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 II — Final Exam
Spring 2020 — 27.05.2020

Name: _____ Matriculation number: _____

Address: _____ Date of Birth: _____

Advice

You have 90 minutes to complete and submit the Final exam of Informatics II. This is an open book exam.

Submit your solution in one of the following ways:

1. You can print the pdf file, use the available whitespace to fill in your solution, scan your solution, and upload the pdf file to OLAT.
2. You can use blank white paper for your solutions, scan the sheets, and upload the pdf file to OLAT. Put your name and matriculation number on every sheet. State all task numbers clearly.
3. You can use a tablet and pen (iPad, Surface, etc) to fill in your solution directly into the pdf file and upload the completed pdf file to OLAT.
4. You can use a text editor to answer the questions and submit the document as pdf.

Notes:

- If you do not have scanner it is possible to take pictures of your solution with your phone. We recommend Microsoft Office Lens or Camscanner. Create a pdf file that includes all pictures and submit a single pdf file.
- There is no extra time for scanning and submission. The allotted time already includes the time for scanning and submission.
- Only submissions through OLAT are accepted. Submissions through email are only considered if OLAT is not working.

Signature:

Correction slot — Please do not fill out the part below

Exercise	1	2	3	4	Total
Points Achieved					
Maximum Points	17	25	21	22	85

Exercise 1 — 17 Points

Complete the boxes by checking the correct checkbox or by filling your answer into the empty box.

1.1 [1 point] Assume chaining is used to resolve collisions for a hash table of size m that stores N elements with unique keys. The worst-case asymptotic time complexity to remove an item from the hash table is $O(N)$.

Answer: ☒ True ☐ False

1.2 [1 point] Assume $f(n) = O(N^2)$ and $g(n) = O(N^3)$. From these complexities it follows that $f(n) + g(n) = O(N^3)$.

Answer: ☒ True ☐ False

1.3 [1 point] The asymptotic time complexity to search an element in a binary search tree with N nodes is $O(\log N)$.

Answer: ☐ True ☒ False

1.4 [1 point] In a max-heap the depth of any two leaves differs by at most 1.

Answer: ☒ True ☐ False

1.5 [1 point] A hash table guarantees a constant lookup time.

Answer: ☐ True ☒ False

1.6 [4 points] Assume a hash table $T[0...9]$ with open addressing is used to store integer keys. Draw the hash table after the keys 18, 34, 88, 66, 70, 80, and 76 have been inserted (in that order). Illustrate all details of the hash table. Assume that conflicts are resolved with double hashing defined as follows:

$$h(k,i) = (h_1(k) + i * h_2(k)) \ mod \ 10$$
$$h_1(k) = (k \ mod \ 10)$$
$$h_2(k) = \lfloor k/10 \rfloor$$

Slot	0	1	2	3	4	5	6	7	8	9
Value	70		66	76	34		88		18	
Status	OCC	EMP	OCC	OCC	OCC	EMP	OCC	EMP	OCC	EMP

1.7 Consider the red-black tree in Figure 1a where black nodes are denoted with a circle and red nodes are denoted with a square. We consider four operations:

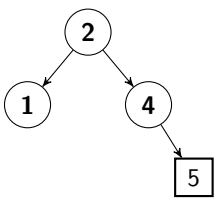
(a) Creating a new node (left and right are set to NULL):
Example: create node 9: `create(9)`

(b) Setting a property (color, key, left, right) of a node:
Example: set the key of node 9 to 5: `9->key = 5`

(c) Rotating a node:
Example: right rotate node 5: `RightRotate(5)`

(d) Deleting a node:
Example: delete node 9: `delete(9)`

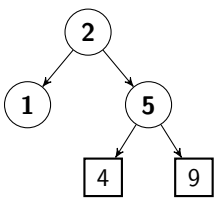
The result of applying the operations in Figure 1b to the red-black tree in Figure 1a yields the red-black tree in Figure 1c.



(a) Before operations

```
create(9)
9->color = red
5->right = 9
5->color = black
4->color = red
LeftRotate(4)
```

(b) Operations



(c) After operations

Figure 1: Tree before and after operations

Use the example notation illustrated in Figure 1b to work out your solutions for the following tasks and the red-black tree in Figure 2.

