Department of Informatics

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

University of Zurich[UZH]

Informatics II — Final Exam
Spring 2021 — 02.06.2021

Name: _____ Matriculation number: _____

Address: _____ Date of Birth: _____

About Exam

You have 90 minutes to answer the questions; you have 20 minutes to download the exam questions and submmit your solutions through EPIS. Only submissions through EPIS are accepted, and only PDF files are accepted.

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 EPIS.
2. You can use blank white paper for your solutions, scan the sheets, and upload the pdf file to EPIS. 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 EPIS.
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.
- The 20 minutes include downloading exams, preparing your solutions for your submission, and submitting the PDF files through EPIS.
- We suggest that you submit your solutions several minutes earlier than the deadline.
- You bear the risk for your last-minute submission.

Signature:

Correction slot — Please do not fill out the part below

Exercise	1	2	3	4	Total
Points Achieved					
Maximum Points	11	12	20	22	65

Exercise 1 — 11 Points

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

1.1 [1 point] The worst-case asymptotic time complexity to search an integer in a binary search tree of N integers is O(N).

Answer: ☒ True ☐ False

1.2 [1 point] The worst-case asymptotic time complexity to search an integer in a red-black tree of N integers is O(log N).

Answer: ☒ True ☐ False

1.3 [1 point] The worst-case asymptotic time complexity to search an integer in a max-heap of N integers is O(log N).

Answer: ☐ True ☒ False

1.4 [1 point] Assume $f_1(n) = O(1)$, $f_2(n) = O(N^2)$, and $f_3(n) = O(N \log N)$. From these complexities it follows that $f_1(n) + f_2(n) + f_3(n) = O(N \log N)$.

Answer: ☐ True ☒ False

1.5 [1 point] The master method applies to calculate the asymptotic time complexity of binary search.

Answer: ☒ True ☐ False

1.6 [6 points] 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 use the key to represent a node. For example, 2 represents the node with key 2. We consider four operations:

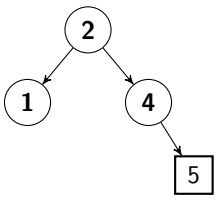
(a) Creating a new node (left and right are set to NULL):
Example: create node 9: `create(9)`

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

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

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

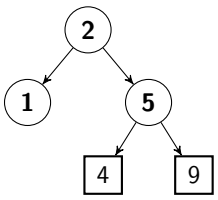
The result of applying the operations in Figure 1b to the red-black tree in Figure 1a yields the red-black tree in Figure 1c.



(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.

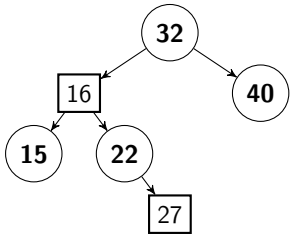
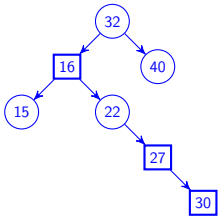


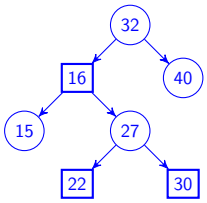
Figure 2: Red-black tree

(a) [3 points] State the operations that are required to insert **30** into the red-black tree in Figure 2.

```
create(30)
27->right = 30
30->color = red
```

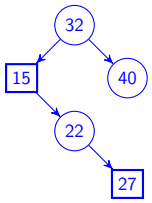
```
LeftRotate(22)
27->color = black
22->color = red
```



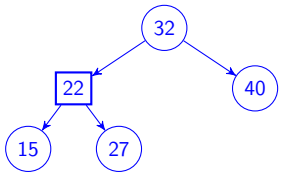


(b) [3 points] State the operations that are required to delete **16** from the red-black tree in Figure 2.
Note: If you have the option to choose a node from either the left or right subtree choose the node from the left subtree.

```
delete(15)
16->key = 15
```




```
LeftRotate(15)
22->color = red
15->color = black
27->color = black
```



Exercise 2 — 12 Points

Consider a hash table `HT` consisting of m slots, using open addressing **with linear probing and a step of size 1**. Write a C code for the function `int HTDelete(int k)`, which deletes key k from the hash table and also rearranges the other elements. The rearrangement leaves the table exactly as it would have been had key k never been inserted i.e., you cannot simply set the status as deleted.

The Hash table `HT` is defined as follows:

```
1   struct elem{
2     int key;
3     int status;    // 0:OCCUPIED, -1: EMPTY
4   };
5
6    struct elem HT[m];
```

The following example shows the state of a hash table after calling `HTDelete(11)`. The hash table uses linear probing with step of size 1 and a hash function $h(k) = k\%m$, with m=5.

