Homework 9

2019.05.02

**1.**

**Exercises 23.1-11 Given a graph G and a minimum spanning tree T, suppose that we decrease the weight of one of the edges not in T. Give an algorithm for finding the minimum spanning tree in the modified graph.**

想法：某一條不在MST的邊的weight變小了，唯一可能加入新的MST只會是weight變小的那條邊所以加進去看看。

加進去後會形成cycle，把cycle裡最大weight的邊刪掉即是新的MST。

Pseudocode：

New\_MST(G){

add edge in the minimum spanning tree (被減的edge)

for(all edge in cycle){

if(weight>max)

max=weight ;

temp\_edge=edge ;

}

Delete temp\_edge from minimum spanning tree

}

**2.**

**Exercises 23.2-5 Suppose that all edge weights in a graph are integers in the range from 1 to |V|. How fast can you make Prim's algorithm run? What if the edge weights are integers in the range from 1 to W for some constant W ?**

1. From 1 to W

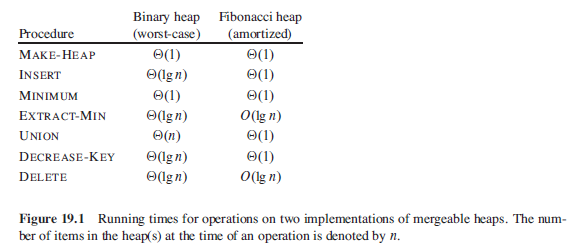
用一個 array 來取代 queue，其大小為 0~W+1。而每一欄中放 weight與欄位編號相同的 vertex。

如此一來 Extract-Min 只需要找到第一個不為空的欄位，所以花的時間為 O(1)。而 Decrease-Key 的話最糟需要 O(E) 時間，因為有可能會發生需要找的 vertex 在串列的最後面。

綜合以上需要看過每一個 edge，所以 total running time = O(E)。

1. From 1 to |V|

如果用上一部分的資料結構去實作，每次 Extract-Min 需要花 𝚯(V)，所以總共需 𝚯(V2)。因此總時間拖慢成 𝚯(V2+E) = 𝚯(V2)。

然而，如果運用不同的資料結構，如 Fibonacci-heap 則可以達到 𝚯(E+VlgV) 的總運行間

**3.**

**Exercises 23.2-7 Suppose that a graph G has a minimum spanning tree already computed. How quickly can we update the minimum spanning tree if we add a new vertex and incident edges to G ?**

