АНАЛИЗ НА РЕШЕНИЕТО НА ЗАДАЧА
ПЕЧАЛБА

Математическа формулировка:

Граф наричаме двойката (множество от върхове, множество от връзки/ребра между върховете). Дърво наричаме свързан граф без цикли, т.е. свързан граф с единствен път между всяка двойка върхове. Така задачата можем да преформулираме като търсене на оптимален път в дърво, максимизиращ сумата от печалби за маршрути, и двата края на които лежат на избрания път.

Основно наблюдение:

Максимална печалба може да бъде получена на път от едно листо до друго листо, тъй като всеки друг път можем да продължим до произволни листа, като при това не намаляваме печалбата, тъй като запазваме вече съществуващите на пътя печалби, а евентуално добавените нови печалби са неотрицателни.

Решение за 20% от точките, O(N^2 * (N + M)):

За всяка двойка листа в дървото изчисляваме печалбата на пътя между фиксираната двойка листа за едно преминаване по пътя.

Решение за 40% от точките, O(N * (N + M):

Да разгледаме двете поддървета A и B, на които се разделя T от първото ребро в списъка с ребра. Ще разглеждаме A и B като дървета с корени (като корените са именно тези върхове, които в са свързани в T с първото ребро от списъка). Тъй като по условие знаем, че реброто между корените на A и B присъства в оптимален път, то този път се състои от част, лежаща в A и част, лежаща в B. За произволни върхове **a** от *A* и **b** от B ще смятаме печалбата profit(a,b) на пътя между a и b, използвайки метода на динамичното програмиране:

profit(a,b) = max{ profit(a, b_prev) + bonuses(b -> a...b_prev),
profit(a_prev, b) + bonuses(a -> a_prev...b) }, при a≠b;
profit(a,b) = 0, при a=b;

където bonuses(x -> first...last) е сумата на бонусите от върха x към върховете на пътя от first до last. Отговорът на задачата са листата u и v, за които profit(u, v) има максимална стойност.

Описаното решение изисква O(N^2) памет за междинните резултати profit(a,b). Времето за изчисляване на поредния profit(a,b) включва проверяване на печалбите от маршрути, водещи само до върховете a и b. Но тъй като всички маршрути, водещи до връх от едното поддърво A или B са не повече от M, то амортизирано за всеки от O(n)

фиксирани върхове в A, разглеждането на печалбите, водещи до всички върхове в B, е O(M).

Друго решение със същата сложност би било за всеки фиксиран стартов връх/листо, с едно обхождане на дървото да бъдат последователно намирани печалбите до всеки друг връх

profit(a,b) = max{ profit(a, b_prev) + bonuses(b -> a...b_prev) }, при a≠b;

profit(a,b) = 0, при a=b;

Решение за 70% от точките, O(N + M * logN):

Да разгледаме същите поддървета A и B от решението за 40% от точките. Този път ще подходим несиметрично относно A и B. Над дървото A ще определим структура от данни, в която да можем бързо и в произволен ред да:

- обновяваме сумата от печалби до произволно поддърво на A (когато срещаме маршрут, водещ до корена на съответното поддърво);
- извличаме лист, за който се достига максимална печалба на пътя от корена и извличане на самата печалба (след всяко прибавяне на печалба от маршрут, ще проверяваме дали пътят от началото на новодобавения маршрут, стигащ до някое листо в A, не е най-изгодният до момента);

Изброените операции можем в произволен ред да извършваме върху интервално дърво[1] IT -- пълно двоично дърво, с листа, съдържащи отделни стойности. В IT с логаритмична от броя на листата сложност може да бъде реализирана произволна адитивна (чието действие се натрупва) операция, например прибавяне на стойност към всички листа на произволно поддърво на интервалното дърво. Също за логаритмично време от IT можем да извличаме листото с максимална стойност, а самата стойност – за константно. Паметта, необходима за IT, зависи линейно от броя на листата.