Slot	Status	Key
0	-1	-1
1	0	11
2	0	22
3	0	31
4	0	2

After Deleting 11 ⟹

Slot	Status	Key
0	-1	-1
1	0	31
2	0	22
3	0	2
4	-1	-1

Table 1: Illustration of function call `HTDelete(11)` on the hash table.

You can use the following helper functions:

1. `int hash(int k,int i)` - return the hash slot for key k and step of size i.
2. `int HTInsert(int k)` - inserts key k in the hash table HT and return the slot number. Returns -1 if hash table is full.

Note: Initially all slots are empty and keys are initialized with -1. The function `HTDelete(int k)` returns the slot number of a deleted key k and -1 if key k does not exist in the hash table.

```
1  int HTDelete(int k){
2    int i=0;
3    int probe = hash(k,i);
4    int actualHashIdx= probe;
5    while(i<m && HT[probe].status == 0 && HT[probe].key!=k){
6      i++;
7      probe = hash(k,i);
8    }
9    if(i>=m || HT[probe].status==-1) return -1;
10   HT[probe].status = -1;
11   HT[probe].key = -1;
12   int j=(probe+1)%m;
13   while(j!=actualHashIdx && HT[j].status!=-1){
14     if(hash(HT[j].key,0)!=j){
15       int tmpKey = HT[j].key;
16       HT[j].key = -1;
17       HT[j].status = -1;
18       HTInsert(tmpKey);
19     }
20     j = (j+1)%m;
21   }
22   return probe;
23 }
```

code/Hash.c

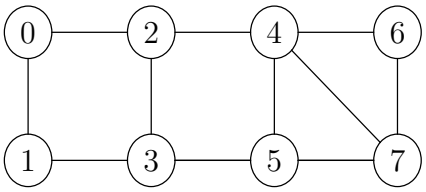
Name: Matriculation number:

Exercise 3 — 20 Points

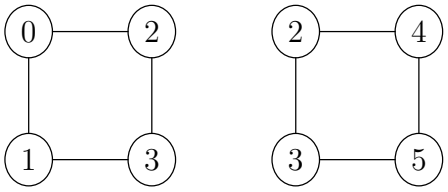
Consider an undirected graph $G = (V, E)$ with vertices V and edges E. A BiQuad subgraph $B = (X, Y)$ is an induced subgraph of a graph G formed from a subset of the vertices of the graph G and all of the edges connecting pairs of vertices in that subset. The vertices X is a subset of vertices V, that is $X \subseteq V$, and edges Y is a subset of edges E, that is $Y \subseteq E$, such that the following conditions hold.

1. A BiQuad subgraph only consists of four vertices i.e. $|X| = 4$, connected to each other.
2. Each vertex is connected to exactly two vertices in the BiQuad graph.

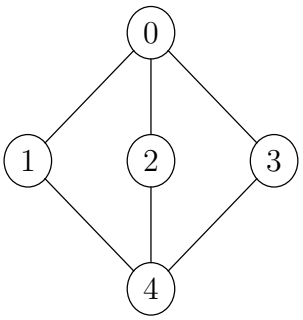
Let $|V|$ be the number of nodes in G. Each node is identified by a unique integer between 0 and $|V| - 1$. Consider the following graph G.

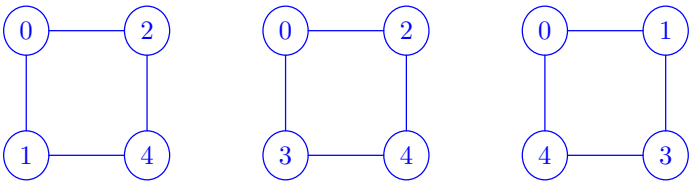


The graph G contains two BiQuad Subgraphs:



3.1 [3 points] State the BiQuad Subgraphs of the following graph G.