**Solution:**把加入vertex的graph再做一次minimum spanning tree.

**分析:** Before: vertex=V and Edge=V-1

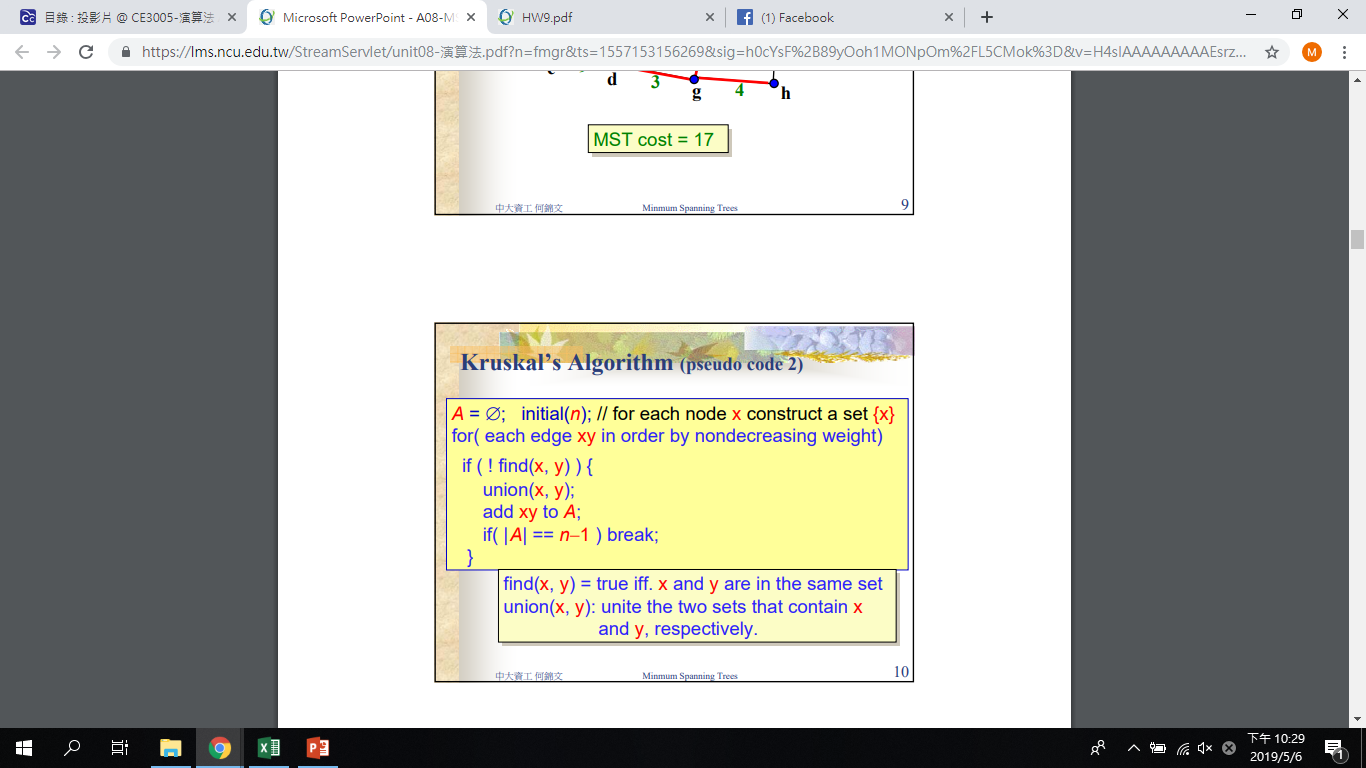
After: vertex=V+1 and Edge=2V-1

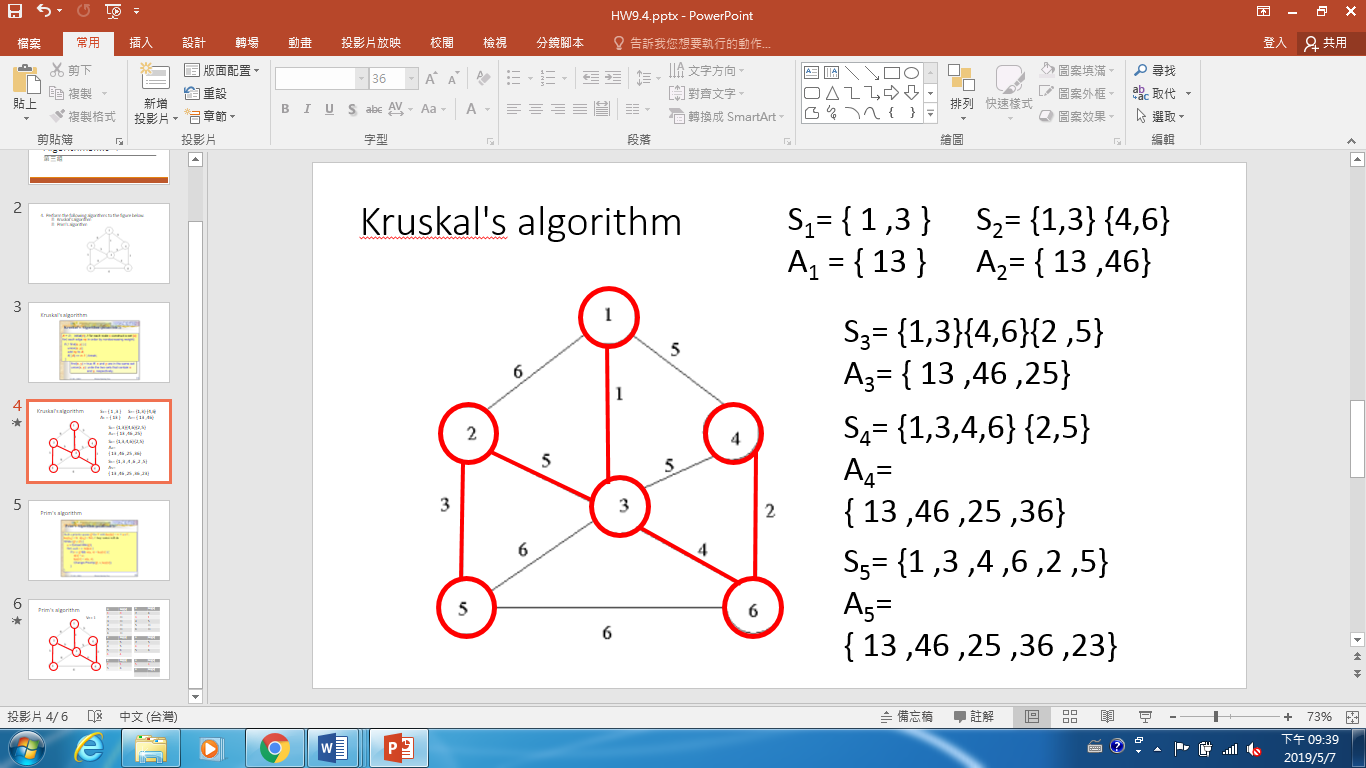
Time complexity=O(3VlgV)=O(VlgV).

