Homework 10 Problem 1

Jin, Yucheng (yucheng9) Fan, Wenyan (wenyanf2) Lu, Hanxiao (hl4)

December 10, 2019

1.

(a) DNF-SAT could be true if there exists at least one clause that can produce a 1. If any clause contains clause like a and ~a, whatever a's value is will not produce a 1 in this clause. In other case, assigning variables with 1 or 0 could makes clause produce 1. Assume we have m clause in the formula L and each clause contains at most k literals.

```
DFN_SAT(L):
    For each clause in L:
        If clause does not contain pattern a ^ ~a:
            Return true
    Return false;
```

The runtime to check each clause is O(k^2) and runtime for checking the whole formula is O(m*k^2). The runtime is polynomial so DNF-SAT is NP-hard.

(b) We demonstrate a way that reduce 3SAT to DNF-SAT. Suppose 3SAT is in the form of $(a \vee b \vee c) \wedge (d \vee e \vee f) \wedge \ldots$, we transform it by using distributive law to the form $(a \wedge d) \vee (a \wedge e) \vee (a \wedge f)$
$\vee (b \wedge d) \vee (b \wedge e) \vee (b \wedge f) \vee (c \wedge d) \vee (c \wedge e) \vee (c \wedge f) \vee \ldots$..

We claim that DNF-SAT is satisfiable if and only if 3SAT is satisfiable.
=> Suppose we are given a DNF-SAT and there exists an assignment that makes formula satisfibale. We transforming DNF-SAT by distributive law to 3SAT and apply the assignment to 3SAT. Since these two formula are equal, we conclude 3SAT is also satisfiable when applied with this assignment.
<= Suppose we are given a 3SAT and there exists an assignment that makes formula satisfibale. We transforming 3SAT by distributive law to DNF-SAT and apply the assignment to DNF-SAT. Since these two formula are equal, we conclude DNF-SAT is also satisfiable when applied with this assignment.

The runtime for this transformation is 3^m where m is the number of clause in the formula. The transformation is exponential.

(c) The algorithm for this decision problem is polynomial. But finding the exact solution to the problem needs exponential runtime. Therefore NP != P in this case.

CS/ECE 374 B Fall 2019
Homework 10 Problem 2

Jin Yucheng (yucheng9)
Fan Wenyan (wenyanf2)
Lu Hanxiao (hl4)

Solution:

(a) Suppose $X[0...n-1]$ is given as an array of positive integers, there are the two main steps to partition X into two subarrays A and B such that $\Sigma A = \Sigma B$:

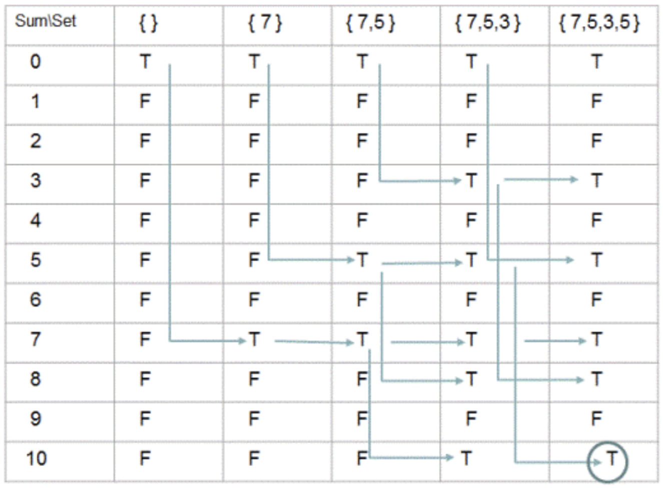
(1) Decide whether X is partitionable, this can be done by calling the subroutine Partitionable(X). If Partitionable(X) returns False then there are no two subarrays with equal sum.

(2) If Partitionable(X) returns True, calculate the sum of elements in X, find a subarray with sum equal to half of the sum of X.

