TECHNICAL UNIVERSITY OF DENMARK

Written exam, December XX, 20XX

Course name: Computationally Hard Problems

Course No.: 02249

Aids allowed: all aids are allowed

Duration: 4 hours

Weighting: Exercise 1 – 10%, Exercise 2 – 15%, Exercise 3 – 10%,
Exercise 4 – 20%, Exercise 5 – 15%, Exercise 6 – 10%, Exercise 7 – 20%.

All exercises should be answered by filling out the boxes below the statement of the assignment or ticking boxes in the multiple-choice exercise. As your solution to the exam, just hand in the page with your answer and the following pages. If you need more space, you may use extra pieces of paper and hand these in along with your solution.

Name: Arianna Taormina

Student id: s16smth

Exercise 1: An integer program (IP) consists of an objective function

$$c_1 x_1 + \cdots + c_m x_m$$

and a number of constraints

$$a_{j1} x_1 + \cdots + a_{jm} x_m \ \leq \ b_j, \quad \text{for } j = 1, \ldots, k,$$

where $c_1, \ldots, c_m \in \mathbb{Z}$, and $a_{j1}, \ldots, a_{jm}, b_j \in \mathbb{Z}$, for $j = 1, \ldots, k$.
The task is to design a language L_{IP} for integer programs.

a) Specify the alphabet Σ_{IP} you use.

Integers, therefore: $\Sigma_{\text{hg}} = \{0, 1, \ldots, 9, -, \#\}$

b) Specify how the language L_{IP} is defined.

We list:

$$\#m\#k\#c_1\#, \ldots, \#c_m\#a_{1,1}\#, \ldots, \#a_{1,m}, \#b_1\# \ldots \#a_{k,1}\#, \ldots, \#a_{k,m}, \#b_k\#$$

- m is the number of coefficients : one integer
- k is the number of constraints : one integer
- m coefficients c of the objective function : m integers
- m a´s plus 1 b for k times : (m+1)*k

c) Describe how one can check whether a given word $w \in \Sigma^*_{\text{IP}}$ is in L_{IP}, and, if so, how the integer program can be reconstructed.

For a word $w \in \Sigma^*_{\text{hg}}$ to be in L_{hg} we need to check the following:

- has to have length equal to 2 integers + m coefficients + (m+1)*k integers $|x| = 2 + m + (m+1) * k$
- integers are separated with #
- there are no two symbols # close to each other
- m and $k > 0$

If any of the above fails then we output No, otherwise Yes.

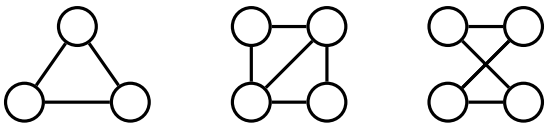
Exercise 2: Consider the following problem.

> **Problem** [SubgraphIsomorphism]
> **Input:** Two undirected graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$. **Output for the decision version:** YES if there exists a subgraph $G' = (V', E')$ of G_1 induced by some subset $V' \subseteq V_1$ such that the graphs G' and G_2 are isomorphic, and NO otherwise. **Output for the optimization version:** A subgraph G' as described above if it exists and NO otherwise.

Definition of possibly unknown terms:

- Two undirected graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are called *isomorphic* if $|V_1| = |V_2|$ and there is an injective mapping $f\colon V_1 \to V_2$ such that for all $v, w \in V_1$ it holds that $\{v, w\} \in E_1 \iff \{f(v), f(w)\} \in E_2$. Informally, this means that the two graphs are identical after re-naming of the nodes.
- Given a subset $V' \subseteq V_1$, the *subgraph* $G' = (V', E')$ of G_1 *induced by* V' satisfies $E' := \{\{v, w\} \in E_1 \mid v \in V' \wedge w \in V'\}$, i. e., it contains all edges that are incident on nodes in V'.

Example: the graph in the middle has two subgraphs (induced by certain vertex subsets) isomorphic to the graph on the left; the graph on the right has no such induced subgraphs.



What you have to do: Show how to convert an algorithm A_d for the decision version of SubgraphIsomorphism into a polynomial-time algorithm A_o for the optimization version of the problem. See next page for details.

a) Describe an algorithm A_o which solves the optimization problem as described above. The running time of A_o has to be polynomial in the input size and A_o may make calls to A_d. Recall that such a call counts as one step.

We use A_d as a subroutine for A_o.

Algorithm 1 [Optimization Algorithm]

```
1: identify number k* ( binary or linear search)
2: call A_d(G_1, G_2)
3: if NO then   return NO
4: else
5:     for vertex in G_1 do
6:         take out one vertex with all its edges and Call A_d(G'_1, G_2)
7:         if NO then   put the vertex and its edges back, mark them has explored
8:         else continue to (5)
```

The algorithm returns the vertices and edges that are isomorphic to G_2

b) Argue that your algorithm is correct.

