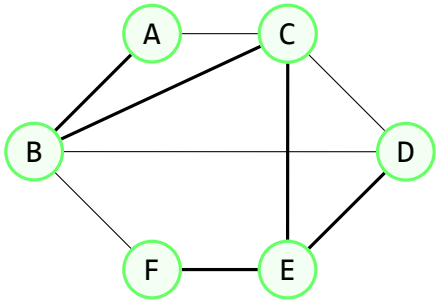
最小生成樹 (Minimum Spanning Tree) 1

1.1 生成樹（Spanning Tree）

給定一張連通圖 G，若 G 的子圖 T 是一棵樹，並且包含 G 的所有頂點，我們說 T 是 G 的生成樹。T 同時也是 G 最少邊數的子圖，使得所有頂點之間連通。T 理所當然會有所有一棵樹該有的性質，由於通常維護樹上資料比維護圖上資料結構簡單，處理一張圖的問題時，我們可能會以一棵生成樹代表之，如下圖所示。



Spanning Tree

注意若所有點不連通，則生成樹不會存在。

1.2 最小生成樹（MST, Minimum Spanning Tree）

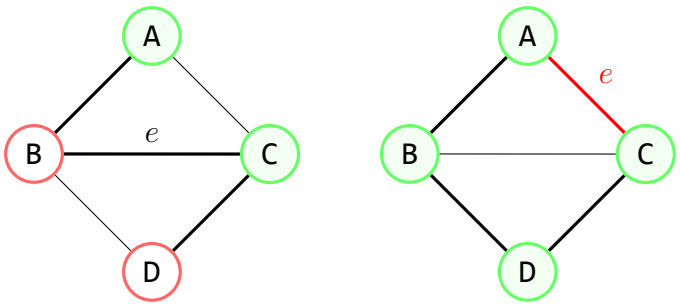
最小生成樹是最小權重生成樹的簡稱，也就是所有生成樹中邊權總和最小的。最小生成樹的形狀不一定唯一，但其邊權和是固定的。最小生成樹有以下一些性質，我們可以利用貪心法求出最小生成樹。

定理 1.2.1: Cut Property

將 G 的頂點集合分成兩個頂點集合 S、$V-S$，設連結兩個頂點集合的邊集為 E_{cut}，其中最小的邊為 e，則必定存在一個包含 e 的最小生成樹。

證明：設所有 MST 均不包含 e，在任一最小生成樹 T 中加上 e 後必會形成一環，除了 e 之外該環上至少有一條屬於 E_{cut} 的邊（否則 S、$V-S$ 不連通），我

們以 e 替換這條邊能夠得到權重不小於 T 且包含 e 的生成樹，假設矛盾。



Cut Property　　Cycle Property

定理 1.2.2: Cycle Property

對於每一個環 C 其上最大的邊 e，必定有不包含它的最小生成樹（也就是不選擇它不會影響 MST 的解答）。

證明：設所有 MST 都包含 e，去除了 e 之後它們都會變為兩棵子樹，而環上有另一不比 e 大的邊可以用來連接兩棵子樹，便構成了權重不小於原本的 MST，並且不包含 e 的生成樹，假設矛盾。

本章介紹的所有找最小生成樹的演算法都是屬於利用了 Cut Property 的 Greedy method。

1.3 Prim's Algorithm

概念

我們可以將 Cut Property 中的 S 視為執行到目前已經確定的 MST 點集，而不斷的以 E_{cut} 中最小的邊擴增 S 的大小，這是 Prim 的主要思想。Prim 和 Dijkstra 演算法的架構相當類似，我們以 V 代表原本的點集，E 代表原本的邊集 V_{new} 代表 MST 中的點集，E_{new} 代表 MST 中的邊集；以下是 Prim 的執行步驟：

1. 初始化：$V_{new} = \{x\}$，其中 x 為任一起始點，$E_{new} = \{\}$。
2. 重複下列操作，直到 $V_{new} = V$：
   a) 選取權值最小的邊 (u, v) 使得 $u \in V_{new}$，而 $v \notin V_{new}$（如果存在多條，則可任選），v 同時也可以說是距離目前的 MST 最近的頂點。
   b) 將 v 加入集合 V_{new} 中，將 (u, v) 加入集合 E_{new} 中。
3. V_{new} 和 E_{new} 即是最後的 MST！

