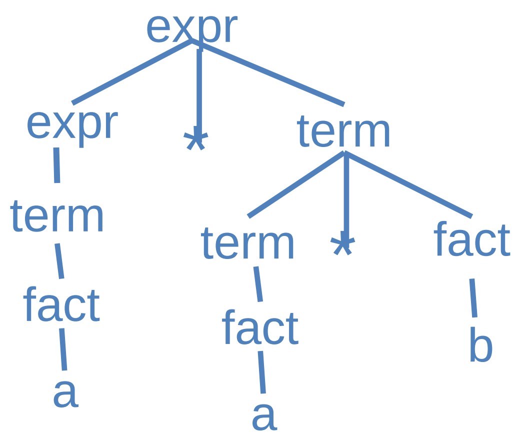
bst



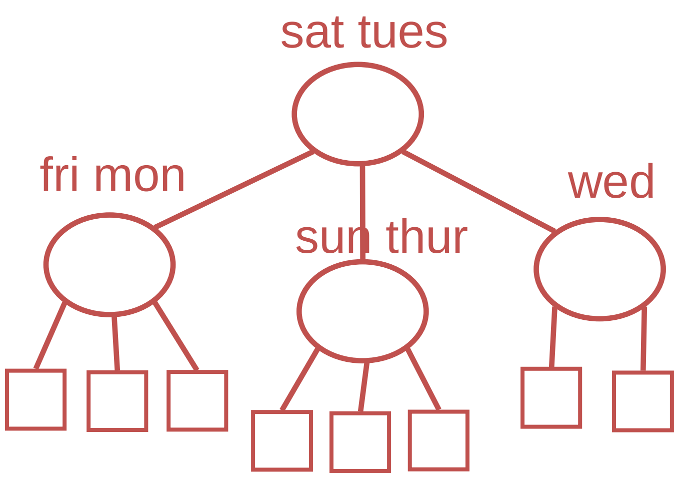
heap



trie



syntax



B-tree (2,3 tree)

Recall Definition of Tree

1. An empty structure is a tree
2. If t1, ..., tk are disjoint trees, then the structure whose root has as its children the roots of t1,...,tk is also a tree
3. Only structures generated by rule 1 and 2 are trees

Alternatively: a connected graph which contains no cycles is a tree

Equivalent statements (see ϕ1)

- Let T be graph with n vertices then the following are equivalent:
  - a) T is a tree (= no cycles and connected)
  - b) T contains no cycles, and has n-1 edges
  - c) T is connected, and has n-1 edges
  - d) T is connected, and every edge is a bridge
  - e) Any two vertices are connected by exactly one path
  - f) T contains no cycles, but the addition of any new edge creates exactly one circuit (cycle with no repeated edges).

Trees, Binary Trees, and Binary Search Trees (continued)

- An **orderly tree** is where all elements are stored according to some predetermined criterion of ordering

