HW2

Exercise 1.

Algorithm 1 Detect-Cycle (V, Adj)

for $v \in V$ **do**
    visited[v] $\leftarrow 0$ ▷ Initialize visit status
EdgeCount $\leftarrow 0$ ▷ Initialize edge counter
for $u \in V$ **do**
    if visited[u] $= 0$ **then** ▷ Perform DFS for each unvisited vertex
        if DETECT-CYCLE-VISIT(u, NIL, Adj, visited, EdgeCount) **then**
            return True ▷ Cycle detected
return False ▷ No cycle found

Algorithm 2 Detect-Cycle-Visit (v, $parent$, Adj, $visited$, $EdgeCount$)

visited[v] $\leftarrow 1$ ▷ Mark current vertex as visited
for $w \in$ Adj[v] **do**
    EdgeCount $\leftarrow$ EdgeCount $+ 1$ ▷ Increment edge counter
    if EdgeCount $\geq \mid V \mid$ **then**
        return True ▷ Examined $\mid V \mid$ edges, cycle must exist
    if visited[w] $= 0$ **then** ▷ Recursively visit unvisited neighbors
        if DETECTCYCLE-VISIT(w, v, Adj, visited, EdgeCount) **then**
            return True ▷ Cycle found in lower level
    else if $w \neq$ parent **then** ▷ Found visited neighbor that's not the parent
        return True ▷ Back edge detected, cycle exists
return False ▷ No cycle found in this path

1. 初始化 visited list - 遍歷所有節點，時間複雜度為 $O(|V|)$

2. DFS 遍歷 - 每個節點最多被 visit 一次：visited: $0 \rightarrow 1$ - EdgeCount 確保最多檢查 $|V|$ 條邊 - 如果檢查了 $|V|$ 條邊仍未找到循環，根據題目條件 $|E| < |V|$，可確定圖中存在循環

3. 總體時間複雜度 - 綜合上述分析，總體時間複雜度為 $O(|V|)$

Exercise 2.

邊的權重為1至W的整數。W的範圍為$| V | \leq W \leq | V |^{100}$，並且W已給定。

標準排序的時間複雜度為O($| E | log | E |$)，但在給定有限值域的情況下可以使用redix-sort。

$b = log_2 W$ bits。因為$W \leq | V |^{100}$，所以$b \leq 100 \cdot log_2 | V |$。

$n = | E |$。

因為$| E | \leq | V |^2, log_2 n = log_2 | E | \leq 2 \cdot log_2 | V | \leq 100 \cdot log_2 | V | = b$

當 $b \geq log_2 n$，設定 $r = log_2 n = log_2 | E |$

排序時間為

$$\begin{aligned} O\left(\frac{b}{r} \cdot (n + 2^r)\right) &= O\left((100 \cdot log_2 | V | / \lfloor log_2 | E | \rfloor)(| E | + | E |)\right) \\ &\approx O\left(\frac{100 \log_2 | V |}{2 \log_2 | V |} \cdot 2 | E |\right) \quad (\text{using } \log_2 | E | \leq 2 \log_2 | V |) \\ &= O(100|E|) \\ &= O(|E|) \end{aligned}$$

$$\begin{array}{llcc} \text{Make-Set} & |V| \text{ times} & \rightarrow & O(|V|) \\ \text{Find-Set} & |E| \text{ times} & \rightarrow & O(|E| \cdot \alpha(|V|)) \\ \text{Union} & \leq |E| \text{ times} & \rightarrow & O(|E| \cdot \alpha(|V|)) \\ \text{Radix-Sort} & \text{for} |E| \text{ elements} & \rightarrow & O(|E|) \\ \text{Total} & & & O(|E| \cdot \alpha(|V|)) \end{array} \tag{1}$$

Exercise 3.

