On the number of non-isomorphic simple connected planar graphs of order 9 and 10

Osamu NAKAMURA
Faculty of Education

The numbers of non-isomorphic simple planar graphs of order p $(1 \leq p \leq 8)$ are listed in [7] $(1 \leq p \leq 6)$ and in [3] $(7 \leq p \leq 8)$. In this paper we shall compute the number of non-isomorphic simple connected planar graphs of order 9 and 10. We have the following theorem.

THEOREM. There are 73029 non-isomorphic simple connected planar graphs of order 9 and there are 1085094 non-isomorphic simple connected planar graphs of order 10. More precisely, we have the following table.

p=1 number=1
(0,1)
p=2 number=1
(1,1)
p=3 number=2
(2,1),(3,1)
p=4 number=6
(3,2),(4,2),(5,1),(6,1)
p=5 number=20
(4,3),(5,5),(6,5),(7,4),(8,2),(9,1)
p=6 number=99
(5,6),(6,13),(7,19),(8,22),(9,19),(10,13),(11,5),(12,2)
p=7 number=647
(6,11),(7,33),(8,67),(9,107),(10,130),(11,130),(12,96),(13,51),(14,17),(15,5)
p=8 number=6016
(7,23),(8,89),(9,236),(10,486),(11,804),(12,1113),(13,1216),(14,1036),(15,641),(16,283),
(17,75),(18,14)
p=9 number=73029
(8,47),(9,240),(10,797),(11,2075),(12,4454),(13,8069),(14,12057),(15,14582),(16,13681),
(17,9762),(18,5031),(19,1799),(20,385),(21,50)
p=10 number=1085094
(9,106),(10,657),(11,2678),(12,8548),(13,22768),(14,51941),(15,99880),(16,159371),
(17,205168),(18,209423),(19,165753),(20,99516),(21,43520),(22,13124),(23,2407),(24,234)

Here, the bracket (q, n) means that q is the size of the graphs and n is the number of graphs with the size q.

These results are obtained by a personal computer with a Pentium 133M Hz processor and 80M byte memory. Our program is written by C++ and find all non-isomorphic simple connected planar graphs of order p ($p \leq 10$). We have used some well-known algorithms. A mathematica program of checking planarity of graph is given in [5]. But this program has some bugs. We debug this program and translate it from mathematica into C++. A mathematica program of checking whether two graphs are isomorphic or not is given in [5]. We translate it from mathematica into C++. Our program is the following.

```
#include <stdio.h>
#include <stdlib.h>
#define TRUE 1
#define FALSE 0
const int maxvertices = 11;
const int maxedges = 64;
const int BITSLEN = 2;
int number, n_vertex, p[20];
struct edge {
    int v[2];
};
struct edges {
    edge e[maxedges];
    int len;
    edges(){len=0;};
} alledges;
struct bigdigit {
    unsigned char len, *d;
    bigdigit(){len = 0; d = NULL;};
    bigdigit(int n);
    bigdigit(const bigdigit&);
    ~bigdigit(){delete [] d;};
    void set(int n);
    bigdigit& operator=(const bigdigit&);
};
bigdigit::bigdigit(int n)
{
    len = n;
    d = new unsigned char[n*(n+1)/2];
    for (int i=0; i<len; i++)
        for (int j=0; j<(n+1)/2; j++)
            d[i*(n+1)/2+j] = 0;
}
bigdigit::bigdigit(const bigdigit& a)
```

```
}
    len = a.len;
    d = new unsigned char[len*(len+1)/2];
    for (int i=0; i<len; i++)
        for (int j=0; j<(len+1)/2; j++)
            d[i*(len+1)/2+j] = a.d[i*(len+1)/2+j];
}
void bigdigit::set(int n)
{
    len = n;
    d = new unsigned char[n*(n+1)/2];
    for (int i=0; i<len; i++)
        for (int j=0; j<(n+1)/2; j++)
            d[i*(n+1)/2+j] = 0;
}
bigdigit& bigdigit::operator=(const bigdigit& a)
{
