Не мисля, че съм виждал задача на национално българско състезание с 14 подзадачи.

Анализ

Подзадача №0

Както винаги оставих подзадача с тестовите примери за обратна връзка от системата.

Подзадача №1

Няма нещо по-нормално от това да напишем алгоритъм за минимално покриващо дърво за задача с минимално покриващо дърво. В дадения граф има $O(\binom{n}{2}) = O(n^2)$ ребра, а два от любимите ни алгоритми за МПД (Минимално покриващо дърво) са Крускал и Прим, и двата от които изискват $O(M \log N)$ време, даващо ни $O(N^2 \log N)$ време.

Постигната сложност: $O(N^2 \log_2 N)$

Имплементация: `abstract_8p.cpp`

Подзадача №2

За тази подзадача е предвидено да се оптимизира някой от горните алгоритми. Имплементацията оптимизира Крускал – log факторът идва от сортирането на ребрата. Съответно, тъй като в задачата ребрата са с малки тегла, ние може да приложим `Count sort`, което да смъкне времето ни до $O(N^2\alpha(N))$, където $\alpha(N)$ е обратната функция на Акерман, която се използва за измерване на сложността на DSU (Union-find).

Постигната сложност: $O(N^2\alpha(N))$

Имплементация: `abstract_14p.cpp`

Подзадачи №3, 4

Третата подзадача е за $O(N^2 \log N)$ имплементация на алгоритъм за МПД, а четвъртата – за $O(N^2\alpha(N))$.

Ограниченията подсказват достатъчно – имаме голямо N и ограничения за A_i, които са симетрични на тези на първите две подзадачи, което би ни навело на мисълта, някак си, да *разкараме* няколко елемента от редицата така че впоследствие дължината ѝ да е до стойността на максималното възможно A_i, след което да опитаме да използваме алгоритъм за МПД. Една от първите идеи, която би трябвало да дойде на състезател е да се опита да *разкара* елементи, които си имат равни в редицата и впоследствие да остави до 1 копие на всяка стойност от 0 до $2^B - 1$ в редицата. Как дори бихме пробвали това? Първото нещо, което може да видим е, че за всяка стойност C, ако срещанията ѝ в редицата са $C = A_{i_1} = A_{i_2} = \cdots = A_{i_k}$, винаги ще е оптимално те да имат ребра помежду си. Именно, тъй като винаги $a|b \geq \min(a, b)$, то ако си представим, че пуснем алгоритъма на Прим от връх i_1, то със сигурност другите i_x-ове ще са му сред минималните съседи ($C|C = C$), откъдето тъй като алгоритъма твърди, че може да построи всяко едно от минималните инцидентни ребра на стартовия връх и да получи МПД, то със сигурност може да изберем друг i_x сред елементите $= C$, да построим реброто между i_1 и i_x, и да продължим с алгоритъма на Прим. Така, Прим позволява активните върхове по всяко едно време да са тези $= C$ (до изчерпването им) – ако в даден момент имаме множество от t върха $= C$, които са активни, то минималният им съсед със сигурност ще е с ребро на тегло $\geq C$, откъдето ако $t < k =$ броя върхове със стойност C, то Прим ще позволи да добавим още един връх от тези $= C$ към активните (защото към него сочи някое минимално ребро между активен и неактивен връх).

Накратко, може да свържем еднаквите по стойност върхове преди да почнем с изпълнението на алгоритъм на МПД. Впоследствие, ако трябва да свържем два върха със стойности u и v, то няма значение с кои копия на стойностите в редицата се свързват (т.е. ако редицата ни е $2\ 2\ 3\ 3$, няма значение дали ще свържем върхове с номера 1 и 4 или 2 и 3), поради което може да пуснем МПД върху редица, съдържаща по 1 копие от всяка стойност, която е съдържала редицата.

Постигната сложност: $O((2^B)^2 B)$

Имплементация: `abstract_12p.cpp`

Постигната сложност: $O((2^B)^2 \alpha(N))$

Имплементация: `abstract_21.cpp`

Подзадачи №5, 6

Петата подзадача е за $O(N^2\ log\ N)$ имплементация на алгоритъм за МПД, а шестата – за $O(N^2 \alpha(N))$.

