第三周 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： ch22、ch23 觀念前半、CH21 題目補充

Ch21 題目補充

- Which of the following statements are true for the Minimum Spanning Tree (MST) of a graph G =(V, E)?

a. MST is the spanning tree that have the minimum weight

b. MST of a graph is not unique

c. MST has exactly |V| -1 edges

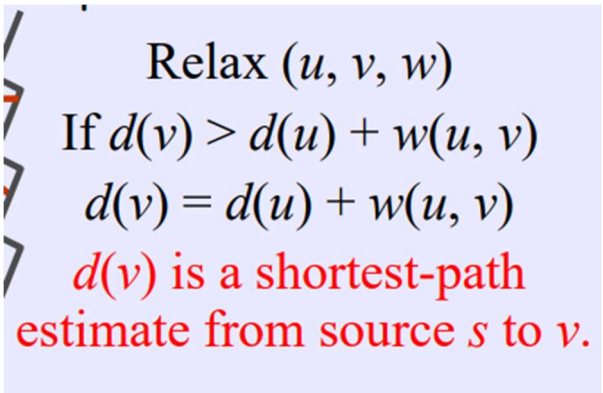
a. T

b. T

c. T

ch22 講義複習

首先我們要先知道甚麼是 relax

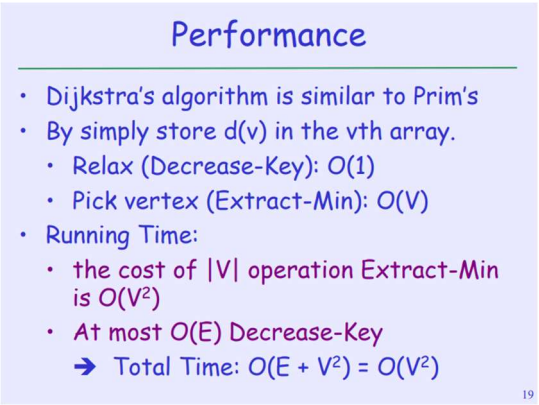


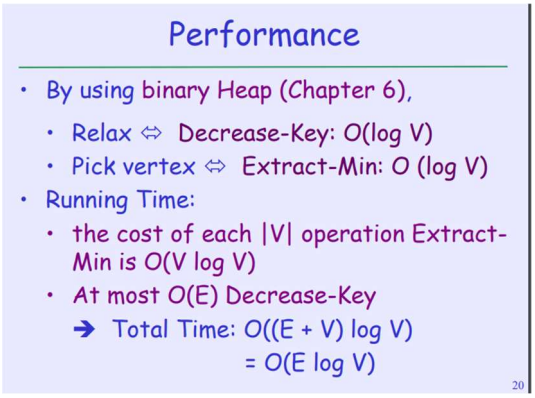
接下來看一下 Dijkstra

```
Dijkstra(G, s)
   For each vertex v,
       Mark v as unvisited, and set d(v) = ∞ ;
   Set d(s) = 0 ;
   while (there is unvisited vertex) {
      v = unvisited vertex with smallest d(v) ;
      Visit v, and Relax all its outgoing edges;
   }
   Return d;
```

例子建議參考老師的講義，接下來簡述證明的邏輯

我們會利用數學歸納法，因為我們是一個一個把最短路徑確認下來的吧，所以假設選到第 n 個以前他都會是最短路，那現在討論選第 n 個時的情況，讓第 n 個選到的點是 v，那我知道選到第 n 個時也會是最短路，因為如果我可以透過其他邊走到 v 且有更短的路徑的畫，那這一次就不會選 v，所以就證明完了，接下來看看他的 performance。





Finding Shortest Path in DAG

We have a faster algorithm for DAG :

```
DAG-Shortest-Path(G, s)
   Topological Sort G ;
   For each v, set d(v) = ∞ ; Set d(s) = 0 ;
   for (k = 1 to |V|) {
      v = k^th vertex in topological order ;
      Relax all outgoing edges of v ;
   }
   return d ;
```

Performance

- DAG-Shortest-Path selects vertex sequentially according to topological order
  - no need to perform Extract-Min
- We can store the d values of the vertices in a single array ➔ Relax takes $O(1)$ time
- Running Time:
  - Topological sort : $O(V + E)$ time
  - $O(V)$ select, $O(E)$ Relax : $O(V + E)$ time

