Задача Магия - НОИ 2 2022 А3

1 Основни наблюдения

1. използването на магия винаги подобрява нещата, затова трябва да го правим възможно най-много;
2. Магия, която не сме изхабили е само пропиляване на възможност за подобряване на резултата;
3. Дори и да имаме повече от $N \times 30$ магия в себе си, това няма значение, понеже и да искаме не можем да изхабим повче;
4. Въпреки, че можем, реално нямаме изгода да пазим магията за по-късно, понеже тя винаги прави еднакво подобрение;
5. Стойността на маршрут е [изминатото разстояние] - [използваната магия];

$O((N^2 \times C + M \times N \times C) \times log(N^2 \times C))$ решение

Тук с C бележим максималната дължина на ребро в графа (в нашия случай 30). Използвайки почти тривиалните наблюдения от миналата секция, можем да направим следното решение. Ще направим граф, където всеки връх ще е някакъв стейт, по-конкретно [на кой връх сме][колко магия имаме]. Това разбира се е с уговорката, че за втората компонента няма да пазим стойности по-големи от $N \times C$, понеже няма смисъл. Един стейт ще пази колко е разстоянието от началното състояние (връх 1 и магия 0 - [1][0]) до текущия стейт. Можем да пресмятаме тези разстояния с Dijkstra. Така ако сме на състояние $[u][m]$ и имаме ребро (u, v, c, d), то можем да се опитаме да подобрим $d[v][min\{max\{0, m - c\} + d, N \times C\}]$ с

$d[u][m] + max\{0, c - m\}$ (Тук с $d[x][y]$ бележим минималното разстояние до връх x с магия y). Изразите с max се получават, понеже се опитваме да използваме цялата си магия m за да редуцираме разстоянието c и в зависимост от това дали имаме твърде много или твърде малко се получават тези сметки. Разбира се към новата си магия прибавяме и d, защото след като минем по пътя придобиваме тази магия. Подробна имплементация има във файла $magic1.cpp$.

$O(N \times M)$ решение

Нека помечтаем малко. Защо не може когато минем някое ребро и вземем магия d от маршрута стойността на разходката до този връх директно да извадим тази магия d, понеже ние планираме да я ползваме в бъдеще. Проблемът е, че ако например сме взели тази магия след отиване при последния връх, ние няма да можем да я използваме. Така се вижда проблемът с това решение, ние не можем да знаем колко от магията, която сме взели на всеки ход ще използваме. Всъщност можем! Нека си представим, че имаме магически(много забавно) функция $leftDist(x)$, която пресмята минималното разстояние от връх x до връх n като започваме с 0 магия, но използваме събрана по пътя такава оптимално. Тогава отговорът на задачата е $leftDist(1)$.

Нека да помислим как да изчисляваме тази функция. В общия случай искаме да пресметнем $leftDist(x)$. Ясно е, че трябва да започнем да се придвижваме към връх n, използвайки ребрата на графа. Нека имаме ребро (x, y, c, d). Можем да използваме това ребро като начало на нашия маршрут. Стойността на най-добрият маршрут, започващ с това ребро е точно $c + max\{leftDist(y) - d, 0\}$, защото е ясно, че трябва да изминем без никаква магия разстоянието c и вече да използваме колкото се може от събраната магия d, за да намалим пътя в бъдеще. Тук наблюдаваме нещо, което доста често се прави при такива задачи - правим някакви ходове, които оптимизират някаква стойност скрито, тоест оптимизацията се появява в неявен вид. Един вид, предварително(понякога и "следварително") прилагаме някаква оптимизация, която знаем как ще се отрази на резултата. Ясно става вече, че най-добрата опция за $leftDist(x)$ е минималната намерена стойност от всички ребра от вида (x, y, c, d) (тоест всички инцидентни на нашия връх x). Сега остана само да видим как се

пресмятат тези стойности възможно най-оптимално.

Тривиално или не, тези стойности могат да се пресметнат чрез алгоритъма на Bellman–Ford. Оставяйки строгото доказателство за коректност като бъдещо занимание на авторчето на това писание, ще преминем директно към описание на идеята.

В най-общ смисъл (*magic2.cpp* или авторския код *magic_bellman_ford.cpp* на задачата за повече детайли) ще обърнем посоката на ребрата на графа и ще запазим другите им параметри и ще търсим минимално "разстояние"от връх n до връх 1, използвайки формулката от горе. По конкретно стойност $leftDist(y)$ ще се опитваме да подобряваме със стойност $c + max\{leftDist(x) - d, 0\}$ за всяко ребро (x, y, c, d) от "обърнатия"граф. И така използвайки тази update функция ще направим един Bellman–Ford и стойността $leftDist(1)$ ще бъде отговор на задачата.

За най-късите пътища

Dijkstra в това решение не може да се използва, понеже начина, по-който разстоянията работят е по-особен, а за Dijkstra се иска нещата да бъдат много прости и предвидливи. По-конкретно, може да се окаже така, че след като сме преминали от връх x до връх y реално сме загубили време ($dist(x) > dist(y)$), един вид сме използвали отрицателно ребро. Dijkstra не работи добре при ситуации като тази. В такива случаи, се използва алгоритъмът на Bellman–Ford.

За протокола, задачата може да се направи и с нещо като Dijkstra. Тъй като аз пиша грешно Dijkstra, тоест не проверявам дали един връх вече е бил използван, за да оптимизира съседите си, то така написан алгоритъма би работил правилно, но няма да е разбира се толкова бърз. По-конреткно, той може да бъде бавен колкото Bellman–Ford в най-лошия случай, но средностатистически е по-бърз. Разбира се, по-умни хора са забелязали тази идея и са разработили още по-бърз алгоритъм наречен SPFA(Shortest Path Faster Algorithm).