The problem can be solved using dynamic programming. Construct a 2-dimensional array A of size $\lceil \frac{1}{2}\text{sum}(X)\rceil \times (n+1)$. $T[i][j]$ is True when $X[0]+X[1]+...+X[j-1]=i$, $T[0][0]$ is True and $T[i][0]$ is False if $i \neq 0$.

```
def PartitionTwo($X[0...n-1]$):
    1 : Calculate the sum of all elements in $X$
    2 : Construct a 2-dimensional array $T$ of size $\lceil \frac{1}{2}\text{sum}(X)\rceil \times (n+1)$
    3 : Initialize entries in the top row as True
    4 : Initialize entries in the leftmost column as False
    5 : $T[0][0] \leftarrow$ True
    6 : for $i \leftarrow 1$ to $\lceil \frac{1}{2}\text{sum}(X)\rceil$:        # build the arrary in bottom up manner
    7 :     for $j \leftarrow 1$ to $n+1$:
    8 :         $T[i][j] \leftarrow T[i][j-1]$
    9 :         if $i \geq X[j-1]$: $T[i][j] = (T[i][j]$ OR $T[i-X[j-1]][j-1])$
    10 : return $T[\lceil \frac{1}{2}\text{sum}(X)\rceil][n]$
```

We can replace line 8 and 9 by implementing the subroutine Partitionable, and we can retrieve A and B by recording the column indices, an example is as follows,



Sum\Set	{ }	{ 7 }	{ 7,5 }	{ 7,5,3 }	{ 7,5,3,5 }
0	T	T	T	T	T
1	F	F	F	F	F
2	F	F	F	F	F
3	F	F	F	T	T
4	F	F	F	F	F
5	F	F	T	T	T
6	F	F	F	F	F
7	F	T	T	T	T
8	F	F	F	T	T
9	F	F	F	F	F
10	F	F	F	T	T

We make $\Theta(\frac{1}{2}\text{sum(X)} \cdot n)$ calls to Partitionable and the time complexity is therefore $\Theta(\frac{1}{2}\text{sum(X)} \cdot n)$. We can also just use recursion and call Partitionable $\Theta(2^n)$ times but it is not as efficient as $\Theta(\frac{1}{2}\text{sum(X)} \cdot n)$.

Reference

[1] IDeserve. "Set Partition Problem | Dynamic Programming". Web. <https://www.ideserve.co.in/learn/set-partition-problem-dynamic-programming>

[2] GeeksforGeeks. "Partition problem | DP-18". Web. <https://www.geeksforgeeks.org/partition-problem-dp-18/>

(b)This is a k-partition problem where $k = n$ and each set has exactly 2 elements. We can solve this problem recursively, we keep an array for sum of each partition and a boolean array to check whether an element is already taken into some partition or not (see Reference[3]). Suppose we are given an array $X[1...2n]$.

First we check base cases. If k is 1 (which is not the case in this problem since $k = 2$), then the array X itself is our answer. If $2n < k$ (which is not the case in this problem if $n \geq 1$), then it is not possible to divide the array into subsets with equal sum. If sum of elements in X is not divisible by k, then it is not possible to partition X into subsets with equal number of elements and equal sum. We will proceed only if k divides the sum of elements in X. Our goal reduces to divide X into k parts where sum of each part should be $\frac{sum(X)}{k}$ (see Reference[3]).

For this question, we construct an array T of $O(n^2)$ size, where each entry, $T[i][j]$ is True if the i^{th} and j^{th} elements in X is a valid subset. Assume Partitionable tells wether $X - X[i] - X[j]$ is partitionable, then we can fill in the array within $O(n^2)$ time.

```
def PartitionK($X[1...2n], k = n$):
    1 : Calculate the sum of all elements in $X$, $\frac{X}{k}$ is the sum of every subset
    2 : Construct a 2-dimensional array $T$ of size $O(n^2)$
    3 : Initialize all entries as False
    4 : for $i \leftarrow 1$ to $n$:
    5 :     for $j \leftarrow 1$ to $n$:
    6 :         if $X[i] + X[j] = \frac{X}{k}$ and Partitionable($X - X[i] - X[j]$) returns True:
    7 :             $T[i][j] \leftarrow$ True
```

If there is no way to partition X into $k = n$ subsets with equal sum, then all entries are False. Also, there may exist multiple ways of partition, and we can see these multiple ways from T.

Reference

[3] GeeksforGeeks. "Partition of a set into K subsets with equal sum". Web. <https://www.geeksforgeeks.org/partition-set-k-subsets-equal-sum/>

(c) I can show PARTITION is NP-complete, because if a problem is NP-complete, then it is NP-hard, so we can prove PARTITION is NP-hard by proving it is NP-complete.

First we show that partition is in NP. Our certificate will be a set of subsets, S, of X. We can verify in polynomial time whether S satisfies the constraint of PARTITION.

