CS 530 Advance Algorithm - Fall 2019
Homework #3

Ziqi Tan U88387934
ziqi1756@bu.edu

October 29, 2019

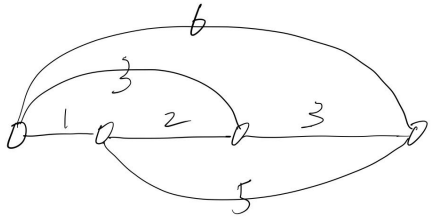
Due: Tuesday, October 29 - to be submitted via Gradescope
Reading : Chapter 35, Section 5, pages 1128-1133.
Review sections 1, 2 and 3 of chapter 34, pages 1047-1077

Problem 1

(i) Give an example of a complete Euclidean graph G with 4 vertices, where all the weights are integers and no two weights are equal.

Example:

Four vertices are all in a line.



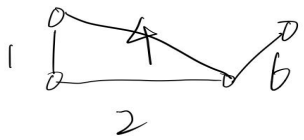
Brief explanation: A **Euclidean graph** is a graph in which the vertices represent points in the plane, and the edges are assigned lengths equal to the Euclidean distance between those points. — Wikipedia

Another example: There exists some Euclidean Graphs where the vertices are not all in one line and the weights are integers.
Just run the code on computer. If you are interested in the code, I post it on my GitHub.

sadf

(ii) Give an example of a graph G which is not a complete graph and which cannot be extended to a complete Euclidean graph. Your graph should have 3 or more vertices. Try to find a graph with as few vertices as possible.

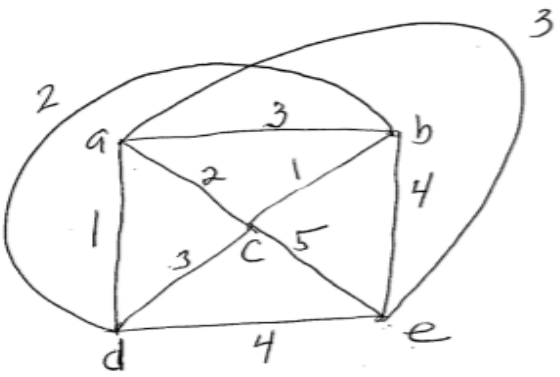
Example:



Brief explanation: The edges with weight 1, 2 and 4 respectively have made this graph a non-Euclidean graph. No matter how we extend the given graph of the example, it will still be a non-Euclidean graph.

Problem 2

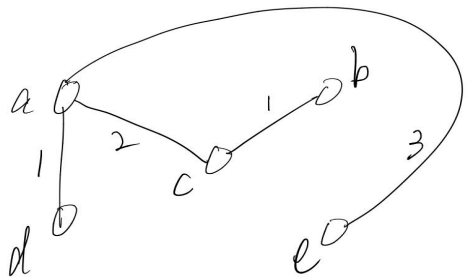
Consider the following Euclidean graph G.



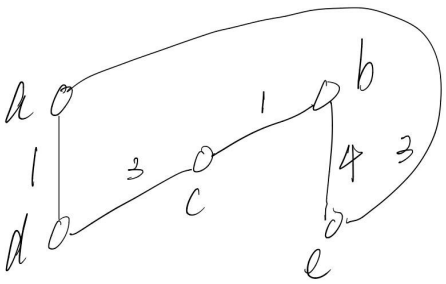
(i) Carry out the 2-approximation algorithm for G (the one given in section 35.2.1 of the textbook). Draw the MWST T* you get for G, state T*'s weight, and state the path you compute which goes twice around T*. Also draw the TSP cycle you obtain from T* and state the weight of the TSP cycle.

Answer:

1. Using Prim Algorithm in section 23.2 to find the minimum weight spanning tree T* for G.



2. A walk of T*, starting at vertex a.
A full walk of the tree visits the vertices in the order: a, d, a, c, b, c, a, e, a.
Cost is $(1 + 2 + 1 + 3) \times 2 = 14$.
3. Delete the repeated vertices and yield a new tour: a, d, c, b, e, a.



Its total cost is 1+3+1+4+3=12.

(ii) Is the weight of the TSP cycle you obtain for G optimal for G? Is it unique, that is, are there other TSP cycles you could have chosen using T* which give different weights than the one you chose? Explain.

Answer:
The TSP cycle obtained in (i) for G is optimal for G.