根據題目假設任一邊 $e = (a, b) \in E'$是節點 a 最輕的邊。

假設一個情況是該邊 $e \notin T$，T 是一顆 minimum spanning tree，並且 $T \in G$。

將$T \cup e$，因為 T 是一顆MST，因此將剛好形成一個 cycle，這個cycle 包含 $e = (a, b)$。

因為 e 是節點 a 最輕的邊，在cycle中任意其餘連通 a 的邊 $e' = (a, c)$ 的權重必將大於e ($w(e') > w(e)$)。

現在，考慮如果從$T \cup e$移除e'，將得到另一個spanning tree $T' = T + e - e'$。

$w(T') = w(T) + w(e) - w(e') < w(T)$，將與 T 是 minimum spanning tree 的假設相矛盾。

將以上論述套用到所有邊 $e \in E'$，所有邊 $e \in E'$皆必將屬於所有 minimum spanning tree。

Exercise 4.

Algorithm 3 Find-Shortest-Path ($G=(V,E,w),\ s,\ t,\ \delta$)

```
let D[1 : |V|] be a new array                                ▷ Distance array
let B[0 : δ] be a new array                                  ▷ Bucket array
for v ∈ V do
    D[v] ← ∞                                                 ▷ Initialize all distances to infinity
D[s] ← 0                                                     ▷ Distance to source is 0
for i = 1 to δ do
    B[i] ← ∅                                                 ▷ Initialize buckets to empty sets
B[0] ← {s}                                                   ▷ Add source to bucket 0
for d = 0 to δ do
    while B[d] ≠ ∅ do
        Remove a vertex v from B[d]
        if v = t then
            return D[t]                                      ▷ Found shortest path to target
        for e = (v, u) ∈ E do
            new_dist ← D[v] + w(v, u)
            if new_dist < D[u] then                          ▷ Relax: Update Distance array and Bucket array
                if D[u] ≠ ∞ and D[u] ≤ δ then
                    B[D[u]] ← B[D[u]] − {u}
                D[u] ← new_dist
                B[new_dist] ← B[new_dist] ∪ {u}
return D[t]                                                  ▷ Return shortest distance to target
```

初始化距離array，從 1 至 $|V|$ - O($|V|$)

初始化Bucket array，從 1 至 δ - O(δ)

根據Bucket距離順序由小至大更新路徑距離，以BFS的方式遍歷節點及臨邊。

每個節點最多被加入(可能被移除) Bucket array一次 - O($|V|$)

每個邊最多被考慮一次，並且使用常數時間 O(1) 更新arrays - O($|E|$)

總時間複雜度: O($\delta + |V| + |E|$)

Exercise 5.

從 Π 到 D

要從predecessor matrix Π 計算出distance matrix D：

從計算單一個 i 到 j 的 d_{ij} 開始:

1. 對於每對頂點 (i, j)，從終點 j 開始。
2. 使用 $\Pi[i, j]$ 找出 j 的predecessor，並持續向前找，直到到達起點 i。
3. 累加沿著這條路徑的所有邊權重，得到 d_{ij}。

對n個起點，及每個起點對應的n-1個終點計算上述過程，即可從 predecessor matrix 轉換成 distance matrix

Pseudocode 如下:

Algorithm 4 Computing-D-from-Π (*G=(V,E,w),* Π)

```
Initialize D[i][j] = ∞ for all i ≠ j, and D[i][i] = 0 for all i
for each edge (i, j) ∈ E do
    D[i][j] = w(i, j)
for each source i ∈ V do
    for each destination j ∈ V where i ≠ j do
        if Π[i][j] ≠ NIL then
            dist = 0
            current = j
            prev = Π[i][current]
            while prev ≠ i do
                dist = dist + w(prev, current)
                current = prev
                prev = Π[i][current]
            dist = dist + w(i, current)
            D[i][j] = dist
return D
```

計算時間複雜度:

$n \leftarrow \mid V \mid$

對於每對頂點 (i, j)，最壞情況下可能需要遍歷長度為 $O(n)$ 的路徑。

總共有 $n * (n - 1)$ 對頂點。

總時間複雜度為 $O(n^3)$。

從 D 到 Π

要從distance matrix D 計算出 predecessor matrix Π:

1. 對於每對頂點 (i, j)，我們需要找出從 i 到 j 的最短路徑上，j 的predecessor。

2. 通過檢查 i 的每個鄰居 k，判斷是否滿足 $D[i, j] = w(i, k) + D[k, j]$。

3. 若找到符合條件的 k，則 k 是沿著從 i 到 j 最短路徑的第一步。

所有可能的predecessor為j的鄰居中方向指向j的(adj)，可藉由計算i到所有鄰居的最短距離和鄰居與j的邊的權重($w_{adj,j}$)確定正確的predecessor。

