ДОМАШНО № 3 ПО "ДИЗАЙН И АНАЛИЗ НА АЛГОРИТМИ"
ЗА СПЕЦИАЛНОСТ "КОМПЮТЪРНИ НАУКИ", 2. КУРС, 1. ПОТОК
(СУ, ФМИ, ЛЕТЕН СЕМЕСТЪР НА 2018 / 2019 УЧ. Г.)

За предложените алгоритмични задачи установете дали са полиномиални, или са NP-пълни. Ако смятате, че някоя задача е полиномиална, съставете алгоритъм с полиномиална времева сложност, опишете го с думи или на псевдокод и анализирайте неговата времева сложност. Ако смятате задачата за NP-пълна, съставете полиномиална редукция и полиномиален алгоритъм за проверка на предложено решение, опишете ги (псевдокодът е задължителен за редукцията), анализирайте сложностите им по време и докажете коректността на редукцията.

Задачи 1 и 2 — играта "Digger"

В известната игра "Digger" един миньор се движи по игралното табло и събира смарагди. Всеки смарагд носи на миньора определен брой точки (един и същ за всички скъпоценни камъни). При взимане на смарагд прозвучава нота от октавата: долно до, ре, ми, фа, сол, ла, си, горно до. При първо взимане на смарагд се изпълнява долно до. Следваща нота (ре, ми, фа и т.н.) прозвучава само ако на предишния ход миньорът е взел смарагд. Когато миньорът премине през празно поле, октавата се прекъсва. От следващия смарагд започва нова октава (т.е. изпълнява се долно до).

Колкото повече смарагди миньорът вземе последователно, толкова по-дълга част от октавата звучи. Ако миньорът вземе смарагди с осем последователни хода, т.е. ако изпълни цялата октава, получава бонус (допълнителни точки). При взимане на следващ смарагд започва нова октава — изпълнява се пак долно до, независимо дали е взет веднага след осемте.

Миньорът иска да получи възможно най-много точки от всяко ниво на играта. Затова трябва да организира ходовете си така, че октавата да прозвучи докрай колкото може повече пъти.

Моделираме всяко ниво чрез ориентиран граф $G(V, E)$ с $|V| = n$ върха и $|E| = m$ ребра:

— Върховете на графа са полетата на игралното табло.

— Ребрата на графа съответстват на възможните ходове по таблото.

— Всеки връх притежава маркер — дали на съответното поле има смарагд.

— Посочен е един връх s — полето, на което се намира миньорът в началото на играта.

Миньорът тръгва от върха s и се движи по ребрата на графа. През един и същ връх на графа миньорът може да премине колкото пъти иска, но ако във върха има смарагд, миньорът го взима задължително при първото преминаване, след което върхът остава празен (без смарагд).

Разглеждаме две алгоритмични задачи за разпознаване с еднакъв вход: $G(V, E)$, n, m, s. Задачите се различават само по въпроса: Възможно ли е миньорът да се движи така, че:

— поне веднъж да се изпълни цялата октава (задача 1) ?

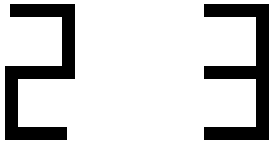
— да вземе всички смарагди, без да прекъсне никоя октава (задача 2) ?

В задача 2 се пита дали е възможно миньорът да вземе всички смарагди на групи по осем. За целта трябва броят k на смарагдите да се дели на 8 и октавата да бъде изцяло изпълнена $k / 8$ пъти.

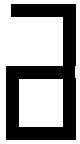
Намерете класовете на времева сложност на задача 1 и задача 2.

Задача 3 — опаковане на графи

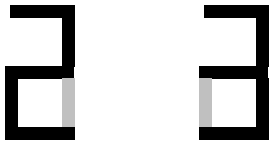
Знаците от някакъв шрифт се изписват само с прави черти, тоест всеки знак се изобразява като начупена линия. Искаме да създадем светлинно табло, което да може да показва всички знаци от този шрифт. Таблото трябва да бъде възможно най-евтино, тоест да има възможно най-малко светещи отсечки. За простота да разгледаме шрифт, съставен само от два знака, например 2 и 3.



Всеки от двата знака се състои от пет отсечки. Следователно табло с общо 5 + 5 = 10 отсечки ще свърши работа. Само че този брой (десет) е излишно голям. Шест отсечки са достатъчни, както се вижда от показания тук пример.



