COMP4600/8460 - Advanced Algorithms **S2 2022**

Assignment 1

[Xueer Zhangsong] ([U7079320]) Date: August 14, 2022

Note: *Plagiarism is strictly prohibited. Even though you may discuss with classmates, you should write your homework assignment by yourself.*

1 Questions

Q1) Prove $\mathsf{isPrime} \in \mathsf{NP} \cap \mathsf{co\text{-}NP}$ (5 Marks)

Q2) Prove $\mathsf{MaximalMatch} \in \mathsf{P}$ (5 Marks)

Q3) Given a connected graph $\mathcal{G}$ and a subset of vertices $\mathcal{R}$ (where $|\mathcal{R}| \geq 2$), prove that $\mathcal{G}$ always contains a spider decomposition of $\mathcal{R}$ (5 Marks)

Q4) Implement greedy algorithm $\mathcal{A}_{\mathsf{setcover}}$ in Python/Java/C. Evaluate the approximation ratio on random $\mathsf{SetCover}$ instances, and compare the running time with exhaustive search method. To generate a random $\mathsf{SetCover}$ instance, you may assign a cost to each cover by a uniform probability distribution, and make each item coverable by a particular cover with a probability = 0.5. Plot and discuss the results of approximation ratios and running times here in this report, and upload your code separately on Wattle (10 Marks)

2 Answers

2.1 Question-1

(1) Prove $isPrime \in \mathsf{co\text{-}NP}$

Given a verifier A:

For an input S and c, if $c|S$ then

A(S,c) returns 1

else

A(S,c) returns 0.

Given a series of non-trivial positive integer that is smaller than S as certificate. For each item c in the certificate verify it using A.

$isComposite(S, c)$ return true if $\exists c$ that A(S,c) = 1, else it returns false.

Since it only takes non-trivial positive integer that is smaller than S as certificate, the size of certificate is less then the the size of input S. If input S has n bits, then size is $O(n)$, which has a polynomial size.

To go through every non-trivial positive integer that is smaller than S and to verify each of them using A. The verification finish in a polynomial time in $O(n^2)$.

Thus $isComposite \in NP$

Since $isComposite = \overline{isPrime}$

$isPrime \in$ co-NP

(2) Prove $isPrime \in$ NP

The number N is prime if and only if it has a primitive root. Which means that a cyclic group of $N-1$ exist. Prove N is prime is equal to prove the group is cyclic.

Assume the primitive root of prime N is r. Firstly, need to prove $r^{N-1} \equiv 1 mod(N)$ can be test in polynomial time. Secondly, prove $r^a \neq 1(modN)$ for $\forall a$ greater than 0 and smaller than $N-1$. Which can be simplify to prove $r^{\frac{N-1}{Pi}} \neq 1(modN)$ for all the prime divisor Pi of $N-1$.

Convert decimal number N to binary form. The length of the binary number is $\log N$. Thus we can verity $r^{N-1} \equiv 1 mod(N)$ and $r^{\frac{N-1}{Pi}} \neq 1(modN)$ in $O(\log n^c)$.

Thus the certificate for proving N is prime is $\{r; (p1, C(p1))...(pk, C(p1))\}$, where $C(pi)$ is the certificate for prime factor pi.

Based on the assumption, if N is prime then $N-1$ should be a composite. The largest prime factor of $N-1$ should smaller then $\sqrt{N}$. Thus, there are at most $\log N - 1$ of prime factor. Thus the size of the certification has a polynomial size of $O(\log n^c)$.

Thus $isPrime \in$ NP

(3) $isPrime \in NP \cap$ co-NP

Based on the result in (1) and (2), we can prove that:

$isPrime \in NP \cap$ co-NP

2.2 Question-2

(1) Maximal match algorithm

Given a graph G = (V,E)

$\bar{E} \leftarrow \emptyset$

[1] While $E \neq \emptyset$

[2] Random pick edge $e = (v, u) \in E$ where $v, u \in V$

[3] $\bar{E} \leftarrow \bar{E} \cup \{(v, u)\}$

[4] Remove all the edges in E that contains vertex v and u

[5] Return $\bar{E}$

(2) Prove the algorithm is correct

Based on the definition of the MaximalMatch, the algorithm should find a set of edges, that no other edges can be added in the set (condition 1), where all of the vertices can only be covered once (condition 2). And there is no requirement in the definition that the number of edges be the maximum in all cases. Thus, there can be several MaximalMatch results for one graph.

The algorithm in (1) first randomly selects an edge e, and then removes all edges that containing two vertices of e from the edge set E. This satisfies the condition 2 of MaximalMatch. The while loop of this algorithm will loop until the edge set E becomes the empty set. When the algorithm is completed, it means that no other edges can be added to the final result. Which satisfies the condition 1 of MaximalMatch.

Thus, this algorithm is correct

(3) Prove the $MaximalMatch \in P$

Assume there are respectively n items in set E. The complexity for line [2] and [3] in the algorithm is $O(1)$. The maximum complexity for line [4] is $O(n)$. Since the number of edges in set E is keep decreasing, the maximum iteration of the while loop is $\frac{n}{2}$. Thus the total complexity for this algorithm is $O(n^2)$.

This algorithm can finished in polynomial time.

Thus $MaximalMatch \in P$

2.3 Question-3

(1) Spider decomposition algorithm

$C \leftarrow \emptyset$

$finalize = False$

$Path$

[1] Find the minimal spanning tree in G that connect all the vertices in R.

