常用例题

逆序对（归并排序解）

> 性质：交换序列的任意两元素，序列的逆序数的奇偶性必定发生改变。

```
LL a[N], tmp[N], n, ans = 0;
void mergeSort(LL l, LL r){
    if (l >= r) return;
    LL mid = (l + r) >> 1, i = l, j = mid + 1, cnt = 0;
    mergeSort(l, mid);
    mergeSort(mid + 1, r);
    while (i <= mid || j <= r)
        if (j > r || (i <= mid && a[i] <= a[j]))
            tmp[cnt++] = a[i++];
        else
            tmp[cnt++] = a[j++], ans += mid - i + 1;
    for (LL k = 0; k < r - l + 1; k++)
        a[l + k] = tmp[k];
}
int main(){
    cin >> n;
    for (int i = 1; i <= n; i++)
        scanf("%lld", &a[i]);
    mergeSort(1, n);
    cout << ans << "\n";
    return 0;
}
```

统计区间不同数字的数量（离线查询）

核心在于使用 `pre` 数组滚动维护每一个数字出现的最后位置，配以树状数组统计数量。由于滚动维护具有后效性，所以需要离线操作，从前往后更新。时间复杂度 $\mathcal{O}(N \log N)$ ，常数瓶颈在于 `map` ，用手造哈希或者离散化可以优化到理想区间；同时也有莫队做法，复杂度稍劣。例题链接 。

```
signed main() {
    int n;
    cin >> n;
    vector<int> in(n + 1);
    for (int i = 1; i <= n; i++) {
        cin >> in[i];
    }

    int q;
    cin >> q;
    vector<array<int, 3>> query;
    for (int i = 0; i < q; i++) {
        int l, r;
        cin >> l >> r;
        query.push_back({r, l, i});
```

```
    }
    sort(query.begin(), query.end());

    vector<pair<int, int>> ans;
    map<int, int> pre;
    int st = 1;
    BIT bit(n);
    for (auto [r, l, id] : query) {
        for (int i = st; i <= r; i++, st++) {
            if (pre.count(in[i])) { // 消除此前操作的影响
                bit.add(pre[in[i]], -1);
            }
            bit.add(i, 1);
            pre[in[i]] = i; // 更新操作
        }
        ans.push_back({id, bit.ask(r) - bit.ask(l - 1)});
    }

    sort(ans.begin(), ans.end());
    for (auto [id, w] : ans) {
        cout << w << endl;
    }
}
```

选数（DFS 解）

从 N 个整数中任选 K 个整数相加。使用 DFS 求解。

```
int n, k; cin >> n >> k;
vector<int> in(n), now(n);
for (auto &it : in) { cin >> it; }
auto dfs = [&](auto self, int k, int bit, int idx) -> void {
    for (int i = idx; i < n; i++) {
        now[bit] = in[i];
        if (bit < k - 1) { self(self, k, bit + 1, i + 1); }
        if (bit == k - 1) {
            int add = 0;
            for (int j = 0; j < k; j++) {
                add += now[j];
            }
            cout << add << endl;
        }
    }
};
dfs(dfs, k, 0, 0);
```

选数（位运算状压）

```cpp
int n, k; cin >> n >> k;
vector<int> in(n);
for (auto &it : in) { cin >> it; }
int comb = (1 << k) - 1, U = 1 << n;
while (comb < U) {
    int add = 0;
    for (int i = 0; i < n; i++) {
        if (1 << i & comb) {
            add += in[i];
        }
    }
    cout << add << "\n";

    int x = comb & -comb;
    int y = comb + x;
    int z = comb & ~y;
    comb = (z / x >> 1) | y;
}
```

网格路径计数

从 $(0,0)$ 走到 (a,b)，规定每次只能从 (x,y) 走到左下或者右下，方案数记为 $f(a,b)$。

- $f(a,b)=\begin{pmatrix} a \\ \frac{a+b}{2} \end{pmatrix}$；
- 若路径和直线 $y=k, k\notin[0,b]$ 不能有交点，则方案数为 $f(a,b)-f(a,2k-b)$；
- 若路径和两条直线 $y=k_1$，$y=k_2$（$k_1<0\le b<k_2$）不能有交点，方案数记为 $g(a,b,k_1,k_2)$，可以使用 $\mathcal{O}(N)$ 递归求解；
- 若路径必须碰到 $y=k_1$ 但是不能碰到 $y=k_2$，方案数记为 $h(a,b,k_1,k_2)$，可以使用 $\mathcal{O}(N)$ 递归求解（递归过程中两条直线距离会越来越大）。