На това табло можем да изобразим всяка от цифрите 2 и 3, като включим едни отсечки да светят, а другите оставим изключени. Черните черти на чертежа означават светещи отсечки на таблото, а сивите черти — изключени, тоест несветещи, отсечки.



В такава ситуация е естествено да използваме теорията на графите: върховете на начупената линия ще бъдат върхове на граф, а отсечките на начупената линия — ребра на графа.

Друга ситуация, различна на пръв поглед, която се свежда до графи по аналогичен начин: Искаме да образуваме работен екип измежду множество наши познати. Някои от тях се познават, а други — не. Екипът трябва да се състои от няколко подгрупи, като се допуска един и същи човек да бъде член на няколко подгрупи. Отново да приемем за простота, че подгрупите са само две. Едната подгрупа да се състои от началник и трима подчинени: началникът е длъжен да познава подчинените си (за да знае на кого какви задачи да възлага), но те не бива да се познават един друг (за да не правят заговори срещу шефа). Другата работна група трябва да бъде шпионска верига от четирима души: първият ще събира сведения и ще ги предава на втория, вторият — на третия, третият — на четвъртия, четвъртият — на нас. С цел секретност (ако някой от тях бъде разкрит, да не се стига лесно до останалите) искаме първият да познава само втория член на веригата, вторият — само първия и третия, третият — само втория и четвъртия, четвъртият — само третия. Пита се колко души най-малко трябва да вземем в екипа.

Тази ситуация може да бъде изобразена така:

Подгрупата, съставена от началник и трима подчинени, прилича на буквата Т: точката, от която се разклоняват трите отсечки, е началникът, а трите върхчета на буквата са неговите подчинени. Шпионската верига може да се изобрази като буквата П: четиримата агенти съответстват на четирите върхчета на буквата.

Можем да съставим екип от 4 + 4 = 8 души, но това е излишно много. Петима членове са достатъчни, ако техните познанства образуват полегналата буква F.

Тази конфигурация съдържа двете желани подгрупи — буквите Т и П.

Примерите по-горе са описания на практически ситуации, които могат да се формализират по различни начини. За да моделираме задачата, ще използваме средства от теорията на графите. Една възможна формулировка гласи:

По дадени неориентирани графи G_1, G_2, ..., G_k да се построи възможно най-малък неориентиран граф G, който съдържа като подграфи всички дадени графи G_1, G_2, ..., G_k.

Тази формулировка добре отразява свързването на обектите (топологичната информация). Това стига за втората ситуация (екипа с подгрупите), но не и за първата (знаците от един шрифт): губим ориентацията на знаците. Например 8 и ∞ са различни знаци, но им съответства един граф. Аналогично, П и С са различни букви, но ако ги изобразим с отсечки, ще изглеждат еднакво, само че завъртени под прав ъгъл една спрямо друга.

Въпреки посочения недостатък все пак ще използваме графи, за да не усложняваме задачата с допълнителни сведения за ориентацията на обектите в пространството, още повече че понякога липсва такава информация (например в задачата за екипа и подгрупите).

Има и друг пропуск: не е казано в какъв смисъл графът G е най-малък. Това може да бъде уточнено по различни начини. Например в първата описана ситуация (знаците от един шрифт) графът G трябва да съдържа възможно най-малко ребра (чертичките на светлинното табло), а пък във втората ситуация (екипът и подгрупите) G трябва да има възможно най-малко върхове (тоест екипът да съдържа възможно най-малко членове).

Можем да опитаме минимизация по двата критерия едновременно — по броя на върховете и по броя на ребрата на графа. Сега задачата гласи:

По дадени неориентирани графи G_1, G_2, ..., G_k да се построи неориентиран граф G с най-малко върхове и ребра, който съдържа като подграфи всички графи G_1, G_2, ..., G_k.

Дадените примери показват, че с тази задача можем да моделираме разнообразни ситуации от практиката. Възниква въпросът има ли бърз (тоест полиномиален) алгоритъм. За съжаление, се оказва, че задачата е NP-пълна, затова не съществува полиномиален алгоритъм (ако P ≠ NP). Нещо повече, задачата остава NP-пълна дори в частния случай $k = 2$. Ето защо ще разгледаме само този частен случай. (Това е достатъчно: щом частният случай е NP-труден, то следва, че и общият случай е такъв.)

