1. Графи. Дървета. Обхождане на графи.

Анотация: Дефиниции за краен ориентиран (мулти)граф и краен неориентиран (мулти)граф. Дефиниции за маршрут (контур) в ориентиран мултиграф и път (цикъл) в неориентиран мултиграф. Свързаност и свързани компоненти на граф. Дефиниция на дърво и кореново дърво. Доказателство, че всяко кореново дърво е дърво и |V|=|E|+1. Покриващо дърво на граф. Обхождане на граф в широчина и дълбочина. Ойлерови обхождания на мултиграф. Теореми за съществуване на Ойлеров цикъл (с доказателство) и Ойлеров път.

Краен ориентиран (мулти)граф

Нека $V = \{v_1, v_2, \ldots, v_n\}$ е крайно множество, елементите на което ще наричаме върхове, а $E = \{e_1, e_2, \ldots, e_m\}$ е крайно множество, елементите на което ще наричаме ребра. **Функцията** $f_G : E \to (V \times V)$, съпоставяща на всяко ребро – **наредена** двойка от върхове, наричаме краен ориентиран (мулти)граф. Записваме $G(V, E, f_G)$ и четем: „краен ориентиран (мулти)граф G с върхове V, ребра E и свързваща функция f_G“.

Това, че в дефиницията по-горе слагаме думичката „мулти“ в скоби, означава, че когато казваме само „краен ориентиран граф“ ще разбираме, че между два върха в графа може да има повече от едно ребро. Ако искаме да индикираме, че между два върха има не повече от едно ребро, трябва да добавим рестрикция над f_G и да изискваме тази функция да е инективно изображение (инекция) – тоест да е изпълнено следното условие: $\forall e_i, e_j \in E \wedge i \neq j \,\big|\, f_G(e_i) \neq f_G(e_j)$.

Графите представяме в равнината, като всеки връх $v \in V$ представяме с точка, а всяко ребро $e \in E$, такова, че $f_G(e) = (v_i, v_j)$ и $i \neq j$ представяме с отсечка, с начало точката, с която сме представили v_i и край – точката, с която сме представили v_j. В случая когато $i = j$, реброто наричаме примка и не може да го представим с отсечка, тъй като тя ще има нулева дължина. Затова го представяме с крива, която има за начало и край точката, с която сме представили v_i и гледаме тази крива да е възможно най-проста, за да не усложняваме графичната репрезентация (в случая на ориентиран граф дефинираме начало и край на реброто и поставяме стрелка в края на отсечката, която го репрезентира).

Краен неориентиран (мулти)граф

Нека $V = \{v_1, v_2, \ldots, v_n\}$ е крайно множество, елементите на което ще наричаме върхове, а $E = \{e_1, e_2, \ldots, e_m\}$ е крайно множество, елементите на което ще наричаме ребра. **Релацията** $R_G : E \subseteq V \times V$ дефинираме по следния начин: два върха от V са в релация тогава и само тогава, когато между тях има ребро. $G(V, E, R_G)$ е неориентиран (мулти)граф. Очевидно релацията R_G е симетрична (тъй като графа е неориентиран). Ако искаме да задължим G да няма примки – додефинираме R_G да е антирефлексивна. Ако пък искаме неориентираният граф да е мултиграф, вместо множеството $E \subseteq V \times V$ взимаме мултимножеството от елементите на $V \times V$, т.е. $E \subseteq \{V \times V\}_{\mathscr{M}}$.

Маршрут (контур) в ориентиран мултиграф

Последователност от върхове $v_{i_0}, v_{i_1}, \ldots, v_{i_k}$ на ориентирания мултиграф $G(V, E, f_G)$ наричаме маршрут в G, ако между всеки два съседни върха от последователността съществува ребро в графа. Формално, $\forall j \in \overline{0, k-1} \,|\, \exists e \in E : f_G(e) = (v_{i_j}, v_{i_{j+1}})$. Числото k наричаме дължина на маршрута. Когато началният и крайният връх на маршрута съвпадат, т.е. $v_{i_0} = v_{i_k}$, тогава маршрута наричаме контур.

