Graph Algorithm II

zhu

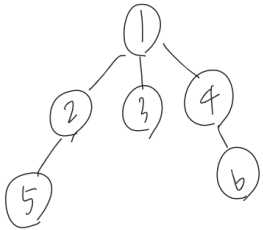
今天會教的東西

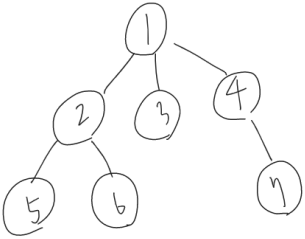
- 樹論
  1. 樹直徑
  2. 樹重心
  3. 樹 dp
- 並查集 DSU
- 最低共同祖先 LCA
- 最小生成樹 MST

樹論

熱身一下

找這棵樹上最長的路徑





樹直徑

甚麼是樹直徑？

- 樹上最長的那條路徑
- 可能會不只一條樹直徑

樹直徑實作

- 如果權重沒有負的
- 那我們可以找隨便一個點當起點，假設該點是 s

樹直徑實作

- 如果權重沒有負的
- 那我們可以找隨便一個點當起點，假設該點是 s
- 先 dfs 找到距離點 s 最遠的點 u

樹直徑實作

- 如果權重沒有負的
- 那我們可以找隨便一個點當起點，假設該點是 s
- 先 dfs 找到距離點 s 最遠的點 u
- 再以 u 為起點，dfs 找到距離 u 最遠的點 v
- u 和 v 之間的路徑，就是一條樹直徑

樹直徑實作

- 如果權重沒有負的
- 那我們可以找隨便一個點當起點，假設該點是 s
- 先 dfs 找到距離點 s 最遠的點 u
- 再以 u 為起點，dfs 找到距離 u 最遠的點 v
- u 和 v 之間的路徑，就是一條樹直徑
- 為甚麼我們可以確定距離樹上任一點最遠的點，必定是樹直徑的兩端？

樹直徑實作

- 如果權重沒有負的
- 那我們可以找隨便一個點當起點，假設該點是 s
- 先 dfs 找到距離點 s 最遠的點 u
- 再以 u 為起點，dfs 找到距離 u 最遠的點 v
- u 和 v 之間的路徑，就是一條樹直徑
- 為甚麼我們可以確定距離樹上任一點最遠的點，必定是樹直徑的兩端？
- 等一下會證 owo

樹直徑實作

- 那如果權重有負的怎麼辦？

樹直徑實作

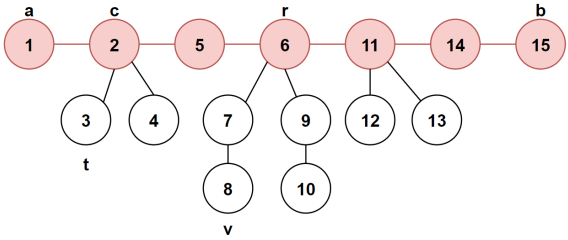
- 那如果權重有負的怎麼辦？
- dp（後面樹 dp 會講）

兩次 dfs 作法的證明

- 好欸那我們回來證明「為甚麼做兩次 dfs 會是對的 ?」

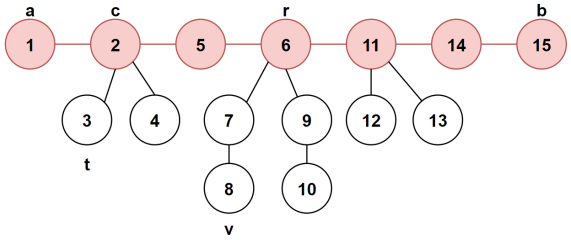
兩次 dfs 作法的證明

- 現在假設 a 到 b 之間的路徑是樹直徑
- $dis(v,a) \geq dis(v,t)$
  $\Longrightarrow (dis(v,a) - dis(v,c)) \geq$
  $(dis(v,t) - dis(v,c))$
  $\Longrightarrow dis(a,c) \geq dis(c,t)$



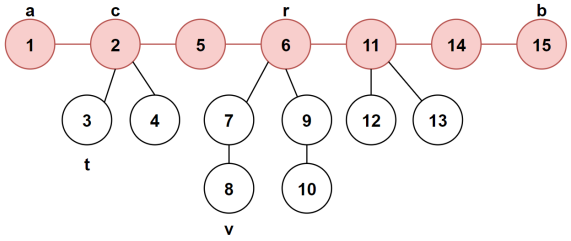
兩次 dfs 作法的證明

- 現在假設 a 到 b 之間的路徑是樹直徑
- $dis(v,a) \geq dis(v,t)$
  $\Longrightarrow (dis(v,a) - dis(v,c)) \geq$
  $(dis(v,t) - dis(v,c))$
  $\Longrightarrow dis(a,c) \geq dis(c,t)$
- 將位在樹直徑上左半邊的節點當作根
- 那它的子樹深度不會超過該節點到樹直徑最左端的距離。



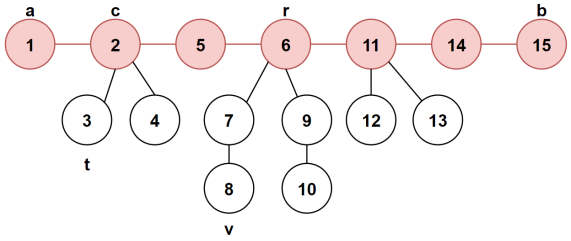
兩次 dfs 作法的證明

- 現在假設 a 到 b 之間的路徑是樹直徑
- $dis(v,a) \geq dis(v,t)$
  $\Longrightarrow (dis(v,a) - dis(v,c)) \geq$
  $(dis(v,t) - dis(v,c))$
  $\Longrightarrow dis(a,c) \geq dis(c,t)$
- 將位在樹直徑上左半邊的節點當作根
- 那它的子樹深度不會超過該節點到樹直徑最左端的距離。
- 同理，我們也可以推出右半邊也是如此



