Полоска-21

Решение.

Казалось бы, а при чём тут бинарные деревья? ☺

Те, кто регулярно следит за RunSite знают, что задачи идут сериями. И что две предыдущие задачи были довольно похожи, и обе решались с помощью бинарных деревьев. Извините, если я невольно ввёл таким образом кого-то в заблуждение – текущая серия задумана как серия о «структурах данных». И в данной задаче бинарных деревьев нет. А есть связанный список. Точнее, двусвязный список. И его вполне достаточно для решения задачи.

Но в этой задаче очень удачно применим так называемый xor-связанный список – структура очень похожая на двусвязный список, хотя и не так часто встречающаяся, и я не мог упустить возможность поговорить об этой структуре.

Итак, основная идея простая и ясная – каждая колонка чисел в сложенной полоске представляет из себя список чисел. Определить, какое число находится на вершине колонки легко. А, учитывая, что числа, расположенные в верхних клетках каждой колонки, идут подряд, то и хранить их тоже легко, и это не занимает много места – достаточно хранить только верхние числа в самой левой и самой правой колонках. Сразу заметим, что и числа, расположенные в нижних клетках колонок, также идут подряд.

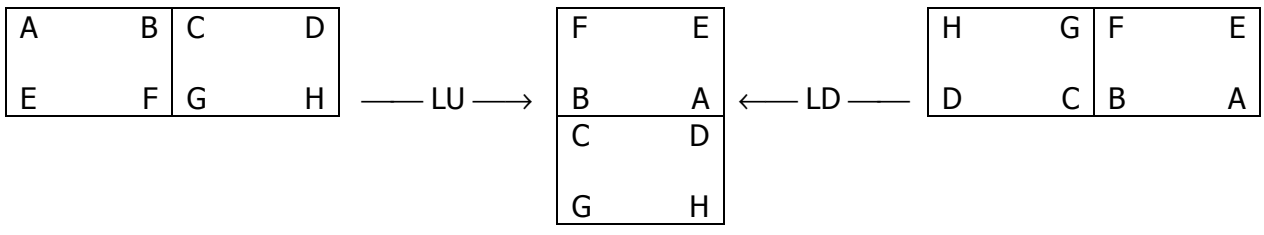
Вдоль колонки надо будет пробежать (конечно, сделать это надо только один раз – в самом конце, - но это довольно важный момент в задаче ☺) – храним колонку, как двусвязный список: каждое число хранит обоих своих соседей. А если какого-то соседа нет – клетка находится на верхнем или нижнем крае колонки – полагаем одного из соседей равным 0.

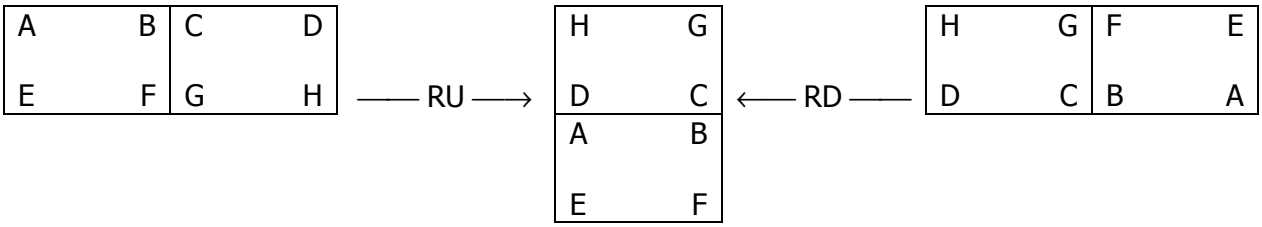
При сгибаниях пары колонок склеиваются – значит надо иметь возможность быстрого и, желательно, простого доступа к нижнему элементу каждой колонки. Для этого будем хранить для каждого числа (кроме двух его соседей, конечно), ещё и число, расположенное на противоположном конце той колонки, в которой наше число расположено на одном из концов. Если число «завернулось» в середину колонки, то эта информация больше не понадобится, ну, и ладно, пусть тогда горит она синим пламенем ☺.

