演算法Homework#4

山下夏輝（Yamashita Natsuki）

R08922160

Reference: 陳建宏（d08944003），吳冠霖(r08922115)，TA, Slides of the class（191226\_Approximation (2).pdf，191212\_NP-Completeness-1.pdf，191212\_NP-Completeness-2.pdf）

**E.**

Reference: <https://www.cc.gatech.edu/~rpeng/CS3510_F17/Notes/Nov20Approx.pdf>

**1-1.**

Total maximum power is 18.

The linking method is below.

1. Create 2 sets: A{} and B{}
2. Put ½ numbers vertex into A{} and rest vertex into B{}
3. Calculate Total\_fitness
   1. Pseudo code  
      for i =0, i = N/2, ++  
       for j =0, j = N/2, ++  
       temp\_Total\_fitness += the value of fitness between A(i) and B(j)

If temp\_Total\_fitness > Total\_fitness

Total\_fitness = temp\_Total\_fitness

1. Run this process over every combinations of vertex in each sets
2. Return 2\*Total\_fitness

**1-2.**

Total Maximum power is 19.

The linking method is below.

1. Sort components of M by the fitness value from bigger to smaller.
2. Assign an attribute on each vertex u and v from the most biggest fitness edge.
   1. Check the attribute.
      1. If both of the attribute of u and v has not been assigned yet,  
         assign “light” on u and “dark” on v.
      2. If one of the attribute of u or v has not been assigned yet,   
         assign the opposite attribute on the vertex whose attribute has not been assigned yet.
3. Create a Set{}
4. Calculate Total\_fitness.
   1. Check the attribute.
      1. If u and v has different attributes, add the value of fitness into Total\_fitness.  
         And put the edge into Set{}
      2. If not, pass this calculation.  
         And put the edge into Set{}
5. Continue this process until every edges in Set{} recursively

**2-1.**

When given a graph 𝐺 and an integer 𝑘 for this problem, it is verified in polynomial time that there a cut in 𝐺 containing at least 𝑘 edges or not. So this problem is in NP. Such as the calculation of the total fitness and comparison of k and total fitness can be done in polynomial time.

MAXIMUM\_CUT = {<G, k>: G is a graph containing an maximum number of cut k}

This problem = {<G, k>: G is a graph containing an maximum cut of maximum value k}

Maximum cut ≤p This problem

At first, we assume the graph G is the one of this probrem. Then we create new graph G’ from it.

1. Add edges between every pair of vertex, when the weight on the edge is more than 1. (If fitness of w1 and w2 is 5, there are 5 edges between w1 and w2, which means that vertex has multiple edges.
2. Run Maximum cut algorithm

Here, this problem is reduced from maximum cut. Therefore, this problem is NP-Hard.

**2-2.**

Reference: <https://www.cc.gatech.edu/~rpeng/CS3510_F17/Notes/Nov20Approx.pdf>

We assume all of the weights on each edges(=fitness) are 1. W1 only has a half of verticesand W2 has the rest of vertices. This ensures that there are the half of the edge between W1 and W2. Then,

This.Total\_fitness ≥ Sum of fitness/2

If maxmum cut can cut every edges,

OPT = Sum of fitness

Then,

OPT / This.Total\_fitness ≤ Sum of fitness / Sum of fitness / 2 = 2

Therefore, it is a 2-approximation.

**3.**

When given a graph 𝐺 and an integer 𝑘 for this problem, it is verified in polynomial time that there a cut in 𝐺 containing at least 𝑘 edges or not. So this problem is in NP. Such as the calculation of the total fitness and comparison of k and total fitness can be done in polynomial time.

Sub problem 3 = {<G, k>: G is a graph containing an maximum cut of maximum value k}

Sub problem 2-1 ≤p This problem

Only the pre-processing below is needed.