    if (this != &a) {
        len = a.len;
        delete. d;
        d = new unsigned char [len*(len+1)/2];
        for (int i=0; i<len; i++)
            for (int j=0; j<(len+1)/2; j++)
                d[i*(len+1)/2+j] = a.d[i*(len+1)/2+j];
    }
    return *this;
}
int operator>(bigdigit d1, bigdigit d2)
{
    unsigned char n1, n2;
    int len = d1.len;
    for (int i=0; i<len; i++)
        for (int j=0; j<len; j++) {
            n1 = d1.d[i*(len+1)/2+j/2]; n2 = d2.d[i*(len+1)/2+j/2];
            n1 >>= 4 * (j % 2); n2 >>= 4 * (j % 2); n1 &= 15; n2 &= 15;
            if (n1 > n2) return -1;
            else if (n1 < n2) return 0;
        }
    return 0;
}
int operator==(bigdigit d1, bigdigit d2)
{
```

```
    unsigned char n1, n2;
    int len = d1.len;
    for (int i=0; i<len; i++)
        for (int j=0; j<len; j++) {
            n1 = d1.d[i*(len+1)/2+j/2]; n2 = d2.d[i*(len+1)/2+j/2];
            n1 >>= 4 * (j % 2); n2 >>= 4 * (j % 2); n1 &= 15; n2 &= 15;
            if (n1 != n2) return 0;
        }
    return -1;
}
struct invariant {
    int s[maxvertices][maxvertices], len;
    bigdigit get_digit();
};
bigdigit invariant::get_digit()
{
    bigdigit digit(len);
    unsigned char uc;
    int min, flag, i, j, k;
    invariant t = *this;
    for (i=0; i<len; i++) {
        min = i;
        for (j=i+1; j<len; j++) {
            flag = 0;
            for (k=0; k<len; k++) {
                if (t.s[min][k] > t.s[j][k]) {
                    flag = -1; break;
                }
                if (t.s[min][k] < t.s[j][k]) break;
            }
            if (flag == -1) min = j;
        }
        for (k=0; k<len; k++) {
            uc = t.s[min][k] << (4*(k%2));
            digit.d[i*(len+1)/2+k/2] |= uc;
        }
        if (min != j)
            for (k=0; k<len; k++)
                t.s[min][k] = t.s[i][k];
    }
    return digit;
}
```

```
struct p_list {
    int p[maxvertices][maxvertices], n_p[maxvertices], len;
};
class adj_matrix {
    unsigned long a[maxvertices][2];
public:
    adj_matrix();
    int get(int v1, int v2);
    void set(int v1, int v2);
    invariant shortest_path();
};
adj_matrix::adj_matrix()
{
    for (int i=0; i<maxvertices; i++)
        a[i][0] = a[i][1] = 0;
}
int adj_matrix::get(int v1, int v2)
{
    int q = v2 / 32, r = v2 % 32;
    return (int)((a[v1][q] >> r) & ((unsigned long)1));
}
void adj_matrix::set(int v1, int v2)
{
    int q = v2 / 32, r = v2 % 32;
    a[v1][q] |= (unsigned long)1 << r;
}
invariant adj_matrix::shortest_path()
{
    invariant inv;
    int t[maxvertices][maxvertices], i, j, q, r, x, y, k, max;
    for (i=0; i<n_vertex; i++)
        for (j=0; j<n_vertex; j++) {
            q = j / 32; r = j % 32;
            t[i][j] = (int)((a[i][q] >> r) & ((unsigned long)1));
        }
    for (y=0; y<n_vertex; y++)
        for (x=0; x<n_vertex; x++)
            if (t[x][y])
                for (j=0; j<n_vertex; j++)
                    if (t[y][j] > 0)
                        if (!t[x][j] || (t[x][y]+t[y][j] < t[x][j]))
                            t[x][j] = t[x][y]+t[y][j];
```

```
    for (i=0; i<n_vertex; i++)
        for (j=0; j<n_vertex-1; j++) {
            max = t[i][j];
            for (k=j+1; k<n_vertex; k++)
                if (t[i][k] > max) {