Pseudocode 如下:

Algorithm 5 Compute-Π-From-D (*G=(V,E,w), D*)

$n \leftarrow\mid V \mid$
Initialize $\Pi[i, j] = \text{NIL}$ for all i, j
for $i = 1$ to n **do**
    for $j = 1$ to n **do**
        if $i \neq j$ **then**
            for each vertex k adjacent to i **do**
                if $D[i, j] = w(i, k) + D[k, j]$ **then**
                    $\Pi[i, j] \leftarrow k$
                    break
return Π

計算時間複雜度:

$n \leftarrow\mid V \mid$

對於每對頂點 (i, j)，我們檢查 i 的所有鄰居（最多 $O(n)$ 個節點）。

總共有 $n * (n - 1)$ 對頂點。

總時間複雜度為 $O(n^3)$。

Exercise 6.

6-1.

Algorithm 6 ADAPTED-FASTER-APSP ($G{=}(V,E,w)$)

$n \leftarrow |V|$
Let $L^{(1)}$ be an $n \times n$ matrix
for $u = 1$ to n **do**
  for $v = 1$ to n **do**
    if $u = v$ **then**
      $L^{(1)}[u, v] \leftarrow 0$
    else if $(u, v) \in E$ **then**
      $L^{(1)}[u, v] \leftarrow w(u, v)$
    else
      $L^{(1)}[u, v] \leftarrow \infty$
$m \leftarrow 1$
while $m < c \cdot \log n$ **do** ▷ where c is the constant in $O(\log n)$
  /* Form 1: min-plus matrix multiplication */
  $L^{(2m)} \leftarrow L^{(m)} \otimes L^{(m)}$
  /* Form 2: Detailed expansion of min-plus matrix multiplication */
  for $i = 1$ to n **do**
    for $j = 1$ to n **do**
      $L^{(2m)}[i, j] \leftarrow \infty$
      for $k = 1$ to n **do**
        $L^{(2m)}[i, j] \leftarrow \min(L^{(2m)}[i, j], L^{(m)}[i, k] + L^{(m)}[k, j])$
  $m \leftarrow 2m$
return $L^{(m)}$

原本每個最短路徑要遞迴 $O(logn)$ 次，導致總複雜度為$O(n^3logn)$。
當每個最短路徑使用 $O(logn)$ 個邊時，總複雜度可以降得更低。
因為限制最短路徑最多使用 $logn$ 個邊，每次遞迴變成 $O(loglogn)$ 次。
總複雜度變成$O(n^3loglogn)$

6-2.

Floyd-Warshall 算法的輸出為一個 distance matrix D 和一個 predecessor matrix Π。

透過檢查distance matrix D，如果任何點到自己的最短距離為負值，則必有一個包含該點的negative cycle。

因此可以以該點為起點，使用 predecessor matrix Π 持續尋找直到回到起點，即建構完成negative cycle。

Pseudocode 如下:

Algorithm 7 FIND-NEGATIVE-CYCLE (D, Π)

```
for v = 1 to |V| do
    if D[v][v] < 0 then                          ▷ Found a vertex on a negative cycle
        cycle ← [v]
        current ← Π[v][v]
        while current ≠ v do
            cycle ← cycle + [current]
            current ← Π[v][current]
        cycle ← cycle + [v]                      ▷ insert v at ending of cycle
        return cycle
return False                                     ▷ No negative cycle found
```

計算時間複雜度:

$n \leftarrow | V |$

遍歷所有節點，找尋到自己距離為負的節點。對每個節點 v 用常數時間檢查 $D[v][v]$，總時間為$O(n)$。

如果找到，則開始建立cycle。最壞的情況下，該循環可能包含所有節點共 n 個。對每個節點 current 用常數時間從 $\Pi[v]$[current] 找predecessor。總時間為 $O(n)$。

因此，總時間複雜度為 $O(n) + O(n) = O(n)$。

Exercise 7.