Now we show that PARTITION is NP-complete via a reduction from SUBSETSUM, which can be expressed as, is there a subset whose sum is equal to m in X? Let (X, m) be an arbitrary instantance of SUBSETSUM. We transform (X, m) into an instance T of the PARTITION problem in the following way (see Reference[4]):

Let T be the set X plus two more elements: $t_{n+1} = 2m + k, t_{n+2} = 3m - k$.

T can be constructed easily in polynomial time. We can conclude that T is a true instance of PARTITION if and only if (X, k) is a true instance of SUBSETSUM.

Assume (X, k) is a true instance of SUBSETSUM. Then there is a subset S of X such that S has size k. Then $((X - S) \cup \{a_{n+1}\}$, $S \cup \{a_{n+2}\})$ is a PARTITION of T. Assume that T is a true instance of PARTITION. Then there exists a partition (S_1, S_2) of T with equal weight. Because all the elements in T is $6m$. So both a_{n+1} and a_{n+2} cannot appear in the same subset. Therefore, we can safely conclude that a_{n+1} is in S_1 and a_{n+2} is in S_2. Then by constructing S_2 - $\{a_{n+2}\}$ such that the new set has sum k (since S_2 has weight $3m$). It follows that there is a subset of weight k in X. Therefore, The given reduction reduces SUBSETSUM to PARTITION, and since SUBSETSUM is NP-complete, it follows that PARTITION is NP-hard (we can finish our proof here). We have shown that PARTITION is in NP, so PARTITION is NP-complete(see Reference[4]).

Reference

[4] Erin McLeish. "Answers to Assignment 4". Web. <https://www.cs.mcgill.ca/ jmerce1/a4ans.html>

■

CS/ECE 374 B Fall 2019
Homework 10 Problem 4

Jin Yucheng (yucheng9)
Fan Wenyan (wenyanf2)
Lu Hanxiao (hl4)

Solution:

For a positive integer n, we can find its prime factors in $O(sqrt(n))$ time. We first find the number of 2's as prime factors, and iterate to $sqrt(n)$.

```python
import math

def Factorize(n):

    for i in range(2, int(math.sqrt(n))+1):
        count = 0;
        while (n % i == 0):
            count += 1;
            n = int(n / i);
        if (count > 0):
            print(i, count);

    # if n is a prime number.
    if (n > 2):
        print(n, 1);
```

```python
a = 21414
Factorize(a)
```

```
2 1
3 1
43 1
83 1
```

```python
b = 1251521358
Factorize(b)
```

```
2 1
3 1
47 1
4438019 1
```

This algorithm is efficient for moderately large numbers, however, since

412023436986659543855531365332575948179811699844327982845455626433876445565248426198098870423161841879261420247188869492560931776375033421130982397485150944909106910269861031862704114880866970564902903653658867433731720813104105190864254793282601391257624033946373269391

is too large, we cannot factorize it within a week. ■

Homework 10 Problem 3

Jin, Yucheng (yucheng9) Fan, Wenyan (wenyanf2) Lu, Hanxiao (hl4)

December 10, 2019

Reference

[1] Hiro Ito, Stefan Langerman, Yuichi Yoshida. (2012). "Algorithms and Complexity of Generalized River Crossing Problems". *International Conference on Fun with Algorithms*.

Solution

This solution is mainly based on Hiro Ito, Stefan Langerman, and Yuichi Yoshida's work, Algorithms and Complexity of Generalized River Crossing Problems.

Below is a restatement of the restatement:

We are given a graph G = (V, E) and the number k. The problem can be described as defining a sequence of subsets of vertices $X_0, \ldots, X_{2m+1}$.

We can think of defining a sequence of subsets of vertices as describing the intermediate state of the solution of the well-known wolf-goat-and-cabbage puzzle. In order to solve the problem better, we will directly study how to get the solution of the well-known wolf-goat-and-cabbage puzzle. The two are equivalent.

First we give a formulation of our problem:

CHARON

INSTANCE: A set of "drivers" D, a set of "customers" C, a family of "forbidden sets" for the "left bank" F_L, one for the "right bank" F_R, one for the "boat" F_B, the size of the "boat" k, the bound of the number of "transportations" T.

QUESTION: Is there a way to transport all of the drivers and customers from the left bank to the right bank of a river using a boat?