                    t[i][j] = t[i][k]; t[i][k] = max; max = t[i][j];
                }
        }
    for (i=0; i<n_vertex; i++)
        for (j=0; j<n_vertex; j++)
            inv.s[i][j] = t[i][j];
    inv.len = n_vertex;
    return inv;
}
struct path {
    int v[maxvertices+1], len;
    int memberP(int x);
    int intersectP(path c);
    void droplast(){len--;}
    int edgeP(int v0, int v1);
    path getcode(path c);
};
int path::memberP(int x)
{
    for (int i=0; i<len; i++)
        if (x == v[i]) return 1;
    return 0;
}
int path::intersectP(path c)
{
    for (int i=0; i<len; i++)
        for (int j=0; j<c.len; j++)
            if (v[i] == c.v[j]) return TRUE;
    return FALSE;
}
int path::edgeP(int v0, int v1)
{
    for (int i=0; i<len-1; i++)
        if (((v0==v[i])&&(v1==v[i+1]))||((v1==v[i])&&(v0==v[i+1])))
            return TRUE;
    if (((v0==v[len-1])&&(v1==v[0]))||((v1==v[len-1])&&(v0==v[0])))
        return TRUE;
```

```
    return FALSE;
}
path path::getcode(path c)
{
    path code;
    int i, j, temp;
    for (i=0; i<c.len; i++)
        for (j=0; j<len; j++)
            if (c.v[i] == v[j]) {
                code.v[i] = j; break;
            }
    code.len = c.len;
    for (i=0; i<code.len-1; i++)
        for (j=i+1; j<code.len; j++)
            if (code.v[i] > code.v[j]) {
                temp = code.v[i]; code.v[i] = code.v[j]; code.v[j] = temp;
            }
    return code;
}
struct components {
    path b[3*maxvertices];
    int len;
    int lockP(path code);
    void add(path code);
};
int lock1P(path a, path b)
{
    path c;
    int i, flag, bk, aj;
    if (a.len <= 1 || b.len <= 1) return FALSE;
    c = b; c.droplast(); flag = FALSE;
    for (i=0; i<c.len; i++)
        if (c.v[i] > a.v[0])
            if (flag == FALSE) {
                flag = TRUE; bk = c.v[i];
            }
        else if (bk > c.v[i]) bk = c.v[i];
    if (flag == FALSE) return FALSE;
    flag = FALSE;
    for (i=0; i<a.len; i++)
        if (a.v[i] > bk)
            if (flag == FALSE){
```

```
                flag = TRUE; aj = a.v[i];
            }
            else if (a.v[i] < aj) aj = a.v[i]
    if (flag == FALSE) return FALSE;
    fi (aj < b.v[b.len-1)] return TRUE;
    else if (aj == b.v[b.len-1] && a.v[0] == b.v[0]) {
        for (i=1; i<a.len; i++)
            if (aj > a.v[i] && a.v[i] > a.v[0]) return TRUE;
    }
    return FALSE;
}
int components::lockP(path code)
{
    for (int i=0; i<len; i++)
        if (lock1P(b[i], code) || lock1P(code, b[i])) return TRUE;
    return FALSE;
}
void components::add(path code)
{
    b[len] = code; len++;
}
struct adj_list {
    int a[maxvertices][maxvertices], n_a[maxvertices], len;
    adj_list();
    int getE();
    path FindCycle();
    int memberP(int ind, int x);
    adj_list removecycle(path c);
    components getcomponents();
    adj_list getsubgraph(path comp);
    adj_list isolatesubgraph(adj_list A, path cycle, path c);
    adj_list joincycle(path cycle);
    path getshortestpaht(int v0, int v1);
    int planargivencycle(path c);
    int planarP();
};
adj_list::adj_list()
{
    for (int i=0; i<maxvertices; i++) {
        n_a[i] = 0;
        for (int j=0; j<maxvertices; j++) a[i][j] = 0;
    }
```

```
}
int adj_list::getE()
{
    int i, cnt = 0;
    for (i=0; i<n_vertex; i++) cnt += n_a[i];
