CS760 Topics in Computational Complexity **Jul-Nov 2024**

Endsem exam

Total Marks: 70

Note: Please write your answers precisely and succinctly. You can directly use any statement proved in the class. Please ask if you are not sure. Please ask if you need to know any definitions.

1. **(1 mark)** A Wang tile is a unit square with its four edges being colored. Given a finite set of Wang tiles it is undecidable to find whether it admits a valid tiling of the plane. What is a valid tiling?

   Answer: A valid tiling is an assignment of tilings to integer grid such that for any two adjacent tiles, their overlapping edges should have same color.

2. **(1 mark)** Consider a graph with 5 vertices and degrees being $2, 3, 3, 2, 2$. What is the stationary distribution for a random walk on this graph?

   Answer: Stationary distribution $(1/6, 1/4, 1/4, 1/6, 1/6, 1/6)$.

3. **(1 mark)** In Conway's game of life, what is the behavior of a glider?

   Answer: Glider keeps moving diagonally forever (with its shape changing in period of 3 steps).

4. **(1 mark)** The area of proof complexity is about understanding whether two particular complexity classes are equal. What are those classes?

   Answer: NP and coNP.

5. **(1 mark)** Which of the following containments are known to be strict, which are known to be equal, and which are unknown.

$$\text{Space}(1) \subseteq \text{Space}(\log\log\log n) \subseteq \text{Space}(\log\log n) \subseteq \text{Space}(\log n).$$

   Answer: $\text{Space}(1) = \text{Space}(\log\log\log n) \subset \text{Space}(\log\log n) \subset \text{Space}(\log n)$.

6. **(1 mark)** What is a problem which is in NC^1, but not in $\text{AC}^0[3]$ (constant depth circuits with AND, OR, NOT, and MOD 3 gates).

   Answer: Parity.

7. **(1 mark)** If $\text{P} \neq \text{NP}$, then isomorphism conjecture will be true or false? Isomorphism conjecture states that all NP-complete languages are p-isomorphic to each other.

   Answer: It can be either true or false, nothing can be said.

8. **(1 mark)** Convergence rate of a random walk on a graph is governed by which eigenvalue of the random walk matrix.

   Answer: second largest eigenvalue (in absolute value).

9. **(1 mark)** In AKS primality test for a given number n, we check the identity $(x+a)^n \equiv (x^n+a) \pmod{n, x^r-1}$ for a small number of a's and an appropriate small number r. Will the test be correct if we ignore mod with $x^r - 1$? If yes, then why do we have it?

   Answer: The test will be correct, but without mod $x^r - 1$ the test will not be efficient. LHS has exponentially large expansion.

10. **(1 mark)** A graph has m edges. If we assign weights to edges randomly and independently from $\{1, 2, \dots, 10m\}$, then as per Isolation Lemma, probability that minimum weight perfect matching is unique is at least _____.

    Answer: 9/10.

11. **(1 mark)** Why is the grid graph not an expander graph (according to the combinatorial definition)?

    Answer: Consider the set of vertices in a $k \times k$ subgrid. Total number edges touching these vertices is roughly $4k^2$. But, out of those the number of edges going out of the set is only around $4k$. Exapnder graph requires constant fraction of edges going out.

12. **(1 mark)** Iterated integer multiplication is shown to be in NC^1 using Chinese remaindering. Suppose we want to multiply integers $x_1, x_2, \dots, x_n$. Then the idea is to choose small numbers $c_1, c_2, \dots, c_k$ (pairwise relatively prime) and compute the product $x_1 x_2 \cdots x_n \bmod c_j$ for each j. If we want to recover the actual product from these, what should be the condition on choice of c_j's.

    Answer: $c_1 c_2 \cdots c_k > x_1 x_2 \cdots x_n$.

13. **(1 mark)** Give a problem which is known to be #P-complete.

    Answer: Counting number of satisfying assignments of a given formula. Computing permanent of a given integer matrix.

14. **(1 mark)** What is known about the relation between complexity classes NP and PPAD (Nash equilibrium is PPAD-complete)?

    Answer: PPAD is contained in NP.

15. **(1 mark)** Let x and y be interpreted as characteristic vectors of two subsets $\{1, 2, \dots, n\}$. Let $\mathrm{DISJ}(x, y) = 1$ if these two subsets are disjoint, otherwise $\mathrm{DISJ}(x, y) = 0$. What is the lower bound on communication complexity of DISJ?