Твърдението за NP-пълнота е неточно: само задачите за разпознаване могат да са NP-пълни, а нашата задача е оптимизационна. Когато казваме, че тя е NP-пълна, подразбираме, че NP-пълна е всъщност съответната задача за разпознаване. Нека я формулираме изрично. Това ще бъде окончателната и най-точна формулировка (разглеждаме само частния случай $k = 2$).

Определяме следната задача за разпознаване, да я наречем *Опаковане на графи*:
— Вход: Дадени са неориентираните графи G_1 и G_2 (това означава, че са дадени също броят на техните върхове n_1 и n_2 и броят на техните ребра m_1 и m_2 — четири цели неотрицателни числа). Дадени са още две цели неотрицателни числа n и m.
— Въпрос: Съществува ли неориентиран граф G с не повече от n върха и не повече от m ребра, който съдържа подграф, изоморфен на G_1, и подграф, изоморфен на G_2 ?

Докажете, че алгоритмичната задача *Опаковане на графи* е NP-пълна.

РЕШЕНИЯ

Задача 1 е полиномиална, тоест принадлежи на класа P, защото съществува алгоритъм, който я решава за полиномиално време. Това е методът пълно изчерпване. Той не е най-бързият, но все пак има полиномиална сложност в задача 1, а това е достатъчно за целите на решението.

Нека v е произволен връх на дадения граф G. Търсим всички пътища с начален връх v, които съдържат осем върха (включително v), тоест седем ребра, не повтарят върхове и всеки техен връх съдържа смарагд. Проверката, дали поредният връх съдържа смарагд, отнема константно време. Дали поредният връх не повтаря върхове, се проверява за време, пропорционално на дължината на построената част от пътя (не повече от 7), тоест тази проверка има времева сложност: $\Theta(1)$.

Щом проверката на всеки нов връх от пътя изисква време $\Theta(1)$, то общото време за проверка на всички върхове от пътя (не повече от 8) също е $\Theta(1)$. Затуй построяването на всеки от пътищата изразходва време $\Theta(1)$, а общото време за всички пътища е от порядъка на техния брой.

Пътищата се строят така: започвайки от върха v, проверяваме дали той съдържа смарагд; ако да — разглеждаме всички ребра, излизащи от v, проверяваме другите им краища за смарагди и разглеждаме всички ребра, излизащи от новите върхове, и тъй нататък. От всеки връх без последния (осмия) излизат не повече от m ребра, следователно за седемте ребра на пътя има не повече от m^7 възможности. Следователно броят на пътищата от описания вид, а значи и времето за тяхното генериране, е от порядък $O(m^7)$.

За всеки от тези пътища p търсим път q от върха s до върха v; пътят q не бива да минава през никой връх от p, различен от v. За целта използваме някоя от схемите търсене в ширина или търсене в дълбочина, като предварително обявяваме за обходени върховете от p, различни от v. Маркирането на върховете от p (седем на брой) става за константно време, а същинското търсене на пътя q — за линейно време $O(n+m)$. Няма значение дали търсим в ширина, или в дълбочина. Ако държим да намерим най-къс път q, трябва да използваме търсене в ширина.

Ако търсенето е успешно, прекратяваме изпълнението на алгоритъма със следния резултат: октавата може да бъде изпълнена поне веднъж. За целта миньорът трябва да се придвижи от s до v по пътя q, а от v трябва да продължи по пътя p до неговия край.

Ако търсенето на пътя q завърши неуспешно, това означава, че октавата от избрания път p не може да бъде изпълнена: или миньорът изобщо не може да стигне от s до v, или може, обаче трябва междувременно да премине през някой от другите седем върха на p, при което е длъжен да вземе съответния смарагд, което ще доведе впоследствие до прекъсване на октавата. Ето защо се отказваме от текущия път p и опитваме някой от другите пътища с осем върха.

Както видяхме, търсенето на път q за всеки един от пътищата p изисква време $O(n+m)$, а броят на възможните пътища p е от порядък $O(m^7)$. Затова проверката на всички пътища p изразходва време $O((n+m)m^7)$. Ако за поне един от тях успеем да намерим подходящ път q, то алгоритъмът приключва с успех: октавата може да се изпълни поне веднъж.