**4.**

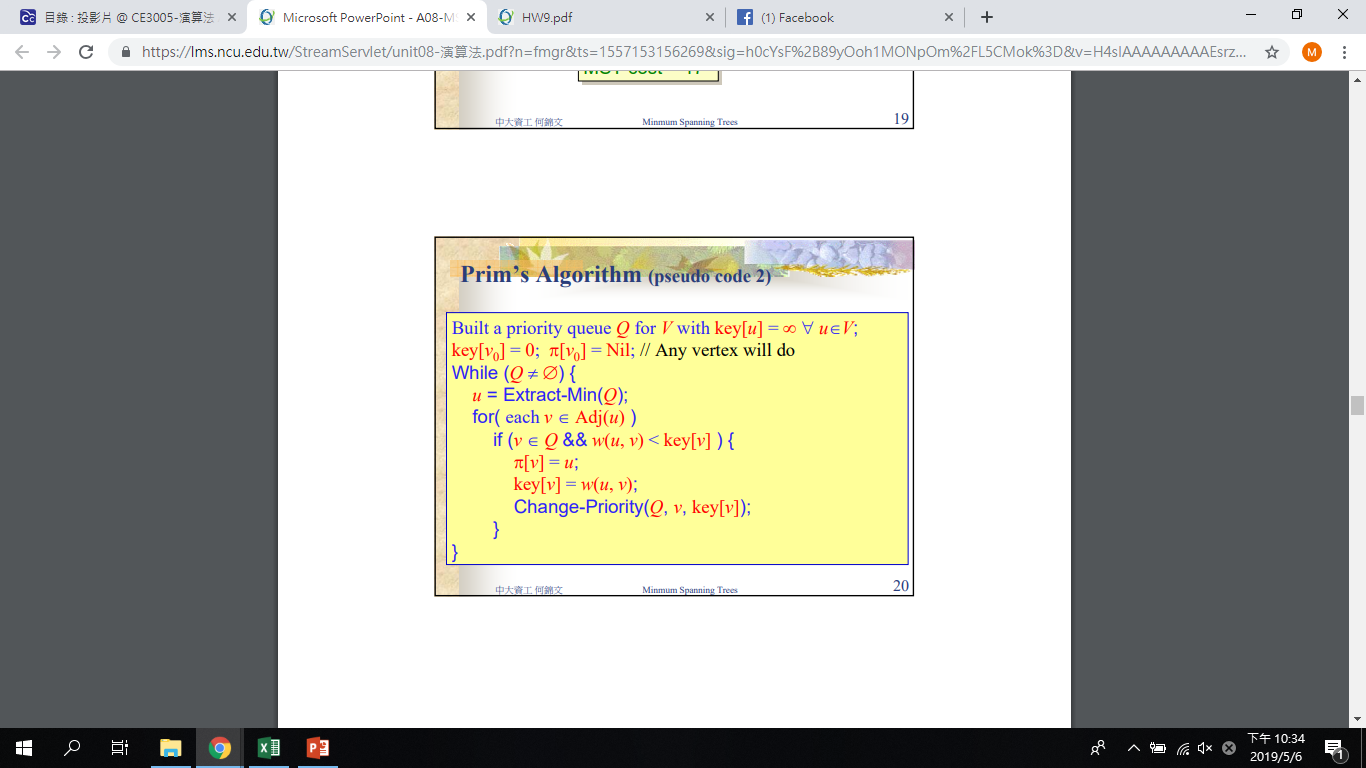
**EXT. 8-1 Perform the following algorithms to the figure below.**