    return cnt / 2;
}
int locate(int v1, int v2)
{
    for (int i=0; i<alledges.len; i++)
        if ((v1 == alledges.e[i].v[0]) && (v2 == alledges.e[i].v[1]))
            return i;
}
struct stack {
    int s[4*maxvertices], n_s;
    void initialize(){n_s = 0;}
    int pop(){return s[--n_s];}
    void push(int x){s[n_s++] = x;}
    int emptyP(){return (n_s == 0) ? 1 : 0;}
};
path FromParent(int parent[], int s)
{
    path lst;
    int i = s, j;
    lst.len = 0;
    while (!lst.memberP(i)) {
        lst.v[lst.len++] = i; i  = parent[i];
    }
    lst.v[lst.len++] = i;
    for (j=0; j < lst.len; j++)
        if (lst.v[j] == i) break;
    for (i=j; i < lst.len; i++) lst.v[i-j] = lst.v[i];
    lst.len -= j;
    return lst;
}
path adj_list::FindCycle()
{
    path lst;
    int i, j, x, parent[maxvertices], seen[maxvertices], n_seen;
    stack queue;
    int flag;
    for (int v=0; v<n_vertex; v++) {
```

```
        for (i=0; i<n_vertex; i++)
            parent[i] = n_vertex;
        parent[v] = -1; n_seen =0;
        queue.initialize(); queue.push(v);
        while (!queue.emptyP()) {
            x = queue.pop(); seen[n_seen++] = x;
            for (i=0; i<n_a[x]; i++)
                if (parent[x] != a[x][i]) parent[a[x][i]] = x;
            if ((*this).memberP(x, v) && parent[x] != v)
                return FromParent(parent, x);
            for (i=0; i<n_a[x]; i++) {
                flag = TRUE;
                for (j=0; j<n_seen; j++)
                    if (a[x][i] == seen[j]) {
                        flag = FALSE; break;
                    }
                if (flag) queue.push(a[x][i]);
            }
        }
    }
    lst.len = 0; return lst;
}
int adj_list::memberP(int ind, int x)
{
    for (int i=0; i<n_a[ind]; i++)
        if (a[ind][i] == x) return TRUE;
    return FALSE;
}
adj_list adj_list::removecycle(path c)
{
    adj_list A;
    int i, j;
    for (i=0; i<n_vertex; i++) {
        A.n_a[i] = 0;
        if (c.memberP(i)) continue;
        for (j=0; j<n_a[i]; j++)
            if (c.memberP(a[i][j])) continue;
            else A.a[i][A.n_a[i]++] = a[i][j];
    }
    return A;
}
void trip(int n, adj_list A, int val[], components &comp, int ind)
```

```
{
    val[n] = 1;
    comp.b[ind].v[comp.b[ind].len++] = n;
    for (int i=0; i<A.n_a[n]; i++)
        if (val[A.a[n][i]] == 0)
            trip(A.a[n][i], A, val, comp, ind);
}
components adj_list::getcomponents()
{
    components comp;
    int ind = 0, val[maxvertices], k;
    for (k=0; k<n_vertex; k++) {
        val[k] = 0; comp.b[k].len = 0;
    }
    for (k=0; k<n_vertex; k++)
        if (val[k] == 0) trip(k, (*this), val, comp, ind++);
    comp.len = ind; return comp;
}
adj_list adj_list::getsubgraph(path comp)
{
    adj_list g;
    for (int i=0; i<n_vertex; i++) {
        g.n_a[i] = 0;
        if (!comp.memberP(i)) continue;
        for (int j=0; j<n_a[i]; j++)
        if (comp.memberP(a[i][j])) {
            g.a[i][g.n_a[i]] = a[i][j]; g.n_a[i]++;
        }
    }
    return g;
}
adj_list adj_list::isolatesubgraph(adj_list A, path cycle, path c)
{
    int i, j;
    adj_list lst;
    for (i=0; i<n_vertex; i++)
        for (j=0; j<A.n_a[i]; j++)
            if (c.memberP(i) && c.memberP(A.a[i][j]))
                lst.a[i][lst.n_a[i]++] = A.a[i][j];