兩次 dfs 作法的證明

- 現在假設 a 到 b 之間的路徑是樹直徑
- $dis(v,a) \geq dis(v,t)$
  $\Longrightarrow (dis(v,a) - dis(v,c)) \geq$
  $(dis(v,t) - dis(v,c))$
  $\Longrightarrow dis(a,c) \geq dis(c,t)$
- 將位在樹直徑上左半邊的節點當作根
- 那它的子樹深度不會超過該節點到樹直徑最左端的距離。
- 同理，我們也可以推出右半邊也是如此
- 所以這邊的結論是：樹直徑上任一節點 x，其子樹深度都不會超過 $min(dis(x,a), dis(x,b))$

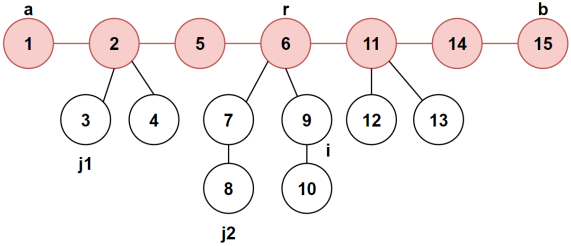


兩次 dfs 作法的證明

- 我們現在需要證明的是：為甚麼第一次 dfs 找到的點會是樹直徑的其中一端？
- Claim: 對於所有點 i，找到距離它最遠的點 j，則 j 必定滿足 $j = a$ or $j = b$

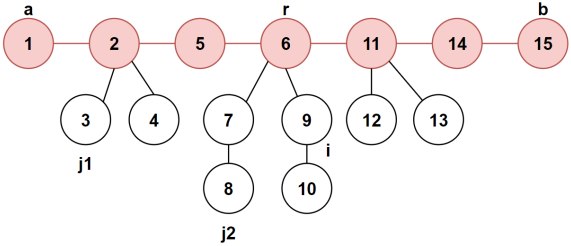
兩次 dfs 作法的證明

- 我們可以用剛剛證明出來的結論得出



兩次 dfs 作法的證明

- 我們可以用剛剛證明出來的結論得出
- if $j = j_1$,
  $dis(i, j_1) = dis(i, r) + dis(r, j_1) \leq dis(i, r) + dis(r, a) = dis(i, a)$
- if $j = j_2$,
  $dis(i, j_2) = dis(i, r) + dis(r, j_2) \leq dis(i, r) + dis(r, a) = dis(i, a)$



兩次 dfs 作法的證明

- 參考資料 Diameter of a tree and its applications

練習

CSES Tree Diameter

找樹直徑長度

Code

```cpp
#include<bits/stdc++.h>
#define fastio ios_base::sync_with_stdio(false);cin.tie(0)
    ;

using namespace std;

const int MAXN=2e5+1;

int n;
vector<int> g[MAXN];
int mx=-1,r=-1;

void dfs(int now,int p,int cnt){
    if(cnt>mx) mx=cnt,r=now;
    for(auto i:g[now]){
        if(i==p) continue;
        dfs(i,now,cnt+1);
    }
}

signed main(){
    fastio

    cin>>n;
    for(int i=0;i<n-1;++i){
        int u,v;cin>>u>>v;
        g[u].emplace_back(v);
        g[v].emplace_back(u);
    }
    dfs(1,1,0);
    int v=r;
    r=-1,mx=-1;
    dfs(v,v,0);
    cout<<mx<<'\n';
}
```

性質

- 樹圓心：樹直徑中間的那個點
- 樹圓心可能會有兩個
- 以樹圓心為根時，樹的深度會最小

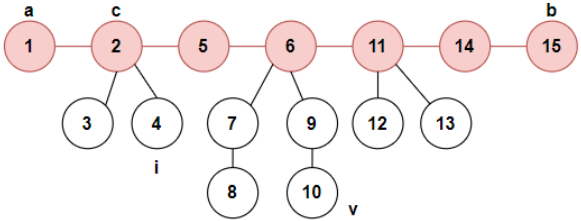
性質

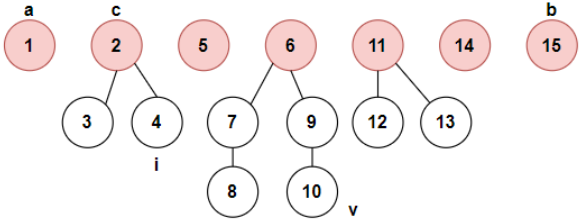
- 樹圓心：樹直徑中間的那個點
- 樹圓心可能會有兩個
- 以樹圓心為根時，樹的深度會最小
- 樹直徑可能不只一條，但都會通過樹圓心

性質

- 樹圓心：樹直徑中間的那個點
- 樹圓心可能會有兩個
- 以樹圓心為根時，樹的深度會最小
- 樹直徑可能不只一條，但都會通過樹圓心
- 把樹上樹直徑的邊拔掉，會得到森林

性質


a
c
b
1
2
5
6
11
14
15
3
4
7
9
12
13
i
8
10
v


a
c
b
1
2
5
6
11
14
15
3
4
7
9
12
13
i
8
10
v

喵喵喵喵喵喵喵喵喵

- 好耶知道了這些性質，我們可以來解決一些酷酷的題目

練習

CodeForces E. Sonya and Ice Cream

給你一棵樹，找 k 個連續的點，使得樹上其他點到該路徑的最大距離最小

練習

- 枚舉所有長度為 k 的路徑，然後慢慢 dfs

練習

- 枚舉所有長度為 k 的路徑，然後慢慢 dfs
- 這樣會 TLE 喔 QQ

練習

- 枚舉所有長度為 k 的路徑，然後慢慢 dfs
- 這樣會 TLE 喔 QQ
- 我們可以想想看，真的需要枚舉所有的路徑嗎？

練習

- 枚舉所有長度為 k 的路徑，然後慢慢 dfs
- 這樣會 TLE 喔 QQ
- 我們可以想想看，真的需要枚舉所有的路徑嗎？
- 分析一下，路徑會有三種可能性
- 第一種：跟樹直徑完全沒有重疊
- 第二種：跟樹直徑有小部分重疊
- 第三種：跟樹直徑完全重疊
- 有發現甚麼嗎？