    Answer: n.

16. **(5 marks)** Let *Reach* denote the s-t reachability problem in a directed graph. Prove that $\mathrm{NL}^{Reach} = \mathrm{NL}$.

    Answer: Clearly, $\mathrm{NL} \subseteq \mathrm{NL}^{\mathrm{Reach}}$. We show the other direction. Recall that for an instance of reachability problem, if the answer is yes, then the path can be given as a proof which can be verified in logspace. If the answer is no, even then there is a proof verifiable in logspace. This was proved when we saw $\mathrm{NL} = \mathrm{coNL}$.

    Given a NL^{Reach} algorithm, we run it non-deterministically as it is, except when it makes a query to *Reach*. Whenever a query to *Reach* is made, we guess the answer (yes or no), then non-deterministically guess the proof and verify in logspace. If the proof is correct then we take our guess as correct and proceed. Otherwise we reject and stop on this path. If there was an accepting path in the original algorithm, then we will have one corresponding path where all our guess to *Reach* queries were correct and we finally accept. If the original algorithm rejects on all paths, then we will also reject whenever our guesses to *Reach* were correct. And on any path if any of our guesses is wrong then we reject anyway. So, we also reject on all paths.

17. **(5 marks)** Given a Turing machine M, we want to decide whether its running time is at most $100n^4 + 200n$ (i.e., if stops within $100|x|^4 + 200|x|$ steps on every input $x \in \{0, 1\}^*$). Prove that this problem is undecidable.

    Hint: For any TM N, you can construct a TM M such that N doesn't halt on input ε (empty string) if and only if M has running time at most $100n^4 + 200n$ for every input of size n.

    Answer: Given any TM N, we construct another TM M that takes an input y and simply runs N on ε for at most $|y|$ steps. If $N(\varepsilon)$ stops before $|y|$ steps, then M goes into infinite loop. Otherwise M stops and accepts. Now, observe that

    - if $N(\varepsilon)$ doesn't halt, then for all y, $M(y)$ stops and accepts in $|y|$ steps.
    - if $N(\varepsilon)$ halts in t steps, then for any y with $|y| \geq t$, $M(y)$ goes into infinite loop.

    We conclude that $N(\varepsilon)$ doesn't halt if and only for all y, $M(y)$ stops in $|y|$ steps. If had an algorithm for deciding whether running time of M is at most $100n^4 + 200n$, then we will get an algorithm for deciding whether $N(\varepsilon)$ halts for any given TM N. We know the latter is undecidable.

18. **(5 marks)** Prove that if $\mathrm{NTIME}(n) \subseteq \mathrm{P}$, then $\mathrm{P} = \mathrm{NP}$. Hint: padding argument.

    Answer: Consider any language L in $\mathrm{NTIME}(n^c)$ for some constant c. We define a new language

$$L' = \{x01^{|x|^c} : x \in L\}.$$

We claim that $L' \in \mathrm{NTIME}(n)$. Given an input y, we first check if it is of the form $y = x01^{|x|^c}$ and reject if not of this form. This can done in time $O(|y|)$. If it is of this form then we run the $\mathrm{NTIME}(n)$ machine for L on input x. It runs in time $O(|x|^c) = O(|y|)$. We accept if and only if L accepts x. This proves that $L' \in \mathrm{NTIME}(n)$.

By given hypothesis, $L' \in \mathrm{P}$. Now, to show that L is in P, observe that given an input x for L, we can compute $y = x01^{|x|^c}$ in polynomial time. Then we can give y to the polynomial time algorithm for L'. And output its answer as it is.

19. **(3+1+2+4 marks)** Consider the following interactive protocol for Graph Isomorphism. Given G_0 and G_1, the prover wants to convince the verifier that G_0 and G_1 are isomorphic (if they are), but does not want to reveal the permutation π which maps G_0 to G_1. They run the following protocol.

   - Prover: randomly chooses a permutation σ and a random bit $b \in \{0, 1\}$. Send $H = \sigma(G_b)$ (applying permutation σ on vertices of G_b) to verifier (b is not revealed).
   - Verifier: Verifier chooses a random bit $b' \in \{0, 1\}$ and sends it to prover (i.e., asks the prover to give the permutation which maps $G_{b'}$ to H).
   - Prover: sends a permutation τ.
   - Verifier accepts if $H = \tau(G_{b'})$.

