Assignment 4 - Written component

● Graded

Student

Tyler NGUYEN

Total Points

8 / 8 pts

Question 1

Question 1 **2** / 2 pts

1.1 **Question 1a)** **1.5** / 1.5 pts

✔ **– 0 pts** Correct

– 1.5 pts Assignment is submitted after the allowed number of late days

– 1.5 pts Incorrect

1.2 **Question 1b)** **0.5** / 0.5 pts

✔ **– 0 pts** Correct

– 0.5 pts Assignment is submitted after the allowed number of late days

– 0.5 pts Incorrect

Question 2

Question 2 **3** / 3 pts

2.1 **Question 2a)** **1** / 1 pt

✔ **– 0 pts** Correct

– 1 pt Assignment is submitted after the allowed number of late days

– 0.25 pts Wrong/missing 1-certificate provided. Recall that 1-certificate requires "Yes"/"True" outputs of the problems.

– 0.75 pts No verification algorithm.

– 0.5 pts 1-certificate is correct, but there is insufficient/excplcit justification on polynomial runtime of verification.

– 1 pt Incorrect

2.2 **Question 2b)** **2** / 2 pts

✔ **– 0 pts** Correct

– 2 pts Assignment is submitted after the allowed number of late days

– 0.4 pts Partially Correct, as you use T-Knapsack to perform a linear search, not binary search. Recall that K as input requires log(K) bits (in other words the input size is not linear to K).

– 1 pt The algorithm is not complete, or justification on running time is insufficient.

– 1.4 pts Algorithm is for 0-1 Knapsack, not Knapsack. In 0-1 Knapsack you are only allowed to take one item of each type, but in regular Knapsack you can take multiple items of the same type. This means the maximum is not simply the sum of the values.

– 2 pts Incorrect

Question 3

Question 3 **3** / 3 pts

✔ **– 0 pts** Correct

– 2 pts Wrong Reduction. Since we have two 0s at each step, we can skip more than once (Or jump more than 2 cells forward)

– 2 pts Wrong Reduction. It's not capturing problem constrains properly.

– 1.5 pts Partially correct.

– 1 pt Reduction not clear enough/Missing example.

– 3 pts Missing submission.

– 3 pts Assignment is submitted after the allowed number of late days

Questions assigned to the following page: 1.1, 2.1, and 1.2

CPSC 413 Assignment 4

CPSC 413 Assignment 4
Author: Tyler Nguyen
UCID: 30158563

Question 1:

a)

if there is an algorithm that solves k-Vertex Cover in polynomial time (in n), we can use this algorithm to solve the Minimum Vertex Cover by a brute force approach of trying all possible values of k from 1 to n, and determining if there is indeed a vertex cover of size k that exists. then the smallest k where the answer is yes would be seen as the size of the Minimum vertex Cover. this approach is polynomial since, we would need to iterate over all values of k that start from 1 to n because the smallest possible vertex cover will have at least one vertex and at most n vertices. then for each k, the given polynomial time algorithm is ran to determine if a vertex cover of size k does exists. if there is a vertex cover of size k, that k will be seen as a potential solution, and the smallest k that is a vertex cover will allow the algorithm to return yes.

now since we are given the polynomial time algorithm for k-Vertex Cover we can determine it has a time complexity of O(n^c) where c is some constant. then as stated for each k the polynomial time algorithm will be ran once, so O(n^c) will be ran once for each k. then since we have to try all k values that range from1 to n, we are running the k-Vertex Cover algorithm n times. which results in a total time complexity of n * O(n^c) which is O(n^(c+1)) which is a polynomial time complexity.

b)

the reduction is from Minimum Vertex Cover to k-Vertex Cover.

Question 2:

a)

i will be proving that T-Knapsack is in NP by giving a 1-certifcate and then describing an algorithm that can verify it in polynomial time which was stated in

Questions assigned to the following page: 2.1 and 2.2

the question.

1-certificate: a list that includes which items are selected to be placed into the knapsack such that a selection S of items from the original list of n items, $S \subseteq \{1,2,...n\}$, where each entry is either 1 or 0 meaning that the item is selected or it is not selected, respectively. Then for each selected item in the subset the corresponding weights w_i and values v_i are included.

verification algorithm: now for the above described 1-certificate, a description of an algorithm that can verify it can be shown in 5 steps: (1) calculate the total weight of the items in the subset, and (2) if the total weight exceeds K then reject the certificate, after (3) calculate the total value of the items in the subset, and (4) if the total value is less than T then reject the certificate. lastly, (5) if both of the conditions are not rejected then the algorithm will accept the certificate. by summing the weights of the items in the certificate which takes n addition operations, this part of the algorithm will take O(n) time. similarly with summing the values of the items in the certificate which takes n additions which is O(n) time, then comparing the the respective totals to K and T will take O(1) time. Therefore the total verification time is O(n) + O(n) + O(1) + O(1) which is O(n) which is linear but can be seen as well as polynomial time since it is a specific type of polynomial function where the polynomial has a degree of 1 and linear time is also a subset of polynomial time. meaning that the verification algorithm for T-Knapsack runs in polynomial time.

example:

- given items: [(w_1,v_1),(w_2,v_2),...,(w_n,v_n)]
- selected items: S = {1,0,1,...,0} with included weights and values of each selected item (1)
- total weight calculated: the respective weights added up, reject certificate if exceeds k
- total value: the respective values added up reject certificate if less than T
- if both sums are within the correct constraints accept certificate

therefore since T-Knapsack has a verification algorithm that operates in polynomial time, this shows that T-Knapsack is in NP.

b)