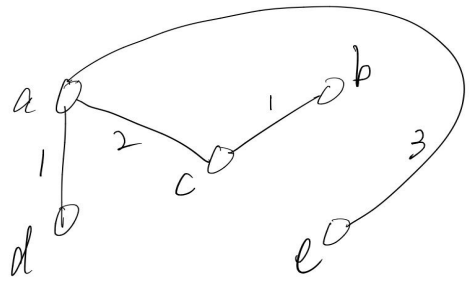
The weight of the TSP cycle obtained from the given T* in (i) is unique. They will get the same cost — 12. List all the possible TSP cycles. Starting at vertex b, the TSP cycle will be b, c, a, d, e, b. It still costs 12.

Problem 3

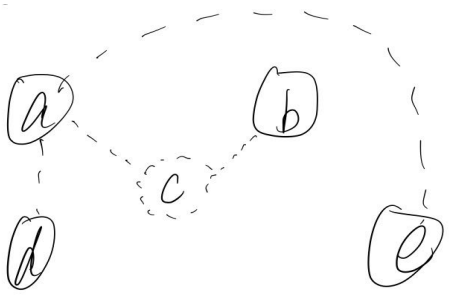
Consider the same graph G as in problem 2, but now, using the same MWST you found there carry out Christofides 1.5 approximation algorithm on G. You can use the TSP you found in problem 2, part (i). Again state the TSP cycle you obtain and state its weight. But here also show the least cost matching you found, and what Euler path you found that contained this matching.

Answer:

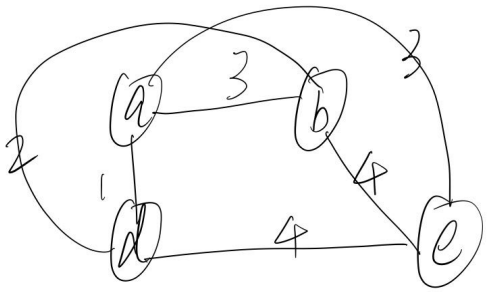
1. Calculate minimum spanning tree T*.



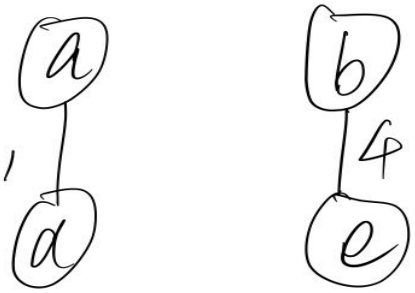
2. Calculate the set of vertices O with odd degree in T*.
   Vertices a, d, b and e have odd degree.



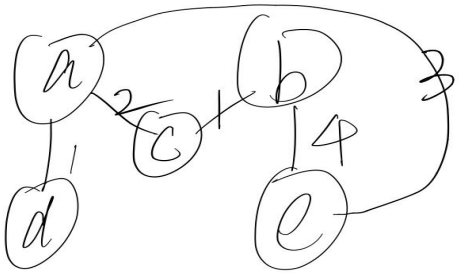
3. Form the subgraph of G using only the vertices of O.



4. Construct a minimum-weight perfect matching M in this subgraph.

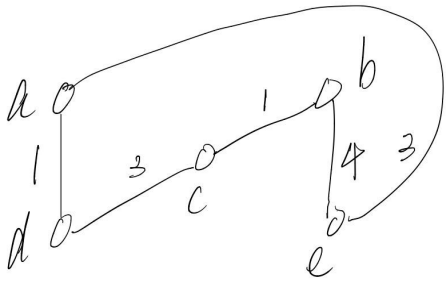


5. Unite matching and spanning tree T ∪ M to form an Eulerian multigraph.



6. Calculate Euler tour: d, a, e, b, c, a, d.

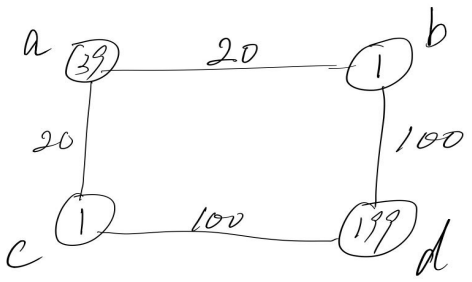
7. Remove repeated vertices, giving the TSP cycle: d, a, e, b, c, d.
   The cost is $1 + 3 + 4 + 1 + 3 = 12$.



