倍增算法

河南省实验中学信息技术组

2026 年 01 月 30 日

倍增

- 任意一个正整数 n 都能被 $2^0, 2^1, 2^2, 2^3, 2^4, \cdots$ 唯一表示，而且唯一表示中包含 2^k 表示说明 n 的二进制的第 $k+1$ 位是 1，例如数字 $5 = 2^2 + 2^0$ 且 5 的二进制为 101。
- 那么在很多问题中，可以对于询问数值为 n 的结果，可以预先处理出 $2^0, 2^1, 2^2, 2^3, 2^4, \cdots$ 的结果，然后用这些结果拼凑出数值为 n 的结果，使得单次询问的时间复杂度降为 $\mathrm{O}(\log N)$。

【例】快速幂

【题目描述】

给定整数 x, n，求 x^n 的值，这个值可能很大，你只需要输出其对 1000000007 取余的结果即可。

【输入格式】

一行三个整数 x, n，$0 \leq x \leq 2 \times 10^9, 0 \leq n \leq 10^{18}$。

【输出格式】

一行一个整数，表示 $x^n \mod 1000000007$ 的值。

【样例 1 输入】

```
2 10
```

【样例 1 输出】

```
1024
```

【样例 2 输入】

```
5 1000000000000000
```

【样例 2 输出】

```
829596975
```

【例】快速幂

- 将 x^n 的指数部分拆分成 2^j。
- 例如，$3^{11}=3^8\times 3^2\times 3^1=3^{2^3}\times 3^{2^1}\times 3^{2^0}=3^{2^3+2^1+2^0}=3^{1011}$。
- 倍增求出 $3^1,3^2,3^4,3^8,\cdots$，若指数对应二进制位为 1，则乘上对应的 3 的指数倍的数，否则不做任何处理。

```
long long fastpow(long long x, long long n)
{
    if(n == 0) return 1;
    long long ans = 1, base = x;
    while(n != 0)
    {
        if(n & 1) (ans *= base) %= M;
        (base *= base) %= M;
        n >>= 1;  // n = n / 2;
    }
    return ans;
}
```

$n(n>>1)$	$n\&1$	ans	$base$
		1	3^1
11—1011	1	3^1	3^2
5—0101	1	$3^1 3^2$	3^4
2—0010	0	$3^1 3^2$	3^8
1—0001	1	$3^1 3^2 3^8$	3^{16}
0—0000			

【例】快速幂

- 快速幂一般习惯使用递归算法实现，代码短而且易于理解。
- 对于高精度数的快速幂、矩阵快速幂等，递归实现一般需要定义数据类型，这时候使用倍增实现反而简单。

```
const int n = 100;
int base[2 * n + 10], ans[2 * n + 10] = {0, 1};
void mul(int a[], int b[]) // a*=b
{
    int c[N * 2 + 10] = { 0 };
    for(int j = 1; j <= N; ++j)
        for(int d = 0, i = 1; i <= N; ++i)
        {
            int k = c[i + j - 1] + a[i] * b[j] + d;
            c[i + j - 1] = k % 10, d = k / 10;
        }
    for(int i = 1; i <= N; ++i) a[i] = c[i];
}
// base=x 求 x^n
while(n)
{
    if(n & 1) mul(ans, base);
    mul(base, base);
    n >>= 1;
}
```

ST 表

- ST 表就是一种基于倍增思想解决区间重叠的数据结构，其中最经典的应用就是求区间最值问题 (RMQ)。
- 给定 n 个整数序列 $a_1, a_2, \cdots, a_n$，有 q 次询问，每次询问区间 $a_l, \cdots, a_r$ 的最大值/最小值。

【例】区间最大值

【题目描述】

给定 n 个整数序列 $a_1, a_2, \cdots, a_n$，有 q 次询问，每次询问区间 $a_l, \cdots, a_r$ 的最大值。

【输入格式】

第一行两个整数 $n, q(1 \le n \le 10^6, 1 \le q \le 10^5)$。

接下来一行 n 个整数 $(-10^9 \le a_i \le 10^9)$。接下来 q 行，每行两个整数 $l, r(1 \le l \le r \le n)$。