Question assigned to the following page: 2.2

if there is an algorithm that solves T-Knapsack in polynomial time we can use this algorithm to solve Knapsack. call this algorithm, A. now we can use this algorithm A to perform a binary search on the value T in order to determine the maximum value that is achievable that does not exceed the weight limit K. the upper bound for the binary search is the sum of all values of v_i because it is not possible to get a higher value. the lower bound can start of as 0. now with our lower and upper bounds, the binary search will proceed by determining the middle value of the current range denoted as “mid”. where “mid” represents the target value of T from T-Knapsack. next using the polynomial time algorithm A, we can solve T-Knapsack to figure out if there is a subset of items that is able to fit within the knapsack capacity K which also has a combined value of at least “mid”. now if algorithm A does determine that there is a subset that exists, we now know that “mid” is attainable which means there could be a higher value that is also obtainable, so we would have to change the lower bound of the search to “mid + 1” which discards all values lower than “mid + 1” since we have already determine that “mid” is attainable so we continue the search to find a value larger than “mid”. however if algorithm A does not determine that there is a subset that exists, we now know that we have looked to high so we need to change the upper bound to “mid - 1” and continue the search to find a maximum value that is less than “mid”. now the binary search will keep going until the lower bound exceeds or meets the upper bound. where the largest value of “mid” that was determined to be attainable by algorithm A before the lower bound was adjusted above it will be the total value that can be placed in the knapsack that does not exceed the weight capacity K. when the binary search does finish the approach would have found the maximum value since, it cannot go higher without exceeding the capacity K and it would have the highest possible value that algorithm A has verified as obtainable.

this approach is polynomial time since given that each items value, v_i, is representable as a 32 bit integer, the sum of all item values denoted as N gives us the initial upper bound for the binary search. the binary search algorithm divides the search space in half with each iteration. then binary search will take O(log N) iterations to find the target value or figure out that it does not exists. since in each iteration the solution to T-Knapsack is used to decide whether a subset with the value “mid” is obtainable. now because we are given an algorithm that can solve T-Knapsack in polynomial time, means that during each iteration of the search the

Questions assigned to the following page: 2.2 and 3

time need to determine whether the current value “mid” is obtainable is some type of polynomial function of the input size lets say P(n) such that n is the number of items or the size of the input to T-Knapsack. therefore the total time to find the optimal solution to Knapsack using binary search and the T-Knapsack algorithm can be seen as O(log N) * P(n) which results in polynomial time. since log N and P(n) are polynomial with respect to the input size. this polynomial time approach shows that if T-Knapsack which is NP-hard, can be solved in polynomial time, then Knapsack can also be solved in polynomial time which informally shows that Knapsack is NP-complete.

Question 3:

algorithm:

for each lily pad x_i that is in Problem A, we need to create a cluster in Problem B, where:

set r_i = x_i, this allows Fred to choose on the lily pad i in Problem A

set g_i = 0, to ensure that Fred never chooses a greed lily pad, which simulates the decision to skip a lily pad in Problem A

set b_i = the (i+1)th lily pads value of x_(i+1) only if the index i is odd, but when the index is even set b_i = 0 to prevent Fred from making two consecutive jumps on blue lily pads

then ensure the last cluster allows a choice for Freds final jump by having non-zero value sfor both r and b if the sequence ends on and odd index or just for r if it ends on and even index.

example 1:

input of [4,-3,5], where we initialize r, g, b arrays for Problem B to zero, then apply the reduction rules to set the values to obtain the Problem B arrays:

r: [4, 0, 5]

g: [0, 0, 0]

b: [0, -3, 0]

now to see that Freds path in Problem B with the correct reduction will mirror the optimal decisions in Problem A:

Question assigned to the following page: 3

1. Fred jumps to r_1 with a value of 4, since this lily pad has the highest value in that cluster

2. he skips r_2 since its the same colour as r_1 and the score is 0, he cannot jump on b_2 because it has a negative value of -3 so his only option is to jump onto g_2 which is 0 since will not decrease his score.

3. then from g_2 he can jump to r_3 with a value of 5, since this lily pad has the highest value in the current cluster

this sequence gives Fred a total of 9 points. which matches the optimal path in Problem A, where Fred jumps from the first lily pad to the third skipping the second lily pad which has the negative value that ultimately would decrease his score. this mapping ensures that the optimal choices and scores are consistent between Problems A and B

example 2:

input of [7,2,3,2], where we initialize r, g, b arrays for Problem B to zero, then apply the reduction rules to set the values to obtain the Problem B arrays:

r: [7, 0, 3, 0]

g: [0, 0, 0, 0]

b: [0, 2, 0, 2]

1. Fred jumps to r_1 since it has the highest value of 7 in the cluster

2. Fred cannot jump on r_2 since it has the same colour as r_1, he also will not jump to g_2 because it has a value of 0, which means he has to jump to b_2 with a value of 2

3. next Fred has to change to a different colour, so he cannot jump to b_3, so he jumps to r_3 and not g_3 since r_3 has a higher value

4. lastly Fred cannot jump on r_4 since it has the same colour as r_3 and its value is also 0, therefore he has to jump to b_4 and no g_4 because b_4 has a higher value than g_4

this sequence gives Fred a total of 14 points. which matches the optimal path in Problem A, where Fred jumps from the first lily pad, then the second lily pad, then the third lily pad and lastly the fourth lily pad. this mapping ensures that the optimal choices and scores are consistent between Problems A and B

Question assigned to the following page: 3

now from these 2 examples, shows that this reduction approach works and maintains the consistent decision making process and the same maximum score between Problems A and B which is shown in the two examples above

this reduction runs in O(n) since the algorithm starts by initializing three new arrays r, g, and b based on the length of the input array from Problem A. where for each element x_i in the input array of Problem A, the algorithm will set the corresponding values in the r, g and b arrays, which involves iterating through each element of the input array exactly once hence we get O(n) time complexity of the reduction