   (a) Suppose prover knows the permutation π such that $G_1 = \pi(G_0)$. How should she compute τ?

   Answer:
   - If $b = b'$ then send $\tau = \sigma$.
   - If $b = 1$ and $b' = 0$ then send $\tau = \sigma\pi$.
   - If $b = 0$ and $b' = 1$ then send $\tau = \sigma\pi^{-1}$.

   (b) Prove that if G_0 and G_1 are isomorphic then verifier will accept with probability 1.

   Answer: If $G_1 = \pi(G_0)$. Then observe that τ computed by prover will be such that $H = \tau(G_{b'})$. Hence, verifier will always accept.

   (c) Prove that if G_0 and G_1 are not isomorphic then irrespective of what strategy is followed by the prover, verifier will reject with probability at least 1/2.

   Answer: If they are not isomorphic. Then H can be isomorphic to only one of G_0 and G_1, irrespective of how H is generated. Since, b' is chosen randomly, with probability 1/2, we have that H is not isomorphic to $G_{b'}$. If they are not isomorphic then there is no τ which is accepted in last step.

   (d) Prove that if G_0 and G_1 are isomorphic then this is a zero knowledge protocol (for an honest verifier). That is, prove that there is a simulator algorithm S such that the output of $S(G_0, G_1)$ (without knowing π) has the same distribution as the outcome of the above protocol (H, b', τ) (assuming that verifier and prover stick to the above protocol).

   Answer: S does the following. Generate $a \in \{0, 1\}$ randomly, generate permutation μ randomly. Generate $J = \mu(G_a)$. Finally give (J, a, μ). We claim that the distribution of S's outcome (J, a, μ) is same as that of the outcome of the protocol (H, b', τ).

   Clearly, b' and a are generated randomly, so have the same distribution. Also, μ is a random permutation, while τ is obtained by taking a random permutation σ and possibly multiplying another permutation with it. So μ and τ have the same distribution. Also, a and μ are independent random variables. Similarly, b' and τ are independent because b' and σ are independent. Finally, H is always such that $H = \tau(G_{b'})$. And J is always such that $J = \mu(G_a)$.

20. **(7 marks)** Show that there exists a subset $S \subseteq \{0, 1\}^n$ of size n^{10} such that for all circuits C on n inputs and size at most n^2, we have

$$\left| \Pr_{x \sim U_n}[C(x) = 1] - \Pr_{y \sim D_S}[C(y) = 1] \right| \leq 1/10.$$

Here U_n is the uniform distribution on length n strings. And D_S is the uniform distribution on strings in S.

Hint: choose S randomly.

The following Hoeffding's inequality may be useful. Suppose $X_1, X_2, \ldots, X_k$ are independent random variables taking values in $\{0,1\}$. Let $X = \sum_i X_i$ be their sum and let $\mu = \mathbb{E}[X]$ denote the sum's expected value. Then for for any $t > 0$,

$$\Pr[X \leq \mu - t] < e^{-2t^2/k} \text{ and } \Pr[X \geq \mu + t] < e^{-2t^2/k}.$$

Answer: We construct the set S as follows. Repeat n^{10} times: pick an element randomly uniformly from $\{0,1\}^n$ and put it in S. We may have put the same element multiple times, we will come to this point later. For now, think of S as a multi-set of size n^{10} (an element may appear multiple times).

Fix a circuit C on n inputs and size at most n^2. Suppose $\Pr_{x \sim U_n}[C(x) = 1] = q$. For $1 \leq i \leq n^{10}$, let X_i indicate the event that the element chosen in i-th iteration is accepted by C. Since we choose elements randomly uniformly from $\{0,1\}^n$, we have that $\Pr[X_i = 1] = q$. Note that $X = \sum_i X_i$ gives us the number of elements from S which are accepted by C. By linearity of expectation $\mathbb{E}[X] = qn^{10}$. That is, we expect qn^{10} elements from S to be accepted by C. Now, let us bound the probability that the number deviates too much from the expectation. From Heoffding's inequality above,

$$\Pr[|X - qn^{10}| \geq n^{10}/10] < 2e^{-n^{10}/50}.$$

Equivalently, with probability (over choice of S) at least $1 - 2e^{-n^{10}/50}$,

$$|q - \Pr_{y \sim D_S}[C(y) = 1]| \leq 1/10,$$

which is the desired property from S. But, we want this property of S for all circuits C of size n^2. The number of such circuits is at most n^{4n^2}. We can take union bound over all the bad events and conclude that probability that S doesn't work for some circuit is at most $2e^{-n^{10}/50} \times n^{4n^2} \lll 1$.