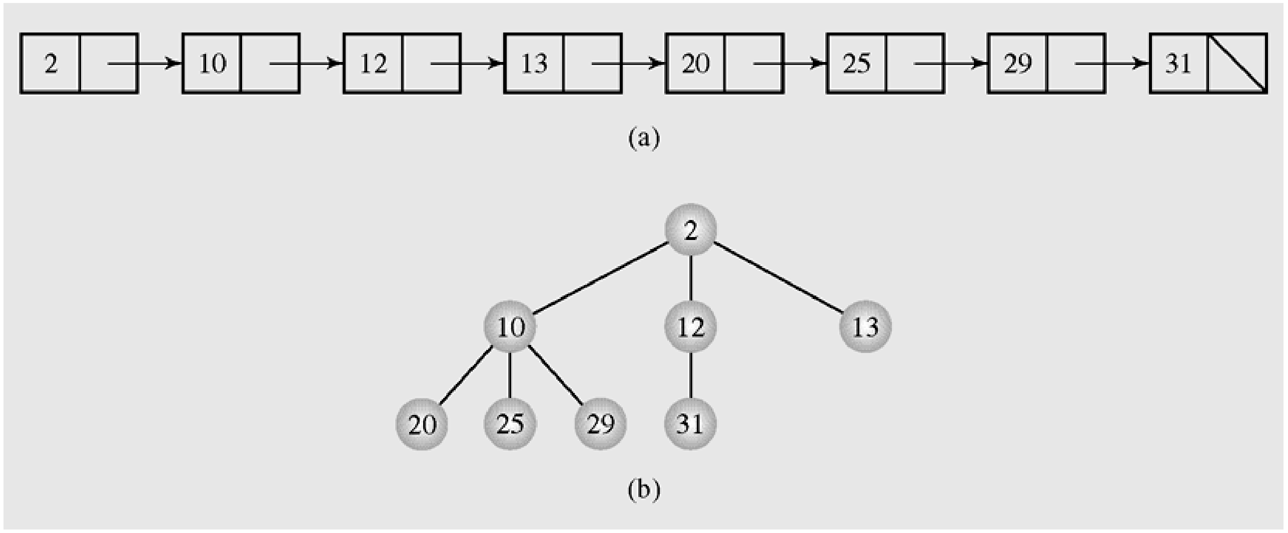


Figure 6-3 Transforming (a) a linked list into (b) a tree

Binary Trees

- A **binary tree** is a tree whose nodes have two children (possibly empty), and each child is designated as either a left child or a right child

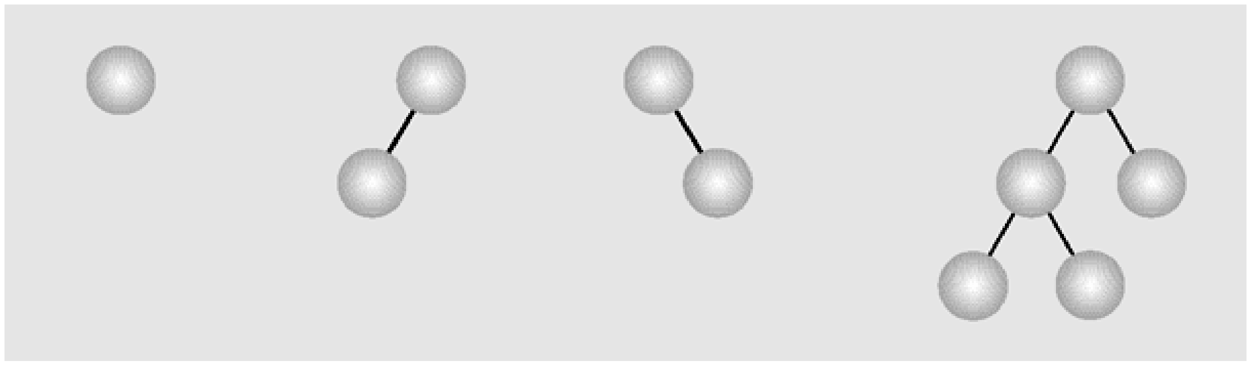
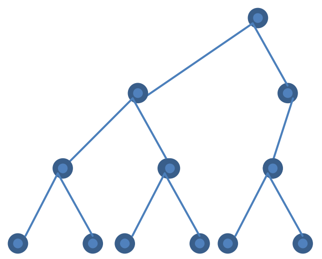
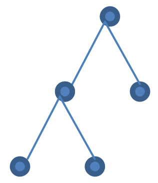


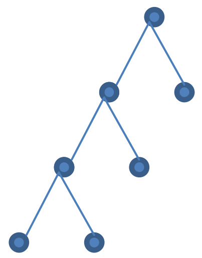
Figure 6-4 Examples of binary trees

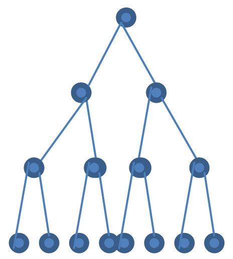
Binary Trees

- In a **complete binary tree**, all the nodes at all levels have two children except the last level.
- A **decision tree** is a binary tree in which all nodes have either zero or two nonempty children


incomplete Binary tree


Complete Binary tree Dutch: compleet)


Decision tree (Dutch: vol)


complete Decision tree