Nombre de fulles d'un arbre — X12047_ca

Implementeu una funció **RECURSIVA** que, donat un arbre binari d'enters, retorna el nombre de fulles que té. Això significa que retorna el nombre de nodes de l'arbre tals que els seus dos fills son buits. Aquesta és la capcelera:

```
// Pre:
// Post: Retorna el nombre de fulles de t
int numberOfLeaves(BinTree<int> t);
```

Aquí tenim un exemple de paràmetre d'entrada de la funció i la corresponent sortida:

```
t:            3
              |
       _______ _______
      |               |
      1               4
      |               |
 ____ ____       ____
|         |     |
2         5     1

=>

3
```

Fixeu-vos que l'enunciat d'aquest exercici ja ofereix uns fitxers que haureu d'utilitzar per a compilar: `main.cc, BinTree.hh, numberOfLeaves.hh`. Us falta crear el fitxer `numberOfLeaves.cc` amb els corresponents `includes` i implementar-hi la funció anterior. Només cal que pugeu `numberOfLeaves.cc` al jutge.

Entrada

La primera linia de l'entrada descriu el format en el que es descriuen els arbres, o bé INLINEFORMAT o bé VISUALFORMAT. Després venen un nombre arbitrari de casos. Cada cas consisteix en una descripció d'un arbre un arbre binari d'enters. Fixeu-vos en que el programa que us oferim ja s'encarrega de llegir aquestes entrades. Només cal que implementeu la funció abans esmentada.

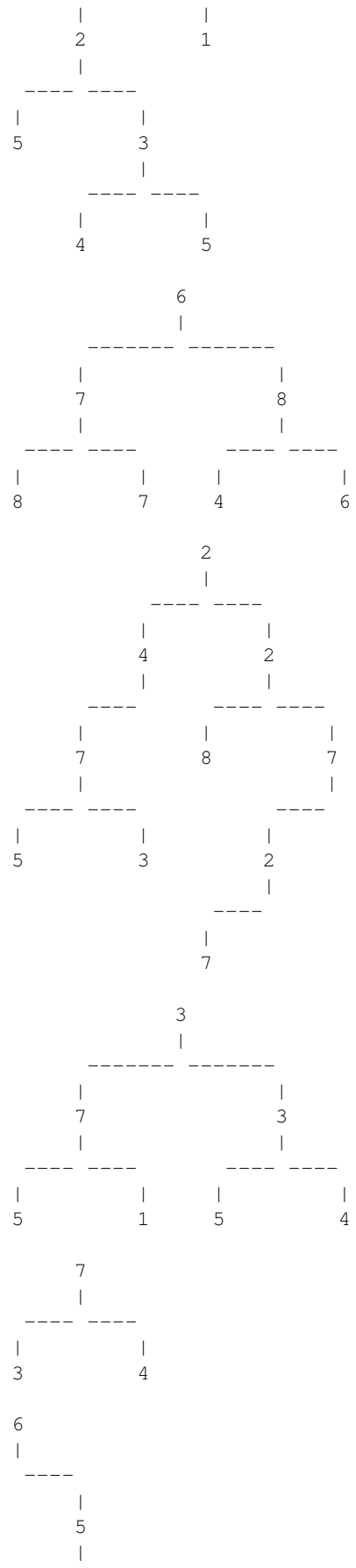
Sortida

Per a cada cas, la sortida conté el corresponent nombre de fulles de l'arbre. Fixeu-vos en que el programa que us oferim ja s'encarrega d'escriure aquesta sortida. Només cal que implementeu la funció abans esmentada.

