ბიტური წანაცვლების რეგისტრები

ინჟინერი კრისტოფერი მუშაობს კომპიუტერის ახალი ტიპის პროცესორზე.

პროცესორს აქვს წვდომა მეხსიერების m ცალ განსხვავებულ b-ბიტიან უჯრაზე (სადაც $m = 100$ და $b = 2000$), რომლებსაც **რეგისტრები** ეწოდებათ და არიან დანომრილი 0-დან ($m - 1$)-მდე. რეგისტრები აღვნიშნოთ, როგორც $r[0], r[1], \ldots, r[m-1]$. თითოეული რეგისტრი არის b ბიტიანი მასივი, დანომრილი 0-დან (უკიდურესი მარჯვენა ბიტი) ($b - 1$)-მდე (უკიდურესი მარცხენა ბიტი). ყოველი i-თვის $(0 \leq i \leq m-1)$ და ყოველი j-თვის $(0 \leq j \leq b-1)$, აღვნიშნოთ i რეგისტრის j-ური ბიტი $r[i][j]$-ით.

ბიტების $d_0, d_1, \ldots, d_{l-1}$ მიმდევრობის (l სიგრძის) **რიცხვითი მნიშვნელობა** არის $2^0 \cdot d_0 + 2^1 \cdot d_1 + \ldots + 2^{l-1} \cdot d_{l-1}$. ვთვლით, რომ i **რეგისტრში ჩაწერილი რიცხვი** არის მისი ბიტების მიმდევრობის რიცხვითი მნიშვნელობა, ანუ $2^0 \cdot r[i][0] + 2^1 \cdot r[i][1] + \ldots + 2^{b-1} \cdot r[i][b-1]$.

პროცესორს გააჩნია 9 ტიპის **ინსტრუქცია** ბიტების შესაცვლელად. ყოველი ინსტრუქცია მოქმედებს ერთ ან მეტ რეგისტრზე და წერს შედეგს ერთ რეგისტრში. შემდგომში ჩვენ $x := y$-ით აღვნიშნავთ x-ის მნიშვნელობის შეცვლას რის შემდეგაც ის ხდება y-ის ტოლი. თითოეული ინსტრუქციის მიერ ჩატარებული მოქმედებები აღწერილია ქვემოთ.

- $move(t, y)$: დააკოპირე ბიტების y რეგისტრი t რეგისტრში. ყოველი j-თვის $(0 \leq j \leq b-1)$, სრულდება $r[t][j] := r[y][j]$.
- $store(t, v)$: გახადე t რეგისტრი v-ს ტოლი, სადაც v არის b ბიტიანი მასივი. ყოველი j-თვის $(0 \leq j \leq b-1)$, სრულდება $r[t][j] := v[j]$.
- $and(t, x, y)$: აიღე x და y რეგისტრების ბიტური AND და ჩაწერე შედეგი t რეგისტრში. ყოველი j-თვის $(0 \leq j \leq b-1)$ სრულდება $r[t][j] := 1$, თუ **ორივე** $r[x][j]$ და $r[y][j]$ არის 1, და სრულდება $r[t][j] := 0$ წინააღმდეგ შემთხვევაში.
- $or(t, x, y)$: აიღე x და y რეგისტრების ბიტური OR და ჩაწერე შედეგი t რეგისტრში. ყოველი j-თვის $(0 \leq j \leq b-1)$ სრულდება $r[t][j] := 1$, თუ **ერთი მაინც** $r[x][j]$ და $r[y][j]$-დან არის 1, და სრულდება $r[t][j] := 0$ წინააღმდეგ შემთხვევაში.
- $xor(t, x, y)$: აიღე x და y რეგისტრების ბიტური XOR და ჩაწერე შედეგი t რეგისტრში. ყოველი j-თვის $(0 \leq j \leq b-1)$ სრულდება $r[t][j] := 1$, თუ **ზუსტად ერთი** $r[x][j]$ და $r[y][j]$-დან არის 1, და სრულდება $r[t][j] := 0$ წინააღმდეგ შემთხვევაში.
- $not(t, x)$: აიღე x რეგისტრის ბიტური NOT და ჩაწერე შედეგი t რეგისტრში. ყოველი j-თვის $(0 \leq j \leq b-1)$, სრულდება $r[t][j] := 1 - r[x][j]$.
- $left(t, x, p)$: წავანაცვლოთ x რეგისტრის ყველა ბიტი მარცხნივ p ადგილით და ჩავწეროთ შედეგი t რეგისტრში. x რეგისტრის ბიტების p ადგილით მარცხნივ წაძვრით

