組別: 姓名: 學號: 分數: 批改者:

一人寫一份，可討論。 上台講解且正確者，全組自行加一分。

Algorithms

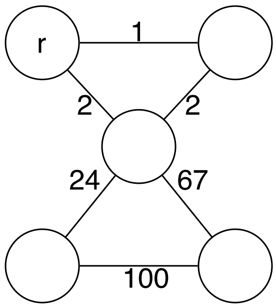
Classwork # Video 6.1-6.4

Each problem is 3 points.

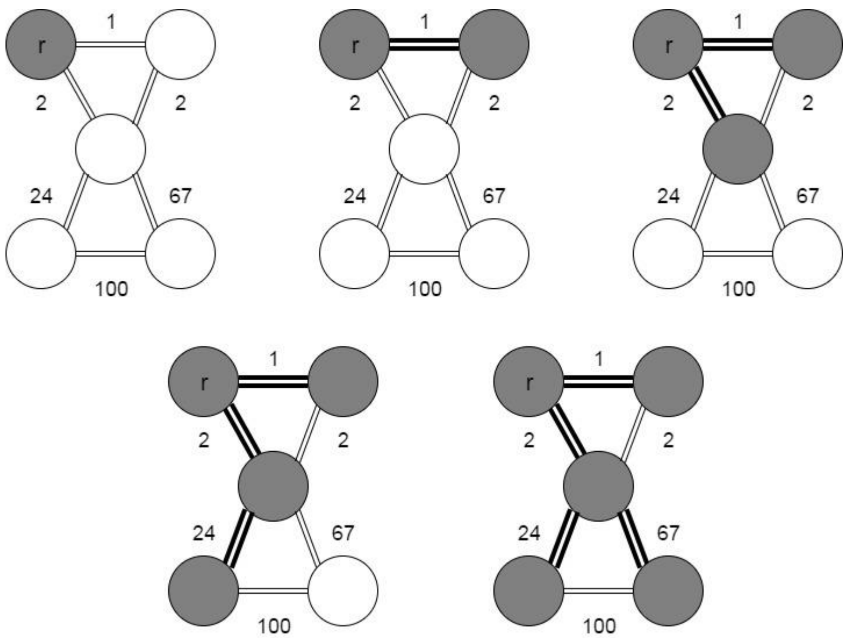
3 = correct. 2 = one mistake. 1 = many mistakes. 0 = no answer.

1. Prim's Algorithm [6.2]

Please use Prim's algorithm to build minimum spanning tree for the undirected graph shown below. In your answer please show the procedures step-by-step for each iteration (adding one safe edge). Please start from the root vertex marked with *'r'* in the following graph.



Sol:



2. **Bottleneck spanning tree [6.1]**

   A bottleneck spanning tree *T* of an undirected graph *G* is a spanning tree of *G* whose largest edge weight is minimum over all spanning trees of *G*. We say that the value of the bottleneck spanning tree is the weight of the maximum-weight edge in *T*. Explain why a minimum spanning tree is a bottleneck spanning tree.

Sol: (請大家練習好好寫證明。一步一步推演，不要含糊籠統)

1. Assume a minimum spanning tree T_s which isn't a bottleneck spanning tree T_b.
2. By definition, the maximum weight w of edge $e=(u,v)$ in T_s must be larger than any edge weight in T_b.
3. We can replace edge e by an edge in T_b and remain the connectivity.
4. Because the T'_s has $n-1$ edges and is a connective graph, it forms another spanning tree whose total edge weight of is smaller than original T_s.
5. Thus, T_s is not a minimum spanning tree. The original assumption is wrong. Hence, a minimum spanning tree T_s must be a bottleneck spanning tree.

1. **Bottleneck spanning tree(cont'd) [6.1]**

   Give a linear-time algorithm that given a graph *G* and an integer *b*, determines whether the value of the bottleneck spanning tree is at most *b*. A clear description will suffice. Pseudo-code is not required.

Sol:

To determine whether the value of the bottleneck spanning tree is at most *b*. We can apply DFS once on a random vertex of *G* ignoring edge weight is bigger than *b* to see if every vertex of *G* is visited. If is, the value of the bottleneck spanning tree is at most *b*.

2. **Enemy relationship [6.3]**

   Romeo and Juliet have finally decided to get married. But preparing the wedding party will not be so easy, as it is well-known that their respective families are bloody enemies. In this problem, you will have to decide which person to invite and which person not to invite.

   We have a list of *N* people who can be invited to the party or not. We have an edge between (u, v), if two people (u, v) are enemies. Now, we already know each pair of enemies. The "enemy" relationship has the following

properties:
(a) If a is an enemy of b, and b is an enemy of c, then a is a friend of c.
(b) The enemies of the friends of a are his enemies.
(c) If a is an enemy of b, then b is an enemy of a.
One person will accept an invitation to the party if and only if **none of his enemies** is invited. You have to find the **maximum number of people** that can be invited, so that all of them accept their invitation.
Please complete the pseudocode below to solve this problem. A set of (u) contains all friends of u. You can use all the disjoint set functions (Ex: *UNION, FIND-SET, MAKE-SET*) mentioned in the class.

MAX_INVITATION()
Let Enemy be a new array
For i = 1 to N
    MAKE-SET(i)
    Enemy [i] = Φ
Foreach edge (u,v)
    SET_ENEMY (u,v)
S_{max} = Φ
For i = 1 to N //traverse all sets
    if i != FIND-SET(i)
        continue
    j = FIND-SET(Enemy [i])
    if $\|S_i\| > \|S_j\|$ or $(\|S_i\| == \|S_j\|$ & $i < j)$
        S_{max} = UNION(S_{max}, S_i)
return S_{max}
	SET_ENEMY(u, v)
	if(FIND_SET(u) == FIND_SET(v))
	    //Enemy in same group
	    report "conflict"
	else
	    //TODO, fill your code here

Sol:

MAX_INVITATION()
Let Enemy be a new array
For i = 1 to N
    MAKE-SET(i)
    Enemy [i] = Φ
Foreach edge (u,v)
    SET_ENEMY (u,v)
S_{max} = Φ
	SET_ENEMY(u, v)
	if(FIND_SET(u) == FIND_SET(v))
	    //Enemy in same group
	    report "conflict"
	else
	    UNION(Enemy [u], v)
	    UNION(Enemy [v], u)
	    Enemy[u] = v

For $i = 1$ to N //traverse all sets
    if i != FIND-SET(i)
        continue
    j = FIND-SET(Enemy [i])
    if $\|S_i\| > \|S_j\|$ or ($\|S_i\| == \|S_j\|$ & $i < j$)
        S_{max} = UNION(S_{max}, S_i)
return S_{max}
	Enemy[v] = u

3. The Kruskal's Algorithm [6.4]

Suppose that all edge weights in a graph are integers in the range from 1 to $|V|$ and $|V| \ll |E|$. How fast can you make Kruskal's algorithm run? Hint: some faster sorting algorithm could be used.

Sol:

If the edge weights were integers in the range from 1 to $|V|$, then we could use counting sort to sort the edges more quickly. This time, sorting would take $O(E+|V|)=O(E)$ time. Thus, we get a total running time of $O\big(E\alpha(V)\big)$.