[2] Set all the vertices of degree one in R as leave nodes.

[3] Set the rest of vertices in R as the center nodes, put them in set C.

[4] while $finalize = False$

[5] $Path \leftarrow \emptyset$

[6] Assign all the leaves to their closest center node and add their path from leaf to center in $Path$.

[7] Go through all the paths. If there are joint vertices on the paths to the same destinations.

[8] $C \leftarrow C \cup \{jointvertices\}$

[9] $finalize = False$

[10] else

[11] $finalize = True$

[12] According to the path in Path, decompose the graph G into spiders

(2) Prove the algorithm Based on the definition of spider decomposition, the vertex in R are the terminals, and can only be the foot (leave or center) of the spider. The algorithm firstly find the smallest spanning tree that contains all the vertices in R and may also contain some non-terminal vertices along the path. This step can ensure all the leave of the spanning tree are vertices in R.

Since the terminal node in the spider can only be leave or center, line [2] [3] assigned the leave of the tree as leave node and rest of the terminal as center. This step can ensure all the vertices in R are foot (leave or center) of the spider.

A spider can't have any joint path from leave to center. Line [4]-[11] is a finalize loop, which firstly assigns each leave node to the closest center node in set C, and records the all the paths from the source to the end. Then check in turn whether there are joint points on the path leading to the same end point. If so, add all joint points to the center set C, set the finalize value to false, and repeat the while loop. Otherwise, set the finalize value to true and end loop. This step can ensure there are no joints in the spider, moreover there won't be trivial spider as well.

Finally, we can use the record in *Path* and decompose G in to spiders of R. This shows that the input is the smallest spanning tree, no matter how G and R change, this algorithm can get the spider decomposition of R

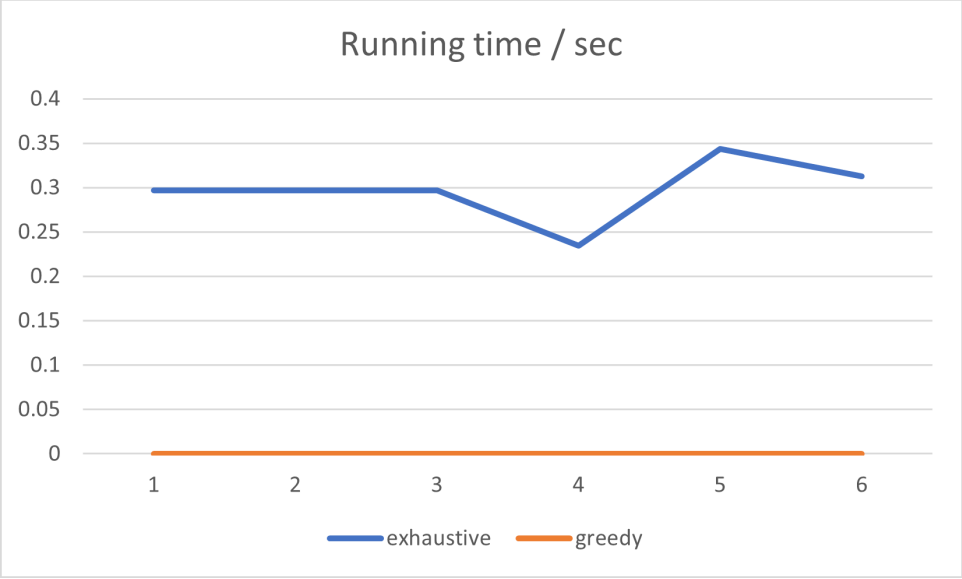
Since G is a connected graph, there always exist a smallest spanning tree of R. Therefore, the algorithm in (1) is feasible, and it can prove that G always contains a spider decomposition of R.

2.4 Question-4

I wrote two algorithms to solve set cover, greedy and exhaustive search. Below are the resulting graphs for both algorithms.

```
running time for greedy:  0.0 seconds
{'cover': [[54, 41, 57, 49, 34, 31, 73, 68, 14, 4, 45, 16, 19, 27, 25, 3, 38, 2, 13, 70, 52, 9, 12, 75], [38, 0, 14, 16, 13, 9, 12, 3, 56, 68, 70, 25, 49, 31,
 2, 34, 52, 75, 73, 19, 33, 21], [34, 25, 19, 31, 6, 73, 2, 0, 14, 70, 27, 75, 16, 68, 38, 21, 41, 45, 52, 54, 12, 4, 56, 59, 33]], 'cost': [4, 14, 45]}
running time for exhaustive:  0.359375 seconds
{'cover': [[34, 25, 19, 31, 6, 73, 2, 0, 14, 70, 27, 75, 16, 68, 38, 21, 41, 45, 52, 54, 12, 4, 56, 59, 33], [54, 41, 57, 49, 34, 31, 73, 68, 14, 4, 45, 16, 1
9, 27, 25, 3, 38, 2, 13, 70, 52, 9, 12, 75]], 'cost': [45, 4]}
```

The above picture shows the calculation results of the two algorithms for 30 items and 15 covers respectively. Obviously, although the result obtained by the greedy algorithm is not the optimal solution, the greedy algorithm is much faster than the exhaustive method.



References

[Vazirani] V. Vazirani, Approximation Algorithms, Springer, (2003).