1 Method of Construction

The goal is to construct all connected simple graphs with given number of vertices up to isomorphism. The enumeration of such graphs is known; see, for example, [1, Chapter 4] or `A001349` of the On-Line Encyclopedia of Integer Sequences. Though the formula is known, the method used there is not useful for our purpose, because the method is, roughly speaking, counting up rooted graphs neglecting connectedness at first, then excluding disconnected ones later. It is hoped that the number of discarded graphs should be as small as possible.

We employ the following trivial principle:

Lemma 1.1 (Principle of the Least Degree (PLD))**.** *Every connected simple graph of order $d(\geq 2)$ can be constructed from a simple graph of order $d-1$ by adjoining a vertex with the least degree.*

Note that subgraph of order $d-1$ is not assumed to be connected. It is, however, possible to use only connected subgraphs for $d \leq 8$.

We are going to describe general construction and prove the claim above. Then we briefly state about isomorphism, summarize them into algorithms and finally make some implementation remarks.

By the principle of the least degree (PLD), we construct a connected simple graph G of order d from a simple graph G' of order $d-1$ and a vertex v_0.

Assume that the subgraph G' has k connected components, and v_0 will have degree m after the construction. Then, the first condition that should be satisfied is $k \leq m$. If m is one, then k has to be one as well, i.e. G' is connected.

Consider the case that $k \geq 2$. Fix a partition of number m into exactly k parts: $m = \sum_{i=1}^{k} m_i$. Then each connected component G_i of G' has its vertex set V_i as a disjoint union of subsets H_i, J_i and L_i:

1. H_i: every vertex v has degree $\deg(v) \geq m+1$.
2. J_i: empty unless $i = 1$, and every vertex v is a cut vertex and has degree $\deg(v) \geq m$.
3. L_i: $|L_i| = m_i$ and every vertex v has degree $\deg(v) \geq m$ when $m_i > 1$ or $\deg(v) \geq m-1$ otherwise.

Each G_i will be connected with v_0 at each vertex in L_i. With these notations, we show the following theorem:

Theorem 1.2. *Every connected simple graph of order $d(\geq 2)$ can be constructed from subgraph of order $d-1$ and a vertex. Moreover, the subgraph of order $d-1$ can be either:*

1. *a connected simple graph or*
2. *$k(>1)$ disconnected graphs G_i satisfying the conditions above.*

Proof. It is proved by mathematical induction.

If $d = 2$, the only subgraph of order $d - 1$ is a connected simple graph; it is a vertex without any edges.

Assume up to $d - 1$, the statement holds.

Any graph G of order d has the least degree vertex v_0. If v_0 is not a cut vertex, the graph $G - v_0$ induced from G by deleting v_0 is still connected graph of order $d - 1$. In particular, if the degree condition for L_i is not satisfied, one can choose non-cut vertex. Thus the remaining case is that all vertices with the least degree are cut vertices.

Pick up one of such cut vertices v. Then, we have a graph $G - v$ induced from G by deleting v. If all connected components of $G - v$ except one has no cut vertex, then all the conditions required for H_i, J_i and L_i are satisfied. Otherwise, we can pick up another cut vertex v' with the least degree from a component with such cut vertices, but the one with the smallest number of cut vertices. Then, this process of changing splitting point will certainly stop in finite steps, since the number of possible choices is finite and the number of the least degree cut vertices in chosen component decreases each time.

□

Corollary 1.3. *The order of a connected simple graph that needs the construction by PLD from k components and degree m vertex is at least $k(m + 2) + 1$.*

Proof. By the construction above, each component G_i has a vertex in H_i. Otherwise, there are only vertices in L_i. It is, however, a contradiction. Assume $m_i = 1$, then $|L_i| = 1$ and the degree of sole vertex is $0 < m - 1$, which contradicts with the condition $\deg(v) \geq m - 1$. On the contrary, $m_i > 1$ implies that each vertex has degree at least m, but then $|L_i| \geq m + 1 > m_i$; it contradicts with the condition $|L_i| = m_i$.

Then, a vertex in H_i has degree at least $m+1$ and thus $|G_i| \geq m+2$. There are k components with order at least $m + 2$ and the adjoined vertex. □

The last corollary shows that it is unnecessary for $d \leq 8$ to consider disconnected subgraphs, as claimed in the introduction. Another useful information is that for $d = 9$, there is only one graph to be constructed from disconnected subgraph: two complete graph K_4s connected by a vertex of degree 2.

The graphs constructed by the method described above, include many isomorphic graphs. We need only one representative per isomorphism class. To achieve it, we check graph isomorphisms.

Note that our consideration about isomorphism explained in this section is not fully explored, since we only use rather small graphs.

