CIFRAS

Usaríamos un árbol general para resolver esta práctica. No binaria, ya que necesitaría mas de 2 hijos en cada nodo. El procedimiento sería el siguiente:

Se crea una lista con 6 valores escogidas aleatoriamente de los números posibles, y se randomiza el valor al que se tiene que acercar.

Se define la estructura que va a usar el árbol como:

```
struct dato_cifra{
  char operacion;
  int numero;
  int resultado;
  list<int> numeros_restantes;
};
```

Los valores de la estructura en el nodo raíz dan igual. En el primer nivel de los nodos hijos, resultado es uno de los valores de la lista de valores. Es decir, es el primer número con el que se va a trabajar. La lista numeros_restantes contendrá los números aun no usados, en este caso serán 5.

En el main se crea el nodo raíz y los 6 hijos. Luego crearemos una función recursiva llamada siguiente_funcion(nodo &n). El funcionamiento es tal:

1- Si no quedan números en la lista numeros_restantes, termina inmediatamente la función.
2- Sino, entra en un bucle, usando un list<int>::iterator para pasar por todos los elementos de la lista numeros_restantes del nodo.
3- Por cada elemento en la lista, se crean hasta 4 hijos posibles del nodo:
    uno para la operación – (se comprueba primero que no resulte en un número negativo)
    uno para la operación +
    uno para la operación / (se comprueba primero que no resulte en un número no entero)
    uno para la operación *

los valores del struct de los nodos creados será el siguiente:
    operación = operación usada (‘-’, ‘+’, ‘/’, ‘*’)
    numero = número que se ha usado en la operación sobre el resultado del nodo anterior
    resultado = el resultado de la operación
    numeros_restantes = lista con los números que quedan sin usar (lista del padre menos el num usado en la operación)
4- Al cerrarse el bucle, se tienen todas las operaciones con los números posibles como hijos. Se invoca entonces la función siguiente_funcion de nuevo, una vez para cada nodo hijo.

Al haber terminado la función, tenemos un árbol con todas las combinaciones de números y operaciones posibles. Con un iterador, se va buscando el que más se acerque al número al que se tiene que acercar. Al encontrarlo, guardamos los valores de éste, el de su padre, el del padre del padre, etc. hasta llegar al root. Así obtenemos todos los pasos hasta llegar a ese número.

Nota: se guardan primero los últimos pasos, hasta llegar al primero, por lo que conviene usar

un contenedor LIFO para guardar y sacar los datos.

Debemos tener en cuenta que si atacamos el problema por fuerza bruta como hemos mencionado anteriormente crearemos un árbol muy grande ya que dependiendo de la forma en que tomemos los números en un conjunto de 6 números por ejemplo tendríamos la cantidad de 720 formas diferentes (6!), después de esto debemos tener en cuenta el orden en el que podemos tomar las operaciones y los operadores que podemos utilizar ya que pueden desde hasta repetirse hasta ser totalmente diferente lo que nos deja 1024 formas diferentes de operar (4^5 4 operadores distintos usados en 5 operaciones ej: “+++++”, “+-x-/”)

Combinando ahora las maneras de tomar los números y las posibles maneras de tomar los operadores nos da que hay 720x1024=737.280 formas diferentes de realizar los cálculos pero aun así todavía debemos de tener en cuenta la prioridad de operadores y el uso de los paréntesis, calcular las distintas formas de poner los paréntesis es algo mas complicado de calcular por lo cual deberíamos de expresar las operaciones en notación polaca y llegaríamos a la conclusión de que habría 42 combinaciones posibles lo que nos da como resultado final 30.965.760 formulas distintas.

Aun así, podríamos acortar este numero identificando patrones no validos como caminos que multiplican o dividen 0,y saltando loops de numeros repetidos.

Alejandro Nieto Alarcón y Yoram Joel Slot