證明

考慮 Cut Property，對於有 n 個頂點的圖 G_n，某個起點 x 其最近的鄰居若是 y，則邊 (x, y) 必定會屬於 MST，之後我們可以將 x, y 看成同一點 z，並以圖 G_{n-1} 代表此剩下 $n-1$ 個點的新圖（原圖連向 x 或 y 的邊均連到 z），我們利用數學歸納法能夠好好的確認 Prim 的正確性。

實作

從上面的流程中需要不斷的選取邊權最小的邊，在 V_{new} 加入新節點時又要不斷插入新邊權。這樣有效支援插入數字以及取最小值的資料結構，不難想到可以用priority_queue來幫助我們。由於概念都很簡單，最難的就是證明，所以我們先看程式碼吧！

```cpp
#include "bits/stdc++.h"
typedef pair<int,int> pii;
vector<pii> g[MAXV]; // adjacency list {weight, to}
int prim(int n){
    int sum=0, v=0LL; // 權重和、已選取頂點數
    bool inMST[MAXV]={};
    // heap 中的 pair 表示
    // {該頂點與MST的最短距離, 不在目前MST中的頂點編號}
    priority_queue<pii,vector<pii>,greater<pii> > pq;
    pq.push({0,0});
    while(v<n && pq.size()){
        pii cur=pq.top(); pq.pop();
        // 如果拿出來的最近頂點已經在MST中則跳過
        if(inMST[t.second]) continue;
        inMST[t.second]=true;
        sum+=t.first, v++;
        for(auto &e:g[t.second]) {
            if(!inMST[e.second]) pq.push(e);
        }
    }
    return sum;
}
```

由於我們至多存取 heap $|V|+|E|$ 次，Prim 演算法的總時間複雜度將會是 $O((|V|+|E|)\log|V|)$，如果用費波納契堆還能進一步優化到 $O(|E|+|V|\log|V|)$。

1.4 Kruskal's Algorithm

這個演算法較不複雜，應該是最常被使用的 MST 算法，可以好好看一下。

概念

Kruskal 演算法是以邊為主角，以下為 Kruskal 的流程：

1. 將所有邊 (u_i, v_i) 按照邊權sort
2. 初始化，將所有點視為獨立的連通塊
3. 由小到大檢查所有邊 (u, v)，若 u 與 v 互不連通，則將這條邊加入 MST 中，並合併 u, v 所在的連通塊，若相連通則略過。
4. 重複前一步驟，直到所有點都相連通。

證明

和 Prim 類似，Kruskal 每次會選取 G_n 中權重最小且連接不同連通塊的邊 (x, y)，此時能夠將 x 及 y 看成同一點，得到 G_{n-1}，利用數學歸納法同樣可以得到證明。

實作

上面的流程中提到要在新建的 MST 中，檢查任兩個點有沒有相連。當然最直覺的做法是每次都 DFS 看兩個點有沒有相連，但這個方法很明顯太慢了。然而你會發現，事實上我們在意的其實就是兩個點所屬的連通塊是否相同。我們可以想到用 Disjoint Sets 的資料結構維護，畢竟程式碼結構真的很簡單，所以直接看 code 吧！

```cpp
struct edge{
    int u,v,w;
};
bool operator<(edge a, edge b){return a.w<b.w;}
vector<edge> edges;
int pa[MAXV],sz[MAXV]; // 大家還記得 dsu 怎麼寫嗎？
void init(int n) {
    for(int i = 0; i < n; i++) pa[i] = i, sz[i] = 1;
}
int anc(int x){
    return x==pa[x] ? x : (pa[x]=anc(pa[x]));
}
bool same(int x,int y){
    return x=anc(x), y=anc(y), x==y;
}
void join(int x,int y){
    if((x=anc(x)) == (y=anc(y))) return;
    if(sz[x] < sz[y]) swap(x, y);
    pa[y] = x, sz[x] += sz[y];
}
int kruskal(int n){
    int CC = n, sum = 0; // 連通塊數、權重和
    init(n); // 初始化 dsu
    sort(edges.begin(),edges.end()); // 按邊權 sort
    for(auto &e:edges) { // 邊權由小到大檢查
        if(!same(e.u,e.v)) {//兩個點若不連通，則加入 MST
            join(e.u,e.v);
            sum += e.w;
        }
    }
    return sum;
}
```

