ДАА Упражнение 9 в 10 (Дизайн на алгоритми върху графи)

Ако не е посочено нищо е графът в задачата, то приемаме че е неориентиран и нетегловен. Граф, който е ориентиран и ацикличен ще наричаме DAG (от англ. directed acyclic graph). със n ще означаваме броят на върховете в граф и с m броят на ребрата. Има няколко начина за представяне на граф. Ние ще разглеждаме следните два:

- чрез матрица на съседство, т.е. имаме квадратна матрица $n \times n$ и в клетка i,j носи информация дали има ребро от връх i до връх j и/или каква е тежестта на това ребро.
- чрез списъци на съседство, т.е. всеки връх има списък кои върхове има ребро. Нямаме никаква информация за това как са подредени върховете в списъка. Ще означаваме с $adj[x]$ списъка на съседство на връх x. Ако не е посочено какво е представянето на графа ще приемаме, че е чрез списъци на съседство.

За линеен алгоритъм ще приемаме, че е със сложност $\Theta(m+n)$.

Забележка: Задачите в това упражнение нямат формални решения с псевдокод, а само е описана идеята на решението. На контролно/изпит това най-вероятно няма да е достатъчно!

зад.1/ Даден е ~~нетегловен~~ неориентиран граф $G=(V,E)$ и два върха $u \in V$ и $v \in V$ от графа. Да се провери дали връх v е достижим от връх u.

Решение: Задачата може да се реши чрез BFS или DFS чрез пускане на алгоритъма от връх v и проверяване дали в обхождането сме достигнали връх u. В лекционните записки това са алгоритмите BFS-VISIT и DFS-VISIT.

зад.2/ Даден е граф и връх u. Да се намери най-краткият път от u до всички останали върхове на графа. Ако не съществува път от u до някой връх в графа, то считаме че разстоянието е $+\infty$.

Решение: Използвайки BFS можем да намерим най-късият път от връх u до всички останали. Това става чрез модификация на кода, така че всеки път като добавяме връх в опашката, разстоянието до него е равно на разстоянието от u до предшестващия връх $+1$. Отначало $dist(u,u) = 0$

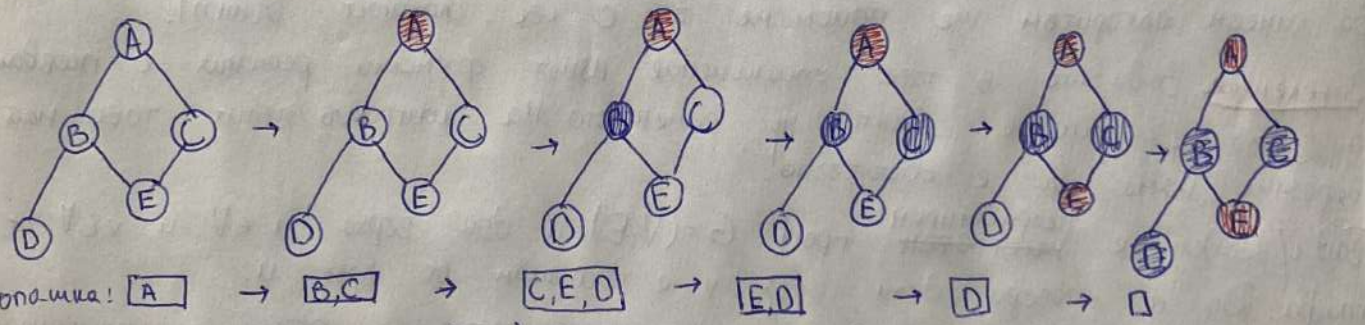
Ако след 1 ~~път~~ пускане на модифицираният код на BFS-VISIT има непосетени върхове, то разстоянието до тях е $+\infty$.

зад.3/ Даден е граф $G=(V,E)$. Да се намери броят на свързаните компоненти на графа.

Решение: Чрез слагане на брояч в кода на BFS/DFS, който да брои колко пъти пускаме BFS-VISIT / DFS-VISIT.

зад.4/ Даден е граф $G=(V,E)$. Да се провери дали графът е двуоцветим (т.е. да няма съседни върхове с еднакви цветове)

Решение: Задачата може да се реши чрез модификация на BFS. Първоначално оцветяваме стартовият връх в червено и добавяме всичките му съседи. След това като извадим връх от опашката и проверяваме дали всичките му съседи са оцветени в еднакъв цвят (или са неоцветени). Ако това не е изпълнено връщаме False. Ако е изпълнено, оцветяваме върха в противоположният цвят на съседите му и добавяме ~~неоцветените~~ посетените върхове в опашката. Ако сме оцветили всички върхове, то връщаме True.