从 $(0,0)$ 走到 $(a,0)$，规定每次只能走到左下或者右下，且必须有**恰好一次**传送（向下 b 单位），且不能走到 x 轴下方，方案数为 $\begin{pmatrix} a+1 \\ \frac{a-b}{2}+k+1 \end{pmatrix}$。

德州扑克

读入牌型，并且支持两副牌之间的大小比较。[代码参考]

```cpp
struct card {
      int suit, rank;
      friend bool operator < (const card &a, const card &b) {
        return a.rank < b.rank;
    }
    friend bool operator == (const card &a, const card &b) {
        return a.rank == b.rank;
    }
    friend bool operator != (const card &a, const card &b) {
        return a.rank != b.rank;
```

```
    }
    friend auto &operator>> (istream &it, card &C) {
        string S, T; it >> S;
        T = "__23456789TJQKA"; //点数
        FOR (i, 0, T.sz - 1) {
            if (T[i] == S[0]) C.rank = i;
        }
        T = "_SHCD"; //花色
        FOR (i, 0, T.sz - 1) {
            if (T[i] == S[1]) C.suit = i;
        }
        return it;
    }
};
struct game {
    int level;
    vector<card> peo;
    int a, b, c, d, e;
    int u, v, w, x, y;
    bool Rk10() { //Rk10: Royal Flush，五张牌同花色，且点数为AKQJT（14,13,12,11,10）
        sort(ALL(peo));
        reverse(ALL(peo));
        a = peo[0].rank, b = peo[1].rank, c = peo[2].rank, d = peo[3].rank, e = peo[4].rank;
        u = peo[0].suit, v = peo[1].suit, w = peo[2].suit, x = peo[3].suit, y = peo[4].suit;

        if (u != v || v != w || w != x || x != y) return 0;
        if (a == 14 && b == 13 && c == 12 && d == 11 && e == 10) return 1;
        return 0;
    }
    bool Dif(vector<card> &peo) { //专门用于检查A2345这种顺子的情况（这是最小的顺子）
        a = peo[0].rank, b = peo[1].rank, c = peo[2].rank, d = peo[3].rank, e = peo[4].rank;
        u = peo[0].suit, v = peo[1].suit, w = peo[2].suit, x = peo[3].suit, y = peo[4].suit;

        if (a != 14 || b != 5 || c != 4 || d != 3 || e != 2) return 0;
        vector<card> peo2 = {peo[1], peo[2], peo[3], peo[4], peo[0]}; //重新排序
        peo = peo2;
        return 1;
    }
    bool Rk9() { //Rk9: Straight Flush，五张牌同花色，且顺连【r1 > r2 > r3 > r4 > r5】
        sort(ALL(peo));
        reverse(ALL(peo));
        a = peo[0].rank, b = peo[1].rank, c = peo[2].rank, d = peo[3].rank, e = peo[4].rank;
        u = peo[0].suit, v = peo[1].suit, w = peo[2].suit, x = peo[3].suit, y = peo[4].suit;

        if (u != v || v != w || w != x || x != y) return 0;
        if (Dif(peo)) return 1; //特判：A2345
        if (a == b + 1 && b == c + 1 && c == d + 1 && d == e + 1) return 1;
```

```
58          return 0;
59      }
60      bool Rk8() { //Rk8: Four of a Kind，四张牌点数一样【r1 = r2 = r3 = r4】
61          sort(ALL(peo));
62          a = peo[0].rank, b = peo[1].rank, c = peo[2].rank, d = peo[3].rank, e =
    peo[4].rank;
63          u = peo[0].suit, v = peo[1].suit, w = peo[2].suit, x = peo[3].suit, y =
    peo[4].suit;
64
65          if (a == b && b == c && c == d) return 1;
66          if (b == c && c == d && d == e) {
67              reverse(ALL(peo));
68              return 1;
69          }
70          return 0;
71      }
72      bool Rk7() { //Rk7: Fullhouse，三张牌点数一样，另外两张点数也一样【r1 = r2 = r3，r4 =
    r5】
73          sort(ALL(peo));
74          a = peo[0].rank, b = peo[1].rank, c = peo[2].rank, d = peo[3].rank, e =
    peo[4].rank;
75          u = peo[0].suit, v = peo[1].suit, w = peo[2].suit, x = peo[3].suit, y =
    peo[4].suit;
76
77          if (a == b && b == c && d == e) return 1;
78          if (a == b && c == d && d == e) {
79              reverse(ALL(peo));
80              return 1;
81          }
82          return 0;
83      }
84      bool Rk6() { //Rk6: Flush，五张牌同花色【r1 > r2 > r3 > r4 > r5】
85          sort(ALL(peo));
86          reverse(ALL(peo));
87          a = peo[0].rank, b = peo[1].rank, c = peo[2].rank, d = peo[3].rank, e =
    peo[4].rank;
88          u = peo[0].suit, v = peo[1].suit, w = peo[2].suit, x = peo[3].suit, y =
    peo[4].suit;
89
90          if (u != v || v != w || w != x || x != y) return 0;
91          return 1;