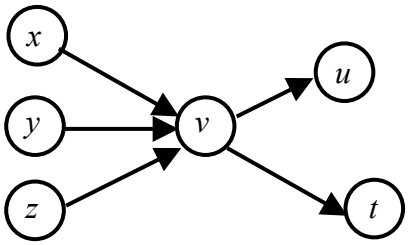
Ако всички проверки са неуспешни, то това означава, че октавата не може да бъде изпълнена с начален връх v. Но може би е възможно тя да се изпълни с друг начален връх. Затуй опитваме последователно с всички върхове на графа, тоест върхът v пробягва множеството V. Алгоритъмът връща съобщение за успех (че октавата може да бъде изпълнена) при първия срещнат връх v, за който проверката, описана по-горе, мине успешно (тоест за поне един от пътищата p с начало v бъде намерен подходящ път q).

В противен случай алгоритъмът връща съобщение за неуспех (че октавата не може да бъде изпълнена нито веднъж), след като обходи всички върхове на графа и не намери подходящ връх. Тъй като върховете на графа са n и проверката на всеки от тях изисква време $O((n+m)m^7)$, то времето на целия алгоритъм е $O(n(n+m)m^7)$. Този израз е полином от висока степен (девета), но все пак е полином, което означава, че алгоритъмът, макар и бавен, е все пак полиномиален. Следователно разглежданата алгоритмична задача за разпознаване (задача 1) е от класа P.

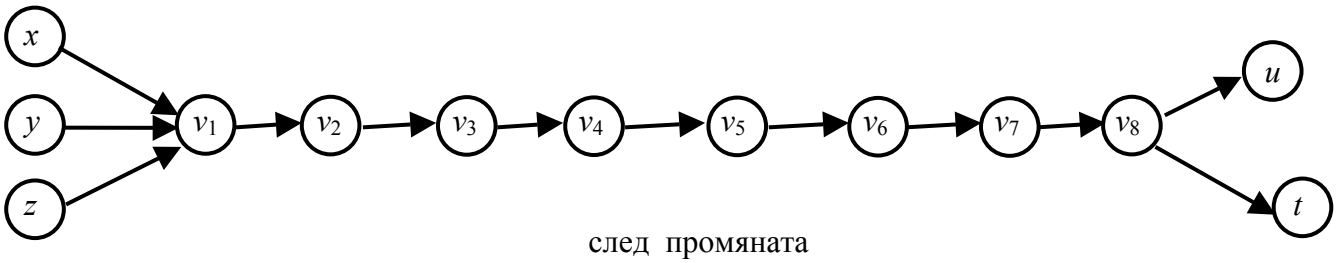
Задача 2 е NP-пълна. Че е NP-трудна, се доказва с помощта на редукция от задачата за разпознаване дали в даден ориентиран граф H има хамилтонов път с дадени начало a и край b. Формално описание на редукцията:

```
HamiltonianPath(H: directed graph;  a,b: vertices of H): bool
(G,s) ← Modify(H,a,b)
return DiggerProblem2(G,s)
```

Процедурата `Modify` се състои в следното: всеки връх v на графа H (с изключение на a и b) се заменя с осем върха, от всеки от които (без последния) излиза ребро към следващия. В първия от осемте върха влизат ребрата, които дотогава са влизали във v, а от последния връх излизат всички ребра, които дотогава са излизали от v.



преди промяната



след промяната

Върха a заменяме с редица от четири (а не осем) върха: a_1, a_2, a_3, a_4.

Върха b също заменяме с редица от четири върха: b_1, b_2, b_3, b_4.

Новия граф означаваме с G. Всеки негов връх съдържа по един смарагд. Ролята на s играе върхът a_1, тоест миньорът започва обхождането от a_1.

Коректност на редукцията: Ако в графа H има хамилтонов път с начало a и край b, той минава през всеки връх на H точно веднъж. На хамилтоновия път от H съответства хамилтонов път в G с начало s, получен чрез замяната на всеки връх от H със съответната редица в G. Например, ако $a\ x\ y\ b$ е хамилтонов път в H, то $a_1\ a_2\ a_3\ a_4\ x_1\ x_2\ x_3\ x_4\ x_5\ x_6\ x_7\ x_8\ y_1\ y_2\ y_3\ y_4\ y_5\ y_6\ y_7\ y_8\ b_1\ b_2\ b_3\ b_4$ е хамилтонов път в G.

