Анализ на задача `maxor`

Тагове: дърво, побитови операции, STL set, ordered set, компресия, префиксни суми, binary trie

Нулева подзадача – 0 точки

Тази подзадача е оставена за обратна връзка от системата.

Подзадача 1 – 12 точки

Можем да пуснем DFS от всеки връх, за да сметнем цената на пътя до всеки друг връх и да вземем максималната.

Сложност: $O(N^2)$

Реализация: `maxor_mitko_12p.cpp`, `maxor_kiki_12p.cpp`

Подзадача 2 – 6 точки

Нека $dist_{i+1}$ е XOR-ът на числата $w_1, w_2, \cdots, w_i$. Тогава XOR-ът на подмасива $[l; r]$ в w е $dist_{r+1} \oplus dist_l$. Това е така, защото $dist_{r+1}$ включва индексите до r включително, а $dist_l$ включва индексите до $l-1$ включително, и когато намерим техния XOR, числата, които са извън интервала, са включени два пъти и се изключват взаимно.

Всъщност, тъй като дървото е пръчка, $dist_i$ е точно цената на пътя от връх 1 до връх i, а $dist_{r+1} \oplus dist_l$ е точно цената на пътя от l до r. Така задачата ни се свежда до това да намерим максималния XOR между два елемента в редицата $dist$.

В тази подзадача се възползваме, че има сравнително малко на брой различни стойности на $dist$, като фиксираме отговора x и обхождаме всяко възможно a, проверявайки дали съществува $dist_b = dist_a \oplus x$.

Сложност: $O(N . \max w_i)$

Реализация: `maxor_mitko_6p.cpp`, `maxor_kiki_6p.cpp`

Подзадача 3 – 16 точки

Отново се възползваме от малкия брой различни стойности, но този път вместо да фиксираме стойност и индекс, можем да фиксираме директно двете стойности. При $w_i \leq 2^{15}$ това звучи бавно, но всъщност в най-лош случай са $\frac{(2^{16})(2^{16}-1)}{2} \approx 2.10^9$ операции и тъй като самите операции са прости, много лесно се оптимизират от компилатора.

Сложност: $O((\max w_i)^2)$

Реализация: `maxor_mitko_16p.cpp`, `maxor_kiki_16p.cpp`

Подзадача 4 – 31 точки

В тази задача трябва да съобразим, че правим побитови операции, а едновременно искаме някакъв максимум. Това ни подсказва, че можем да потърсим начин за сравняване на двоични числа и някакви техни свойства. Тук се намира ключовото наблюдение – число с бит i, равен на единица, е по-голямо от всяко едно число, което има единствено някои от по-малките битове, равни на единица (тук разглеждаме битовете от голям към малък). Лесно го доказваме: най-голямото число само с битове след позиция i, равни на единица, е $2^i - 1$, а тривиално $2^i > 2^i - 1$. Следователно ние ще приоритизираме по-големите битове, защото винаги е по-оптимално да ги вземем вместо по-малките.

След като имаме това, бихме се замислили дали всъщност можем да фиксираме нашия

отговор бит по бит. Оказва се, че можем. Нека обходим редицата, фиксирайки позицията j на едното число, което ще вземем. За всяко такова j, ще ходим по битовете от голям към малък, опитвайки се да направим този бит в XOR-а ни единица. Тоест, ако в a_j някой бит е нула, тогава ще искаме числа с единица на този бит, а в противен случай – с нула. Обърнете внимание, че тези числа, които търсим, също трябва да отговарят на миналите битове, които сме фиксирали, тоест трябва да пазим битмаска. Ако няма числа, които отговарят на тези критерии, тогава обръщаме нашия бит, тоест слагаме нула на тази позиция в XOR-а.

Единственият проблем е проверяването дали има такива числа за фиксирания XOR. Нека предварително сме си построили префиксни суми px върху масива-брояч на стойностите $dist_x$. Това, което всъщност искаме, е интервала от числа в $dist$, на които първите битове съвпадат с фиксираните от битмаската. Най-малкото число l, което отговаря на тези битове, е самата битмаска, защото надясно от тези позиции има само нули. Най-голямото число r пресмятаме, като заместим тези нули вдясно с единици. Накрая можем лесно да намерим броя на тези числа: $px_r - px_{l-1}$.