92      }
93      bool Rk5() { //Rk5: Straight，五张牌顺连【r1 > r2 > r3 > r4 > r5】
94          sort(ALL(peo));
95          reverse(ALL(peo));
96          a = peo[0].rank, b = peo[1].rank, c = peo[2].rank, d = peo[3].rank, e =
    peo[4].rank;
97          u = peo[0].suit, v = peo[1].suit, w = peo[2].suit, x = peo[3].suit, y =
    peo[4].suit;
98
99          if (Dif(peo)) return 1; //特判：A2345
100         if (a == b + 1 && b == c + 1 && c == d + 1 && d == e + 1) return 1;
101         return 0;
```

```
    }
    bool Rk4() { //Rk4: Three of a kind，三张牌点数一样【r1 = r2 = r3，r4 > r5】
        sort(ALL(peo));
        reverse(ALL(peo));
        a = peo[0].rank, b = peo[1].rank, c = peo[2].rank, d = peo[3].rank, e = peo[4].rank;
        u = peo[0].suit, v = peo[1].suit, w = peo[2].suit, x = peo[3].suit, y = peo[4].suit;

        if (a == b && b == c) return 1;
        if (b == c && c == d) {
            swap(peo[3], peo[0]);
            return 1;
        }
        if (c == d && d == e) {
            swap(peo[3], peo[0]);
            swap(peo[4], peo[1]);
            return 1;
        }
        return 0;
    }
    bool Rk3() { //Rk3: Two Pairs，两张牌点数一样，另外有两张点数也一样（两个对子）【r1 = r2 > r3 = r4】
        sort(ALL(peo));
        reverse(ALL(peo));
        a = peo[0].rank, b = peo[1].rank, c = peo[2].rank, d = peo[3].rank, e = peo[4].rank;
        u = peo[0].suit, v = peo[1].suit, w = peo[2].suit, x = peo[3].suit, y = peo[4].suit;

        if (a == b && c == d) return 1;
        if (a == b && d == e) {
            swap(peo[2], peo[4]);
            return 1;
        }
        if (b == c && d == e) {
            swap(peo[0], peo[2]);
            swap(peo[2], peo[4]);
            return 1;
        }
        return 0;
    }
    bool Rk2() { //Rk2: One Pairs，两张牌点数一样（一个对子）【r1 = r2，r3 > r4 > r5】
        sort(ALL(peo));
        reverse(ALL(peo));
        a = peo[0].rank, b = peo[1].rank, c = peo[2].rank, d = peo[3].rank, e = peo[4].rank;
        u = peo[0].suit, v = peo[1].suit, w = peo[2].suit, x = peo[3].suit, y = peo[4].suit;

        vector<card> peo2;
        if (a == b) return 1;
        if (b == c) {
```

```
            peo2 = {peo[1], peo[2], peo[0], peo[3], peo[4]};
            peo = peo2;
            return 1;
        }
        if (c == d) {
            peo2 = {peo[2], peo[3], peo[0], peo[1], peo[4]};
            peo = peo2;
            return 1;
        }
        if (d == e) {
            peo2 = {peo[3], peo[4], peo[0], peo[1], peo[2]};
            peo = peo2;
            return 1;
        }
        return 0;
    }
    bool Rk1() { //Rk1: high card
        sort(ALL(peo));
        reverse(ALL(peo));
        return 1;
    }
    game (vector<card> New_peo) {
        peo = New_peo;
        if (Rk10()) { level = 10; return; }
        if (Rk9()) { level = 9; return; }
        if (Rk8()) { level = 8; return; }
        if (Rk7()) { level = 7; return; }
        if (Rk6()) { level = 6; return; }
        if (Rk5()) { level = 5; return; }
        if (Rk4()) { level = 4; return; }
        if (Rk3()) { level = 3; return; }
        if (Rk2()) { level = 2; return; }
        if (Rk1()) { level = 1; return; }
    }
    friend bool operator < (const game &a, const game &b) {
        if (a.level != b.level) return a.level < b.level;
        FOR (i, 0, 4) if (a.peo[i] != b.peo[i]) return a.peo[i] < b.peo[i];
        return 0;
    }
    friend bool operator == (const game &a, const game &b) {
        if (a.level != b.level) return 0;
        FOR (i, 0, 4) if (a.peo[i] != b.peo[i]) return 0;
        return 1;
    }
};
void debug(vector<card> peo) {
    for (auto it : peo) cout << it.rank << " " << it.suit << "  ";
    cout << "\n\n";
}
int clac(vector<card> Ali, vector<card> Bob) {
    game atype(Ali), btype(Bob);
    if (atype < btype) return -1;
    else if (atype == btype) return 0;
```

```
    return 1;
}
```

