第二周 簽名表：

名字	到	離
陳大佑		
巫思翰		
梁高銓		
韓欣蓉		

講義整理人：梁高銓

今日 Topic： ch21, 考古 (大部分上週講義有，建議作業念熟)

Ch21 : MST

課本習題：

> Let (u, v) be a minimum-weight edge in a connected graph G. Show that (u, v) belongs to some minimum spanning tree of G.

證明 kruskal 的過程，Suppose not, there exist OPT s.t. OPT + (u, v) have cycle, delete one of (w, v) is still spanning tree T and W(T) < W(OPT), 矛盾，所以(u,v) 會落在某個 OPT 裡

> Give a simple example of a connected graph such that the set of edges $\{(u, v)$: there exists a cut $(S, V - S)$ such that (u, v) is a light edge crossing $(S, V - S)\}$ does not form a minimum spanning tree.

考慮三個點的完全圖，且每一個邊都一樣的 weight，試試看這樣取到的就會是全部的邊

> Let e be a maximum-weight edge on some cycle of connected graph $G = (V, E)$. Prove that there is a minimum spanning tree of $G' = (V, E - \{e\})$ that is also a minimum spanning tree of G. That is, there is a minimum spanning tree of G that does not include e.

我們只需要 show T 不會有 e，假設他有 e，那麼我們根據第一題，應該可以得到一個 T'是更小的 spanning tree，矛盾。

> Show that a graph has a unique minimum spanning tree if, for every cut of the graph, there is a unique light edge crossing the cut. Show that the converse is not true by giving a counterexample.

Suppose the MST is not unique, T, T'會有一條(u,v)跟(x,y)使得 (u,v)在 T 不在 T'， (x,y)在 T'不在 T，然後考慮 S = {x, u}，那 S 的 cut 存在一個最小的，我們不妨假設他不是(u, v)，那就可以將 (u,v)換成更小的 Spanning tree，矛盾

Counter example 考慮三個邊，分別重 1, 1, 2，這樣他就有唯一的 MST，但不滿足所有 cut 都有最小。

> Argue that if all edge weights of a graph are positive, then any subset of edges that connects all vertices and has minimum total weight must be a tree. Give an example to show that the same conclusion does not follow if we allow some weights to be nonpositive.

Spanning tree : connected、no cycle

Connected is satisfy when we choose, we focus on no cycle.

Suppose there is a cycle, then we can remove the edge from the cycle to make the total weight be smaller.

Counter example : 考慮三個邊，分別重-1, -1, -1，最小取得的就會是三個邊

> Let T be a minimum spanning tree of a graph $G = (V, E)$, and let V' be a subset of V. Let T' be the subgraph of T induced by V', and let G' be the subgraph of G induced by V'. Show that if T' is connected, then T' is a minimum spanning tree of G'.

Suppose not, 存在一個 MST 在 G' 裡，叫 S，W(S) < W(T')，那我只需要將 T 裡面跟 S 有重疊到的點都換成 S 的邊連接，那我會取得一顆更 ` 小的 spanning tree，這跟我們一開始說 T 是 MST 矛盾

> Given a graph G and a minimum spanning tree T, suppose that we decrease the weight of one of the edges in T. Show that T is still a minimum spanning tree for G. More formally, let T be a minimum spanning tree for G with edge weights given by weight function w. Choose one edge $(x, y) \in T$ and a positive number k, and define the weight function w' by
>
> $$w'(u,v) = \begin{cases} w(u,v) & \text{if } (u,v) \neq (x,y), \\ w(x,y) - k & \text{if } (u,v) = (x,y). \end{cases}$$
>
> Show that T is a minimum spanning tree for G with edge weights given by w'.

Suppose T is not an MST in G', then there exist T' is MST and consider

(x,y) in T' imply W(T') + k < W(T) (in original G)

(x,y) not in T' imply T' is MST in original G (因為減掉一點再取都不是了，推回去也不會是)，矛盾

> Kruskal's algorithm can return different spanning trees for the same input graph G, depending on how it breaks ties when the edges are sorted into order. Show that for each minimum spanning tree T of G, there is a way to sort the edges of G in Kruskal's algorithm so that the algorithm returns T.

Suppose that we wanted to pick T as our minimum spanning tree. Then, to obtain this tree with Kruskal's algorithm, we will order the edges first by their weight, but then will resolve ties in edge weights by picking an edge first if it is contained in the minimum spanning tree, and treating all the edges that aren't in T as being slightly larger, even though they have the same actual weight.

