4.3. Kĩ thuật bảng thưa (Sparse table)

Đây là kĩ thuật hay dùng nhất trong giải các bài toán về LCA.

Nhận xét rằng mọi số nguyên dương đều có thể biểu dưới dạng nhị phân. Nhận xét này khá quan trọng, từ đó ta có thể cải tiến cách làm của các thuật toán đã giới thiệu ở trên bằng cách nhảy lên các lũy thừa của 2 (binary lifting), từ đó việc tìm $LCA(u,v)$ chỉ có độ phức tạp $O(\log(N))$.

Giả sử $depth(u) - depth(v) = 5 = 5_{(10)} = 101_{(2)} = 1.2^2 + 0.2^1 + 1.2^0$

Khi đó để u nhảy lên tổ tiên v trên nó 5 bậc thì ta nhảy đến cha trước đó 2^2 bậc, sau đó nhảy tiếp cha 2^0 bậc.

Ta sẽ sử dụng bảng thưa (Sparse table), cha thứ 2^j của đỉnh i là $T[i][j]$.

Định nghĩa Sparse table theo công thức truy hồi như sau:

$$\begin{cases} T[u][0] = par[u] & \text{Cha cấp } 2^0 \\ T[u][i] = T\big[T[u][i-1]\big][i-1] & \text{Cha cấp } 2^i \\ & (\text{Vì } 2^i = 2^{i-1} + 2^{i-1}) \end{cases}$$

Đánh giá độ phức tạp của thuật toán:

- Độ phức tạp tiền xử lí (xây dựng bảng thưa) : $\boldsymbol{O(N * log(N))}$
- Độ phức tạp của một truy vấn là: $\boldsymbol{O(log(N))}$.

➔ Độ phức tạp thuật toán chung: $\boldsymbol{O(N * log(N) + Q * log(N))}$.

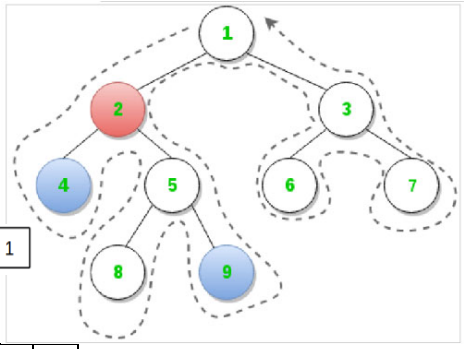
Chương trình tham khảo:

```
#include <bits/stdc++.h>
using namespace std;
int n, q, depth[100005], T[100005][17];
vector<int> adj[100005];
void dfs(int u, int p) { ///tim to tien o tang tren cua u
   depth[u] = depth[p] + 1;
   T[u][0] = p;
   for(int  i = 1; i < 17; i++)
      T[u][i] = T[T[u][i - 1]][i - 1];
   for(int v : adj[u]) {
      if(v == p)
         continue;
      dfs(v, u);
   }
}
int lca(int u, int v) { ///Tim LCA Sparse Table
   if(depth[u] < depth[v])
      swap(u, v);
   for(int i = 16; i >= 0; i--) ///nhay den cung do sau
      if(depth[T[u][i]] >= depth[v])
         u = T[u][i];
   if(u == v)
      return u;
   for(int i = 16; i >= 0; i--)///nhay den LCA
      if(T[u][i] != T[v][i]) {
         u = T[u][i];
         v = T[v][i];
```

```
        }
    return T[u][0];
}
int main() {
    cin >> n;
    for(int i = 1; i < n; i++) {
        int u, v;
        cin >> u >> v;
        adj[u].push_back(v);
        adj[v].push_back(u);
    }
    dfs(1, 0);
    cin >> q;
    while(q--) {
        int u, v;
        cin >> u >> v;
        cout << "lca(" << u << "," << v << ")=" << lca(u, v) << "\n";
    }
}
```

Để ý thấy rằng dù là Brute Force, chia căn, hay sparse table đều dùng chung một cách làm đó là: đưa 2 đỉnh về cùng tầng đã, sau đó lại đưa tiếp chúng về đến LCA.

4.4. Dùng Euler tour



Quan sát đồ thị bên. Thứ tự các đỉnh được gọi đến theo đường nét đứt.

Với đồ thị trên ta có dãy các đỉnh được gọi đến như sau:

1	2	4	2	5	8	5	9	5	2	1	3	6	3	7	3	1

Độ sâu tương ứng là:

1	**2**	**4**	**2**	**3**	**4**	**3**	**4**	**5**	**2**	**1**	**2**	**3**	**2**	**3**	**2**	**1**

Nhận xét thấy rằng $LCA(u, v)$ là đỉnh được gọi đến trong đoạn duyệt đến u, v mà có độ sâu nhỏ nhất.

Ví dụ: $LCA(4,9) = 2$ (đỉnh có độ sâu nhỏ nhất).

Kĩ thuật này có thể gọi nôm na là duỗi cây thành mảng. Đây là cách làm cũng rất hay, có nhiều ứng dụng. Kĩ thuật này có nhiều biến thể, nhưng chung quy nó đều kết hợp với DFS, trong khi DFS thì xác định một thứ tự duyệt đến, duyệt xong các đỉnh,…

Việc tìm đỉnh có độ sâu nhỏ nhất sau khi duỗi cây thành mảng là dạng bài toán Range Minimum Query (RMQ). Để giải quyết vấn đề này, chúng ta có thể sử dụng cấu trúc dữ liệu Sparse table hoặc Segment tree để giải quyết. Khi đó độ phức tạp thuật toán để trả lời $\boldsymbol{Q}$ truy vấn là: $\boldsymbol{O(Q * log(N))}$.

Vì đây là dạng quy bài toán LCA về RMQ, nên trong chuyên đề này không thể hiện lại chương trình mẫu. Bạn đọc có thể tự tham khảo tại: http://vnoi.info/wiki/translate/topcoder/Range-Minimum-Query-and-Lowest-Common-Ancestor