Да разгледаме един пример. Досега сме фиксирали битовете 101, а надясно има още три позиции, които не сме фиксирали. Следователно най-малкото число, което е кандидат за този отговор, е $101000_{(2)}$, а най-голямото – $101111_{(2)}$. След като пресметнем тези граници, използваме префиксните суми, за да намерим нашите кандидати. Ако няма такива, не можем да постигнем единица на тази позиция в XOR-а, и обръщаме този бит в битмаската. Накрая получаваме най-доброто a_i за нашето a_j, намираме максимума от тези $a_i \oplus a_j$ и това е отговорът ни.

Сложност: $O(\max w_i + N \log(\max w_i))$

Реализация: `maxor_mitko_31p.cpp`, `maxor_kiki_31p.cpp`

Подзадача 5 – 43 точки

Подхождаме по абсолютно същия начин, но тук префиксни суми няма да ни свършат работа. Вместо това можем да използваме много различни структури, например `set`, `ordered_set` или дори да компресираме стойностите $dist$.

Сложност: $O(N \log(\max w_i) \log N)$

Реализация: `maxor_mitko_43p.cpp`, `maxor_kiki_43p.cpp`

Подзадачи 6-9

Ще се опитаме да намерим цената на пътя от a до b в произволно дърво. Нека $dist_i$ е цената на пътя от 1 до i, което можем лесно да намерим чрез DFS. Тогава търсената цена е точно $dist_a \oplus dist_b$, тъй като това включва всички ребра по пътя им веднъж, а ребрата на върховете над тях (от 1 до LCA-то им) се включват два пъти и взаимно се изключват. Така след едно DFS задачата ни се свежда до абсолютно същото нещо – намирането на максимален XOR между две числа в редица.

Реализации:

- Подзадача 6 – `maxor_mitko_13p.cpp`, `maxor_kiki_13p.cpp`
- Подзадача 7 – `maxor_mitko_45p.cpp`, `maxor_kiki_45p.cpp`
- Подзадача 8 – `maxor_mitko_63p.cpp`, `maxor_kiki_63p.cpp`
- Подзадача 9:
  - чрез `set`: `maxor_mitko_100p_set.cpp`
  - чрез `ordered_set`: `maxor_mitko_100p_oset.cpp`
  - чрез компресия: `maxor_mitko_100p_comp.cpp`

Алтернативно решение – 100 точки

Ще подходим подобно на миналото решение – фиксираме битове от най-старшия до най-младия, но този път ще фиксираме отговора във външния цикъл. Ако в момента сме фиксирали някакъв отговор x и искаме да проверим дали е възможен, трябва да проверим дали съществуват две числа a и b, за които $a \oplus b \geq x$.

Ако $a \oplus b$ е по-голямо от x, това идва от по-малките битове, т.е. тези, които не разглеждаме. Следователно би било по-лесно напълно да игнорираме тези битове и да проверяваме дали съществуват a, b с $a \oplus b = x$. Това става лесно – фиксираме едното число a и проверяваме дали има число $a \oplus x$ (чрез свойствата на XOR $a \oplus b = x \implies b = a \oplus x$). Това можем да направим, като отново поддържаме числата в някаква структура – `unordered_map`, `unordered_set` ($O(1)$) или `set` ($O(\log N)$).

