基础算法

LIS 最长上升子序列 O(nlogn)

Dilworth:对于任意有限偏序集，其最大反链中元素的数目必等于最小链划分中链的数目.
将一个序列剖成若干个单调不升子序列的最小个数等于该序列最长上升子序列的个数

```
std::vector<int> f(n);
int len = 0;
f[0] = 0;
for (int i = 0;i < n;i++)
{
    int l = 0, r = len, res = 0;
    while (l <= r)
    {
        int mid = l + r >> 1;
        if (f[mid] < nums[i]) {
            res = std::max(res, mid);
            l = mid + 1;
        }
        else r = mid - 1;
    }
    res++;
    len = std::max(len, res);
    f[res] = nums[i];
}
```

二分

```
auto l = l, r = r;
auto check = [&](auto x)->bool {

    };
while (l < r) {
    auto mid = l + r >> 1;
    if (check(mid))r = mid;
    else l = mid + 1;
}
```

整体二分

```
int cal(auto x) {
    // todo
}

void solve(int ql, int qr, int l, int r) {
    if (ql > qr)return;
    if (l > r)return;
    if (l == r) {
        for (int q = ql;q <= qr;++q)ans[q] = l;
        return;
    }
    int mid = l + r + 1 >> 1;
    int cnt = cal(mid);
    solve(std::max(ql, cnt + 1), qr, l, mid - 1);
    solve(ql, std::min(qr, cnt), mid, r);
}

void solve() {
    //input
    solve(ql, qr, 0, n, zf);
    //todo
}

signed main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(0), std::cout.tie(0);
    int t;std::cin >> t;
    while (t--) {
        solve();
        std::cout << '\n';
    }
    return 0;
}
```

三分

```
while (r - l > eps) {
  mid = (l + r) / 2;
  lmid = mid - eps;
  rmid = mid + eps;
  if (f(lmid) < f(rmid))
    r = mid;
  else
    l = mid;
}
```

单调队列优化多重背包 O(NC)

```cpp
int maxValue(int N, int C, std::vector<int> s, std::vector<int> v, std::vector<int> w) {
    std::vector<int> dp(C + 1, 0);
    std::vector<int> g(C + 1, 0);// 辅助队列，记录的是上一次的结果
    std::vector<int> q(C + 1, 0);// 主队列，记录的是本次的结果

    // 枚举物品
    for (int i = 0; i < N; i++) {
        int vi = v[i];
        int wi = w[i];
        int si = s[i];

        // 将上次算的结果存入辅助数组中
        g = dp;

        // 枚举余数
        for (int j = 0; j < vi; j++) {
            // 初始化队列，head 和 tail 分别指向队列头部和尾部
            int head = 0, tail = -1;
            // 枚举同一余数情况下，有多少种方案。
            // 例如余数为 1 的情况下有：1、vi + 1、2 * vi + 1、3 * vi + 1 ...
            for (int k = j; k <= C; k += vi) {
                dp[k] = g[k];
                // 将不在窗口范围内的值弹出
                if (head <= tail && q[head] < k - si * vi) head++;
                // 如果队列中存在元素，直接使用队头来更新
                if (head <= tail) dp[k] = std::max(dp[k], g[q[head]] + (k - q[head]) / vi * wi);
                // 当前值比对尾值更优，队尾元素没有存在必要，队尾出队
                while (head <= tail && g[q[tail]] - (q[tail] - j) / vi * wi <= g[k] - (k - j) /
                // 将新下标入队
                q[++tail] = k;
            }
        }
    }
    return dp[C];
}
```

分解质因数

```
    for (int i = 2; i <= std::sqrt(num); ++i){
        while (num % i == 0){
            num /= i;
            // todo
        }
    }
    if (num != 1){
        // todo
    }
```

高精度加法

```
// C = A + B, A >= 0, B >= 0
vector<int> add(vector<int> &a,vector<int> &b){
    //c为答案
    vector<int> c;
    //t为进位
    int t=0;
    for(int i=0;i<a.size()||i<b.size();i++){
        //不超过a的范围添加a[i]
        if(i<a.size())t+=a[i];
        //不超过b的范围添加b[i]
        if(i<b.size())t+=b[i];
        //取当前位的答案
        c.push_back(t%10);
        //是否进位
        t/=10;
    }
    //如果t!=0的话向后添加1
    if(t)c.push_back(1);
    return c;
}
```

高精度减法

```
// C = A - B, 满足A >= B, A >= 0, B >= 0
vector<int> sub(vector<int> &A, vector<int> &B)
{
    //答案
    vector<int> C;
    //遍历最大的数
    for (int i = 0, t = 0; i < A.size(); i ++ )
    {
        //t为进位
        t = A[i] - t;
        //不超过B的范围t=A[i]-B[i]-t;
        if (i < B.size()) t -= B[i];
        //合二为一，取当前位的答案
        C.push_back((t + 10) % 10);
        //t<0则t=1
        if (t < 0) t = 1;
        //t>=0则t=0
        else t = 0;
    }
    //去除前导零
    while (C.size() > 1 && C.back() == 0) C.pop_back();
    return C;
}
```

高精度比大小（cmp函数）

```
//高精度比大小
bool cmp(vector<int> &A, vector<int> &B) {
        if (A.size() != B.size())
                return A.size() > B.size();
        for (int i = A.size() - 1; i >= 0; i -- )
                if (A[i] != B[i])
                        return A[i] > B[i];
        return true;
}
```

高精度乘低精度

```
// C = A * b, A >= 0, b >= 0
vector<int> mul(vector<int> &A, int b)
{
    //类似于高精度加法
    vector<int> C;
    //t为进位
    int t = 0;
    for (int i = 0; i < A.size() || t; i ++ )
    {
        //不超过A的范围t=t+A[i]*b
        if (i < A.size()) t += A[i] * b;
        //取当前位的答案
        C.push_back(t % 10);
        //进位
        t /= 10;
    }
    //去除前导零
    while (C.size() > 1 && C.back() == 0) C.pop_back();

    return C;
}
```

高精度乘高精度

高精度加减乘除：https://www.bilibili.com/video/BV1LA411v7mt/

```
vector<int> mul(vector<int> &A, vector<int> &B) {
    vector<int> C(A.size() + B.size()); // 初始化为 0，C的size可以大一点

    for (int i = 0; i < A.size(); i++)
        for (int j = 0; j < B.size(); j++)
            C[i + j] += A[i] * B[j];
    for (int i = 0,t = 0; i < C.size(); i++) { // i = C.size() - 1时 t 一定小于 10
        t += C[i];
        C[i] = t % 10;
        t /= 10;
    }

    while (C.size() > 1 && C.back() == 0) C.pop_back(); // 必须要去前导 0，因为最高位很可能是 0
    return C;
}
```

高精度除低精度

```
// A / b = C ... r, A >= 0, b > 0
vector<int> div(vector<int> &A, int b, int &r)//高精度A，低精度b，余数r
{
    vector<int> C;//答案
    r = 0;
    for (int i = A.size() - 1; i >= 0; i -- )
    {
        r = r * 10 + A[i];//补全r>=b
        C.push_back(r / b);//取当前位的答案
        r %= b;//r%b为下一次计算
    }
    reverse(C.begin(), C.end());//倒序为答案
    while (C.size() > 1 && C.back() == 0) C.pop_back();//去除前导零
    return C;
}
```

高精度除高精度

```
vector<int> div(vector<int> &A, vector<int> &B, vector<int> &r) {
        vector<int> C;
        if (!cmp(A, B)) {
                C.push_back(0);
                r.assign(A.begin(), A.end());
                return C;
        }
        int j = B.size();
        r.assign(A.end() - j, A.end());
        while (j <= A.size()) {
                int k = 0;
                while (cmp(r, B)) {
                        r = sub(r, B);
                        k ++;
                }
                C.push_back(k);
                if (j < A.size())
                        r.insert(r.begin(), A[A.size() - j - 1]);
                if (r.size() > 1 && r.back() == 0)
                        r.pop_back();
                j++;
        }
        reverse(C.begin(), C.end());
        while (C.size() > 1 && C.back() == 0)
                C.pop_back();
        return C;
}
```

一个贪心算法总是做出当前最好的选择，也就是说，它期望通过局部最优选择从而得到全局最优的解决方案。---**《算法导论》**