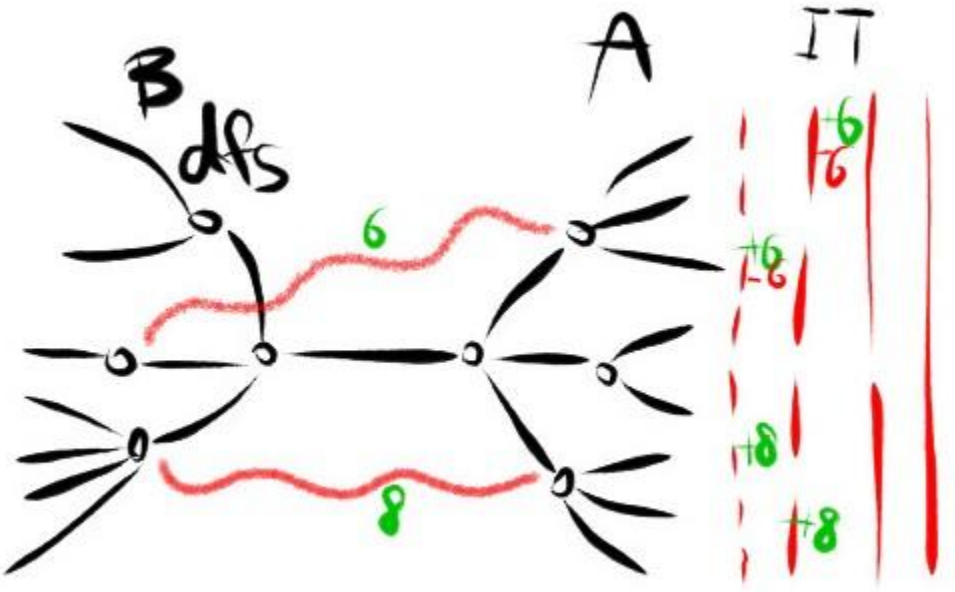
Да разгледаме IT с поне толкова листа, колкото в A, и на всяко листо от A да съпоставим листо в IT, така че на всяко поддърво на A да съответства последователен интервал от листа в IT. Такова съответствие може да бъде постигнато чрез обхождане на A в дълбочина, започвайки от корена и давайки на листата последователни индекси, под които ще бъдат срещани в IT. За всеки връх v в A ще запомним интервала interval(v) в IT, в който лежат всички листа от съответното поддърво.

Щом за 70% от тестовете оптимален път преминава през реброто, разделящо поддърветата A и B, значи печалбите от маршрути в един оптимален път през това ребро трябва да бъдат сума от печалбите на маршрути от A до A, от B до B и от B до A (в общия случай).

Да пресметнем първо печалбите за пътищата от корена на A до всяко листо на A. За всеки маршрут m(u, v, profit) от върха u до върха v с печалба profit, такива че и двата лежат в A и u се намира на пътя от корена на A до v, нека добавим profit към interval(v) в IT. Т.е. печалбата от корена на A до всяко листо в interval(v) ще бъде увеличена с profit, тъй като всички пътища от корена до лист с индекс в interval(v) преминават през v, като в същото време никой друг път до листо от A не може да получи тази печалба.

Всеки маршрут m(u, v, profit) носи печалба на всички пътища от a до b, такива че, a лежи в поддървото на B с корен u, a b лежи в поддървото на A с корен v (в случай, че v лежи в A) или в цялото дърво A (в случай, че пътя от u преминава през v до корена на A). Последователно за всеки връх в B ще допуснем, че е начало на път, водещ до листо в A и максимизиращ печалбите от включените в него маршрути. Ще обходим B в дълбочина, поддържайки инвариантно свойството, че намирайки се във връх u в B, интервалното дърво IT ще поддържа печалбите до всяко листо, получавани по пътя от u (вместо от корена на A) до съответното листо. Така, спускайки се към листата в B, ще добавяме в IT печалбите от маршрутите, започващи от текущия връх, след което ще се обръщаме към IT с надеждата, че от текущия връх постигаме по-голяма печалба до някое листо в A. Напускайки текущия връх, изкачвайки се към корена на B, изваждаме добавените печалби, които е можело да бъдат получени само при продължение на пътя през напуснатия връх.

Така, ако оптимален път преминава от листото u в B до листото v в в A (а такива листа със сигурност съществуват по „основното наблюдение“ от по-горе), то той е бил разгледан, тъй като всички върхове u от B са обходени в дълбочина, а от изпълненото инвариантно свойство за IT следва, че листото v е максимизирало пътя от u.