練習

- 枚舉所有長度為 k 的路徑，然後慢慢 dfs
- 這樣會 TLE 喔 QQ
- 我們可以想想看，真的需要枚舉所有的路徑嗎？
- 分析一下，路徑會有三種可能性
- 第一種：跟樹直徑完全沒有重疊
- 第二種：跟樹直徑有小部分重疊
- 第三種：跟樹直徑完全重疊
- 有發現甚麼嗎？
- 最優解會是第三種可能性！

Practice

- CSES Tree Diameter
- TIOJ 1213 樹論之最遠距點對
- CodeForces D. Tree Tag
- Diameter of a tree and its applications 這篇 blog 上面放的題目

樹重心

甚麼是樹重心？

- 拔除後可以使所有連通塊的最大值最小的點
- 把它拔掉以後，每個連通塊大小都不超過 $N/2$
- 樹重心可能不只一個，但最多就兩個

樹重心實作

- 隨便找一個節點，把它當根
- DFS 去找每個點的子樹大小

樹重心實作

- 隨便找一個節點，把它當根
- DFS 去找每個點的子樹大小
- 把該點拔去以後，會產生兩個連通塊
- 分別是「子樹大小」和「節點數 - 子樹大小」

樹重心實作

- 隨便找一個節點，把它當根
- DFS 去找每個點的子樹大小
- 把該點拔去以後，會產生兩個連通塊
- 分別是「子樹大小」和「節點數 - 子樹大小」
- 根據前面的定義，如果大小都不超過 $N/2$，那它就是樹重心（之一）

練習

NEOJ 293 樹重心

給一棵樹，找樹重心，若有多個，那就輸出編號最小的

Code

```cpp
#include<bits/stdc++.h>
#define fastio ios_base::sync_with_stdio(false);cin.tie(0);

using namespace std;

const int MAXN=1e5+1;

vector<int> g[MAXN];
int ans,n;

int dfs(int now,int p){
    int cnt=0,b=1;
    for(auto& i:g[now]){
        if(i==p) continue;
        int r=dfs(i,now);
        if(r>n/2) b=0;
        cnt+=r;
    }
    if((n-1-cnt)>n/2) b=0;
    if(b) ans=min(ans,now);
    return cnt+1;
}

signed main(){
    fastio

    int t;cin>>t;
    while(t--){
        cin>>n;
        fill(g,g+MAXN,vector<int>());
        for(int i=0;i<n-1;++i){
            int u,v;cin>>u>>v;
            g[u].emplace_back(v);
            g[v].emplace_back(u);
        }
        ans=MAXN;
        dfs(0,0);
        cout<<ans<<'\n';
    }
}
```

Practice

- CodeForces C. Link Cut Centroids
- CodeForces F. Santa Clauses and a Soccer Championship

樹 dp

樹 dp

- 就是在樹上做 dp

帶負權樹直徑

- 現在來講剛剛的樹直徑，不過它帶負權
- 因為它帶負權，所以它的路徑不會一直都是越走越長
- 可以用動態規劃來解決它

帶負權樹直徑

- 首先，先訂狀態
- 令 $dpf[i] =$ 在 i 子樹中的最長路徑，$dps[i] =$ 在 i 子樹中的次長路徑
- 那答案就是 $max\{dpf[i] + dps[i]\}$

練習

CSES Tree Diameter

找樹直徑長度

Code

```cpp
#include <bits/stdc++.h>
#define fastio ios_base::sync_with_stdio(0); cin.tie(0);
       cout.tie(0);

using namespace std;

const int MAXN=2e5+1;
vector<int> g[MAXN];
int dp[MAXN], ans;

void dfs(int u, int p){
    int mx=0,smx=0;
    for(auto& i:g[u]){
        if(i==p) continue;
        dfs(i,u);
        if(dp[i]>mx) smx=mx,mx=dp[i];
        else if(dp[i]>smx) smx=dp[i];
    }
    dp[u]=mx+1;
    ans=max(ans,mx+smx);
}

signed main(){
    fastio
    int n;cin>>n;

    for(int i=1;i<=n-1;i++){
        int u,v;cin>>u>>v;
        g[u].emplace_back(v);
        g[v].emplace_back(u);
    }

    dfs(1,1);

    cout<<ans<<'\n';
}
```

喵喵喵喵喵喵喵

- 接下來來講換根 dp
- 我們可以直接先看例題喵

換根 dp 例題

CodeForces F. Tree with Maximum Cost

每個點有一個值，令他為 a_i。
定義以 c 為根的樹的花費為 $\sum_{i=1}^{n} dist(i, c) \cdot a_i$。
問最大花費會是多少

換根 dp 例題

- 最直觀的想法大概就是對每個點都做一次 dp

換根 dp 例題

- 最直觀的想法大概就是對每個點都做一次 dp
- 可是這樣會 TLE

換根 dp 例題

- 最直觀的想法大概就是對每個點都做一次 dp
- 可是這樣會 TLE
- 這個時候我們可以用換根 dp

換根 dp 例題

- dp[i] 表示以 i 為根節點的子樹花費

換根 dp 例題

- dp[i] 表示以 i 為根節點的子樹花費
- 假設我們已經處理完以 v 為根的樹了，並且令它的花費為 c，現在根要換成 u

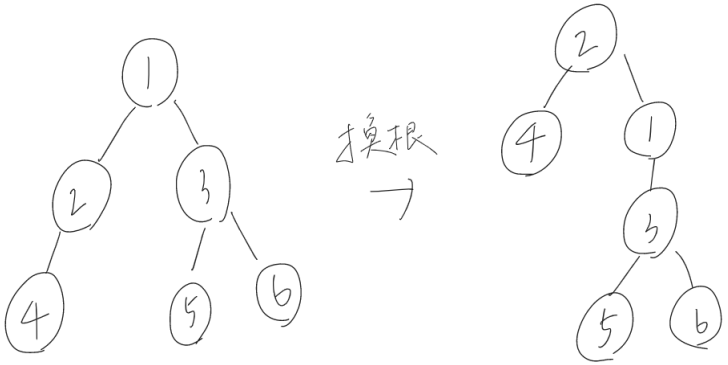
換根 dp 例題

- dp[i] 表示以 i 為根節點的子樹花費
- 假設我們已經處理完以 v 為根的樹了，並且令它的花費為 c，現在根要換成 u
- 要修改的東西有哪些？

換根 dp 例題

- dp[i] 表示以 i 為根節點的子樹花費
- 假設我們已經處理完以 v 為根的樹了，並且令它的花費為 c，現在根要換成 u
- 要修改的東西有哪些？
- 可以先看一下這張圖