** Kruskal's algorithm (Please list each step according to PPT unit8 p.10)**

****



** Prim's algorithm (Please list each step according to PPT unit8 p.20)**

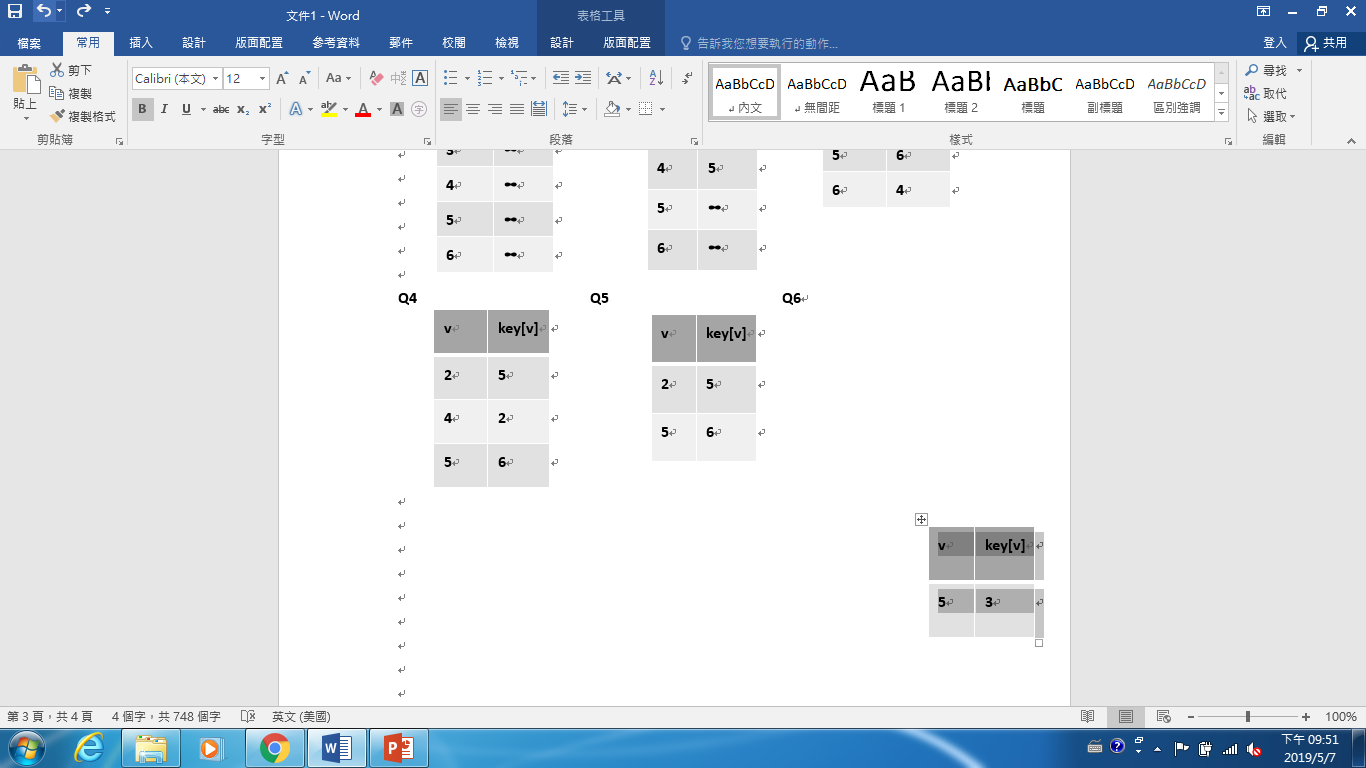


|  |  |
| --- | --- |
| **v** | **key[v]** |
| **1** | **0** |
| **2** | **∞** |
| **3** | **∞** |
| **4** | **∞** |