  ➔ Total Time: $O(V + E)$

Bellman-Ford Algorithm

```
Bellman-Ford(G, s)     // runs in O(VE) time
   For each v, set d(v) = ∞ ; Set d(s) = 0 ;
   for (k = 1 to |V|-1)
      Relax all edges in G in any order ;
   /* check if s reaches a neg-weight cycle */
   for each edge (u,v),
      if (d(v) > d(u) + weight(u,v))
            return "something wrong !!" ;
   return d ;
```

Path-Relaxation Property

- Consider any shortest path p from $s = v_0$ to v_k, and let $p = (v_0, v_1, \ldots, v_k)$. If we relax the edges (v_0, v_1), (v_1, v_2), …, (v_{k-1}, v_k) in order, then $d(v_k)$ is the shortest path from s to v_k. Proof by induction (omit)

Performance

- When no negative edges
  - Dijkstra's algorithm
    - Using array $O(V^2)$
    - Using Binary heap implementation: $O(E \lg V)$
    - Using Fibonacci heap: $O(E + V \log V)$
- When DAG
  - DAG-Shortest-Paths: $O(E + V)$ time
- When negative cycles
  - Using Bellman-Ford algorithm: $O(V E) = (V^3)$

ch22 作業習題

*

> Given a weighted, directed graph $G = (V, E)$ with no negative-weight cycles, let m be the maximum over all vertices $v \in V$ of the minimum number of edges in a shortest path from the source s to v. (Here, the shortest path is by weight, not the number of edges.) Suggest a simple change to the Bellman-Ford algorithm that allows it to terminate in $m + 1$ passes, even if m is not known in advance.

By the upper bound theory, we know that after m iterations, no d values will ever change. Therefore, no d values will change in the $(m + 1)$-th iteration. However, we do not know the exact m value in advance, we cannot make the algorithm iterate exactly m times and then terminate. If we try to make the algorithm stop when every d values do not change anymore, then it will stop after $m + 1$ iterations.

> Let $G = (V, E)$ be a weighted, directed graph with weight function $w : E \to \mathbb{R}$. Give an $O(VE)$-time algorithm to find, for each vertex $v \in V$, the value $\delta^*(v) = \min_{u \in V}\{\delta(u, v)\}$.

題目需要理解一下，所以就是要找出所有路徑中，走到 v 那一點的最短有多長？那看到時間複雜度是 O(VE)大概知道我們要用 bellman ford 但要怎麼用？

Initially, we will make each vertex have a D value of 0, which corresponds to taking a path of length zero starting at that vertex. Then, we relax along each edge exactly $V-1$ times. Then, we do one final round of relaxation, which if any thing changes, indicated the existence of a negative weight cycle. The code for this algorithm is identical to that for Bellman ford, except instead of initializing the values to be infinity except at the source which is zero, we initialize every d value to be infinity. We can even recover the path of minimum length for each vertex by looking at their π values.

Note that this solution assumes that paths of length zero are acceptable. If they are not to your liking then just initialize each vertex to have a d value equal to the minimum weight edge that they have adjacent to them.

*

Suppose that a weighted, directed graph $G = (V, E)$ has a negative-weight cycle. Give an efficient algorithm to list the vertices of one such cycle. Prove that your algorithm is correct.

Begin by calling a slightly modified version of DFS, where we maintain the attribute $v.d$ at each vertex which gives the weight of the unique simple path from s to v in the DFS tree. However, once $v.d$ is set for the first time we will never modify it. It is easy to update DFS to keep track of this without changing its runtime. At first sight of a back edge (u, v), if $v.d > u.d + w(u, v)$ then we must have a negative-weight cycle because $u.d + w(u, v) - v.d$ represents the weight of the cycle which the back edge completes in the DFS tree. To print out the vertices print v, u, $u.\pi$, $u.\pi.\pi$, and so on until v is reached. This has runtime $O(V + E)$.

*

> The PERT chart formulation given above is somewhat unnatural. In a more natural structure, vertices would represent jobs and edges would represent sequencing constraints; that is, edge (u, v) would indicate that job u must be performed before job v. We would then assign weights to vertices, not edges. Modify the DAG-SHORTEST-PATHS procedure so that it finds a longest path in a directed acyclic graph with weighted vertices in linear time.

Introduce two new dummy tasks, with cost zero. The first one having edges going to every task that has no in edges. The second having an edge going to it from every task that has no out edges. Now, construct a new directed graph in which each e gets mapped to a vertex v_e and there is an edge $(v_e, v_{e'})$ with cost w if and only if edge e goes to the same vertex that e' comes from, and that vertex has weight w. Then, every path through this dual graph corresponds to a path through the original graph. So, we just look for the most expensive path in this DAG which has weighted edges using the algorithm from this section.