მიიღება v მასივი, რომელიც შედგება b ბიტისგან. ყოველი j-თვის $(0 \leq j \leq b-1)$, $v[j] = r[x][j-p]$ თუ $j \geq p$, და $v[j] = 0$ წინააღმდეგ შემთხვევაში. ყოველი j-თვის $(0 \leq j \leq b-1)$ სრულდება $r[t][j] := v[j]$.

- $right(t, x, p)$: წავანაცვლოთ x რეგისტრის ყველა ბიტი მარჯვნივ p ადგილით და ჩავწეროთ შედეგი t რეგისტრში. x რეგისტრის ბიტების p ადგილით მარჯვნივ წაძვრით მიიღება v მასივი რომელიც შედგება b ბიტისგან. ყოველი j-თვის $(0 \leq j \leq b-1)$, $v[j] = r[x][j+p]$, თუ $j \leq b-1-p$, და $v[j] = 0$ წინააღმდეგ შემთხვევაში. ყოველი j-თვის $(0 \leq j \leq b-1)$ სრულდება $r[t][j] := v[j]$.

- $add(t, x, y)$: შევკრიბოთ x და y რეგისტრებში ჩაწერილი რიცხვითი მნიშვნელობები და ჩავწეროთ შედეგი t რეგისტრში. შეკრება სრულდება მოდულით 2^b. ფორმალურად, ვთქვათ, X არის x რეგისტრის რიცხვითი მნიშვნელობა და Y არის y რეგისტრის რიცხვითი მნიშვნელობა ოპერაციის დაწყებამდე. T იყოს t რეგისტრის რიცხვითი მნიშვნელობა ოპერაციის შემდეგ. თუ $X + Y < 2^b$, მაშინ ჩავწეროთ t-ს ბიტები ისე, რომ $T = X + Y$. წინააღმდეგ შემთხვევაში ჩავწეროთ t-ს ბიტები ისე, რომ $T = X + Y - 2^b$.

კრისტოფერს სურს ახალ პროცესორზე შეასრულოს ორი ტიპის დავალება. დავალების ტიპი აღინიშნება მთელი s რიცხვით. ორივე ტიპის დავალებისთვის თქვენ უნდა შეადგინოთ **პროგრამა**, რომელიც ზემოთ აღწერილი ინსტრუქციების მიმდევრობაა.

პროგრამის **შესატანი მონაცემები** შედგება n ცალი მთელი $a[0], a[1], \ldots, a[n-1]$ რიცხვებისგან, სადაც თითოეული k ბიტიანია, ე.ი., $a[i] < 2^k$ ($0 \leq i \leq n-1$). პროგრამის შესრულების დაწყებამდე ყველა შემოსული რიცხვი ჩაწერილია თანმიმდევრობით რეგისტრში 0 ისე, რომ ყოველი i-თვის $(0 \leq i \leq n-1)$ რიცხვითი მნიშვნელობა k ბიტიანი $r[0][i \cdot k], r[0][i \cdot k + 1], \ldots, r[0][(i+1) \cdot k - 1]$ მიმდევრობისა არის $a[i]$-ს ტოლი. შევნიშნოთ, რომ $n \cdot k \leq b$. რეგისტრში 0 ყველა სხვა ბიტი (ე.ი. ისინი, რომელთა ინდექსებიც $n \cdot k$-დან $b-1$-ის ჩათვლითაა) და სხვა რეგისტრების ყველა ბიტი ინიციალიზირებულია 0-ებით.