Kruskal 主要的時間花費在排序邊 $O(|E|\log|E|)$，排序之後的合併只需要 $O(|E| \cdot \alpha(|E|, |V|)$ 即能完成，故 Kruskal 的總時間複雜度為 $O(|E|\log|E|)$。

1.5 Borůvka's Algorithm

又名 Sollin 演算法，它其實是最早被發明的 MST 多項式時間複雜度演算法，不過卻有點像是 Prim 和 Kruskal 的混合版，似乎很少人在競賽中使用這個演算法。

概念與實作

首先，一開始所有頂點都被設為是獨立的連通塊。對於每個連通塊，找出其連到其他連通塊的邊之中最短的邊（可以 $O(|E|)$ 掃過一遍），把所有連通塊對應的邊連上（$O(|V|)$），並對新的連通塊們重複執行這個步驟，直到只剩下一個連通塊。

證明

同樣由 Cut Property 可以知道每個連通塊 S 向外連的最短邊 e 一定屬於 MST，對點數強數歸能夠得知 Borůvka 的正確性。

```cpp
vector<edge> edges; // edge 同前面的宣告
int boruvka(int n){
    int CC = n, sum = 0; // 連通塊數、權重和
    edge cheapest[MAXV] = {}; // 連通塊向外連最短的邊
    init(n); // 使用並查集
    while(CC != 1){
        for(int i = 0; i < n; i++) cheapest[i].w = 1e9;
        for(auto &e:edges){
            int fu = anc(e.u), fv = anc(e.v);
            if(fu == fv) continue;
            // 找到每個連通塊往外最短的邊
            if(e < cheapest[fu]) cheapest[fu] = e;
            if(e < cheapest[fv]) cheapest[fv] = e;
        }
        for(int i = 0; i < n; i++) {
            if(i != anc(i)) continue;
            auto &e = cheapest[i];
            // 嘗試將每個連通塊以最短邊往外連
            if(!same(e.u,e.v)) {
                join(e.u,e.v);
                sum += e.w, --CC;
            }
        }
    }
    return sum;
}
```

可以注意到，每一輪操作中每一個連通塊都會和其他連通塊合併，也就是總連通塊數至少會減少為原先的一半，因此最多需要執行 $O(\log |V|)$ 輪合併操作，總複雜度 $O((|V|+|E|)\log |V|)$（森林結構維持在最佳狀態，DSU 操作的總時間複雜度，從 $O(|E|\cdot\alpha(|E|,|V|))$ 下降至 $O(|E|)$ —演算法筆記）。

據某些大陸人說，Borůvka 有時靈活性比 Prim 和 Kruskal 都好，因不須將頂點或邊直接比較，但筆者目前還沒找到必須要用 Borůvka 才能解決的題目；不過值得一提的是，利用 Borůvka 的想法能夠進一步得到隨機期望複雜度線性（$O(|V|+|E|)$）的最小生成樹做法，在此暫不贅述。

1.6 例題

習題的唷

習題 1.6.1: 圖論之最小生成樹（TIOJ 1211）

給你一個加權的無向圖 (weighted undirected graph)，請問這個圖的最小生成樹 (minimum spanning tree) 的權重和為多少？（$|V| \le 10^5, |E| \le 10^6, 1 \le w_i \le 1000$）

習題 1.6.2: 咕嚕咕嚕呱啦呱啦（TIOJ 1795）

給定 N 個點 M 條邊，以及所有邊的邊權重，是否有辦法建構出一顆生成樹之權重總和剛好為 K？另外，任意一條邊的權重只有可能為 0 or 1。（$N \le 10^5, M \le 3 \times 10^5$）

習題 1.6.3: 蓋捷運（OJ 71）

給定一張圖，每條邊上有兩個權值 X, Y，求所有生成樹 T 中下述比率的最大值。

$$\frac{\sum_{e \in T} e_X}{\sum_{e \in T} e_Y}$$

（$n, m \le 2 \times 10^5$，$1 \le x, y \le 10^9$）

習題 1.6.4: 機器人組裝大賽（TIOJ 1445）

給定一張圖，請輸出其最小生成樹的權重以及所有生成樹中權重和不嚴格第二小的權重和。（$|V| \le 1000, |E| \le \frac{|V|(|V|-1)}{2}, w_i \ge 0$，保證答案在`long long`內）