Път (цикъл) в неориентиран мултиграф

Последователността от върхове $v_{i_0}, v_{i_1}, \ldots, v_{i_k}$ на неориентирания мултиграф $G(V, E, R_G : E \subseteq \{V \times V\}_{\mathscr{M}})$ наричаме път в G, ако между всеки два съседни върха от последователността съществува ребро в графа. Формално $\forall j \in \overline{0, k-1} \,|\, (v_{i_j}, v_{i_{j+1}}) \in R_G$ (симетрична). Тъй като в този случай на мултиграф, релацията е над мултимножеството $\{V \times V\}_{\mathscr{M}}$, то ще дефинираме тривиалния път от u до u, за всяко $u \in V$ да бъде с нулева дължина (тегло). Числото k наричаме дължина на пътя. Когато началният и крайният връх на пътя съвпадат, т.е. $v_{i_0} = v_{i_k}$, тогава пътя наричаме цикъл.

Свързаност

За всеки ориентиран или неориентиран (мулти)граф $G(V, E)$, дефинираме релацията $P_G \subseteq V \times V$ по следния начин: $(v_i, v_j) \in P_G \Leftrightarrow \exists$ маршрут (за ориентирания случай) или път (за неориентирания случай) в G от v_i до v_j и от v_j до v_i.

- Два върха u и v принадлежащи на множеството от върхове V на неориентирания (мулти)графа $G(V, E, f)$ ще наричаме свързани, ако са в релация P_G.
- Два върха u и v принадлежащи на множеството от върхове V на ориентирания (мулти)граф $G(V, E, R)$ ще наричаме силно свързани, ако са в релация P_Q. Ако има път само от единя до другия връх, то тези върхове са слабо свързани.

Твърдение

Релацията P_G е релация на еквиваленсност. Това е така, тъй като лесно може да се докаже, че тази релация е рефлексивна, симетрична и транзитивна. Рефлексивна е по тривиални причини, тъй като може да додефинираме нулев тривиален път от всеки връх до себе си. Симетричността следва директно от начина, по който релацията е зададена, а транзитивността може лесно да се покаже. Нека $(u, v) \in P_G$ и $(v, w) \in P_G$. Това означава, че имаме път от u до v и път от v до w. Тогава ще имаме път от w до v и път от v до u директно от дефиницията на релацията. След като имаме тези пътища може да построим пътища от v до w и от w до v, минаващи през u, което директно означава, че $(v, w) \in P_G$ и това показва, че релацията е и транзитивна. В случай, че графът G е ориентиран, може да заменим понятието „път“ с „маршрут“ в твърденията по-горе и пак ще са верни.

Клас на еквиваленстност породен от елемент

Нека V е непразно множество и R_E е релация на еквивалентност в $\{V \times V\}_{\mathscr{M}}$. Клас на еквивалентност относно R_E, съдържащ елемента $v \in V$ се нарича следното множество: $[v] = \{u \,|\, u \in V \wedge u R_E v\}$.

Компоненти на свързаност

Дефинираната от нас релация на еквивалентност P_Q разбива множеството от върхове V на графа G на класове на еквивалентност. Всеки клас на еквивалентност относно P_Q е свързана компонента в граф, ако графа G е неориентиран. В случай, че графа G е ориентиран, тази компонента се нарича силно свързана компонента.

Един неориентиран граф е свързан тогава и само тогава, когато има една компонента на свързаност.

Един ориентиран граф може да е свързан, но да не представлява силно свързана компонента.

Дърво
Дърво наричаме всеки свързан неориентиран граф, в който няма цикли.
По-надолу, навсякъде където се появява символа G да се чете като неориентиран граф.