პროგრამის შესრულება გულისხმობს ინსტრუქციების თანმიმდევრობით შესრულებას. ბოლო ინსტრუქციის შესრულების შემდეგ პროგრამის **გამოსატანი მონაცემები** გამოითვლება რეგისტრში 0 ბიტების საბოლოო მნიშვნელობებით. კერძოდ, გამოტანილი იქნება მიმდევრობა, შემდგარი n ცალი $c[0], c[1], \ldots, c[n-1]$ რიცხვისაგან, სადაც ყოველი i-თვის ($0 \leq i \leq n-1$) $c[i]$ არის რიცხვითი მნიშვნელობა რეგისტრში 0 ბიტების მიმდევრობისა $i \cdot k$-დან $(i+1) \cdot k - 1$-მდე. შევნიშნოთ, რომ პროგრამის შესრულების შემდეგ რეგისტრი 0-ის დარჩენილი ბიტები (რომელთა ინდექსები არის არანაკლები $n \cdot k$) და სხვა რეგისტრების ბიტები ნებისმიერი შეიძლება იყოს.

- პირველი ტიპის $(s = 0)$ დავალებაა მოცემულ $a[0], a[1], \ldots, a[n-1]$ რიცხვებში უმცირესის პოვნა. კერძოდ, $c[0]$ უნდა იყოს მინიმუმი $a[0], a[1], \ldots, a[n-1]$ რიცხვებს შორის. $c[1], c[2], \ldots, c[n-1]$ მნიშვნელობები ნებისმიერი შეიძლება იყოს.

- მეორე ტიპის $(s = 1)$ დავალებაა $a[0], a[1], \ldots, a[n-1]$ რიცხვების დალაგება არაკლებადი მიმდევრობით. კერძოდ, ყოველი i-თვის ($0 \leq i \leq n-1$), $c[i]$ უნდა იყოს ტოლი $a[0], a[1], \ldots, a[n-1]$ რიცხვებს შორის $1+i$-ური უმცირესი რიცხვისა (ე.ი., $c[0]$ არის მოცემულ რიცხვებს შორის უმცირესი).

დაეხმარეთ კრისტოფერს შეადგინოს პროგრამები, შემდგარი არაუმეტეს q ინსტრუქციისგან, რომლებიც ამ დავალებებს შეასრულებს.

იმპლემენტაციის დეტალები

თქვენ უნდა მოახდინოთ შემდეგი პროცედურის იმპლემენტაცია:

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

- s: დავალებების რაოდენობა.
- n: შემომავალი მთელი რიცხვების რაოდენობა.
- k: ყოველ შემოსულ რიცხვში ბიტების რაოდენობა.
- q: ინსტრუქციების მაქსიმალური დაშვებული რაოდენობა.
- პროცედურა გამოძახებული იქნება ზუსტად ერთხელ და მან უნდა იპოვოს დავალების შესასრულებელი ინსტრუქციების მიმდევრობა.

პროცედურამ მიმდევრობის ასაგებად უნდა გამოიძახოს შემდეგი პროცედურებიდან ერთი ან მეტი:

```
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)
```

- თითოეული პროცედურა პროგრამაში ამატებს $move(t, y)$ $store(t, v)$, $and(t, x, y)$, $or(t, x, y)$, $xor(t, x, y)$, $not(t, x)$, $left(t, x, p)$, $right(t, x, p)$ ან $add(t, x, y)$ ინსტრუქციას, შესაბამისად.
- შესაბამის ინსტრუქციებში t, x და y უნდა იყვნენ 0-დან ($m - 1$)-ის ჩათვლით.
- შესაბამის ინსტრუქციებში t, x და y შესაძლოა არ იყვნენ წყვილ-წყვილად განსხვავებული.
- $left$ და $right$ ინსტრუქციებში p უნდა იყოს 0-დან b-ს ჩათვლით.
- $store$ ინსტრუქციებში v-ს სიგრძე უნდა იყოს b.

ამოხსნის გასატესტად თქვენ შეგიძლიათ გამოიძახოთ შემდეგი პროცედურა:

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

- ამ პროცედურის გამოძახებები იგნორირებული იქნება ამოხსნის შეფასებისას.
- სანიმუშო გრადერში ეს პროცედურა პროგრამას უმატებს $print(t)$ ოპერაციას.
- როდესაც სანიმუშო გრადერს პროგრამის შესრულებისას ხვდება $print(t)$ ოპერაცია, ის ბეჭდავს n ცალ k ბიტიან მთელ რიცხვს, რომლებიც იქმნება t რეგისტრის პირველი $n \cdot k$ ბიტისგან (დეტალებისთვის იხილეთ სექცია "სანიმუშო გრადერი").
- t უნდა აკმაყოფილებდეს პირობას $0 \leq t \leq m - 1$.
- ამ პროცედურის გამოძახება არ ითვლება ინსტრუქციების რაოდენობაში.

