Piles quàntiques X76901_ca

En aquest exercici modificarem la classe `Stack` afegint dos nous mètodes `entangle` i `disentangle` i canviant el comportament del mètode `push` com descrivim a continuació.
El nou mètode `entangle` rebrà un altre `Stack` com a paràmetre. Una crida `s0.entangle(s1)` provocarà que `s0` quedi enllaçat a `s1` de manera que, a partir de llavors, sempre que fem un `push` sobre `s0`, l'element afegit al cim de `s0` serà afegit també al cim de `s1`.
Aquest efecte no es propaga per una seqüència d'enllaços. Per exemple, si hem executat `s0.entangle(s1)` i `s1.entangle(s2)`, executar `s0.push(value)` afegirà també `value` al cim de `s1`, però això no es propagarà a afegir `value` al cim de `s2`.
Successius `entangle` sobre una mateixa pila fan que només l'últim estigui actiu. Per exemple, si hem executat `s0.entangle(s1)` i després `s0.entangle(s2)`, llavors `s0` està enllaçat a `s2` però no a `s1`.
Una crida `s0.disentangle()` cancel.larà l'efecte de l'últim `entangle` actiu.
Fixeu-vos en aquest exemple per tal d'acabar d'entendre-ho:

```
Stack<int> s0, s1;
s0.push(1);     // s0: 1
s0.push(2);     // s0: 1,2
s1.push(3);     // s1: 3
s1.push(4);     // s1: 3,4
s0.entangle(s1);
s0.push(5);     // s0: 1,2,5  s1: 3,4,5
s1.push(6);     // s0: 1,2,5  s1: 3,4,5,6
s1.entangle(s0);
s0.push(7);     // s0: 1,2,5,7  s1: 3,4,5,6,7
s1.push(8);     // s0: 1,2,5,7,8  s1: 3,4,5,6,7,8
s0.disentangle();
s0.push(9);     // s0: 1,2,5,7,8,9  s1: 3,4,5,6,7,8
s1.push(10);    // s0: 1,2,5,7,8,9,10  s1: 3,4,5,6,7,8,10
s1.disentangle();
s0.push(11);    // s0: 1,2,5,7,8,9,10,11  s1: 3,4,5,6,7,8,10
s1.push(12);    // s0: 1,2,5,7,8,9,10,11  s1: 3,4,5,6,7,8,10,12
```

D'entre els fitxers que s'adjunten en aquest exercici, trobareu `stack.hh`, a on hi ha una implementació de la classe genèrica `Stack`. Haureu d'implementar els dos nous mètodes `entangle` i `disentangle` dins `stack.hh` a la part pública de la classe (podeu trobar les capçaleres comentades dins `stack.hh`), i modificar el mètode `push` convenientment. Necessitareu també algun atribut addicional per tal de recordar si la pila té un `entangle` actiu i amb qui, amb les convenients inicialitzacions.
D'entre els fitxers que s'adjunten a l'exercici també hi ha `main.cc` (programa principal), i el podeu compilar directament, doncs inclou `stack.hh`. Només cal que pugeu `stack.hh` al jutge.
Observació: En els jocs de proves no es copiaran piles. Per tant, no cal que adapteu els mètodes que copien piles, de manera que no cal decidir si l'entanglement d'una pila s'hereta sobre una còpia.

Entrada

L'entrada del programa comença amb una declaració d'unes quantes piles d'enters (`s0, s1, ...`), i després té una seqüència de comandes sobre les piles declarades. Com que ja us oferim el `main.cc`, no cal que us preocupeu d'implementar la lectura d'aquestes entrades. Només cal que implementeu la extensió de la classe pila abans esmentada.
Podeu suposar que les comandes no faran coses extranyes, com un entangle d'una pila amb sí mateixa. També podeu suposar que les comandes faran disentangles només sobre piles que tinguin un entangle actiu. Però pot ser el cas que es faci un entangle sobre una pila que ja tingui un entangle actiu. Com mencionavem abans, en aquestes situacions només l'últim entangle aplica.

Sortida

Per a cada comanda d'escriptura sobre la sortida s'escriurà el resultat corresponent. El `main.cc` que us oferim ja fa això. Només cal que implementeu la extensió de la classe cua abans esmentada.

