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. k-CLIQUE $\leq_p$ Set-Packing.

We have known that the k-CLIQUE problem is NP-complete.
For an instance of k-CLIQUE problem whose graph is G, we can construct an instance of Set-Packing. Let each node v in graph G to be a subset of C, every edge (u, v) to be two subsets from C, so that the two subsets are disjoint.

<1> "$\Rightarrow$":

If C is a yes instance of Set-Packing problem, then we can construct a graph with the above methods, so that it is also a yes instance for the k-CLIQUE problem.

<2> "$\Leftarrow$":

If G is a yes-instance for the k-CLIQUE problem, then there must have a clique with k vertices that are all connected, then all the corresponding k subsets are disjoint, then it is a yes-instance to the Set-Packing 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 k-CLIQUE $\in$ NP-complete; Set-Packing $\in$ NP; k-CLIQUE $\leq_p$ 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.

<1> "$\Rightarrow$":

Since there are at most n variables in a formula ϕ. So for a yes instance of SAT, we can set $k = n$, then it is also a yes instance for the Stingy-SAT (ϕ, n).

<2> "$\Leftarrow$":

For a yes instance of the Stingy-SAT (ϕ, k), then without the constrains of k, the SAT problem with ϕ has less constrains, which is also a yes instance for SAT problem.

<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. 3-SAT $\leq_p$ Set-Cover.

We can reduce 3-SAT to Set-Cover.
Given a 3-SAT problem, we can construct a Set-Cover problem.
Let the 3-SAT problem has $x_1, \cdots, x_n$ variables, $C_1, \cdots, C_m$ clauses, each clause has 3 literals.
We can construct the subsets: Let S_{x_i} to be clauses C_j that x_i appears in C_j.
Let $S_{\neg x_i}$ to be clauses C_j that $\neg x_i$ appears in C_j.
And we can set k to be the number of variables, if we can cover all the clauses.

<1> "$\Rightarrow$":

If 3-SAT has a yes instance, then each clause has at least one true literal. Choose the subsets corresponding to the literals that are true in the satisfying assignment. This selection covers all clauses, as each clause has at least one true literal by the definition of a satisfying assignment. So it is also a yes instance for the Set-Cover problem.

<2> "$\Leftarrow$":

If Set-Cover has a yes instance, which means that we can cover all clauses with at most n subsets. So there exists a selection of literals that covers every clause. Assign true to the literals selected in the solution and false to their complements. So it is also a yes instance for the 3-SAT problem.

<3> Polynomial Time:

The construction of the Set-Cover problem from the 3-SAT problem costs polynomial with $|V|$ and $|E|$ time.

Which is a polynomial time.

So we have proved that 3-SAT $\leq_p$ Set-Cover.

Since 3-SAT $\in$ NP-complete; Set-Cover $\in$ NP; 3-SAT $\leq_p$ 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.