換根 dp 例題



換根 dp 例題

- 以這張圖為例，當根節點變為 2 時，有些點的深度變深，有些點的深度變小
- 深度變小 $\rightarrow$ 扣除、深度變深 $\rightarrow$ 增加
- 深度變小的：節點 4
- 深度變深的：節點 $1, 3, 5, 6$

換根 dp 例題

- 以這張圖為例，當根節點變為 2 時，有些點的深度變深，有些點的深度變小
- 深度變小 $\rightarrow$ 扣除、深度變深 $\rightarrow$ 增加
- 深度變小的：節點 4
- 深度變深的：節點 $1, 3, 5, 6$
- 需要扣除的：a_4
- 需要增加的：a_1, a_3, a_5, a_6
- 需要更新的：$dp[1]\text{-}=dp[2]$、$dp[2]+=dp[1]$

換根 dp 例題

- 以這張圖為例，當根節點變為 2 時，有些點的深度變深，有些點的深度變小
- 深度變小 $\rightarrow$ 扣除、深度變深 $\rightarrow$ 增加
- 深度變小的：節點 4
- 深度變深的：節點 $1, 3, 5, 6$
- 需要扣除的：a_4
- 需要增加的：a_1, a_3, a_5, a_6
- 需要更新的：$dp[1]\text{-}=dp[2]$、$dp[2]+=dp[1]$
- 最後記得要把扣掉或增加的東西還原

並查集

甚麼是並查集？

- 並查集，又叫 DSU（Disjoint Set Union-find）
- 一種資料結構，可以維護一些集合，支援**合併**和**查詢**兩種操作

操作

- 每個集合都是一棵樹

操作

- 每個集合都是一棵樹
- 最初每個點都是一個集合，且自己就是自己的根節點的父節點

操作

- 每個集合都是一棵樹
- 最初每個點都是一個集合，且自己就是自己的根節點的父節點
- 要查詢兩個點是否位於同一個集合，就看他們的祖先是不是同一個

操作

- 每個集合都是一棵樹
- 最初每個點都是一個集合，且自己就是自己的根節點的父節點
- 要查詢兩個點是否位於同一個集合，就看他們的祖先是不是同一個
- 要把兩個集合合併，就讓一個集合的根節點的父節點變成另一個集合的根節點

Code

```
vector<int> dsu,sz;

void init(){
    dsu.resize(n+1);
    for(int i=1;i<=n;++i) dsu[i]=i;
}

int findDSU(int a){
    if(dsu[a]==a) return dsu[a];
    findDSU(dsu[a]);
}

void unionDSU(int a,int b){
    a=findDSU(a);
    b=findDSU(b);
    if(a==b) return;
    dsu[b]=a;
}
```

優化

- 你會發現合併以後可能是一條鏈

優化

- 你會發現合併以後可能是一條鏈
- 如果是一條鍊，那麼每次向上找祖先時，複雜度會是 $O(n)$

優化

- 你會發現合併以後可能是一條鏈
- 如果是一條鍊，那麼每次向上找祖先時，複雜度會是 $O(n)$
- 所以我們的目標是：
  不讓他找祖先時需要走那麼多步，又或者是說，乾脆不要讓他有機會形成一條鏈