Exemple d'entrada 1

```
VISUALFORMAT
```

```
    7
    |
 ____ ____
```



Exemple de sortida 1

```
4
4
4
4
```

```
2
2
1
2
3
1
```

Exemple d'entrada 2

```
INLINEFORMAT
0(55(29,-47(-15,98)),-18)
-94(82(-21,80),-16(63,-85))
-27(-50(6(13,-56),),23(2,36(-2(-37,),)))
-56(-5(-100,-37),7(-70,-18))
5(-3,-32)
50(,-23(-17,91))
41
91(59(75,-46),)
55(,62(-31(-10,69),-74(67,)))
-56
12(96(-22(88,),31(15,-92)),-47(70,))
-58(4,-1(27,-35))
78
-91(89(35(-95,-24),-50(,77)),-95)
-69
89(-93(,-72),-31(-76,-91))
-25(93,76)
32(-71,73(-68(,-12(,-70)),-86(-61(-68,58),-39)))
68(-10(22,60),91)
89(-7(-20,37),)
```

Exemple de sortida 2

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

Observació

La vostra funció i subfuncions que creeu han de treballar només amb arbres. Heu de trobar una solució **RECURSIVA** del problema. Avaluació sobre 10 punts:

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

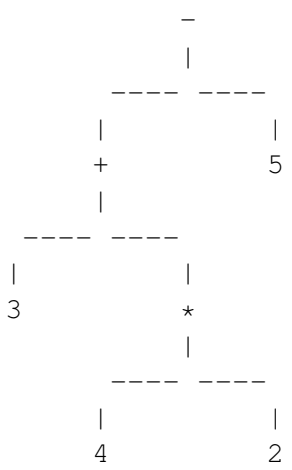
Entenem com a solució ràpida una que és correcta, de cost lineal i capaç de superar els jocs de proves públics i privats. Entenem com a solució lenta una que no és ràpida, però és correcta i capaç de superar els jocs de proves públics.

Informació del problema

Autor : PRO2
Generació : 2023-10-27 14:21:09

Nombre d'expressions amb avaluació negativa — X30191_ca

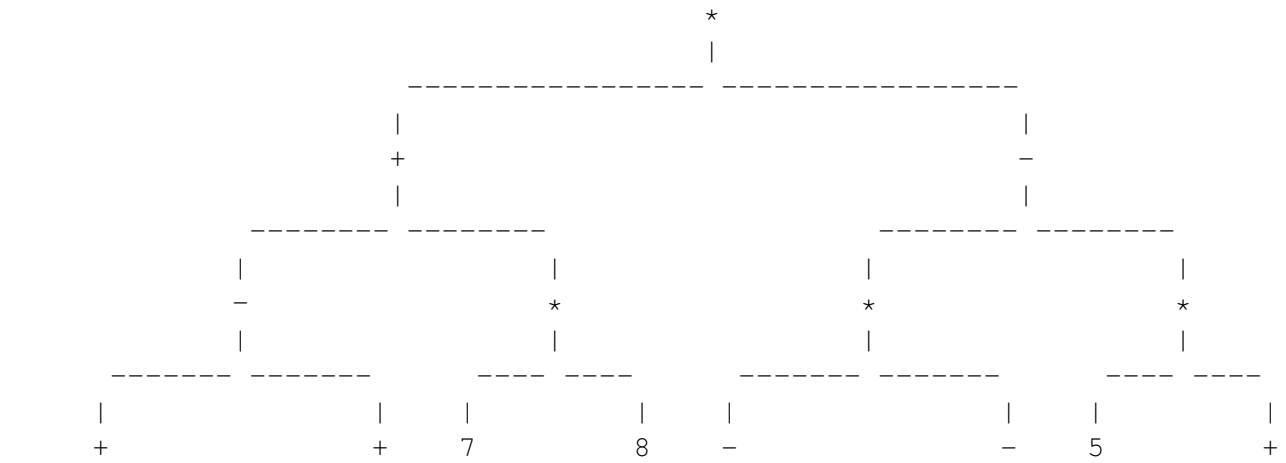
INTRODUCCIÓ:
En aquest exercici considerarem arbres que representen expressions sobre els operadors `+`, `-`, `*`, i sobre operands naturals. Per exemple, el següent arbre representa l'expressió `3+4*2-5`.

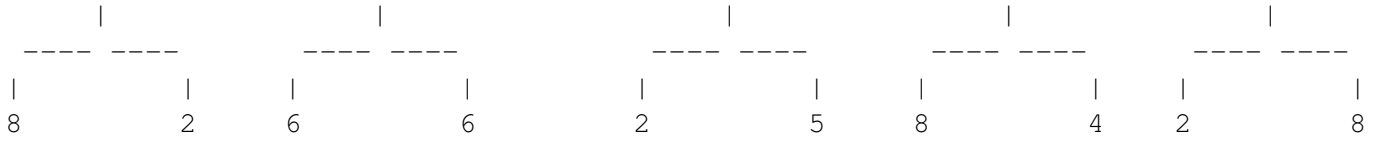


EXERCICI:
Implementeu una funció que, donat un arbre binari d'strings que representa una expressió correcta sobre naturals i operadors `+`, `-`, `*`, retorna el nombre de subexpressions tals que la seva avaluació és estrictament menor que 0. Aquesta és la capcelera:

```
// Pre:  t és un arbre no buit que representa una expressió correcta
//       sobre els naturals i els operadors +,-,*.
//       Les operacions no produeixen errors d'overflow.
// Post: Retorna el nombre de subexpressions de l'expressió representada per t
//       amb avaluació estrictament menor que 0.
int numNegative(BinTree<string> t);
```

Aquí tenim un exemple de paràmetre d'entrada de la funció i la corresponent sortida:





=>

5

Fixeu-vos que l'enunciat d'aquest exercici ja ofereix uns fitxers que haureu d'utilitzar per a compilar: `main.cc`, `BinTree.hh`, `numNegative.hh`, `utils.hh`, `utils.cc`. Us falta crear el fitxer `numNegative.cc` amb els corresponents `includes` i implementar-hi la funció anterior. Valdrà la pena que utilitzeu algunes de les funcions oferides a `utils.hpp`. Només cal que pugeu `numNegative.cc` al jutge.

Entrada

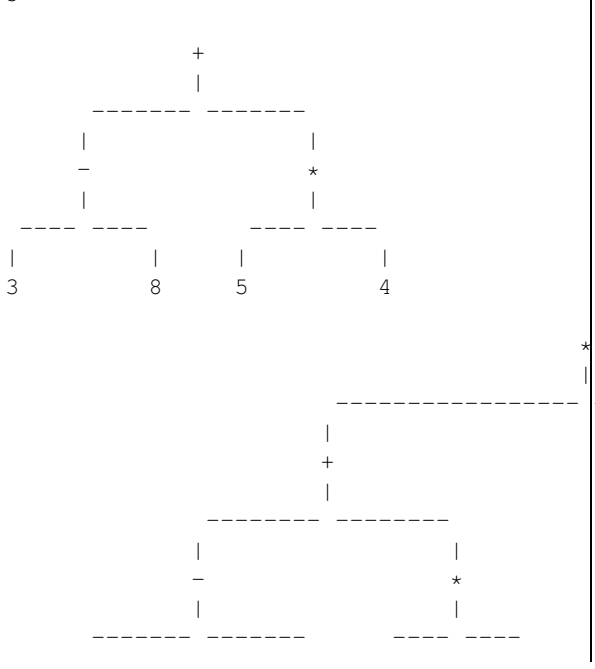
La primera linia de l'entrada descriu el format en el que es descriuen els arbres, o bé INLINEFORMAT o bé VISUALFORMAT. Després venen un nombre arbitrari de casos. Cada cas consisteix en una descripció d'un arbre un arbre binari d'enters. Fixeu-vos en que el programa que us oferim ja s'encarrega de llegir aquestes entrades. Només cal que implementeu la funció abans esmentada.