【输出格式】

共 q 行，对于每个询问输出一个整数表示区间的最大值。

【样例 1 输入】

```
8 5
1 3 -1 -3 5 3 6 7
1 8
1 7
5 8
1 5
3 4
```

【样例 1 输出】

```
7
6
7
5
-1
```

【例】区间最大值

- 定义 $f[i][j]$ 表示 $a[i:i+2^j-1]$ 的最大值，也就是从 $a[i]$ 开始长度为 2^j 的区间最大值。
- 显然有 $f[i][0]=a[i]$，对于任意的 $f[i][j]$，它表示的区间为 $[i:i+2^j-1]$，那么我们将其拆分为 $[i:i+2^{j-1}-1]$ 和 $[i+2^{j-1}:i+2^j-1]$，二者恰好为 $f[i][j-1]$ 和 $f[i+2^{j-1}][j-1]$，于是

$$f[i][j]=\max(f[i][j-1],f[i+2^{j-1}][j-1])$$

$f[i][j]$	0	1	2	3
1	1	3	3	7
2	3	3	5	
3	-1	-1	5	
4	-3	5	6	
5	5	5	7	
6	3	6		
7	6	7		
8	7			

【例】区间最大值

- 对于询问区间 $[l:r]$ 的最大值，有两种处理方法：
  1. 将区间拆分成若干个长度为 2^k 的小区间。例如 $[1:7]$ 拆分成 $[1:4],[5:6],[7:7]$，它们分别为 $f[1][2],f[5][1],f[7:7]$。
  2. 将区间 $[l:r]$ 拆分成两个重叠的区间 $[l:l+2^t-1]$ 和 $[r-2^t+1:r]$，其中 $t=\lfloor \log_2(r-l+1) \rfloor$。例如 $[1:7]$ 拆分成 $[1:4]$ 和 $[3:7]$，区间长度都为 $4=2^2$，二者恰好为 $f[1][2]$ 和 $f[3][2]$。

  实际中第二种方法是常使用的方法。
- 定义 $f[][]$ 一般第二维至少为 $\log N$，一般题目 $n \le 10^6$，一般定义到 20 即可。
- 预处理时间复杂度：$O(N\log N)$；单次询问的时间复杂度可以认为为：$O(1)$。

【例】区间最大值


```
const int T = 20;
int f[N][T + 5];

int qmax(int l, int r)
{
    int t = log2(r - l + 1);
    return max(f[l][t], f[r - (1 << t) + 1][t]);
}

// 预处理
for(int i = 1; i <= n; ++i) f[i][0]= a[i];
for(int j = 1; j <= T; ++j)
    for(int i = 1; i <= n - (1 << j) + 1; ++i)
        f[i][j] = max(f[i][j - 1], f[i + (1 << (j - 1))][j - 1]);
while(q--)
{
    int l, r; scanf("%d%d", &l, &r);
    printf("%d\n", qmax(l, r));
}
```

最近公共祖先

- 给定一颗有根树，若结点 z 既是结点 x 的祖先，也是结点 y 的祖先，则称 z 是 x, y 的公共祖先。在 x, y 的所有公共祖先中，深度最大的一个称为 x, y 的最近公共祖先 (Lowest Common Ancestor,LCA)，记为 $\mathrm{lca}(x, y)$
- 利用倍增法可以在线查询最近公共祖先：预处理时间复杂度为 $\mathrm{O}(N \log N)$，单次查询时间复杂度 $\mathrm{O}(\log N)$。

【例】接竹竿

小杨同学想用卡牌玩一种叫做“接竹竿”的游戏。

游戏规则是：每张牌上有一个点数 v，将给定的牌依次放入一列牌的末端。若放入之前这列牌中已有与这张牌点数相同的牌，则小杨同学会将这张牌和点数相同的牌之间的所有牌全部取出队列（包括这两张牌本身）。

小杨同学现在有一个长度为 n 的卡牌序列 A，其中每张牌的点数为 A_i $(1 \le i \le n)$。小杨同学有 q 次询问。第 i 次（$1 \le i \le q$）询问时，小杨同学会给出 l_i, r_i 小杨同学想知道如果用下标在 $[l_i, r_i]$ 的所有卡牌按照下标顺序玩“接竹竿”的游戏，最后队列中剩余的牌数。

【例】接竹竿

【输入格式】

一行包含一个正整数 T，表示测试数据组数。

对于每组测试数据，第一行包含一个正整数 n，表示卡牌序列 A 的长度。

第二行包含 n 个正整数 $A_1, A_2, \ldots, A_n$，表示卡牌的点数 A。

第三行包含一个正整数 q，表示询问次数。

接下来 q 行，每行两个正整数 l_i, r_i 表示一组询问。

【输出格式】

对于每组数据，输出 q 行。第 i 行（$1 \leq i \leq q$）输出一个非负整数，表示第 i 次询问的答案。

【数据范围】

$1 \leq T \leq 5$，$1 \leq n \leq 1.5 \times 10^4$，$1 \leq q \leq 1.5 \times 10^4$，$1 \leq A_i \leq 13$。

【例】接竹竿

【样例 1 输入】

```
1
6
1 2 2 3 1 3
4
1 3
1 6
1 5
5 6
```

【样例 1 输出】

```
1
1
0
2
```

【样例 1 解释】

对于第一次询问，小杨同学会按照 $1, 2, 2$ 的顺序放置卡牌，在放置最后一张卡牌时，两张点数为 2 的卡牌会被收走，因此最后队列中只剩余一张点数为 1 的卡牌。

对于第二次询问，队列变化情况为：

$\{\} \to \{1\} \to \{1,2\} \to \{1,2,2\} \to \{1\} \to \{1,3\} \to \{1,3,1\} \to \{\} \to \{3\}$。因此最后队列中只剩余一张点数为 3 的卡牌。

【例】接竹竿

- 对于每个询问，利用栈模拟接竹竿的过程。
- 时间复杂度：$O(TQN)$。

```
int top = 0;
for(int i = l; i <= r; ++i)
{
    int flag = 0;
    for(int j = top; j >= 1; --j)
        if(a[i] == st[j])
        {
            flag = 1;
            while(st[top] != a[i]) --top;
            --top;
            break;
        }
    if(!flag) st[++top] = a[i];
}
printf("%d\n", top);
```

【例】接竹竿

- 利用模拟来接竹竿速度慢的原因是每次都只消除一串，可以利用倍增法快速消除。
- 定义 $f[i][j]$ 表示从 $a[i]$ 开始消除 2^j 次能消除的位置。
- 然 $f[i][0]=a[i]$ 后面第一个与 $a[i]$ 相同的牌出现的位置，如果没有，则令其为 $n+1$。
- 显然 $f[i][j]=f[f[i][j-1]+1][j-1]$，要求 $f[i][j-1]+1\leq n$。
- 那么对于询问 $[l,r]$，令 $i=l$，表示当前尝试消除的牌。
- 首先看 $f[i][0]$，如果 $f[i][0]>r$，说明消除一次会越界，那么就令 i 加 1，当前牌无法消除，记录答案 $+1$。一直执行上述判定，直到牌全部用完或者当前牌可以消除。
- 如果当前牌可以消除，那么从大到小枚举消除的次数 2^j，如果 $f[i][j]\leq r$，那么就令 $i=f[i][j]+1$。
- 重复以上过程直到能消除的牌全部消除完。
- 时间复杂度：$\mathrm{O}(TQ\log N)$。

【例】接竹竿


```
for(int j = 0; j <= 20; ++j)
    for(int i = 1; i <= n; ++i)
        f[i][j] = n + 1;
for(int i = 1; i <= 13; ++i) p[i] = n + 1;  // p[i] 为第 i 种牌后面离它最近的牌
for(int i = n; i >= 1; --i) f[i][0] = p[a[i]], p[a[i]] = i;
for(int j = 1; j <= 20; ++j)
    for(int i = 1; i <= n; ++i)
        if(f[i][j - 1] + 1 <= n)
            f[i][j] = f[f[i][j - 1] + 1][j - 1];
// 询问
int ans = 0;
while(l <= r)
{
    while(l <= r && f[l][0] > r) ++l, ++ans;  // 消除一次越界
    if(l > r) break;
    for(int j = 20; j >= 0; --j)
        if(f[l][j] <= r) { l = f[l][j] + 1; break;}
}
```

【例】Genius ACM

Advanced CPU Manufacturer (ACM) is one of the best CPU manufacturers in the world. 每天，该公司生产 n 台 CPU 并销售到世界各地。

ACM 公司的质检部门会对生产出的 CPU 进行成组测试，对一组（若干个）CPU 进行测试的方法如下：

1. 随机从该组 CPU 中选取 m 对（即 $2m$ 台），若总数不足 $2m$ 台，则选取尽量多对。
2. 对于每一对 CPU，测量它们之间的 Relative Performance Difference (RPD)，并把第 i 对的 RPD 记为 D_i。RPD 的计算方法在后面给出。
3. 该组 CPU 的 Sqared Performance Difference (SPD）由以下公式给出：$SPD = \sum_i D_i^2$
4. 该组 CPU 通过质检，当且仅当 $SPD \leq k$, 其中 k 是给定常数。

ACM 公司生产的 CPU 性能很好，而质检部门制定的标准更是过于严格。通常他们把 n 台 CPU 作为一整组进行测试，这导致一些性能良好的 CPU 无法通过测试，生产部门对此颇有微词。作为质检部门的领导，小 S 在不更改质检测试流程的前提下，想出了这样一个主意：如果能够把 n 台 CPU 恰当地分成连续的若干段，使得每段 CPU 都能够通过成组测试，就可以解决当下的问题。

现在，小 S 已经知道了 n 台各自的性能表现 $P_1, \cdots, P_n$，两台 CPU 的 RPD 被定义为它们性能表现的差的绝对值。请你帮忙计算一下，至少把这些 CPU 分成多少段，才能使得每一段都能通过成组测试。

【例】Genius ACM

【输入格式】

每个测试点包含多组数据，第一行整数 $T(T \le 12)$ 给出数据组数。

对于每组数据，第一行三个整数 $n, m, k($，第二行 n 个整数 $P_1, \cdots, P_n$。

【输出格式】

对于每组数据，输出一个整表示答案。

【样例 1 输入】

```
2
5 1 49
8 2 1 7 9
5 1 64
8 2 1 7 9
```

【样例 1 输出】

```
2
1
```

【数据范围】

$T \le 12, 1 \le n, m \le 5 \times 10^5, 0 \le k \le 10^{18}, 0 \le P_i \le 2^{20}$

【例】Genius ACM

- 对于 n 个数的序列 a 以及整数 k，希望能把 a 分成若干段，使得每一段的 SPD 值都不超过 k 且分成的段数最少。
- SPD 值的计算为：从段中取 m 对数 (不重复，如果不够尽量取)，使得每对数差的平方的和尽可能的小。
- 对于一个序列，SPD 值最大的显然为最大的 m 个数和最小的 m 个数，分别按照最大最小配对。
- 为了让 a 分成的段数最少，显然每一段应该在 $SPD \leq k$ 的前提下尽量长。
- 于是问题转化为：对于一个左端点 x，右端点 $x \leq y \leq n$ 在满足 $SPD \leq k$ 的前提大，最大能取到多少。

【例】Genius ACM

- 对于一个左端点 x，在右端点 y 向右枚举时，SPD 值应该是单调递增的，故而可以考虑二分答案。
- 对于枚举的二分中间值 mid，求区间 $[x, mid]$ 的的最大 SPD 值，判断是否 $\leq k$。
- 时间复杂度：$\mathrm{O}(N^2 \log^2 N)$。

```
bool check(int x, int mid)
{
    for(int i = x; i <= mid; ++i) t[i] = a[i];
    sort(t + x, t + mid + 1);
    long long s = 0;
    int cnt = 0;
    for(int i = x, j = mid; i < j; ++i, --j)
    {
        s += (t[i] - t[j]) * (t[i] - t[j]);
        ++cnt;
        if(cnt == m) break;
    }
    return s <= k;
}
```

```
int ans = 0;
for(int x = 1; x <= n;)
{
    int l = x, r = n;
    while(l < r)
    {
        int mid = (l + r + 1) / 2;
        if(check(x, mid)) l = mid;
        else r = mid - 1;
    }
    ++ans;
    x = l + 1;
}
```

【例】Genius ACM

- 二分答案每一次二分的过程都需要排序，时间复杂度过高。
- 对于区间左端点 x，可以倍增的扩展区间右端点 y:
  - 首先，令右端点 $y = x$，令倍增区间长度 $p = 1$。
  - 求出 $[x, y + p]$ 这一段区间的 SPD 值并判断是否 $\leq k$，如果成立则令 $y+ = p, p* = 2$，否则令 $p/ = 2$。
  - 重复上一步知道 p 的值变为 0，此时 y 值即为所求。

```
int ans = 0;
for(int x = 1; x <= n;)
{
    int y = x, p = 1;
    b[x] = a[x];
    while(p)
    {
        if(y + p <= n && check(x, y, p)) y += p, p *= 2;
        else p /= 2;
    }
    ++ans;
    x = y + 1;
}
```

【例】Genius ACM

- 上述 check 函数中，如果对区间 $[x, y+p]$ 进行排序，那么总体的时间复杂度为 $\mathrm{O}(N\log^2 N)$。

```
bool check(int x, int y, int p)
{
    for(int i = x; i <= y + p; ++i) b[i] = a[i];
    sort(b + x, b + y + p + 1);
    long long s = 0;
    int cnt = 0;
    for(int i = x, j = y + p; i < j; ++i, --j)
    {
        s += (b[i] - b[j]) * (b[i] - b[j]);
        ++cnt;
        if(cnt == m) break;
    }
    return s <= k;
}
```

- 实际上每次对区间 $[x, y+p]$ 排序时，区间 $[x, y]$ 已经有序，那么只需要对 $[y+1, y+p]$ 排序，然后合并有序序列即可，时间复杂度可以降低到 $\mathrm{O}(N\log N)$。

【例】喷泉

大家都知道喷泉吧？现在有一个喷泉由 n 个圆盘组成，从上到下以此编号为 $1 \sim n$，第 i 个圆盘的直径为 d_i，容量为 c_i，当一个圆盘里的水大于了这个圆盘的容量，那么水就会溢出往下流，直到流入半径大于这个圆盘的圆盘里。如果下面没有满足要求的圆盘，水就会流到喷泉下的水池里。

现在给定 q 组询问，每一组询问这么描述：向第 r_i 个圆盘里倒入 v_i 的水，求水最后会流到哪一个圆盘停止。

如果最终流入了水池里，那么输出 0。

注意，每个询问互不影响。

【例】喷泉

【输入格式】
第一行两个整数 n, q 代表圆盘数和询问数。
接下来 n 行每行两个整数 d_i, c_i 代表一个圆盘。
接下来 q 行每行两个整数 r_i, v_i 代表一个询问。
【输出格式】
q 行每行一个整数代表询问的答案。
【数据范围】
$2 \le n \le 10^5, 1 \le q \le 2 \times 10^5, 1 \le C_i \le 1000, 1 \le D_i, V_i \le 10^9, 1 \le r_i \le n$。