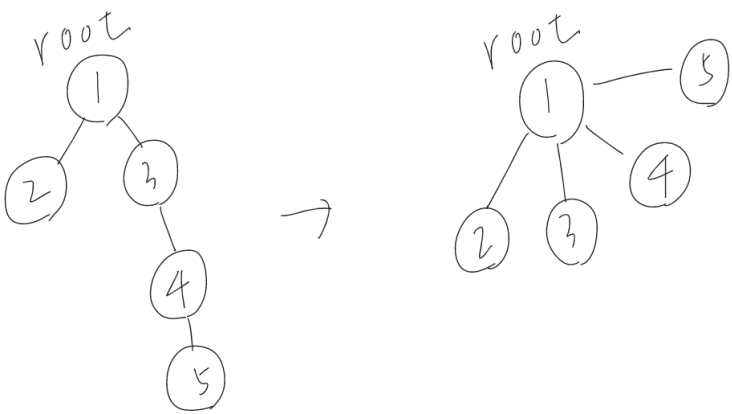
優化 - 路徑壓縮

- 就是把路變短

優化 - 路徑壓縮

- 就是把路變短
- 讓每個點往上走一步就會到根節點
- 在遞迴（findDSU）的時候做掉一整條
- 這樣複雜度會變 $O(\log n)$

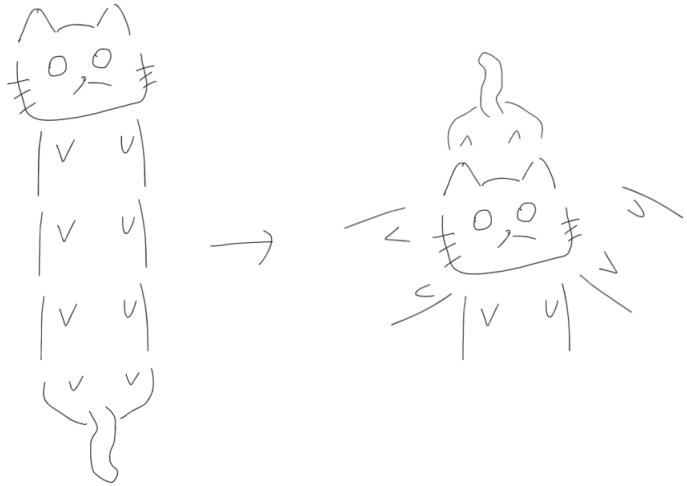
路徑壓縮



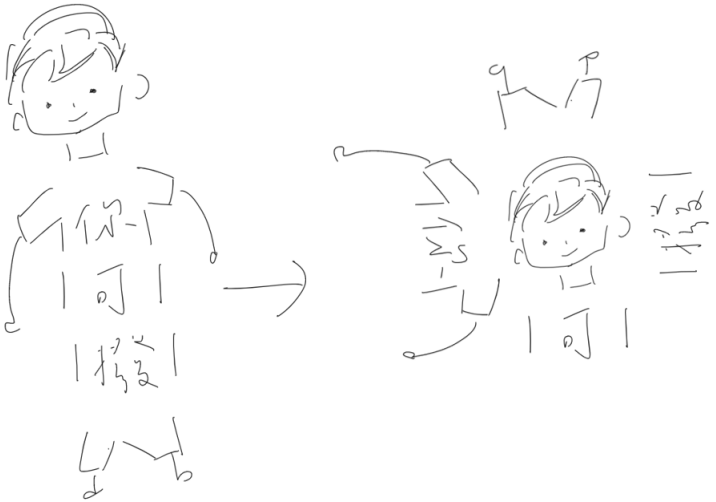
Code

```
vector<int> dsu,sz;

void init(){
    dsu.resize(n+1);
    for(int i=1;i<=n;++i) dsu[i]=i;
}

int findDSU(int a){
    if(dsu[a]!=a) dsu[a]=findDSU(dsu[a]);
    return dsu[a];
}

void unionDSU(int a,int b){
    a=findDSU(a);
    b=findDSU(b);
    if(a==b) return;
    dsu[b]=a;
}
```

補充-路徑壓縮（X



補充-路徑壓縮（X



優化 - 啟發式合併

- 除了把路變短，我們可以一開始就讓路不要那麼長

優化 - 啟發式合併

- 除了把路變短，我們可以一開始就讓路不要那麼長
- 在合併的時候將較大集合的根節點當作小集合的根節點的父節點

優化 - 啟發式合併

- 除了把路變短，我們可以一開始就讓路不要那麼長
- 在合併的時候將較大集合的根節點當作小集合的根節點的父節點
- 這樣每往上走一個點，子樹大小至少會變兩倍

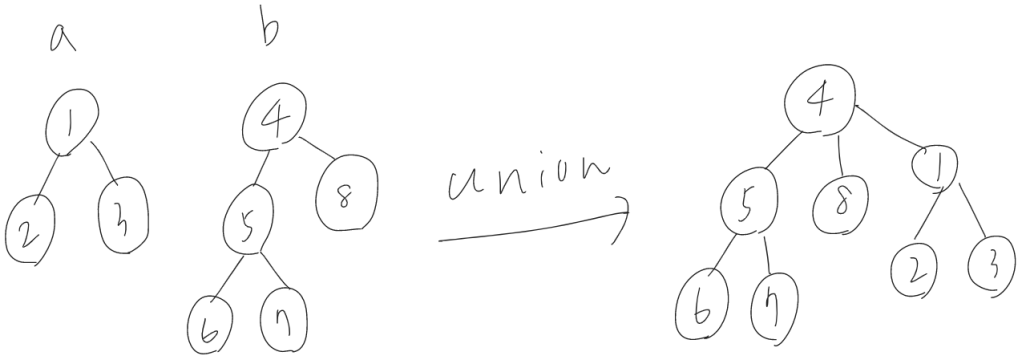
優化 - 啟發式合併

- 除了把路變短，我們可以一開始就讓路不要那麼長
- 在合併的時候將較大集合的根節點當作小集合的根節點的父節點
- 這樣每往上走一個點，子樹大小至少會變兩倍
- 因此樹的深度最多會是 $O(\log n)$
- 換句話說，最多走 $O(\log n)$ 步就會抵達根節點

優化 - 啟發式合併

- 除了把路變短，我們可以一開始就讓路不要那麼長
- 在合併的時候將較大集合的根節點當作小集合的根節點的父節點
- 這樣每往上走一個點，子樹大小至少會變兩倍
- 因此樹的深度最多會是 $O(\log n)$
- 換句話說，最多走 $O(\log n)$ 步就會抵達根節點
- 實作的話多維護一個集合大小就好

啟發式合併



$size[a] < size[b]$

Code

```
1 void init(){
2     dsu.resize(n+1);
3     for(int i=1;i<=n;++i) dsu[i]=i;
4 }
5
6 int findDSU(int a){
7     if(dsu[a]==a) return dsu[a];
8     findDSU(dsu[a]);
9 }
10
11 void unionDSU(int a,int b){
12     a=findDSU(a);
13     b=findDSU(b);
14     if(sz[a]<sz[b]) swap(a,b);
15     dsu[b]=a;
16     sz[a]+=sz[b];
17 }
```

優化

- 好耶好快
- 還可以更快 !!!

優化

- 好耶好快
- 還可以更快 !!!
- 兩個優化一起用！

優化

- 好耶好快
- 還可以更快 !!!
- 兩個優化一起用！
- 兩個一起用的時候複雜度是 $O(\alpha(n))$
- α 是一個成長率超級無敵霹靂小的函數
- 非常接近常數

Code

```
1 void init(){
2     dsu.resize(n+1);
3     for(int i=1;i<=n;++i) dsu[i]=i;
4 }
5
6 int findDSU(int a){
7     if(dsu[a]!=a) dsu[a]=findDSU(dsu[a]);
8     return dsu[a];
9 }
10
11 void unionDSU(int a,int b){
12     a=findDSU(a);
13     b=findDSU(b);
14     if(sz[a]<sz[b]) swap(a,b);
15     dsu[b]=a;
16     sz[a]+=sz[b];
17 }
```

優化

- 但並不是所有題目都可以用以上兩種優化
- 因為這兩種優化會改到樹本身的結構

練習

CodeForces D. The Number of Imposters

imposter 只會說謊，crewmate 只會說真話。
給 m 個線索，例如：a 說 b 是 crewmate。
在符合這些關係的前提下，輸出最多會有幾個 imposter，若他給的線索已經造成矛盾，那就輸出 -1。