3.2 [17 points] Assume an undirected graph G with n vertices. An adjacency matrix $graph[n, n]$ is used to represent the edges of the graph. Give a C code that determines and prints all BiQuad subgraphs of graph G.

```
1  void BiQuad(int graph[N][N],int s) {
2    int i=0,u=0,k=0,q=0, neighbors[N],tmpN=-1;
3    for (i = 0; i < N; i++) { dist[i] =
        -1;color[i]=0;pred[i]=-1;neighbors[i]=-1;}
4    dist[s] = 0; color[s] = 1;
5    enqueue(s);
6    while(!isEmpty()){
7      int v = dequeue();
8      int count = 0,tmpN=-1;
9    for(u=0;u<N;u++){
10       if(graph[v][u]==1){
11         if(color[u] == 0){
12           color[u]=1; dist[u] = dist[v]+1; pred[u]=v;
13           enqueue(u);
14       }else if(color[u]==2 && dist[u]<=dist[v]){
15         neighbors[u]=1;
16         count=count + 1;
17         tmpN=u;
18       }
19     }
20   }
21   if(count>1){
22     for(u=0;u<N;u++){
23       for(q=u+1;q<N;q++){
24         if(neighbors[u]==1 && neighbors[q]==1){
25           int fourthVertex = -1, distDiff = dist[q]-dist[u];
26           if(distDiff==0 || distDiff==1 || distDiff==-1){
27             if(graph[u][pred[q]]==1){
28               fourthVertex = pred[q];
29             }else if(graph[q][pred[u]]==1){
30               fourthVertex = pred[u];
31             }
32           }
33           if(fourthVertex!=-1){
34             printf("BiQuad Subgraph: %d, %d, %d, %d
                   \n",fourthVertex,q,v,u);
35           }
36           if(distDiff == 0){
37             for(k=0;k<N;k++){
38               if(k!=pred[q] && color[k]==2 && graph[q][k]==1 &&
                     graph[u][k]==1){
39                 printf("BiQuad Subgraph: %d, %d, %d, %d
                       \n",k,q,v,u);
40               }
41             }
42           }
43         }
44       }
45       neighbors[u]=-1;
46     }
```

```
47    }else if(count == 1){
48       neighbors[tmpN] = -1;
49    }
50    color[v]=2;
51    }
```

code/Graph.c

Exercise 4 — 22 Points

Consider a matrix M with integer-valued elements. The task is to compute the number of zeros in a rectangle of M. The rectangle is defined by the upper left coordinate $(0, 0)$ and lower right coordinate (i, j). Both coordinates are **included** in the rectangle. 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 number of zeros in the rectangle whose upper left is $(0, 0)$ and lower right corner is (i, j) in M.

	0	1	2
0	0	-2	0
1	0	0	-9
2	25	0	1

Matrix M_0

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

Solution S_0 for M_0

Figure 3: Illustration of input matrix M and solution matrix S

4.1 [3 points] Consider Figure 4 with the 3×4 matrix M. Complete the corresponding solution matrix S.

12	0	4	-5
11	9	0	3
0	-4	0	8

Matrix M

0	1	1	1
0	1	2	2
1	2	4	4

Solution S for M

Figure 4: Input matrix M and solution matrix S

4.2 [14 points] Write a C function `void computeS(int m, int n)` that computes the matrix S for an $m \times n$ matrix M. Both M and S have been initialized, and you can directly use them in your solution.

```
1  void computeS(int m, int n) {
2    if(M[0][0] == 0) {
3      S[0][0] = 1;
4    }
5    for (int i = 1; i < n; i++) {
6      if(M[0][i] == 0) {
7        S[0][i] = S[0][i - 1] + 1;
8      } else{
9         S[0][i] = S[0][i - 1];
10     }
11   }
12   for (int i = 1; i < m; i++) {
13     if (M[i][0] == 0) {
14       S[i][0] = S[i - 1][0] + 1;
15     } else {
16        S[i][0] = S[i - 1][0];
17     }
18   }
19   for (int i = 1; i < m; i++) {
20     for (int j = 1; j < n; j++) {
21       if (M[i][j] == 0) {
22         S[i][j] = S[i - 1][j] + S[i][j - 1] - S[i - 1][j - 1] +
               1;
23       } else {
24         S[i][j] = S[i - 1][j] + S[i][j - 1] - S[i - 1][j - 1];
25       }
26     }
27   }
28 }
```

code/dp.c

4.3 [5 points] Consider that the upper left coordinate is an arbitrary coordinate (a, b). Provide a recursive problem formulation for determining $n(a, b, i, j)$ that is the number of zeros contained in the rectangle whose upper left coordinate is (a, b) and lower right coordinate is (i, j) using S.

$$n(a,b,i,j) = \begin{cases} S[i,j] - S[i][b-1] & \texttt{a = 0 and b > 0} \\ S[i,j] - S[a-1,j] & \texttt{a > 0 and b = 0} \\ S[i,j] - S[a-1,j] - S[i][b-1] + S[a-1][b-1] & \texttt{a > 0 and b > 0} \end{cases}$$