ShanghaiTech University

CS240 Algorithm Design and Analysis

Spring 2024

Problem Set 4

Name: Zhou Shouchen

StudentID: 2021533042

Due: 23:59, May 31, 2024

1. Submit your solutions to the course Gradescope.
2. If you want to submit a handwritten version, scan it clearly.
3. Late homeworks submitted within 24 hours of the due date will be marked down 25%. Homeworks submitted more than 24 hours after the due date will not be accepted unless there is a valid reason, such as a medical or family emergency.
4. You are required to follow ShanghaiTech's academic honesty policies. You are allowed to discuss problems with other students, but you must write up your solutions by yourselves. You are not allowed to copy materials from other students or from online or published resources. Violating academic honesty can result in serious penalties.

Problem 1:

Given a set C, a collection of subsets of C and an integer $k \geq 1$, the Set-Packing problem asks if there are k subsets from the collection which are pairwise disjoint (i.e. no two sets share an element). Show that the Set-Packing problem is NP-complete.

Solution:

To prove that the Set-Packing problem is NP-complete, we need to demonstrate two things:

1. **Set-Packing is in NP**: We need to show that a given solution can be verified in polynomial time.
2. **Set-Packing is NP-hard**: We need to reduce a known NP-complete problem to Set-Packing in polynomial time.

1. Set-Packing is in NP

A problem is in NP if a solution can be verified in polynomial time. For the Set-Packing problem, given a collection of subsets and an integer k, a "solution" would be a list of k subsets. We can verify whether these k subsets are pairwise disjoint by checking each pair of subsets to ensure they do not share any element. There are $\binom{k}{2}$ pairs of subsets to check, and for each pair, we can check for intersection in time proportional to the sizes of the subsets involved. Thus, if the subsets and k are polynomial in the size of the input, this verification process is polynomial, confirming that Set-Packing is in NP.

2. Set-Packing is NP-hard

To show that Set-Packing is NP-hard, we reduce from a known NP-complete problem. A convenient choice is the Independent Set problem, which is well-known to be NP-complete.

Independent Set Problem

Given a graph $G = (V, E)$ and an integer k, the problem asks whether there exists a set of k vertices in G such that no two vertices in the set are adjacent.

Reduction from Independent Set to Set-Packing

1. **Construction**: Given an instance of the Independent Set problem, i.e., a graph $G = (V, E)$ and an integer k, construct an instance of Set-Packing as follows:

   - For each vertex v in V, create a subset S_v containing all the edges that are incident to v in G. Specifically, $S_v = \{e \in E \mid v \in e\}$.
   - Let the collection of subsets be $\{S_v \mid v \in V\}$.
   - The goal is to find whether there exists a set of k subsets from this collection that are pairwise disjoint.

2. **Correctness of the Reduction**:

   - If there is an independent set of size k in G, then the corresponding subsets $\{S_v \mid v \text{ is in the independent set}\}$ are pairwise disjoint. This is because if two vertices u and v are independent (not adjacent), there are no edges shared between S_u and S_v.
   - Conversely, if there exists a set of k pairwise disjoint subsets in the Set-Packing instance, the vertices corresponding to these subsets form an independent set in G. No two subsets share an edge, which means their corresponding vertices do not share an edge and thus are not adjacent.

3. **Polynomial Time**: The reduction constructs a subset for each vertex and assigns the incident edges to each subset, both of which can be accomplished in polynomial time relative to the size of V and E.

Conclusion

Given that we can verify solutions in polynomial time and that we can reduce the Independent Set problem (which is NP-complete) to Set-Packing in polynomial time, it follows that Set-Packing is NP-complete. This shows that it is not only a complex problem but one for which no polynomial-time algorithm is likely to exist unless P = NP.

Problem 2:

Given a Boolean CNF (conjunctive normal form) formula ϕ and an integer $k \geq 1$, the Stingy-SAT problem asks whether the formula has a satisfying assignment in which at most k variables are set to true. Prove that Stingy-SAT is NP-complete.

Solution:

To prove that the Stingy-SAT problem is NP-complete, we follow two steps:

1. **Stingy-SAT is in NP**: We need to verify that a given solution (a set of variable assignments) can be checked in polynomial time. The verification involves ensuring no more than k variables are set to true and that the CNF formula ϕ is satisfied under this assignment.

2. **Stingy-SAT is NP-hard**: We reduce the NP-complete problem SAT to Stingy-SAT. Given a CNF formula ψ from a SAT instance:

   - Construct a Stingy-SAT instance (ϕ, k) where $\phi = \psi$ and k is the total number of variables in ψ.
   - **Correctness**: If ψ is satisfiable, any satisfying assignment (up to the total number of variables) is a valid solution for Stingy-SAT. Conversely, a solution to the Stingy-SAT instance indicates that ψ is satisfiable.

   This reduction is polynomial as it involves only setting ϕ and k based on ψ.

Conclusion: Since Stingy-SAT can verify solutions in polynomial time and can be reduced from SAT in polynomial time, it is NP-complete.

Problem 3:

Given a set C, a collection of subsets of C and an integer $k \geq 1$, the Set-Cover problem asks whether there are at most k subsets from the collection which cover C, i.e. whose union includes all of C. Show that Set-Cover is NP-complete. Do not use a reduction from a problem which is very similar to Set-Cover.

Solution:

To prove that the Set-Cover problem is NP-complete, we first establish that it is in NP because a candidate solution can be verified in polynomial time by checking that the union of selected subsets covers all elements of set C and that no more than k subsets are used.

Reduction from 3-SAT to Set-Cover:

- **Input Construction**:
  - **Set** C: Corresponds to clauses in the 3-SAT instance.
  - **Collection of Subsets**: For each variable x_i, create subsets for x_i = true and x_i = false, each containing clauses satisfied by these assignments.
- **Correctness of Reduction**:
  - If the 3-SAT formula is satisfiable, the corresponding subsets cover all clauses in C.
  - Conversely, a cover for all clauses implies a satisfying assignment for the 3-SAT formula.
- **Polynomial Time**: Direct construction from the 3-SAT formula, polynomial in the number of variables and clauses.

This reduction demonstrates that Set-Cover is NP-hard. Coupled with the fact that Set-Cover is in NP, it is therefore NP-complete.

Problem 4:

Consider a list with n unique positive integers, and suppose we iterate through the numbers one by one in a random order. As we do this, we maintain a variable b equal to the largest number we have seen so far. Initially b is set to 0 . Compute the expected number of times b is updated.

For example, if the input list is $[4, 7, 5]$, and we iterate through the list in the order $3, 1, 2$, then b is updated twice, when we see 5 and 7.

Solution:

https://math.stackexchange.com/questions/205714/iterating-over-a-list-of-n-distinct-integers-how-many-times-will-the-running-ma