团

性质

- 一个**团**是一个*无向图*的完全子图。
- 一个**极大团**就是一个不是任何其他团的真子集的团。
- 一个**最大团**是一个图点数最多的团。

最大团和最大独立集有一定的关系。一个图的最大独立集是该图补图的最大团，反之亦然。

求解

使用 Bron-Kerbosch 算法求无向图的极大团。

该算法可以看作是一个比较高级的暴力搜索。它在每一个搜索的时刻，都维护了三个点集：

- All：找到的极大团应当包含这个集合中所有的点。
- Some：搜索时可能会将这个集合中的点加入极大团中。其中的点需要和 All 中的点均相邻。
- None：搜索时不会再考虑这个集合中的点。其中的点曾经在 Some 中，但包括其和 All 中的点的极大团已经被发现过了。

初始调用时，将 All 和 None 设为空集，Some 设为图的点集。递归函数具体操作如下：

1. 如果 Some 和 None 均为空，那么 All 中点集形成一个极大团。

2. 否则，枚举 Some 中的点。设当前枚举到 v。

    1. 试图使用它：即将其加入 All，然后令 Some 和 None 均和 v 的邻居取交，向下递归。
    2. 恢复相关环境，将 v 从 Some 中移除，放入 None。表明已经找过和 v 有关的极大团，不再考虑它。

伪代码可以表示如下。

```
algorithm BronKerbosch1(R, P, X) is
    if P and X are both empty then
        report R as a maximal clique
    for each vertex v in P do
        BronKerbosch1(R ∪ {v}, P ∩ N(v), X ∩ N(v))
        P := P \ {v}
        X := X ∪ {v}
```

从不变量的角度看，这个算法的正确性在于：和 All 中所有点相连的点，都在 Some 或者 None 中。当 Some 为空时，就找到了团；当 None 也为空时，就找到了一个和之前不同的极大团。

如果不考虑 None，那么就可以求解团。

枢纽优化

求解极大团时，可以对上述算法进行优化，即每一层递归求解时从 Some 中选定一个*枢纽* p，设 Some 中和 p 相邻的点集为 P^-，其余点为 P^+。只使用和 P^+ 中的点。

> 这里暂时不考虑 p 有自环。

上述优化成立的原因在于 P^- 的任意子集与 All 的并都*不会构成极大团*，因为我们必然可以加入 p 到这个并集中以构成一个更大的团。因此可以只关注 P^+ 的点。

应用

求极大团个数

有时我们需要求极大团的个数，那么只需要按照 Bron-Kerbosch 算法实现即可。

这部分的代码实现如下。

```
#define MAXN 150
bool g[MAXN][MAXN];
int all[MAXN], some[MAXN][MAXN], none[MAXN][MAXN], ans;
int n, m;          // |V|, |E|
void dfs(int dep, int s_siz, int n_siz){
    if (!s_siz && !n_siz) {
        ++ans;
        return ;
    }
    int pivot = some[dep][1];
    REP(i, 1, s_siz){
        int u = some[dep][i];
        if (g[u][pivot]) continue;
        all[dep + 1] = u;
        int s_siz2 = 0, n_siz2 = 0;
        REP(j, 1, s_siz){
            if (g[u][some[dep][j]])
                some[dep + 1][++s_siz2] = some[dep][j];
        }
        REP(j, 1, n_siz){
            if (g[u][none[dep][j]])
                none[dep + 1][++n_siz2] = none[dep][j];
        }
        dfs(dep + 1, s_siz2, n_siz2);
        some[dep][i] = 0;       // invalidate
        none[dep][++n_siz] = u;
    }
}
void count_clique(){
    ans = 0;
    REP(i, 1, n)
        some[1][i] = i;
    dfs(1, n, 0);
}
```

求最大团

有时我们需要求最大团的点数，那么可以对原始算法做一定修改，使之运行的更加快速。

由于 None 是用来辅助判定极大团的，这里可以不用判定，因此我们考虑不记录 None，只记录 Some。这样我们找到的就不是极大团而是*团*。

还可以做几种剪枝：

1. 固定搜索顺序：最初按照点的编号降序搜索，即对于 Some 为全集时，每次取出的点编号降序。且对于当前点 i，只使用编号大于或等于 i 的点构成团。
2. 利用单调性：设 ans_i 为在利用点 $i, i+1, \cdots, n$ 进行搜索时，得到的最大团。那么显然 $ans_i \le ans_{i+1} + 1$。如果达到了这个上界，那么就可以直接退出了。

3. 利用此前结果：由于团的性质，如果要加入某个点 v 到团内，且 All 的大小加上 ans_v 未超过当前已有最好结果，那么可以剪枝。或者如果 All 的大小和 Some 的大小未超过当前已有最好结果，那么可以剪枝。

这部分的代码实现如下。

```
#define MAXN 150
bool g[MAXN][MAXN];
int cur, curnode, some[MAXN][MAXN];
int n, ans[MAXN];
bool dfs(int dep, int s_siz){
    if (!s_siz) {
        cur = max(cur, dep);
        return cur == n - curnode + 1;
    }
    REP(i, 1, s_siz){
        int u = some[dep][i];
        if (dep + s_siz - i + 1 <= cur || dep + ans[u] <= cur) return false;
        int s_siz2 = 0;
        REP(j, i + 1, s_siz){
            if (g[u][some[dep][j]])
                some[dep + 1][++s_siz2] = some[dep][j];
        }
        if (dfs(dep + 1, s_siz2)) return true;
    }
    return false;
}
int largest_clique(){
    ans[n] = cur = 1;
    REPR(i, n - 1, 1){
        curnode = i;
        int s_siz = 0;
        REP(j, i + 1, n){
            if (g[i][j]) some[1][++s_siz] = j;
        }
        dfs(1, s_siz);
        ans[i] = max(ans[i + 1], cur);
    }
    return ans[1];
}
```

> 这里似乎不能使用枢纽元优化，原因待查明。

求团个数

利用上述求最大团的程序，可以实现对团的数目进行计数。

最大团计数

记 $f(n)$ 为含有 n 个点的无向图最多可能拥有的最大团的数量。有公式

$$f(n) = \begin{cases} n, & \text{if } n \leq 1 \\ 3^{\frac{n}{3}}, & \text{if } n \equiv 0 \pmod 3 \\ 4 \cdot 3^{\lfloor \frac{n}{3} \rfloor - 1}, & \text{if } n \equiv 1 \pmod 3 \\ 2 \cdot 3^{\lfloor \frac{n}{3} \rfloor}, & \text{if } n \equiv 2 \pmod 3 \end{cases}$$

具体怎么推导的参见这篇文章。

例题

POJ 1419、POJ 2989、HDU 1530、LOJ 6688。

参考资料

- Bron–Kerbosch algorithm - Wikipedia
- A note on the problem of reporting maximal cliques