```cpp
template<typename val_t>
class Partial_Permutation {
private:
  int n;
  vector<bool> used;
  vector<vector<val_t>> result;
  vvi facts; // iPj
  void core_func(const vector<val_t> &a, int n, int r, int start) {
    if (r == 0 || n < r) return;
    int m = facts[n - 1][r - 1];
    int cnt = 0;
    Loop(i, Partial_Permutation::n) {
      if (!used[i]) {
        Loop(j, m) {
          result[start + m * cnt + j].push_back(a[i]);
        }
        used[i] = true;
        core_func(a, n - 1, r - 1, start + m * cnt);
        used[i] = false;
        cnt++;
      }
    }
  }
  void make_facts(int n) {
    facts = vvi(n + 1, vi(n + 1));
    Loop(i, n + 1) {
      facts[i][0] = 1;
      Loop(j, i) {
        facts[i][j + 1] = facts[i][j] * (i - j);
      }
    }
  }
public:
  vector<vector<val_t>> get_partial_permutation(const vector<val_t> &a, int r) {
    n = int(a.size());
    if (n < r) return {};
    used = vector<bool>(n, false);
    make_facts(n);
    result = vector<vector<val_t>>(facts[n][r]);
    core_func(a, n, r, 0);
    return result;
  }
};


class Partial_Permutation_String {
private:
  int n;
  string a;
  vector<bool> used;
  vector<string> result;
  vvi facts; // iPj
  void core_func(const string &a, int n, int r, int start) {
    if (r == 0 || n < r) return;
    int m = facts[n - 1][r - 1];
    int cnt = 0;
    Loop(i, Partial_Permutation_String::n) {
      if (!used[i]) {
        Loop(j, m) {
          result[start + m * cnt + j] += a[i];
        }
        used[i] = true;
        core_func(a, n - 1, r - 1, start + m * cnt);
        used[i] = false;
        cnt++;
      }
    }
  }
  void make_facts(int n) {
    facts = vvi(n + 1, vi(n + 1));
    Loop(i, n + 1) {
```

```
72        facts[i][0] = 1;
73        Loop(j, i) {
74          facts[i][j + 1] = facts[i][j] * (i - j);
75        }
76      }
77    }
78  public:
79    vector<string> get_partial_permutation(const string &a, int r) {
80      n = int(a.size());
81      if (n < r) return {};
82      used = vector<bool>(n, false);
83      make_facts(n);
84      result = vector<string>(facts[n][r]);
85      core_func(a, n, r, 0);
86      return result;
87    }
88  };
```