    for (i=0; i<n_vertex; i++)
        for (j=0; j<n_a[i]; j++)
            if ((cycle.memberP(i) || cycle.memberP(a[i][j]))
```

```
                && (c.memberP(i) || c.memberP(a[i][j])))
                lst.a[i][lst.n_a[i]++] = a[i][j];
    return lst;
}
int aux_lockP(path C[], int n_C, path cycle, int ind,
              components in, components out)
{
    components newin, newout;
    path code;
    if (ind == n_C) return FALSE;
    code = cycle.getcode(C[ind]);
    if (!in.lockP(code)) {
        newin = in; newin.add(code);
        if (!aux_lockP(C, n_C, cycle, ind+1, newin, out))
            return FALSE;
    }
    if (!out.lockP(code)) {
        newout = out; newout.add(code);
        return aux_lockP(C, n_C, cycle, ind+1, in, newout);
    }
    return TRUE;
}
int interlockP(path C[], int n_C, path cycle)
{
    components in, out;
    in.len = 0;
    out.len = 0;
    return aux_lockP(C, n_C, cycle, 0, in, out);
}
adj_list adj_list::joincycle(path c)
{
    adj_list g = *this;
    for (int i=0; i<c.len-1; i++) {
        g.a[c.v[i]][g.n_a[c.v[i]]] = c.v[i+1]; g.n_a[c.v[i]]++;
        g.a[c.v[i+1]][g.n_a[c.v[i+1]]] = c.v[i]; g.n_a[c.v[i+1]]++;
    }
    if (c.len > 2) {
        g.a[c.v[0]][g.n_a[c.v[0]]] = c.v[c.len-1]; g.n_a[c.v[0]]++;
        g.a[c.v[c.len-1]][g.n_a[c.v[c.len-1]]] = c.v[0];
        g.n_a[c.v[c.len-1]]++;
    }
    return g;
```

```
}
#define INT_MAX 30000
path adj_list::getshortestpath(int v0, int v1)
{
    path shortestpath, temp;
    int weight[maxvertices][maxvertices], i, j, next, min;
    int visited[maxvertices], distance[maxvertices], prev[maxvertices];
    int pos, cnt;
    for (i=0; i<n_vertex; i++)
        for (j=0; j<n_vertex; j++)
            if ((*this).memberP(i, j)) weight[i][j] = 1;
            else weight[i][j] = INT_MAX;
    for (i=0; i<n_vertex; i++) weight[i][i] = 0;
    for (i=0; i<n_vertex; i++) {
        visited[i] = FALSE; distance[i] = INT_MAX;
    }
    distance[v0] = 0; next = v0;
    do {
        i = next; visited[i] = TRUE; min = INT_MAX;
        for (j=0; j<n_vertex; j++) {
            if (visited[j]) continue;
            if (weight[i][j] < INT_MAX
                && distance[i] + weight[i][j] < distance[j]) {
                distance[j] = distance[i] + weight[i][j]; prev[j] = i;
            }
            if (distance[j] < min) {
                min = distance[j]; next = j;
            }
        }
    } while (min < INT_MAX);
    temp.v[0] = v1; cnt = 1; pos = v1;
    do {
        pos = prev[pos]; temp.v[cnt] = pos; cnt++;
    } while (pos != v0);
    for (i=0; i<cnt; i++) shortestpath.v[i] = temp.v[cnt-i-1];
    shortestpath.len = cnt;
    return shortestpath;
}
int singlebridgeP(adj_list g, path c)
{
    adj_list newg;
    path shortestpath;
```

```
    int len, int i;
    if (c.len == 1)
        return g.planarP();
    newg = g.joincycle(c);
    shortestpath = g.getshortestpath(c.v[0], c.v[1]);
    len = shortestpath.len;
    for (i=2; i<c.len; i++) shortestpath.v[len+i-2] = c.v[i];
    shortestpath.len = len + c.len - 2;
    return newg.planargivencycle(shortestpath);
}
int adj_list::planargivencycle(path cycle)
{
    adj_list A, b[maxvertices];
    int n_b, i, j, ind = 0, n_C;
    components comp;
    path C[maxvertices];
    A = (*this).removecycle(cycle); comp = A.getcomponents();