We are almost done. We also need to consider the bad event when S has some elements multiple times. The probability that this happens is at most $n^{20}/2^n$. So, the probability that S has n^{10} distinct elements and that it works for all circuits is at least $1 - 2e^{-n^{10}/50} \times n^{4n^2} - n^{20}/2^n > 0$. This proves the existence of desired set S.

21. **(5 marks)** Prove that if $SAT \in$ BPP then $SAT \in$ RP. Hint: self reducibility.

Recall that BPP algorithm can give wrong answer with some probability for both yes and no instances. While an RP algorithm can give wrong answers only for yes instances.

Answer: Suppose we have a BPP algorithm for SAT with success probability $2/3$. By repeating the algorithm multiple times and taking majority answer, we can boost the success probability to say, $1 - 1/n^2$. Let the improved algorithm be A. Now, we will use the standard self reducibility of SAT to construct a satisfying assignment (it it exists). We first test if the given formula is satisfiable using A. If the answer is no, we return unsatisfiable. Otherwise, we set x_1 as True and then test if the obtained formula is satisfiable (using A). If yes, then we continue with x_1 as True, otherwise we set x_1 as False and continue. We continue like this with all n variables and get an assignment to all variables. We check if the assignment is indeed satisfying, which we can do with certainty (with zero error probability). If the assignment is satisfying we output satisfiable, otherwise we output unsatisfiable.

Observe that if the formula was unsatisfiable then we can never get a satisfying assignment and we will always say unsatisfiable. In other words, for no instances, the algorithm never gives wrong answers. On the other hand for yes instances, the Algorithm A may give wrong answer in the beginning or at any intermediate call. We call algorithm A at most n times. By union bound, it is correct on all n calls with probability at least $1 - 1/n$. If the formula is satisfiable and if algorithm A always gives correct answer, then we will certainly get a satisfying assignment. Thus, we can conclude that for yes instances, our algorithm is successful with probability at least $1 - 1/n$.

22. **(5 marks)** Prove that if ρ-approximation of MAX-4-SAT is NP-hard for some constant ρ, then ρ'-approximation of MAX-3-SAT is NP-hard for some appropriate constant ρ'. Recall that MAX-k-SAT asks for the maximizing the number of satisfied clauses in a given k-SAT formula.

Answer:

We need do a reduction from MAX-4-SAT to MAX-3-SAT, which is approximation preserving. Suppose the given 4-SAT formula ϕ has m clauses $C_1, C_2, \ldots, C_m$ in variables $x_1, x_2, \ldots, x_n$. For each of the m clauses (with 4 literals), we construct a set of clauses with 3 literals. Let the clause C_i be $(\ell_1 \vee \ell_2 \vee \ell_3 \vee \ell_4)$ for some literals $\ell_1, \ell_2, \ell_3, \ell_4$. We will introduce a new variable y_i and replace C_i with following two clauses

$$C_i' = (\ell_1 \vee \ell_2 \vee y_i), \quad C_i'' = (\ell_3 \vee \ell_4 \vee \neg y_i).$$

By replacing all the clauses similarly, we get a new formula, say ψ, in variables $x_1, x_2, \ldots, x_n, y_1, y_2, \ldots, y_m$. Observe that if C_i is true under an assignment σ to x_j variables then C_i' and C_i'' both are true under σ and an appropriate assignment to y_i. Moreover, if C_i is false under an assignment σ to x_j variables then at most one of C_i' and C_i'' can be true under σ and any assignment to y_i.