Problem 4

The Weighted Vertex Cover Problem is, you are given an undirected graph G with vertex weights wi. The goal is to find a minimum-weight vertex cover of S. (Here the weight of the VC is the sum of the weights of the vertices in S.)

Example:



(i) Assume that you want to solve the minimum weighted vertex cover problem with the same algorithm (given on page 1109 of the book) for the unweighted vertex cover problem except that now you order the edges you consider from smallest average weight to largest and consider them in that order. Here the average weight of an edge is the average of the weights of the 2 vertices of that edge. In this case we still get a vertex cover for the graph but that this vertex cover can be a very bad approximation.

Show this. That is find a vertex-weighted connected graph with at least 4 vertices where the VC that you get from this algorithm is not even a 20(twenty)-approximation of the optimal weighted vertex cover.

Answer:
Optimal cover $C^* = \{b, c\}$ where the weight is $2 = 1 + 1$.
Threrefore, $OPT = 2$.

Here is the greedy approximation algorithm:
Let C be the cover set.
Initially, $C = \varnothing$.
Choose $edge(a, b)$ with the smallest average weight $20 = (39 + 1)/2$.
Then add a, b into the cover set C.
$C = \{a, b\}$ and remove all edges incident on either a or b.
Choose edge(c, d) which is the only edge left in the graph.
$C = \{a, b, c, d\}$ will be the final output. The weight is 1+1+39+199 = 240.
Therefore, $Greedy > 20 * OPT$ where $Greedy = 240$ and $OPT = 2$ in this case.

(ii) Do the same things for this problem but with the "greedy" algorithm where you order the vertices from smallest to largest weights and choose the vertices in that order to put into your vertex cover C that is constructed. As usual, once a vertex enters C you remove it and all its incident edges from the graph.

Answer:
Order the vertices: b, c, a, d.
Choose b and remove all its incident edges.
Choose c and remove all its incident edges.
The output is $\{b, c\}$, the weight of which is $2 = 1 + 1$.

Problem 5

(i) Problem 35.4-3 on page 1127

In the MAX-CUT problem, we are given an unweighted undirected graph $G = (V, E)$. We define a cut $(S, V - S)$ as in Chapter 23 and the weight of a cut as the number of edges crossing the cut. The goal is to find a cut of maximum weight. Suppose that for each vertex , we randomly and independently place in S with probability $1/2$ and in $V - S$ with probability $1/2$. Show that this algorithm is a randomized 2-approximation algorithm.

Note: you can find the definition and some discussion about cuts in graphs on Page 626 of the textbook (and elsewhere). This problem requires a randomized algorithm and a randomized 2-approximation. Your job is to show that the expected value of the cut you obtain is always at least $1/2$ as big as the size of the optimal maxcut.

Proof:

Consider the expected value of the size of the cut.

For each edge, the probability that this edge is in the cut is $1/2$.

Explain:

Consider a graph with vertex $a(v_a)$ and $b(v_b)$ with an edge e_1.

$$P(e_1 \in Cut) = P(v_a \in S \cap v_b \in V - S) + P(v_b \in S \cap v_a \in V - S)$$
$$= \frac{1}{2} \times \frac{1}{2} + \frac{1}{2} \times \frac{1}{2} = \frac{1}{2}$$

Assume a complete graph has n edges.

The probability that the size of the cut $P(Size)$ can be written as follows:

$$P(Size = k) = C_n^k (\frac{1}{2})^k (\frac{1}{2})^{n-k} = C_n^k (\frac{1}{2})^n \tag{1}$$

where $C_n^k = \frac{n!}{k!(n-k)!}$.

Let E be the expected value of the size of the cut.

$$E(Size) = \sum_{k=0}^{n} kP(Size = k) = \sum_{k=0}^{n} kC_n^k (\frac{1}{2})^n \tag{2}$$

All we need to do is to **prove** $E(Size) = \frac{n}{2}$.