    for (i=0; i<comp.len; i++) {
        if (comp.b[i].len == 1 && (*this).n_a[comp.b[i].v[0]] == 0)
            continue;
        if (!cycle.intersectP(comp.b[i])) {
            b[ind] = (*this).isolatesubgraph(A, cycle, comp.b[i]); ind++;
        }
    }
    n_b = ind;
    for (i=0; i<maxvertices; i++) C[i].len = 0;
    for (i=0; i<n_b; i++)
        for (j=0; j<cycle.len; j++)
            if (b[i].n_a[cycle.v[j]] > 0) C[i].v[C[i].len++] = cycle.v[j];
    n_C = n_b;
    for (i=0; i<n_vertex; i++)
        for (j=0; j<n_a[i]; j++) {
            if (i > a[i][j]) continue;
            if (cycle.memberP(i) && cycle.memberP(a[i][j]))
                if (!cycle.edgeP(i, a[i][j])) {
                    C[n_C].v[0] = i; C[n_C].v[1] = a[i][j];
                    C[n_C].len = 2; n_C++;
                }
        }
    if (interlockP(C, n_C, cycle)) return FALSE;
    for (i=0; i<n_b; i++)
        if (!singlebridgeP(b[i], C[i])) return FALSE;
```

```
    return TRUE;
}
int adj_list::planarP()
{
    components comp;
    int i, E;
    adj_list A;
    path cycle;
    comp = (*this).getcomponents();
    for (i=0; i<comp.len; i++) {
        if (comp.b[i].len == 1) continue;
        A = (*this).getsubgraph(comp.b[i]); E = A.getE();
        if (comp.b[i].len > 2 && E > 3 * comp.b[i].len - 6) return FALSE;
        if (E < comp.b[i].len + 3) continue;
        cycle = A.FindCycle();
        if (cycle.len == 0) continue;
        cycle.droplast();
        if (!A.planargivencycle(cycle)) return FALSE;
    }
    return TRUE;
}
class bits {
    unsigned long b[BITSLEN];
public:
    bits();
    void initialize();
    int get(int ind);
    void set(int ind);
    void unset(int ind);
    void put();
    int parity();
    friend int operator==(bits G1, bits G2);
    adj_matrix get_adj();
    adj_list get_a_list();
    int connectedP();
};
bits::bits()
{
    for (int i=0; i<BITSLEN; i++) b[i] = 0;
}
void bits::initialize()
{
```

```
    for (int i=0; i<BITSLEN; i++) b[i] = 0;
}
int bits::get(int ind)
{
    int q = ind / 32, r = ind % 32;
    return ((b[q] >> r) & (unsigned long)1);
}
void bits::set(int ind)
{
    int q = ind / 32, r = ind % 32;
    b[q] |= (unsigned long)1 << r;
}
void bits::unset(int ind)
{
    int q = ind / 32, r = ind % 32;
    b[q] &= ~((unsigned long)1 << r);
}
void bits::put()
{
    printf("|");
    for (int i=0; i<BITSLEN; i++)
        for (int j=0; j<32; j++) {
            if (((b[i] >> j) & (unsigned long)1))
                printf("|%d,%d|", alledges.e[32*i+j].v[0],
                                  alledges.e[32*i+j].v[1]);
        }
    printf("|\n");
}
int bits::parity()
{
    int i, j, cnt = 0;
    for (i=0; i<BITSLEN; i++)
        for (j=0; j<32; j++)
            if ((b[i] >> j) & (unsigned long)1) cnt++;
    return cnt;
}
int operator==(bits G1, bits G2)
{
    for (int i=0; i<BITSLEN; i++)
        if (G1.b[i] != G2.b[i] return 0;
    return -1;
}
```

```
adj_matrix bits::get_adj()
{
    adj_matrix a;
    int i, j, v1, v2;
    for (i=0; i<BITSLEN; i++)
        for (j=0; j<32; j++)
            if (b[i] & ((unsigned long)1 << j)) {
                v1 = alledges.e[32*i+j].v[0];
                v2 = alledges.e[32*i+j].v[1];
                a.set(v1-1, v2-1); a.set(v2-1, v1-1);
        }