There are two approaches; one is to find normal form of graphs so that isomorphic graphs can be detected by comparing the normal form identity, another is to construct explicit isomorphism between two graphs so that isomorphic graphs can be distinguished when an isomorphism is found. Both methods have automorphism groups of every graph in mind. The automorphism group for a graph with d vertices is considered as a subgroup of the symmetric group of d elements. Thus, it grows, in general, rapidly with d.

Proposition 1.4. *Let the number of vertices having degrees ν in a graph G be n_ν, then the automorphism group of G is a subgroup of $\hat{S} = S_{n_1} \times S_{n_2} \times \cdots \times S_{n_t}$, where t is the highest degree and S_μ is the μ-th symmetric group.*

This proposition has weakness for regular graphs; in such case $\hat{S}$ is trivially known $S_{|V|}$ itself. However, in many other cases, the group $\hat{S}$ is rather small.

Data structure for graphs in the following algorithm is an incidence matrix. This choice is made because the edges play central role in our application.

Our main driver algorithm is ConnectedSimpleGraphs written in pseudocode below, followed by sub-algorithms ConnectedConstruct and DisconnectedConstruct.

Algorithm 1.5. (ConnectedSimpleGraphs)
input: d
output: All connected simple graph of order d representatives up to isomorphism

```
graphs ⟵ empty list
if d = 1:
        append the simple graph of order 1 to graphs
else:
        append all graphs in ConnectedConstruct(d) to graphs
        append all graphs in DisconnectedConstruct(d) to graphs
        while ∃G_1, G_2 ∈ graphs s.t. G_1 ≅ G_2:
                remove G_2 from graphs
return graphs.
```

Algorithm 1.6. (ConnectedConstruct)
input: d
output: Connected simple graphs of order d constructed by adjoining a vertex to a connected simple graph of order $d-1$

```
graphs ⟵ empty list
for G' in ConnectedSimpleGraphs(d − 1):
        # G' is given as an incidence matrix
        add a row to G'          # add v_0
        m ⟵ 0
        𝒢' ⟵ [G']
        while m ≤ min_{G∈𝒢'} min_{v∈G}(deg(v)):
                𝒢 ⟵ empty list
                for G in 𝒢':
                        for e ∉ E(G) and adjacent to v_0:
                                if G + e does not break the condition m + 1 ≤
                                min_{v∈G+e}(deg(v)):
                                        Append G + e to 𝒢
```

Append all graphs in $\mathcal{G}'$ to `graphs`
if $\mathcal{G}$ is not empty:
$\mathcal{G}' \longleftarrow \mathcal{G}$
else:
break
$m \longleftarrow m + 1$
return `graphs`.

Algorithm 1.7. (DisconnectedConstruct)
input: d
output: Connected simple graphs of order d constructed by adjoining a vertex to disconnected simple graph of order $d - 1$

`graphs` $\longleftarrow$ empty list
for k, m s.t. $1 + k(m + 2) \leq d$:
for partition of m into exactly k parts $m = \sum_{i=1}^{k} m_i$:
for combinations $(\{m_{\nu_1}\}, \{m_i \mid i \neq \nu_1\})$:
for partition of $d - 1$ into exactly k parts $d = 1 + \sum_{i=1}^{k} d_i$ where $d_i \geq m_i$:
Choose G_i with d_i vertices and satisfying the conditions about H_i, J_i and L_i and connect them with a new vertex.
Append all obtained graphs to `graphs`.
return `graphs`.

There are a few points we omit the detail in the algorithms above.

The first point is that the algorithms hold all graphs in memory at once, even redundant graphs to be deleted later. The bigger the number of vertices becomes, the harder it becomes to do so. Therefore, it is usually necessary to split the set of graphs. The simplest way is to split it by the number of edges in addition to the number of vertices. Modification is an easy exercise, and left to the reader.

The usage of list append is another over-simplified point. In reality, in order to check isomorphism between graphs, it is useful to sort the graphs according to a linear order; though the choice of order does not have much importance as long as it is easy to calculate. The searching of isomorphic graph, then, can be done using the binary search algorithm. Therefore, "append" in the algorithm should be read as "insert to the appropriate position if there is no isomorphic graph already".

The last point is isomorphism check. The property we use is explained in Proposition 1.4. It is far from optimal. There are, however, more powerful graph isomorphism algorithms. If we will continue the computation, maybe [2] will be one of the references.

References

[1] F. Harary and E. M. Palmer. *Graphical Enumeration*. Academic Press, New York and London, 1973.

[2] B. D. McKay. Practical graph isomorphism. *Congressus Numerantium*, 30:45 – 87, 1981.