```cpp
namespace mod_op {

  const ll MOD = // (ll)1e9 + 7;

    class modll {
    private:
      ll val;
      ll modify(ll x) const { ll ret = x % MOD; if (ret < 0) ret += MOD; return ret; }
      ll inv(ll x) const {
        if (x == 0) return 1 / x;
        else if (x == 1) return 1;
        else return modify(inv(MOD % x) * modify(-MOD / x));
      }
    public:
      modll(ll init = 0) { val = modify(init); return; }
      modll(const modll& another) { val = another.val; return; }
      modll& operator=(const modll &another) { val = another.val; return *this; }
      modll operator+(const modll &x) const { return modify(val + x.val); }
      modll operator-(const modll &x) const { return modify(val - x.val); }
      modll operator*(const modll &x) const { return modify(val * x.val); }
      modll operator/(const modll &x) const { return modify(val * inv(x.val)); }
      modll& operator+=(const modll &x) { val = modify(val + x.val); return *this; }
      modll& operator-=(const modll &x) { val = modify(val - x.val); return *this; }
      modll& operator*=(const modll &x) { val = modify(val * x.val); return *this; }
      modll& operator/=(const modll &x) { val = modify(val * inv(x.val)); return *this; }
      bool operator==(const modll &x) { return val == x.val; }
      bool operator!=(const modll &x) { return val != x.val; }
      friend istream& operator >> (istream &is, modll& x) { is >> x.val; return is; }
      friend ostream& operator << (ostream &os, const modll& x) { os << x.val; return os; }
      ll get_val() { return val; }
  };

  modll pow(modll n, ll p) {
    modll ret;
    if (p == 0) ret = 1;
    else if (p == 1) ret = n;
    else {
      ret = pow(n, p / 2);
      ret *= ret;
      if (p % 2 == 1) ret *= n;
    }
    return ret;
  }

  vector<modll> facts;

  void make_facts(int n) {
    if (facts.empty()) facts.push_back(modll(1));
    for (int i = (int)facts.size(); i <= n; ++i) facts.push_back(modll(facts.back() * (ll)i));
    return;
  }

  vector<modll> ifacts;
  vector<modll> invs;

  void make_invs(int n) {
    if (invs.empty()) {
      invs.push_back(modll(0));
      invs.push_back(modll(1));
    }
    for (int i = (int)invs.size(); i <= n; ++i) {
      // because 0 = MOD = kq + r, 1/k = -q/r
      invs.push_back(invs[(int)MOD % i] * ((int)MOD - (int)MOD / i));
    }
    return;
  }

  void make_ifacts(int n) {
    make_invs(n);
    if (ifacts.empty()) ifacts.push_back(modll(1));
    for (int i = (int)ifacts.size(); i <= n; ++i) ifacts.push_back(modll(ifacts.back() * invs[i]));
```

```
72      return;
73    }
74
75    //nCr
76    modll combination(ll n, ll r) {
77      if (n >= r && r >= 0) {
78        modll ret;
79        make_facts((int)n);
80        make_ifacts((int)n);
81        ret = facts[(unsigned)n] * ifacts[(unsigned)r] * ifacts[(unsigned)(n - r)];
82        return ret;
83      }
84      else return 0;
85    }
86
87    modll get_fact(ll n) {
88      make_facts((int)n);
89      return facts[(int)n];
90    }
91
92    modll get_ifact(ll n) {
93      make_ifacts((int)n);
94      return ifacts[(int)n];
95    }
96
97    vector<vector<modll>> Stirling_nums2;
98    vector<vector<modll>> Stirling_nums2_sum;
99
100   void make_Stirling_nums2(int n) {
101     for (int i = (int)Stirling_nums2.size(); i <= n; ++i) {
102       Stirling_nums2.push_back(vector<modll>(i + 1));
103       Stirling_nums2_sum.push_back(vector<modll>(i + 1, 0));
104       Loop(j, i + 1) {
105         if (j == 0) Stirling_nums2[i][j] = 0;
106         else if (j == 1) Stirling_nums2[i][j] = 1;
107         else if (j == i) Stirling_nums2[i][j] = 1;