    return a;
}
adj_list bits::get_a_list()
{
    adj_list A;
    int i, j, v0, v1;
    for (i=0; i<BITSLEN; i++)
        for (j=0; j<32; j++)
            if (b[i] & ((unsigned long)1 << j)) {
                v0 = alledges.e[32*i+j].v[0] - 1;
                v1 = alledges.e[32*i+j].v[1] - 1;
                A.a[v0][A.n_a[v0]++] = v1; A.a[v1][A.n_a[v1]++] = v0;
            }
    A.len = n_vertex; return A;
}
void visit(int ind, adj_matrix a, int val[])
{
    int i;
    val[ind] = 1;
    for (i=0; i<n_vertex; i++)
        if (a.get(ind, i))
            if (val[i] == 0) visit(i, a, val);
}
int bits::connectedP()
{
    adj_matrix a;
    int i, val[maxvertices];
    for (i=0; i<maxvertices; i++) val[i] = 0;
    a = (*this).get_adj(); visit(0, a, val);
    for (i=0; i<n_vertex; i++)
        if (val[i] == 0) return 0;
```

```
    return 1;
}
void set_alledges(int n)
{
    int i, k, c[3];
    c[0] = -1; k = 1; c[1] = 1;
    do {
        for (i=k+1; i<=2; i++) c[i] = c[i-1] + 1;
        for (i=1; i<=2; i++) alledges.e[alledges.len].v[i-1] = c[i];
        alledges.len++; k = 2;
        while (c[k] == n-2+k) k--;
        c[k]++;
    } while (k != 0);
}
bits initial(int len)
{
    bits t;
    int i, q, r;
    for (i=0; i<len; i++) t.set(i);
    return t;
}
p_list get_list(invariant inv1, invariant inv2)
{
    p_list perm;
    int i, j, k, flag;
    perm.len = n_vertex;
    for (i=0; i<n_vertex; i++) {
        perm.n_p[i] = 0;
        for (j=0; j<n_vertex; j++) {
            flag = -1;
            for (k=0; k<n_vertex; k++)
                if (inv1.s[i][k] != inv2.s[j][k]) {
                    flag = 0; break;
                }
            if (flag) perm.p[i][perm.n_p[i]++] = j;
        }
    }
    return perm;
}
int perm_P(int *t, int n_t)
{
    for (int i=0; i<n_t-1; i++)
```

```
        for (int j=i+1; j<n_t; j++)
            if (t[i] == t[j]) return 0;
    return -1;
}
int perm_check(p_list plist, bits G1, bits G2, int cnt)
{
    bits TG;
    int v1, v2, temp, pos;
    for (int i=0; i<plist.n_p[cnt]; i++) {
        p[cnt] = plist.p[cnt][i];
        if (cnt < plist.len-1) {
            if (perm_check(plist, G1, G2, cnt+1) == -1) return -1;
            else continue;
        }
        if (!perm_P(p, n_vertex)) continue;
        TG.initialize();
        for (i=0; i<alledges.len; i++) {
            if (!G1.get(i)) continue;
            v1 = p[alledges.e[i].v[0]-1]+1;
            v2 = p[alledges.e[i].v[1]-1]+1;
            if (v1 > v2) {
                temp = v1; v1 = v2; v2 = temp;
            }
            TG.set(locate(v1, v2));
        }
        if (TG == G2) return -1;
    }
    return 0;
}
int iso_P(bits G1, invariant inv1, bits G2, invariant inv2)
{
    p_list plist = get_list(inv1, inv2);
    return perm_check(plist, G1, G2, 0);
}
struct patternlist {
    bits G;
    patternlist *next;
};
struct node {
    bits G;
    patternlist *list;
    node *l, *r;
```

```
} *head, *z;
void treeinitialize(int n)
{
    z = new(node); z->l = z; z->r =z; z->list = NULL;
    head = new(node); head->l = z; head->r = z;
    (head->G).initialize(); head->list = NULL;
}
int treesearch(node *x, bigdigit digit1, invariant inv1, bits G)
{
    patternlist *pl;
    adj_matrix adj2;
    invariant inv2;
    bigdigit digit2;
    z->G = G;
    do {
