Алгоритмична неподатливост

Деф. (Задача за разпознаване**)**

Задача за разпознаване (ЗЗР) ще наричаме задача, която има точно два изхода - "*да"* и *"не"*.

Деф. (локална)

Със {ЗЗР} ще означаме класът съдържащ всички ЗЗР

Деф. (класът от ЗЗР P**)**

P = {ЗЗР за които има детерминиран алгоритъм с полиномиална времева сложност при най–лоши входни данни и отговора *"да"*}

Деф. (класът от ЗЗР NP**)**

NP = {ЗЗР за които има недетерминиран алгоритъм с полиномиална времева сложност при най–лоши входни данни и отговор *"да"*}

Забележка При отговор *"не"* може дори да зацикля недетерминирания алгоритъм! Интересуваме се да работи (и при това полиномиално) при отговор *"да"*.

Деф. (класът от ЗЗР NP - алтернативна дефиниция**)**

NP = {задачи за разпознаване за които при отговор *"да"* има сертификат с полиномиална дължина и има
алгоритъм с полиномиална времева сложност, който проверява отговора *"да"* с помощта на сертификата}

Деф. (класъв от ЗЗР co–NP**)**

co–NP = $\{A \in \{\text{ЗЗР}\} \mid \overline{A} \in \text{NP}\}$

Наблюдение

co–NP = {ЗЗР за които има недет. алгоритъм с полиномиална времева сложност при най–лоши входни данни и отговор *"не"*} =
= {задачи за разпознаване за които при отговор *"не"* има сертификат с полиномиална дължина и има
алгоритъм с полиномиална времева сложност, който проверява отговора *"не"* с помощта на сертификата}

Деф. (m-сводимост**)**

Ще казваме, че ЗЗР A е m-сводима към ЗЗР B, ако има тотална изчислима функция f : ∀ вход x на A е изпълнено :
· $f(x)$ е вход за B
· изходите на A и B за входове съответно x и $f(x)$ съвпадат
Ще означаваме с $A \leq_m B$.

Забележка Казва се m-сводимост, защото функцията f ни е от типа "many to one", т.е не е инективна. Има и други типове сводимости, но няма да се занимаваме с тях в текущия курс.

Деф. (m-сводимост за полиномиално време)

Ще казваме, че ЗЗР A е m-сводима за полиномиално време към ЗЗР B, ако има тотална изчислима функция f : ∀ вход x на A е изп. :
· $f(x)$ е вход за B
· изходите на A и B за входове съответно x и $f(x)$ съвпадат
· $f(x)$ се изчислява чрез детерминиран алгоритъм за полиномиално време
Ще означаваме с $A \leq_m^p B$.

Забележка m-сводимост и m-сводимост за полиномиално време имат и обобщена дефиниция (не само за ЗЗР), но в текущия курс няма да са ни необходими.

Деф. (класът от ЗЗР NP–hard**)**

NP–hard = $\{A \in \{ЗЗР\} \mid \forall\, B \in \text{NP}$ е изпълнено : $B \leq_m^p A\}$

Деф. (класът от ЗЗР NP–complete**)**

NP–complete = $\{A \in \{ЗЗР\} \mid A \in \text{NP}\ \&\ \forall\, B \in \text{NP}$ е изпълнено : $B \leq_m^p A\}$

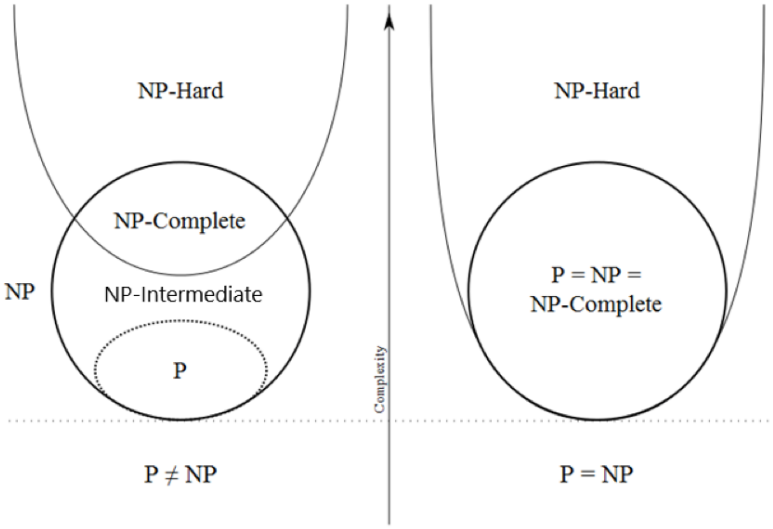
Твърдение Нека $A \in \{ЗЗР\}$. Тогава $A \in$ NP–complete $\leftrightarrow$ ($A \in$ NP $\&\ A \in$ NP–hard).

