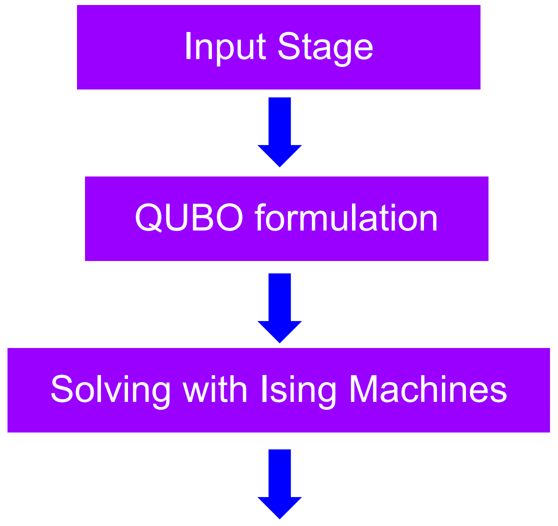
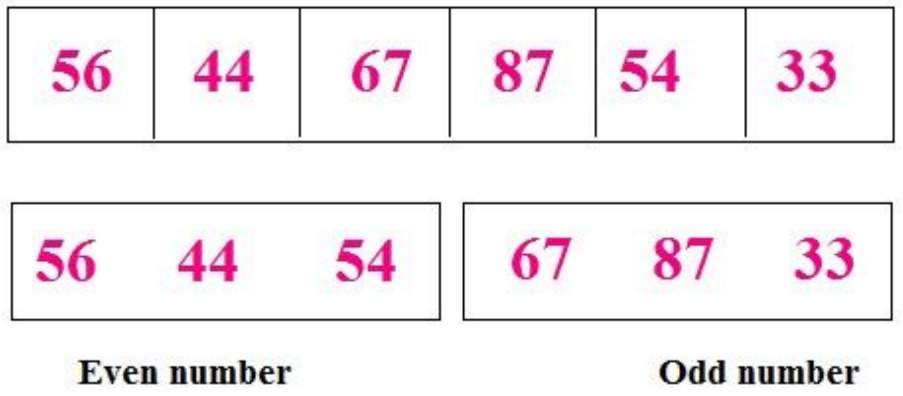
Even - Odd Classification On Annealing Machines

General strategy for solving a combinatorial optimisation problem on Ising machines with QUBO or Binary Quadratic matrix

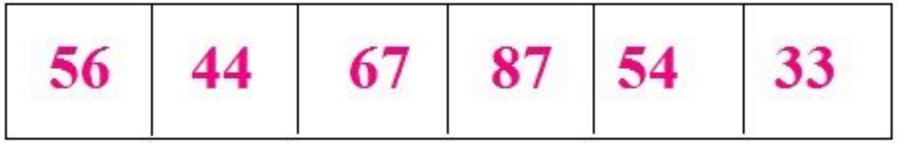




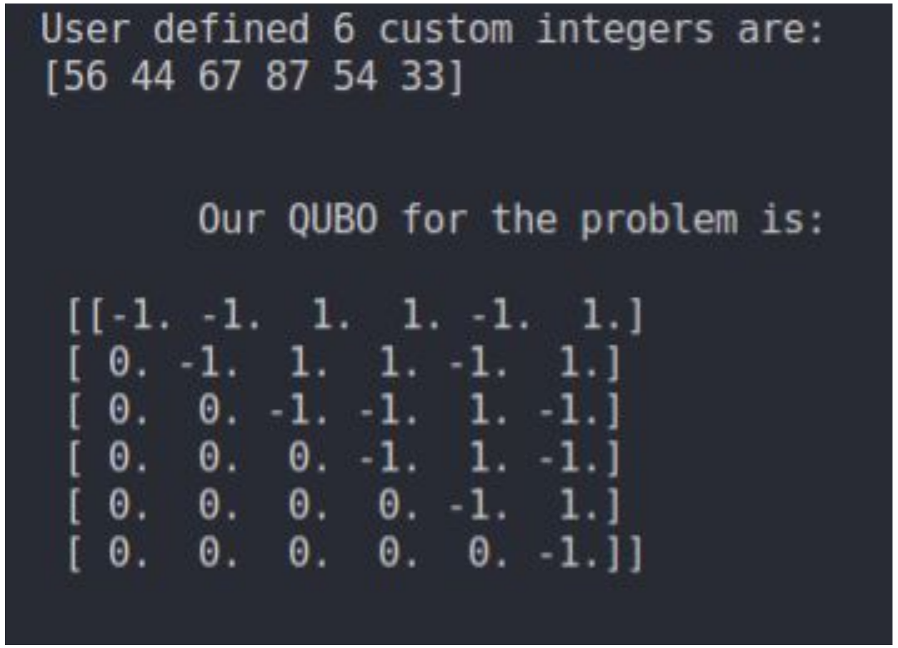
- Suppose an array of positive integers is given.
- Our aim is to finding the even numbers and odd number and grouping them in two different subsets.