        adj2 = (x->G).get_adj(); inv2 = adj2.shortest_path();
        digit2 = inv2.get_digit();
        if (digit2 == digit1) break;
        if (digit2 > digit1) x = x->l;
        else                 x = x->r;
    } while(1);
    if (x == z) return 0;
    pl = x->list;
    while(pl != NULL) {
        adj2 = (pl->G).get_adj(); inv2 = adj2.shortest_path();
        if (iso_P(G, inv1, pl->G, inv2) return -1;
        pl = pl->next;
    }
    return 0;
}
int treeinsert(node *x, bigdigit digit1, bits G)
{
    node *p;
    patternlist *pl;
    adj_matrix adj2;
    invariant inv2;
    bigdigit digit2;
    z->G = G;
    do {
        adj2 = (x->G).get_adj(); inv2 = adj2.shortest_path();
        digit2 = inv2.get_digit();
        if (digit2 == digit1) break;
```

```
        p = x;
        if (digit2 > digit1) x = x->l;
        else                 x = x->r;
    } while (1);
    if (x == z) {
        if ((x = new(node)) == NULL) return 0;
        x->G = G; x->l = z; x->r = z;
        if ((x->list = new(patternlist)) == NULL) return 0;
        x->list->G = G; x->list->next = NULL;
        adj2 = (p->G).get_adj(); inv2 = adj2.shortest_path();
        digit2 = inv2.get_digit();
        if (digit2 > digit1) p->l = x;
        else                 p->r = x;
        return -1;
    }
    else {
        if ((pl = new(patternlist)) == NULL) return 0;
        pl->G = G; pl->next = x->list; x->list = pl;
        return -1;
    }
}
int new_P(bits G)
{
    adj_matrix adj = G.get_adj();
    invariant inv = adj.shortest_path();
    bigdigit digit = inv.get_digit();
    if (treesearch(head, digit, inv, G) == -1) return 0;
    if (treeinsert(head, digit, G) == 0) {
        printf("\nnew() overflow!!\n"); exit(1);
    }
    return -1;
}
void graph(bits G, bits Rest)
{
    bits NewG, NewRest;
    int i, j;
    adj_list A;
    for (i=0; i<alledges.len; i++) {
        if (!Rest.get(i)) continue;
        NewG = G; NewG.set(i);
        if (new_P(NewG)) {
            A = NewG.get_a_list();
```

```
            if (A.planarP()) {
                if (NewG.connectedP()) {
                    printf("%4d : ", ++number); NewG.put();
                }
                NewRest = Rest;
                for (j=0; j<=i; j++) NewRest.unset(j);
                graph(NewG, NewRest);
            }
        }
    }
}
main(int argc, char *argv[])
{
    bits G, Rest;
    if (argc == 1) {
        printf("usage: planar <number>\n"); exit(0);
    }
    n_vertex = atoi(argv[1]);
    if (n_vertex > maxvertices) {
        printf("vertices are too large!\n"); exit(0);
    }
    set_alledges(n_vertex);
    Rest = initial(alledges.len);
    treeinitialize(n_vertex);
    number = 0;
    graph(G, Rest);
}
```

References

1. Ronald GOULD. *Graph theory*. Benjamin Cummings, Menlo Park, Calif., 1988
2. Brain W. KERNIGHAM and Dennis M. RITCHIE. プログラム言語C第2版共立出版 1989
3. Osamu NAKAMURA. *On the number of non-isomorphic simple connected planar graphs of order 7 and 8*, RESEARCH REPORTS of KOCHI UNIVERSITY, Vol.44, NATURAL SCIENCE, KOCHI, JAPAN 1995
4. R. SEDGEWICK. アルゴリズムC++近代科学社 1994
5. Steven S. SKIENA. Mathematica 組み合せ論とグラフ理論 トッパン 1992
6. Bjarne STROUSTRUP. プログラミング言語C++第2版 トッパン 1993
7. R. J. WILSON. グラフ理論入門 近代科学社 1985

Manuscript received:September 30,1996
Published: December 25,1996