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:

1. Set-Packing is in NP.

For a collection of subsets and an integer k, we can directly check each pair of subsets to verify whether they are pairwise disjoint. There are $\binom{k}{2}$ pairs of subsets to check, and for each pair, we can check whether they have the same element in $O(|C|)$ time by iterating through the elements, where $|C|$ is the number of elements in the set C. So we can verify the solution is valid or not in $O\left(\binom{k}{2}|C|\right)$ time.
So Set-Packing $\in$ NP.

2. Independent-Set $\leq_p$ Set-Packing.

We have known that the Independent-Set problem is NP-complete.
Given a graph $G = (V, E)$ and an integer k, the Independent Set problem asks whether there exists a set of k vertices in G such that no two vertices in the set are adjacent.

<1> "$\Rightarrow$":

For an instance of Independent-Set problem, construct an instance of Set-Packing as follows:
Let each node in the graph be a subset in the Set-Packing problem. For each node v in V, create a subset S_v containing all the edges that are incident to v in G. Specifically, $S_v = \{e \in E \mid v \in e\}$.
Since it is a yes instance of Independent-Set problem, there exists a set of k vertices in G such that no two vertices in the set are adjacent. This means that there exists a set of k subsets in the Set-Packing problem that are pairwise disjoint. So it is a yes instance of Set-Packing problem.

<2> "$\Leftarrow$":

For an instance of Set-Packing problem, construct an instance of Independent-Set as follows:
Let each subset in the Set-Packing problem be a node in the graph. For each subset S_v in the Set-Packing problem, create a node v in the graph.

Since it is a yes instance of Set-Packing problem, there exists a set of k subsets in the collection which are pairwise disjoint. This means that there exists a set of k vertices in G such that no two vertices in the set are adjacent. So it is a yes instance of Independent-Set problem.

<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.

So we have proved that Independent Set $\leq_p$ Set-Packing.

Since Independent-Set $\in$ NP-complete; Set-Packing $\in$ NP; Independent-Set $\leq_p$ Set-Packing.
Therefore, Set-Packing $\in$ NP-complete.

So above all, we have proved that Set-Packing is a NP-complete problem.

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:

1. Stingy-SAT is in NP.

We can varify a CNF formula ϕ is satisfied or not in polynomial time.
And then check the number of variables set to true is no more than k in $O(n)$ time, where n is the length of ϕ.
So the total time complexity is polynomial.
So Stingy-SAT $\in$ NP.

2. SAT $\leq_p$ Stingy-SAT.

We can reduce SAT to Stingy-SAT.
Given a SAT problem, we additionally check whether the number of variables set to true is no more than k.

<1> "$\Rightarrow$":
If the SAT problem with firmula ϕ with k variables is satisfied, then the number of variables set to true is no more than k.
So the Stingy-SAT (ϕ, k) is satisfied.

<2> "$\Leftarrow$":
If the Stingy-SAT (ϕ, k) is satisfied, then without the constrains of k, the SAT problem is also satisfied.

<3> Polynomial Time:
The construction only needs to check the number of variables set to true, which is in $O(n)$ time.
So it could be done in polynomial time.
So we have proved that SAT $\leq_p$ Stingy-SAT.

Since SAT $\in$ NP-complete; Stingy-SAT$\in$ NP; SAT $\leq_p$ Stingy-SAT.
Therefore, Stingy-SAT $\in$ NP-complete.

So above all, we have proved that Stingy-SAT is a NP-complete problem.

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:

1. Set-Cover is in NP.

Given a collection of subsets of C, and the integer k, we can visit each subset C_i, and mark the elements in C_i as covered.
The visit would cost $O(k|C|)$ time, where $|C|$ is the number of elements in $|C|$.
Then we check whether all elements in C are covered.
So the total time complexity is $O((k+1)|C|)$, which is polynomial.
So Set-Cover $\in$ NP.

2. Vertex-Cover $\leq_p$ Set-Cover.

We can reduce Vertex-Cover to Set-Cover.
Given a Vertex-Cover problem, we can construct a Set-Cover problem.
Given a graph $G = (V, E)$, we can construct a set C which contains all the edges in E.
For each vertex $v \in V$, we can construct a subset C_i which contains all the edges incident to v.

