Bit Kaydırma Registerları

Mühendis Christopher yeni bir bilgisayar işlemcisi üzerinde çalışmaktadır.

İşlemci üzerinde **register** adı verilen m farklı b-bit boyunda hafıza hücreleri vardır ($m = 100$ ve $b = 2000$), ve 0'dan $m-1$'e numaralandırılmışlardır. Registerları $r[0], r[1], \ldots, r[m-1]$ olarak gösteriyoruz. Her bir register b bit uzunluğunda 0'dan (en sağdaki bit) $b-1$'e (en soldaki bit) numaralandırılmış bir arraydir. Her bir i $(0 \leq i \leq m-1)$ ve j $(0 \leq j \leq b-1)$ için, i registerının j. bitini $r[i][j]$ ile gösteriyor olacağız.

$d_0, d_1, \ldots, d_{l-1}$ şeklinde (rastgele bir l uzunluğunda) herhangi bir bit dizisi verildiğinde, bu dizinin **tamsayı değeri** $2^0 \cdot d_0 + 2^1 \cdot d_1 + \ldots + 2^{l-1} \cdot d_{l-1}$ değerine eşittir. i numaralı **registerda bulunan tamsayı değer** o registerın bit dizisinin tamsayı değeridir, yani, $2^0 \cdot r[i][0] + 2^1 \cdot r[i][1] + \ldots + 2^{b-1} \cdot r[i][b-1]$.

Bu işlemcide, registerlardaki bitleri değiştirmek için kullanılabilecek 9 tip **komut** bulunmaktadır. Her bir komut bir ya da daha fazla register üzerinde çalışıp sonucu registerlardan birisine kaydeder. Aşağıda, $x := y$ gösterimini x'in değerini değiştirip y yapan operasyonu belirtmek için kullanacağız. Her bir komutun gerçekleştirdiği operasyon aşağıda açıklanmıştır.

- $move(t, y)$: y registerındaki bit arrayini t registerına kopyala. Her bir j $(0 \leq j \leq b-1)$ için, $r[t][j] := r[y][j]$ işlemini gerçekleştir.
- $store(t, v)$: Register t'yi v'ye eşitle, burada v, b bitlik bir arraydir. Her bir j $(0 \leq j \leq b-1)$ için, $r[t][j] := v[j]$ işlemini gerçekleştir.
- $and(t, x, y)$: Register x ve y'yi bitwise-AND'leyip sonucu t registerına koy. Her bir j $(0 \leq j \leq b-1)$ için, eğer **hem** $r[x][j]$ **hem de** $r[y][j]$ 1 ise $r[t][j] := 1$ yap, değilse $r[t][j] := 0$ yap.
- $or(t, x, y)$: Register x ve y'yi bitwise-OR'layıp sonucu t registerına koy. Her bir j $(0 \leq j \leq b-1)$ için, $r[x][j]$ ya da $r[y][j]$'den **en az biri** 1 ise $r[t][j] := 1$ yap, değilse $r[t][j] := 0$ yap.
- $xor(t, x, y)$: Register x ve y'yi bitwise-XOR'layıp sonucu t registerına koy. Her bir j $(0 \leq j \leq b-1)$ için, $r[x][j]$ ya da $r[y][j]$'den **tam olarak biri** 1 ise $r[t][j] := 1$ yap, değilse $r[t][j] := 0$ yap.
- $not(t, x)$: Register x'in bitwise-NOT'ını alıp sonucu t registerına koy. Her bir j $(0 \leq j \leq b-1)$ için, $r[t][j] := 1 - r[x][j]$ işlemini gerçekleştir.
- $left(t, x, p)$: Register x'teki bütün bitleri p kere sola kaydır ve sonucu t registerına koy. x registerındaki bitlerin p kere sola kaydırılmasının sonucunda b bitten oluşan bir v arrayi elde

edilir. Her bir j $(0 \leq j \leq b-1)$ için, eğer $j \geq p$ ise $v[j] = r[x][j-p]$ olur, değilse de $v[j] = 0$ olur. Her bir j $(0 \leq j \leq b-1)$ için, $r[t][j] := v[j]$ işlemini gerçekleştir.

