TU Berlin, February 11, 2020

Name: ..... **Matr.-Nr.: .....**

Advanced Algorithmics Test Exam

Exercise No.:	1	2	3	4	5	6	7	8	**Sum**
Points:	20	12	20	20	16	20	20	12	140
Achieved:									

Time limit: 120 Minutes

Max. number of points: 140 Points
Best grade: $\geq$ 86 Points

General hints:

- You are not allowed to use any technical aids or learning material during the exam.
- Do not use a pencil. Use a black or blue pen (non-erasable).
- Write your name and matriculation number on each sheet.
- **If not explicitly excluded in the task, then all answers must be justified! Answers without justification receive 0 points.**

We wish you success!

Name: ..... **Matr.-Nr.:** .....

Task 1: **Integer Linear Programming** (20 Points)

Provide integer linear programming (ILP) formulations for the two following problems (without justification).

To this end, define the variables, all constraints, and the objective function (maximization or minimization).

(a) **Clique** (10 P)

Input: An undirected graph $G = (V, E)$.

Task: Find a largest clique in G, that is, find a largest vertex subset $V' \subseteq V$ such that every two distinct vertices in V' are adjacent.

(b) k-**Plex** (10 P)

Input: An undirected graph $G = (V, E)$.

Task: Find a largest k-plex in G, that is, find a largest vertex subset $V' \subseteq V$ such that every vertex in V' is adjacent to all but at most k vertices in V'.

Hint: A 1-plex is a clique.

Name: ..... **Matr.-Nr.: .....**

Task 2: **Online Algorithms** (12 Points)

Consider a two-level memory system that consists of a fast memory (called cache) that can hold k memory pages and of an arbitrarily large slow memory.

Each memory request of an application specifies a page number in the memory system. A request is *served* if the corresponding page is in the cache. If a requested page is not in the cache, then a *cache miss* occurs and a page must be moved from the cache to the slow memory so that the requested page can be loaded into the free location of the cache.

Our goal is to minimize the number of occurring cache misses when serving a sequence of memory page requests that appear in an online fashion. To this end, consider the following strategy:

On a cache miss, remove the page in fast memory that was first loaded. (If a page is loaded more than once into the fast memory, then always the last loading time is memorized. If the page is already in the cache, then it is not loaded again and the loading time is *not* updated!)

Show that this strategy is **not** 3-competitive.

Hint: Assume that both an optimal strategy and the strategy above start with the first k requested pages already in fast memory.

Definition. Let $A(\sigma)$ be the cost incurred by online algorithm A and $\mathrm{OPT}(\sigma)$ be the cost incurred by an optimal offline algorithm. Then, A is called ***c*-competitive** if there is a constant d such that $A(\sigma) \leq c \cdot \mathrm{OPT}(\sigma) + d$ for all σ.

Name: ..... **Matr.-Nr.:** .....

Task 3: **Circular String Linearization** (20 Points)

(a) Compute the suffix array and the longest common prefix array of the string (6 P)

$$S = \texttt{mississippi}.$$

(b) We say that for some $0 \leq i \leq n-1$, the *cyclic shift* S_i of a string S of length n is the string $S_i = S[i, n-1] \circ S[0, i-1]$ if $i > 0$ and otherwise $S_0 = S$. (Here, $\circ$ denotes the concatenation and $S[i, j]$, $0 \leq i \leq j < n$, is the substring of S starting at position i and ending at position j.) Consider the following problem: (14 P)

Circular String Linearization
Input: A string S.
Task: Find a lexicographically first cyclic shift of S.

Describe a linear-time algorithm for this problem. You can use a suffix array and a longest common prefix array and assume that you have an algorithm to compute them in linear time.

Hint: The definitions of suffix array and longest common prefix array are as follows:

Definition. A **suffix array** for a string S of length n is an array $A[0], \ldots, A[n-1]$ that lists starting positions of suffixes of S in lexicographical order. That is,

$$\forall 0 \leq i < n-1 : S[A[i], n-1] <_{\text{lex}} S[A[i+1], n-1].$$

Definition. A **longest common prefix (LCP) array** L for a suffix array A of a string S stores the length of the longest common prefix between each two consecutive suffixes in the suffix array. That is,

$$\forall 1 \leq i < n : L[i] = \arg\max_x \; S[A[i-1], A[i-1]+x-1] = S[A[i], A[i]+x-1],$$

where $S[i, i-1]$ is defined as the empty string for all i. Furthermore, $L[0]$ is set to some default value, e.g. zero.

Name: ..... **Matr.-Nr.:** .....

Task 4: **Independent Set Approximation** (20 Points)

Consider the following algorithm for Independent Set which is defined as follows.

Independent Set

Input: An undirected graph $G = (V, E)$.

Task: Find the largest independent set in G, that is, find the largest vertex subset $V' \subseteq V$ such that no two vertices in V' are adjacent.

Herein, $\deg_{V'}(v)$ is the degree of v and $N_{V'}[v]$ is the (closed) neighborhood of v, that is,

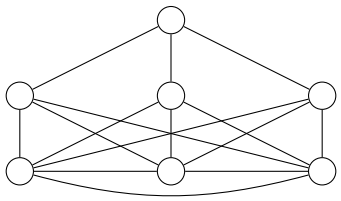
$$N_{V'}[v] = \{u \in V' \mid \{u, v\} \in E\} \cup \{v\} \qquad \text{and} \qquad \deg_{V'}(v) = |N[v]| - 1.$$

Algorithm 1 Algorithm for Independent Set.

```
1: K ← ∅; V' ← V;
2: while V' ≠ ∅ do                                   ▷ Main routine
3:     u ← arg min_{v∈V'} {deg_{V'}(v)};
4:     V' ← V' \ N[u];
5:     K ← K ∪ {u};
6: return K;
```

If there is a tie in line 4, then the algorithm chooses a vertex with minimum degree at random.

(a) Mark one possible solution (the set K) that might be produced by Algorithm 1 in the graph below (without justification). (6 P)



(b) Disprove that Algorithm 1 is a 0.5 approximation even if the algorithm breaks all ties optimally. That is, provide an example graph where Algorithm 1 produces a set K of less than half the size of an optimal independent set, even if all ties are resolved in the best possible way. (14 P)

Name: ..... **Matr.-Nr.:** .....

Task 5: **Massive Data: Matrix Multiplication** (16 Points)

Assume that you have an internal memory of size M, an external memory of size at least $N \cdot B$, and a block size B, where $B^2 < M < N$. Moreover, assume that a pointer to a cell in the external memory can be stored in one cell. As in the lecture, we can only read and write directly into the internal memory. In addition, we can copy a block of B cells from the internal memory to the external memory, as well as we can copy a block of B cells from the external memory into the internal memory. Each copy operation is one I/O operation.

The following procedure is an implementation of the school method for matrix multiplication. The integer in row $i \in \{1, \ldots, N\}$ and column $j \in \{1, \ldots, N\}$ of matrix $X \in \mathbb{Z}^{N \times N}$ is denoted by X_{ij}.

```
1: procedure MM(Matrices V, W ∈ Z^{N×N})
2:     for i from 1 to N do
3:         for j from 1 to N do
4:             C_ij ← 0
5:             for k from 1 to N do
6:                 C_ij ← C_ij + V_ik · W_kj
7:     return C
```

(a) Assume that the matrices V, W, and C are stored in a row-by-row layout in the external memory and that the internal memory has an optimal page replacement strategy. Determine the asymptotic number of I/O operations of the matrix multiplication algorithm given above. (8 P)

(b) Assume that the internal memory has an optimal page replacement strategy. Choose an individual layout for V, W, and C in the external memory such that the matrix multiplication algorithm given above needs at most $\mathcal{O}(\frac{N^3}{B})$ I/O operations. (8 P)

Name: ..... **Matr.-Nr.:** .....

