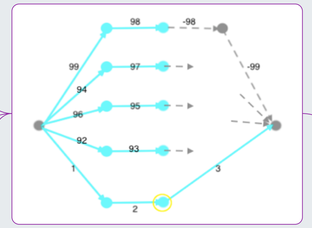
Graph

- 最短路徑問題
  - Dijkstra's algo.
    - single source to other destinations
    - 不可有負邊
      - 對頂點插旗
        - 一路插旗子來到黃點
        - 站在黃點上看，終點比出發點的其他鄰近點都近
        - 殊不知其他路線後面有蟲洞可以跳



    - Greedy
  - Bellman-Ford
    - single source to other destinations
    - 允許負邊
      - 對不同步數的路線，逐步修正
    - Dynamic Programming
  - Floyd-Warshall
    - all pair of vertex
    - 允許負邊
    - Dynamic Programming
  - 都不允許負長度的 cycle
- Edge 種類
  - Tree edge
    - DFS 經過的邊
  - Back edge
    - DFS 一直線畫出來，反向連回祖先的稱之
    - 可用於有無 cycle 的判斷
      - back edge ↔ cycle
      - 無向圖中以 pi 和 color 判斷 back edge
        - for vertex u, v and edge (u, v)
          - 由 u 出發追蹤
          - 到 v 的時候，u 為灰
          - v.pi = u，故 (u,v) 不為後退邊
      - 有向圖中不需要 pi，color 即可判斷
  - Forward edge (有向圖)
    - DFS 一直線畫出來，正向連到子孫的稱之
  - Cross edge (有向圖)
    - 非上述三種稱之
    - 實際運作上 F&C 還需要用 discovered time 區別
- Spanning Tree
  - 連通、無 cycle
  - min Spanning Tree
    - Kurskal's algo.
      - 對 G 的所有展開樹而言，edge cost 之和最小者稱之
      - edge cost 皆不同，則為唯一
      - 找最小 Cost
        - 使用 min-Heap
          - 依序 delete
            - O(loge)
      - cycle 判斷
        - 用第五章的 disjoint set 之 Union & Find
          - 將 vertex 視為 disjoint set
          - Find(u) ≠ Find(v)
            - Union(u, v)
            - edge (u,v) 加入 spanning tree
          - Find(u) == Find(v)
            - skip it
        - Union = O(1), Find = O(略)
          - 相加趨近於 O(1)
      - total = O(eloge)
        - 或 O(elogv)
          - e < v^2
            - 最多邊就是 Kv
      - **對 edge 挑選**
    - Prim's algo.
      - **對 vertex 挑選**
        - 依連接的 edge cost 挑選要加入的點
    - Sollin's algo.
      - 一開始每個頂點都視為獨立的 root
      - 每棵樹找出最小 cost 的邊，將其連接的 vertex 加入該 tree
      - 刪去重複的邊
        - e.g. (u,v) & (v,u)
      - 重複直到只剩一顆樹，或 E 為空
    - 都是 greedy
- AOV/AOE Network & Topplogical sort
  - 略
- 基礎
  - 尤拉 邊
  - 漢彌爾頓 點
  - storngly connected 有向圖，任兩點之間都有路徑到到
  - biconnected 沒有切點
- 表示法
  - Adjacency matrix
    - 點對點有無邊連結
    - n*n
    - 空間需求大、適合邊多的圖
  - Adjacency list
    - vertex 組成陣列
    - 以 link list 串連與該點相臨的點
    - 簡單應用
      - 易於檢查頂點對有無連接 O(e)
      - 易於檢查頂點 degree
      - 計算總邊數 V = 2|E|
  - Adjacency Multilist
    - vertex 組成陣列
    - edge 以下述 node structure 表示 vi, vj, 指向下一個包含vi的node, 指向下一個包含vj的node
    - vertex[x] 指向首個相連的 dege 對應的 node structure
- Graph Traversal (DFS, BFS)
  - pseudo code
    - 簡易版

```
for ( int i=0 ; i < |V| ; i++ )
    visited[i] = false;

void DFS(vertex v)
{
    visited[v] = true;
    for ( all vertex va adjacent to v )
        if ( visited[va] == false )
            DFS(va);
}
```

```
void BSF(vertex v)
{
    visited[v] = true;
    Queue Q = new Queue;
    Q.enqueue(v);
    while ( Q.empty() == false )
    {
        V = Q.dequeue();
        for ( all vertex va adjacent to v )
            if ( visited[va] == false )
                {
                    visited[va] = true;
                    Q.enqueue(va);
                }
    }
}
```

      - O(v^2) if use Adjacency matrix
      - O(v+e) if use Adjacency list
    - 進階版

```
time = 0;   // global clock

// initialize, for all vertex
v.color=white;

void DFS(Graph G, vertex v)
{
    for ( all vertex in G )
        if ( v.color == white )
            DFS_visit(G, v);
}

void DFS_visit(Graph G, vertex v)
{
    v.time = ++time;
    v.color = gray;
    for ( all vertex u adjacent to v )
    {
        if ( u.color == white )
            DFS_visit(G, u);
    }
    v.color = black;
    v.f = ++time;   // finish time
}
```

```
// initialize, for all vertex
v.color=white; v.d=Inf; v.pi=NULL;

void BSF(vertex v)
{
    v.color = gray;
    v.d = 0;
    v.pi = NULL;
    Queue Q = new Queue;
    Q.enqueue(v);
    while ( Q.empty() == false )
    {
        u = Q.dequeue();
        for ( all vertex ua adjacent to u )
        {
            if ( ua.color == white )
            {
                ua.d = u.d + 1;
                ua.color = gray;
                ua.pi = u;
                Q.enqueue();
            }
        }
        u.color = black;
    }
}
```

      - O(v^2) if use Adjacency matrix
      - O(v+e) if use Adjacency list
  - 應用
    - 有無 cycle
    - 連通與否 追蹤完看 visited 或 color 就知道了
    - 求 spanning tree
- Articulation point 相關
  - 切點, biconnected 定義：略
  - How to find
    - 1. 畫表，找出所有頂點的 DFN 值
      - row 1: vertex number
      - row 2: DFN i.e. DFS number
      - row 3: low 值
    - 2. 畫出 DFS spanning tree, 標出後退邊
    - 3. 計算各頂點的 low 值 low(x) min{ DFN(y) }
      - 其中 y 為 x 之後代(包含 x)
      - 或者 x 之後代透過一條 back edge 連接的點
    - 4. 逐點判斷
      - root 子點 ≧ 兩個 則為切點
      - not root, vertex x low(w) ≧ DFN(x) w 為 x 之子點