Следствие Нека $A \in \{ЗЗР\}$. Тогава $A \in$ NP–complete $\rightarrow A \in$ NP–hard.

Деф. (класът от ЗЗР NP–Intermediate**)**

NP–Intermediate = $\{A \in \{ЗЗР\} \mid A \in \text{NP}\ \&\ A \notin$ NP–hard $\&\ A \notin P\}$

Диаграма на описаните по-горе класове:



Примери за доказване на NP–Complete задачи:

(1) SAT (satisfiability problem) ∈ NP–Complete
{ Вход : булева формула в КНФ
{ Изход : има ли оценка на променливите, такава че да се удволетворява входната формула ?

Нека да разгледаме примерен вход и изход:

{ Вход : $(x_1 \vee x_2 \vee x_3 \vee x_4) \wedge (\overline{x_1} \vee x_2 \vee \overline{x_5} \vee x_8) \wedge (\overline{x_1} \vee \overline{x_2}) \wedge x_6 \wedge x_7$
{ Изход : да // $x_1 = T, x_2 = F, x_3 = F, x_4 = T, x_5 = F, x_6 = T, x_7 = T, x_8 = F$

Тук примерен сертификат е следния: $\langle x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8\rangle = \langle T, F, F, T, F, T, T, F\rangle$ има и други, но ние търсим поне един.

Th (Cook-Levin)

SAT е NP–complete

Твърдение Нека $C \in$ NP–hard. Нека $D \in \{ЗЗР\}$. Тогава ако $C \leq_m^p D$, то $D \in$ NP–hard.

Забележка SAT е първата известна NP–complete задача (и дори първата NP–hard). Не е изобщо тривиално да се доказва, че всички задачи от даден клас (в случая NP) могат да се m–сведат за полиномиално време до дадена задача. Много по-лесно е да имаме вече известна за нас NP–hard задача и да m–свеждаме за полиномиално време нашата задача до вече известната. Оттам ще знаем, че всяка задача може да се m–сведе за полиномиално време до нашата задача от транзитивността на $\leq_m^p$. Ще разгледаме пример директно отдолу и още примери по-нататък.

(2)3–SAT ∈ NP–complete
{ Вход : булева формула в КНФ, такава че всяка нейна клауза е с точно 3 литерала
{ Изход : има ли оценка на променливите, такава че да се удволетвори входната формула?

{ Вход : $(x_1 \vee x_2 \vee x_4) \wedge (x_3 \vee \overline{x_4} \vee x_6) \wedge (x_3 \wedge x_5 \wedge \overline{x_6})$
{ Изход : да // $x_1 = T,\ x_2 = F,\ x_3 = T,\ x_4 = F,\ x_5 = F,\ x_6 = F$ - тук просто клаузите са от по точно 3 литерала.

Ясно е, че 3–SAT е частен случай на SAT. От това обаче **не** следва, че 3–SAT ∈ NP–complete. Сега ще го докажем чрез m-свеждане за полиномиално време. Тоест ще докажем, че SAT $\leq_m^p$ 3–SAT.

Зад. 1

Докажете, че 3–SAT ∈ NP–complete. Може да използвате наготово факта, че SAT ∈ NP–complete.

Решение:

За да докажем, че 3–SAT ∈ NP–complete, ще докажем:
1) 3–SAT ∈ NP
2) 3–SAT ∈ NP–hard

2) 3–SAT ∈ NP–hard

SAT ∈ NP–complete ⇒ SAT ∈ NP–hard. Ще докажем, че SAT $\leq_m^p$ 3–SAT, откъдето ще следва, че 3–SAT ∈ NP–hard.