ბოლო ინსტრუქციის შემდეგ `construct_instructions` უნდა დასრულდეს. შემდგომში პროგრამა შესრულდება ტესტების გარკვეულ რაოდენობაზე, რომელთაგან თითოეული შედგება n ცალი k-ბიტიანი $a[0], a[1], \ldots, a[n-1]$ რიცხვისაგან. თქვენი ამოხსნა გაატარებს ტესტს, თუ გამოსატანი მონაცემები $c[0], c[1], \ldots, c[n-1]$ აკმაყოფილებს შემდეგ პირობებს:

- თუ $s = 0$, $c[0]$ არის უმცირესი $a[0], a[1], \ldots, a[n-1]$ რიცხვებს შორის.
- თუ $s = 1$, ყოველი i-თვის ($0 \leq i \leq n-1$), $c[i]$ არის ($1 + i$)-ური უმცირესი რიცხვი $a[0], a[1], \ldots, a[n-1]$ რიცხვებს შორის.

თქვენი პროგრამის შეფასებისას შესაძლებელია შემდეგი შეცდომები:

- `Invalid index`: რეგისტრის არასწორი (შესაძლოა უარყოფითი) ინდექსი, რომელიც გადაეცა t, x ან y პარამეტრად რომელიმე გამოძახებაში.
- `Value to store is not b bits long`: v-ს სიგრძე, რომელიც გადაეცა `append_store`-ს არ იყოს b-ს ტოლი.
- `Invalid shift value`: p-ს მნიშვნელობა, რომელიც გადაეცა `append_left` ან `append_right`-ს არ არის 0-დან b-ს ჩათვლით.
- `Too many instructions`: თქვენმა პროცედურამ სცადა q-ზე მეტი ინსტრუქციის დამატება.

მაგალითები

მაგალითი 1

ვთქვათ, $s = 0$, $n = 2$, $k = 1$, $q = 1000$. გვაქვს ორი რიცხვი $a[0]$ და $a[1]$, თითოეული $k = 1$ ბიტით. პროგრამის შესრულებამდე $r[0][0] = a[0]$ და $r[0][1] = a[1]$. პროცესორის ყველა სხვა ბიტი არის 0. პროგრამის ინსტრუქციების შედეგად გვსურს, რომ $c[0] = r[0][0] = \min(a[0], a[1])$, რომელიც არის $a[0]$ და $a[1]$-ს შორის მინიმუმი.

შესაძლებელია 4 სხვადასხვა შესატანი მონაცემები:

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

შევნიშნოთ, რომ ოთხივე შემთხვევაში $\min(a[0], a[1])$ არის $a[0]$ და $a[1]$-ის ბიტური AND. შესაბამისად, შესაძლო ამონახსნი იქნება პროგრამის შედგენა შემდეგი ინსტრუქციებით:

1. `append_move(1, 0)`: ამატებს ინსტრუქციას, რომელიც აკოპირებს $r[0]$-ს $r[1]$-ში.
2. `append_right(1, 1, 1)`: ამატებს ინსტრუქციას, რომელიც იღებს $r[1]$-ს ყველა ბიტს, წაანაცვლებს მათ მარჯვნივ 1 ბიტით და შემდეგ წერს შედეგს $r[1]$-ში. რადგან ყველა რიცხვი 1 ბიტიანია, $r[1][0]$ იქნება $a[1]$-ის ტოლი.
3. `append_and(0, 0, 1)`: ამატებს ინსტრუქციას, რომელიც იღებს $r[0]$ და $r[1]$-ის ბიტურ AND-ს, შემდეგ კი წერს შედეგს $r[0]$-ში. ამ ინსტრუქციის შესრულების შემდეგ $r[0][0]$ იქნება $r[0][0]$ და $r[1][0]$-ს ბიტური AND, რომელიც არის $a[0]$ და $a[1]$-ის ბიტური AND, როგორც გვსურდა.