If the result of the call to A_o is YES we continue examining the vertices of G. We need exactly V_2 vertices, while G_1 has more vertices than V_2 the algorithm keeps on running. The algorithm excludes vertices that are not part of the isomorphic subset.

c) Show the polynomial running time of your algorithm.

- G_1 has n_1 vertices and m_1 edges
- G_2 has n_2 vertices and m_2 edges

For each node in G_1 we exclude and compare to G_2 to check if there is an isomorphism, via A_d that is called in constant time. Therefore the total running time is: $O((n_1 + m_1 + n_2 + m_2) * n)$

Exercise 3: Consider the following problem:

Problem [BinPacking]
Input: A sequence $s_1, s_2, \ldots, s_n$ of rational numbers, all being greater than 0 and less than 1, and a natural number B.
Interpretation: We want to pack n objects with sizes $s_1, s_2, \ldots, s_n$ into as few bins as possible. Every bin has a capacity of 1. The objects can be be packed into the bins such that there is no space between them. The objects cannot be divided. The sum of the sizes of the objects in a certain bin cannot exceed 1.
Output: YES if the objects can be packed into at most B bins and NO otherwise.

Prove that BinPacking is in the class $\mathcal{NP}$. We suggest the following structure for your proof:

1) Design a deterministic algorithm $A(\boldsymbol{X}, R)$ (in the following just called A) which takes as input a problem instance $\boldsymbol{X}$ and a random sequence R. Especially:
   1a) Specify what the random sequence R consists of: bits, integers in some finite range etc.
   1b) Specify how A interprets R as a guess.
   1c) Specify how A verifies the guess.

Let $s1, s2, ..., s_n$. be the sequence of n rational numbers all greater than 0 and smaller than 1.

a) The random string R consists of random sequence of integers of length n, where each integer is an integer number between 1 and B.

$$R = r_1 r_2 ... r_n$$

$$\text{with } 1 \leq r_i \leq B \text{ for } i = 1...n$$

b) The algorithm A takes R as an assignation guess and swiping from left to right bins each s_i in the correspondent guessed bin.

$$\text{if } r_i = 1 \text{ then } s_i \in Bin_1$$

$$\text{if } r_i = n \text{ then } s_i \in Bin_n$$

c) A computes the sum of s_i in each of the bins and compares the sum to the max sum per bin, which is one. If the sum per bin is less or equal to 1 for each bin then returns YES, otherwise returns NO.

2) Show that the following two conditions are met:

2a) If the answer to $\boldsymbol{X}$ is YES, then there is a string R_0 with positive probability such that $A(\boldsymbol{X}, R_0) = \text{YES}$.

a) "If the answer to X is YES, then there is a string R0 with positive probability such that A(X, R0) = YES." If the B-partition exists, it can be represented by a string of numbers in range 1 to B, of length n, for example:

$$S = 0.1, 0.9, 0.5, 0.5$$

$$B = 2$$

$$Bin_1 = \{0.1, 0.9\}; Bin_2 = \{0.5, 0.5\}$$

$$R0 = 1, 1, 2, 2$$

For a sequence of length n, if every integer can have only at most B values, there are B^n possible combinations, among which, if exists, is R0. So the probability of R0 to be generated by a uniform random choice from this set of sequences is small but positive.

2b) If the answer to $\boldsymbol{X}$ is NO, then $A(\boldsymbol{X}, R) = \text{NO}$ for all R.

b)"If the answer to X is NO, then A(X,R) = NO for all R." If the B-partition doesn't exists, none of the possible sequences will return a YES. The algorithm A will always output NO in this case, hence the second condition is satisfied.

3) Show that the running time of A is polynomially bounded.

In the first step for each $i = 1...n$ the algorithm looks at r_i value, from a finite set (in this case B values, anyhow it is search-able in constant time) and assigns consequentially s_i to one of the three disjoint sets. Time complexity O(1)*O(n) = O(n). Then for each subset the algorithm sums over the elements and finally compares the B sums. Time complexity O(n)+O(1) = O(n). Hence the overall time complexity of the algorithm is O(n).

Exercise 4: Consider the following problem:

Problem [TwoCliques]
Input: An undirected graph $G = (V, E)$ and a positive integer k. **Output:** YES if G contains at least two vertex-disjoint cliques of size at least k each. and NO otherwise.
(Two cliques $V_1, V_2 \subseteq V$ are called vertex-disjoint if $V_1 \cap V_2 = \emptyset$).