底下利用這章節的，就是 DAG-FIND-SHOTEST-PATH，但因為要找最長，所以序再 relax 裡的需要修正一下。且初始化是負無窮。

> Give an efficient algorithm to count the total number of paths in a directed acyclic graph. Analyze your algorithm.

We will compute the total number of paths by counting the number of paths whose start point is at each vertex v, which will be stored in an attribute $v.paths$. Assume that initial we have $v.paths = 0$ for all $v \in V$. Since all vertices adjacent to u occur later in the topological sort and the final vertex has no neighbors, line 4 is well-defined. Topological sort takes $O(V + E)$ and the nested for-loops take $O(V + E)$ so the total runtime is $O(V + E)$.

Algorithm 1 PATHS(G)

```
1: topologically sort the vertices of G
2: for each vertex u, taken in reverse topologically sorted order do
3:     for each vertex v ∈ G.Adj[u] do
4:         u.paths = u.paths + 1 + v.paths
5:     end for
6: end for
```

*

> Give a simple example of a directed graph with negative-weight edges for which Dijkstra's algorithm produces incorrect answers. Why doesn't the proof of Theorem 24.6 go through when negative-weight edges are allowed?

Consider any graph with a negative cycle. RELAX is called a finite number of times but the distance to any vertex on the cycle is $-\infty$, so Dijkstra's algorithm cannot possibly be correct here. The proof of theorem 24.6 doesn't go through because we can no longer guarantee that

$$\delta(s, y) \leq \delta(s, u).$$

Btw, 就是繞的回去的意思，例子自己造造看

*

> Professor Newman thinks that he has worked out a simpler proof of correctness for Dijkstra's algorithm. He claims that Dijkstra's algorithm relaxes the edges of every shortest path in the graph in the order in which they appear on the path, and therefore the path-relaxation property applies to every vertex reachable from the source. Show that the professor is mistaken by constructing a directed graph for which Dijkstra's algorithm could relax the edges of a shortest path out of order.

Trivial not true，例子造造看，也可以參考以下

Consider the graph on 5 vertices $\{a, b, c, d, e\}$, and with edges $(a, b), (b, c), (c, d), (a, e), (e, c)$ all with weight 0. Then, we could pull vertices off of the queue in the ordera, e, c, b, d. This would mean that we relax (c, d) before (b, c). However, a shortest pat to d is $(a, b), (b, c), (c, d)$. So, we would be relaxing an edge that appears later on this shortest path before an edge that appears earlier.

*

> We are given a directed graph $G = (V, E)$ on which each edge $(u, v) \in E$ has an associated value $r(u, v)$, which is a real number in the range $0 \leq r(u, v) \leq 1$ that represents the reliability of a communication channel from vertex u to vertex v. We interpret $r(u, v)$ as the probability that the channel from u to v will not fail, and we assume that these probabilities are independent. Give an efficient algorithm to find the most reliable path between two given vertices.

We now view the weight of a path as the reliability of a path, and it is computed by taking the product of the reliabilities of the edges on the path. Our algorithm will be similar to that of DIJKSTRA, and have the same run-time, but we now wish to maximize weight, and RELAX will be done inline by checking products instead of sums, and switching the inequality since we want to maximize reliability. Finally, we track that path from y back to x and print the vertices as we go.

Algorithm 2 RELIABILITY(G, r, x, y)

```
1:  INITIALIZE-SINGLE-SOURCE(G, x)
2:  S = ∅
3:  Q = G.V
4:  while Q ≠ ∅ do
5:      u = EXTRACT-MIN(Q)
6:      S = S ∪ {u}
7:      for each vertex v ∈ G.Adj[u] do
8:          if v.d < u.d · r(u, v) then
9:              v.d = u.d · r(u, v)
10:             v.π = u
11:         end if
12:     end for
13: end while
14: while y ≠ x do
15:     Print y
16:     y = y.π
17: end while
18: Print x
```

Btw，應該是 extract-max，因為每次要取最大的，如果往外繞只會更小。

*

> Suppose that we are given a weighted, directed graph $G = (V, E)$ in which edges that leave the source vertex s may have negative weights, all other edge weights are nonnegative, and there are no negative-weight cycles. Argue that Dijkstra's algorithm correctly finds shortest paths from s in this graph.

