```cpp
#include "../bits/stdc++.h"

// 橋 (= 取り除くと連結成分が増える辺) 列挙
// グラフを適当な頂点を根とする dfs 木として以下の値を更新する
// order[v] := 行きがけ順, low[v] := 部分グラフでの min{order[i]}
// 各辺 e について, 子から order[親]以下に遷移できない <=> e は橋
// 各辺 e について, low[子] > order[親] <=> e は橋
// が, low[v] は陽に持たないで良い
// verified: http://judge.u-aizu.ac.jp/onlinejudge/review.jsp?rid=3382012
class LowLink
{
    using Graph = std::vector<std::vector<int>>;
    using P = std::pair<int, int>;
    std::vector<int> order;
    int next = 0;
    Graph G;
    std::set<P> bridges;

    void add(int a, int b)
    {
        bridges.insert(P(std::min(a, b), std::max(a, b)));
    }
    int dfs(int cur, int pre)
    {
        int res = order[cur] = next++;
        for (const auto &to : G[cur])
        {
            if (to == pre)
                continue;
            if (order[to] >= 0)
                res = std::min(res, order[to]);
            else
            {
                int low = dfs(to, cur);
                if (low > order[cur])
                    add(cur, to);
                res = std::min(res, low);
            }
        }
        return res;
    }

  public:
    LowLink(int _v) : G(_v, std::vector<int>()), order(_v, -1) {}
    void addEdge(int s, int t)
    {
        G[s].push_back(t);
    }
    std::set<P> bridge()
    {
        // 連結であることを想定
        dfs(0, -1);
        // 非連結の場合
        // for(int i = 0;i < (int)order.size(); i++) if(order[i] == -1) dfs(i, -1);
        return bridges;
    }
};

/*
二重辺連結成分分解
- 二重辺連結 := どの辺を取り除いても連結である, すなわち橋を含まないようなグラフ
- 二重辺連結成分 := グラフの内二重辺連結な部分グラフであり, 極大 (= どの頂点集合を追加しても二重辺連結にならない)なもの

橋が列挙できたら適当に dfs すれば ok
*/

```