Proof:

$$\sum_{k=0}^{n} kC_n^k (\frac{1}{2})^n = \frac{n}{2} \tag{3}$$

Equation 3 is what we need to prove and it also can be written as:

$$\sum_{k=0}^{n} kC_n^k = n2^{n-1} \tag{4}$$

According to Binomial theorem, we have:

$$(1 + x)^n = \sum_{k=0}^{n} C_n^k x^k \tag{5}$$

Calculate the derivation of x for equation 5:

$$n(1+x)^{n-1} = \sum_{k=0}^{n} kC_n^k x^{k-1} \tag{6}$$

When $x = 1$, we will get

$$n2^{n-1} = \sum_{k=0}^{n} kC_n^k \tag{7}$$

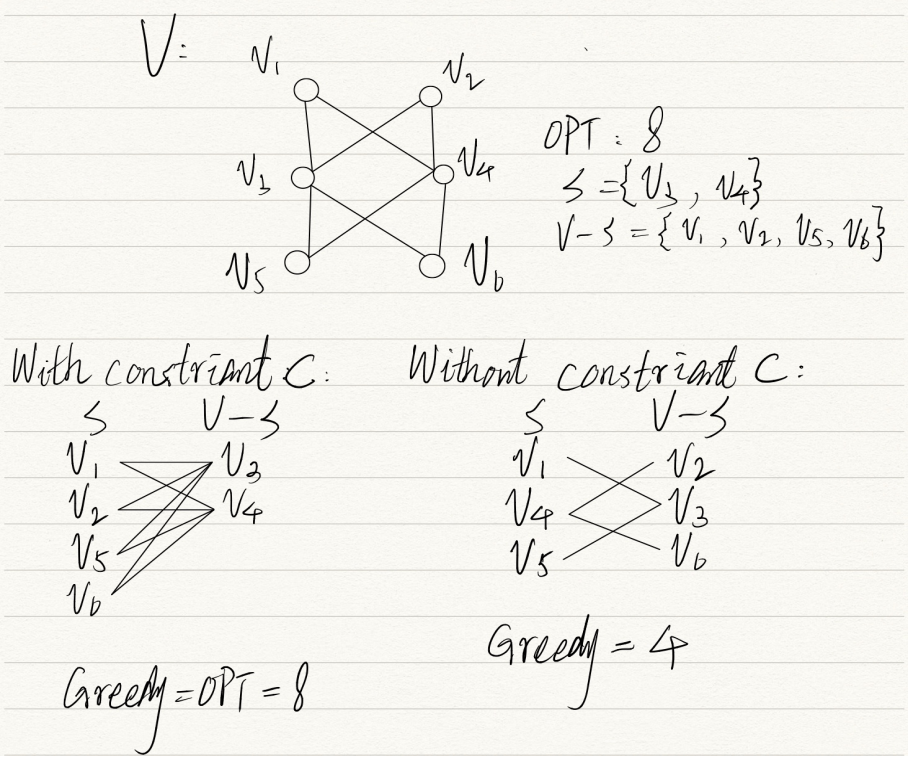
which exactly is the equation we want to prove.

The expected value is exactly the half of the number of the edges.
Therefore, this algorithm is a randomized 2-approximation algorithm.

(ii) Now try the same problem but with no randomization, just use a greedy approach. At each step i, add vertex v_i to one side of the cut or the other by putting v_i on the side which increases the cut size the most.

(a) Give an example of a graph G with at least 5 edges where this algorithm gives a cut size exactly 1/2 of the maximum cut size.

Answer: If I give one more constraint C on this algorithm, we cannot find such a graph where this algorithm gives 1/2 of the max cut. The C is that when vertex v_i increases the cut size no matter what side does it take, we put all these vertice on one side. If we replace this restriction by randomly taking a side, there is a graph where this algorithm gives a cut size exactly 1/2 of the maximum cut size.
Example: (refering to Jiaqian Sun)



Greedy without constriant is 4 that is 1/2 of the maximun cut size 8.

(b) Can you find a graph H with largest cut size at least 4 where you end up with a cut whose size is less than 1/2 of the maximum cut size ? Explain.

Answer:

You cannot find that graph, because this greedy algorithm is a 2-approximation algorithm: $Greedy(I) \geq \frac{1}{2}OPT$.

Proof: Let vertex set S be one side of the cut and $V - S$ be the other side. At each step i, consider where to put vertex v_i. Let E_i be the number of edges incident to current S and $V - S$, E_i^S be the number of edges incident to current S ,and E_i^{V-S} be the number of edges incident to current $V - S$.

$$E_i^s + E_i^{V-S} = E_i \tag{8}$$

If $E_i^s \geq E_i^{V-S}$, we put v_i into $V - S$. Otherwise, put it into S.