應用 - DSU 判二分圖

- 轉換一下題目

應用 - DSU 判二分圖

- 轉換一下題目
- 給你一張圖，然後判斷它是不是二分圖

應用 - DSU 判二分圖

- 轉換一下題目
- 給你一張圖，然後判斷它是不是二分圖
- 就 DFS 塗顏色就好了 R

應用 - DSU 判二分圖

- 轉換一下題目
- 給你一張圖，然後判斷它是不是二分圖
- 就 DFS 塗顏色就好了 R
- 那我們把題目改一下
- 在每一條邊被加入的時候，都判斷它是不是二分圖

應用 - DSU 判二分圖

- 在每一條邊被加入的時候，都判斷它是不是二分圖

應用 - DSU 判二分圖

- 在每一條邊被加入的時候，都判斷它是不是二分圖
- 把每個節點都複製一次，一個綁上標記 1，另一個綁上標記 0
- 1 代表這個節點要塗黑色，0 則表示要塗白色

應用 - DSU 判二分圖

- 在每一條邊被加入的時候，都判斷它是不是二分圖
- 把每個節點都複製一次，一個綁上標記 1，另一個綁上標記 0
- 1 代表這個節點要塗黑色，0 則表示要塗白色
- 如果加進某條邊會產生矛盾，那就代表不是二分圖

應用 - DSU 判二分圖

- 在每一條邊被加入的時候，都判斷它是不是二分圖
- 把每個節點都複製一次，一個綁上標記 1，另一個綁上標記 0
- 1 代表這個節點要塗黑色，0 則表示要塗白色
- 如果加進某條邊會產生矛盾，那就代表不是二分圖
- 矛盾？
- 會位在同一個連通塊表示他們之間有某種推導關係
- 所以可以檢查，如果 $\{v, 0\}$ 和 $\{v, 1\}$ 位在同一個連通塊上，就代表產生矛盾了

Practice

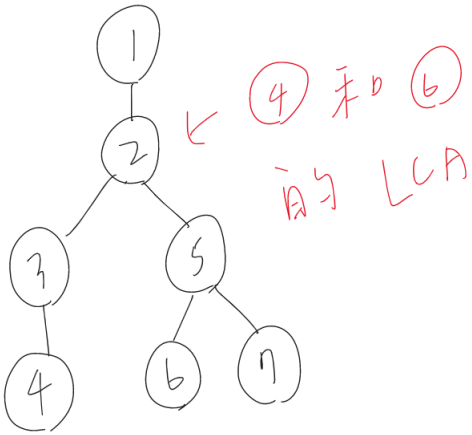
- CodeForces DSU edu
- Zerojudge 為什麼你們都喜歡撞來撞去的？
- APCS 2021/11 p4 真假子圖
- CodeForces G. MinOr Tree
- Codeforces 731D Vessels
- CodeForces D. Restructuring Company
- CodeForces D. Mr. Kitayuta's Colorful Graph
- CodeForces F. Vicky's Delivery Service
- SPOJ Strange Food Chain

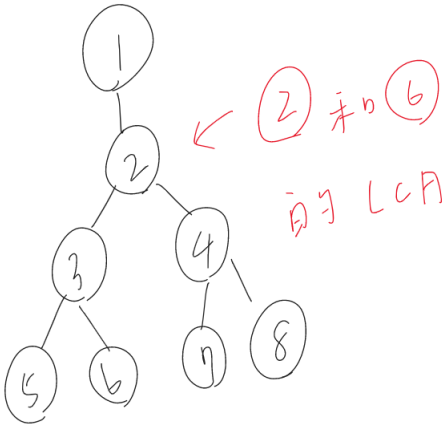
最低共同祖先

甚麼是最低共同祖先？

- 最低共同祖先，又叫 LCA（Lowest Common Ancestor）
- 找 a 和 b 的 LCA，如果 a 是 b 的祖先，那 a 就是他們的 LCA
- 否則就是找他們兩個的祖先中，深度最深的那個點

甚麼是最低共同祖先？





最低共同祖先

- 很簡單 R
- 就一步一步走嘛
- 樹上走路誰不會 R

最低共同祖先

- 很簡單 R
- 就一步一步走嘛
- 樹上走路誰不會 R
- 對 R... 好水喔　然後你就 TLE 了

最低共同祖先

- 很簡單 R
- 就一步一步走嘛
- 樹上走路誰不會 R
- 對 R... 好水喔　然後你就 TLE 了
- 所以你不能用走的，要用跳的
- 有一個東西叫**倍增法**

倍增法

- 假設我們今天要找 a 和 b 的 LCA，而且兩點之間沒有祖孫關係
- 把 a 往上跳
- 我們可以開一個陣列 $p[i][v]$，代表從點 v 往上跳 2^i 條邊會到的節點
- 轉移式：$p[i][v] = p[i-1][p[i-1][v]]$
- 我們可以枚舉 $i = \lfloor \log_2 n \rfloor \sim 0$，看從 a 往上跳 2^i 格會跳到哪裡
- 如果 a 跳到的地方是 b 的祖先，就不要跳，否則就往上跳 2^i 格
- 然後 LCA 就是 $p[0][a]$

倍增法

- 假設我們今天要找 a 和 b 的 LCA，而且兩點之間沒有祖孫關係
- 把 a 往上跳
- 我們可以開一個陣列 $p[i][v]$，代表從點 v 往上跳 2^i 條邊會到的節點
- 轉移式：$p[i][v] = p[i-1][p[i-1][v]]$
- 我們可以枚舉 $i = \lfloor \log_2 n \rfloor \sim 0$，看從 a 往上跳 2^i 格會跳到哪裡
- 如果 a 跳到的地方是 b 的祖先，就不要跳，否則就往上跳 2^i 格
- 然後 LCA 就是 $p[0][a]$
- 預處理 $O(n \log n)$、單筆詢問 $O(\log n)$

倍增法

- 假設我們今天要找 a 和 b 的 LCA，而且兩點之間沒有祖孫關係
- 把 a 往上跳
- 我們可以開一個陣列 $p[i][v]$，代表從點 v 往上跳 2^i 條邊會到的節點
- 轉移式：$p[i][v] = p[i-1][p[i-1][v]]$
- 我們可以枚舉 $i = \lfloor \log_2 n \rfloor \sim 0$，看從 a 往上跳 2^i 格會跳到哪裡
- 如果 a 跳到的地方是 b 的祖先，就不要跳，否則就往上跳 2^i 格
- 然後 LCA 就是 $p[0][a]$
- 預處理 $O(n \log n)$、單筆詢問 $O(\log n)$
- 好像直接看 code 會比較好理解

