ビットレジスタ (Bit Shift Registers)

技術者 Christopher は新型のコンピュータープロセッサを作っている.

プロセッサは m 個の **レジスタ** と呼ばれる b ビットのメモリ $(m = 100, b = 2000)$ にアクセスできる. レジスタには 0 から $m-1$ までの番号が付けられており, それぞれ $r[0], r[1], \ldots, r[m-1]$ で表す. 各レジスタは b ビットの配列であり, それぞれ 0 (右端のビット) から $b-1$ (左端のビット) までの番号が付けられている. 各 i, j $(0 \leqq i \leqq m-1, 0 \leqq j \leqq b-1)$ について, レジスタ i の第 j ビットを $r[i][j]$ で表す.

長さ l のビット列 $d_0, d_1, \ldots, d_{l-1}$ (l は任意) について, この **整数変換値** を $2^0 \cdot d_0 + 2^1 \cdot d_1 + \ldots + 2^{l-1} \cdot d_{l-1}$ とする. また, **レジスタ i における整数変換値** を, それをビット列として表したときの整数変換値, つまり $2^0 \cdot r[i][0] + 2^1 \cdot r[i][1] + \ldots + 2^{b-1} \cdot r[i][b-1]$ とする.

プロセッサはレジスタに格納されているビットを変更するために, 9 種類の **命令** を扱うことができる. 各命令は 1 つ以上のレジスタを扱い, 計算結果を 1 つのレジスタに格納するものである. 以降, $x := y$ を「メモリ x の値を y に変更する操作」とする. 各命令における操作は以下の通りである:

- $move(t, y)$: レジスタ t の値を「レジスタ y の値」に設定する. すなわち, 各 j $(0 \leqq j \leqq b-1)$ について $r[t][j] := r[y][j]$ にする.
- $store(t, v)$: レジスタ t の値を v に設定する (v は b ビットの配列である). すなわち, 各 j $(0 \leqq j \leqq b-1)$ について $r[t][j] := v[j]$ にする.
- $and(t, x, y)$: レジスタ t の値を「レジスタ x と y のビット単位 AND」に設定する. すなわち, 各 j $(0 \leqq j \leqq b-1)$ について, $r[x][j]$ と $r[y][j]$ が **両方** 1 であった場合 $r[t][j] := 1$ にして, それ以外の場合 $r[t][j] := 0$ にする.
- $or(t, x, y)$: レジスタ t の値を「レジスタ x と y のビット単位 OR」に設定する. すなわち, 各 j $(0 \leqq j \leqq b-1)$ について, $r[x][j]$ と $r[y][j]$ の **少なくとも一方が** 1 であった場合 $r[t][j] := 1$ にして, それ以外の場合 $r[t][j] := 0$ にする.
- $xor(t, x, y)$: レジスタ t の値を「レジスタ x と y のビット単位 XOR」に設定する. すなわち, 各 j $(0 \leqq j \leqq b-1)$ について, $r[x][j]$ と $r[y][j]$ の **一方だけが** 1 であった場合 $r[t][j] := 1$ にして, それ以外の場合 $r[t][j] := 0$ にする.
- $not(t, x)$: レジスタ t の値を「レジスタ x のビット単位 NOT」に設定する. すなわち, 各 j $(0 \leqq j \leqq b-1)$ について $r[t][j] := 1 - r[x][j]$ にする.
- $left(t, x, p)$: レジスタ t の値を「レジスタ x を p 個だけ左シフトさせた値」に設定する. ここで, レジスタ x を p 個だけ左シフトさせた値を b ビットの配列 v とするとき, 各 j $(0 \leqq j \leqq b-1)$ に対して, $j \geqq p$ の場合 $v[j] = r[x][j-p]$, それ以外の場合 $v[j] = 0$ である. そして, 各 j について $r[t][j] := v[j]$ とする.