Предишните подзадачи *чистеха* редицата от *излишни* елементи, а именно ако еднаквите. Сигурен съм, че не малко от вас са видяли друга идея, с която пак (*на практика*) *чистим излишни* елементи – а именно, ако в редицата има 0, ние директно свързваме всички елементи с 0, защото отново се минимизира OR-ът за съответните елементи (и *изтриваме* всички елементи, различни от 0). Това би ни подсказало, че горната идея се генерализира, защото и за двата вида триения използваме, че OR-ът на два елемента се минимизира, т.е. $u|v = max(u, v)$. Кога се случва това? Еми ако $u|v = v$, то на всяка позиция, на която v има нула в двоичния си код, и u има. Иначе казано, v е супермаска на u. Следва логичния въпрос – ако v е супермаска на някое u, то със сигурност по Прим аргумента ни от по-горе, има МПД, в което има ребро между v и u; тогава, има ли смисъл да свързваме v с нещо различно от u. Очевидният отговор е не – каквото и да свържем с v, ще получим OR по-голям или равен на този, който бихме получили с v, а те по начало са свързани по предния аргумент, откъдето, по логиката на Крускал, няма смисъл. Така може да *платим* цената на супермаските и да ги изтрием от редицата, след което една идея е да приложим любимия си алгоритъм за МПД. Тази операция ще я наричаме *филтриране*.

Колко числа остават? Опитайте се да се замислите за възможно най-лош пример, който ви хрумва.

Когато работих задачата и стигнах до същия въпрос, на мен ми хрумна следното – ако две различни числа a и b имат равна бройка битове, то не може a да е супермаска на b. Иначе, a ще има 1-ца на всяка позиция на която и b има, откъдето тъй като a има същия брой битове единици като b, то единиците им трябва да съвпадат, откъдето са равни, противоречие. Така, ако вземем всички числа с k единици, за някое k, то със сигурност ще образуваме такова множество.

След това пробвах различни конструкции и всички от тях се разпадаха, заради *нестабилността* на условието да няма супермаски – едно число да е в множеството означава, че няма негови супермаски и негови подмаски, което маха *много* числа с различен брой единици от текущото.

Така и реших да проверя дали някой друг преди мен е мислил по този въпрос. Оказва се, че е относително известна задача, като Теоремата на Шпернер гласи, че оптималния пример е да се вземат именно числа с равен брой битове, откъдето бройката им е $\binom{B}{K}$, ще изберем $K = \lfloor \frac{B}{2} \rfloor$. Теоремата разглежда твърдението в еквивалентната постановка за множества. И един съвет – когато анализ спомене теорема, бъдете сигурни, че разбирате защо теоремата би била вярна, дори и да не отделяте време за самото доказателство.

Добре, но колко числа наистина остават? $\frac{B!}{(\lfloor B/2 \rfloor !)(\lceil B/2 \rceil !)}$ не означава много за четащия го. Аз лично си го сметнах на компютъра и изглеждаше, че е нещо от сорта на $2^B/B$, което е достатъчно за целите на задачата. За екстра любопитните, може да сте чували, че има формула, която дава много точно приближение на $x!$, а именно $x! \approx \sqrt{2\pi x} \cdot (\frac{x}{e})^x$ по приближението на Стърлинг, откъдето с малко разписване излиза, че $\binom{B}{\lfloor \frac{B}{2} \rfloor} \approx \frac{2^B}{\sqrt{\pi * \frac{B}{2}}}$, откъдето, тъй като приближението е сходимо, може да заключим по-добрата сложност.

Постигната сложност: $O((2^B)^2)$

Имплементация: `abstract_24p.cpp`

Постигната сложност: $O(\frac{(2^B)^2}{B}\alpha(N))$

Имплементация: `abstract_30p.cpp`

По-бърз квадратен алгоритъм за МПД

Има начин да се оптимизира решението на шестата подзадача още повече. Една идея е да се използва модификация за Прим на пълни графи, кръстена – Dense Prim (Гъст Прим), защото се държи добре за пълни графи (които, наистина, са гъсти). Ще правим Прим без приоритетна опашка. Тъй като Прим N пъти търси най-добър съседен неактивен връх до компонентата на активните, ние може ръчно с $O(N)$ цикъл да намерим този връх. Може допълнително да забележим, че всеки един връх става активен точно веднъж. Заради това може с $O(N)$ цикъл да обходим неактивните върхове за да *релаксираме*[1] минималните инцидентните ребро към неактивните върхове. Така Прим се свежда до две вложени обхождания, даващи изключително чисто квадратно решение, което може да се използва към оптимизациите към предните подзадачи.

