Department of Informatics

Prof. Dr. Michael Böhlen
Binzmühlestrasse 14
8050 Zurich
Phone: +41 44 635 4333
Email: boehlen@ifi.uzh.ch

University of Zurich[UZH]

Informatics IIb — Final Exam
Spring 2016 — 9.6.2016

Name: _____ Matriculation number: _____

Advice

If you only attend the module *Informatik IIb*, you have 90 minutes to complete the exam. If you attend both modules, Informatik IIa and IIb, you have 120 minutes overall to complete both parts of the exam.

The following rules apply for the written part of the exam:

- Answer the questions on the exam sheets or the backside. Mark clearly which answer belongs to which question. Additional sheets are provided upon request. If you use additional sheets, put your name and matriculation number on each of them.
- Check the completeness of your exam (22 numbered pages).
- Use a pen in blue or black colour for your solutions. Pencils and pens in other colours are not allowed. Solutions written in pencil will not be corrected.
- Stick to the terminology and notations used in the lectures.
- For the exam Informatik IIb, only the following items are allowed:
  - One A4 sheet (2-sided) with your personal handwritten notes, written by yourself. Sheets that do not conform to this specification will be collected.
  - A foreign language dictionary is allowed. The dictionary will be checked by a supervisor.
  - No additional items are allowed, notably calculators, computers, PDAs, smart-phones, audio-devices or similar devices may not be used. Any cheating attempt will result in a failed test (meaning 0 points).
- Put your student legitimation card ("Legi") on the desk.

Signature:

Correction slot — Please do not fill out the part below

Exercise	1	2	3	4	Total
Points Achieved					
Maximum Points	15	15	15	15	60

Exercise 1 15 Points

Graphs

1.1 [6 points] Consider the graph in Fig. 1 and apply depth first search (DFS) on it. If in a step multiple nodes can be chosen, select the node that comes first in alphabetical order.

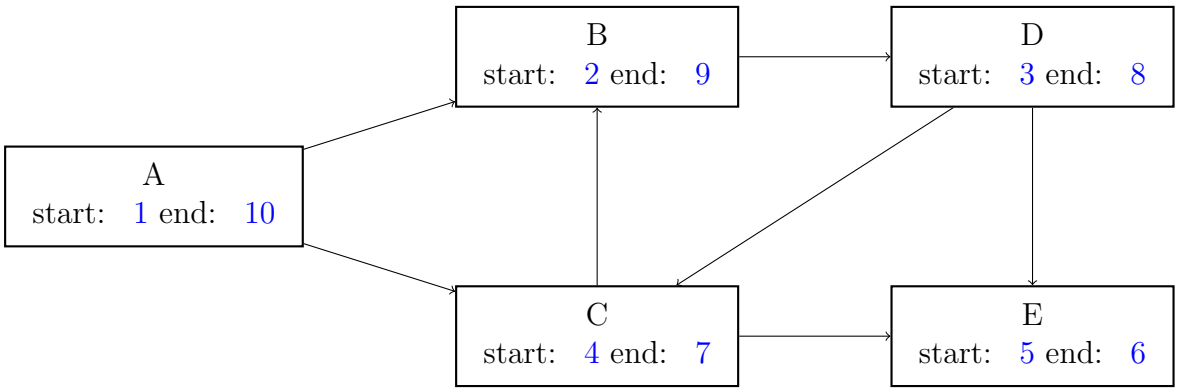


Figure 1: Directed Graph

i) Write down the start and end timestamps that each node gets assigned during a DFS.

ii) Write down the resulting parenthesis structure defined by the depth first search.
(A(B(D(C(EE)C)D)B)A)

Name: Matriculation number:

1.2 [6 points] Use C or pseudocode to describe an algorithm that receives as input a graph and returns true if the input graph is acyclic and false otherwise.

This question can be answered with a modified DFS. If DFS does not yield a back edge (edge between two grey nodes) then the input graph is acyclic.

Solution 1

```
1  Algorithm: IsAcyclic(G)
2  foreach v ∈ G.V do
3      v.color = WHITE;
4  acyclic = TRUE;
5  foreach v ∈ G.V do
6      if v.color == WHITE then
7          acyclic = acyclic AND IsAcyclicRec(G, v);
8  return acyclic;
```

```
1  Algorithm: IsAcyclicRec(G, v)
2  acyclic = TRUE;
3  v.color = GREY;
4  foreach u ∈ v.adj do
5      if v.color == WHITE then
6          acyclic = acyclic AND IsAcyclicRec(G, u);
7      else if v.color == GREY then
8          acyclic = FALSE;
9  s.color = BLACK;
10 return acyclic;
```

Solution 2

```
1  Algorithm: IsAcyclic(G)
2  foreach v ∈ G.V do
3      v.color = WHITE;
4  foreach v ∈ G.V do
5      if v.color == WHITE then
6          if !IsAcyclicRec(G, v) then
7              return FALSE;
8  return TRUE;
```

```
1  Algorithm: IsAcyclicRec(G, v)
2  v.color = GREY;
3  foreach u ∈ v.adj do
4      if v.color == WHITE then
5          if !IsAcyclicRec(G, u) then
6              return FALSE;
7      else if v.color = GREY then
8          return FALSE;
9  s.color = BLACK;
10 return TRUE;
```

1.3 [3 points] What is the best and worst case complexity of the algorithm of Task 1.2?

The algorithm is a modification of DFS.

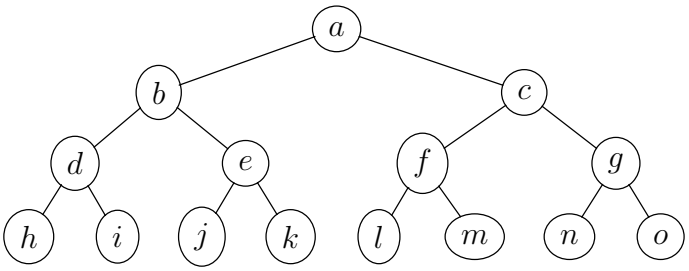
Solution 1
It runs a full DFS. The complexity for best and worst case is: $T(n) = O(V + E)$.

Solution 2
The algorithm exits early if a cycle is discovered. The worst case is when the algorithm is not acyclic and the whole graph must be scanned. The complexity of the worst case is: $T(n) = O(V + E)$. The best case is that the first vertices that are scanned form a cycle. In this case the algorithm can exit early, and the complexity is $T(n) = O(1)$.

Exercise 2 15 Points

Binary Search Trees

2.1 [5 points] Consider the following `binary search tree`, representing a sequence of integer values. The values of nodes have been hidden and are referred to through the labels "a", ..., "o".



Which nodes are the predecessors of nodes "c" and "l"?

- Predecessor of "c" is "m".
- Predecessor of "l" is "a".

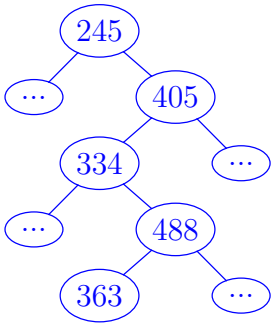
2.2 [4 points] Assume a `binary search tree` with numbers as keys and a search on this tree for 363. For each of the following series, determine if the sequence of values can be key values of nodes visited by this search? Justify your answer.

(a) 245, 405, 334, 488, 363

(b) 537, 284, 420, 413, 363

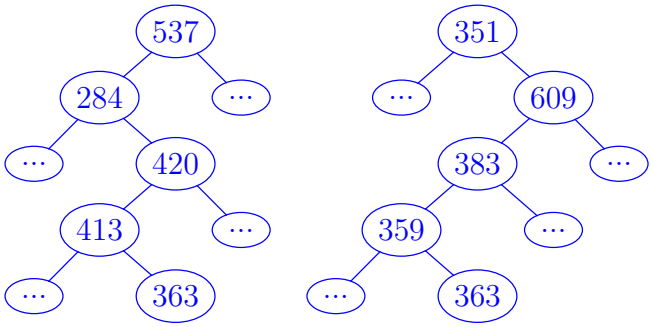
(c) 351, 609, 383, 359, 363

The first series of numbers implies a BST as follows.



This BST is not a valid, because 488 is in the left subtree of 405. So, it does not satisfy the BST property that all nodes in the left substree of a node A have key values less than the key value of A.

The second and third series imply the following valid BST:



2.3 [6 points] Use C to define the datatype of a node of a `binary search tree` with integer keys. Then, use C or pseudocode to describe an algorithm that is called with the root of a binary search tree with integer keys and prints the values of its nodes in descending order.

```
1 struct bst_node {
2   int key;
3   struct bst_node *left, *right;
4 };
```

```
1 Algorithm: PrintDescending(root)
2 if root ≠ NIL then
3     PrintDescending(root.right);
4     print(root.key);
5     PrintDescending(root.left);
```

Name: Matriculation number:

Exercise 3 15 Points

Rotated Array

The operation of rotating an array by 1 element is the operation of shifting every element one step towards the end of the array and make the last element the first element of the array. By repeating this operation m times an array is rotated by m elements. For example, consider the following array $A[1..n]$:

1	2	3	4	5	6	7
1	2	3	4	5	6	7

If A is rotated by three elements the result is the following:

1	2	3	4	5	6	7
5	6	7	1	2	3	4

3.1 [2 points] Consider the following arrays and m values. Write the result array after rotating them by m elements.

m = 2

1	2	3	4	5
5	6	18	36	81

1	2	3	4	5
36	81	5	6	18

m = 3

1	2	3	4	5
43	47	61	63	85

1	2	3	4	5
61	63	85	43	47

m = 7

1	2	3	4	5
43	49	65	67	87

1	2	3	4	5
67	87	43	49	65

3.2 [3 points] Specify all special cases for m that must be considered and provide examples of the input data for each of them.

#	Case	A	n	m	result
1	$m = 0$	$[1 \quad 2 \quad 3]$	3	0	A = $[1 \quad 2 \quad 3]$. The array does not change.
2	$m = n$	$[1 \quad 2 \quad 3]$	3	3	A = $[1 \quad 2 \quad 3]$. A full rotation by n elements results in the same array.
3	$m < n$	$[1 \quad 2 \quad 3]$	3	2	A = $[2 \quad 3 \quad 1]$. The array is rotated by m elements.
4	$m > n$	$[1 \quad 2 \quad 3]$	3	7	A = $[3 \quad 1 \quad 2]$. Rotations by multiples of n result in the same array. The result is the same as if the array is rotated by $m \ mod \ n$ elements.

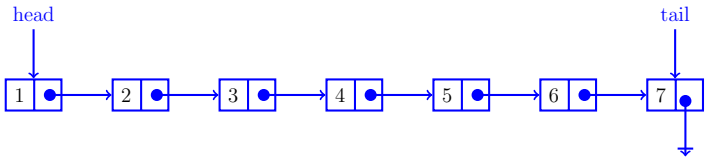
3.3 [3 points] Use C or pseudocode to describe an algorithm, with asymptotic complexity $O(n^2)$, that receives an array A of size n and a positive integer m and rotates A by m elements.

```
1 Algorithm: RotateArray(A, n, m)
2 m = m mod n;
3 for i = 1 to m do
4 |   temp = A[n];
5 |   for j = n to 2 do
6 |   |   A[j] = A[j-1];
7 |   A[1] = temp;
```

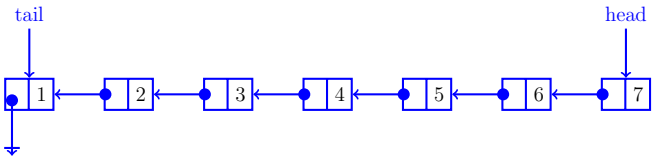
3.4 [3 points] Model array $A = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{bmatrix}$ as a linked list with an explicit pointer to the head and the tail. Define the appropriate data type for the nodes of the list and draw the linked list with its head and its tail.

```
1 struct node { int key; struct node *next; }
2 struct node *head, *tail;
```

Solution 1



Solution 2



3.5 [4 points] Use C or pseudocode to describe an algorithm that rotates the elements by m elements. Determine the asymptotic complexity of your algorithm.

```
1 struct list* head;
2 struct list* tail;
```

../code/ex35.c

Solution 1

```
1  void rotateList1(int m) {
2      int i;
3      int n;
4      struct list* previous;
5      struct list* current;
6
7      current = head;
8      n = 0;
9      while (current != NULL) {
10         n++;
11         current = current->next;
12     }
13     m = m % n;
14
15     for (i=0; i<m; i++) {
16         current = head;
17         while (current->next != NULL) {
18             previous = current;
19             current = current->next;
20         }
21         tail->next = head;
22         tail = previous;
23         head = previous->next;
24         previous->next = NULL;
25     }
26 }
```

../code/ex35.c

Comlexity: $O(n^2)$

Solution 2

```
1  void rotateList2(int m) {
2      int i;
3      int n;
4      struct list* current;
5
6      current = head;
7      n = 0;
8      while (current != NULL) {
9          n++;
10         current = current->next;
11     }
12     m = m % n;
13
14     for (i=0; i<m; i++) {
15         tail->next = head;
16         tail = head;
17         head = head->next;
18         tail->next = NULL;
19     }
20 }
```

../code/ex35.c

Comlexity: $O(n)$

Name: Matriculation number:

Exercise 4 15 Points

Jobs Scheduling

The Hubble Space Telescope, one of the NASA's Great Observatories, is usually shared by a lot of research groups. Suppose that each research group submits a task that Hubble has to accomplish. Each task is defined by its start time (*stime*), its end time (*etime*), and a profit (*profit*) that is the result of completing the task. A scheduler receives a list of tasks that may conflict with each other. The goal is to select which tasks to accomplish so that there are no conflicts (Hubble can perform one task at a time) and the achieved profit is maximized. Tasks are always submitted as an array `tA[0..n-1]` as follows:

```
1  struct task {
2    int stime;
3    int etime;
4    int profit;
5  };
6  struct task tA[n];
```

The tasks in tA are ordered in **ascending order by their end time**.

Example: Assume that `4 tasks` (task0, task1, task2, and task3) are submitted that are represented by the following instantiation of `tA[]`:

	0	1	2	3
stime:	1	2	5	4
etime:	2	6	19	24
profit:	50	120	200	100

The maximum profit that can be achieved is 250, by performing task0 and task2.

4.1 [3 points] Use C or pseudocode to define the function

```
int latestNonConflictTask(struct task tA[], int taskNo)
```

that gets as input the array with the submitted tasks (`tA[]`) and a task number (`taskNo`), and returns the number of the latest task (i.e., the task with the maximum *etime*) that ends before or at the same time that the task with number `taskNo` starts.

```
1 int latestNonConflictTask(struct task tasksArray[], int taskNo) {
2     int j;
3
4     for (j=taskNo−1; j ≥0; j−−)
5         if (tasksArray[j].end ≤tasksArray[taskNo].start)
6             return j;
7     return −1;
8 }
```

../code/schedule_jobs.c

4.2 [3 points] Assume $P(n)$ is the maximum profit achieved when the first n tasks are taken into consideration. Give a recursive definition of $P(n)$.

$$P(n) = \begin{cases} 0, & \text{if } n = 0 \textbf{ or } l = -1 \\ max(P(l) + tA[n-1].profit, P(n-1)), & \text{if } n \geq 1 \textbf{ and } l \geq 0 \\ max(tA[n-1].profit, P(n-1)), & \text{if } n \geq 1 \textbf{ and } l = -1 \end{cases}$$

where $l = \text{latestNonConflictingTask}(n-1)$

4.3 [3 points] Use C or pseudocode to recursively compute the maximum profit that can be achieved, given a list of tasks (`tA[]`) and the number of tasks to be considered (`n`).

```
1  int findMaxProfit(struct task tasksArray[], int n) {
2      int l;
3      int inclProfit, exclProfit;
4
5      if (n==0)
6          return 0;
7
8      l = latestNonConflictTask(tasksArray, n−1);
9      if (l==−1)
10         inclProfit = tasksArray[n−1].profit;
11     else
12         inclProfit = tasksArray[n−1].profit + findMaxProfit(tasksArray, l+1);
13
14     exclProfit = findMaxProfit(tasksArray, n−1);
15
16     return max(inclProfit, exclProfit);
17 }
```

../code/schedule_jobs.c

4.4 [3 points] An efficient computation of the maximum profit and the actual scheduling using dynamic programming is based on array `int P[n]` and matrix `boolean sol[n, n]`, where `P[i]` is the maximum profit achieved when the first $i+1$ tasks are taken into consideration and `sol[j][i]` corresponds to whether task `j` is scheduled when the first $i+1$ tasks are taken into consideration or not. Complete the following tables for the following submitted tasks.

tA[]:

	0	1	2	3
stime:	1	3	4	4
etime:	3	10	15	18
profit:	15	20	30	10

P[]:

i	**0**	**1**	**2**	**3**
-	15	35	50	50

sol[][]:

j\i	**0**	**1**	**2**	**3**
0	1	1	0	0
1	0	1	0	0
2	0	0	1	1
3	0	0	0	0

4.5 [3 points] Given the array of tasks (`tA[]`) and the number (`n`) of tasks to be considered use dynamic programming to compute the maximum profit that can be achieved.

```
1  int findMaxProfitDP(struct task tasksArray[], int n) {
2      int i,l,j;
3      int inclProf, exclProf;
4      int profit[n];
5
6      profit[0] = tasksArray[0].profit;
7
8      for (i=0; i<n; i++)
9      {
10         inclProf = tasksArray[i].profit;
11         l = latestNonConflictTask(tasksArray, i);
12         if (l != -1)
13             inclProf += profit[l];
14
15         exclProf = profit[i-1];
16         profit[i] = max(inclProf, exclProf);
17     }
18
19     return profit[n-1];
20 }
```

../code/schedule_jobs.c