Броят на върховете на G се дели на осем, тъй като от всеки връх на H (с изключение на a и b) се получават по осем върха на G плюс осемте върха a_1, a_2, a_3, a_4, b_1, b_2, b_3, b_4. Тъй като във всеки връх на G има смарагд (по построение), то броят на смарагдите $k = n$ е кратен на осем. Освен това хамилтоновият път в G не повтаря върхове, затова никой връх не е празен при преминаване на миньора през него. Следователно никоя октава не се прекъсва и броят на октавите е максимален — една осма от броя на смарагдите.

Обратно, нека в G има маршрут с начало $s = a_1$, по който миньорът събира всички смарагди на групи по осем. При първите три хода миньорът няма избор: от $s = a_1$ отива в a_2, оттам — в a_3, накрая — в a_4. След това преминава към друг връх, например x_1, при което е принуден да мине през цялата редица $x_1\,x_2\,x_3\,x_4\,x_5\,x_6\,x_7\,x_8$ поради структурата на графа G. Смарагдите от $x_1\,x_2\,x_3\,x_4$ завършват октавата, започната от $a_1\,a_2\,a_3\,a_4$, а смарагдите от $x_5\,x_6\,x_7\,x_8$ започват нова октава. За да не я прекъсне, миньорът трябва да потърси непосетен връх y_1 и минава през цялата редица $y_1\,y_2\,y_3\,y_4\,y_5\,y_6\,y_7\,y_8$. Смарагдите от $y_1\,y_2\,y_3\,y_4$ завършват октавата, започната от $x_5\,x_6\,x_7\,x_8$, а тези от $y_5\,y_6\,y_7\,y_8$ започват нова октава. Като продължим тези еднотипни разсъждения, установяваме, че за да не прекъсне октава, миньорът трябва да минава само през непосетени върхове. От друга страна, понеже трябва да събере всички смарагди и във всеки връх има смарагд, то миньорът трябва да премине през всеки връх поне веднъж. От тези две твърдения следва, че миньорът минава през всеки връх на G точно веднъж. При това, за да завърши последната октава, без да започне нова, миньорът трябва да приключи маршрута си с върховете $b_1\,b_2\,b_3\,b_4$ в този ред. Така се получава хамилтонов път в G, който се състои от последователни осморки (или четворки), получени от един и същи връх на H, започва с $a_1\,a_2\,a_3\,a_4$ и завършва с $b_1\,b_2\,b_3\,b_4$. Като изтрием индексите и повторенията на буквите, получаваме съответен хамилтонов път в H. Например от пътя $a_1\,a_2\,a_3\,a_4\,x_1\,x_2\,x_3\,x_4\,x_5\,x_6\,x_7\,x_8\,y_1\,y_2\,y_3\,y_4\,y_5\,y_6\,y_7\,y_8\,b_1\,b_2\,b_3\,b_4$ получаваме $a\,x\,y\,b$. Полученият път в H е хамилтонов, защото минава през всеки връх на H по веднъж (а това е така, защото съответният път в G минава през всеки връх на G по веднъж и поради съответствието между върховете на G и върховете на H).

Доказахме следната еквивалентност: В произволния ориентиран граф H има хамилтонов път от върха a до върха b тогава и само тогава, когато в съответния му граф G, получен чрез описаната модификация, има маршрут, започващ от върха $s = a_1$, по който миньорът събира всички смарагди на групи по осем (без да прекъсне октава).

Следователно функциите `HamiltonianPath(H,a,b)` и `DiggerProblem2(G,s)` трябва винаги да връщат равни стойности, както е всъщност и според описанието на редукцията. Следователно тя е коректна.

Бързина на редукцията: Модификацията изисква едно обхождане на графа, като всеки връх се обработва за константно време, затова времето за цялата модификация е линейно: $\Theta(m + n)$, следователно полиномиално. Тоест редукцията е достатъчно бърза за целите на доказателството.

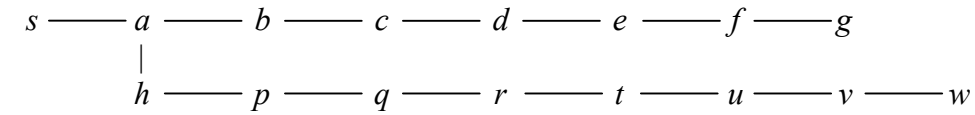
Дотук доказахме, че задача 2 е NP-трудна. Остава да докажем, че тя принадлежи на класа NP. Това следва от съществуването на полиномиален алгоритъм за проверка на предложено решение. Сертификат е маршрутът, по който миньорът трябва да събере смарагдите (този маршрут може да повтаря върхове). Маршрутът може да бъде зададен например като масив от върхове на графа.