Prove that TwoCliques is $\mathcal{NP}$-complete. To do this, you may use that the problem is in $\mathcal{NP}$.
Hint: Your reduction could start from Clique and increase the number of vertices and edges by a factor of about 2.
Especially show:

(A) Find a suitable problem P_c which is known to be $\mathcal{NP}$-complete.

(B) Prove $P_c \leq_p$ TwoCliques, especially:

(B.1) Describe a transformation T which transforms every instance $\boldsymbol{X}$ of P_c into an instance $T(\boldsymbol{X})$ of TwoCliques and which runs polynomial in the size $\|\boldsymbol{X}\|$ of $\boldsymbol{X}$.

(B.2) Show that if the answer to $\boldsymbol{X}$ is YES then so is the answer to $T(\boldsymbol{X})$.

(B.3) Show that if the answer to $T(\boldsymbol{X})$ is YES then so is the answer to $\boldsymbol{X}$.

Exercise 5: Consider the following algorithm.

repeat
  $a \leftarrow rand(1, 10)$;
  $b \leftarrow rand(1, 10)$;
  $c \leftarrow 10 \cdot a + b$;
until c is an even number
$print(c)$;

a) What is the expected running time of the algorithm? It suffices to determine the expected number of iterations of the repeat-until loop.

b) What is the probability that the number 13 is printed? What is the probability that the number 42 is printed?

c) What is the expected value of the number that the algorithm prints?
Hint: The summation formula $\sum_{i=0}^{k} i = \frac{k(k+1)}{2}$ might be useful.

Exercise 6: Consider the following six clauses over the boolean variables $\{x_1, x_2, x_3\}$:

$$\begin{aligned} c_1 &= x_1 \vee \overline{x_2} \\ c_2 &= x_3 \\ c_3 &= x_1 \vee x_2 \vee \overline{x_3} \\ c_4 &= \overline{x_1} \vee \overline{x_2} \vee x_3 \\ c_5 &= \overline{x_2} \vee \overline{x_3} \\ c_6 &= \overline{x_1} \vee x_2 \end{aligned}$$

A truth assignment to x_1, x_2, x_3 can satisfy at most 5 clauses.

a) Construct the relaxed linear program for the set of clauses as described in Section 5.3 of the notes.

b) One solution for a correctly constructed relaxed linear program is
$\hat{z}_1 = 1,\ \hat{z}_2 = 2/3,\ \hat{z}_3 = 1,\ \hat{z}_4 = 1,\ \hat{z}_5 = 1, \hat{z}_6 = 1,$
$\hat{y}_1 = 1/3,\ \hat{y}_2 = 1/3,\ \hat{y}_3 = 2/3$, which gives a target function value of $5\frac{2}{3}$.

Suppose you have access to a random number generator which produces random real numbers in the interval $[0, 1]$ according to uniform distribution.

What you have to do: Apply randomized rounding to find a truth assignment for x_1, x_2, and x_3, where the values given by the random number generator are 0.2234, 0.3422, 0.6943 respectively. Explain how you compute the truth assignment, and show how many clauses it satisfies.

Exercise 7: [**Multiple choice**] Answer the multiple choice questions below. There are 10 questions in total. An incorrect answer is counted negatively, that is, it cancels out a correct answer. However, the total number of points will be at least 0.

1.) If a problem is not in the class $\mathcal{NP}$ then it is not in the class $\mathcal{RP}$ either. ☒ True ☐ False

2.) The expression $rand(0,2) \neq rand(2,5)$ is always true. ☒ True ☐ False

3.) Las Vegas algorithms never give a wrong answer. ☐ True ☐ False

4.) If a randomized algorithm for a decision problem has one-sided error and gives the wrong answer with probability exactly 3/4, then it can be converted to one that gives the wrong answer with probability exactly 9/16. ☐ True ☐ False

5.) Section 5.4 of the lecture notes describes an RP-algorithm for 3-SAT. ☐ True ☐ False

6.) The expected value printed by the following program snippet is 3.5. ☐ True ☐ False

```
i ← 2.5
while (rand(3,4) + rand(1,2) = 5) do
  i ← i + 1
end while
print(i);
```

7.) The probability that $(rand(2,3) = 2 \vee rand(3,4) = 3)$ is true is 1/2. ☐ True ☐ False

8.) The TSP tour $(1,2,3,4,5,6)$ can be transformed into the tour $(6,5,4,3,2,1)$ by a single 2-OPT mutation. ☐ True ☐ False

9.) The randomized search heuristic (1+1) EA optimizes every function $f\colon \{0,1\}^n \to \mathbb{R}$ in expected time $O(2^n)$. ☐ True ☐ False

10.) There is a polynomial-time algorithm for KNAPSACK with approximation ratio no worse than 1.001. ☐ True ☐ False