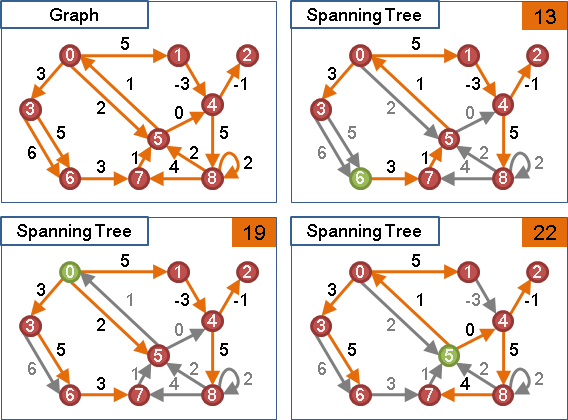
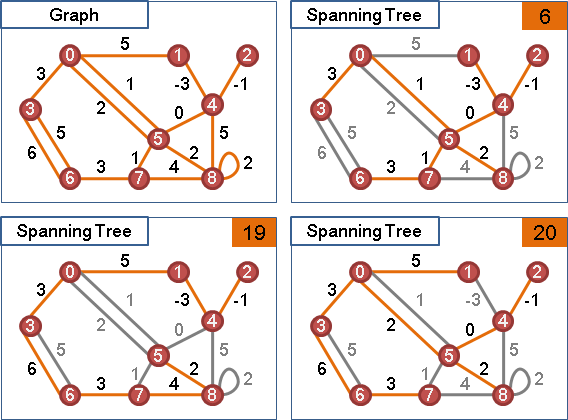
**Spanning Tree**

程度★　難度★

**Spanning Tree**

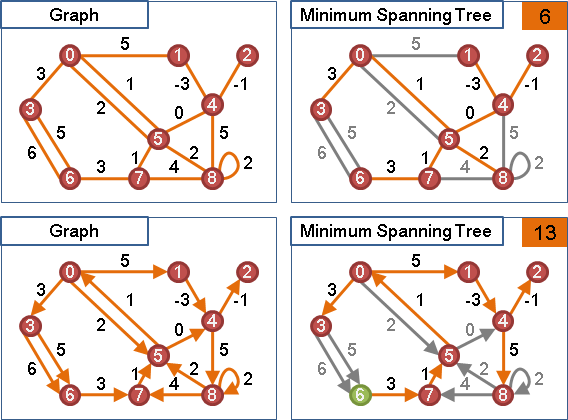
中譯「生成樹」，從一張圖上分離出一棵包含圖上所有點的樹，便是這張圖的生成樹。一張圖的生成樹可能會有很多種。

生成樹也可以有權重。當圖上每條邊都有權重時，生成樹的權重為樹上每條邊的權重總和。



**Minimum Spanning Tree （ MST ）**

中譯「最小生成樹」。權重最小的生成樹就是最小生成樹。一張圖的最小生成樹可能會有很多種。



無向圖的最小生成樹亦有人稱 Minimum Cost Spanning Tree ，中譯「最小成本生成樹」。有向圖的最小生成樹亦有人稱 Minimum Arborescence 、 Optimum Branchings ，中文網路上有人稱呼為「最小樹形圖」。

**Spanning Forest**

當一張圖是完全連通的時候，必然有生成樹。當一張圖有部份不連通的時候，則沒有生成樹，但還是會有許多棵「生成子樹」所構成的「生成森林」。就宛如 DFS tree 與 DFS forest 的關係一樣。

**延伸閱讀：經典生成樹問題**

Minimum (Cost) Spanning Tree [P]

權重最小的生成樹。

Minimum Bottleneck Spanning Tree [P]

權重最大的邊，使其權重最小的生成樹。

【註：此處Bottleneck定義為權重最大的邊。】

Minimum Diameter Spanning Tree [P]

直徑最短的生成樹。

Maximum Leaf Spanning Tree [NP-hard]

葉子最多的生成樹。

Minimum Degree Spanning Tree [NP-hard]

度數最多的點，使其度數最少的生成樹。

Minimum Routing (Cost) Spanning Tree [NP-hard]

所有兩點之間路徑，權重總和最小的生成樹。

Minimum Ratio Spanning Tree [NP-hard]

有兩種權重，分開計算。兩種權重比值最小的生成樹。

Minimum Edge-disjoint Spanning Trees [P]

邊不重疊，權重最小的k棵生成樹們。

Minimum Congestion Spanning Trees [P]

重疊的邊將額外增加權重，權重最小的k棵生成樹們。

Minimum k-Spanning Tree [NP-hard]

權重最小的生成子樹，生成子樹剛好是k個點。

Steiner Tree [NP-hard]

權重最小的生成子樹，生成子樹剛好是給定的k個點。

DFS Tree [P]

使用 Depth-first Search 找到的無向（有向）生成樹。

BFS Tree [P]

使用 Breadth-first Search 找到的無向（有向）生成樹。

Shortest Path Tree [P]

樹根到樹上各點都是最短路徑的無向（有向）生成樹。

Minimum Cut Tree [P]

任意兩點間路徑的瓶頸，形成兩點間最小割的無向生成樹。

Dominator Tree [P]