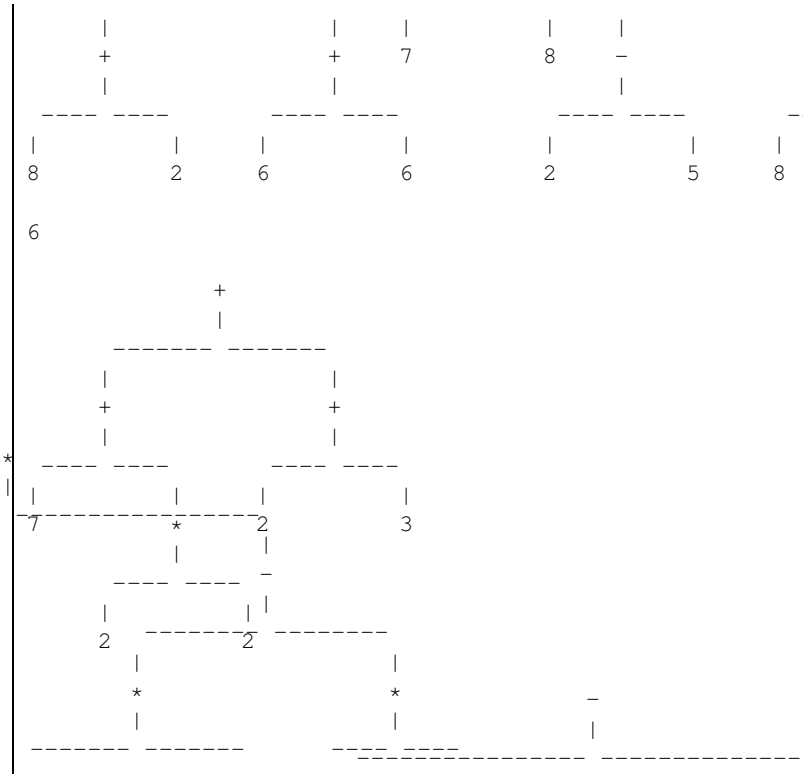
Sortida

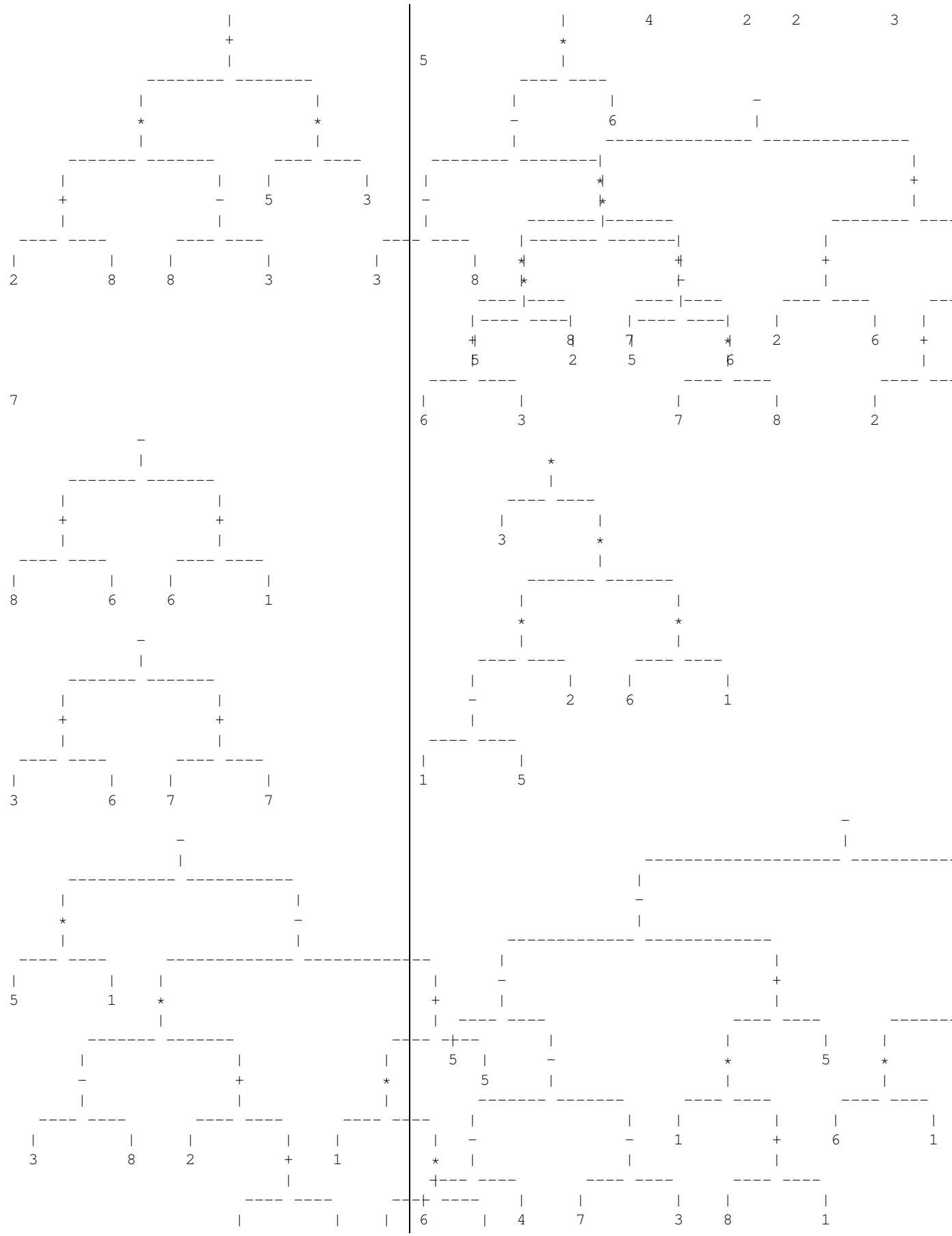
Per a cada cas, la sortida conté el corresponent nombre de subexpressions negatives. Fixeu-vos en que el programa que us oferim ja s'encarrega d'escriure aquesta sortida. Només cal que implementeu la funció abans esmentada.