N*M 数独字典序最小方案

规则：每个宫大小为 $2^N * 2^M$，大图一共由 $M * N$ 个宫组成（总大小即 $2^N 2^M * 2^N 2^M$），要求每行、每列、每宫都要出现 1 到 $2^N * 2^M$ 的全部数字。输出字典序最小方案。

下例为 $2, 1$ 和 $1, 2$ 时数独字典序最小的示意。

	0	1	2	3	4	5	6	7
0	1	2	3	4	5	6	7	8
1	3	4	1	2	7	8	5	6
2	5	6	7	8	1	2	3	4
3	7	8	5	6	3	4	1	2
4	2	1	4	3	6	5	8	7
5	4	3	2	1	8	7	6	5
6	6	5	8	7	2	1	4	3
7	8	7	6	5	4	3	2	1

	0	1	2	3	4	5	6	7
0	1	2	3	4	5	6	7	8
1	5	6	7	8	1	2	3	4
2	2	1	4	3	6	5	8	7
3	6	5	8	7	2	1	4	3
4	3	4	1	2	7	8	5	6
5	7	8	5	6	3	4	1	2
6	4	3	2	1	8	7	6	5
7	8	7	6	5	4	3	2	1

公式：(i, j) 格所填的内容为 $\left(i \bmod 2^N \oplus \left\lfloor \frac{j}{2^M} \right\rfloor\right) \cdot 2^M + \left(\left\lfloor \frac{i}{2^N} \right\rfloor \oplus j \bmod 2^M\right) + 1$，注意 i, j 从 0 开始。

高精度进制转换

$2 - 62$ 进制相互转换。输入格式："转换前进制 转换后进制 要转换的数据"。注释：进制排序为 0-9，A-Z，a-z。

```
#include<bits/stdc++.h>
using namespace std;
map<char, int> mp; //将字符转化为数字
map<int, char> mp2; //将数字转化为字符
int main(){
    for(int i = 0; i < 10; i++) mp[(char)i + 48] = i, mp2[i] = (char)i + 48;
    for(int i = 10; i < 36; i++) mp[(char)i + 55] = i, mp2[i] = (char)i + 55;
    for(int i = 36; i < 62; i++) mp[(char)i + 61] = i, mp2[i] = (char)i + 61;

    int tt = 1, a, b; cin >> tt;
    while(tt--){
        string s, sh;
        vector<int> nums, ans;
        cin >> a >> b >> s;
        for(auto c : s) nums.push_back(mp[c]);
        reverse(nums.begin(), nums.end());
        while(nums.size()){ //短除法，将整个大数一直除 b ，取余数
            int remainder = 0;
            for(int i = nums.size() - 1; ~i; i--){
```

```
                nums[i] += remainder * a;
                remainder = nums[i] % b;
                nums[i] /= b;
            }
            ans.push_back(remainder); //得到余数
            while(nums.size() && nums.back() == 0) nums.pop_back(); //去掉前导 0
        }
        reverse(ans.begin(), ans.end());
        for(int i : ans) sh += mp2[i];
        cout << a << ' ' << s << endl;
        cout << b << ' ' << sh << endl << endl;
    }
    return 0;
}
```