| **5** | **∞** |
| **6** | **∞** |

|  |  |
| --- | --- |
| **v** | **key[v]** |
| **2** | **6** |
| **3** | **1** |
| **4** | **5** |
| **5** | **∞** |
| **6** | **∞** |

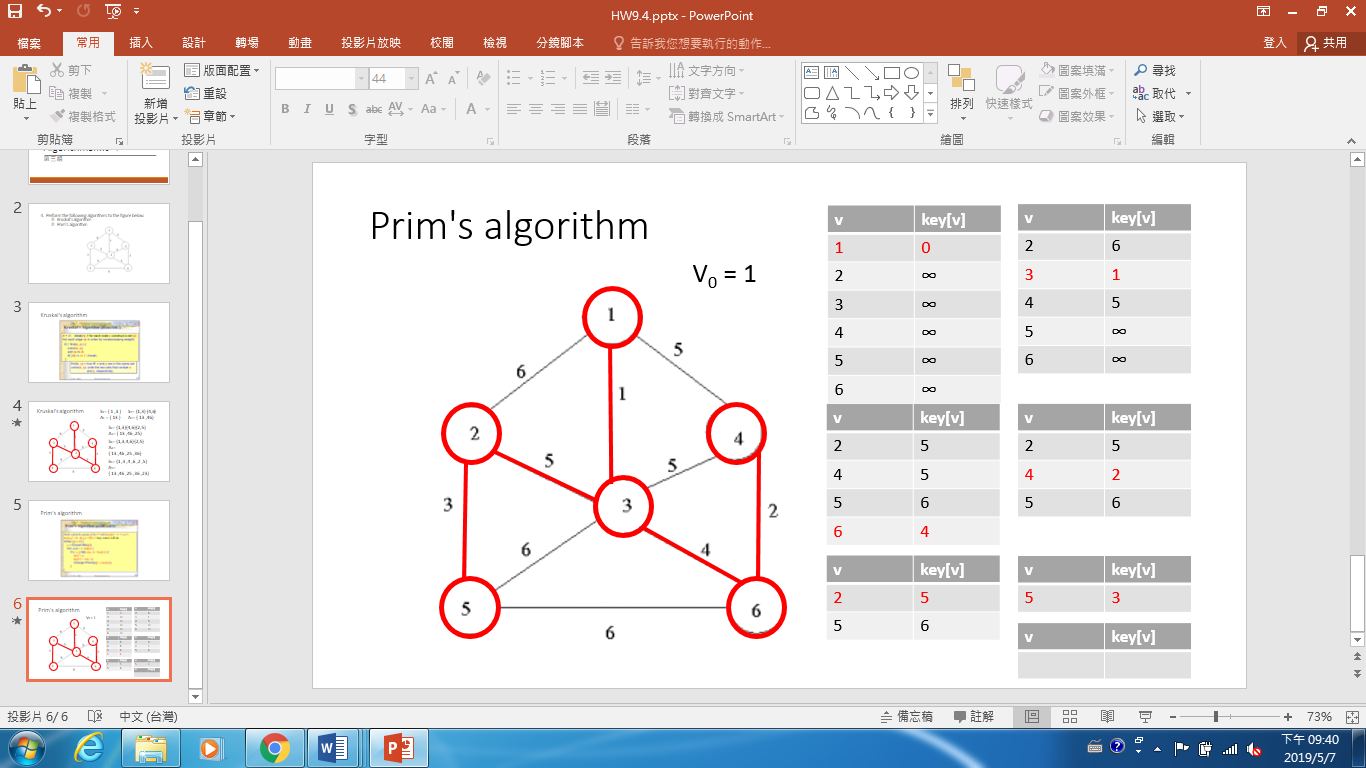
|  |  |
| --- | --- |
| **v** | **key[v]** |
| **2** | **5** |
| **4** | **5** |
| **5** | **6** |
| **6** | **4** |