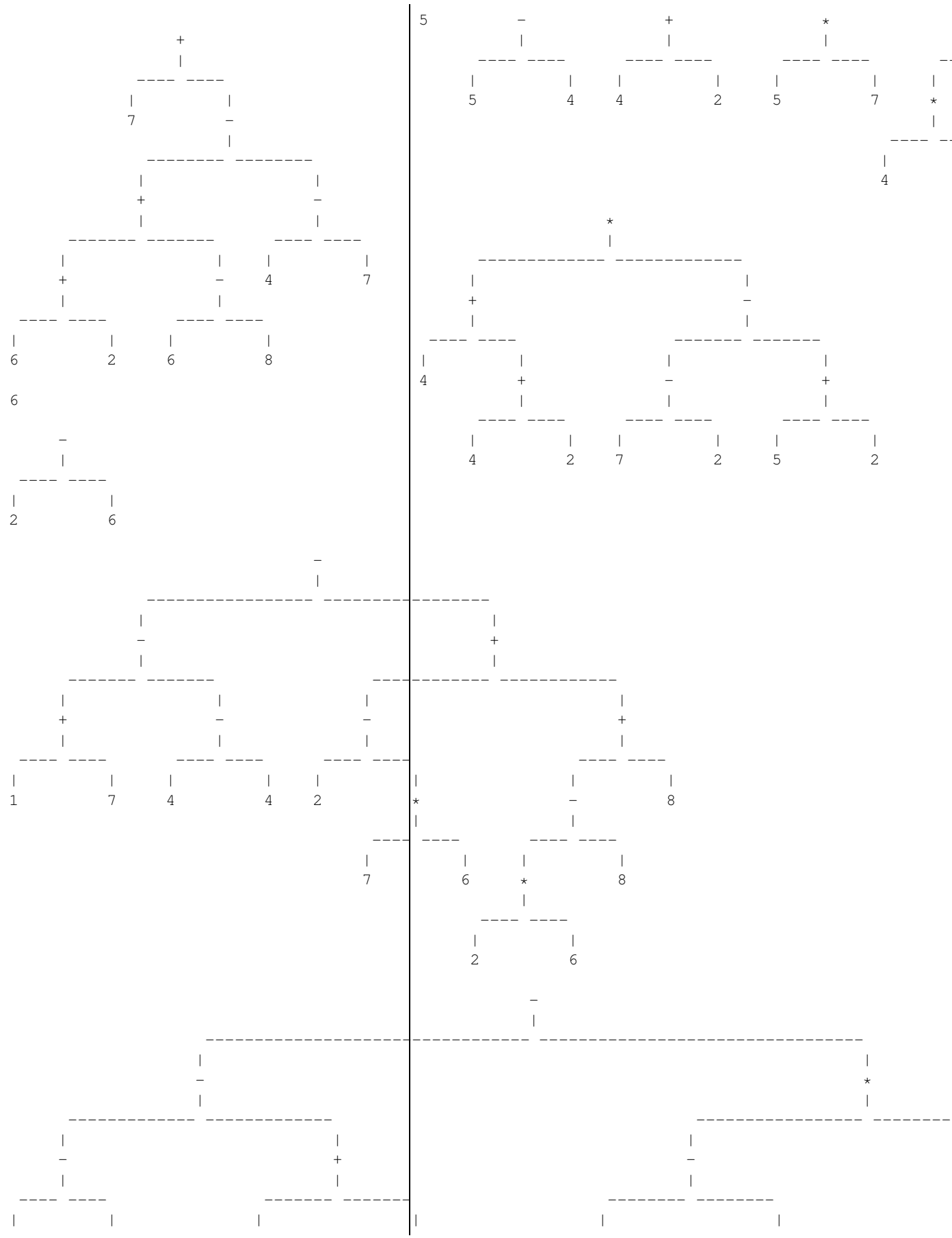
Exemple d'entrada 1

```
VISUALFORMAT
3
```









