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.

——————Lösung——————

max

$$\sum_{v_i \in V} x_i$$

subject to

$$\begin{aligned} \forall \{v_i, v_j\} \notin E: \quad & x_i + x_j \leq 1 \\ \forall v_i \in V: \quad & x_i \in \{0, 1\} \end{aligned}$$

(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.

——————Lösung——————

max

$$\sum_{v_i \in V} x_i$$

subject to

$$\begin{aligned} \forall v_i \in V: \quad & \sum_{v_j \in V \setminus N[v_i]} x_j \leq k + |V| \cdot (1 - x_i) \\ \forall v_i \in V: \quad & x_i \in \{0, 1\} \end{aligned}$$

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

————Lösung————

FIFO is not 3-competitive:

Consider the following input:

$$(p_1, p_2, \ldots, p_k, p_{k+1})^{c+1}$$

Here, the pages $p_1, \ldots, p_{k+1}$ come one after the other and this sequence of $k+1$ page requests is repeated $c+1$ times. The optimum strategy would evict the following pages (in the following order): $p_k, p_{k-1}, \ldots, p_{k-c-1}$, resulting in more thans c cache misses. Having $p_1, \ldots, p_k$ in the fast memory, FIFO has for every page request starting with p_{k+1} a cache miss. This results in $(k+1)c + 1 = ck + c + 1$ cache misses. The competitive factor is thus $> (ck+c+1)/c = k+1+1/c$. This implies that FIFO is not 3-competitive.

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.

——————Lösung——————

(a)

Suffix Array	Suffix	LCP
10	`i`	0
7	`ippi`	1
4	`issippi`	1
1	`ississippi`	4
0	`mississippi`	0
9	`pi`	0
8	`ppi`	1
6	`sippi`	0
3	`sissippi`	2
5	`ssippi`	1
2	`ssissippi`	3

(b) The following procedure solves **Circular String Linearization**:

- Concatenate input string S with itself: $S' = S \circ S$.
- Compute the suffix array A of S'.
- Iterate over A (from $i = 0$ to $|S'| - 1$) and output the first entry $A[i]$ with $A[i] < |S|$.

Running time: Since we only iterate once over the suffix array A of S', this procedure clearly runs in linear time.

Correctness: First, note since we output a position $A[i]$ with $A[i] < |S|$ the suffix of S' starting at $A[i]$ contains a cyclic shift of S as a prefix.
Assume for contradiction that $S_{A[i]}$ is not a lexicographically first cyclic shift of S and a lexicographically first cyclic shift of S starts at some position $j < |S|$ with $j \neq A[i]$. Then the suffix of S' starting at position j contains a lexicographically first cyclic shift of S as a prefix. Hence, the suffix of S' starting at position j is lexicographically smaller than the suffix of S' starting at $A[i]$. This implies that position j appears earlier in the suffix array A than position $A[i]$—a contradiction to the assumption that we output the first position in the suffix array that is smaller than $|S|$.

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,

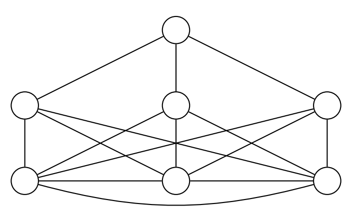
$$N_{V'}[v] = \{u \in V' \mid \{u, v\} \in E\} \cup \{v\} \qquad \text{and} \qquad \deg_{V'}(v) = |N[v]| - 1.$$

Algorithm 1 Algorithm for Independent Set.

1: $K \leftarrow \emptyset$; $V' \leftarrow V$;
2: **while** $V' \neq \emptyset$ **do** ▷ Main routine
3: $\quad u \leftarrow \arg\min_{v \in V'}\{\deg_{V'}(v)\}$;
4: $\quad V' \leftarrow V' \setminus N[u]$;
5: $\quad K \leftarrow K \cup \{u\}$;
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)



——————Lösung——————

The top vertex and exactly one of the vertices in the bottom row.

(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)

——————Lösung——————

Add two vertices two the second and third layer in a similar fashion as the previous three. The algorithm will always choose the top vertex and one of the bottom vertices and the optimal solution would pick all vertices in the second layer. Thus the algorithm computes a solution that is only $\frac{2}{5} = 0.4 < 0.5$ of the optimal solution.

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 $i \in \{1, \dots, N\}$ and column $j \in \{1, \dots, N\}$ of matrix $X \in \mathbb{Z}^{N \times N}$ is denoted by X_{ij}.

procedure MM(Matrices $V, W \in \mathbb{Z}^{N \times N}$)
  for i from 1 to N **do**
    for j from 1 to N **do**
      $C_{ij} \leftarrow 0$
      for k from 1 to N **do**
        $C_{ij} \leftarrow C_{ij} + V_{ik} \cdot W_{kj}$
  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)

——————Lösung——————

First, we determine the number of I/O request required to compute C_{ij}. Observe that we need to sum up the i-th row of V with the j-th column of W. Since V has a row-by-row layout, we need $(\frac{N}{B})$ I/O read operations for the row of V. Since W has a row-by-row layout and $N > B$, we need $\mathcal{O}(N)$ I/O read operations for the column of W. Hence, to compute C_{ij} we have $\mathcal{O}(\frac{N}{B} + N)$ many I/O read operations, and thus $\mathcal{O}((\frac{N}{B} + N)N^2) = \mathcal{O}(N^3)$ in total to compute C. The algorithm writes C in a row-by-row fashion. Therefore, we can flush a block of B entries of C with one I/O write into the external memory.