Еквивалентни твърдения
- $G(V, E)$ е дърво
- Дърво е неориентиран свързан граф без цикли
- Всеки два върха от V от $G(V, E)$ са свързани с точно един прост път (прост или нормален път е този, в който не се повтарят нито ребра нито върхове)
- G е свързан (има точно една компонента на свързаност) и броя на ребрата е с единица по-малък от броя на върховете $|E| = |V| - 1$ (ще докажем по-долу)
- G е свързан и тази свързаност е минимална по изключване на ребра (т.е. премахването на което и да е ребро от G ще увеличи броя на компонентите на свързаност с една (ще станат две))
- G е ацикличен и тази ацикличност е максимална по включване на ребра (т.е. добавянето на ново ребро между два върха от V ще образува цикъл)

Кореново дърво
Индуктивна дефиниция с операция по присъединяване на връх. $D(\{r\}, \emptyset)$ е дърво с корен r и единствен връх r (този връх е и корен и листо). Нека $D(V, E)$ е дърво с корен r и листа $l_1, l_2, \ldots, l_k$. Нека $v \in V$ и $u \notin V$. Тогава $D'(V \cup u, E \cup \{(v, u)\})$ е дърво с корен r. Ако $\exists i \in \overline{1, k} : v = l_i$ (v е било листо в D), то листата на D' ще са $l_1, \ldots, l_{i-1}, u, l_{i+1}, \ldots, l_k$. Ако $\forall i \in \overline{1, k} : v \neq l_i$ (v не е било листо в D), то листата на D' ще са $l_1, l_2, \ldots, l_k, u$.

Всяко кореново дърво е дърво. Доказателство:
База: $D(\{r\}, \emptyset)$ е свързан граф без цикли.
Индукционна хипотеза: Да допуснем, че $D(V, E)$ е свързан граф без цикли.
Индукционна стъпка:
Добавяме върха w по следния начин: $D'(V', E') = D'(V \cup \{w\}, E \cup \{(u, w)\})$.
1. D' е свързан, тъй като между всеки два върха от V има път (от индукционната хипотеза), а от w до всеки друг връх има път с първо ребро (w, u).
2. Да допуснем, че в D' има цикъл. Тогава (u, w) ще участва в цикъл, тъй като от индукционната хипотеза D е ацикличен и единствения възможен начин да се е образувал цикъл ще е в следствие от добавянето на новото ребро свързващо новодобавения връх w. Но $d(w) = 1 \Rightarrow$ противоречие ($\Rightarrow\Leftarrow$). Следователно D' е ацикличен.

Дълбочина (височина) на връх $v \in V$ от кореновото дърво $D(V, E)$ с корен r наричаме дължината на единствения път от корена r до v (броя на ребрата). Височината на дървото D наричаме максималната дълбочина от дълбочините на всички върхове от V.

Разклоненост на връх $v \in V$ от кореновото дърво $D(V, E)$ с корен r наричаме броя на съседните върхове на v минус един (искаме само броя на синовете, без бащата). Разклоненост на дървото D наричаме максималната разклоненост от разклоненостите на всички върхове от V.

$|\mathbf{V}| = |\mathbf{E}| + \mathbf{1}$ за всяко дърво $\mathbf{D}(\mathbf{V}, \mathbf{E})$
Ще докажем твърдението чрез структурна индукция.
База: За дървото $D(\{r\}, \emptyset)$ имаме $|V| = 1$, тъй като множеството от върхове V се състои само от един връх r – корена на дървото и $|E| = 0$. Очевидно $|V| = 1 = 0 + 1 = = |E| + 1$.

Индукционна хипотеза: Да допуснем, че $T_i(V_i, E_i)$, за $i = \overline{1, k}$, са k дървета, за които е изпълнено, че $\forall i \in \overline{1, k} : |E_i| = |V_i| - 1$.

Индукционна стъпка: Разглеждаме дърво D с корен r и синове корените на k-те поддървета от индукционната хипотезата. За D сега имаме, че $|V| = 1 + \sum_{i=1}^{k} |V_i|$ и $|E| = \sum_{i=1}^{k} \big(|E_i| + 1\big) = k + \sum_{i=1}^{k} |E_i|$. Остава само да проверим верността на желаното равенство.

$$|E| = k + \sum_{i=1}^{k} |E_i| = k + |V_1| - 1 + |V_2| - 1 + \ldots + |V_k| - 1 = \not{k} + \sum_{i=1}^{k} |V_i| - \not{k} = |V| - 1$$

Което искахме да докажем. □

Покриващо дърво на граф
Дървото $D(V, E')$ наричаме покриващо дърво за графа $G(V, E)$, ако $E' \subseteq E$.