- $right(t, x, p)$: レジスタ t の値を「レジスタ x を p 個だけ右シフトさせた値」に設定する. ここで, レジスタ x を p 個だけ右シフトさせた値を b ビットの配列 v とするとき, 各 j $(0 \leqq j \leqq b-1)$ に対して, $j \leqq b-1-p$ の場合 $v[j] = r[x][j+p]$, それ以外の場合 $v[j] = 0$ である. そして, 各 j について $r[t][j] := v[j]$ とする.

- $add(t, x, y)$: レジスタ t の値を「レジスタ x と y の整数変換値を加算した値を 2^b で割った余り」に設定する. 以下, 厳密に述べる. X を操作前におけるレジスタ x の整数変換値, Y を操作前におけるレジスタ y の整数変換値, T を操作後におけるレジスタ t の整数変換値とする. そのとき, $X + Y < 2^b$ の場合 $T = X + Y$ とし, それ以外の場合 $T = X + Y - 2^b$ とする.

Christopher はあなたに新型のプロセッサを使って 2 種類の問題を解いてほしい. 問題の種類は整数 s で表される. どちらの問題に対しても, あなたは **プログラム** (前述の命令の列)を構成する必要がある.

プログラムの **インプット** は n 個の整数 $a[0], a[1], ..., a[n-1]$ からなる. 各整数は k ビット, すなわち $a[i] < 2^k$ $(0 \leqq i \leqq n-1)$ である. プログラムが実行される前, すべてのインプットはレジスタ 0 に順番に保存され, 各 i $(0 \leqq i \leqq n-1)$ について長さ k のビット列 $r[0][i \cdot k], r[0][i \cdot k + 1], \ldots, r[0][(i+1) \cdot k - 1]$ の整数変換値は $a[i]$ となる. レジスタ 0 の第 $n \cdot k$ ビット以降や, 他のレジスタのビットはすべて 0 で初期化されている. なお, $n \cdot k \leqq b$ が成り立つ.

プログラムを実行するとは, 命令の列を順番に実行することであるとする. 最後の命令が実行された後, **アウトプット** はレジスタ 0 における最終的な値によって計算される. 具体的に述べると, アウトプットは n 個の整数 $c[0], c[1], \ldots, c[n-1]$ であり, 各 i $(0 \leqq i \leqq n-1)$ について $c[i]$ はレジスタ 0 の第 $i \cdot k$ ビットから第 $(i+1) \cdot k - 1$ ビットまでのビット列における整数変換値である. なお, 最後の命令が実行された後において, レジスタ 0 の第 $n \cdot k$ ビット以降や, 他のレジスタのビットはどうでもいい.

- 1 つ目の問題 $(s = 0)$ は $a[0], a[1], \ldots, a[n-1]$ の最小値を求めるものである. 具体的に述べると, $c[0]$ は $a[0], a[1], \ldots, a[n-1]$ の最小値でなければならないが, $c[1], c[2], \ldots, c[n-1]$ の値はどうでもいい.

- 2 つ目の問題 $(s = 1)$ は $a[0], a[1], \ldots, a[n-1]$ の値を昇順にソートするものである. 具体的に述べると, 各 i $(0 \leqq i \leqq n-1)$ について $c[i]$ は $a[0], a[1], \ldots, a[n-1]$ の中で $i+1$ 番目に小さい整数でなければならない. (特に $c[0]$ は最小値である)

Christopher が問題を解けるように, 彼に高々 q 個の命令からなるプログラムを与えよ.

実装の詳細

あなたは以下の関数を実装しなさい.

```
void construct_instructions(int s, int n, int k, int q)
```

- s: 問題の種類.
- n: インプットで与えられる整数の個数.
- k: インプットで与えられる整数のビット数.
- q: 許される命令の数の上限.
- この関数はちょうど 1 回だけ呼び出され, 問題を解くための命令の列を構築する必要がある.