Проверката на предложено решение се състои от шест стъпки:

1) Отхвърляме маршрута, ако е твърде дълъг (подробности — на следващата страница). Това сравнение изисква константно време.
2) Проверяваме дали маршрутът започва от върха s. Необходимо време: константа.
3) Проверяваме дали маршрутът е валиден, тоест дали номерата на върховете са от 1 до n и дали между всеки два последователни върха от маршрута има ребро. Време: $O(kn)$, където k е броят на върховете на маршрута (от всеки връх излизат най-много $n - 1$ ребра).
4) Маркираме всички върхове на графа като непосетени. Необходимо време: $\Theta(n)$.
5) Проверяваме дали маршрутът не прекъсва октава. За целта трябва да изминем маршрута. Докато се движим по маршрута, маркираме върховете му като посетени. Тази стъпка изразходва време, линейно спрямо дължината на маршрута, тоест $\Theta(k)$.
6) Проверяваме дали маршрутът минава през всички смарагди. За тази цел обхождаме върховете на графа и проверяваме има ли непосетен връх със смарагд. Време: $\Theta(n)$.

Следователно цялото време за проверка на предложено решение е $O(kn)$. Остава да намерим полиномиална горна граница на k като функция на n и m. Тази граница участва в сравнението от стъпка 1. Границата не бива да бъде твърде голяма (например експоненциална), защото тогава проверката на предложено решение ще бъде бавна. Границата не бива да бъде и твърде малка, защото няма да съществува достатъчно къс сертификат.

Нямаме право да вземем $k \le n$, защото може да се наложи миньорът да повтори връх. *Пример:*



Ребрата от примера са двупосочни (всяко ребро на чертежа е двойка от противопосочни ребра); начертани са без стрелки, за да не се претрупва картинката. Във всеки връх на графа има смарагд. Началният връх е s. Очевидно е, че има единствен начин миньорът да вземе всички смарагди, без да прекъсне октава: това е маршрутът $sabcdefgfedcbahpqrtuvw$. Ето защо има случаи, когато миньорът е принуден да повтори върхове.

Следователно, ако в стъпка 1 на алгоритъма за проверка на предложено решение отхвърлим сертификатите, за които $k > n$, то в някои случаи няма да съществува сертификат, който да бъде приет от алгоритъма. Затова се налага да вземем по-голяма горна граница.

Нека в даден граф е възможно миньорът да вземе всички смарагди, без да прекъсне октава. Нека k е броят върхове на най-късия маршрут, по който миньорът може да стори това. Не е нужно да знаем точната стойност на k: достатъчно е да посочим горна граница.

Видяхме, че на миньора може да се наложи да повтори някои върхове. Възможно ли е обаче да има повторение на върхове в частта от маршрута, която започва от началния връх s и завършва с края на първата октава? Тази част се състои от два отрязъка: вторият съдържа осемте върха на октавата, а първият — останалите върхове. Върховете от втория отрязък съдържат смарагди, а върховете от първия отрязък — не (в противен случай маршрутът би прекъснал октава или споменатата по-горе октава не би била първа). Следователно двата отрязъка нямат общ връх. Вторият отрязък (със смарагдите) сам по себе си не повтаря връх: при повторно преминаване върхът ще се окаже празен (смарагдът е взет при първото преминаване) и октавата ще се прекъсне. А първият отрязък (той може да бъде и празен) сам по себе си също не повтаря връх, защото иначе ще съдържа цикъл, който може да се премахне от маршрута в противоречие с минималността му. Тоест двата отрязъка на тази част от маршрута на миньора нито имат общ връх, нито всеки от тях сам по себе си повтаря връх. Следователно цялата тази част от маршрута не съдържа повторения.

Втората част от маршрута — от върха след края на първата октава до края на втората октава включително — също не съдържа повторения сама по себе си, въпреки че може да повтаря връх от първата част на маршрута. Това се доказва с аналогични разсъждения.