1. Every vertex is colored as light or dark. If the neighbor vertex is the same color, the edges should be removed.
2. Run Maximum cut algorithm
3. Run 1. and 2. on every combinations of the colored vertex graph.
4. return maximum one in 3..

**4.**

This problem is the component of Hamiltonian Path. Because, “there exists at least one link to w” means that every vertex is connected and each vertex has at least one vertex which has the non-same color adjacent. In addition to it, we should find the other fitness which is not in Hamiltonian Path(=2nd link, 3rd link,,, nth link) and add them to total fitness. Then, it is more difficult than Hamiltonian Path apparently.

Here, Hamiltonian Path is NP-complete. Therefore, the component of this problem is more than NP-complete or NP-hard.

**5.**

Reference：<https://www.geeksforgeeks.org/subset-sum-problem-dp-25/>

Subset Sum Problem can be reduced in polynomial time with DP. The matrix of sum of t /2 and N can be used for determining whether there is a subset whose sum equals to sum of t /2.

1. Sort N from lower number to larger number
2. Create the matrix[i][j] (N \* sum of t / 2) of boolean
3. For i = 0 to n, For j = 0, to sum of t / 2
4. Record TRUE or FALSE in every cells with DP
   1. If Ni ≤ j, matrix[i+1][j] is matrix[i][j] or matrix[i][j-matrix[i]]
   2. If Ni > j, matrix[i+1][j] is matrix[i][j]
5. Return matrix[i+1][j]

Then, this problem can be solved in polynomial time.

**F**

**1-1.**

[INDEXING]

Si is the i-th set selected by GREEDYSETCOVER.

Fi is the set of S when Si is put into it.

Ci is the set of elements when elements of Fi-1 is inserted into C.

When an element xk is about to be put into C, the price of xk is 1/|Fi-Ci| and also there are at least n-k+1 uncovered numbers, so that the price of xk is 1/|Fi-Ci|≤ OPT/n-xk+1.

**1-2.**

Reference: <https://en.wikipedia.org/wiki/Set_cover_problem>

According to 1-1., the price of xk can be indicate as = 1/|Fi-Ci|, and also at most OPT/n-xk+1. Then, the sum of price of all x is |I| through this algorithm.

|I| = Σ1/n-xk+1 \* OPT

＝ (1/n + 1/n-1 + 1/n-2 … + 1) \* OPT //Harmonic number

= O(ln n + 1) \* OPT

Therefore, this is (ln n+O(1))-approximate solution.

**2-1.**

This problem ≤p cover set problem

Goal： collect pairs of sets Mi + Mj (i≠j)

1. Make all pairs of sets of M which can be merged
2. Collect pairs of sets and cover all strings

Ex: Let M1 is abbb and M2 is bcc. As they are merged, abbbcc and abbbbc comes up as the pairs. M1 and M2 is in C and it covers a, b and c when the pairs of set are collected.

1. Count costs

Ex: The cost is 6 when abbbcc is collected in C.

The cost is 7 when abbbbcc is collected in C.

Briefly saying, it becomes cover set problem that 2 sets is collected to C.

**2-2.**

Using GREEDYSETCOVER, it can be (2 ln n + O(1))-approximate solution. Because every time we collect 2 sets and redundant strings would be collected into C. Here, at first we assume that 2 sets are collected every time but they are not merged.

According to 1-2: |I| = Σ1/n-xk+1 \* OPT

Then this problem: |I| = 2(Σ1/n-xk+1) \* OPT

= O(2(ln n) + 2(1)) \* OPT

= O(2 ln n + 1) \* OPT

In addition to it, from the statement of this problem, len(s)+len(t) > len(y). So we consider the influence of merging pair of strings. Then,

|I| < O(2(ln n) + 1) \* OPT

So it is (2 ln n + O(1))-approximate solution.

(appendix): if the merged strings have more than 2 sets of set in themselves, apparently the solution is better than my explanation above. Therefore, in this case, |I| < O(2(ln n) + 1) \* OPT as well.