就是老師說的，只有 source 出來的那幾條有負邊，直覺來想他沒辦法繞回來拿到更小的 path，接下來就是比較正規的說法。

> The proof of correctness, Theorem 24.6, goes through exactly as stated in the text. The key fact was that $\delta(s, y) \leq \delta(s, u)$. It is claimed that this holds because there are no negative edge weights, but in fact that is stronger than is needed. This always holds if y occurs on a shortest path from s to u and $y \neq s$ because all edges on the path from y to u have nonnegative weight. If any had negative weight, this would imply that we had "gone back" to an edge incident with s, which implies that a cycle is involved in the path, which would only be the case if it were a negative-weight cycle. However, these are still forbidden.

Ch22 附錄 Dijkstra 證明 in textbook

each time it adds a vertex u to set S.

Theorem 22.6 (Correctness of Dijkstra's algorithm)
Dijkstra's algorithm, run on a weighted, directed graph $G = (V, E)$ with nonnegative weight function w and source vertex s, terminates with $u.d = \delta(s, u)$ for all vertices $u \in V$.

Proof We will show that at the start of each iteration of the **while** loop of lines 6–12, we have $v.d = \delta(s, v)$ for all $v \in S$. The algorithm terminates when $S = V$, so that $v.d = \delta(s, v)$ for all $v \in V$.

The proof is by induction on the number of iterations of the **while** loop, which equals $|S|$ at the start of each iteration. There are two bases: for $|S| = 0$, so that $S = \emptyset$ and the claim is trivially true, and for $|S| = 1$, so that $S = \{s\}$ and $s.d = \delta(s, s) = 0$.

For the inductive step, the inductive hypothesis is that $v.d = \delta(s, v)$ for all $v \in S$. The algorithm extracts vertex u from $V - S$. Because the algorithm adds u into S, we need to show that $u.d = \delta(s, u)$ at that time. If there is no path from s to u, then we are done, by the no-path property. If there is a path from s to u, then, as Figure 22.7 shows, let y be the first vertex on a shortest path from s to u that is not in S, and let $x \in S$ be the predecessor of y on that shortest path. (We could have $y = u$ or $x = s$.) Because y appears no later than u on the shortest path and all edge weights are nonnegative, we have $\delta(s, y) \leq \delta(s, u)$. Because the call of EXTRACT-MIN in line 7 returned u as having the minimum d value in $V - S$, we also have $u.d \leq y.d$, and the upper-bound property gives $\delta(s, u) \leq u.d$.

Since $x \in S$, the inductive hypothesis implies that $x.d = \delta(s, x)$. During the iteration of the **while** loop that added x into S, edge (x, y) was relaxed. By the convergence property, $y.d$ received the value of $\delta(s, y)$ at that time. Thus, we have

$$\delta(s, y) \leq \delta(s, u) \leq u.d \leq y.d \quad \text{and} \quad y.d = \delta(s, y) ,$$

so that

$$\delta(s, y) = \delta(s, u) = u.d = y.d .$$

Hence, $u.d = \delta(s, u)$, and by the upper-bound property, this value never changes again. ■

ch23 講義複習

- Define: $l_{ij}^{(m)}$ = minimum weight of any path from i to j that contains at most m edges.
- $l_{ij}^{(0)} = \begin{cases} 0 & \text{if } i = j \\ \infty & \text{if } i \neq j \end{cases}$
- Then $l_{ij}^{(m)} = \min\{ l_{ij}^{(m-1)}, \min_{1 \le k \le n} \{l_{ik}^{(m-1)} + w_{kj}\}\} = \min_{1 \le k \le n} \{l_{ik}^{(m-1)} + w_{kj}\}$ (why?)

- Given matrices $L^{(m-1)}$ and W return $L^{(m)}$

```
1  n <- L.row
2  Let L' = (l'_ij) be a new n x n matrix
3  for i = 1 to n
4     for j = 1 to n
5        l'_ij = ∞
6        for k = 1 to n
7           l'_ij = min(l'_ij, l_ik + w_kj)
8  return L'
```

```
1  n = W.rows
2  L(1) = W
3  for m = 2 to n-1
4       let L(m) be a new n x n matrix
5       L(m) = EXTENDED-SHORTEST-
   PATHS( L(m-1), W )
6  return L(n-1)
```