Task 6: **Quantum Gates and Circuits** (20 Points)

Let $G = \{\mathbb{I}, X, Y, Z, H\}$ be the set of the following 1-qubit quantum gates:

$$\mathbb{I} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}, Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}, H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}.$$

(a) Give a 1-qubit quantum circuit with three 1-qubit quantum gates from G that on input $|0\rangle$ yields state $-i|0\rangle$ last before measuring. (i denotes the imaginary unit) (8 P)

(b) Give a 2-qubit quantum circuit with two 1-qubit quantum gates from G per qubit that on input $|00\rangle$ yields state $|\psi\rangle = (\psi_1, \psi_2, \psi_3, \psi_4)^T \in \mathbb{R}^4$ last before measuring such that (12 P)

- $\sum_{i=1}^{4} \psi_i = 0$ and
- each element in $\{|00\rangle, |01\rangle, |10\rangle, |11\rangle\}$ is measured with probability 1/4.

(Measurements are taken with respect to the basis $\{|00\rangle, |01\rangle, |10\rangle, |11\rangle\}$).

Hint: Recall that $|0\rangle = (1,0)^T$, $|1\rangle = (0,1)^T$, $|00\rangle = (1,0,0,0)^T$, $|01\rangle = (0,1,0,0)^T$, $|10\rangle = (0,0,1,0)^T$, $|11\rangle = (0,0,0,1)^T$, and that for state $|\psi\rangle = (\psi_1, \psi_2, \psi_3, \psi_4)^T \in \mathbb{R}^4$ the probability of measuring $|01\rangle$ is $|\phi_2|^2$.

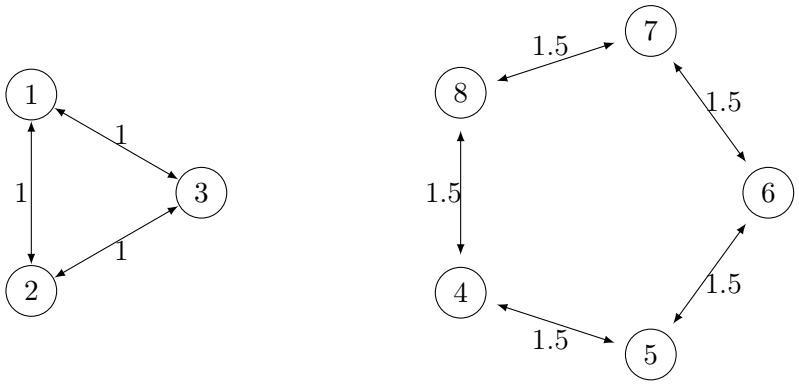
Name: ..... **Matr.-Nr.: .....**

Task 7: **Hedonic Games** (20 Points)

We consider the task of finding a core stable outcome for the following special case of Hedonic Games where agents are placed in the plane and the preferences are based on the average Euclidean distance to the members of the coalition. More precisely, let $a \in A$ be an agent and let X and Y be two coalitions with $a \in X$, $a \in Y$, and $X \neq Y$. Agent a strictly prefers coalition X to coalition Y if the average distance to the agents in X is smaller than the average distance to the agents in Y or if $Y = \{a\}$. Agent a weakly prefers coalition X to coalition Y if the average distance to the agents in X is the same as the average distance to the agents in Y.

Core Stable Partition for Distance-Based Hedonic Games
Input: A set $A \subseteq \mathbb{Q}^2$ of agents positioned in the plane.
Task: Find a core-stable partition of the agents.

(a) Given is the following example instance. (10 P)



Decide whether (i) the core and whether (ii) the strict core is empty. If it is empty, then prove this. If it is non-empty, then provide one element of the core and argue why no blocking coalition exists.

(b) Describe a polynomial-time algorithm that finds a strictly core-stable partition. Analyze the running-time and show the correctness of the algorithm. Can you adapt this algorithm to find a core-stable partition? (10 P)

Hint: Recall that a coalition S is *blocking* if each agent in S weakly prefers S to its current coalition and at least one agent in S strictly prefers S to its current coalition. A coalition S is *strictly blocking* if each agent in S prefers S to its current coalition. A partition of the agents into coalitions is *(strictly) core stable* if there is no (strictly) blocking coalition. The *(strict) core* is the set that contains all (strictly) core stable partitions.

Name: ..... **Matr.-Nr.: .....**

Task 8: **Randomized and Online Algorithms** (12 Points)

Describe in **at most 40 words** the connection between randomized algorithms and online algorithms.