За кошмар на автора, това решение се държеше изключително добре на подзадачите, предвидени за решения с по-добри сложности, което обяснява големите ограничения. Тъй като човечеството все още не е цивилизация тип II по скалата на Кардашев, и нямаме Сфера на Дайсън около слънцето, нашите компютри са много лимитирани в енергията която консумират и нямаше как да вдигна достатъчно ограниченията, така че да разгранича това решение от по-добрите. Така това решение изкарва подзадачата за $N = 2^B = 2^{18}$, поради което тя е 15 точки, вместо 25.

Постигната сложност: $O(\frac{(2^B)^2}{B})$

Имплементация: `n2_steroids_45p.cpp`

[1] Термин за когато се опитваме подобряваме нещо, особено в графова постановка (например Дейкстра непрекъснато релаксира минималните пътища към върховете).

По-бързо решение

А сега вече към частта от задачата, в която наистина започваме да работим с OR.

Подзадача №7, №8 и №9

Едно от най-често срещаните неща при задачи с побитови операции е да се гледа от най-старши бит към най-младши, защото тези с 1 са винаги $>$ от тези с 0 – $10000…0 > 01111…1$ в двоична както $10000…0 > 09999…9$ в десетична.

Нека си представим как сме пуснали алгоритъма на Крускал по нашата редица. Нека разпредели числата на две групи – тези, които имат 0 на най-горния си бит (т.е. $B-1$-вия) и тези, които имат 1. Тогава, в даден момент, всички елементи с 0 ще бъдат свързани, защото OR-ът им ще има 0 за най-горен бит, а същевременно, няма да има нито едно ребро, свързвашо елемент единица, с някой друг (защото OR-ът с него със сигурност ще има 1 за най-горен бит). Този аргумент важи, защото може да *замразим* Крускал по някое време, през което ще е минал всички ребра между два елемента с нула за най-горен бит, и през нито едно останало.

И какво от това? Ами, имаме напредък – една част от задачата ни за построяване на МПД е да построим МПД за по-малък на брой елементи от началото, след което да продължим с останалите. Това ни дава основанието да търсим рекурсивно решение.

Нека се опитаме да намерим именно рекурсивно решение. Нека да приемем, че рекурсията е такава, че приема редица от числа, и намира цената на минималното покриващо дърво.

Тогава, може да вземем числата с най-горен бит 0, да го изтрием от тях, и да намерим МПД рекурсивно на същите числа, които вече ще са с един бит по-малко. Какво правим за единиците? Ами, ние знаем, че те не са свързани помежду си, като всички 0 са свързани помежду си. Така знаем 2 неща – знаем, колко още ребра ще ни трябват за да досвържем всичко в МПД, както и знаем, че всяко ребро ще включва елемент с най-горен бит 1-ца. Така, всяко останало ребро, което ще добавим, в МПД, ще има най-горен бит 1-ца, която може директно да я предплатим, като добавим бройРебраЗаДобавянеКъмМПД (което при тази идея е $=$ броя единици) $\cdot$ ценатаНаЕдиницатаВЕдноРебро към отговора. Така, добавяме тази стойност към отговора и може да изтрием най-горния бит на единиците и да извикваме рекурсивното свързване за цялата редица наново, в която всяко едно число ще има 1 по-малък бит.

Обаче това има проблем – ние докато правим рекурсията, свързваме върхове, та в монентът в който викаме рекурсията наново за цялата редица, по-долните нива не могат да считат, че елементите не са свързани. Така трябва леко да променим идеята на рекурсията. Една модифицирана версия би била рекурсията да приема редица, която е свързана в някакви компоненти от предишни извиквания, като намира цената за досвързване на редицата в МПД. Също, ще поддържаме DSU, което да пази свързаността на редицата, като след приключване на работата на рекурсията, е свързало всички върхове в една компонента.