- $right(t, x, p)$: Register x'teki bütün bitleri p kere sağa kaydır ve sonucu t registerına koy. x registerındaki bitlerin p kere sağa kaydırılmasının sonucunda b bitten oluşan bir v arrayi elde edilir. Her bir j $(0 \leq j \leq b-1)$ için, eğer $j \leq b-1-p$ ise $v[j] = r[x][j+p]$ olur, değilse de $v[j] = 0$ olur. Her bir j $(0 \leq j \leq b-1)$ için, $r[t][j] := v[j]$ işlemini gerçekleştir.

- $add(t, x, y)$: Register x ve y'deki tamsayı değerleri toplayıp sonucu t registerına koy. Toplama işlemi mod 2^b'de gerçekleştirilir. X, x registerındaki, Y de y registerındaki işlemden önceki tamsayı değerleri ve T de t registerına işlemden sonra konulan tamsayı değeri göstersin. Eğer $X + Y < 2^b$ ise, t'nin bitleri, $T = X + Y$ olacak şekilde belirlenir, değilse t'nin bitleri $T = X + Y - 2^b$ olacak şekilde belirlenir.

Christopher yeni işlemciyi kullanarak iki tür problem çözmek istiyor. Çözülecek problem tipi bir s tamsayısı ile gösterilir. Her bir problem tipi için, yukarıdaki komutlardan oluşan bir **program** oluşturmalısınız.

Programın **girdi**si her biri k bitten oluşan n tane tamsayıdır $a[0], a[1], \ldots, a[n-1]$, yani, $a[i] < 2^k$ ($0 \leq i \leq n-1$). Program çalışmaya başlamadan önce girdideki bütün sayılar sıralı olarak register 0'a konulurlar, yani her bir i $(0 \leq i \leq n-1)$ için $r[0][i \cdot k], r[0][i \cdot k + 1], \ldots, r[0][(i+1) \cdot k - 1]$'daki k bitin tamsayı değeri $a[i]$'ye eşit olur. $n \cdot k \leq b$ olduğuna dikkat ediniz. Register 0'daki diğer bütün bitler (yani $n \cdot k$ ile $b-1$ indeksi arasındaki bitler, sınırlar dahil) ve diğer registerlardaki diğer bütün bitler 0 olarak varsayılan değerlerle ilk kullanıma hazırlanmıştır.

Bir programı çalıştırmak demek komutlarını sırayla çalıştırmak demektir. Son komut çalıştıktan sonra, programın **çıktısı** 0 registerındaki bitlerin son değerlerine bakılarak hesaplanır. Diğer bir deyişle, çıktı n tamsayıdan oluşan bir $c[0], c[1], \ldots, c[n-1]$ dizisidir, öyle ki, her bir i ($0 \leq i \leq n-1$) için, $c[i$, 0 registerındaki $i \cdot k$'dan $(i+1) \cdot k - 1$'a kadar olan bitlerin tam sayı değeridir. Programın çalışması bittikten sonra 0 registerındaki diğer bitler (yani en az $n \cdot k$ indeksine sahip olanlar) ve diğer bütün registerlardaki bütün bitler rastgele bitler olabilir.

- İlk problem $(s = 0)$, $a[0], a[1], \ldots, a[n-1]$ olarak verilen tamsayı girdilerden en küçüğünü bulmaktır. Yani, $c[0]$, $a[0], a[1], \ldots, a[n-1]$'ların en küçüğü olmalıdır. $c[1], c[2], \ldots, c[n-1]$'ların değerleri rastgele olabilir.

- İkinci problem $(s = 1)$, $a[0], a[1], \ldots, a[n-1]$ olarak verilen tamsayıları küçükten büyüğe sıralamaktır. Yani, her bir i ($0 \leq i \leq n-1$) için, $c[i]$ değeri $a[0], a[1], \ldots, a[n-1]$ içindeki $1+i$. en küçük tamsayı olmalıdır (yani, $c[0]$ girdiler içindeki en küçük sayı olmalıdır).