В исходной ситуации оба соседа каждого числа – нулевые, каждое число является верхним в своей колонке, а на противоположном конце каждой колонки лежит само это число.

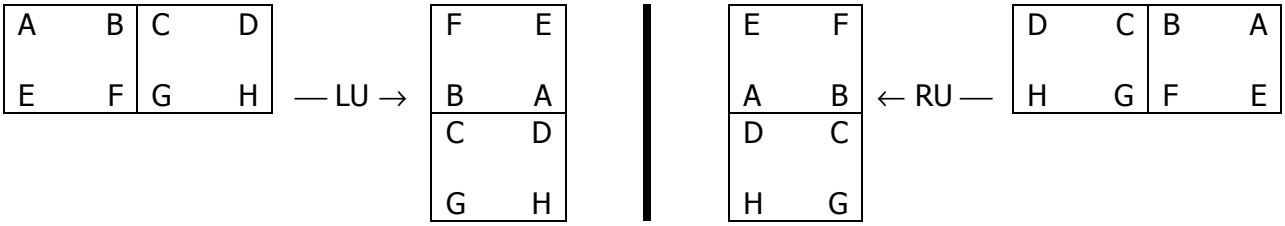
Склеивание пар колонок происходит обычным образом – ничего особенного. Разве что пара небольших технических приёмов, позволяющих немножко сократить код и, на мой взгляд, делающих его понятнее.

Во-первых, заметим, что сгибания вверх можно заменить переворачиванием полоски на 180^{o} с последующим сгибанием вниз. Это хорошо видно на следующих двух схемах:





Во-вторых, немного по разному приходится обрабатывать случаи, когда число в левом верхнем углу меньше числа в правом верхнем углу и когда оно больше. Чтобы объединить эти два случая мы мысленно отражаем полоску относительно вертикально стоящего зеркала, совершаем с зеркальным изображением соответствующий сгиб, а затем результат сгиба отражаем зеркально обратно. Конечно при этом сгиб LU становится сгибом RU и наоборот – на схеме ниже это хорошо видно. А сгибы вниз нас уже не интересуют.



Ну, и напоследок, самое вкусное – xor-связанный список. Будем для каждого числа хранить не двух его соседей, скажем p1 и p2, а их xor-сумму (которую будем, как это принято обозначать знаком $\oplus$) $p=p1 \oplus p2$. И теперь, помня, что $a \oplus b \oplus b=a$, имеем $p2=p \oplus p1$, и $p2=p \oplus p1$, т.е., зная xor-сумму соседей и одного из них, легко найти второго соседа. В начале каждого списка один из соседей равен 0 и, отталкивась от этого, мы можем восстановить весь список!

В прилагаемых pas-файлах приводятся обе реализации: в файле **`ribbon21_2.pas`** реализован вариант с обычным двусвязным списком, в файле **`ribbon21_1.pas`** – с xor-связанным списком. Заметим, что второй вариант не только существенно экономичнее по памяти, но и работает побыстрее – всё-таки проверки, какой из указателей на соседей уже занят, а какой ещё свободен, занимают некоторое ощутимое время.

Понятно, что сложность описанного алгоритма равна $O(2^N)$ – ведь склейка двух колонок в одну занимает константное время, а всего будет ровно 2^N-1 склейка.

Для сравнения в файле **`ribbon21_A.pas`** приводится и совсем простая алгоритмически программа: в ней для каждого числа хранятся номер колонки и номер «слоя» в котором оно находится, и при каждом сгибе эти величины пересчитываются по очевидным формулам. Понятно, что сложность этого алгоритма равна $O(N \cdot 2^N)$ – многовато.

Учебных материалов в это раз нет. Я ограничусь двумя ссылками на статьи в Wiki: английскую -

http://en.wikipedia.org/wiki/XOR_linked_list

и русскую-

http://ru.wikipedia.org/wiki/Xor-%D1%81%D0%B2%D1%8F%D0%B7%D0%B0%D0%BD%D0%BD%D1%8B%D0%B9_%D1%81%D0%BF%D0%B8%D1%81%D0%BE%D0%BA

Оттуда можно пройтись и посмотреть другие линейные структуры данных.