倍增法

- 有另一種寫法

倍增法

- 有另一種寫法
- 找 a 和 b 的 LCA
- 我們先把 a 和 b 拉到同一個高度
- 讓 a 和 b 同時一起往上跳
- 讓他們靠近但不會碰在一起

倍增法

- 有另一種寫法
- 找 a 和 b 的 LCA
- 我們先把 a 和 b 拉到同一個高度
- 讓 a 和 b 同時一起往上跳
- 讓他們靠近但不會碰在一起
- 最後我們會找到的點 v 是他們 LCA 的子節點
- 所以 LCA 是 v 的父節點

倍增法

- 有第三種寫法

倍增法

- 有第三種寫法
- 樹壓平 + RMQ
- 這個東西今天不會講，因為會用到資料結構的東西
- 有興趣的自己上網查 XD

練習

CSES Company Queries I

找點 x 往上 k 步會到的點

Code

```cpp
#include<bits/stdc++.h>
#define fastio ios_base::sync_with_stdio(false);cin.tie(0)
    ;

using namespace std;

const int MAXN=2e5+1;

int n,q;
vector<vector<int>> g(MAXN);
vector<vector<int>> p(20,vector<int>(MAXN));
vector<int> dph(MAXN);

void dfs(int now,int pa,int d){
    dph[now]=d;
    p[0][now]=pa;
    for(int i:g[now]) dfs(i,now,d+1);
}

signed main(){
    fastio

    cin>>n>>q;
    for(int i=2;i<=n;++i){
        int e;cin>>e;
        g[e].emplace_back(i);
    }
    dfs(1,1,0);
    for(int i=1;i<20;++i){
        for(int j=1;j<=n;++j){
            p[i][j]=p[i-1][p[i-1][j]];
        }
    }
    while(q--){
        int u,d;cin>>u>>d;
        if(d>dph[u]){
            cout<<"-1\n";
            continue;
        }
        for(int i=0;i<20;++i){
            if(1<<i&d) u=p[i][u];
        }
        cout<<u<<'\n';
    }
}
```

練習

CSES Company Queries II

找 LCA

Code

```cpp
const int MAXN=2e5+1;

int n,q;
vector<vector<int>> g;
vector<int> in,out;
vector<vector<int>> v;
int t=0;

void dfs(int now,int p){
    v[0][now]=p;
    in[now]=t++;
    for(auto i:g[now]){
        if(i==p) continue;
        dfs(i,now);
    }
    out[now]=t++;
}
bool valid(int a,int b){
    return in[a]<=in[b]&&out[a]>=out[b];
}
int lca(int a,int b){
    if(valid(a,b)) return a;
    for(int i=19;i>=0;--i){
        if(!valid(v[i][a],b)) a=v[i][a];
    }
    return v[0][a];
}
signed main(){
    fastio

    cin>>n>>q;
    g.resize(n+1);
    in.resize(n+1,0);
    out.resize(n+1,0);
    v.resize(20,vector<int>(n+1));
    for(int i=2;i<n+1;++i){
        int u;cin>>u;
        g[u].emplace_back(i);
    }
    dfs(1,1);
    for(int i=1;i<20;++i){
        for(int j=1;j<=n;++j) v[i][j]=v[i-1][v[i-1][j]];
    }
    while(q--){
        int u,v;cin>>u>>v;
        cout<<lca(u,v)<<'\n';
    }
}
```

Practice

- CodeForces D. Misha, Grisha and Underground

最小生成樹

甚麼是最小生成樹？

- 最小生成樹，又叫 MST（Minimum Spanning Tree）
- 給你一張圖，從圖上找一些邊，讓它是一棵樹並且使邊權總合最小
- 可能不唯一
- 兩種演算法：Kruskal's Algorithm、Prim's Algorithm

Kruskal's Algorithm

- 把邊權由小排到大
- 從最小的開始，把該條邊加進樹
- 並查集維護：如果邊所連接的兩點祖先相同，就會產生環
- 如果會形成環，就略過它
- 它算是一種 greedy
- 時間複雜度 $O(|E| \log |E|)$
- 稀疏圖適用

Kruskal's Algorithm 證明

令 Kruskal 算法得出的生成樹為 K，另一棵 MST 為 T。
按照 Kruskal 算法加邊的順序，找到第一條不在 T 裡的邊 e_1，把 e_1 加到 T 裡必定會形成環，環上必定會有一條不在 K 中的邊 e_2（一定只有一條），因為 Kruskal 算法先選了 e_1，代表 $w(e_1) \leq w(e_2)$。
然後再把 e_2 拔掉，這樣是生成樹而且權重比原本的 T 還要小，所以一直重複上述動作，就可以把 T 中權重較大的邊替換掉，最後 T 就會 $= K$，K 就是最小生成樹

Prim's Algorithm

- 起點：隨便一個點，然後把它加到樹上
- 每次選樹上的所有點連出去的邊中權重最小的
- 然後把點跟邊加到樹裡面，最後就會得到 MST
- 時間複雜度 $O(|V| + |E| \log |V|)$

Prim's Algorithm 證明

需要證明的是：走的每一步都在 MST 中
如果第 k 步成立，這個時候的邊集為 E，且包含在 MST 中。令 MST 為 T，接下來要加入的邊為 e_1。
如果 e_1 在 T 中，那就會成立。
否則將 e_1 加入 T 中，加入後必定會形成環，環上必定會有一條不在 E 中的邊 e_2。
e_2 不可能小於 e_1，因為算法先選了 e_1。
e_2 不可能大於 e_1，因為如果 e_2 大於 e_1，$T + e_1 - e_2$ 會比 MST 更小。
因此 $e_1 = e_2$，$T + e_1 - e_2$ 是 MST，E 包含在其中。