【例】喷泉

倍增算法

河南省实验中学信息技术组

倍增

例题

快速幂

ST 表

最近公共祖先

接竹竿

Genius ACM

喷泉

练习

【样例 1 输入】

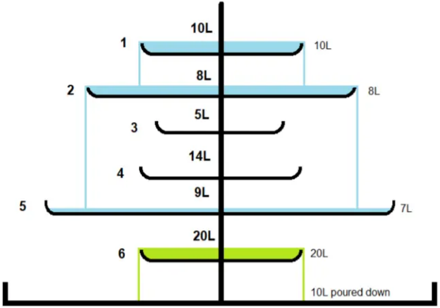
```
6 5
4 10
6 8
3 5
4 14
10 9
4 20
1 25
6 30
5 8
3 13
2 8
```

【样例 1 输出】

```
5
0
5
4
2
```

【样例 1 解释】

前两个询问的解释如下图所示：



【例】喷泉

- 不妨将水池看作直径和容量都无穷大的圆盘，编号为 $n+1$。
- 第 i 个圆盘中的水如果溢出会流到下面第一个直径 $> d_i$ 的圆盘。
- 由于圆盘的情况是固定的，所以每个圆盘的下一个直径 $> d_i$ 的圆盘可以预处理出来，不妨令其为 nxt_i。
- 对于第 i 个圆盘，从 i 到 $j(j>i)$ 的最大值是单调递增的，所以可以二分这个区间最大值，求出区间 $[i,j]$ 最大值 $> d_i$ 的最小位置。
- 区间最大值可以利用 ST 表解决，整体时间复杂度为：$\mathrm{O}(N\log^2 N)$。

```
for(int i = 1; i <= n; ++i)
{
    int l = i, r = n + 1;  // 找到大于 d[i] 的最小位置
    while(l < r)
    {
        int m = (l + r) / 2;
        if(qmax(l, m) <= d[i]) l = m + 1;
        else r = m;
    }
    nxt[i] = l;
}
```

【例】喷泉

- 对于每个询问，从 r_i 开始，如果水溢出，则考虑能流到的下一个圆盘，直到水流尽为止，时间复杂度：$O(NQ)$。
- 显然每一次溢出时只处理一个圆盘太慢，可以考虑用倍增解决。
- 定义 $f[i][j]$ 表示从第 i 个圆盘开始，水溢出 2^j 次所到达的圆盘编号，为了记录水量定义 $g[i][j]$ 表示水溢出所经过的共 2^j 个圆盘的水量，显然有

$$f[i][0] = nxt[i], f[i][j] = f[f[i][j-1]][j-1]$$
$$g[i][0] = c[nxt[i]], g[i][j] = g[i][j-1] + g[f[i][j-1]][j-1]$$

```
for(int i = 1; i <= n; ++i)f[i][0] = nxt[i], g[i][0] = c[nxt[i]];
for(int j = 1; j <= T; ++j)
    for(int i = 1; i <= n; ++i)
        f[i][j] = f[f[i][j - 1]][j - 1],
        g[i][j] = g[i][j - 1] + g[f[i][j - 1]][j - 1];
```

【例】喷泉

- 对于当前圆盘 r，首先判断其水能否流出当前盘。
- 然后从大到小枚举溢出的圆盘数 2^j，不断考虑能否溢出到 $f[r][j]$ 即可。
- 时间复杂度：$\mathrm{O}(N\log N)$。

```
int r, v; scanf("%d%d", &r, &v);
if(v > c[r])
{
    v -= c[r];
    for(int j = T; j >= 0; --j)
        if(v > g[r][j]) v -= g[r][j], r = f[r][j];
    r = f[r][0];
}
printf("%d\n", r == n + 1 ? 0 : r);
```

练习

- 取余运算 (COGS 1130)
- 麦森数 [NOIP 2003](COGS 47)
- 延绵的山峰 (COGS 58)
- Balanced Lineup(COGS 3704)
- 滑动窗口 (COGS 495)
- 区间最大公约数 (COGS 3064)
- 策略游戏 [CSP-S 2022](COGS 3782)

练习

- 接竹竿 (COGS 4187)
- Genius ACM(COGS 2491)
- 国旗计划 [SCOI 2015](COGS 3842)
- 喷泉 (COGS 3069)
- ZAB-Frog(COGS 3281)
- 三角形 (COGS 3262)
- 活动 (洛谷 P7562)