108         else Stirling_nums2[i][j] = Stirling_nums2[i - 1][j - 1] + Stirling_nums2[i - 1][j] * modll(j);
109         if (j > 0) Stirling_nums2_sum[i][j] = Stirling_nums2_sum[i][j - 1] + Stirling_nums2[i][j];
110       }
111     }
112   }
113
114   modll get_Stirling_num2(ll n, ll r) {
115     if (n >= r && r >= 0) {
116       make_Stirling_nums2((int)n);
117       return Stirling_nums2[(int)n][(int)r];
118     }
119     else return 0;
120   }
121
122   modll get_Stirling_num2_sum(ll n, ll r) {
123     if (n >= r && r >= 0) {
124       make_Stirling_nums2((int)n);
125       return Stirling_nums2_sum[(int)n][(int)r];
126     }
127     else return 0;
128   }
129
130   vector<vector<modll>> partition_nums;
131   vector<vector<modll>> partition_nums_sum;
132
133   void make_partition_nums(int n) {
134     for (int i = (int)partition_nums.size(); i <= n; ++i) {
135       partition_nums.push_back(vector<modll>(i + 1));
136       partition_nums_sum.push_back(vector<modll>(i + 1, 0));
137       Loop(j, i + 1) {
138         if (j == 0) partition_nums[i][j] = 0;
139         else if (j == 1) partition_nums[i][j] = 1;
140         else if (j == i) partition_nums[i][j] = 1;
141         else partition_nums[i][j] = partition_nums[i - 1][j - 1] + (i >= j * 2 ? partition_nums[i - j]
            [j] : 0);
```

```
142         if (j > 0) partition_nums_sum[i][j] = partition_nums_sum[i][j - 1] + partition_nums[i][j];
143       }
144     }
145   }
146   modll get_partition_num(ll n, ll r) {
147     if (n >= r && n >= 0) {
148       make_partition_nums((int)n);
149       return partition_nums[(int)n][(int)r];
150     }
151     else return 0;
152   }
153
154   modll get_partition_num_sum(ll n, ll r) {
155     if (n >= r && r >= 0) {
156       make_partition_nums((int)n);
157       return partition_nums_sum[(int)n][(int)r];
158     }
159     else return 0;
160   }
161
162   //log_a(b), if x does not exist, return -1
163   ll disc_log(modll a, modll b) {
164     ll ret = -1;
165     ll m = ceilsqrt(MOD);
166     unordered_map<ll, ll> mp;
167     modll x = 1;
168     Loop(i, m) {
169       mp[x.get_val()] = i;
170       x *= a;
171     }
172     x = modll(1) / pow(a, m);
173     modll k = b;
174     Loop(i, m) {
175       if (mp.find(k.get_val()) == mp.end()) k *= x;
176       else {
177         ret = i * m + mp[k.get_val()];
178         break;
179       }
180     }
181     return ret;
182   }
183 }
184
185 using namespace mod_op;
186 typedef vector<modll> vmodll;
187 typedef vector<vector<modll>> vvmodll;
188
189 // the number of methods of dividing n factors into r groups
190 // recommend to consider corner case (n == 0 or r == 0) irregularly
191 modll grouping(ll n, ll r, bool distinct_n, bool distinct_r, bool enable_empty_r) {
192   int mode = (distinct_n ? 0b100 : 0) + (distinct_r ? 0b010 : 0) + (enable_empty_r ? 0b001 : 0);
193   if (n < 0 || r < 0) return 0;
194   switch (mode) {
195   case 0b000:
196     return get_partition_num(n, r);
197   case 0b001:
198     return get_partition_num_sum(n, r);
199   case 0b010:
200     return combination(n - 1, r - 1);
201   case 0b011:
202     return combination(n + r - 1, r - 1);
203   case 0b100:
204     return get_Stirling_num2(n, r);
205   case 0b101:
206     return get_Stirling_num2_sum(n, r);
207   case 0b110:
208     return get_Stirling_num2(n, r) * get_fact(r);
209   case 0b111:
210     return pow(modll(r), n);
211   default:
212     return 0;
```

```
  }
}

using namespace mod_op;
using vmodll = vector<modll>;
using vvmodll = vector<vmodll>;
using vvvmodll = vector<vvmodll>;
```