CSE431 HW5

Siqi Cheng, 50388579

Problem 1: (1):

Certificate: A set of 3 tuples representing joints covering the entire graph.

Verifier Algorithm:

We have to make sure each vertex is in one 3-tuple only, and verify for each 3-tuple (vi, vj, vk), (vi, vj) and (vj, vk) are edges.

```
def verifier(G, joints):
    V, E = G
    covered_vertices = empty set
    for joint in joints:
        if len(joint) != 3:
            return False
        vi, vj, vk = joint
        if not ((vi, vj) in E or (vj, vi) in E) or not ((vj, vk) in E or (vk, vj) in E):
         Add vi, vj, vk to covered_vertices
    If covered_vertices = V:
        Return True
    Else:
        Return False
```

(2):

Certificate: A 2 × 2 submatrix of $\begin{bmatrix} \mathbf{1} & \mathbf{0} \\ \mathbf{0} & \mathbf{1} \end{bmatrix}$ in the original matrix.

Verifier Algorithm:

```
Function verifier(matrix, i, j):
    if matrix[i][j] == 1 and matrix[i+1][j] == 0 and matrix[i][j+1] == 0 and matrix[i+1][j+1] == 1:
        return True
    return False
```

(3):

Certificate: A sequence of moves (UP, DOWN, LEFT, RIGHT) from the starting position to the destination.

Verifier Algorithm:

```
Function verifier(grid, moves):
    position = starting position
    orientation = standing
    For each move in moves:
        if move == "UP":
            if orientation == standing:
                position.y -= 2
                orientation = horizontal
            elif orientation == horizontal and position is horizontal on y-axis:
                position.y -= 1
                orientation = standing
            else:
                position.x -= 1
        if not position_is_valid(grid, position, orientation):
            return False
    return (position is at destination and orientation is standing)
Function position_is_valid(grid, position, orientation):
    Check if the block, given its position and orientation, is on valid cells of the grid.
    Return True if valid, otherwise False
```

(4):

Certificate: A subset of k vertices that forms an independent set in G.

Verifier Algorithm:

```
Function verifier(G, S, k):
   if |S| != k:
      return False
   for every pair of vertices (u, v) in S:
      if (u, v) is an edge in E:
         return False
   return True
```

Problem 2:

(1): Given a 3-SAT instance of ϕ = (x1 ∨ x2 ∨ x3) ∧ (x1 ∨ x2 ∨ x3), what is the SUBSET-SUM instance we get using the above reduction? Is it a yes-instance or a no-instance?

Given:

n = 3(x1, x2, x3)

m = 2(2 clauses)

For x1:

a1 = 10^(m+1) + sum(10^1 + 0) = 1010

b1 = 10^ (m+1) +sum(0 + 10^2) = 1100

For x2:

a2 = 10^ (m+2) + sum(10^1 + 10^2) = 10^4 + 110 = 10110

b2 = 10^ (m+2) + sum(0 + 0) = 10^4 +0 = 10000

For x3:

a3 = 10^(m+3) + sum(0 + 0) = 10^5 +0 = 100000

b3 = 10^(m+3) + sum(10^1 + 10^2) = 10^5 +110 = 100110

For clause c1 & c2:

d1 = h1 = 10^1 = 10

d2 = h2 = 10^2 = 100

The set S is:

{1010,1100,10110,10000,100000,100110,10,10,100,100}

For the target t:

$$\sum_{i=1}^{n} 10^{m+i} + 3 * \sum_{j=1}^{m} 10^{j}$$

$\sum_{i=1}^{3} 10^{2+i} + 3 * \sum_{j=1}^{2} 10^{j}$ = 10^3 +10^4 +10^5+3(10^1+10^2)=
1000+10000+100000+330=111330

b3 + a2 + b1 + d1 = 100110+10110+1100+10=111330

The subset {b1, a2, b3, d1} from S sums up to t = 111330, so the SUBSET-SUM instance derived from the 3-SAT problem is **yes instance**

(2): 2. Given a 3-SAT instance of ϕ = (x1 ⋁ x2) ⋀ (x1 ⋁ x2) ⋀ (x1 ⋁ x2) ⋀ (x1 ⋁ x2)

Given: n=2 (x1 & x2)

m=4 (for the 4 clauses)

For x1:

a1 = 10^(m+1) + sum(10^1 + 10^2 + 0 + 0)= 10^5+10 +100 =100110

b1 = 10^(m+1) + sum(0 + 0 + 10^3 + 10^4)= 10^5 + 10^3 +10^4 = 111000

For x2:

a2 = 10^(m+2) + sum(10^1 + 0 + 10^3 + 0)= 10^6+10 +1000 =1001010

b2 = 10^(m+2) + sum(0 + 10^2 + 0 + 10^4)= 10^6 + 10^2 +10^4 = 1010100

For the clauses:

d1=h1=10

d2=h2=100

d3=h3=1000

d4=h4=10000

The set S is:

S={100110, 111000, 1001010, 1010100, 10, 10, 100, 100, 1000, 1000, 10000, 10000}

For the target t:

$$\sum_{i=1}^{n} 10^{m+i} + 3 * \sum_{j=1}^{m} 10^{j}$$

$\sum_{i=1}^{2} 10^{4+i} + 3 * \sum_{j=1}^{4} 10^{j}$ = 10^6 +10^5 +3(10^1+10^2+10^3+10^4)= 1133330

We cannot find any set of number sum equal to t 1133330, thus, the SUBSET-SUM instance derived from the 3-SAT problem is a no-instance, indicating that the 3-SAT formula ϕ is unsatisfiable.

Problem 3

(1):

Input:

A set V of loot boxes, where each loot box i has an integer value vi

Output:

Does there exist a subset V' ⊆ V such that:

$$\sum_{vi \in V'} vi = \frac{1}{2}\sum_{i=1}^{m} vi$$

This problem called **Bank Robbery Mission problem.**

(2):

Certificate: The subset V′ ⊆ V which sums to half the total value of all loot boxes is the certificate.

Verifier Algorithm: Given an instance of V and a certificate

1. V' is a subset of V.
2. The sum of elements in V' is equal to half the total value of all elements in V.

```
function verifier(V, V_prime):
  total_value = sum(V)
  if not all(v in V for v in V_prime):
    return False
  return sum(V_prime) == total_value / 2
```

(3):

No, this problem is not in P. As we can see, the bank robbery mission problem is NP-hard.

we can use a direct polynomial-time reduction from the Partition Problem.

An instance of the Partition Problem, a set W of integers. Use W as our set of loot boxes V for the Bank Robbery Mission problem. Each integer w in W becomes a value vi in V. A solution to the Partition Problem (two subsets with equal sums) corresponds directly to a solution to the

Bank Robbery Mission problem (a subset whose sum is half of V's total value). Therefor, Bank Robbery Mission problem is NP-hard, since the Partition Problem is NP-complete.