樹根到樹上各點的支配點，形成的有向生成樹。

**Minimum Spanning Tree:   
Prim's Algorithm**

程度★　難度★★

**用途**

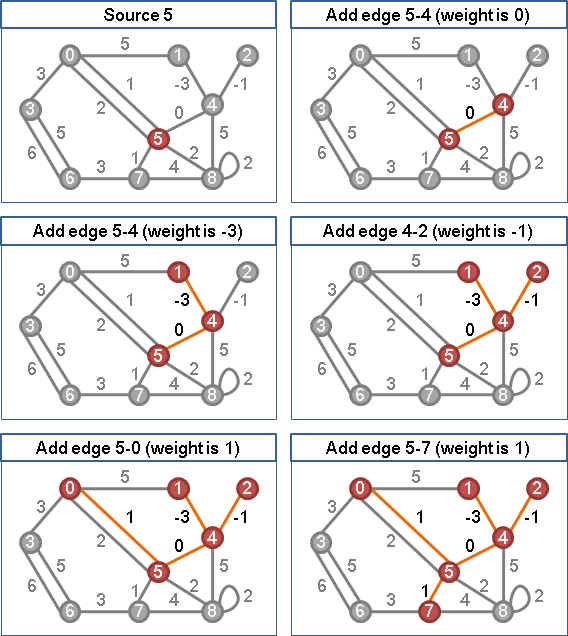
求出無向圖的其中一棵最小（大）生成樹。

**演算法**

和 Dijkstra's Algorithm 的概念相同，請參考「[Shortest Path: Dijkstra's Algorithm](http://www.csie.ntnu.edu.tw/~u91029/Path.html)」。

主要的差異是： Dijkstra's Algorithm 屢次找不在樹上、離根最近的點， Prim's Algorithm 屢次找不在樹上、離樹最近的點。

另外一個差異是：最短路徑樹會有特定起點，而最小生成樹可以選定任何一點作為樹根。



**時間複雜度**

圖的資料結構為 adjacency matrix 的話，便是 O(V^2) ；圖的資料結構為 adjacency lists 的話，還是 O(V^2) 。

找出一棵最小生成樹（adjacency matrix）

1. int w[9][9];    // 一張有權重的圖
2. int d[9];       // 紀錄目前的MST到圖上各點的距離。
3. int parent[9];  // 紀錄各個點在MST上的父親是誰
4. bool visit[9];  // 紀錄各個點是不是已在MST之中
6. void prim()
7. {
8. for (int i=0; i<9; i++) visit[i] = false;
9. for (int i=0; i<9; i++) d[i] = 1e9;
11. d[0] = 0;   // 可以選定任何點作為樹根，這裡以第零點作為樹根。
12. parent[0] = 0;
14. for (int i=0; i<9; i++)
15. {
16. int a = -1, b = -1, min = 1e9;
17. for (int j=0; j<9; j++)
18. if (!visit[j] && d[j] < min)
19. {
20. a = j;  // 記錄這一條邊
21. min = d[j];
22. }
24. if (a == -1) break; // 與起點相連通的MST都已找完
25. visit[a] = true;
26. //      d[a] = 0;           // 註解後，得到MST每條邊權重。
28. for (b=0; b<9; b++)
29. // 以下與Dijkstra's Algorithm略有不同
30. if  (!visit[b] && w[a][b] < d[b])
31. {
32. d[b] = w[a][b]; // 離樹最近，不是離根最近。
33. parent[b] = a;
34. }
35. }
36. }

**© 2010 [tkcn](http://tkcnandy.blogspot.com/). All rights reserved.**

11

9

17

17

11

15

2

3

15

3

7

0

1

2

3

4

5

6

7

8

9

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **節點編號** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** |
| 距離樹的距離 | 0 | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ |

Next Play Stop Reset

**UVa**[**10034**](http://uva.onlinejudge.org/external/100/10034.html)[**10147**](http://uva.onlinejudge.org/external/101/10147.html)[**10307**](http://uva.onlinejudge.org/external/103/10307.html)[**10397**](http://uva.onlinejudge.org/external/103/10397.html)[**10600**](http://uva.onlinejudge.org/external/106/10600.html)[**10842**](http://uva.onlinejudge.org/external/108/10842.html)

**Minimum Spanning Tree:   
Kruskal's Algorithm**

程度★　難度★★

**用途**

求出無向圖的其中一棵最小（大）生成樹。若是圖不連通，則是求出其中一叢最小（大）生成森林。

**演算法**

一、兩棵 MST ，要合併成一棵 MST 時，以兩棵 MST 之間權重最小的邊進行連結，當然會是最好的。

二、三棵 MST ，要合併成一棵 MST 時，先連結其中兩棵連結權重最小的 MST ，然後才連結第三棵，總是比較好。

三、一個單獨的點，可以視作一棵 MST 。

由以上三點，可以歸納出一個 greedy 演算法：以權重最小的邊連結各棵 MST ，一定比較好。

一、一開始圖上每一個點，各自是一棵最小生成子樹MSS。

二、圖上所有邊，依照權重大小，由小到大排序。

三、依序嘗試圖上所有邊，作為最小生成樹（森林）上的邊：

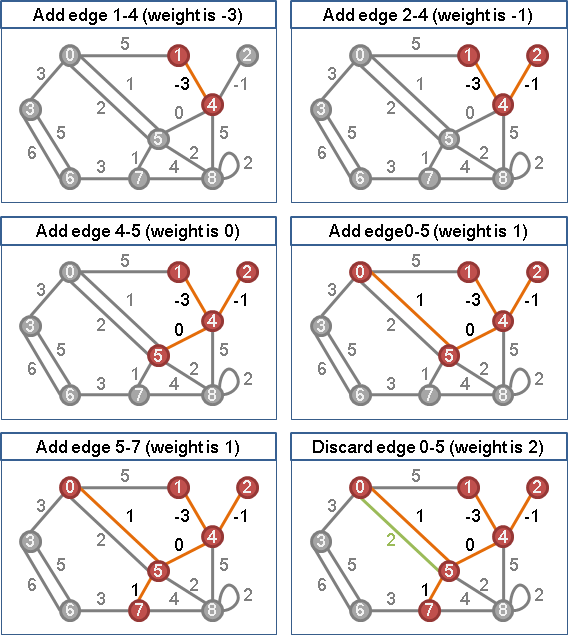
　甲、兩端點分別位於兩棵MSS，也就是產生了橋：

　　　用這條邊連接兩棵MSS，合併成一棵MSS。

　　　這條邊會是最小生成樹（森林）上的邊。

　乙、兩端點皆位於同一棵MSS，也就是產生了環：

　　　捨棄這條邊。



每次選中的邊，都是 MST 上的邊。沒有選中的邊，不論這張圖以後又增加了多少邊，絕不會成為 MST 上的邊。

**時間複雜度**

一、排序圖上所有邊，需時 O(ElogE) 。

二、連接 MSS ，一般是運用「[Disjoint-set Forest](http://www.csie.ntnu.edu.tw/~u91029/DisjointSets.html)」，需時 O(E\*α(E,V)) 。