Обхождане на графи
Различните алгоритмични схеми, които дават отговор на въпроса „Кой е следващият връх от графа който да посетим?“ задават и различни алгоритми за обхождане на графи.

Разглеждаме свързан граф $G(V, E)$. В случай, че графа не е свързан, може да приложим алгоритъма за обхождане върху всяка негова компонента на свързаност като отделен граф. Целта е да се посетят всички върхове от графа, т.е. обхождането да представлява покриваща гора (в сучай на свързан граф гората ще се състои само от едно дърво) на G.

В обхожданията и в широчина и в дълбочина ще имаме еднакви първи три стъпки на алгоритъма:

1. Нека броя на върховете на графа $G(V, E)$ е $|V| = n$. Номерираме тези върхове с числата от 0 до $n - 1$. Тези числа ще използваме за идентификационни номера за всеки връх;
2. Създаваме празно множество $VIS = \{\emptyset\}$, в което да добавяме идентификационните номера на вече посетените върхове;
3. Избираме идентификационен номер на начален връх $k : 0 \leq k \leq n - 1$, който да разгледаме първи;

Обхождане в широчина (разширен вариант по нива):

4. Създаваме празна опашка $QUE \leftarrow \{\emptyset\}$. Опашка е структура от данни, на принципа за добавяне и изваждане на елемент FIFO: „First In – First Out“ (първият добавен елемент е първия изваден);
5. Добавяме избрания връх в опашката и добавяме идентификационния му номер в множеството с посетени върхове: $QUE \leftarrow \{v_k\}, VIS \leftarrow VIS \cup \{k\}$;
6. Създаваме променлива $lvl \leftarrow 0$, с която да отбелязваме нивото на което се намират разглежданите върхове;
7. Докато има върхове в опашката (докато $|QUE| > 0$):

- създаваме променлива $size$, която инициализираме с текущия размер на опашката: $size \leftarrow |QUE|$;

8. Докато $size \neq 0$ (разглеждаме всички върхове от ниво lvl):
   - вадим връх от опашката, който ще запаметим като текущ в променливата $current$;
   - процесираме върха (т.е. прилагаме бизнес логиката си върху него спрямо задачата);
   - добавяме идентификационният му номер в множеството с посетени върхове $VIS \leftarrow VIS \cup current_{id}$;
   - добавяме всичките му непосетени съседи в опашката;
9. Актуализираме размера на опашката и нивото: $size \leftarrow |QUE|, lvl \leftarrow lvl + 1$;

Обхождане в дълбочина:

4. Инициализираме текущият връх с началния и неопределен баща: $current \leftarrow k$, $parent(k) = \emptyset$;
5. Избираме непосетен съсед $child$ на текущия връх $current$;
6. Ако намерим такъв връх – траверсираме напред: текущият връх става негов баща: $parent(child) \leftarrow current$ и актуализираме новия текущ връх $current \leftarrow child$;
7. Добавяме идентификационният номер на върха $child$ в множеството с посетени върхове: $VIS \leftarrow VIS \cup \{child_{id}\}$
8. Ако няма такъв връх – връщаме се назад: $current \leftarrow parent(current)$ и се връщаме рекурсивно на стъпка 5 от алгоритъма.

Процесирането на всеки връх при обхождането в дълбочина може да стане в развиването (преди рекурсивното извикване) или в свиването (след рекурсивното извикване) на рекурсията, в зависимост от задачата.

Очевидно обхождането в дълбочина използва рекурсивна техника, което носи своите рискове. Параметрите на тези рекурсивни извиквания се съхраняват в стековата памет, която не е безкрайна и при графи с достатъчно много на брой върхове може да препълним тази памет. Този проблем може да решим, като предоставим итеративна алгоритмична схема на обхождането в дълбочина. Това може да стане с една единствена промяна в алгоритмичната схема на обхождането в широчина. Необходимо е просто да променим структурата от данни в точка 4 от алгоритъма от такава, която е FIFO в такава, която е LIFO „Last In – First Out“ (последният добавен е първия изваден). Т.е. единственото което трябва да променим е вместо празна опашка QUE да инициализираме празен стек STC.