Let $val(\phi)$ and $val(\psi)$ be the maximum number of clauses that can be satisfied in ϕ and ψ respectively. From above discussion we conclude that $val(\psi) = val(\phi) + m$. To show NP-hardness, we need to consider decision versions of both the problems. NP-hardness of ρ-approximation of MAX-4-SAT means that for some constant c, it is NP-hard to distinguish between 4-SAT instances which either have $val(\phi) \geq cm$ or have $val(\phi) \leq \rho cm$. Let us abbreviate this as NP-hardness of $(c, \rho c)$-4-SAT. From the above reduction, we get that it is NP-hard to distinguish between 3-SAT instances instances which either have $val(\psi) \geq (c+1)m$ or have $val(\psi) \leq (\rho c + 1)m$. That is, we get NP-hardness of $((c+1)/2, (\rho c+1)/2)$-3-SAT.

If c was 1, i.e., we knew NP-hardness for $(1, \rho)$-4-SAT, then we immediately get NP-hardness of $(1, (1+\rho)/2)$-3-SAT. But, if c is different from 1, then we have to use another argument to bound the ratio $\frac{\rho c+1}{c+1}$. Note that in any 4-SAT formula, every clause (OR of at most 4 literals) is satisfied by at least half of the assignments. By averaging argument, there must be an assignment which satisfies at least half of the clauses. Hence, we can assume $\rho c \geq 1/2$. Then, we get that

$$\frac{\rho c + 1}{c+1} \leq \frac{1 + 1/2}{1 + 1/(2\rho)} = 3\rho/(2\rho+1).$$

The inequality holds, because LHS is a decreasing function in c, so the max is attained at $\rho c = 1/2$. We conclude NP-hardness of $((c+1)/2, \rho'(c+1)/2)$-3-SAT, where $\rho' \leq 3\rho/(2\rho+1)$.

23. **(7 marks)** Prove that PCP$(1, \log n)$ = P. Recall that PCP$(1, \log n)$ denotes the class of languages whose yes instances have probabilistically checkable proofs which can be verified by a verifier using $O(1)$ random bits and $O(\log n)$ queries to the proof.

    Answer: P $\subseteq$ PCP$(1, \log n)$ is straighforward, because verifier can just run the polynomial time algorithm, while ignoring the proof.

    For the other direction, suppose the verifier algorithm uses c random bits and $d \log n$ queries to the proof. For each of the 2^c choices of random bits, the verifier may query a distinct set of $d \log n$ bits. So, at most $2^c d \log n$ bits of the proof can possibly be queried, and other bits in the proof, if any, are not useful. There are at most $2^{2^c d \log n} = n^{2^c d}$ possibilities of these bits. One can go over all these possibilities. For each choice of proof bits, go over 2^c choices of random bits and explicitly compute the probability of acceptance. If there exists one choice of proof bits such that the probability of acceptance is 1, then we accept. Otherwise reject. The running time of this deterministic algorithm will be $O(2^c n^{2^c d})$ times the running time of the verifier algorithm. As c and d are constants, the overall running time is polynomial.

24. **(3+3 marks)** Suppose we have a pseudorandom generator (PRG) $G \colon \{0,1\}^\ell \to \{0,1\}^m$ for $m = 2^{10\sqrt{\ell}}$, that can be computed in $2^{O(\ell)}$ time and fools circuits of size $O(m^3) = O(2^{30\sqrt{\ell}})$. Using this PRG, for every problem in BPP, we can design a deterministic algorithm, but not necessarily polynomial time. Roughly, what is the best running time upper bound we can get?

    Roughly, what is the smallest function $g(n)$ such that a average case circuit lower bound of $\Omega(g(n))$ will give us a PRG with above parameters. You can directly write the answer, proof is not required.

    Answer: Suppose the randomized algorithm runs in time n^c and uses at most n^c random bits. Let us take $n^c = m = 210\sqrt{\ell}$, i.e., $\ell = c^2 \log_n^2 /100$. We go over all choices of seed s (of length ℓ) and compute $G(s)$. For each choice of s, we run the algorithm with $G(s)$ as the choice of random bits and then take the majority answer over choices of s. This is correct, every algorithm with at most $O(n^{3c})$ running time thinks of $\{G(s) : s\tilde{U}_\ell\}$ as almost random. And the running time of our algorithm is only $O(n^c)$. By going over all choices of s, the running time of the algorithm is multiplied by $2^\ell = 2^{O(c^2 \log^2 n)}$. This is called quasi-polynomial time.

    The circuit lower bound we need is roughly $\Omega(2^{b\sqrt{n}})$ for some constant b.