故時間複雜度為 O(ElogE) 。

找出一棵最小生成樹（edge list）

1. const int V = 100, E = 1000;
3. // Edge List
4. struct Edge {int a, b, c;} edges[E];
5. bool operator<(const Edge& e1, const Edge& e2) {return e1.c < e2.c;}
7. // Disjoint-sets Forest
8. int p[V];
9. int init() {for (int i=0; i<V; ++i) p[i] = i;}
10. int find(int x) {return x == p[x] ? x : (p[x] = find(p[x]));}
11. void union(int x, int y) {p[find(x)] = find(y);}
13. void Kruskal()
14. {
15. init();
17. // 圖上所有邊，依照權重大小，由小到大排序。
18. sort(edges, edges+E);   // O(NlogN)
20. // 嘗試找出最小生成樹上的V-1條邊
21. int i, j;
22. for (i = 0, j = 0; i < V-1 && j < E; ++i)
23. {
24. // 同名參照，方便閱讀。
25. Edges& e = edges[j];
27. // 產生環，則捨棄。直到產生橋。
28. while (find(e.a) == find(e.b)) j++;
30. // 產生橋，則以此邊連接兩棵MSS。
31. union(e.a, e.b);
33. // 印出最小生成樹（森林）上的邊。
34. cout << "起點：" << edges[j].a
35. << "終點：" << edges[j].b
36. << "權重：" << edges[j].c;
38. j++;    // 別忘記累計索引值。也可以寫入迴圈。
39. }
41. if (i == V) cout << "得到最小生成樹";
42. else        cout << "得到最小生成森林";
43. }

迴圈的部份也可以寫成這樣。

找出一棵最小生成樹（edge list）

1. // 嘗試圖上所有邊，作為最小生成樹（森林）。
2. for (i = 0, j = 0; i < V-1 && j < E; ++j)
3. {
4. // 同名參照，方便閱讀。
5. Edges& e = edges[j];
7. // 產生環，則捨棄。
8. if (find(e.a) == find(e.b)) continue;
10. // 產生橋，則以此邊連接兩棵MSS。
11. union(e.a, e.b);
13. // 印出最小生成樹（森林）上的邊。
14. cout << "起點：" << e.a
15. << "終點：" << e.b
16. << "權重：" << e.c;
17. }

**UVa**[**908**](http://uva.onlinejudge.org/external/9/908.html)[**10369**](http://uva.onlinejudge.org/external/103/10369.html)[**10807**](http://uva.onlinejudge.org/external/108/10807.html)

**Minimum Spanning Tree:   
Borůvka's Algorithm**

程度★★　難度★★

**用途**

求出無向圖的其中一棵最小（大）生成樹。若是圖不連通，則是求出其中一叢最小（大）生成森林。

**演算法**

一、一開始圖上每一個點，各自是一棵最小生成子樹MSS。

二、重複以下步驟，直到形成最小生成樹（森林）：

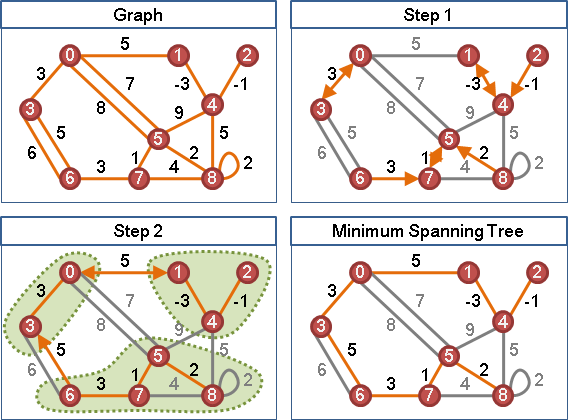
　甲、每棵MSS同時找權重最小、然後索引值最小的聯外邊。

　　　（即是MSS與MSS之間的邊，而不是MSS之內的邊。）

　　　用這些邊連接MSS們。

　口、聯外邊通常會重複，無妨。

　口、聯外邊不會形成環。



找權重最小的聯外邊，是為了得到最小生成樹；當權重一樣小，則再找索引值最小的聯外邊，是為了避免聯外邊形成環。亦可改用聯外邊的兩端點索引值。

證明很簡單：任一環上權重最大、索引值最大的邊，絕不會被選中，故無法形成環。

**時間複雜度：尋找權重最小、然後索引值的聯外邊**

所有邊掃描一次，以表格隨時紀錄每棵 MSS 聯外邊的權重最小值，時間複雜度為 O(V+E) 。

**時間複雜度： Disjoint Sets**

連接 MSS 們，一般是運用「[Disjoint-sets Forest](http://www.csie.ntnu.edu.tw/~u91029/DisjointSets.html)」。

由於 Disjoint-sets Forest 不斷更新至最佳狀態，所以 union 與 find 的均攤時間複雜度，會從 O(α(E,V)) 下降至 O(1) 。