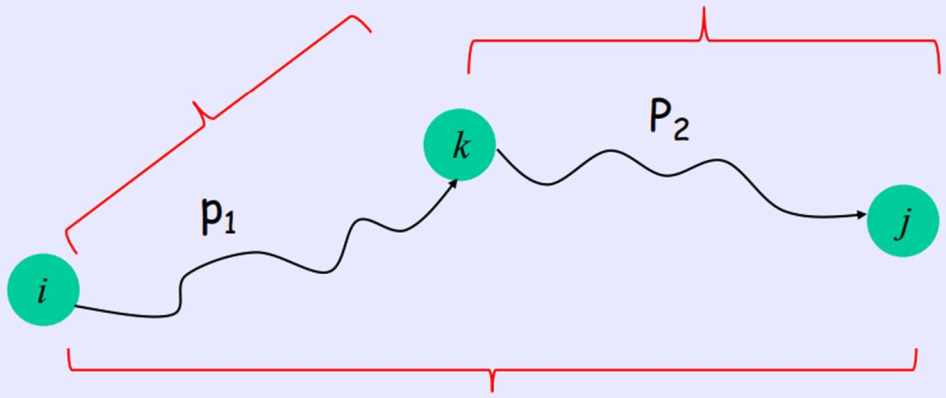
Time complexity : $O(n^4)$

```
FASTER-ALL-PAIRS-SHORTEST-PATHS(W)
1.  n =W.row
2.  L(1) =W
3.  m =1
4.  while m < n-1
5.       let L(2m) be a new n x n matrix
6.       L(2m) = Extend-Shorest-Path(L(m), L(m))
7.       m = 2m
8.  return L(m)
```

O(n^3 lgn)

p_1: all intermediate vertices in {1, 2, 3…., k-1}

p_2: all intermediate vertices in {1, 2, 3…., k-1}



P: all intermediate vertices in {1, 2, 3…., k}

FLOYD-WARSHALL(W)

1. n = W.rows
2. $D^{(0)} = W$
3. for k = 1 to n
4. Let $D^{(k)} = (d_{ij}^{(k)})$ be a new n x n matrix
5. for i = 1 to n
6. for j = 1 to n
7. $d_{ij}^{(k)} = \min (d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)})$
8. return $D^{(n)}$

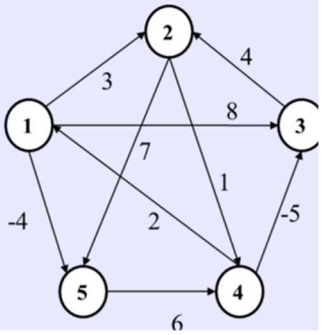
Complexity: $O(n^3)$

Example

$$D^{(0)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \qquad D^{(1)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix}$$

$d_{ij}^{(1)} = \min (d_{ij}^{(0)}, d_{ik}^{(0)} + d_{kj}^{(0)})$ k = 1

$d_{41}^{(1)} = \min (d_{41}^{(0)}, d_{41}^{(0)} + d_{11}^{(0)}) = 2$
$d_{42}^{(1)} = \min (d_{42}^{(0)}, d_{41}^{(0)} + d_{12}^{(0)}) = 5$
$d_{43}^{(1)} = \min (d_{43}^{(0)}, d_{41}^{(0)} + d_{13}^{(0)}) = -5$
$d_{44}^{(1)} = \min (d_{44}^{(0)}, d_{41}^{(0)} + d_{14}^{(0)}) = 0$
$d_{45}^{(1)} = \min (d_{45}^{(0)}, d_{41}^{(0)} + d_{15}^{(0)}) = -2$



$$D^{(1)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \qquad D^{(2)} = \begin{pmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix}$$

k = 1 k = 2

$d_{41}^{(2)} = \min (d_{41}^{(1)}, d_{42}^{(1)} + d_{21}^{(1)}) = 2$
$d_{42}^{(2)} = \min (d_{42}^{(1)}, d_{42}^{(1)} + d_{22}^{(1)}) = 5$
$d_{43}^{(2)} = \min (d_{43}^{(1)}, d_{42}^{(1)} + d_{23}^{(1)}) = -5$
$d_{44}^{(2)} = \min (d_{44}^{(1)}, d_{42}^{(1)} + d_{24}^{(1)}) = 0$
$d_{45}^{(2)} = \min (d_{45}^{(1)}, d_{42}^{(1)} + d_{25}^{(1)}) = -2$

補充

Longest_path use dp and topologic sort

https://courses.cs.washington.edu/courses/cse421/18au/lecture/lecture-17.pdf