$$E_i^s \geq E_i^{V-S} \tag{9}$$

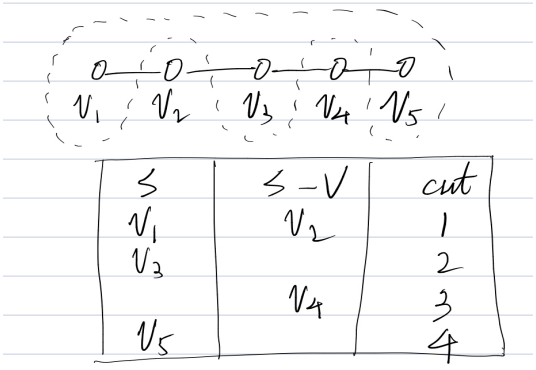
From equation (8) and (9), we can draw that

$$E_i^S \geq \frac{E_i}{2} \tag{10}$$

Therefore, assuming the OPT is the number of all edges in graph H,
$Greedy(I) = \sum_{i=1}^{n} E_i^{Side} \geq \frac{1}{2}OPT = \frac{1}{2}\sum_{i=1}^{n} E_i$ where $Side$ is S or $V - S$.

(c) Can you find a graph F with largest cut size at least 4 where you end up with a cut whose size is equal to the optimal (that is largest) cut size ? Explain.

Example:



The max cut is 4 and the greedy cut is also 4.

Problem 6

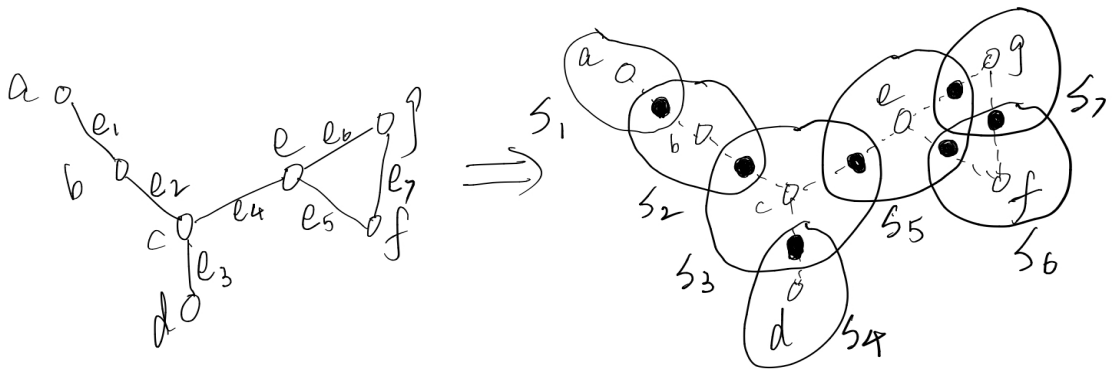
Give a reduction from the Vertex Cover decision problem to the Set Cover decision problem. Specifically, given a VC problem instance you need to convert it into a set cover problem instance. Then show how you can use that SC instance to solve the original VC problem.

Answer:

1. Vertex cover (VC) problem:
   Given a graph $G = (V, E)$, we need to find the minimum size of a vertex set C to cover/connect all the edges in G.

2. Set cover (SC) problem:
   Given a set U and a list of subsets S_i, the union of which are U, we need to find the mininmum size of a subset set to cover U.

3. Reduce VC problem to SC problem:
   Firstly, the edges in G in VC problem will become the points/elements in SC problem. Secondly, the edges incident to a vertex in G will become a subset in SC problem. Thus, each subset in SC problem represent a vertex in VC problem.
   Example:



In the vertex cover problem:

$$G = (V, E)$$
$$V = \{a, b, c, d, e, f, g\}$$
$$E = \{e_1, e_2, e_3, e_4, e_5, e_6, e_7\}$$

The cover is $C = \{b, c, e, f\}$ and the size of this vertex problem is $|C| = 4$.

In the set cover problem:

$$U = \{e_1, e_2, e_3, e_4, e_5, e_6, e_7\} = E$$

Subsets:

$$S_1 = \{e_1\}, S_2 = \{e_1, e_2\}$$
$$S_3 = \{e_2, e_3, e_4\}, S_4 = \{e_3\}$$
$$S_5 = \{e_4, e_5, e_6\}, S_6 = \{e_5, e_7\}$$
$$S_7 = \{e_6, e_7\}$$

The cover is $C = \{S_2, S_3, S_5, S_6\}$ and the size of this set cover problem is $|C| = 4$, the same as the result of vertex cover.
Therefore, vertex cover problem can be reduced to set cover problem.