Elements of the QUBO matrix or Binary Quadratic Matrix will be given by,

$$m_{ij} = (-1)^{(1+N_i+N_j)}$$



56	44	67	87	54	33

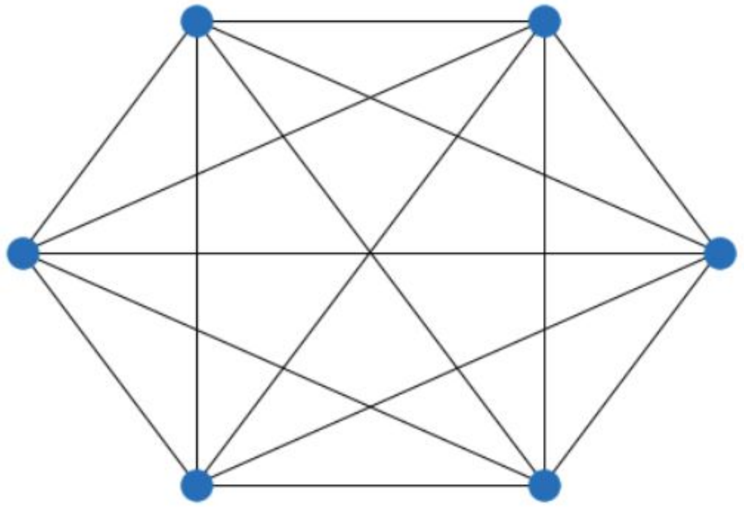


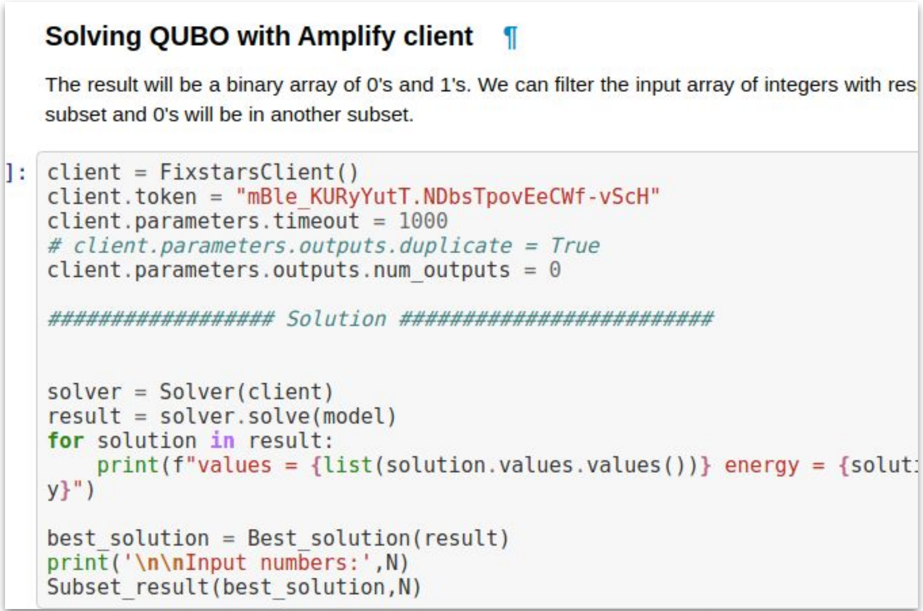
```
User defined 6 custom integers are:
[56 44 67 87 54 33]


       Our QUBO for the problem is:

 [[-1. -1.  1.  1. -1.  1.]
 [ 0. -1.  1.  1. -1.  1.]
 [ 0.  0. -1. -1.  1. -1.]
 [ 0.  0.  0. -1.  1. -1.]
 [ 0.  0.  0.  0. -1.  1.]
 [ 0.  0.  0.  0.  0. -1.]]
```

QUBO for the Input array

Every problem on Ising machines can be converted into graph.

- Every graph **for this problem** is a COMPLETE all to all connected graph.
- Edges have weights either 1 or -1.

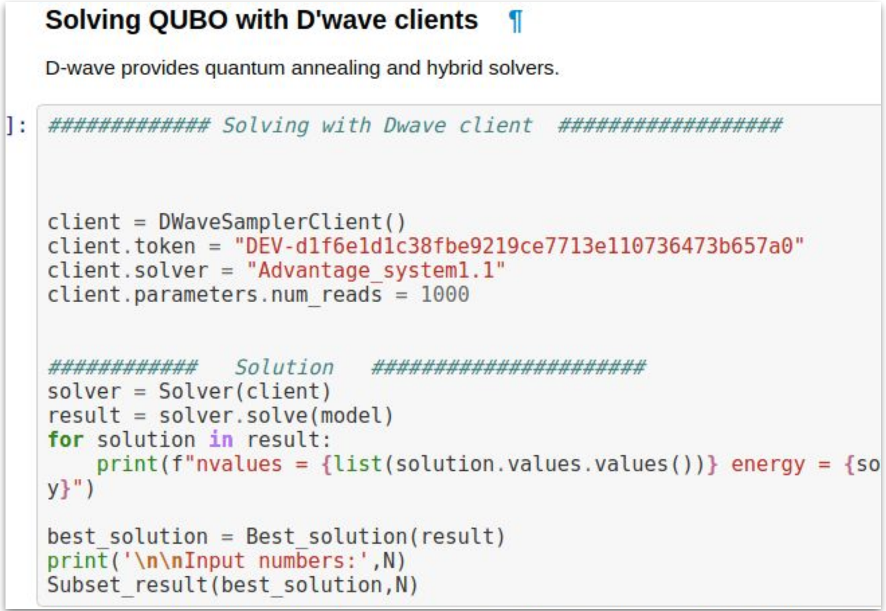




Solving QUBO with Amplify client ¶

The result will be a binary array of 0's and 1's. We can filter the input array of integers with res
subset and 0's will be in another subset.

```
]: client = FixstarsClient()
   client.token = "mBle_KURyYutT.NDbsTpovEeCWf-vScH"
   client.parameters.timeout = 1000
   # client.parameters.outputs.duplicate = True
   client.parameters.outputs.num_outputs = 0

   ################## Solution #########################


   solver = Solver(client)
   result = solver.solve(model)
   for solution in result:
       print(f"values = {list(solution.values.values())} energy = {soluti
   y}")

   best_solution = Best_solution(result)
   print('\n\nInput numbers:',N)
   Subset_result(best_solution,N)
```