Bu problemleri çözebilmek için her biri en fazla q komuttan oluşan programlar oluşturup Christopher'a yardım edin.

Implementasyon Detayları

Aşağıdaki fonksiyonu implement etmelisiniz:

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

- s: problem tipi.
- n: girdideki tamsayıların sayısı.
- k: her bir tamsayı girdideki bit sayısı.
- q: en fazla kullanılabilecek komut sayısı.
- Bu fonksiyon tam olarak bir kez çağrılır ve istenilen problemi çözen bir komut dizisi oluşturmalıdır.

Bir komut dizisi oluşturmak için fonksiyonunuz aşağıdaki fonksiyonları bir ya da daha fazla kez çağırmalıdır

```
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)
```

- Yukarıdaki her bir fonksiyon programa ilgili $move(t, y)$ $store(t, v)$, $and(t, x, y)$, $or(t, x, y)$, $xor(t, x, y)$, $not(t, x)$, $left(t, x, p)$, $right(t, x, p)$ or $add(t, x, y)$ komutunu ekler.
- İlgili bütün komutlar için, t, x, y en az 0 ve en fazla $m - 1$'dir.
- İlgili bütün komutlar için, t, x, y birbirlerinden farklı olmak zorunda değildir.
- $left$ ve $right$ komutları için, p en az 0 ve en fazla b'dir.
- $store$ komutu için, v'nin uzunluğu b olmalıdır.

Çözümünüzü test etmek için ayrıca aşağıdaki fonksiyonu çağırabilirsiniz:

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

- Çözümünüz değerlendirilirken bu fonksiyona yapılan çağrılar gözardı edilecektir.
- Örnek grader'da, bu fonksiyon programa bir $print(t)$ operasyonu ekler.
- Örnek grader bir programı çalıştırırken bir $print(t)$ komutuyla karşılaştığında, n tane k-bitlik tamsayılar basar bu tamsayılar da t registerının ilk $n \cdot k$ bitinden oluşur (detaylar için "Örnek Grader" bölümüne bakınız).
- t, $0 \leq t \leq m - 1$ şartını sağlamalıdır.
- Bu fonksiyona yapılan çağrılar programda kullanılan komut sayısına sayılmazlar.

Son komutu da ekledikten sonra, `construct_instructions` return etmelidir. Program daha sonra belirli bir sayıda test case ile değerlendirilir. Her bir test case n tane k-bit tamsayılardan oluşan bir $a[0], a[1], \ldots, a[n-1]$ dizisidir. Eğer programınızın çıktısı olan $c[0], c[1], \ldots, c[n-1]$ aşağıdaki koşulları sağlarsa o test case'te başarılı olmuş olur:

- Eğer $s = 0$ ise, $c[0],\ a[0], a[1], \ldots, a[n-1]$ içinden en küçük sayı olmalıdır.
- Eğer $s = 1$ ise, her bir i ($0 \leq i \leq n-1$) için, $c[i],\ a[0], a[1], \ldots, a[n-1]$ içindeki en küçük $1 + i$. tamsayı olmalıdır.

Çözümünüzün değerlendirilmesi sonucunda aşağıdaki hata mesajlarından birisini alabilirsiniz:

- `Invalid index`: Komutlardan birinde $t,\ x$ ya da y için yanlış (muhtemelen negatif) bir register indeksi verilmiştir.
- `Value to store is not b bits long`: `append_store` komutuna verilen v'nin boyu b değildir.
- `Invalid shift value`: `append_left` ya da `append_right` komutlarına verilen p'nin değeri 0 ve b arasında değildir.
- `Too many instructions`: Programınız q'dan fazla komut içermektedir.

Örnekler

Örnek 1

$s = 0,\ n = 2,\ k = 1,\ q = 1000$ olsun. Girdi olarak iki tamsayı vardır: $a[0]$ ve $a[1]$, ve her biri $k = 1$ bitten oluşmaktadır. Program çalışmaya başlamadan önce, $r[0][0] = a[0]$ ve $r[0][1] = a[1]$'dir. İşlemcideki diğer bütün bitler 0'dır. Programdaki bütün komutlar çalıştırıldıktan sonra, $c[0] = r[0][0] = \min(a[0], a[1])$ olması gerekmektedir ki bu da $a[0]$ ve $a[1]$'den en küçük olanıdır.