Преди добавяне на печалба 8 към съответните листа на IT, е нужно да изтрием захвърлената печалба 6

Решение за 100% от точките, O((N + M * logN) * logN):

За останалата част от тестовете ще комбинираме решението за 70% (което ще наречем *владей*/*conquer*) със схемата разделяй и владей. Ще изберем връх u и подмножество на децата му ks, така че броят на върховете в така образувания подграф е приблизително равен на върховете в подграфа, образуван от същия връх и остатъка от децата. За да сведем задачата до *conquer,* трябва да имаме разделящо ребро – нека си го направим като заместим u с реброто (u1, u2), където u1 има деца ks, а u2 има деца kids(u)–

ks (останалите деца на u). След като намерим оптимален път, преминаващ през (u1, u2) с *conquer*, премахваме това ребро и получаваме две несвързани дървета A и B, същите използвани от *conquer*. Остава да решим задачата рекурсивно за всяко от тях поотделно, тъй като ако оптимален път не преминава през (u1, u2), то значи остава да лежи изцяло в едно от двете поддървета.

Да разгледаме намирането на разделящ връх. Както споменахме по-горе, е нужно разцепването да е равномерно. Ще докажем, че показания по-долу алгоритъм разделя в най-лошия случай на поддървета с количества върхове в отношение 1:2 (1/3 от дървото в едното поддърво, 2/3 от дървото в другото).

Да изберем произволен вътрешен връх (не листо) за корен и да намерим броя върхове/теглото на всяко поддърво. Ако е изпълнено че теглото на поддървото образувано от най-тежкото дете е поне половината от цялото дърво, ще се преместим в някое от децата му. При преместването на корена в някое дете, обновяваме теглото на бившия корен, който сега е дете на новия корен (бившо дете), както и на новия корен, след което отновнотърсим дете със същото свойство.

В момента, в който не се преместим, образуваме множеството от деца ks по следния начин: добавяме най-голямото дете в ks. Докато теглото на ks (сумата от теглата на всички деца в него) е извън интервала [1/3, 2/3] от теглото на цялото дърво – добавяме произволно дете в него. Ако най-голямото дете има тегло < 1/3, то значи и всички останали имат също тегло < 1/3, а значи и добавянето на произволно дете в ks не може да „прескочи“ интервала [1/3, 2/3]. Ако най-голямото дете има тегло >= 1/3, то също е и < 1/2, иначе щяхме да сме се прехвърлили в него в по-горното търсене на разделящ връх.

Сега остава да направим анализ на сложността на решението. Намирането на разделящ връх отнема O(N) за началното претегляне. Последващото търсене може да се разглежда като ново обхождане в дълбочина, което не се връща назад. Следователно тази първа част от разделянето е O(N). Разцепването на върха в отсечка и последвалото премахване на отсечка е също O(N) (за да улесним решението старият граф бива копиран в 2 нови графа, вместо да бъде използван директно).

Решението *conquer* има сложност O(N + MlogN). При разделянето броят на бонусите в двете нови дървета е не по-голям от броя бонуси в старото дърво, понеже бонусите само могат да бъдат премахнати (от едното към другото поддърво). Следователно, при разделянето на задачата (N, M), двете нови задачи (N1, M1) и (N2, M2) са такива че N1 + N2 = N+1, M1 + M2 <= M. Решението им има комбинирана сложност O(N1 + M1 * logN1 + N2 + M2 * logN2) <= O((N1 + N2) + (M1 + M2) * logN) <= O(N + MlogN). Това означава, че за всяко ниво на разделяне добавяме по O(N + MlogN) към сложността на решението. Броя нива е log_3 N (ако разделяме винаги 1:2), което е асимпотично еквивалентно на O(logN), и, следователно, крайната сложност става

O((N + MlogN) * logN).

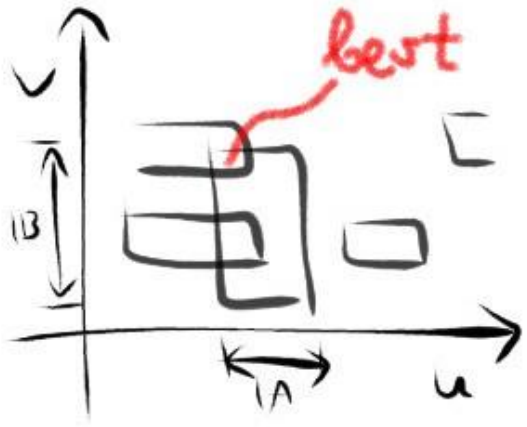
Решение на Михаил Пядеркин за 100% от точките, O(N + M * logN):

Това решение беше дадено след състезанието от един от ръководителите на отбора на Руската Федерация – Михаил Пядеркин.