一、初始化，Disjoint-sets Forest為最佳狀態！

二甲、E次find：

　　　每次find都是O(1)。

　　　Disjoint-sets Forest保持最佳狀態！

　乙、V次union：

　　　每次union都是O(1)。

　　　Disjoint-sets Forest只需修改V條邊，就能達到最佳狀態！

　丙、V次union當中所呼叫的V次find：

　　　Disjoint-sets Forest需要修改的V條邊，由每次find均攤。

　　　每次find均攤都是O(1)。

三甲、E次find：

　　　Disjoint-sets Forest尚待修改的V條邊，由每次find均攤。

　　　每次find均攤都是O(1)。

　　　因為每一個點都有find，

　　　所以Disjoint-sets Forest更新至最佳狀態！

　乙、V次union：

　　　同前。

　丙、V次union當中所呼叫的V次find：

　　　同前。

由此可知每次union與find的均攤時間複雜度為O(1)。

**時間複雜度**

所有邊掃描一次、連接 MSS 們，時間複雜度為 O(V+E) 。

每棵 MSS 相互連接，最差的情況是兩兩互接， MSS 總數量僅下降一半，所以運氣不好時需要 logV 個回合，故最差時間複雜度為 O((V+E)logV) ，可以簡單寫成 O(ElogV) 。

當圖上的邊為隨機分布時，平均只需要一至兩個回合，故平均時間複雜度為 O(V+E) 。

1. const int V = 100, E = 1000;
2. struct {int a, b, c;} edges[E];
3. int d[V];   // 各棵MSS的最小聯外邊的權重
4. int e[V];   // 各棵MSS的最小聯外邊的索引值
6. // Disjoint-sets Forest
7. int p[V];
8. int init() {for (int i=0; i<V; ++i) p[i] = i;}
9. int find(int x) {return x == p[x] ? x : (p[x] = find(p[x]));}
10. void union(int x, int y) {p[find(x)] = find(y);}
12. void Borůvka()
13. {
14. init();
16. while (true)
17. {
18. int cross\_edge = 0;
19. for (int i=0; i<V; ++i) d[i] = 1e9;
21. for (int i=0; i<E; ++i)
22. {
23. int a = find(edges[i].a);
24. int b = find(edges[i].b);
25. int c = edge[i].c;
26. if (a == b) continue;
27. cross\_edge++;
28. if (c < d[a] || c == d[a] && i < e[a])
29. d[a] = c, e[a] = i;
30. if (c < d[b] || c == d[b] && i < e[b])
31. d[b] = c, e[b] = i;
32. }
34. if (cross\_edge == 0) break;
36. for (int i=0; i<V; ++i)
37. if (d[i] != 1e9)
38. union(edges[e[i]].a, edges[e[i]].b);
39. }
40. }

**Directed Minimum Spanning Tree**

程度★★　難度★★

**前情提要**

直接套用無向圖的演算法，會發現邊的方向亂七八糟，無法形成有向樹。

在無向圖當中，兩棵 MST ，要合併成一棵 MST 時，以兩棵 MST 之間權重最小的邊進行連結，會是最好的。但是在有向圖當中，連接兩棵有向樹，不一定會形成有向樹。

**想法**

<http://www.ce.rit.edu/~sjyeec/dmst.html>

生成樹的基本概念是：連接圖上各點的樹。從這個概念下手，引用先前 Borůvka's Algorithm 的概念，然後考慮邊的方向性，就想到兩個粗糙的演算法：

有向圖上，每一個點，如果要被連接到，都要至少有一條出邊，除了樹葉以外。

每一個點，找權重最小的出邊，會比較好。

有向圖上，每一個點，如果要被連接到，都要剛好有一條入邊，除了樹根以外。

每一個點，找權重最小的入邊，會比較好。

入邊只需要用到一條，樹根也只有一個，所以從入邊下手是比較容易的。樹根是個例外；我們可以暫且假定我們已經知道最小生成樹的樹根是哪個點，就不必顧慮例外，事情就更好辦了。

**檢驗想法**

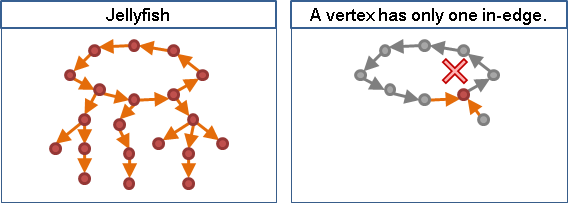
緊接著，我們要審視這個想法還有沒有例外。

運氣好的時候，每個點權重最小的入邊，剛好形成一棵生成樹，那麼這一定是最小生成樹。

運氣普通的時候，依照 Kruskal's Algorithm 的經驗，每個點權重最小的入邊，很有可能形成環。

**水母（沒有正式名稱，因為像水母就把它叫做水母）**

由於每個點僅有一條入邊，一旦入邊們形成環，此環一定只有多餘出邊，沒有多餘入邊──形成一個像是太陽、或者說是水母的圖。水母可以看作是很多棵樹，然後用一只環串起樹根。



**把水母改裝成最小生成樹**

每個點權重最小的入邊，一般狀況下可能會形成許多隻水母。最小生成樹不得有環，所以水母是不合格的。

水母是權重最小的連接方式，最小生成樹的權重一定是略高、等高於水母。如此便產生一個策略：嘗試拆除水母的某一條邊，並且更改為另一條邊。雖然很可能增加整體權重，但是也有機會成為最小生成樹了。

一、更改水母腳的邊：

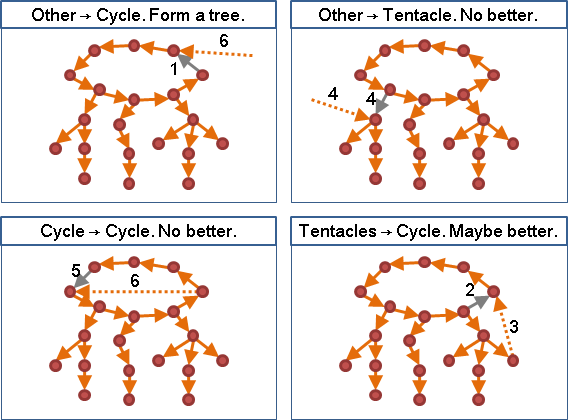
　　不但增加整體權重，而且水母環仍舊存在。之後沒有更好。

二、更改水母環的邊：

　甲、新邊是自身水母環的弦：形成一個更小的水母環。之後沒有更好。

　乙、新邊是由自身水母腳連來：形成一個更大的水母環。

　丙、新邊是由其他水母連來：自身水母環消失，變成其他水母的腳。



乙、原本水母環上的邊，之後仍可更改，所以先改後改都沒差，所以先行更改整體權重增加最少的邊，一定比較好。

丙、既然水母環會消失，更改整體權重增加最少的邊，顯然比較好。

結論：只需要更改水母環的邊，而且要讓整體權重增加最少。

**演算法：給定樹根的有向最小生成樹（ Chu-Liu/Edmonds Algorithm ）**

把進入水母環的邊，全部看過一遍，就能找到權重增加最少的新邊。另外，把看過的新邊，直接修改成權重增加量，並且收縮水母環；如此一來，只要是看過的新邊，就不用看第二遍了，可以降低時間複雜度。