Сложност: $O(N \log(\max w_i))$ или $O(N \log(\max w_i) \log N)$

Реализация:

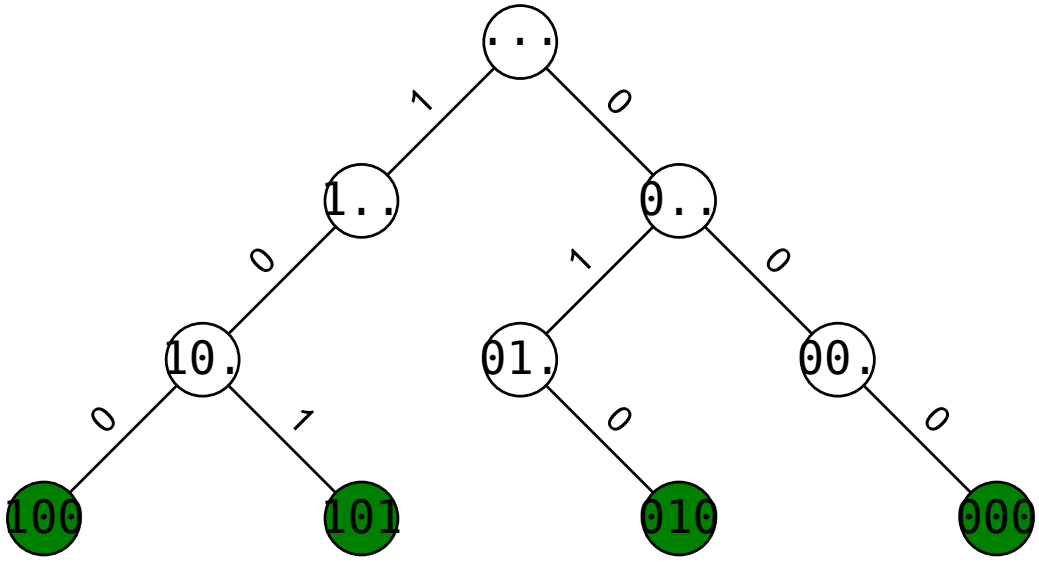
- чрез `unordered_set`: `maxor_mitko_100p.cpp`
- чрез `unordered_map`: `maxor_mitko_100p_map.cpp`, `maxor_kiki_100p.cpp`

Решение с trie – 100 точки

Това решение е по-скоро за любопитните, тъй като не е в конспекта за възрастовата група. Съществува структура, която поддържа точно това, което използваме в първото решение за 100 точки – binary trie. Накратко представлява дърво, където коренът е празен битов низ, а всеки битов низ, добавен към структурата, може да се стигне от корена. Всяко ребро представлява бит, който се добавя към низа, който представлява този връх. Дървото има най-много $O(N \log(\max a_i))$ върха и дълбочина най-много $O(\log(\max a_i))$.

Как го използваме в решението за тази задача? Лесно: поддържаме си върха, на който отговаря фиксираният ни XOR. Когато сме фиксирали бит, който искаме на другото число, на текущата позиция, проверяваме дали има такова число, като отиваме в детето на върха, което има ребро към него, равно на този бит. Ако няма такова дете, знаем, че няма такова число и ще използваме другия бит.

Малък пример: $N = 4, dist = \{100_{(2)}, 000_{(2)}, 010_{(2)}, 101_{(2)}\}$



За първия елемент фиксираме бит 0, детето съществува, после бит 1, детето съществува, после бит 1, детето не съществува и обръщаме бита, получавайки стойност $010_{(2)}$ и XOR $110_{(2)}$.

За втория елемент фиксираме бит 1, детето съществува, после бит 1, детето не съществува

и обръщаме бита, после бит 1, детето съществува и получаваме стойност $101_{(2)}$ и XOR $101_{(2)}$.

За третия елемент фиксираме бит 1, детето съществува, после бит 0, детето съществува, после бит 1, детето съществува и получаваме стойност $101_{(2)}$ и XOR $111_{(2)}$.

За четвъртия елемент фиксираме бит 0, детето съществува, после бит 1, детето съществува, после бит 0, детето съществува и получаваме стойност $010_{(2)}$ и XOR $111_{(2)}$.

Максималната от тези стойности е $111_{(2)} = 7_{(10)}$, което е отговорът ни.

Сложност: $O(N \log(\max a_i))$

Реализация: `maxor_mitko_100p_trie.cpp`

Автори: Димитър Шапатов, Кирил Зашев