<1> "$\Rightarrow$":
Given a Vertex-Cover problem, we can construct a Set-Cover problem.
Given a Vertex-Cover problem, we can construct a set C which contains all the edges in E.
For each vertex $v \in V$, it represents a subset C_i, and the edge e connects to v represents an element in C_i.
Then we can prove that the Vertex-Cover problem has a solution if and only if the Set-Cover problem has a solution and the solution must have at most k vertices.

<2> "$\Leftarrow$":
If a Set-Cover problem has a solution, then the solution must have at most k subsets.
Then we can prove that the Vertex-Cover problem has a solution and the solution must have at most k vertices.

Let each subset C_i in the solution corresponds to a vertex v in the solution.
And each element in C_i corresponds to an edge e incident to v and a duplicate single node v'.
Then we can prove that the Vertex-Cover problem has a solution if and only if the Set-Cover problem has a solution and the solution must have at most k vertices.

<3> Polynomial Time:
The construction of the Set-Cover problem from the Vertex-Cover problem costs $O(|V| + |E|)$ time.
Which is a polynomial time.

So we have proved that Vertex-Cover $\leq_p$ Set-Cover.

Since Vertex-Cover $\in$ NP-complete; Set-Cover $\in$ NP; Vertex-Cover $\leq_p$ Set-Cover.
Therefore, Set-Cover $\in$ NP-complete.

So above all, we have proved that Set-Cover is a NP-complete problem.

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:

We firstly introduce a Lemma:

$$\sum_{k=i}^{n} \binom{k-1}{i-1} = \binom{n}{i}$$

Proof of the Lemma:

1. When $n = i$, the left side of the equation is $\binom{i-1}{i-1} = 1$, and the right side of the equation is $\binom{i}{i} = 1$. The equation holds.

2. Suppose the equation holds when $n = m \geq i$, i.e.

$$\sum_{k=i}^{m} \binom{k-1}{i-1} = \binom{m}{i}$$

then we need to prove the equation holds when $n = m + 1$:

$$\sum_{k=i}^{m+1} \binom{k-1}{i-1} = \sum_{k=i}^{m} \binom{k-1}{i-1} + \binom{m}{i-1} = \binom{m}{i} + \binom{m}{i-1} = \binom{m+1}{i}$$

The equation holds when $n = m + 1$.

So above all, we have proved the Lemma.

So for the list, since the numbers are unique, so W.L.O.G, we can assume the list is $[1, 2, 3, ..., n]$.
Let the expected number of times b is updated to be E.

So

$$
\begin{aligned}
E &= \frac{1}{n!} \sum_{n!\text{ permutations}} \#\text{ times } b \text{ is updated in this permutation} \\
&= \frac{1}{n!} \sum_{n!\text{ permutations}} \sum_{i=1}^{n} \#\text{ times } b \text{ is updated at the } i\text{-th number} \\
&= \frac{1}{n!} \sum_{i=1}^{n} \sum_{n!\text{ permutations}} \#\text{ times } b \text{ is updated at the } i\text{-th number} \\
&= \frac{1}{n!} \sum_{i=1}^{n} \sum_{k=i}^{n} \binom{k-1}{i-1} (i-1)!(n-i)! \quad \text{(let the } i\text{-th number is } k\text{, and it is the biggest among first } i \text{ numbers)} \\
&= \frac{1}{n!} \sum_{i=1}^{n} (i-1)!(n-i)! \sum_{k=i}^{n} \binom{k-1}{i-1} \\
&= \frac{1}{n!} \sum_{i=1}^{n} (i-1)!(n-i)! \binom{n}{i} \qquad \text{(Lemma)} \\
&= \sum_{i=1}^{n} \frac{(i-1)!(n-i)!}{n!} \binom{n}{i} \\
&= \sum_{i=1}^{n} \frac{i!(n-i)!}{n!} \binom{n}{i} \frac{1}{i} \\
&= \sum_{i=1}^{n} \frac{1}{\binom{n}{i}} \binom{n}{i} \frac{1}{i} \\
&= \sum_{i=1}^{n} \frac{1}{i} \\
&= \log n + \gamma
\end{aligned}
$$

So above all, the expected number of times b is updated is $\log n + \gamma$, where γ is the Euler-Mascheroni constant.