Така горната ни идея не се променя много – пак извикваме рекурсия за 0-те, единствено бройРебраЗаДобавянеКъмМПД се променя, като това ще е равно на броя компоненти из 1-ците, които са различни от компонентата на 0-те, след свързването им (нищо не забранява някоя 0 и 1 да са в една и съща компонента, което се случва често!).

Така, вече имаме работещ алгоритъм. Колко пък ли е бърз? Ако редицата е само от 0, то ние ще имаме 2^B-та извиквания, като при всяко ще минаваме през цялата редица за да делим 0 от 1-ци, което

дава квадратна сложност отново.

Заради това може да махнем еднаквите. Тогава, ще се опитаме да оценим сложността на база идеята, че липсват 1-ците в моментите, когато намираме МПД на 0-лите. Заради това, ще дадем номер на всяко едно извикване на рекурсията. При първото извикване, номера ще бъде от B на брой x-а, например за $B = 4$, номера ще е $xxxx$. Тогава, за всяко извикване на нова рекурсия, се добавя цифра 0 към номера на рекурсията, която намира МПД за нулите, като се пише 1 към номера на рекурсията, която решава задачата наново за всички числа. Така, начална рекурсия $xxxx$ ще извика рекурсии с номера $0xxx$ и $1xxx$, а $0xxx$ ще извика рекурсии $00xx$ и $01xx$. Това го правим за следното твърдение – рекурсия с определен номер не съдържа числа, които не са подмаски в частта на номера на рекурсията, която има цифри. Тоест, 1100 не може да бъде измежду числата, които $01xx$ решава, защото 11 не е подмаска на 01. Добре, и? Сега ще оценим колко е сборът на дължините на редиците в най-дънното ниво на рекурсията, т.е. където всички номера нямат x-ове. Това всъщност, заради аргумента с номерацията, е сборът на броят събмаски за всяко число от 0 до $2^B - 1$, защото номерата са именно сортираната пермутация на числата в този интервал. Така, се питаме за

$$\sum_{x=0}^{2^B-1} countSubmasks(x)$$

Колко подмаска има x? Еми, за всяка подмаска y, тя има 2 опции за бит за 1-ците в x, и е навсякъде 0, където x е 0. Така, ако x има b единици и лимита за x е $x < 2^B$, то броят събмаски е 2^b. Така, има точно $\binom{B}{b}$ числа с b единици, което дава ограничението:

$$\sum_{x=0}^{2^B-1} countSubmasks(x) = \sum_{b=0}^{B} \binom{B}{b} \cdot 2^b = \sum_{b=0}^{B} \binom{B}{b} \cdot 1^{B-b} \cdot 2^b = (1+2)^B = 3^B$$

Което следва от Биномната теорема, или иначе казано, разкриване на скоби на $(1+2)^B$.

Така по една бърза оценка изглежда, че сложността е $O(3^B \cdot B)$, защото сборът на дължините на редиците на последното ниво е $\geq$ от сборът на предпоследното и т.н. Обаче, има и по-добра оценка. Не сложихме тези x-ове в номерата на върховете случайно - те са за да покажат, че първите k бита, в които сме сложили цифри, фиксираме някоя събмаска, а x-овете ги заменяме с произволни $B - k$ цифри. Ние ще се целим да свържем предния аргумент за 3^B за листата към останалите върхове в дървото на рекурсията. При тях отново ще имаме сумарно 3^k събмаски за фиксираните им първи k цифри, като за всяко число, което е събмаска в първите си k цифри, има 2^{B-k} начина да запълним позициите на x-овете с 0 и 1. Така, получаваме ограничението, че сборът от дължините на редиците измежду рекурсиите, с номера, с k фиксирани цифри, е $3^k \cdot 2^{B-k}$ – например, за рекурсиите с номера $00x, 01x, 10x, 11x$, знаем, че $00, 01, 10, 11$ имат сумарно 9 подмаски, които ги допълваме с още един бит, за да станат числа с B бита, откъдето излиза, че сборът от дължините на редиците е до $2 \cdot 9 = 18$. Така, финалното ни ограничение е:

$$\sum_{k=0}^{B} 3^k \cdot 2^{B-k} = \sum_{k=0}^{B} 3^B \left(\frac{2}{3}\right)^k$$
$$= 3^B \sum_{k=0}^{B} \left(\frac{2}{3}\right)^k$$
$$= 3^B \cdot \frac{1 - \left(\frac{2}{3}\right)^{B+1}}{1 - \frac{2}{3}}$$
$$< 3^B \cdot \frac{1 - 0}{1 - \frac{3}{3}}$$
$$= 3^{B+1}.$$

Което следва от формулата за сбор на геометрична прогресия.

Така алгоритъма има сложност $O(3^B \alpha(N))$. Само по себе си, решението изкарва 45 точки, поради липсата ни на късмет с бързото квадратно решение.

Една оптимизация е да приложим филтрирането от 5 и 6 подзадача, което вече ни дава 60 точки, което беше оригиналния план за точкуване на решението.

Постигната сложност: $O(3^B \alpha(N))$

Имплементация: `abstract_60p.cpp`

Подзадача №10

За тази подзадача е планирано решение, което е махнало обратния акерман и е направило филтриране. Подхода за махането на акерман е описан при подзадача №12.

Подзадача №11

Може да приложим следната оптимизация допълнително към филтрирането – когато решението достигне момент, в който всички елементи са в 1 компонента, прекъсваме рекурсията. Теоретично доказаната от нас горна граница за сложността на решението остава същата, но имаме основание да вярваме, че всъщност е $O(2.56^B)$, защото на тестовете, които са по Спернер, а именно всички числа с точно k единици, сложността на решението клони към това (а другите тестове не са толкова "смислени", което е същата интуитивна постановка зад Теоремата на Спернер).

Постигната сложност: $O(3^B \alpha(N)$ или $2.56^B \alpha(N))$

Имплементация: `kopchev_70p.cpp`

Подзадача №12

Вместо да пазим глобално DSU, може да забележим, че може да пренаправим рекурсията, така че директно параметрите ѝ да поддържат свързаността на редицата, т.е. да приема вектор от вектори, където един от тези вектори съдържа стойностите в една свързана компонента, а обединението на векторите да дава редицата. След изпълнение на рекурсията за намиране на МПД на 0-те, обединяваме векторите им в един, като впоследствие извикваме процедурата за всички числа с изтрит най-горен бит.

Обединяването на векторите в един е евтино, защото има общо $N - 1$ добавяния на ребра към минималното покриващо дърво по време на изпълненията на рекурсията, поради което ако мърджваме чрез Small-to-Large, $O(N\ log_2 N)$ аргумента за DSU, който по принцип правим, важи.

Постигната сложност: $O(3^B$ или $2.56^B)$

Имплементация: `abstract_emil_77p.cpp`

Пълни решения

Макар и красотата на горната идея (която Ви съветвам да я прочетете, защото е красива), двете ни пълни решения са базирани на други идеи.

Авторово решение

Едно нещо, което може да забележите е, че евристично е добра идея да свързваме малки стойности, просто OR-ът им е логично да е по-малък, защото единиците им са на по-ниски позиции.

Това ми беше в съзнанието като интуиция за това решение. Същевременно, търсех и по-добро обяснение на въпроса ”Защо бих свързал всичко с 0, ако имаше такъв елемент в редицата?” от това, което намерихме през супермаските.

Нека сме изтрили редицата от еднакви стойности. Тогава ще започнем със следното наблюдение.

Твърдение. Бройката на ребра към някой връх със стойност x, които са към върхове с по-малки стойности е $\leq B$

Доказателство. Нека $x > y$. Тогава има бит в който x и y се различават и най-горният такъв има свойството, че x има 1 на този бит, а y има 0. Тогава, ако x е свързан с $> B$ числа в МПД, то ще има y_1 и y_2, които споделят един и същи такъв бит. Нека този бит е на позиция b, тогава задължително $y_1|y_2 < min(x|y_1, x|y_2)$, защото $y_1|y_2$ споделя еднакви битове с $x|y_1$ и $x|y_2$ до съответния бит b, като на позиция b, $y_1|y_2$ има бит 0, защото и y_1, и y_2 имат бит 0, а $x|y_1$, $x|y_2$ имат бит 1, защото x има такъв бит.

Това наблюдение е по-полезно, защото казва, че за всяко x може да сведем ребрата към по-малки върхове, които трябва да вземем в предвид за строене на МПД, до B.