Изобщо казано, маршрутът се разбива на няколко части, всяка от които започва от върха след края на поредната октава (или от s) и завършва с края на следващата октава. Две различни части от маршрута може да притежават общи върхове, но никоя част сама по себе си не съдържа повторения на върхове, затова се състои от не повече от n върха.

Броят на частите е равен на броя на октавите, тоест една осма от броя на смарагдите. Следва, че броят на частите на маршрута не надхвърля $n / 8$.

Тогава най-малкият брой k на върховете на маршрут, по който миньорът събира смарагдите, без да прекъсне октава, удовлетворява неравенството $k \le n \cdot n / 8 = n^2 / 8$. Тази горна граница трябва да бъде заложена в стъпка 1 от алгоритъма за проверка на предложено решение.

Щом $k \le n^2 / 8$, то $k = O(n^2)$. Заместваме в $O(kn)$ — времето за проверка на сертификат — и получаваме полиномиална времева сложност: $O(n^3)$. Следователно задача 2 е от класа NP. По-горе доказахме, че тя е NP-трудна. Ето защо тази задача е NP-пълна.

Задача 3. Че алгоритмичната задача *Опаковане на графи* е NP-трудна, се доказва с редукция от задачата *Изоморфизъм на подграфи:* по дадени неориентирани графи G_1 и G_2 с брой върхове съответно n_1 и n_2 и брой ребра съответно m_1 и m_2 да се разпознае дали графът G_1 съдържа подграф, изоморфен на графа G_2. Описваме редукцията на псевдокод:

```
Изоморфизъм на подграфи(G1,G2,n1,n2,m1,m2)
n ← n1; m ← m1
return Опаковане на графи(G1,G2,n1,n2,m1,m2,n,m)
```

Коректност на редукцията: Функцията *Изоморфизъм на подграфи*, дефинирана с кода, връща истина точно когато извикването на функцията *Опаковане на графи* връща истина, тоест когато има неориентиран граф G с не повече от n върха и не повече от m ребра, съдържащ подграф, изоморфен на G_1, и подграф, изоморфен на G_2. Тъй като $n = n_1$ и $m = m_1$, графът G трябва да съвпада с G_1. Изискването G да съдържа подграф, изоморфен на G_2, се превръща в изискване G_1 да съдържа подграф, изоморфен на G_2.

Доказахме, че функцията *Изоморфизъм на подграфи*, дефинирана с кода, връща истина точно когато графът G_1 съдържа подграф, изоморфен на G_2. Това съвпада с определението на алгоритмичната задача *Изоморфизъм на подграфи*, така че редукцията е коректна.

Бързина на редукцията: Присвояванията $n \leftarrow n_1$ и $m \leftarrow m_1$ общо изискват константно, тоест полиномиално, време.

Дотук доказахме, че задачата *Опаковане на графи* е NP-трудна. Остава да докажем, че тя принадлежи на класа NP. За целта съставяме алгоритъм с полиномиална времева сложност, проверяващ предложено решение на задачата. Освен параметрите G_1, G_2, n_1, n_2, m_1, m_2, n и m този алгоритъм ще има още един параметър — т. нар. сертификат. Сертификатът ще съдържа описание на G (списъци на ребрата, излизащи от върховете) и влагането на G_1 и G_2 в G, описано с два масива $A_1[1... n_1]$ и $A_2[1... n_2]$ от цели числа от 1 до n вкл. — номерата на върховете на G, съвпадащи със съответните върхове на G_1 и G_2 (индексите са номера на върхове на G_1 и G_2): $A_i[k]$ е номерът на върха на G, съвпадащ с k-тия връх на G_i, $i = 1, 2$.

Проверката на предложено решение се състои от следните стъпки:

1) За време $\Theta(n_1 + n_2)$ проверяваме дали масивите A_1 и A_2 съдържат само допустими стойности: 1, 2, 3, . . . , n. Ако намерим недопустима стойност, то сертификатът е невалиден (край на алгоритъма).
2) С помощта на сортиране чрез броене проверяваме за време $\Theta(n)$ дали A_1 и A_2 поотделно съдържат повторения. Ако това е така, сертификатът е невалиден (край на алгоритъма).
3) За всеки два върха от G_1, свързани с ребро, проверяваме дали съответните им върхове от G също са свързани с ребро. Ако поне една двойка върхове нарушава това изискване, то влагането на G_1 в G е неправилно и сертификатът е невалиден (край на алгоритъма); иначе отиваме на стъпка 4. Двойките върхове от G_1, свързани с ребро, са $\Theta(m_1)$ на брой. Обхождането им представлява обхождане на ребрата (и върховете) на G_1, което изисква време $\Theta(n_1 + m_1)$. Проверката на всяка двойка съдържа обхождане на списъците от ребра, излизащи от съответните два върха на G. Тези ребра са не повече от m за всеки връх, така че времето за проверка на всяка отделна двойка върхове е $O(m)$, а общо за всички двойки върхове — $O(m_1 m)$. Цялото време за стъпка 3 е $O(n_1 + m_1 + m_1 m) = O(n_1 + m_1 m)$.
4) Аналогично, за време $O(n_2 + m_2 m)$ проверяваме дали графът G_2 е вложен правилно в G.
5) За време $O(n + m)$ проверяваме дали G има не повече от n върха и не повече от m ребра.

Сертификатът е валиден, ако и само ако удовлетворява всички проверки. Редът им може да е различен, например петата проверка може да мине на първо място.

Входът на този алгоритъм има дължина $\Theta(n_1 + n_2 + n + m + m_1 + m_2)$, тоест линейна функция на участващите променливи. Времето на целия алгоритъм е $O(n_1 + n_2 + n + m + m_1 m + m_2 m)$, което е функция от втора степен на участващите променливи, а значи и на дължината на входа. Това означава, че алгоритъмът има квадратична, следователно полиномиална времева сложност. Затова задачата *Опаковане на графи* принадлежи на класа NP.

По-горе беше доказано, че алгоритмичната задача *Опаковане на графи* е NP-трудна. Щом е NP-трудна и принадлежи на NP, то тази задача е NP-пълна.

СХЕМА ЗА ТОЧКУВАНЕ НА ДОМАШНОТО

Задача 1 носи 2 точки, от които:
— за съставяне на полиномиален алгоритъм: 1 точка;
— за анализ на времевата сложност на алгоритъма: 1 точка.

Ако е пропуснат изводът, че класът на сложност е P, но всичко друго е правилно, се дава 1 т. за цялото решение (тоест отнема се 1 т. за пропуска).

Ако отговорът е грешен (посочен е друг клас на сложност, не P) или алгоритъмът е грешен, или времевата сложност на алгоритъма не е полиномиална в най-лошия случай — 0 точки.

Задачи 2 и 3 носят по 4 точки всяка. Тези точки се разпределят така:
— за доказателство, че алгоритмичната задача е от NP: 2 точки;
— за доказателство, че алгоритмичната задача е NP-трудна: 2 точки.

Ако е пропуснат изводът, че задачата е NP-пълна, но всичко друго е правилно, се дават 3 т. за цялото решение (тоест отнема се 1 т. за пропуска).

Ако задачата е класифицирана грешно — 0 точки за цялото решение.

Двете точки за първия етап (доказателството, че задачата е от класа NP) се разпределят така:
— за съставяне на полиномиален алгоритъм: 1 точка;
— за анализ на времевата сложност на алгоритъма: 1 точка.

Ако предложеният алгоритъм е грешен или неговата времева сложност не е полиномиална в най-лошия случай — 0 точки за целия етап.

Двете точки за втория етап (доказателството, че задачата е NP-трудна) се разпределят така:
— за полиномиална редукция, описана на псевдокод и с доказана коректност: 1 точка;
— за анализ на бързината на редукцията: 1 точка.

Ако редукцията е грешна в какъвто и да било смисъл (представлява некоректен алгоритъм или има грешна посока, или не е полиномиална, или изхожда от задача, която не е NP-трудна) — 0 точки за целия етап.

Ако редукцията не е описана на псевдокод или описанието е неразбираемо — 0 точки за целия етап.

Ако не е доказана коректността на редукцията, но няма други пропуски от този етап, се дава само втората точка от етапа (за анализ на времевата сложност на алгоритъма).

Точката за анализ на времевата сложност на алгоритъма за проверка на предложено решение на задача 2 се дава за разсъждения, с които е доказана горна граница на времето за проверка — полином на n, m и k. Не се изисква горна граница на k като функция на n и m. Ако такава граница е доказана (тоест времето на алгоритъма за проверка е ограничено отгоре от полином на n и m), се дават **допълнителни 3 точки.** В този случай сборът от точките за цялото домашно може да е по-голям от 10 т.