Programa girdi olarak verilebilecek sadece 4 farklı girdi vardır:

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

Görüldüğü üzere her 4 girdide de, $\min(a[0], a[1])$ değeri $a[0]$ ve $a[1]$'ın bitwise-AND'lenmiş haline eşittir. O nedenle olası bir çözüm programı aşağıdaki komutlarla oluşturulabilir:

1. `append_move(1, 0)`, $r[0]$, $r[1]$'e kopyalanır.
2. `append_right(1, 1, 1)`, $r[1]$'deki bütün bitler 1 bit sağa kaydırılır ve sonuc $r[1]$'e konulur. Her tamsayı 1-bit uzunluğunda olduğu için, bu işlem $r[1][0]$'in $a[1]$'a eşit olmasına yol açar.
3. `append_and(0, 0, 1)`, $r[0]$ ve $r[1]$ bitwise-AND'lenip sonuç $r[0]$'a konulur. Bu komuttan sonra, $r[0][0]$, $r[0][0]$ ve $r[1][0]$'in bitwise-AND'lenmiş hali yani istenildiği gibi $a[0]$ ve $a[1]$'in bitwise-AND'lenmiş hali olur.

Örnek 2

$s = 1,\ n = 2,\ k = 1,\ q = 1000$ olsun. Bir önceki örnekte olduğu gibi, programa verilebilecek 4 farklı girdi vardır. Bu 4 girdide de, $\min(a[0], a[1]),\ a[0]$ ve $a[1]$'nın bitwise-AND'i ve $\max(a[0], a[1])$ da $a[0]$ ve $a[1]$'nın bitwise-OR'udur. Olası bir çözüm aşağıdaki komutlarla oluşturulabilir:

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)`

Bu komutlar çalıştıktan sonra, $c[0] = r[0][0]$, $\min(a[0], a[1])$ değerini, ve $c[1] = r[0][1]$ $\max(a[0], a[1])$ değerini içerir, yani girdi sıralanmış olur.

Kısıtlar

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

Altgörevler

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

Örnek Grader

Örnek grader girdiyi aşağıdaki formatta okur:

- satır 1 : s n k q

Bunu rastgele sayıda satırlar takip eder. Her biri bir test case'i belirtir. Her bir test case aşağıdaki formatta belirtilir:

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

ve girdinin n tamsayıdan oluşan $a[0], a[1], ..., a[n-1]$ dizisi olduğunu gösterir. Test case'lerin bittiğini göstermek için test case'lerden sonra tek bir satırda -1 yer alır.

Örnek grader ilk önce `construct_instructions(s, n, k, q)` fonksiyonunu çağırır. Bu çağrı yukarıda belirlenen kısıtlara uymazsa, Implementasyon Detaylarında belirtilen hata mesajlarında birisini yazıp çıkar. Aksi takdirde, örnek grader ilk olarak programdaki her bir komutu sırayla basar. $store$ komutu için, v, 0. indeksten $b - 1$. indekse basılır.

Daha sonra örnek grader her bir test case için programı çalıştırır.

Her bir $print(t)$ komutu, i ($0 \leq i \leq n-1$) için, $d[i]$'nin değeri t registerındaki $i \cdot k$'den $(i+1) \cdot k - 1$'ye olan bitlerin gösterdiği tam sayı olan, $d[0], d[1], \ldots, d[n-1]$ tamsayı dizisini `register` t: $d[0]\ d[1]\ \ldots\ d[n-1]$ formatında basar.

Bütün komutlar çalıştırıldığında, örnek grader programın çıktısını basar.

$s = 0$ ise, her bir test case için grader'ın çıktısı aşağıdaki formattadır:

- $c[0]$.

$s = 1$ ise, ther bir test case için örnek grader'ın çıktısı aşağıdaki formattadır:

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

Bütün test case'leri çalıştırdıktan sonra, örnek grader `number of instructions:` X basar ve X programınızdaki komut sayısıdır.