**Q1 Q2 Q3**

**Q4 Q5 Q6**

|  |  |
| --- | --- |
| **v** | **key[v]** |
| **2** | **5** |
| **4** | **2** |
| **5** | **6** |

|  |  |
| --- | --- |
| **v** | **key[v]** |
| **2** | **5** |
| **5** | **6** |

**Q7**

|  |  |
| --- | --- |
| **v** | **key[v]** |
|  |  |

**5.**

**EXT. 8-2 The following statements may or may not be correct. In each case, either prove it (if it is correct) or give a counterexample (if it isn't correct). Always assume that the graph G=(V,E) is undirected and connected. Do not assume that edge weights are distinct unless this is specifically stated.**

**a. If a graph G has more than |V|-1 edges, and there is a unique heaviest edge, then this edge cannot be part of any MST.**

**b. If G has a cycle with a unique heaviest edge e, then e cannot be part of any MST.**

**c. Let e be any edge of minimum weight in G. Then e must be part of some MST.**

**d. If the lightest edge in a graph is unique, then it must be part of every MST. e. If e is part of some MST of G, then it must be lightest edge across some cut of G.**

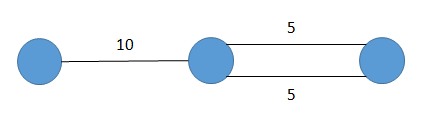
**f. If G has a cycle with a unique lightest edge e, then e must be part of every MST.**

**g. Prim's algorithm works correctly when there are negative edges.**

**h. (For any r>0, define an r-path to be a path whose edges all have weight**

a.False

反例：



b.True

因為MST不可以有cycle，所以cycle裡面至少有一個edge不在MST中。故我們可以選擇不要將權重最大的那個編放入MST。

c.False

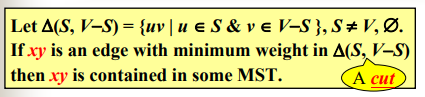
若有多個edge權重皆是最小值，不一定都會被放入MST中。

d.True

使用Kruskal演算法時，第一步就是將最小權重的邊放入MST。

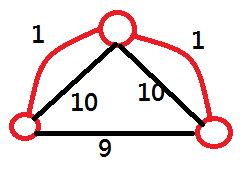
e.True

請見MST基本引理(PPT CH8 P.5)



f.False

反例:



g.True

Prim演算法只是不斷尋找權重最小的邊，negative edge沒有影響。或是你也可以將所有edge都加上一個正的常數使它們皆為正，做出來的MST相同。

h.True

r代表從頂點s到t經過各edge權重的upper bound。若MST中不包含r-path，則MST中必定存在一個edge e，其權重w>r。那我們就可以從r-path找到另一個edge e'，其權重w' < r < w，做出新的生成樹，其權重總和較原先的MST更小。