物品装箱

有 N 个物品，第 i 个物品为 $a[i]$ ，有无限个容量为 C 的空箱子。两种装箱方式，输出需要多少个箱子才能装完所有物品。

从前往后装（线段树解）

xxxxxxxxxx47 1template vector<Point> halfcut(vector<Line> lines) {2 sort(lines.begin(), lines.end(), & {3 auto d1 = l1.b - l1.a;4 auto d2 = l2.b - l2.a;5 if (sign(d1) != sign(d2)) {6 return sign(d1) == 1;7 }8 return cross(d1, d2) > 0;9 });10 deque<Line> ls;11 deque<Point> ps;12 for (auto l : lines) {13 if (ls.empty()) {14 ls.push_back(l);15 continue;16 }17 while (!ps.empty() && !pointOnLineLeft(ps.back(), l)) {18 ps.pop_back();19 ls.pop_back();20 }21 while (!ps.empty() && !pointOnLineLeft(ps[0], l)) {22 ps.pop_front();23 ls.pop_front();24 }25 if (cross(l.b - l.a, ls.back().b - ls.back().a) == 0) {26 if (dot(l.b - l.a, ls.back().b - ls.back().a) > 0) {27 if (!pointOnLineLeft(ls.back().a, l)) {28 assert(ls.size() == 1);29 ls[0] = l;30 }31 continue;32 }33 return {};34 }35 ps.push_back(lineIntersection(ls.back(), l));36 ls.push_back(l);37 }38 while (!ps.empty() && !pointOnLineLeft(ps.back(), ls[0])) {39 ps.pop_back();40 ls.pop_back();41 }42 if (ls.size() <= 2) {43 return {};44 }45 ps.push_back(lineIntersection(ls[0], ls.back()));46 return vector(ps.begin(), ps.end());47}c++

```
const int N = 1e6 + 10;
int T, n, a[N], c, tr[N << 2];
void pushup(int u){
    tr[u] = max(tr[u << 1], tr[u << 1 | 1]);
}
void build(int u, int l, int r){
    if (l == r) tr[u] = c;
    else {
        int mid = l + r >> 1;
        build(u << 1, l, mid);
        build(u << 1 | 1, mid + 1, r);
        pushup(u);
    }
}
void update(int u, int l, int r, int p, int k){
    if (l > p || r < p) return;
```

```
    if (l == r) tr[u] -= k;
    else {
        int mid = l + r >> 1;
        update(u << 1, l, mid, p, k);
        update(u << 1 | 1, mid + 1, r, p, k);
        pushup(u);
    }
}
int query(int u, int l, int r, int k){
    if (l == r){
        if (tr[u] >= k) return l;
        return n + 1;
    }
    int mid = l + r >> 1;
    if (tr[u << 1] >= k) return query(u << 1, l, mid, k);
    else return query(u << 1 | 1, mid + 1, r, k);
}
int main() {
    cin >> n >> c;
    for (int i = 1; i <= n; i++) cin >> a[i];
    build(1, 1, n);
    for (int i = 1; i <= n; i++)
        update(1, 1, n, query(1, 1, n, a[i]), a[i]);
    cout << query(1, 1, n, c) - 1 << " ";
}
```

选择最优的箱子装（multiset 解）

选择能放下物品且剩余容量最小的箱子放物品

```
void solve(){
    cin >> n >> c;
    for (int i = 1; i <= n; i++) cin >> a[i];
    multiset <int> s;
    for (int i = 1; i <= n; i++){
        auto it = s.lower_bound(a[i]);
        if (it == s.end()) s.insert(c - a[i]);
        else {
            int x = *it;
            // multiset 可以存放重复数据，如果是删除某个值的话，会去掉多个箱子
            // 导致答案错误，所以直接删除对应位置的元素
            s.erase(it);
            s.insert(x - a[i]);
        }
    }
    cout << s.size() << "\n";
}
```

浮点数比较

比较下列浮点数的大小： $x^{yz}, x^{zy}, (x^y)^z, (x^z)^y, y^{xz}, y^{zx}, (y^x)^z, (y^z)^x, z^{xy}, z^{yx}, (z^x)^y$ 和 $(z^y)^x$ 。

```cpp
vector<pair<ld, int>> val = {
    {log(x) * pow(y, z), 0}, {log(x) * pow(z, y), 1}, {log(x) * y * z, 2},
    {log(x) * z * y, 3},     {log(y) * pow(x, z), 4}, {log(y) * pow(z, x), 5},
    {log(y) * x * z, 6},     {log(y) * z * x, 7},     {log(z) * pow(x, y), 8},
    {log(z) * pow(y, x), 9}, {log(z) * x * y, 10},    {log(z) * y * x, 11}};

sort(val.begin(), val.end(), [&](auto x, auto y) {
    if (equal(x.first, y.first)) return x.second < y.second; // queal比较两个浮点数是否相等
    return x.first > y.first;
});
cout << ans[val.front().second] << endl;
```

/END/