опашка: [A] → [B,C] → [C,E,D] → [E,D] → [D] → []

зад.5/ Даден е граф $G=(V,E)$. Да се провери дали съществува цикъл в графа.

Решение: Когато DFS минава през списъка на съседство на даден връх, може върховете да са оцветени в бяло, сиво или черно. Ако по време на изпълнението на алгоритъма "видим" се някой от съседите на текущият връх е оцветен в сиво, то тогава съществува цикъл в графа. Аргументацията защо това е така има в лекционните записки.

Алгоритмите за най-къс път, изучавани в курса по DAA са Dijustra, Bellman-Ford и Floyd-Warshall. Първият работи тогава и само тогава когато всички тегла на ребрата са неотрицателни. Вторият работи, когато има ребра с отрицателно тегло, но не и когато графът съдържа отрицателни цикли. Третият намира най-къс път от всеки до всеки връх и работи при наличие на отрицателни ребра и цикли.

зад.6/ Даден е ориентиран граф $G=(V,E)$. Да се предложи алгоритъм с оптимална сложност по време и памет, който отговаря на въпроса "Съществува ли цикъл с отрицателно тегло в графа G?"

Решение: Можем да използваме алгоритъма на Bellman-Ford, който е със сложност $\Theta(n(n+m))$. Понеже прост път в граф може да съдържа най-много $n-1$ ребра и алгоритъма на Bellman-Ford извършва $n-1$ на брой итерации, като на всяка итерация (потенциално) подобрява най-късите пътища, то ако направим още една итерация и има подобрение в някой от най-късите пътища, то със сигурност графа съдържа отрицателен цикъл.

Заб: Защо отрицателните цикли са "лоши"? Ако имате чейндж бюро или множество от чейндж бюра и моделирате обмена на валути като граф, то ако в графа съществува цикъл с отрицателно тегло, то можете да правите тези размени на валути и да правите "безплатни пари". Същото е приложимо и при спортните залози. За повече информация потърсете "arbitrage betting".

зад.7/ На изпит има n на брой студенти. Знаем кои двойки студенти могат да си подсказват безопасно/опасно. За подадени студенти A и B и число $k \in \mathbb{N}$ да се провери дали A може да подскаже на B с по-малко от k опасни подсказвания.

Решение: Моделираме задачата като граф. Всеки от студентите ще е връх на графа и ще добавим ребро (u,v) с тегло 0, ако студент u може да подскаже на студент v безопасно, или с тегло 1, ако студент u може да подскаже на студент v опасно. След това можем да използваме алгоритъма на Dijkstra от връх A и ако разстоянието до B е $< k$ да върнем истина. Ако разстоянието е $\geq k$ то връщаме false.

зад.8/ Даден е речник от думи на неизвестна за нас азбука. Да се провери дали по подадените думи в речника няма противоречие (т.е. дали думите ~~могат да се~~ са подредени лексикографски нарастващо) и ако не съществува противоречие да се състави една възможна азбука от тези символи.

Пример:

abc
acc
baby
bdj
cafe

Тук думите са подредени лексикографски нарастващо, като една примерна азбука е a, d, b, c, y, j, f, e. В червено са посочени зависимостите, които задължително трябва да се спазят в азбуката.

a → b → c
↘ d

abc
acc
bdj
bad
ccc
damn

Тук думите не са лексикографски наредени, понеже ако моделираме зависимостите като граф, то получаваме цикличност, което няма как да се случи в линейна наредба каквато е азбуката.

d → a → b → c

Решение: За да видим дали има противоречия в азбуката, то трябва да видим за всички двойки съседни думи коя е първата буква, която се разминава в двете думи. Тогава ако думите са лексикографски подредени, то тогава получаваме "зависимост" в азбуката, т.е. коя буква е преди коя в неизвестната азбука. Единствената особеност в лексикографската наредба е че думата aa е преди aaa. Ако във нашият граф на зависимостите съществува цикъл, то тогава не съществува азбука, която да отговаря на следната лексикографска наредба на думите (има противоречие). Ако не съществува цикъл в нашия граф на зависимостите, то можем да върнем една възможна азбука като направим топологическа сортировка на графа.