この関数は以下の関数を 1 回以上を呼び出すことで, 命令の列を構成しなければならない.

```
void append_move(int t, int y)
void append_store(int t, bool[] v)
void append_and(int t, int x, int y)
void append_or(int t, int x, int y)
void append_xor(int t, int x, int y)
void append_not(int t, int x)
void append_left(int t, int x, int p)
void append_right(int t, int x, int p)
void append_add(int t, int x, int y)
```

- 各関数はそれぞれ $move(t, y)$ $store(t, v)$, $and(t, x, y)$, $or(t, x, y)$, $xor(t, x, y)$, $not(t, x)$, $left(t, x, p)$, $right(t, x, p)$, $add(t, x, y)$ という命令をプログラムに追加する.
- すべての命令について, t, x, y は 0 以上 $m - 1$ 以下でなければならない.
- すべての命令について, t, x, y は相異なっている必要はない.
- $left$ 命令と $right$ 命令については, p は 0 以上 b 以下でなければならない.
- $store$ 命令については, v の長さは b でなければならない.

あなたはプログラムをテストするために, 以下の関数を呼び出すこともできる.

```
void append_print(int t)
```

- この関数の呼び出しは, 採点の際には無視される.
- 採点プログラムのサンプルでは, この関数は $print(t)$ という命令をプログラムに追加する.
- 採点プログラムのサンプルが $print(t)$ 命令を実行すると, レジスタ t の最初の $n \cdot k$ ビットから計算された n 個の k ビット整数が出力される. (詳細は「採点プログラムのサンプル」を参照のこと)
- t は 0 以上 $m - 1$ 以下でなければならない.
- この関数呼び出しを行っても, 命令の個数には加算されない.

最後の命令を追加した後, `construct_instructions` は return しなければならない. その後, あなたの構築したプログラムはいくつかのテストケースを用いて評価される. 各テストケースは n 個の k ビットの整数 $a[0], a[1], \ldots, a[n-1]$ からなる. アウトプット $c[0], c[1], \ldots, c[n-1]$ が以下の条件を満たすときに限り, テストケースに正解したとみなされる:

- $s = 0$ のとき, $c[0]$ は $a[0], a[1], \ldots, a[n-1]$ の最小値でなければならない.
- $s = 1$ のとき, 各 i $(0 \leqq i \leqq n-1)$ について, $c[i]$ は $a[0], a[1], \ldots, a[n-1]$ の中で $1 + i$ 番目に小さい整数でなければならない.

あなたの解答の評価は次に示すエラーメッセージのどれか 1 つで示されるかもしれない:

- `Invalid index`: 関数呼び出しで指定されたレジスタの番号 t, x, y が不正であった (負であったかもしれない).
- `Value to store is not b bits long`: `append_store` 関数における v の長さが b ではなかった.

- `Invalid shift value`: `append_left` 関数／`append_right` 関数における p の値が 0 以上 l 以下ではなかった.
- `Too many instructions`: あなたの関数は q 回以上の命令を追加しようとした.

入出力例

入出力例 1

$s = 0,\ n = 2,\ k = 1,\ q = 1000$ とする. インプットは 2 つの整数 $a[0], a[1]$ であり, それぞれ $k = 1$ ビットである. プログラムの実行が始まる前, $r[0][0] = a[0]$ かつ $r[0][1] = a[1]$ であり, その他のビットはすべて 0 に初期化されている. すべての命令が実行された後, $c[0] = r[0][0] = \min(a[0], a[1])$ を満たす必要がある.

インプットとして考えられるものは以下の 4 通りしかない:

- ケース 1: $a[0] = 0, a[1] = 0$
- ケース 2: $a[0] = 0, a[1] = 1$
- ケース 3: $a[0] = 1, a[1] = 0$
- ケース 4: $a[0] = 1, a[1] = 1$

ここで 4 つのケースすべてについて, $\min(a[0], a[1])$ は $a[0]$ AND $a[1]$ と等しい. したがって, 以下の関数呼び出しによって作られるプログラムが, 正しい解の 1 つとして考えられる.

1. `append_move(1, 0)`: $r[1]$ の値を「$r[0]$ の値」に設定する命令を追加する.
2. `append_right(1, 1, 1)`: $r[1]$ の値を「$r[1]$ を 1 個だけ右シフトさせた値」に設定する命令を追加する. 各整数の長さは 1 ビットであるため, $r[1][0]$ の値は $a[1]$ と等しくなる.
3. `append_and(0, 0, 1)`: $r[0]$ の値を「$r[0]$ と $r[1]$ の AND」に設定する命令を追加する. その命令が実行された後, $r[0][0]$ は $r[0][0]$ AND $r[1][0]$ に設定され, これはこの問題の解となる $a[0]$ AND $a[1]$ と等しい.