一、刪去所有自己連向自己的邊。

二、移除樹根的全部入邊。

三、判斷樹根能不能連到圖上各個點，否則生成樹不存在。

四、重複以下步驟，直到形成生成樹為止：

　甲、每一個點，找出權重最小的入邊。O(E)

　乙、找出所有水母。如果沒有水母就表示目前已是最小生成樹。O(V)

　丙、調整進入水母環的邊的權重。O(E)

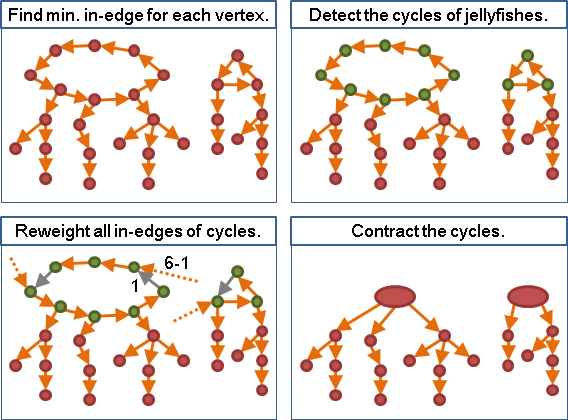
　　　w(a, x) -= w(å, x)，

　　　x是水母環上一點。

　　　åx是x點的最小入邊，也是水母環上的邊。

　　　ax為其他地方連入x點的邊。

　丁、收縮水母環成為一點。O(E)



**時間複雜度：給定樹根的有向最小生成樹**

最糟的情況是每個步驟中剛好產生一直水母環有兩個點的水母，水母環進行收縮後，整張圖只減少一個點。所以最多收縮 V-1 次水母環。每次皆以 Graph Traversal 找出每個點的最小入邊，因此整體的時間複雜度為 O(VE) 。

採用更困難的實作方式，時間複雜度還可以達到 O(V^2) 、 O(ElogV) 、 O(E+VlogV) 。實作方式是一直線地往回找入邊，每當形成環，就調整權重並縮環。粗略的時間複雜度分析如下：

一、每個點最多走一次。O(V)

二、最多縮環V-1次（兩點縮成一點），收縮之後出現的新點，最多走V-1次。O(V)

三、縮環時，環上的點會額外重覆走一次。由一與二可知道是O(2V)。

四、由一可知，每些點對應的入邊，最多都被掃過一次。O(E)

　　由二可知，每些點對應的入邊，最多都被掃過一次。O(E)

　　由三可知，也是一樣。O(2E)

五、縮環採用Disjoint-sets Forest，時間複雜度為O(α(E))。

　　由三可知，可以降低成常數。

**實作：給定樹根的有向最小生成樹**

O(VE) 的實作。

固定樹根：找出一棵最小生成樹＋計算最小生成樹權重（edge list）

1. int V, E;
2. struct Edge {int a, b, c;} edge[40000];
3. int d[1000], p[1000], v[1000], n[1000], m[1000];
4. // 每個點最小入邊的權重，每個點最小入邊的來源，
5. // 拜訪過，水母環，已收縮。
7. int MST(int r)
8. {
9. memset(m, 0, sizeof(m));
11. // 目前生成樹的權重，累計收縮水母環而失去的權重。
12. int w1 = 0, w2 = 0;
13. while (true)    // 一旦形成生成樹就停止。最多執行V-1次。
14. {
15. /\* O(E) graph traversal.
16. find minimum in-edge for each vertice.
18. --->o
19. \*/
20. memset(d, 1, sizeof(d));
21. memset(p, -1, sizeof(p));
23. for (int i=0; i<E; ++i)
24. {
25. int& a = edge[i].a;
26. int& b = edge[i].b;
27. int& c = edge[i].c;
28. if (a != b && b != r && c < d[b])
29. d[b] = c, p[b] = a;
30. }
32. /\* O(V) jellyfish detection
33. \_\_\_
34. /   \
35. \\_\_\_/
36. \_/|||\\_
37. //1\\
38. \*/
39. memset(v, -1, sizeof(v));
40. memset(n, -1, sizeof(n));
42. w1 = 0;
43. bool jf = false;
44. for (int i=0; i<V; ++i)
45. {
46. if (m[i]) continue;
47. if (p[i] == -1 && i != r) return 1e9;
48. if (p[i] >= 0) w1 += d[i];
50. // 找水母環
51. int s;
52. for (s = i; s != -1 && v[s] == -1; s = p[s])
53. v[s] = i;
55. // 標記水母環上的點，以及將會被收縮掉的點。
56. if (s != -1 && v[s] == i)
57. {
58. jf = true;
59. int j = s;
60. do
61. {
62. n[j] = s; m[j] = 1;
63. w2 += d[j]; j = p[j];
65. } while (j != s);
66. m[s] = 0;
67. }
68. }
69. if (!jf) break;
71. /\* O(E) edge reweighting and cycle contraction
72. \_\_\_
73. /   \ <-
74. \\_\_\_/
75. \*/
76. for (int i=0; i<E; ++i)
77. {
78. int& a = edge[i].a;
79. int& b = edge[i].b;
80. int& c = edge[i].c;
81. if (n[b] >= 0) c -= d[b];
82. if (n[a] >= 0) a = n[a];
83. if (n[b] >= 0) b = n[b];
84. if (a == b) edge[i--] = edge[--E];
85. }
86. }
87. return w1 + w2;
88. }