給定起點 s 和終點 t ，要找最大數量的vertex-disjoint path。

這個問題可以 reduce 成一個 maximum flow 問題，然後使用Ford-Fulkerson 或 Edmonds-Karp Algorithm 解決。

透過將flow network 每個邊的權重都變為 1，保證路徑不被重複占用。但這樣僅保證 edge-disjointness，仍然可以經過同個節點。

因此，可以將除了起點和終點以外的節點v，進一步拆分成兩個節點 vin 和 vout，同樣將權重設為1，確保單一節點v只能經過一次。

Pseudocode 如下:

Algorithm 8 Maximum-Vertex-Disjoint-Paths (*G=(V,E,w), s, t*)

Create an empty flow network $G' = (V', E', w')$
$V' \leftarrow \{s, t\}$
for $v \in V \setminus \{s, t\}$ **do**
  $V' \leftarrow V' \cup \{v_{in}, v_{out}\}$
  $E' \leftarrow E' \cup \{(v_{in}, v_{out})\}$
  $w'(v_{in}, v_{out}) \leftarrow 1$
for $e = (u, v) \in E$ **do**
  if $u = s$ **then**
    $E' \leftarrow E' \cup \{(s, v_{in})\}$
    $w'(s, v_{in}) \leftarrow 1$
  else if $v = t$ **then**
    $E' \leftarrow E' \cup \{(u_{out}, t)\}$
    $w'(u_{out}, t) \leftarrow 1$
  else
    $E' \leftarrow E' \cup \{(u_{out}, v_{in})\}$
    $w'(u_{out}, v_{in}) \leftarrow 1$
$flow \leftarrow$ Ford-Fulkerson(G', s, t)
$paths \leftarrow$ Extract-Paths-From-Flow$(G', flow)$
return $paths$

Algorithm 9 Extract-Paths-From-Flow (*G'=(V',E',w'), flow*)

$paths \leftarrow \emptyset$ ▷ Initialize empty list of paths
while $\exists P = (s, v_{in1}, v_{out1}, v_{in2}, v_{out2}, \ldots, t)$ such that $flow(e) > 0$ for all $e \in P$ **do**
    $path \leftarrow \emptyset$ ▷ Initialize current path
    $path \leftarrow path \cup \{v\}$
    $current \leftarrow v_{out}$
    while $current \neq t$ **do**
        if $\exists(current, t) \in E'$ such that $flow(current, t) > 0$ **then**
            $current \leftarrow t$
        else
            Select $(current, v_{in}) \in E'$ such that $flow(current, v_{in}) > 0$
            $path \leftarrow path \cup \{v\}$
            $current \leftarrow v_{out}$
    $paths \leftarrow paths \cup \{path\}$
    for each edge $e \in P$ **do**
        $flow(e) \leftarrow flow(e) - 1$
return *paths*

建立 G' 花費時間 $O(|V| + |E|)$。

使用Ford-Fulkerson with BFS 計算 Maximum flow 花費時間 $O(|V'| \cdot |E'|^2)$。

$|V'| = 2 \cdot |V| - 2 \approx 2 \cdot |V|$ 以及 $|E'| = |E| + |V| - 2 \approx |E| + |V|$，計算 Maximum flow 花費時間 $O(|V| \cdot |E|^2)$。

從 flow 中提取 paths，除了s 和 t 每個節點最多被檢查一次，並且每個邊也最多被檢查一次，時間為 $O(|E| + |V|)$。

總時間複雜度為 $O(|V| \cdot |E|^2)$。

Exercise 8.

一個演算法被認為是 polynomial-time 表示時間複雜度可以表達成$O(n^k)$，其中 k 是某個常數。本題目中算法的時間複雜度為 $O(|V|^2 \cdot |E|^9 \cdot \delta)$。其中決定算法是否為 polynomial-time 的關鍵因素為 δ，因為 δ 可能以輸入大小 n 的指數級增加，表示算法的運行時間可能以輸入大小 n 的指數級增加。例如，$\delta = 2^n$，其中 n 為輸入大小，那算法運行花費的時間將為 $O(|V|^2 \cdot |E|^9 \cdot 2^n)$，為exponential-time 而非 polynomial-time。總而言之，本題目中的算法不是一個 polynomial-time 的算法，因為它的運行時間由 δ 決定，而 δ 可能以輸入大小 n 的指數級增加。