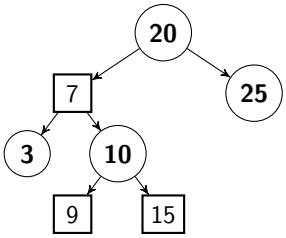
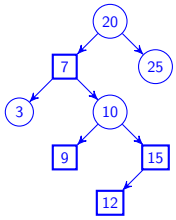


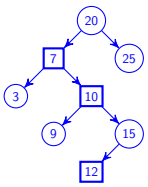
Figure 2: Red-black tree

(a) [5 points] State the operations that are required to insert **12** into the red-black tree in Figure 2

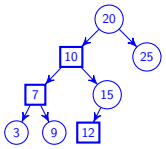
```
create(12)
15->left = 12
12->color = red
```



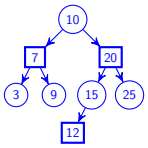

```
9->color = black
15->color = black
10->color = red
```




```
LeftRotate(7)
```



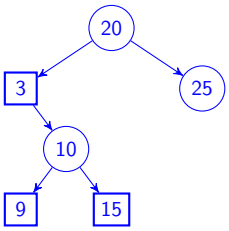

```
10->color = black
20->color = red
RightRotate(20)
```

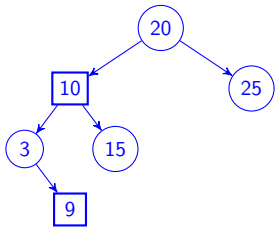


(b) [3 points] State the operations that are required to delete **7** from the red-black tree in Figure 2.
Note: If you have the option to choose a node from either the left or right subtree choose the node from the left subtree.

```
delete(3)
7->key = 3
```

```
10->color = red
3->color = black
15->color = black
LeftRotate(3)
```





Exercise 2 — 25 Points

Consider the C code fragment in Figure 3 that defines a structure for a linked list. Each node has a field n that points to the next node in the list, a field p that points to the previous node in the list, and a field k with the key of the node. Variable h points to the head (first node) of the list and variable t points to the tail (last node) of the list. On the right side in Figure 3 an example list is shown.

```
struct node {
    int k;
    struct node* n;
    struct node* p;
};

struct node *h, *t;
struct node *a, *b;
```

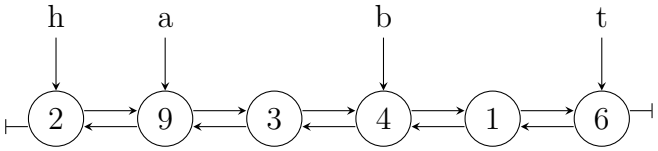


Figure 3: Linked list in C

The task is to develop a C procedure

```
void xchg(struct node* a, struct node* b, ...) {
```

with arguments a and b that point to distinct elements of the list. You may assume that a points to a node that in the list comes before the node b points to. The result of `xchg` is a list that is equal to the original list except that elements a and b have been exchanged. The solution must be implemented by modifying pointers. From within procedure `xchg` no global variable may be accessed.

1. [5 points] State the full signature of `xchg` by completing the header of the above C procedure, i.e., specify all arguments that are required to correctly exchange the elements pointed to by a and b. Show how procedure `xchg` must be called. Assume h, t, a, and b have been initialized appropriately.

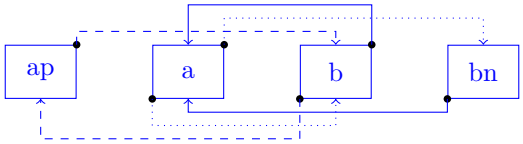
```
void xchg(struct node* a, struct node* b,
          struct node** h, struct node** t) { ... }
```

call: `xchg(a, b, &h, &t);`

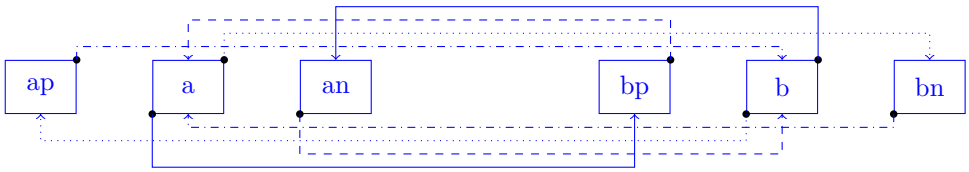
Name: Matriculation number:

2. [4 points] Sketch a picture that illustrates how the pointers must be changed.

   a and b are adjacent:



   a and b are not adjacent:



3. [16 points] State the C code for procedure `xchg`.

```
1  void xchg(struct node* a, struct node* b,
2            struct node** h,struct node** t){
3
4    struct node* ap = a->p;
5    struct node* an = a->n;
6    struct node* bp = b->p;
7    struct node* bn = b->n;
8
9    if (an == b) {
10     b->n = a;
11     a->p = b;
12   } else {
13     an->p = b;
14     bp->n = a;
15     b->n = an;
16     a->p = bp;
17   }
18   if (ap != NULL) {ap->n = b;} else {*h = b;}
19   if (bn != NULL) {bn->p = a;} else {*t = a;}
20   b->p = ap;
21   a->n = bn;
22 }
```

code/linkedlist.c

Name: Matriculation number:

Exercise 3 — 21 Points

Consider a directed graph $G = (V, E)$ with nodes $v \in V$ and $u \in V$. Node u is a **k-hop neighbor** of node v if there is at least one path from v to u and the minimum distance from v to u is k. The distance from v to u is infinite if there is no path from v to u.