**UVa**[**11183**](http://uva.onlinejudge.org/external/111/11183.html)

**演算法：有向最小生成樹**

一、額外建立一個點，作為樹根。

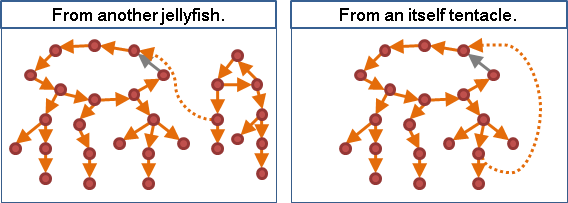
二、額外建立樹根到圖上各點的邊，權重設定為非常大的值。

三、求出給定樹根的最小生成樹。

　　如果用到兩條以上的新邊，則生成樹不存在。

**無向生成樹 v.s. 有向生成樹**

根據 Kruskal's Algorithm 提到的最小生成樹相連性質，可以知道連接多隻水母，就和連接多棵最小生成樹的道理是一樣的，以權重小的邊來連接是最好的。唯一不同的是， Kruskal's Algorithm 一旦發現造成環的邊，就直接捨棄； Chu-Liu/Edmonds Algorithm 則是留下造成環的邊（形成水母），並且嘗試各種打開環的方式：有時候增大水母環，有時候兩隻水母連接成為一隻水母。



**Minimum Bottleneck Spanning Tree**

程度★★　難度★

**註記**

一張圖上、一棵生成樹上、一條路徑上，權重最小的邊，稱作「瓶頸」。

然而，為了前後文連貫，此處將定義更改為權重最大的邊。古早人也是如此定義。

**Minimum Bottleneck Spanning Tree**

一張無向圖的所有生成樹當中，權重最大的邊（瓶頸），其權重最小的生成樹，稱作「最小瓶頸生成樹」，可能有許多棵。

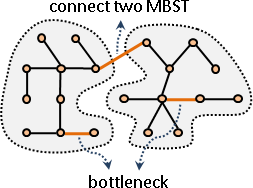
一個簡單的方式，是以最小生成樹 MST ，作為最小瓶頸生成樹 MBST 。

Kruskal's Algorithm 造就 MST 的最後一條邊，就是瓶頸。

**證明**

既然膽敢宣稱 MST 是 MBST ，那麼也許 MST 與 MBST 當中有些相近的性質。有時不妨率由舊章，以現有的 MST 性質，推定未知的 MBST 性質。大膽假設、小心求證，不然只怕是東施效顰。

MST 有著一個關鍵性質：以權重最小的邊，連接兩棵 MST ，可以構成一棵 MST 。依樣畫葫蘆， MBST 或許也有著一個關鍵性質：以權重最小的邊，連接兩棵 MBST ，可以構成一棵 MBST 。



此處用中文囉哩囉嗦證明之。若用數學式子，也許只消兩行：

甲、連接的邊，權重大於等於原本兩棵 MBST 的瓶頸權重，則會成為新樹的瓶頸。由於選擇了權重最小的邊當作連接的邊，連接的邊又是新樹的瓶頸，新樹的瓶頸權重當然也最小──新樹是一棵 MBST 。

乙、連接的邊，權重小於原本兩棵 MBST 的瓶頸權重，則不會成為新樹的瓶頸。新樹的瓶頸由原本兩棵 MBST 的瓶頸二選一，選權重大的那個成為新樹的瓶頸。因為原本兩棵 MBST 的瓶頸權重已經最小了，新樹的瓶頸權重當然也最小──新樹是一棵 MBST 。

新性質是正確的！由於 MST 和 MDST 都可以用權重最小的邊構造而得，因此在每一種 MST 演算法當中，每個步驟的 MST 也隨時是 MBST 。

儘管 MST 一定是 MBST ，但是小心 MBST 不見得是 MST 。儘管兩棵 MBST 以權重最小的邊相連，一定是一棵 MBST ，但是一棵 MBST 移除權重最大的邊，不見得是兩棵 MBST 。

**演算法**

事實上 MBST 有一個 O(V+E) 的演算法。

一、二分搜尋法，搜尋圖上所有邊的權重，找出MBST的瓶頸。

　　二分時，採用O(N)的中位數演算法。

二、每枚舉一個瓶頸，權重小於等於瓶頸的邊，皆可作為生成樹。

　甲、掃描一次，找出權重小於等於瓶頸的邊。

　乙、Graph Traversal，判斷圖上各點是否連通。

　　　若連通，則此瓶頸定可形成生成樹。反之則無法形成生成樹。

　丙、連通的點，合併為一點。以後就不需要重新遍歷了。

三、若要構造生成樹，在乙步驟，去掉形成環的邊（back edge）即可。

　　MST與MBST相異之處就在於：

　　MBST可以去掉環上任意一條邊，MST必須去掉環上權重最大的邊。

**最小生成樹 v.s. 最小割樹**

最小生成樹：任意兩點之間的路徑，最寬的邊盡量窄。

最小割樹：任意兩點之間的所有通道，最寬的切面盡量窄。

**UVa**[**11603**](http://uva.onlinejudge.org/external/116/11603.html)[**10816**](http://uva.onlinejudge.org/external/108/10816.html)**ICPC**[**4848**](http://livearchive.onlinejudge.org/external/48/4848.pdf)

**延伸閱讀： Minimum Bottleneck Path**

一張無向圖上，兩點之間的所有路徑當中，瓶頸權重最小的一條路徑，稱作「最小瓶頸路徑」，可能有許多條。

最小生成樹上的所有路徑，都是原圖的最小瓶頸路徑。證明方式同前，只是把生成樹改成了路徑。

如果需要所有兩點之間的最小瓶頸路徑的其中一個瓶頸，則可以使用 DP ：從長度為一的最小瓶頸路徑開始，逐步推導出更長的最小瓶頸路徑。 O(V^2) 時間建表、 O(1) 時間查詢。