მაგალითი 2

ვთქვათ, $s = 1,\ n = 2,\ k = 1,\ q = 1000$. წინა მაგალითის მსგავსად, შესაძლებელია 4 სხვადასხვა შესატანი მონაცემი. ოთხივე შემთხვევაში $\min(a[0], a[1])$ არის $a[0]$ და $a[1]$-ის ბიტური AND, ხოლო $\max(a[0], a[1])$ არის $a[0]$ და $a[1]$-ის ბიტური OR. შესაძლო ამონახსნია შემდეგი გამოძახებები:

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)`

ამ ინსტრუქციების შესრულების შედეგად $c[0] = r[0][0]$ არის $\min(a[0], a[1])$, და $c[1] = r[0][1]$ არის $\max(a[0], a[1])$, რომელიც ასორტირებს შესატან მონაცემებს.

შეზღუდვები

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

ქვეამოცანები

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

სანიმუშო გრადერი

სანიმუშო გრადერს შეაქვს მონაცემები შემდეგი ფორმატით:

- სტრიქონი 1 : $s\ n\ k\ q$

ამას მოჰყვება რაღაც რაოდენობის სტრიქონები, რომელთაგან თითოეული აღწერს ერთ ტესტს. თითოეული ტესტი მოცემულია შემდეგი ფორმატით:

- $a[0]\ a[1]\ \ldots\ a[n-1]$

და აღწერს ტესტს, რომლის შესატანი მონაცემები შედგება n რიცხვისგან $a[0], a[1], ..., a[n-1]$. ყველა ტესტის აღწერას მოსდევს ერთი სტრიქონი, რომელიც შეიცავს მხოლოდ რიცხვს -1.

სანიმუშო გრადერი ჯერ იძახებს `construct_instructions(s, n, k, q)`-ს. თუ ეს გამოძახება არღვევს პირობაში მოცემულ რომელიმე შეზღუდვას, სანიმუშო გრადერს გამოაქვს შეცდომა, როგორც ეს აღწერილია "იმპლემენტაციის დეტალები"-ს სექციაში და ასრულებს მუშაობას. წინააღმდეგ შემთხვევაში, სანიმუშო გრადერი წერს `construct_instructions(s, n, k, q)`-ს მიერ დამატებულ ბრძანებებს თანმიმდევრობით. $store$ ინსტრუქციისთვის, v იწერება 0 ინდექსიდან $b-1$ ინდექსამდე.

შემდეგ სანიმუშო გრადერი თანმიმდევრობით ასრულებს ტესტებს. თითოეული ტესტისთვის ის უშვებს აგებულ პროგრამას შესაბამის ტესტზე.

ყოველი $print(t)$ ოპერაციისთვის, ვთქვათ $d[0], d[1], \ldots, d[n-1]$ არის რიცხვების მიმდევრობა, სადაც ყოველი i-თვის ($0 \le i \le n-1$) $d[i]$ არის რიცხვითი მნიშვნელობა ბიტების $i \cdot k$ .. $(i+1) \cdot k - 1$ მიმდევრობისა რეგისტრში t (ოპერაციის შესრლებისას). გრადერი ბეჭდავს მიმდევრობას შემდეგი ფორმატით: `register` t: $d[0]$ $d[1]$ $\ldots$ $d[n-1]$.

ყველა ინსტრუქციის შესრულების შემდეგ სანიმუშო გრადერი ბეჭდავს პროგრამის გამოსატან მონაცემებს.

თუ $s = 0$, სანიმუშო გრადერს გამოაქვს მონაცემები შემდეგი ფორმატით:

- $c[0]$.

თუ $s = 1$, სანიმუშო გრადერს გამოაქვს მონაცემები შემდეგი ფორმატით:

- $c[0]$ $c[1]$ $\ldots$ $c[n-1]$.

ყველა ტესტის შესრულების შემდეგ სანიმუშო გრადერი ბეჭდავს `number of instructions:` X, სადაც X თქვენს პროგრამაში ინსტრუქციების რაოდენობაა.