	На пълното решение	На частичните решения
Тагове	Динамично програмиране Сортировки	Пълно изчерпване Обхождане на графи Разширения на графи

Анализ

Подзадача №1

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

Подзадача №2

За решение на задачата е предвидено пълно изчерване. Чрез DFS обхождане на графа може да се разгледа всеки възможен път, започващ от 1. Броя на тези пътища, макар на пръв поглед да изглежда, че клони към $M!$, всъщност клони към 2^M. Причината за това е, че всяко подмножество от ребра би могло да има единствен начин на обхождане – по сортирания ред на теглата им. Решението преминава и втората подзадача, тъй като в нея броя валидни пътища от 1 са до N на брой.

Постигната сложност: $O(N + 2^M)$

Имплементация: `walk_17p.cpp`

Подзадача №3

Графът зададен от подзадачата е дърво. От там следва, че между всяка двойка върхове има точно 1 път. Така може да направим DFS от връх №1 и да проверим дали единствения път от 1 до всеки друг връх изпълнява условието $c_{next} > c_{current}$.

Постигната сложност: $O(N + M)$

Имплементация: `walk_7p.cpp`

Подзадача №4

Задачата за най-дълъг път в граф е от клас NP-complete, така че трябва да разберем какво е специалното в текущия граф, че автора да е успял да реши задачата. Може да забележим, че ако разширим графа с параметър `[последно ребро, участвало в пътя до текущия връх | връх]` и построим ориентирани ребра, означаващи връзките между върховете в него, то графа ще се окаже DAG. Така в графа ще има $O(NM)$ върха и (M^2) ребра. Вместо да се търси най-дългия път, завършващ на даден връх, може да се намира най-дългия път започващ от него и завършващ на 1. Решението на тази подзадача може да се довърши със стандартното динамично за най-дълъг път в DAG.

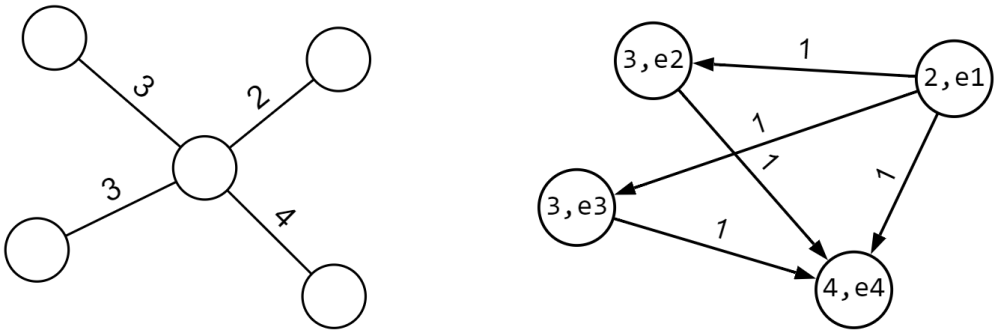
Постигната сложност: $O(NM + M^2)$

Имплементация: `walk_46p.cpp`

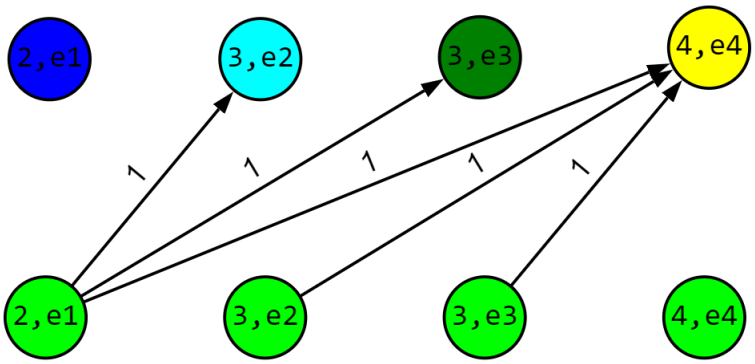
Подзадача №$5, 6$

Да помислим какво можем да оптимизираме в предната подзадача. Дори, ако направим разширение, в което пазим последния съсед вместо индекса на последното ребро сложността пак ще е същата заради върховете с много съседи. Следователно ни трябва по-умна конструкция, чрез която да не разглеждаме всяка двойка (`предишен съсед, следващ съсед`) директно.

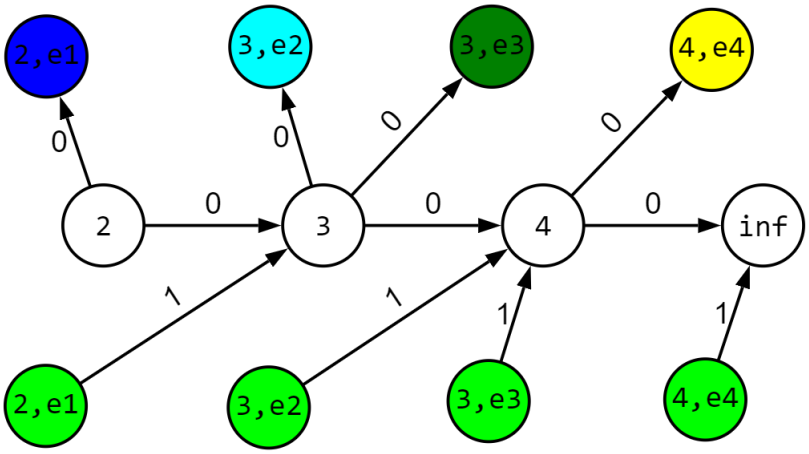
Понеже главното движение в задачата е по самите ребра е удобно да направим ребров граф – на всяко ребро съпоставяме връх, а оригиналните върхове се трият. Сега остава да навържем ребровия граф подходящо. Една директна, но квадратна конструкция би била просто да свържем всяко ребро с по-големите такива от същия оригинален връх:



Първото число на всеки връх е теглото му, второто е индекса в оригиналния граф. Съществува един проблем и той е, че така може да обикаляме по съседите на оригинален връх и да ги броим като един път (път $e1 \rightarrow e2 \rightarrow e4$ в тази конструкция не е валиден в оригиналния граф). Затова ще направим разширение на ребровия граф. На всяко ребро от оригиналния граф ще съпоставим два върха – съответно за двете посоки на движение по него:



Цвета на връх е посоката на реброто, на което съответства (ребра влизащи в един връх имат еднакви цветове). Остава само да оптизираме начина на строене, което може да бъде лесно направено с помощта на няколко изкуствени върха по следния начин:



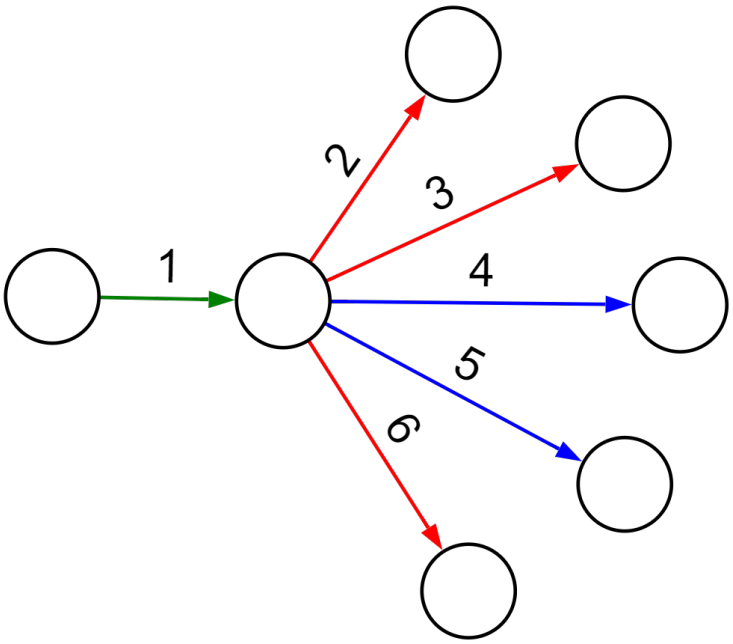
За всяка група от ребра с еднакви тегла правим връх, който сочи към тях. Отделно имаме един изкуствен връх, който можем да считаме за отговарящ на ребра с безкрайно тегло (той е удобен, когато търсим най-дългия път до оригиналния връх). Навързваме изкуствените върхове, а "влизащите ребра (тези с еднакъв цвят)" към групата отговаряща на по-голямо тегло. Така си гарантираме, че всяко следващо ребро ще е с по-голямото тегло. Решението е със същата сложност като това за 100 точки, но поради голямата константа от изкуствените върхове и работата с вектори, то се държи значително по-бавно. Всъщност, пълното решение имплицитно върши същото като текущото, но е много по-ефективно.

Постигната сложност: $O(N + Mlog_2M*)$

Имплементация: `walk_80p.cpp`

Подзадача №$7, 8$

Подзадачата е с подобна идея на горната, но е оптимизирана. Ще разкажа решението отначало, така че може да считате, че анализа продължава от подзадача №4. Нека разбием всяко неориентирано ребро на 2 ориентирани. Така, нека за всяко ребро да изчислим дължината на най-дългия път, който започва от него, включващ го и завършващ на 1.



Walk

Нека да разгледаме как да изчислим дължината на зеленото ребро (№1). Нека червените ребра (№2, 3, 6) да са с по-малки стойности, а сините (№4, 5) – с по-големи или равни. Тогава:

$$dp_1 = max_{edge \in red}(dp_{edge}) + 1$$

Защото ние може да изберем произволно червено ребро, с което да продължим, добавяйки 1 към дължината на пътя му. Червените ребра трябва да са с по-малки стойности, защото ние в решението ни намираме търсените пътища в задачата в обратния ред на този от условието (ходим от u към 1 вместо от 1 към u). Следвайки тази идея, ние може да сортираме ребрата, да ги обхождаме от най-малко към най-голямо и да изчисляваме dp-тата им, осигурявайки си, че ще сме изчислили dp-тата на всички червени ребра за текущото. Този алгоритъм ще е със сложност $O(M^2)$, тъй като за всяко ребро може да обходим неограничен брой други.

Нека оптимизираме идеята. Вместо всеки път да обхождаме ребрата на върха, от който продължаваме, ние може да поддържаме динамично и за всеки връх, което да е максималния път, извиращ от някое от неговите червени ребра. Първоначално за всеки връх различен от 1 дължината на такъв път е $-\infty$, а за 1 дължината е 0. Постепенно изчислявайки динамичните на ребрата, ние ще актуализираме стойностите на динамичните на върховете. В подзадача №5 може да направим актуализацията директно на момента, но в подзадача №6 трябва да сме по-внимателни. В последната подзадача ще е грешно директно да прилагаме промяната, защото може текущото ребро да се окаже червено за някое следващо в сортирания ред, което има същото тегло като текущото – представете си граф, в който всички ребра имат едно и също тегло и разгледайте как ще действа директната промяна. Вместо това, ние може да разбием ребрата на групички от еднакви тегла и с за всяка една групичка първо да изчислим стойностите на динамичното за ребрата в нея, след което да приложим промените към динамичните на върховете. Така решението ни е вярно и изкарва 100 точки.

Постигната сложност: $O(N + Mlog_2\ M)$

Имплементация: `walk_100p.cpp`

Идея и тестове: Александър Гатев

Решение и анализ: Борис Михов