亦可利用「[Lowest Common Ancestor](http://www.csie.ntnu.edu.tw/~u91029/LowestCommonAncestor.html)」。 O(VlogV) 時間建表、 O(logV) 時間查詢。

有向圖的情況，就請讀者自行研究了。最簡單的作法是修改最短路徑演算法。

**UVa**[**11354**](http://uva.onlinejudge.org/external/113/11354.html)

**延伸閱讀： Second-best Minimum Spanning Tree**

一張無向圖，權重最接近最小生成樹的另一棵生成樹，稱作「次小生成樹」。有可能與最小生成樹權重相等。

一、先求出一棵最小生成樹。

二、求出樹上所有兩點ij之間，權重最大的邊（瓶頸）。記為E(i,j)。

　　（恰好是所有兩點間最小瓶頸路徑。）

三、窮舉每一條不在最小生成樹上的邊pq：

　甲、把邊pq添加到最小生成樹上，勢必形成環。

　乙、然後拆除邊E(p,q)，勢必又形成樹，此樹權重已然盡量少。

　丙、記下此樹。

四、剛剛得到的E-(V-1)棵樹之中，權重最小者便是次小生成樹。

一與二各有數種演算法，時間複雜度也跟著改變。

**UVa**[**10462**](http://uva.onlinejudge.org/external/104/10462.html)

**Minimum Diameter Spanning Tree**

程度★★　難度★

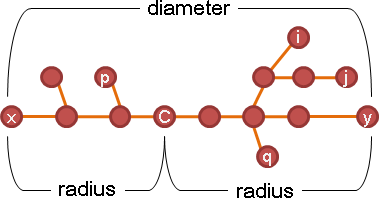
**最小直徑生成樹**

一張無向圖的所有生成樹當中，直徑最小的生成樹，可能有許多棵。

目前尚未有直接的演算法。目前是以絕對中心當作起點的最短路徑樹 SPT ，作為最小直徑生成樹 MDST 。關於絕對中心與最短路徑樹，可參考「[Central Vertex](http://www.csie.ntnu.edu.tw/~u91029/Path3.html)」。

**證明（ Hassin & Tamir, 1995 ）**

證明很簡單。在原論文中，證明過程可以寫成五行數學式子，閒來無事不妨朝聖一下。以下則是講得詳細一點：



甲、絕對中心的偏心距是最小的，位於 SPT 的直徑中央。半徑（偏心距）最短，所以直徑也是最短。把直徑拉成一直線來看，就清楚多了。

d(c, x) = d(c, y)

因為 d(c, x) 會盡量小，所以 2 \* d(c, x) = d(c, x) + d(c, y) 也會盡量小。

乙、絕對中心的 SPT 上面隨便一條路徑，都小於等於直徑長度。把路徑藉由絕對中心切成兩段，就清楚多了。

d(p, q)

= d(c, p) + d(c, q)

≤ d(c, x) + d(c, y) 切成兩條，分別ＰＫ。

d(i, j)

≤ d(c, i) + d(c, j) 切成兩條，分別ＰＫ。

≤ d(c, y) + d(c, y)

**最短路徑樹不見得是最小直徑生成樹**

各位也可以思考一下，為什麼絕對中心的 SPT 才是 MDST ，而一般中心的 SPT 並不見得是 MDST 。

G:

0

/|\

/ | 1

/ | \

4---3---2---5

MDST:

0

|

| 1

| \

4---3---2---5

SPT, source is 0:

0

/|\

/ | 1

/ | \

4 3 2---5

SPT, source is 1:

0

/|\

/ | 1

/ | \

4 3 2---5

SPT, source is 2:

0

\

1

\

4---3---2---5

SPT, source is 3:

0

|\

| 1

|

4---3---2---5

SPT, source is 4:

0

/ \

/ 1

/

4---3---2---5

SPT, source is 5:

0

\

1

\

4---3---2---5

可以看到MDST的直徑長度是3，而SPT的直徑長度都是4和5。

也就是說，一般中心的SPT不一定就是MDST。

**UVa**[**10805**](http://uva.onlinejudge.org/external/108/10805.html)**Timus**[**1569**](http://acm.timus.ru/problem.aspx?space=1&num=1569)**Sphere 1479**