**6.**

**Problem 16-4 Scheduling variations Consider the following algorithm for the problem from Section 16.5 (Slide Unit06 P.26) of scheduling unit-time tasks with deadlines and penalties. Let all n time slots be initially empty, where time slot i is the unit-length slot of time that finishes at time i. We consider the tasks in order of monotonically decreasing penalty. When considering task aj, if there exists a time slot at or before aj's deadline dj that is still empty, assign aj to the latest such slot, filling it. If there is no such slot, assign task aj to the latest of the as yet unfilled slots.**

** Argue that this algorithm always gives an optimal answer.**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Tasks | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| Deadlines | 3 | 7 | 1 | 5 | 1 | 3 | 2 |
| penalties | 20 | 15 | 10 | 5 | 3 | 2 | 1 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 3(c) | 6(e) | 1(a) | 5(f) | 4(d) | 7(g) | 2(b) |

Time Slot( step : a to b )

Total penalties = 5 (task4) + 1 (task7) = 6

和找最大 profit 做法相同，但結果找超過 deadline 的 task。

透過以上例子可觀察，先將 penalties 大的 task 完成，因此超過 deadline 的 task 的 penalties必為最小。

** Use the fast disjoint-set forest presented in Section 21.3 to implement the algorithm efficiently. Assume that the set of input tasks has already been sorted into monotonically decreasing order by penalty. Please write down your pseudo-code and analyze the running time of your implementation.**

Assume

* MAKE-SET(x) returns a pointer to the element x which is now its own set. Store attributes x.low and x.high at the representative x of each disjoint set.This will give the earliest and latest time of a scheduled task in the block.
* UNION(x,y) maintains this attribute.
* An array A ,A[i] = 𝑎\_𝑖. task 𝑎\_(1^ ) has the greatest penalty, task 𝑎\_2 has the second greatest penalty, and so on,
* An array D such that D[i] contains a pointer to the task with deadline i. The size of D is at most n, since a task with deadline later than n can't possibly be scheduled on time.

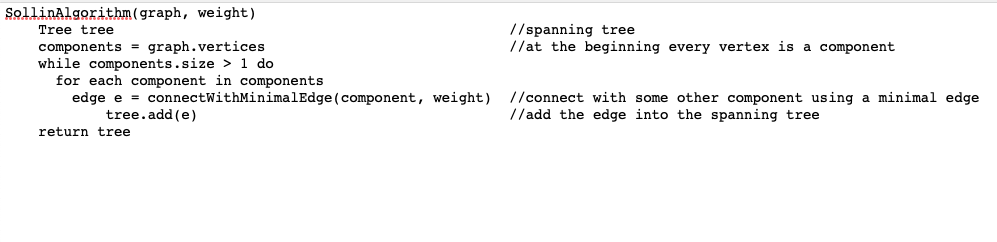
Pseudo cod

SCHEDULING-VARIATAIONS(A)  
let D[1….n] is a new array  
For i =1 to n  
　　a[i].time= a[i].deadline   
　　if D[a[i].deadline]!=NIL   
　　　　 y=FIND-SET(D[a[i].deadline])  
　　　　a[i].time = y.low - 1   
　　x = MAKE-SET(a[i])   
　　D[a[i].time] = x  
　　x.low = x.high = a[i].time   
　　if D[a[i].time - 1] != NIL  
　　　　UNION(D[a[i].time - 1], D[a[i].time])  
　　if D[a[i].time + 1] != NIL  
　　　　UNION(D[a[i].time], D[a[i].time + 1])

Time complexity : O(nα(n))

**7.**

**有關Minimum Spanning Tree，除了Kruskal's Algorithm和Prim's Algorithm外， 還有第三種演算法；此演算法第一步是先將圖中每個點當作一棵樹，然後從 各點分別尋找與其連接的最小權重(Weight)邊加入樹中，接著……。 請完成所有步驟（須寫出Pseudo-code）並分析此演算法。**



時間複雜度分析：每一次在component相連時，最差的情況是只有兩兩component相連，例如8個連3次成4個，4個連2次再連成2個，最後在連成1個，因此每次最多做O(lg|V|)次相連，而每一次相連時找最低的weight需要O(|E|)，因此時間複雜度為O(|E|\*lg|V|)。