練習

CSES Road Reparation

找最小生成樹

Code - Kruskal

```cpp
#include <bits/stdc++.h>
#define fastio ios_base::sync_with_stdio(false);cin.tie(0)
      ;

#pragma GCC optimize("Ofast")
#pragma GCC optimize("unroll-loops")

using namespace std;

vector<int> dsu,sz;

int findDSU(int a){
    if(dsu[a]!=a) dsu[a]=findDSU(dsu[a]);
    return dsu[a];
}

void unionDSU(int a,int b){
    a=findDSU(a);
    b=findDSU(b);
    if(sz[a]<sz[b]) swap(a,b);
    dsu[b]=a;
    sz[a]+=sz[b];
}

signed main() {
    fastio

    int n,m;cin>>n>>m;
    dsu.resize(n+1);sz.resize(n+1,1);
    for(int i=1;i<=n;++i) dsu[i]=i;
    vector<pair<int,pair<int,int>>> g;
    while(m--){
        int a,b,c;cin>>a>>b>>c;
        g.emplace_back(make_pair(c,make_pair(a,b)));
    }
    sort(g.begin(),g.end());

    int cnt=0;
    long long ans=0;
    for(auto i:g){
        int w=i.first,u=i.second.first,v=i.second.second;
        if(findDSU(u)==findDSU(v)) continue;
        ans+=w;cnt++;
        unionDSU(u,v);
    }
    if(cnt<n-1) cout<<"0\n";
    else cout<<ans<<'\n';
}
```

Code - Prim

```cpp
#include <bits/stdc++.h>
#define fastio ios_base::sync_with_stdio(false);cin.tie(0);
#pragma GCC optimize("Ofast")
#pragma GCC optimize("unroll-loops")

using namespace std;

#define pii pair<int, int>

signed main(){
    fastio
    int n, m;cin >> n >> m;
    vector<pii> g[n+1];
    vector<int> vis(n+1,0);
    for(int i = 0;i < m;++i){
        int a, b, c;cin >> a >> b >> c;
        g[a].emplace_back(make_pair(b,c));
        g[b].emplace_back(make_pair(a,c));
    }

    priority_queue<pii, vector<pii>, greater<>> pq;
    pq.push({0,1});
    int cnt = 0;
    long long ans = 0;
    while(!pq.empty()){
        int now = pq.top().second, d = pq.top().first;
        pq.pop();
        if(vis[now]) continue;
        vis[now] = 1;
        ans += d; cnt++;
        for(pii i:g[now]){
            if(!vis[i.first]) pq.push({i.second,i.first});
        }
    }

    if(cnt < n) cout << "IMPOSSIBLE\n";
    else cout << ans << '\n';
}
```

Cycle Property

- 定義：
  假設 T 是一個帶權的最小生成樹，在存在一條邊 e 以後，則會形成一個環，並且 e 會比環上任一條邊的邊權都大。
- 證明：
  我們可以利用反證法，如果環上存在一條邊 d 比 e 大，那我們可以把 d 拔掉，換成 e，那麼這樣得到的生成樹會比 T 還要小，矛盾。

Cut Property

- 定義：
  若將圖 G 分成兩個連通塊，那我們找到的最小生成樹中，必存在一條連接兩個連通塊的邊，而且這條邊是所有可以連接兩個連通塊的邊中最小的。
- 證明：
  一樣可以用反證法，我覺得你們可以自己證。

應用

- 找非嚴格次小生成樹。

應用

- 找非嚴格次小生成樹。
- 根據 cut property 我們可以發現，次小生成樹跟最小生成樹相差一條邊

應用

- 找非嚴格次小生成樹。
- 根據 cut property 我們可以發現，次小生成樹跟最小生成樹相差一條邊
- 既然我們知道只差一條邊了，假設那條邊連接 u 和 v，那就枚舉不在 MST 上的邊
- 讓這些邊去跟在 MST 中 u 到 v 路徑上的邊做替換

應用

- 找非嚴格次小生成樹。
- 根據 cut property 我們可以發現，次小生成樹跟最小生成樹相差一條邊
- 既然我們知道只差一條邊了，假設那條邊連接 u 和 v，那就枚舉不在 MST 上的邊
- 讓這些邊去跟在 MST 中 u 到 v 路徑上的邊做替換
- 顯然替換掉 MST 上最大的邊會是最優的

應用

- 找非嚴格次小生成樹。
- 根據 cut property 我們可以發現，次小生成樹跟最小生成樹相差一條邊
- 既然我們知道只差一條邊了，假設那條邊連接 u 和 v，那就枚舉不在 MST 上的邊
- 讓這些邊去跟在 MST 中 u 到 v 路徑上的邊做替換
- 顯然替換掉 MST 上最大的邊會是最優的
- 不可能每次要找最大邊都全部掃過吧

應用

- 找非嚴格次小生成樹。
- 根據 cut property 我們可以發現，次小生成樹跟最小生成樹相差一條邊
- 既然我們知道只差一條邊了，假設那條邊連接 u 和 v，那就枚舉不在 MST 上的邊
- 讓這些邊去跟在 MST 中 u 到 v 路徑上的邊做替換
- 顯然替換掉 MST 上最大的邊會是最優的
- 不可能每次要找最大邊都全部掃過吧
- 對！所以可以用前面講過的倍增法去維護它

應用

- 找非嚴格次小生成樹。
- 根據 cut property 我們可以發現，次小生成樹跟最小生成樹相差一條邊
- 既然我們知道只差一條邊了，假設那條邊連接 u 和 v，那就枚舉不在 MST 上的邊
- 讓這些邊去跟在 MST 中 u 到 v 路徑上的邊做替換
- 顯然替換掉 MST 上最大的邊會是最優的
- 不可能每次要找最大邊都全部掃過吧
- 對！所以可以用前面講過的倍增法去維護它
- 那你現在知道怎麼找非嚴格次小生成樹了，那嚴格次小呢？

Practice

- TIOJ 2164 運送蛋餅
- TIOJ 1445 機器人組裝大賽
- AtCoder D - Sum of Maximum Weights
- CodeForces D. GCD and MST
- CodeForces G. Xor-MST
- CodeForces D. Edges in MST
- CodeForces E. MST Company