Exemple de sortida 1

```
0
1
5
0
0
3
0
0
1
```

```
3
0
1
4
4
2
0
1
2
4
2
```

Exemple d'entrada 2

```
INLINEFORMAT
3
+(-(3,8),*(5,4))
*(+(-(+(8,2),+(6,6)),*(7,8)),-(*(-(2,5),-(8,4)),*(5,+(2,8))))
6
+(+(7,*(2,2)),+(2,3))
-(+(*(+(2,8),-(8,3)),*(5,3)),*(-(-(3,8),*(5,2),-(5,6))),6))
7
-(+(8,6),+(6,1))
-(+(3,6),+(7,7))
-(*(5,1),-(*(-(3,8),+(2,+(4,2))),+(*(1,*(2,3)),5)))
5
-(*(*(+(6,3),8),+(7,*(7,8))),+(+(2,6),+(+(2,7),-(3,5))))
*(3,*(*(-(1,5),2),*(6,1)))
-(-(-(5,-(-(6,4),-(7,3))),+(*(1,+(8,1)),5)),-(+(*(6,1),-(5,-(4,2))),-(6,7)))
+(7,-(+(+(6,2),-(6,8)),-(4,7)))
6
-(2,6)
-(-(+(1,7),-(4,4)),+(-(2,*(7,6)),+(-(*(2,4),8),8)))
-(-(-(5,-(5,4)),+(+(4,2),*(5,7))),*(-(-(*(2,4),*(7,8)),-(2,2)),-(*(+(1,5),-(5,3)),+(3,3))))
*(+(4,+(4,2)),-(-(7,2),+(5,2)))
```

```
0
1
5
0
3
0
1
3
1
4
2
4
0
1
2
4
```

Observació

La vostra funció i subfuncions que creeu han de treballar només amb arbres. Heu de trobar una solució **RECURSIVA** del problema. Avaluació sobre 10 punts:

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

Entenem com a solució ràpida una que és correcta, de cost lineal 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