Можем да подходим и по различен начин. Да номерираме всички листа на T в ред на обхождане в дълбочина, започвайки от произволен връх. Така всеки маршрут свързва две поддърветата в T, носейки печалба която и двойка листа от едното и от другото поддърво да изберем. Листата на всяко поддърво са номерирани последователно, т.е. образуват интервали IA и IB (I[a; b] включва листата от a до b, ако a<=b, а в противен случай включва листата [a; last_leaf_idx] и [first_leaf, b]). Така всеки маршрут носи бонус на всяка двойка листа (u,v), лежащи в интервалите IA и IB съответно.

С целочислената точка (u,v) в равнина можем да изобразим път от листото u до листото v в T. Така интервалите IA и IB на фиксиран маршрут носят печалба на всички точки/пътища, лежащи в правоъгълника, образуван от IA и IB. Остава да намерим такава целочислена точка, която е покривана от правоъгълници/маршрути с максимална сума на печалбите. Тази задача може да бъде решена чрез метода на метящата права (sweep line [4]), движейки се в ред на увеличаване на индексите u, поддържайки в интервално дърво сумите на печалбите за всички точки/пътища с първо листо u и свободно второ листо.

За разлика от предните, това решение може да бъде написано в изключително сбит вид (<100 реда код).



Послеслов:

Тази задача възникна по естествен начин при решаването на проблем в областта на биоинформатиката, свързан с възстановяването на големи геномни последователности, използвайки къси геномни „парченца“/рийдове (които неизбежно се получават, поради ограничението, че технически до момента не е известен метод за „четене“/секвениране на геноми с дължини, по-големи от няколко хиляди „букви“/нуклеотиди, а геномите на живите организми включват милиони и дори милиарди нуклеотиди).

В термините на оригиналната задача [5], „градовете“ представляваха парчета ДНК, а „преките пътища“ между тях – възможно за удължаването на единия рийд с другия. Не всички удължавания съответстват на реалната последователност, която е била накъсана преди рийдовете да бъдат прочетени, и точно в това се заключава задачата – да бъде намерен правилен път, представляващ дълъг геномен участък. За целта бива използвана различна допълнителна информация относно взаимното положение на различни участъци – например, т.н. двойни рийдове, които представляват два обикновени рийда, но с допълнителната информация, че се намират на горе-долу фиксирано разстояние. Нашата идея беше да използваме специфичен „дефект“, наречен „химерни рийдове“ (това са рийдове, които всъщност не се срещат в генома, а са парчета включващи две различни части от генома), тъй като двете части на химерния рийд най-често са близко разположени в амплифицирания геном. Така, удължаването на път можем да направим в това направление, към което има най-много „маршруги“/химерни рийдове. Сложности представляват цикличността на този геномен граф, но поради съществуването на ефективни точни алгоритми в ациклични графи, разрешаването на циклите може да бъде адекватно решение. Идеята да намираме най-вероятни продължения в посоката след дадена последователност бива логично разширена с намиране на едновременно продължение в двете посоки, така че комбинацията от продължаващите пътища да включва колкото е възможно повече химерни рийдове (може би вече разпознавате именно подзадачата за 70% от точките).

Вместо заключение искам да направя реклама на Rosalind[6] – това е съвсем нов, но достатъчно наситен безплатен сайт с биоинформатични задачи, в който освен самите задачи, ценност представлява и достъпния за информатици начин на излагане на молекулярните/биологичните основи. Появяват се все повече изчислителни задачи дори в доскоро „мътни“ области като биологията.

Препратки:

[1] http://community.topcoder.com/tc?module=Static&d1=tutorials&d2=binaryIndexedTrees – увод в двоичните индексни дървета

[2] http://en.wikipedia.org/wiki/Divide_and_conquer_algorithm – подход „Разделяй и владей“

[3] http://en.wikipedia.org/wiki/Amortized_analysis – амортизиран анализ

[4] http://community.topcoder.com/tc?module=Static&d1=tutorials&d2=lineSweep – метяща права

[5] http://pesho.info/wp-content/uploads/chimeras-final.pdf – презентация относно първоначалната биоинформатична задача от лятото на 2013г.

[6] http://rosalind.info – online judge със задачи по биоинформатика

Автори: Петър Иванов, Искрен Чернев