Нека $x = x_1 \wedge x_2 \wedge x_3 \wedge \ldots \wedge x_n$ е вход за SAT. Тоест x_i е дизюнкция от литерали: $x_i = x_{i,1} \vee x_{i,2} \vee \ldots \vee x_{i,n_i}$, $i \in \{1, \ldots, n\}$. Тоест общия вид на x е следния: $x = (x_{1,1} \vee x_{1,2} \vee \ldots \vee x_{1,n_1}) \wedge (x_{2,1} \vee \ldots \vee x_{2,n_2}) \wedge \ldots \wedge (x_{n,1} \vee \ldots \vee x_{n,n_n})$. Искаме да докажем, че SAT $\leq_m^p$ 3–SAT. Гледаме дефиницията на m-сводимост за полиномиално време. Трябва да конструираме изчислима тотална функция f, такава че:

- $f(x)$ е вход за 3–SAT
- изходите на SAT и 3–SAT за входове съответно x и $f(x)$ съвпадат
- $f(x)$ се изчислява чрез детерминиран алгоритъм за полиномиално време

Стъпка 1:

Нека дефинираме функция f_1, която приема вход $x = x_1 \wedge \ldots x_n = (x_{1,1} \vee \ldots \vee x_{1,n_1}) \wedge (x_{2,1} \vee \ldots \vee x_{2,n_2}) \wedge \ldots \wedge (x_{n,1} \vee \ldots \vee x_{n,n_n})$ за SAT. Нека да разгледаме всички клаузи на x. Тоест да разгледаме $x_1, x_2, \ldots, x_n$. Ако има клауза с един или два литерала, т.е $n_i \in \{1, 2\}$, то просто копираме първия литерал. Тоест ако $x = (x_{1,1}) \wedge (x_{2,1} \vee x_{2,2}) \wedge (x_{3,1} \vee x_{3,2} \vee x_{3,3})$, то $f_1(x) = (x_{1,1} \vee x_{1,1} \vee x_{1,1}) \wedge (x_{2,1} \vee x_{2,2} \vee x_{2,1}) \wedge (x_{3,1} \vee x_{3,2} \vee x_{3,3})$. Ясно е, че това може да го направим за линейно време (т.е. за полиномиално).

Стъпка 2:

Нека дефинираме функцията f_2, която приема вход $x = x_1 \wedge \ldots \wedge x_n = (x_{1,1} \vee \ldots \vee x_{1,n_1}) \wedge (x_{2,1} \vee \ldots \vee x_{2,n_2}) \wedge \ldots \wedge (x_{n,1} \vee \ldots \vee x_{n,n_n})$ с ограничението, че $\forall\, i \in \{1, \ldots, n\} : n_i \geq 3$. Нека вземем една произволна клауза x_i с да кажем 6 литерала за илюстрация на идеята, т.е. $x_i = (x_{i,1} \vee x_{i,2} \vee x_{i,3} \vee x_{i,4} \vee x_{i,5} \vee x_{i,6})$. Тогава $f_2(x_i) = (x_{i,1} \vee x_{i,2} \vee t_{i,1}) \wedge (\overline{t_{i,1}} \vee x_{i,3} \vee t_{i,2}) \wedge (\overline{t_{i,2}} \vee x_{i,4} \vee t_{i,3}) \wedge (\overline{t_{i,3}} \vee x_{i,5} \vee x_{i,6})$, където $t_{i,1}$, $t_{i,2}$ и $t_{i,3}$ са нови променливи. Тази идея можем да я разширим за конюнкция от дизюнкции по очевиден начин. Ясно е, че можем да конструираме f_2 за линейно време (т.е. за полиномиално).

Стъпка 3:

Нека x е вход за SAT. Нека $f = f_2 \circ f_1$. Тогава е ясно, че:

- $f(x)$ е вход за 3–SAT
- изходите на SAT и 3–SAT за входове съответно x и $f(x)$ съвпадат (убедете се)
- $f(x)$ се изчислява чрез детерминиран алгоритъм за полиномиално време (даже линейно)

Оттук заключваме, че SAT $\leq_m^p$ 3–SAT $\xrightarrow{\text{SAT}\in\text{NP–hard}}$ 3–SAT $\in$ NP–hard.

1) 3–SAT ∈ NP

Има два основни начина да се докаже, че дадена задача е NP. Ще ги разгледаме като 1.1) и 1.2). **НЕ** е необходимо да пишете и двата. Изберете си предпочитан метод и използвайте него.