Autor : PRO2
Generació : 2023-10-25 18:50:15

Nombre de persones amb monedes — X38065_ca

Heu d'implementar un programa que manegui un conjunt de persones, i cuantes monedes té cada persona. El conjunt està inicialment buit. Tindrem instruccions per afegir una persona al conjunt i una indicació de cuantes monedes té. També tindrem instruccions per a eliminar una persona del conjunt. I també tindrem instruccions que ens demanen que escrivim per la sortida cuantes persones hi ha en un moment donat en el conjunt amb una certa cuantitat de monedes fixada. Fixeu-vos en la descripció de l'entrada i la sortida d'aquest exercici per a veure'n els detalls.
Observació: Podeu seguir l'enfoc que considereu oportú, i podeu utilitzar qualsevol de les estructures de dades presentades al curs (**string, vector, stack, queue, list, map**) de la manera que considereu oportuna. Però tingueu en compte que la vostra elecció pot afectar a l'eficiència de la vostra solució, i per tant al fet de poder superar tots els jocs de proves o només els públics (cosa que us deixarà amb la meitat de la nota).

Entrada

Cada linia de l'entrada consisteix en una instrucció del següent tipus, a on `name` és un string no buit qualsevol de menys de 20 caràcters, i numcoins és un natural positiu qualsevol que cap en una variable de tipus `int`:

- `ADD name numcoins`
- `DELETE name`
- `NUMPEOPLE numcoins`

La instrucció `ADD name numcoins` ens demana que afegim algú anomenat `name` al conjunt, i que aquesta persona nova té `numcoins` monedes.
La instrucció `DELETE name` ens demana que eliminem algú anomenat `name` del conjunt.
La instrucció `NUMPEOPLE numcoins` ens demana que escrivim per la sortida el nombre actual de persones del conjunt que tenen exactament `numcoins` monedes.
Podem suposar que les entrades son tals que els noms de persones s'utilitzen com a molt un cop i de forma coherent, és a dir: donat un `name` concret, podem suposar que hi haurà com a molt una instrucció `ADD name numcoins` i com a molt una instrucció `DELETE name`. A més a més, si apareix una instrucció `DELETE name`, necessàriament haurà aparegut abans un `ADD name numcoins` amb el mateix `name`.

Sortida

Per a cada instrucció `NUMPEOPLE numcoins`, s'escriurà una línia per la sortida amb el nombre actual de persones que tenen `numcoins` monedes.

Exemple d'entrada

```
NUMPEOPLE 10
NUMPEOPLE 15
ADD laura 46
ADD david 46
ADD sonia 46
DELETE laura
ADD carles 46
ADD sandra 10
ADD joel 15
DELETE sonia
ADD nuria 46
```

```
NUMPEOPLE 15
ADD hector 15
NUMPEOPLE 46
ADD merce 46
ADD oscar 15
NUMPEOPLE 15
DELETE joel
DELETE merce
DELETE hector
NUMPEOPLE 46
DELETE oscar
DELETE david
DELETE carles
DELETE sandra
NUMPEOPLE 10
NUMPEOPLE 46
DELETE nuria
NUMPEOPLE 46
```

Exemple de sortida

```
0
0
1
3
3
3
0
1
0
```

Observació

Avaluació sobre 10 punts:

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

Entenem com a solució ràpida una que és correcta, de cost $n \log(n)$ o inferior, 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

Autor : PRO2
Generació : 2023-10-25 18:39:00

Canviar paréntesis i corxets de tancar per a produïr una seqüència ben parentitzada X62454_ca