Обхождане в широчина намира най-късите пътища от началния връх до всички останали върхове в графа.

Ойлерови обхождания на мултиграф

Път в свързания граф G, който минава през всяко ребро на G точно веднъж, наричаме Ойлеров път. Ако графът G има Ойлеров път, който е цикъл (началният и крайният връх на Ойлеровия път съвпадат), то графът G наричаме Ойлеров граф.

Степен на връх v

Броят на ребрата, които излизат от даден връх $v \in V$ от неориентирания мултиграф $G(V, E)$ наричаме степен на връх v и бележим с $d(v)$.

Теореми за съществуване на Ойлеров цикъл (с доказателство) и Ойлеров път

Един граф G е Ойлеров (т.е. има Ойлеров цикъл) $\Leftrightarrow$ G е свързан и всеки негов връх е от четна степен.

Единг граф G има Ойлеров път, който не е цикъл $\Leftrightarrow$ G е свързан и има точно два върха от нечетна степен.

Доказателство:

($\Rightarrow$) Нека G е Ойлеров граф и нека $v = v_0, v_1, \ldots, v_k = v$ е Ойлеров цикъл в G. Всяко срещане на връх u в редицата $v_0, \ldots, v_{k-1}$ съответства на две ребра през u: ако $u = v_0$, ребрата са $\{v_0, v_1\}$ и $\{v_{k-1}, v_0\}$; ако $u = v_i$ и $i > 0$, ребрата са $\{v_i, v_{i+1}\}$ и $\{v_{i-1}, v_i\}$. Тъй като всички ребра в графа се срещат точно по веднъж в цикъла, степента на произволен връх $u \in V$ е равна на два пъти броя на срещанията на u в цикъла. Така $d(u)$ е четно число.

($\Leftarrow$) Нека сега G е свързан граф, в който всеки връх има четна степен и $u \in V$. Ще построим Ойлеров цикъл C през u.

Обявяваме u за текущ връх t, $C = (u)$.

1. Докато има необходено ребро $e = \{t, v\}$ през t, добавяме t към C, обявяваме e за обходено и v за текущ връх t, $C = (u)$.

Тъй като при всяко преминаване през цикъла, броят на необходените ребра намалява, цикълът ще завърщи след краен брой стъпки k, и $C = (u = v_0, v_1, \ldots, v_k)$. При това твърдим, че $v_k = u$. Наистина, да допуснем, че $v_k \neq u$. Тъй като алгоритъмът завършва, всички ребра през v_k са обходени и на $k + 1$-вата стъпка няма необходени ребра през v_k. На стъпката, на която v_k става текущ, обхождаме едно ребро през v_k и броят на обходените ребра през v_k е нечетен. На следващата стъпка, ако v_k престане да е текущ, обхождаме второ ребро през v_k и броят на обходените ребра през v_k става четен. Така на последната стъпка, на която v_k става текущ, броят на обходените ребра през v_k е нечетен, но тогава алгоритъмът завършва, защото няма как да се избере следващ текущ връх, всички ребра през v_k са обходени. Това означава, че през v_k има нечетен брой ребра, което противоречи на четността на степента на v_k.

2. Следователно $v_k = u$ и C е цикъл. Ако C съдържа всички ребра, C е Ойлеров цикъл. Иначе, C съдържа поне един връх v_i, който е край на необходено ребро. Това следва от свързаността на графа. Повтаряме цикъла 1. с начален връх v_i като строим нов цикъл $C' = (v_i = u_0, u_1, \ldots, u_m = v_i)$. Заменяме C с цикъла
$$C = (v_0, \ldots, v_{i-1}, v_i = u_0, u_1, \ldots, u_m = v_i, v_{i+1}, \ldots, v_k)$$
и отново се връщаме на точка 2.

Problem: FIND EULER CYCLE
Solution: LINK TO SOLUTION