Let $|V|$ be the number of nodes in G. Each node is identified by a unique integer between 1 and $|V|$. A directed edge $(v, u) \in E$ is represented as a pair of nodes and denotes an edge from node $v \in V$ to node $u \in V$.

3.1 [2 points] Assume a graph $G = (V, E)$ with vertices $V = \{1, 2, 3, 4, 5, 6\}$ and edges $E = \{(1, 2), (1, 4), (1, 5), (2, 3), (2, 5), (3, 1), (4, 6)\}$. Draw the adjacency matrix of G.

Adjacency Matrix:

	1	2	3	4	5	6
1	0	1	0	1	1	0
2	0	0	1	0	1	0
3	1	0	0	0	0	0
4	0	0	0	0	0	1
5	0	0	0	0	0	0
6	0	0	0	0	0	0

3.2 [4 points] Let $k = 2$. Determine all 2-hop neighbors of node 1? Explain your approach.

- The 2-hop neighbors of node 1 are nodes 3 and 6.
- Run a BFS search until first node in the queue has a distance of k. Dequeue and report nodes as long as they have a distance equal to k.

3.3 [15 points] Assume a directed graph G with n nodes. An adjacency matrix $a[n, n]$ is used to represent the edges of the graph. Give a pseudocode algorithm that prints all k-hop neighbors of node v.

Algorithm: khop(G, v, k)

```
1  for (i = 1; i <= n; i++) do dist[i] = -1;
2  dist[v] = 0;
3  InitQueue(Q);
4  while (v ≠ −1 && dist[v] < k) do
5      for (i = 1; i ≤ n; i++) do
6          if (a[v, i] == 1 && dist[i] == −1) then
7              dist[i] = dist[v] +1;
8              Enqueue(Q,i);
9      v = Dequeue(Q);
10 while v ≠ −1 && dist[v] == k do
11     print(v);
12     v = Dequeue(Q);
```

```c
void khop(int n, int v, int k) {
  for (int i = 0; i < n; i++) { dist[i] = -1; }
  h = 0; t = 0; dist[v] = 0;
  while (v != -1 && dist[v] < k) {
    for (int i = 0; i < n; i++){
      if (a[v][i] == 1 && dist[i] == -1) {
        dist[i] = dist[v] + 1;
        enqueue(i);
      }
    }
    v = dequeue();
  }
  while (v != -1 && dist[v] == k) {
    printf("%d ", v);
    v = dequeue();
  }
  printf("\n");
}
```

code/khop.c

Exercise 4 — 22 Points

Consider a matrix M with integer-valued elements. The task is to compute the length of the side of the largest square that only includes negative numbers. The idea for the solution is to compute an auxiliary matrix S with the same dimensions as matrix M. The value of element $S[i, j]$ is the length of the largest square whose lower right corner is located at position (i, j) in M. Figure 4 shows an example with a 3×3 matrix M_0 and the corresponding solution matrix S_0.

	0	1	2
0	1	-1	6
1	-3	-2	-1
2	-5	-1	1

Matrix M_0

	0	1	2
0	0	1	0
1	1	1	1
2	1	2	0

Solution S_0 for M_0

Figure 4: Illustration of input matrix M and solution matrix S

Note that in order to compute the largest square with the lower right corner at position (i, j) it is sufficient to consider the solutions for the squares with lower right corners at positions $(i - 1, j)$, $(i, j - 1)$ and $(i - 1, j - 1)$.

4.1 [4 points] Consider Figure 5 with the 4×4 matrix M. Complete the corresponding solution matrix S.

-1	0	-1	0
-1	0	-1	-1
-1	-1	-1	0
-1	-1	-1	0

Matrix M

1	0	1	0
1	0	1	1
1	1	1	0
1	2	2	0

Solution S for M

Figure 5: Input matrix M and solution matrix S

4.2 [6 points] Provide a recursive problem formulation for determining the side length of the largest square in matrix M that includes only negative integers.

$$S[i,j] = \begin{cases} 0 & \text{M[i,j]} \geqslant 0 \\ min(min(S[i-1,j-1], S[i-1,j]), S[i,j-1]) + 1 & M[i,j] < 0 \end{cases}$$

4.3 [12 points] Write a C function `int maxLen(int m)` that computes and returns the length of the largest square in an $m \times m$ matrix M that includes negative integers only.

```
1  int maxLen(int m) {
2    int maxSize = 0;
3    for (int i = 0; i < m; i++) {
4      for (int j = 0; j < m; j++) {
5        if (i == 0 || j == 0) {
6          if (M[i][j] < 0) {
7            S[i][j] = 1;
8            maxSize = 1;
9          }
10         else { S[i][j] = 0; }
11       } else {
12         if (M[i][j] < 0) {
13           S[i][j] = 1 + min( min(S[i-1][j-1], S[i][j-1]),
14                              S[i-1][j] );
15           maxSize = max(maxSize, S[i][j]);
16         }
17       }
18     }
19   }
20   return maxSize;
21 }
```

code/dp2.c