RESTRICTIONS:
1. Initially all drivers, all customers, and the boat are on the left bank.
2. The capacity of the boat is k, i.e., it can transport at most k persons (drivers and customers) at a time.
3. Only drivers can operate the boat, i.e., at least one driver must be on the boat in any transportation.
4. It is forbidden to transport exactly the members of a forbidden set in F_B in the boat.
5. It is forbidden to leave exactly the members of a forbidden set in F_L (resp., F_R) on the left bank (resp., the right bank).
6. The number of transportations is at most T . (Note: One way is counted as one transportation, i.e., a set of going and returning is counted as two transportations.)

For example, the Wolf-Goat-Cabbage Problem is formulated as follows:
$D = \{\text{Man}\}$,
$C = \{\text{Wolf, Goat, Cabbage}\}$
$F_L = F_R = \{\{\text{Wolf, Goat}\}, \{\text{Goat, Cabbage}\}, \{\text{Wolf, Goat, Cabbage}\}\}$,

$F_B = \emptyset$,

$T = \infty$.

Theorem. CHARON is NP-hard.

We use the following problem, which is known to be NP-complete.
(Refer to "12.13 Other Useful NP-hard Problems" in Chapter 12 "NP-Hardness")

EXACT3DIMENSIONALMATCHING (X3M)

INSTANCE: A set $M \subseteq W \times X \times Y$, where W, X, and Y are disjoint sets having the same number q of elements.

QUESTION: Does M contain a matching, that is, a subset $M \subseteq M'$ such that $|M'| = q$ and no two elements of M' agree in any coordinate?

Proof of Theorem: First we prove for the case of $k = 3$. Let (W, X, Y, M) is an instance of X3M. We assume that $q = |W| = |X| = |Y|$ is even. (Otherwise it is enough to add dummy elements $w' \notin W$, $x' \notin X$, $y' \notin Y$, and (w' , x' , y') in W, X, Y, and M, respectively.) Give numbers to the elements of W as $\{w_1, ..., w_q\}$.
We construct an instance (D, C, F_B, T) of CHARON as follows.

$D = W \cup \{w_0\}$ $(w_0 \notin W)$,

$C = X \cup Y$,

$F_B = \{\{p, p', p''\} \mid p, p', p'' \in D \cup C\} - M - \{\{w_0, w_{2i\_1}, w_{2i}\} \mid i \in \{1, ..., q/2\}\}$,

$T = 3q - 1$.

We prove the equivalence of them.

(i) Assume that there is a matching $M' \subseteq M$. Let $M \ \ = \{(w_i, x_i, y_i) \mid i = 1, ..., q\}$ wlog. The way of a feasible transportation is as follows:
All customers of X and Y are moved to the right bank by the following way:

$\{w_1, x_1, y_1\} \in M'$ moves to the right bank, and w_1 moves back to the left bank.

$\{w_2, x_2, y_2\} \in M'$ moves to the right bank, and w_2 moves back to the left bank.

.

.

.

$\{w_q, x_q, y_q\} \in M'$ moves to the right bank, and w_q moves back to the left bank.

And all drivers, $w_0, w_1, \ldots, w_q$ are moved to the right bank by the following way:
$\{w_0, w_1, w_2\}$ moves to the right bank, and w_0 moves back to the left bank.
$\{w_0, w_3, w_4\}$ moves to the right bank, and w_0 moves back to the left bank.

.

.

.

$\{w_0, w_{q/2\_3}, w_{q/2\_2}\}$ moves to the right bank, and w_0 moves back to the left bank.

$\{w_0, w_{q/2\_1}, w_{q/2}\}$ moves to the right bank.

This process requires $3q - 1$ transportations.

(ii) Assume that there is a way to move them in at most $3q - 1$ transportations. At most three persons can be moved from the left bank to the right bank at a time, and at least one person must be moved back from the right bank to the left bank when the boat returns. Thus the number of persons on the right bank increases at most two by a pair of going and returning (two transportations). By considering the last three persons can share the boat and they don't need to back the boat to the left bank, we see that at most $3q + 1$ persons can be moved in $T = 3q - 1$ transportations. We have $3q + 1$ persons and thus three persons have to share in every transportation from the left bank to the right bank. From this, any customer in X and Y cannot move back from the right bank to the left bank. Consequently, the set of transportations including customers in X and Y must be a matching in M.
Thus we proved that it is NP-hard if $k = 3$. For $k > 4$, the same proof works by separating each y_i into $k - 2$ elements $y^1_i, \ldots, y^{k-2}_i$ and M is replaced by $M^* := \{\{w, x, y^1_i, \ldots, y^{b-2}_i\} | \{w, x, y_i\} \in M\}$.