Exemple d'entrada 1

```
Stack<int> s0 , s1 ;
s0 .push( 1 );
s0 .push( 2 );
s1 .push( 3 );
s1 .push( 4 );
cout<< s0 <<endl;
cout<< s1 <<endl;
cout<< s0 .size()<<endl;
cout<< s1 .size()<<endl;
cout<< s0 .top()<<endl;
cout<< s1 .top()<<endl;
s0 .entangle( s1 );
s0 .push( 5 );
s1 .push( 6 );
cout<< s0 <<endl;
cout<< s1 <<endl;
cout<< s0 .size()<<endl;
cout<< s1 .size()<<endl;
cout<< s0 .top()<<endl;
cout<< s1 .top()<<endl;
s1 .entangle( s0 );
s0 .push( 7 );
s1 .push( 8 );
cout<< s0 <<endl;
cout<< s1 <<endl;
cout<< s0 .size()<<endl;
cout<< s1 .size()<<endl;
cout<< s0 .top()<<endl;
cout<< s1 .top()<<endl;
s0 .disentangle();
s0 .push( 9 );
s1 .push( 10 );
cout<< s0 <<endl;
cout<< s1 <<endl;
cout<< s0 .size()<<endl;
cout<< s1 .size()<<endl;
cout<< s0 .top()<<endl;
cout<< s1 .top()<<endl;
s1 .disentangle();
s0 .push( 11 );
s1 .push( 12 );
cout<< s0 <<endl;
cout<< s1 <<endl;
cout<< s0 .size()<<endl;
cout<< s1 .size()<<endl;
cout<< s0 .top()<<endl;
cout<< s1 .top()<<endl;
```

Exemple de sortida 1

```
2 1 2
2 3 4
2
2
2
4
3 1 2 5
4 3 4 5 6
3
4
5
6
5 1 2 5 7 8
6 3 4 5 6 7 8
5
6
8
8
7 1 2 5 7 8 9 10
7 3 4 5 6 7 8 10
7
7
10
10
8 1 2 5 7 8 9 10 11
8 3 4 5 6 7 8 10 12
8
8
11
12
```

Exemple d'entrada 2

```
Stack<int> s0 , s1 , s2 ;
s1 .push( -20 );
s2 .push( -1 );
cout<< s0 .size()<<endl;
s1 .push( 7 );
s2 .pop();
s2 .entangle( s0 );
cout<< s0 <<endl;
s1 .pop();
s2 .push( 0 );
s2 .push( -15 );
cout<< s1 .top()<<endl;
s1 .pop();
s0 .push( 8 );
s2 .disentangle();
s1 .push( 7 );
cout<< s1 .size()<<endl;
s0 .push( 7 );
s2 .push( 18 );
s2 .pop();
s0 .entangle( s2 );
cout<< s1 .size()<<endl;
s2 .entangle( s1 );
cout<< s2 .top()<<endl;
cout<< s1 .size()<<endl;
cout<< s0 <<endl;
s0 .push( -9 );
s1 .entangle( s0 );
s2 .push( 9 );
cout<< s0 <<endl;
cout<< s2 <<endl;
s2 .push( 13 );
s2 .disentangle();
s0 .pop();
s2 .pop();
s0 .pop();
s1 .push( -7 );
cout<< s2 <<endl;
s1 .disentangle();
s1 .pop();
s0 .pop();
s0 .push( -14 );
s1 .push( 3 );
s2 .pop();
cout<< s0 .top()<<endl;
s1 .push( 16 );
cout<< s2 .size()<<endl;
s1 .push( -11 );
s1 .push( -20 );
s0 .push( 16 );
s0 .pop();
s0 .push( -9 );
s1 .push( -1 );
cout<< s2 .size()<<endl;
s0 .disentangle();
s1 .entangle( s0 );
s2 .push( 13 );
s2 .pop();
s0 .push( 17 );
cout<< s0 <<endl;
s2 .push( -5 );
s0 .push( -8 );
s1 .pop();
s0 .push( -19 );
s2 .entangle( s0 );
s2 .push( -12 );
s1 .entangle( s2 );
s1 .push( -8 );
s2 .entangle( s0 );
cout<< s1 .top()<<endl;
s1 .push( 1 );
s2 .push( -11 );
s2 .entangle( s0 );
cout<< s2 .size()<<endl;
s2 .push( -10 );
cout<< s1 <<endl;
s2 .pop();
s1 .push( 18 );
s2 .push( 6 );
s1 .push( -2 );
s0 .push( -20 );
cout<< s0 <<endl;
cout<< s1 <<endl;
cout<< s2 <<endl;
```

Exemple de sortida 2

```
0
0
-20
1
1
-15
1
4 0 -15 8 7
5 0 -15 8 7 -9
4 0 -15 -9 9
```

```
4 0 -15 -9 9
-14
4
6
6 0 -15 8 -14 -9 17
-8
11
9 7 9 13 3 16 -11 -20 -8 1
13 0 -15 8 -14 -9 17 -8 -19 -12 -11 -10 6 -20
11 7 9 13 3 16 -11 -20 -8 1 18 -2
14 0 -15 -9 9 16 -9 -5 -12 -8 1 -11 18 6 -2
```

Observació

Avaluació sobre 10 punts:

- Solució lenta: 5 punts.
- solució ràpida: 10 punts.

Entenem com a solució ràpida una que és correcta, on totes les operacions tenen cost **CONSTANT** (en part gràcies a que treballem amb piles d'enters), i capaç de superar els jocs de proves públics i privats. Entenem com a solució lenta una que no és ràpida, però és correcta i capaç de superar els jocs de proves públics.

Informació del problema

Autoria: PRO2

Generació: 2026-01-27T18:55:40.506Z