入出力例 2

$s = 1,\ n = 2,\ k = 1,\ q = 1000$ とする. 前述の通り, インプットとして考えられるものは 4 通りしかない. ここで 4 つのケースすべてについて, $\min(a[0], a[1])$ は $a[0]$ AND $a[1]$ と等しく, $\max(a[0], a[1])$ は $a[0]$ OR $a[1]$ と等しい. あり得る正答として, 以下の呼び出しが考えられる.

1. `append_move(1,0)`
2. `append_right(1,1,1)`
3. `append_and(2,0,1)`
4. `append_or(3,0,1)`
5. `append_left(3,3,1)`
6. `append_or(0,2,3)`

すべての命令が実行された後, $c[0] = r[0][0]$ には $\min(a[0], a[1])$ が格納されており, $c[1] = r[0][1]$ には $\max(a[0], a[1])$ が格納されている. これはインプットを昇順にソートしたものである.

制約

- $m = 100$
- $b = 2000$
- $0 \leqq s \leqq 1$
- $2 \leqq n \leqq 100$
- $1 \leqq k \leqq 10$
- $q \leqq 4000$
- $0 \leqq a[i] \leqq 2^k - 1$ ($0 \leqq i \leqq n - 1$)

小課題

1. (10 点) $s = 0, n = 2, k \leqq 2, q = 1000$
2. (11 点) $s = 0, n = 2, k \leqq 2, q = 20$
3. (12 点) $s = 0, q = 4000$
4. (25 点) $s = 0, q = 150$
5. (13 点) $s = 1, n \leqq 10, q = 4000$
6. (29 点) $s = 1, q = 4000$

採点プログラムのサンプル

採点プログラムのサンプルは以下の形式で入力を読み込む.

- 1 行目: s n k q

その後何行の入力が続き, 1 行につき 1 つのテストケースが与えられる. 各テストケースは以下の形式で入力を読み込む:

- $a[0]$ $a[1]$ $\ldots$ $a[n-1]$ (インプットが $a[0], a[1], ..., a[n-1]$ であることを意味する)

そして最終行は -1 であり, 全部のテストケースを入力し終わったことを意味する.

採点プログラムのサンプルは, 最初に関数 `construct_instructions(s, n, k, q)` を呼び出す. もし関数呼び出しが課題文で述べられている何らかの制約に違反している場合, 採点プログラムのサンプルは「実装の詳細」セクションの最後に記載されたエラーメッセージの 1 つを表示し, 終了する. そうでない場合, 採点プログラムのサンプルは最初に `construct_instructions(s, n, k, q)` で追加された命令を順番に出力する. 特に $store$ 命令については, v は第 $0, 1, 2, ..., b-1$ ビットの順に出力される.

次に, 採点プログラムのサンプルはテストケースを順番に処理する. 各テストケースについては以下の通り:

- まず, あなたの構成したプログラムを実行する.
- $print(t)$ 命令が呼び出された時, 採点プログラムのサンプルは「`register` t: $d[0]$ $d[1]$ $\ldots$ $d[n-1]$」という形式で出力する. ただし, $0 \leqq i \leqq n-1$ に対して, $d[i]$ は指示が

実行された時点における，レジスタ t の第 $i \cdot k$ ビットから第 $(i+1) \cdot k - 1$ ビットまでの整数変換値とする．

- 全部の命令が実行された後，採点プログラムのサンプルは以下の形式でアウトプットを出力する．
  - $s = 0$ の場合： $c[0]$
  - $s = 1$ の場合： $c[0]\ c[1]\ \ldots\ c[n-1]$

全部のテストケースが実行された後，採点プログラムのサンプルは「`number of instructions:` X」と出力する（X はあなたのプログラムにおける命令の数）．