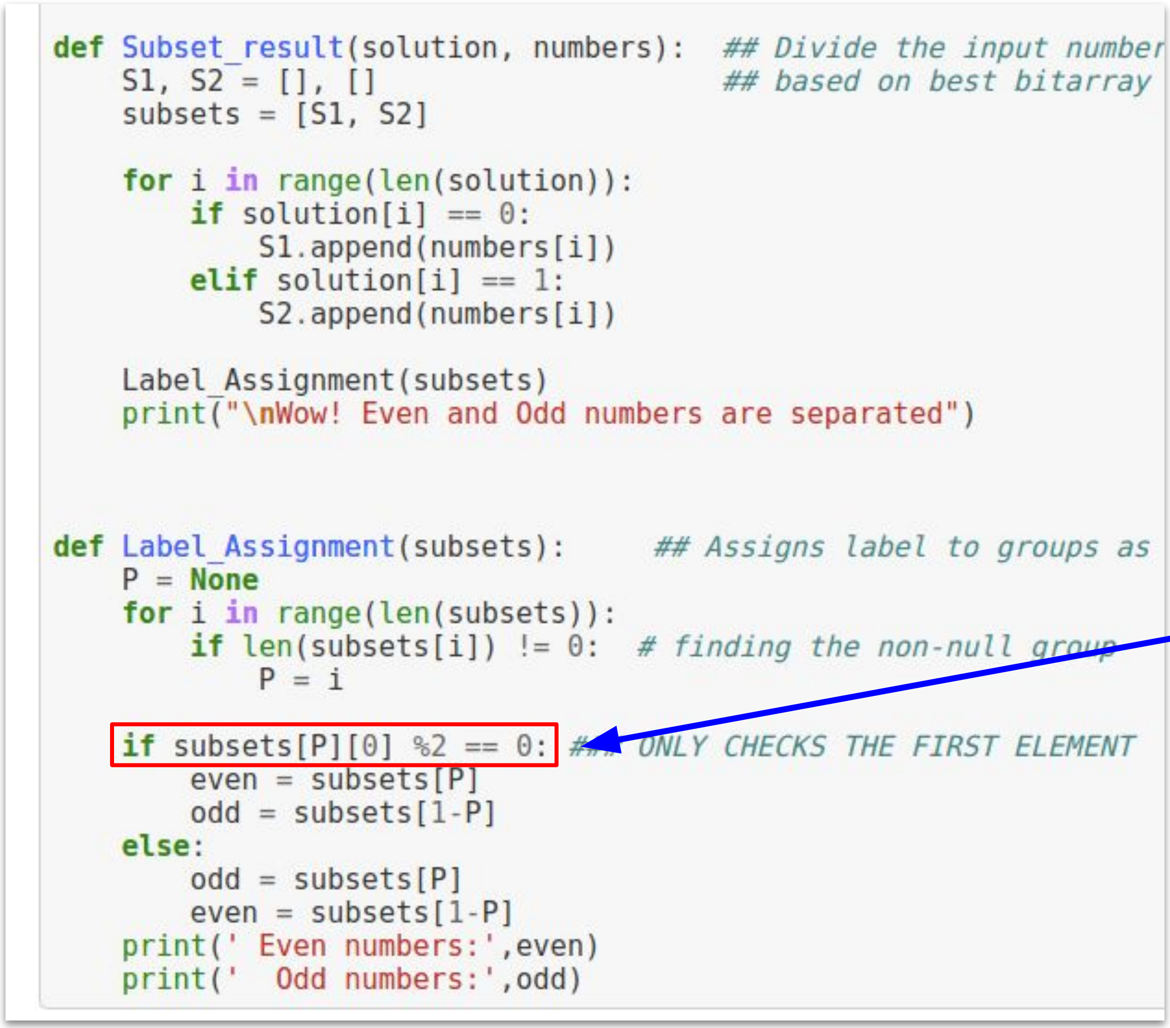


Solving QUBO with D'wave clients ¶

D-wave provides quantum annealing and hybrid solvers.

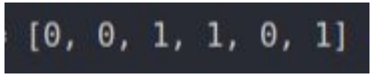
```
]: ############# Solving with Dwave client  ##################



   client = DWaveSamplerClient()
   client.token = "DEV-d1f6e1d1c38fbe9219ce7713e110736473b657a0"
   client.solver = "Advantage_system1.1"
   client.parameters.num_reads = 1000


   ############   Solution   ######################
   solver = Solver(client)
   result = solver.solve(model)
   for solution in result:
       print(f"nvalues = {list(solution.values.values())} energy = {so
   y}")

   best_solution = Best_solution(result)
   print('\n\nInput numbers:',N)
   Subset_result(best_solution,N)
```

- Here we have solved “BinaryQuadraticModel” with Amplify client and Fixstars D-wave(wrapper) client.



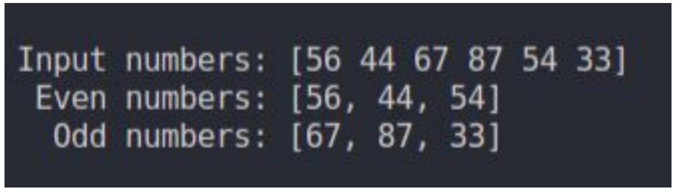
```python
def Subset_result(solution, numbers):   ## Divide the input number
    S1, S2 = [], []                     ## based on best bitarray
    subsets = [S1, S2]

    for i in range(len(solution)):
        if solution[i] == 0:
            S1.append(numbers[i])
        elif solution[i] == 1:
            S2.append(numbers[i])

    Label_Assignment(subsets)
    print("\nWow! Even and Odd numbers are separated")



def Label_Assignment(subsets):      ## Assigns label to groups as
    P = None
    for i in range(len(subsets)):
        if len(subsets[i]) != 0:   # finding the non-null group
            P = i

    if subsets[P][0] %2 == 0:   ## ONLY CHECKS THE FIRST ELEMENT
        even = subsets[P]
        odd = subsets[1-P]
    else:
        odd = subsets[P]
        even = subsets[1-P]
    print(' Even numbers:',even)
    print('  Odd numbers:',odd)
```

- Solver produces best binary array.
- **Subset_result** creates two subsets based on 1's and 0's.



```
[0, 0, 1, 1, 0, 1]
```

- 2 subsets are now separated, but we do not know which one is even, which one is odd.
- **Label_Assignment** assigns label to individual subset.
- This **Label_Assignment** function **only checks the first element** of a non-null subset with classical even number checking protocol.
- Assignment is done with testing a single sample from a non-null subset.



```
Input numbers: [56 44 67 87 54 33]
 Even numbers: [56, 44, 54]
  Odd numbers: [67, 87, 33]
```

Thank You