ExampleSolution(words[1...n][1...l] : масив от думи с max дължина l)
1. G - празен граф
2. for i ← 1 to n-1
3. minLen ← min(words[i].length, words[i+1].length)
4. for j ← 1 to k
5. if(j=k and words[i][j] = words[i+1][j] and words[i].length > words[i+1].length)
6. return false // случаят с aa и aaa
7. if (words[i][j] != words[i+1][j])
8. добави върховете words[i][j] и words[i+1][j] към графа и добави ребро (words[i][j], words[i+1][j])
9. break
10. провери за цикъл в G например чрез DFS
11. ако има цикъл върни false, иначе върни топ. сортировка на G.

Тук за простота на псевдокода разчитам, че всички букви от азбуката са добавени като върхове в графа.

зад. 9/ На шахматна дъска 8×8 са поставени кон, топ и царица. С колко най-малко хода на коня можем да вземем топа и царицата при условие, че те не променят първоначалната си позиция?

Решение: Ще моделираме задачата като графов. Върховете на графа ще са всички възможни клетки (a1, a2, ..., h7, h8) а ребрата ще са възможните ходове на коня. Сега трябва да намерим най-късият път от коня до царицата, от коня до топа и от топа до царицата (което е същото като от царицата до топа). Това може да стане чрез BFS. Накрая отговора е min(distance(кон, царица), (кон, топ) + distance(топ, царица).

зад. 10/ Дадени са n на брой комуникационни кули. Дадени са ни цените за изграждането на връзка между всички двойки комуникационни кули. Предложете алгоритъм, който построява връзки между кулите за минимална цена, така че да може да се комуникира от всяка кула до всяка друга (позволено е преминаването през междинни кули).

Решение: На пръв поглед задачата може да изглежда като задача за намиране на МПД, но това не е напълно вярно. Това е понеже ребрата между някои кули 0 може да са отрицателни, а задачата за намиране на МПД е дефинирана, когато ребрата са в $\mathbb{R}^+$. Ребрата може да са отрицателни понеже ако цената за построяване на връзка е например 200лв., а на нас ще ни платят 1000лв., то общо построяването на тази връзка ще струва −800лв. (т.е. ще спечелим 800лв.) Тогава за да решим тази задача първоначално ще добавим всички отрицателни ребра след което ще допостроим "МПД"-то докато нямаме само 1 свързана компонента (подобно на алгоритъма на Kruskal). Понеже графът е пълен, то тогава е по-добре да използваме алгоритъма на Prim, но преди това направим всички ребра да са в $\mathbb{R}^+$. Това е валидно (виж задачата в мъдрионните записки) и може да стане като добавим достатъчно голяма константа към теглата на всяко ребро (поне |най-лошото| + 1). Тогава използваме алгоритъма на Prim, след което връщаме оригиналните тегла на ребрата и добавяме всички отрицателни ребра, които не са били добавени от алгоритъма на Prim.

зад. 11/ Дадени са n на брой точки в равнината $p_1 \dots p_n$. Дадена ни е функция $Dist(p_i, p_j)$, която изчислява разстоянието между 2 точки за $\Theta(1)$ време. Дадено ни е k, което е естествено число и е по-малко от n. Предложете алгоритъм със сложност $O(n^2 \lg n)$, който разбива точките на k дяла, така че минималното разстояние между два дяла да е максимално.

Решение: Ще моделираме задачата като графова. Върховете ще са точките, а ребрата ще са тегловни с тегло $Dist(p_i, p_j)$ $\forall i \neq j$. Ще използваме алгоритъма на Kruskal за намиране на МПД, като спрем изпълнението, когато дяловете, които изпълнението на алгоритъма конструира, станат от n на k. (вместо цял работи алгоритъма в ютуб/лекционните записки). Решението е коректно, понеже сме взели най-леките ребра и минималното разстояние между два дяла ще е максимално и то ще е точно това ребро, което при добавяне би направило дяловете k-1 на брой.

зад. 12/ Дадено е дърво. Да се намери диаметъра на дървото.

Решение: Диаметър на дърво е най-дългият път между 2 върха в дървото. Него искаме да го намерим като от произволен връх пуснем BFS и намерим най-отдалечения връх от него. След това отново пускаме BFS от намерения най-отдалечен връх и диаметърът на дървото ще е най-дългият път който намерим със второто пускане на BFS.

Заб: Предложеният алгоритъм работи единствено, ако подаденият граф е дърво. Съобразете защо със следният контрапример: