**COMP 7270 Assignment 3 10 Problems 200 points No late submissions!**

**Due by 11:59 PM Tuesday 04/11**

**Upload your submission well before this deadline. Even if you are a few minutes late, as a result of which Canvas marks your submission late,** **your assignment may not be accepted**.

Instructions:

1. This is an individual assignment. You should do your own work. **Any evidence of copying will result in a zero grade and additional penalties/actions.**
2. Late submissions **will not** be accepted unless prior permission has been granted or there is a valid and verifiable excuse.
3. **Think carefully; formulate your answers, and then write them out concisely** using English, logic, mathematics and pseudocode (no programming language syntax).
4. Algorithms should be provided in numbered pseudocode steps.
5. **Type your answers in this Word document and submit it. If that is not possible, use a word processor to type your answers as much as possible (you may hand-write/draw equations and figures), turn it into a PDF document and upload**.

**1. (Multiple choice)**

Suppose q is a problem in NP but outside P. Suppose r is a NP-Complete problem. Explain why, if we show that any instance of r can be transformed to a corresponding instance of q in polynomial time, this means that EVERY problem in NP can then be transformed into q instance by instance.

A. Because r is NP-Complete, so it can be transformed in polynomial time to any and all problems in NP.

B. Because r is NP-Complete, any and all problems in NP can be transformed in polynomial time to it.

C. Because r is NP-Complete, any and all problems in NP can be transformed in polynomial time to it, and because of the polynomial time transformation from r to q, any and all problems in NP can be transformed in polynomial time to q.

D. Because if we show that any instance of r can be transformed to a corresponding instance of q in polynomial time, this means that r and q are one and the same.

E. none of the above.

Answer: **C**

**2. (Multiple choice) The Vertex Cover Problem (VCP):**

Determine whether a graph has a subset S with k nodes (k≤n, n=total # of nodes in the graph) such that each of its edges is connected to at least one node in this subset. VCP is in NP. Why? Because though no polynomial time algorithm to find a VC of size k in a graph is known, if a graph and a possible VC (a set S of k nodes in the graph, k≤ n) are given, one can easily write a polynomial time algorithm to check if those k nodes form a vertex cover or not. Which of the following is a correct strategy for this algorithm? Assume that the graph G is undirected, unweighted and represented as an Adjacency Matrix A, and the possible solution S is provided as a set of nodes.

A. For each node s in the set S, scan the matrix row corresponding to s and when a 1 is encountered in cell A[s,i], replace that with a 0 and also replace the 1 in A[i,s] with a 0. When rows corresponding to all nodes in S have been processed, scan the entire matrix A for any remaining 1's. If there are any 1's left, S is not a correct VC; if there are no 1's left, S is a correct VC.

B. For each node s in the set S, scan the matrix row corresponding to s and when a 0 is encountered in cell A[s,i], replace that with a 1 and also replace the 0 in A[i,s] with a 1. When rows corresponding to all nodes in S have been processed, scan the entire matrix A for any remaining 0's. If there are any 0's left, S is not a correct VC; if there are no 0's left, S is a correct VC.

C. Check if matrix A is symmetric. If it is, then S is a correct VC. If it isn't, S is not a correct VC.

D. For each node s in the set S, scan the matrix row corresponding to s and when a 1 is encountered in cell A[s,i], check if node i is in set S. If i is in set S then S is not a correct VC: report that and quit. If i is not a node in S then continue scanning. If after all rows corresponding to nodes in S have been scanned and the algorithm did not quit, then S is a correct VC.

E. All of the above strategies are correct.

**Answer: A**

**3.**

In class, we talked about HCP and ECP problem pair. They look very similar, but one has linear time solution, and the other is an NPC problem. Can you come up with **three** other problem pairs of such kind? i.e., the problem conditions are similar, yet one is a P problem, and the other is an NPC or even an NP-hard problem.

1.Shortest VS longest simple paths

2.Eular tour VS Hamiltonian cycle

3 2-CNF satisfiability VS 3-CNF satisfiability

**4.**

Let A[1], . . . , A[2k] be a list of values that are sorted in increasing order. Your goal is find the i ∈ [k] such that f(i) = A[i + k] − A[i] is maximized. Unfortunately you don’t have time to test all k pairs but luckily you are happy with an approximate solution. (1) Design a 2-approximation that only requires reading O(1) values from the list. (2) Design a (1 + ϵ)-approximation that only requires reading O(ϵ-1 log k) values from the list.

**Hint:** For the first part consider reading A[1], A[k], A[k + 1], and A[2k].

For the second part, one possible approach involves performing O(1/ϵ) binary searches. Remember to analyze the running time and prove the approximation ratio.

(1) 2-approximation means this solution is at least >= 2\* optimal solution

Algorithm-Approximate-solution

1 f(1)= A[k+1]- A[1]

2 f(k)= A[2K]-A[k]

3 return max(f(1), f(k))

prove: assume the optimal solution is A[i+k]-A[i], so A[i+k]-A[i]<= 2 max(f(1),f(k)).

First ,we assume f(1)>f(k),so A[i+k]-A[i]<=2(A[k+1]-A[1])

Assume A[i+k]-A[i]>2(A[k+1]-A[1]), because i⊆[1,k], this is a increasing order,

so A[i+k]-A[1]>=A[i+k]-A[i]>2A[k+1]-2A[1]

so A[i+k]-A[k+1]> A[k+1]-A[1].

Also i+k<=2k

So A[k+k]-A[k]>A[k+1]-A[1], which is contradict with our first assumption as f(1)>f(2),

So our assumption is wrong , so A[i+k]-A[i]<=2(A[k+1]-A[1]).

If we assume f(k)>f(1), vice versa.

This running time is O(n) , approximation ratio is 2

(2)

* BinarySearch - Maxize(A, k):
* Let q = 0
* i = k, j = 2k
* while (i < k)
* q1 = A[i + k] – A[i]
* q2 = A[j + k] – A[j]
* middle = i + (j - i)/2
* if (BinarySeach(A[middle], max(q1, q2)) == 1)
* q = max(q, max(q1,q2))
* else if (BinarySeach(A[middle], max(q1, q2)) > 1)
* i++
* else
* j--
* return q

**5.**

Let G = (V, E) be an undirected graph in which each node has degree at most d. Design a polynomial time algorithm that finds an independent set whose size is **at least** 1/(d + 1) times that of the largest independent set. Remember to analyze the running time and prove the approximation ratio.

1 While G(E,V) is not empty do

2 Let v be a node in G

3 S ← S ∪ {v}  
4 Remove v and its neighbors from G

5 end while

6 Output S

Running time = Ο(n/(d+1))

Analysis: once we removed a node and its neighbors , at most d+1 nodes will be moved . if n is the number of nodes , the total times to repeat this program is >= [n/(d+1)],which is the running time.

Ratio is 1/(d+1)

**6.**

**You need to do the Reading Assignment on** **Subset Sum first.**

The input to the Min-Squares problem is a set of positive integers x1, . . . , xn, r and L. We want to know if it is possible to partition x1, . . . , xn into r sets S1, S2, . . . , Sr such that



Prove that Min-Squares is NP-Complete via a reduction from Subset Sum. (Read the Hint in Q7).

Answer:

If we want to show this problem is NP-complete , following 4 things we need to do.

First we need to show there is a non-polynomial – time algorithm to solve this problem ,which means it belongs to NP

Second we need to show any NP-complete problem B can be reduced to A.

Third the reduction of B to A is polynomial time

Forth the original problem has a solution only B has a solution

Since SUBSET-SUM is defined as follows: Given a set X of integers and a target number t, find a subset Y ⊆X such that the members of Y add up to exactly t, assume S = {S1,S2,S3….Sr} , t = {t1,t2,t3…tr} for each subset S’, exist ,

so=x1+(x1+x2)+……(x1+x2+….xr)<=t1+t2+…tr. We now show that Subset-Sum can be reduced to Min-Squares. Given r subsets and the sum of each subset are t1,t2…tr. The Min-Squares is find the minimum result in t12, t22…tr2 and do t12+t22+t32+…tr2 we could get minimum square. There are some possibility ways to partition the x. If we separate each element to r parts like x1/r,x2/r...xn/r. Then we combine those parts into a subset in order to get r subsets with same sum value. Those r subsets have the same subset-sum which is equal to (X1+X2+…Xn)/r. The sum of all these r subsets should be in this case and it can be checked in polynomial time. If we partition them without this way, the sum of all subset should be , which can be checked in polynomial time However if we partition them in another way ,the sum should be . This formula is greater 2 or equal to2

We have shown that Subset Sum can be reduced to Min-Squares in polynomial time and the Min-Squares has a solution if and only if Subset Sum has a solution. I

Conclusion:

Since Subset Sum Problem is a known NP-Complete problem, this proves that Min-Squares is an NP-Complete problem too!

**7.**

Design (and analyze) a factor 4 approximation algorithm for Min-Squares, i.e.,

Minimizing the following.



Hint: For Q6 and Q7, note that

.

Answer : OPT <= 4\*2

(2<=4\*2

4\*(2<=2

4\*2 <=2

2 <= 2

partition[x1….xn] into 4 parts

Algorithm

let S be a new array

let result and sum be integers

result =0

for i=1 to 4

sum =0

for j =1 to n/4

sum = sum+random x

remove the x

si= sum \*sum

result = result +Si

return result