**Minimum Diameter Minimum Cost Spanning Tree**

最小直徑最小成本生成樹。從所有最小生成樹當中，找到直徑最小者，是 NP-complete 問題。

至於從所有最小直徑生成樹中，找到權重最小者，筆者尚未找到相關文獻。

**其他 Spanning Tree**

程度★★★　難度★

**Minimum Steiner Tree**

一張無向圖上給定 k 個點，然後用圖上的邊連接這 k 個點，使得 k 個點相互連通，並且盡量減少這些邊的總權重。

為了減少權重，當然要盡量去除多餘的邊，所以這些邊一定沒有環，而是一棵樹。

連接給定點的子樹，稱作 Steiner Tree ， Steiner 是人名。注意到 Steiner Tree 並不是生成樹，只是概念上近似於最小生成樹。

求出權重最小的 Steiner Tree 是 NP-complete 問題。

特殊情況：

當k = 1時，Minimum Steiner Tree就是一個點。

當k = 2時，Minimum Steiner Tree就是此兩點間最短路徑。

當k = V時，Minimum Steiner Tree就是最小生成樹。

<http://www.prefield.com/algorithm/dp/steiner_tree.html>

**ICPC**[**3271**](http://livearchive.onlinejudge.org/external/32/3271.pdf)

**Minimum k-Spanning Tree**

k-Spanning Tree 是一棵剛好有 k 個點的生成子樹。求出權重最小的、剛好有 k 個點的生成子樹，是 NP-complete 問題。

**Degree-Constrained Minimum Spanning Tree**

每個點限制連接邊數上限的最小生成樹。是 NP-complete 問題。

當上限規定為兩條邊時，會成為 Hamilton Path 。

**UVa**[**10605**](http://uva.onlinejudge.org/external/106/10605.html)

**Minimum Ratio Spanning Tree**

求出一張圖的其中一棵最小（大）比率生成樹，手法等同於求最小比率環。時間複雜度等同於求 O(logR) 次最小生成樹。

一、設定一比率r後，把原圖轉換成新圖，除法轉換成差值。

二、新圖上一棵權重為零的生成樹，就是原圖上一棵比率為r的生成樹。

　　新圖上一只零環，就是原圖上一只比率為r的環。

三、當新圖上有一棵負權重的生成樹，表示這棵樹比率比r小：

　甲、比率設更小，設成r'之後，

　　　這棵樹就可以變成零權重生成樹，就是原圖上比率為r'的生成樹。

　　　找到了一棵比率更小的生成樹。

　乙、至於要找一棵負權重的生成樹，直接找最小生成樹就行了。

**ICPC**[**3465**](http://livearchive.onlinejudge.org/external/34/3465.pdf)[**4326**](http://livearchive.onlinejudge.org/external/43/4326.pdf)

**Minimum Edge-disjoint Spanning Trees**

A Note on Finding Minimum-Cost Edge-Disjoint Spanning Trees. James Roskind and Robert E. Tarjan. Mathematics of Operations Research, 1985, 10(4), 701-708.

**UVa**[**10807**](http://uva.onlinejudge.org/external/108/10807.html)

**Enumerate Spanning Trees**

程度★★★　難度★

時間複雜度 O(V+E+N) ，其中 N 是生成樹數目。

<http://www.ics.uci.edu/~eppstein/pubs/Epp-TR-95-50.pdf>

【待補文字】

**Count Spanning Trees**

程度★★★　難度★★

**Matrix Tree Theorem**

Laplacian matrix 的任意一個 cofactor ，其絕對值大小，就是各種生成樹的總數目。

cofactor 就是隨便砍掉某一行與某一列，剩下來的矩陣，然後加上係數 +1 或 -1 。

<http://en.wikipedia.org/wiki/Kirchhoff's_theorem>

證明會用到一個性質:

給定一個無向圖，兩點之間最多只有一邊。

如果原圖是樹，Laplacian matrix 隨便砍掉一橫行之後，絕對值都是1。

如果原圖不連通，Laplacian matrix 隨便砍掉一橫行之後，絕對值都是0。

－

Laplacian matrix 可以寫成 F \* transpose(F) 的形式，F是incidence matrix。

把F的隨便一個橫行給砍了，變成E，

然後用 Bxxx-Cxxx 展開 E \* transpose(E)，

展開之後會變成兩兩(V-1)x(V-1)方陣相乘，然後相加。

所有E取V-1的各種可能都會被展開出來。(還沒証，不要問，很可怕)

每一種可能就代表V-1條邊，有可能成為生成樹，有可能不行。

－

兩個(V-1)x(V-1)方陣，乘出來，剛好就是V個點的 Laplacian matrix 砍掉某一橫行。

如果是生成樹的話就會是1，所以統統加一加，就是生成樹數目。

1. int adj[9][9];      // adjacency matrix
2. int matrix[9][9];   // Laplacian matrix
4. int count\_spanning\_tree()
5. {
6. memset(matrix, 0, sizeof(matrix));
8. for (int i=0; i<9; ++i)
9. for (int j=0; j<9; ++j)
10. if (i != j && adj[i][j])
11. {
12. matrix[i][i]++;
13. matrix[i][j] = -1;
14. }
16. // 求 determinant
17. retutn det(9-1);
18. }

**UVa**[**10766**](http://uva.onlinejudge.org/external/107/10766.html)**Sphere 2670**

**Feedback Edge Set**

程度★★　難度★

**Feedback Edge Set**

刪除圖上的邊，使得圖上無環，所有刪除的邊稱作 Feedback Edge Set 。

無向圖無環，即是樹。 Minimum Feedback Edge Set 即是 Maximum Spanning Tree 以外的所有邊。

有向圖無環，即是有向無環圖 DAG 。 Minimum Feedback Edge Set 是 NP-hard 問題。

**Feedback Vertex Set**

無論無向圖還是有向圖，都是 NP-hard 問題。

[**演算法筆記**](http://www.csie.ntnu.edu.tw/~u91029/index.html)

[Spanning Tree](http://www.csie.ntnu.edu.tw/~u91029/SpanningTree.html#1)

[Minimum Spanning Tree:  
Prim's Algorithm](http://www.csie.ntnu.edu.tw/~u91029/SpanningTree.html#2)

[Minimum Spanning Tree:  
Kruskal's Algorithm](http://www.csie.ntnu.edu.tw/~u91029/SpanningTree.html#3)

[Minimum Spanning Tree:  
Borůvka's Algorithm](http://www.csie.ntnu.edu.tw/~u91029/SpanningTree.html#4)

[Directed Minimum Spanning Tree](http://www.csie.ntnu.edu.tw/~u91029/SpanningTree.html#5)

[Minimum Bottleneck Spanning Tree](http://www.csie.ntnu.edu.tw/~u91029/SpanningTree.html#6)

[Minimum Diameter Spanning Tree](http://www.csie.ntnu.edu.tw/~u91029/SpanningTree.html#7)

[其他Spanning Tree](http://www.csie.ntnu.edu.tw/~u91029/SpanningTree.html#8)

[Enumerate Spanning Trees](http://www.csie.ntnu.edu.tw/~u91029/SpanningTree.html#9)

[Count Spanning Trees](http://www.csie.ntnu.edu.tw/~u91029/SpanningTree.html#10)

[Feedback Edge Set](http://www.csie.ntnu.edu.tw/~u91029/SpanningTree.html#11)