With this ordering, we will still be finding a tree of the same weight as all the minimum spanning trees $w(T)$. However, since we prioritize the edges in T, we have that we will pick them over any other edges that may be in other minimum spanning trees.

重點題目

> Suppose that we represent the graph $G = (V, E)$ as an adjacency matrix. Give a simple implementation of Prim's algorithm for this case that runs in $O(V^2)$ time.

At each step of the algorithm we will add an edge from a vertex in the tree created so far to a vertex not in the tree, such that this edge has minimum weight. Thus, it will be useful to know, for each vertex not in the tree, the edge from that vertex to some vertex in the tree of minimal weight. We will store this information in an array A, where $A[u] = (v, w)$ if w is the weight of (u, v) and is minimal among the weights of edges from u to some vertex v in the tree built so far. We'll use $A[u].1$ to access v and $A[u].2$ to access w.

```
PRIM-ADJ(G, w, r)
    initialize A with every entry = (NIL, ∞)
    T = {r}
    for i = 1 to V
        if Adj[r, i] != 0
            A[i] = (r, w(r, i))
    for each u in V - T
        k = min(A[i].2)
        T = T ∪ {k}
        k.π = A[k].1
        for i = 1 to V
            if Adf[k, i] != 0 and Adj[k, i] < A[i].2
                A[i] = (k, Adj[k, i])
```

重點題目

> Suppose that all edge weights in a graph are integers in the range from 1 to $|V|$. How fast can you make Kruskal's algorithm run? What if the edge weights are integers in the range from 1 to W for some constant W?

If the edge weights are integers in the range from 1 to $|V|$, we can make Kruskal's algorithm run in $O(E\alpha(V))$ time by using counting sort to sort the edges by weight in linear time. I would take the same approach if the edge weights were integers bounded by a constant, since the runtime is dominated by the task of deciding whether an edge joins disjoint forests, which is independent of edge weights.

> Suppose that the edge weights in a graph are uniformly distributed over the halfopen interval $[0, 1)$. Which algorithm, Kruskal's or Prim's, can you make run faster?

For input drawn from a uniform distribution I would use bucket sort with Kruskal's algorithm, for expected linear time sorting of edges by weight. This would achieve expected runtime $O(E\alpha(V))$.

考古題

5. (10%) What is the running time and space of BFS if we present its input graph by an adjacency list?

```
BFS(G, source):
    Queue qq
    Source change to discovered
    qq.push(source)
    while until qq is not empty do
        now = qq.pop()
        for vertx in G do
            if vertex in adj[now] and vertex is undiscovered then
                vertex set to discovered
                qq.push(vertex)
        now set to finished
```

---

time complexity：對於每個點都會遍歷到，且每個點都會花 |V|的時間去查找相鄰且沒被尋找過的點，所以是 O(V^2)，空間複雜度，需要 V + V^2 的儲存空間。

1. (10%) Describe an efficient algorithm that, given a set $\{x_1, x_2, \ldots, x_n\}$ of points on the real line, determines the smallest set of unit-length closed intervals containing all the given points. Argue that your algorithm is correct.

所有點都佔一格，如果相鄰 < 1 就拉在同一格，這樣的會是最少的 lenth to cover，給定一個不是這樣的 OPT，可以透過位移跟減少長度拿到這樣的解

1. Let $Y = \{y_1, y_2 \ldots, y_n\}$
2. Initialize $S = \emptyset$
3. for k: 1 *to* n (while $Y \neq \emptyset$): // start from the leftmost point
   i. Take one point $y_k \in Y$, we have $I_k = [y_k, y_k + 1]$
   ii. $S \leftarrow S \cup \{I_k\}$ // put the interval into S
   iii. $Y \leftarrow Y - \{y_l : y_l \in I_k\}$ // remove the points in I_k from set Y
4. Return S

Proof

We claim that there is an optimal solution which contains the unit-length interval $[y_j, y_j + 1]$. Suppose that there exists an optimal solution S^* such that y_1 is covered by $[x', x' + 1] \in S^*$ where $x' < 1$. Since y_1 is the leftmost element of the given set, there is no other points lying in $[x', y_1)$. Consequently, if we replace $[x', x' + 1]$ *in* S^* by $[y_1, y_1 + 1]$, we will get another optimal solution. This proves the claim and thus explains the greedy choice property. Now, by solving the remaining

subproblem after removing all the points lying in $[y_1, y_1 + 1]$, that is, to find an optimal set of intervals, denoted as S', which cover the points to the right of $y_1 + 1$, we will get an optimal solution to the original problem by taking union of $[y_1, y_1 + 1]$ and S'.

Btw, 考試前把作業看完然後一起加油 \owo/！