In total this gives $\mathcal{O}((\frac{N}{B} + N)N^2 + \frac{N^2}{B}) = \mathcal{O}(N^3)$ I/O operations.

(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)

——————Lösung——————

We say V and C have a row-by-row layout, but W a column-by-column layout. Now, we repeat the analysis from above and notice that we need only $(\frac{N}{B})$ I/O read operations for the column of W. Thus in total we have $\mathcal{O}((\frac{N}{B} + \frac{N}{B})N^2 + \frac{N^2}{B}) = \mathcal{O}(\frac{N^3}{B})$ I/O operations.

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 $|\phi_2|^2$.

——————Lösung——————

(to (a).)

$$|0\rangle - Y - Z - X - M-$$

$$XZY|0\rangle = XZi|1\rangle = X(-i|1\rangle) = -i|0\rangle = i^2 \cdot i|0\rangle = i^3|0\rangle$$

(to (b).)

$$|0\rangle - H - \mathbb{I} - M-$$
$$|0\rangle - H - Z - M-$$

$$(\mathbb{I} \otimes Z)(H \otimes H)|00\rangle = (\mathbb{I} \otimes Z)(\frac{1}{2}(|00\rangle + |01\rangle + |10\rangle + |11\rangle))$$
$$= \frac{1}{2}(|00\rangle - |01\rangle + |10\rangle - |11\rangle).$$

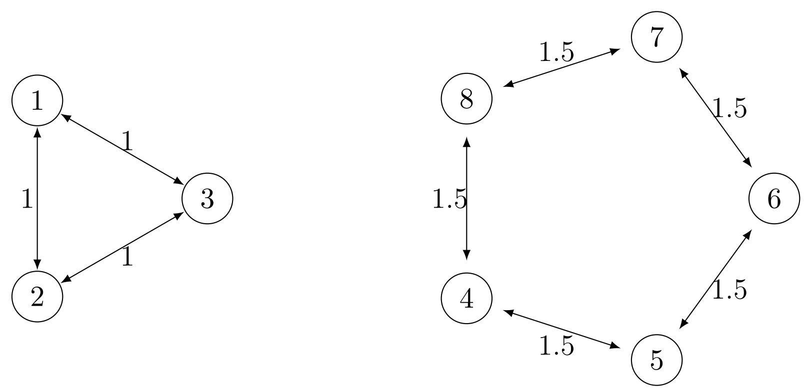
Name: .... **Matr.-Nr.:** ....

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.

——————Lösung——————

(i): The core is empty as each partition either puts three or more candidates from $\{4, 5, 6, 7, 8\}$ into one coalition or there is a candidate $\ell \in \{4, 5, 6, 7, 8\}$ that is not matched with other candidates in this set. Either way there is a candidate $\ell \in \{4, 5, 6, 7, 8\}$ that is either in a coalition $\{\ell\}$ or it is in a coalition that does not consist of closest candidates only. Thus a coalition of ℓ and one of its closest neighbors is blocking as the other one is at least as satisfied and ℓ strictly prefers this new coalition.

(ii): The strict core is not empty. One element in it is $\{1, 2\}, \{3, 4\}, \{5, 6\}, \{7, 8\}$. Observe that there is no possible coalition that $1, 2, 5, 6, 7$ or 8 would strictly prefer over this. Thus, the only two candidates that might want to change their status are 3 and 4. Since the only other option would be to split the coalition and both candidates do not prefer this over the partition above, this partition is strictly core stable.

————————————————————

(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)

——————Lösung——————

Consider the following algorithm:

1. Find two candidates u, v with minimum distance.
2. Build a new coalition $\{u, v\}$ and remove u and v from the input.
3. Repeat steps 1 and 2 until no two candidates are left.
4. If there is a candidate x left, build a new coalition $\{x\}$.

There are $\frac{n}{2}$ repetitions the third step. Steps 1,2 and 4 take $O(n^2)$ time. Thus the algorithm runs in $O(n^3)$ time.

We prove the correctness by induction and contradiction. Assume towards a contradiction that there is a blocking coalition to the partition computed by the algorithm. We make an induction over n. If $n = 0$ or $n = 1$, then there is only one possible partition, so there cannot be another (blocking) one. If $n \geq 2$, let $\{u, v\}$ denote the first coalition selected by the algorithm. Assume that an agent $x \in \{u, v\}$ is part of a strictly blocking coalition X. Thus, the average distance of x to any agent from X is smaller then $d(u, v)$ and the size of the coalition is at least two. This contradicts the fact, that u and v are the closest candidates in the input instance. If neither u nor v is part of a strictly blocking coalition, then we can ignore (remove) them. Thus we have an input instance of size $n - 2$. By induction, there is no strictly blocking coalition for the this smaller instance.

Due to misleading formulations we will not discuss the second part of the question.

————————————————

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.

——————Lösung——————

see lecture slides for multiple answers.

————————————————