Нека фиксираме кой е съответния първи бит b, в който x ще се различава от y. Тогава, от всички y, които се различават от x, ще вземем това y, за което $x|y$ е минимално – Крускал ни задължава.

Така, ако успеем да намерим тези B на брой y-ка, които описахме, ще сме успяли да намалим броя ребра от N^2 на NB, след което да направим (NB) Крускал, който използвахме във втора подзадача, за да построим МПД.

Останалото може да направим с Динамичното програмиране, чиито вид може да го срещнете като SOS DP. Динамичното ще бъде със стейт $dp[value][bit]$, което намира това y в редицата което дава минимално $y|value$ и е под ограничението, че в най-горните си bit бита е равно на $value$, а в останалото е произволно. За да го пресметнем, то може да разгледаме какъв би бил $bit + 1$-вия бит (отгоре-надолу) на y, съответно най-доброто y със същия бит като $value$ ще се съдържа в $dp[value][bit + 1]$, а това с различен бит ще се съдържа в $dp[value\text{^}(1 << bit)][bit + 1]$, защото на $y|value$ ще има OR равен на 1 в bit (различни са по битове, следователно OR-ът там е 1), откъдето остава единствено да минимизираме OR-а на y в останалите битове на $valye$, което $dp[value\text{^}(1 << bit)][bit + 1]$ прави, защото $value\text{^}(1 << bit)$ има еднакви битове като $value$ след bit-та позиция (отгоре надолу).

Постигната сложност: $O(2^B \cdot B + NB\alpha(N))$

Имплементация: `abstract_100p.cpp`

Решение на Даниел Койнов

Нека се опитаме да приложим Крускал директно. Тогава ще се целим да свържем първо всички върхове, които имат ребро с тегло 0, после ребро с тегло 1 и т.н. Кои ребра биха тегло $= x$? Ако е между два върха със стойности a и b, задължително a и b трябва да са подмаски на x, иначе $a|b$ ще има единица на място, където x няма. За да може две подмаски a и b на x да не дават $a|b = x$, то трябва да има друго число y, което е подмаска на x и a и b са подмаски на y. Например 1001 и 1010 не дават OR равен на 1111, защото и двете са подмаски на 1011, което е истинския им OR.

Така, нека за $O(3^B)$ намерим кои числа са събмаски на x за всяко x, като сме махнали еднаквите. Тогава, по горната идея, може да обходим $x = 0, 1, \cdots, 2^B - 1$ и за да получим бройката на компонентите, които Крускал би свързал с ребро с тегло x, може свържем двойки всички числа, които са събмаски на едно и също $y < x$, и да преброим колко са различните компоненти сред събмаските на x. По-точно, алгоритъма ни се изпълнява от следното

- Разглеждаме $x = 0, 1, \cdots$;
- Преброяваме броят на компонентите сред елементите, чрез DSU, които са подмаски на x, и ако бройката им е cnt, добавяме $(cnt - 1) \cdot x$ към отговора;
- Свързваме всички елементи, които са подмаски на x да са в една компонента, чрез DSU.

За да оптимизираме решението до $O(2^B \cdot B \cdot \alpha(N))$, може да забележим, че ако x има $k > 0$ единици, то x ще има до B събмаски, които са с $k - 1$ бита, откъдето сред числата, които са събмаски ще има до $B + 1$ различни брой свързани компоненти (по 1 за събмаска и 1 за всички елементи $= x$). Това може да го ползваме, като намерим по 1 представител от събмаска (което може да си го пазим по подобие на динамично), и впоследствие наново проверим броя различни компоненти, след което да извлечем представител на свързаната компонента на числото x.

Постигната сложност: $O(2^B \cdot B \cdot \alpha(N))$

Имплементация: `abstractAlternative_100p.cpp`

Послепис

От теоретична гледна точка, най-доброто решение е това, използващо Dense Prim, защото то е най-бързото полиномиално решение измежду дадените. Решенията за повече точки използват фактът, че 2^B е малко, може което расте експоненциално на база дължината на входа. В крайна сметка, ние ходим на олимпиади, за да решаваме задачи не заради теоритичния им принос, а заради това, че са красиви, което им позволява да са абстрактни.

Автор: Борис Михов