Preliminars:
En aquests preliminars expliquem qué és un mot ben-parentitzat sobre `(,),[,]`. Si ja teniu clar aquest concepte, podeu deixar de llegir els preliminars i anar directament a l'exercici en sí.
Un mot ben parentitzat és un string `s` format amb els caràcters d'obrir i tancar parèntesis o corxets, és a dir `(,),[,]`, que cumpleix el següent. A base de reemplaçar la subseqüència `()` pel mot buit, i la subseqüència `[]` pel mot buit, tant com sigui possible, *s* s'acaba convertint en el mot buit.
Per exemple, considereu el mot `(()[()[]])`. Si reemplacem les dues ocurrències de `()` pel mot buit ens queda `([[]])`. Ara, si reemplacem la ocurrència de `[]` pel mot buit ens queda `([])`. Ara, si reemplacem la nova ocurrència de `[]` pel mot buit ens queda `()`. Finalment, si reemplacem la nova ocurrència de `()` pel mot buit ens queda el mot buit. Per tant, el mot inicial era ben parentitzat.
Considereu aquest altre exemple: `([()[])]`. Si reemplacem la ocurrència de `()` i la ocurrència de `[]` per mots buits ens queda `([)]`. Aquí ja no podem aplicar cap més reemplaçament. Per tant, el mot inicial no era ben-parentitzat.
Exercici:
Tindrem strings d'entrada que poden no ser ben-parentitzats, però que segur que es poden convertir en ben-parentitzats a base de canviar alguns `)` per `]`, i canviar alguns `]` per `)`. Heu d'implementar un programa que, per a cada cas, indica quants canvis s'han de fer per tal que es converteixi en ben-parentitzat.
Observació: Podeu seguir l'enfoc que considereu oportú, i podeu utilitzar qualsevol de les estructures de dades presentades al curs (**string, vector, stack, queue, list, map**) de la manera que considereu oportuna. Noteu, però, que enfocaments diferents poden donar lloc a solucions més o menys eficients, i que superin només els jocs de proves públics o tots els jocs de proves, de manera que la nota acabarà depenent d'això.

Entrada

L'entrada conté un nombre arbitrari de casos, un per línia. Cada cas consisteix en un string no buit sobre `(,),[,]` tal que, canviant alguns `)` per `]` i canviant alguns `]` per `)` es pot convertir en ben-parentitzat. Pot ser el cas que l'string ja sigui ben-parentitzat i que per tant calguin 0 canvis.

Sortida

Per a cada cas, escriviu en una línia el nombre de canvis que calen per a convertir l'string en ben parentitzat.

Exemple d'entrada 1

```
()
[)
()
[]
```

```
([)]
[(])
([])
[()]
()[)(][]
((()()][()())
```

```
(((](([])][))[(]]
((([](([]))[])[()]
(][()))(()()](([()())[()()))
[([](([]))[])[(]][([](([]))[])[())
(()(())(()())(((()())(()()))))
[[](())(()())(((()())(()()]]]
[[][[]][[][]][[][]][[][]]]]
```

Exemple de sortida 1

```
1
1
0
0
2
2
0
0
2
2
4
0
5
4
0
2
0
```

Exemple d'entrada 2

```
((()))()
[[][[]))(([]])()[())
[[()]]()[]
[(((])))[[))
[)
[[[]]()[)())[([])]
[(())((])](])
((()))()
[]
[]
[())[[]][][][([]]]
((((()))(])()()[]
[)
([)()[[[]][)])(][]
[][]
()
((]][)()[](]
[](((])])
(([]][])
[][)([]][]
[(()(())](([[][())])]
[][]
[[[]][]](())
([][()](]
[[])[][)
[])()([[[)]))())()
(())()()
(]
([()()])()
[()]
(()()](][([])[[])](()
[(())
([])())([][))
[[(]]([()])][)
()
[[[](]]()())
((((()))([]][([])))
([[[]]()]()()())[[])
[]()
[[)(][))()
```

Exemple de sortida 2

```
0
4
0
4
1
2
1
0
0
0
3
1
1
3
0
0
4
1
1
2
4
0
0
1
2
4
0
1
0
0
3
2
1
2
0
2
2
2
0
4
```

Observació

Avaluació sobre 10 punts:

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

Entenem com a solució ràpida una que és correcta, de cost lineal 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

Autor : PRO2
Generació : 2023-10-25 18:54:51