1.1) 3–SAT ∈ NP (чрез недетерминиран алгоритъм с полиномиална времева сложност)

Това е по-формалния начин, но изисква нещо с което не сте се сблъсквали досега - да пишете недетерминирани алгоритми. Недетерминираните алгоритми ще ги пишем формално, чрез псевдокод. Те са аналогични на детерминираните, с разликата, че един-два реда от програмата ще са *"специални"* (т.е. ще правим недетерминиран избор там). Нека да разгледаме как би изглеждал недетерминиран алгоритъм с полиномиална времева сложност за текущата задача:

```
1. 3–SAT–ND–poly(x) : // x = (x_{1,1} ∨ x_{1,2} ∨ x_{1,3}) ∧ … ∧ (x_{n,1} ∨ x_{n,2} ∨ x_{n,3}) − вход за 3–SAT
2.     atoms ← extractAtoms(x) // примерно (p ∨ q ∨ ¬ r) ∧ (¬ p ∨ ¬ s ∨ t) ⟼ {p, q, r, s, t}
3.     for each atom ∈ atoms do
4.         atom ← choose({FALSE, TRUE}) // non deterministically
5.     for i ← 1 to n
6.         f ← FALSE
7.         for j ← 1 to 3 // проверяваме дали поне един от трите литерала е истина за конкретната оценка на атомите
8.             if checkLiteral(x[i][j], atoms) then
9.                 f ← TRUE
10.        if f = FALSE then
11.            return FALSE
12.    return TRUE
```

Може да забележим, че псевдокода изглжда напълно нормално - т.е. всеки ред е детерминиран, освен ред 4. Разлика е, че аналогично на недетерминираните автомати, за даден вход връща TRUE, ако **поне един** клон на изпълнение ни върне TRUE. Иначе връща FALSE.

1.2) 3–SAT ∈ NP (използвайки полиномиално голям сертификат и дет. алг. с полиномиална времева сложност)

Като цяло правим същото нещо, но от небето ни е паднал сертификата (приемаме го като аргумент на функцията):

```
1. 3–SAT–certificate–poly(x, atoms) : // x − вход за 3–SAT, atoms − сертификат (оценки на атомите от x)
2.     for i ← 1 to n
3.         f ← FALSE
4.         for j ← 1 to 3 // проверяваме дали поне един от трите литерала е истина за конкретната оценка на атомите
5.             if checkLiteral(x[i][j], atoms) then
6.                 f ← TRUE
7.         if f = FALSE then
8.             return FALSE
9.     return TRUE
```

Забележка Както виждате е абсолютно аналогично на горното, но не сме оказали какво е точно atoms. На контролни/домашни трябва да е абсолютно недвусмислено какво представлява сертификата. В **1.1)** се разбира недвусмислено, доколкото в **1.2)** не толкова - ще е добре да добавите бонус обяснения към псевдокода за какво представлява сертификата.

Забележка Най-вероятно 3–SAT ∉ co–NP. Защо?

(3) *K*–Clique ∈ NP–Complete
{ Вход : $G = \langle V, E\rangle$ − неор. граф; $k \in \mathbb{N}$
{ Изход : Има ли клика в G с размер точно k?

Зад. 2

Докажете, че K–Clique $\in$ NP–complete. Може да използвате наготово факта, че 3–SAT $\in$ NP–complete.

1) K–Clique $\in$ NP

```
1. K–Clique(G = ⟨V, E⟩, k) : // V = {v_1, …, v_n}
2.     V' ← {}
3.     for i ← 1 to n
4.         if choose({TRUE, FALSE}) then
5.             V' ← V' ⋃ {v_i}
6.     if | V' | ≠ k then
7.         return FALSE
8.     for each v_i, v_j ∈ V' : i < j do
9.         if {v_i, v_j} ∉ E then
10.            return FALSE
11.    return TRUE
```

Коректност Трябва да докажем: $\begin{cases} \text{поне едно изпълнение на функцията връща TRUE} & \text{, има клика с размер } k \\ \text{всички изпълнения на функцията връщат FALSE} & \text{, няма клика с размер } k \end{cases}$.

· Нека има клика с размер k. Нека върховете на тази клика са $V'' = \{v_{i_1}, \ldots, v_{i_k}\}$, $i_1 < \ldots < i_k$. Трябва да покажем клон на изпълнение, който връща TRUE. Лесно се проверява, че в клона в който недетерминирания избор choose({TRUE, FALSE}) ни е избрал: $\begin{cases} \text{TRUE} & \text{, } i \in \{i_1, \ldots, i_k\} \\ \text{FALSE} & \text{, else} \end{cases}$, то функцията ще върне TRUE.

· Нека няма клика с размер k. Трябва да покажем, че функцията винаги връща FALSE. Ясно е, че функцията може да върне TRUE само ако $| V' | = k$. Нека допуснем, че функцията е върнала TRUE за някой избор Б.О.О. $V' = \{v_1, \ldots, v_k\}$. Тогава за да е върнали TRUE, то трябва да има ребро между всички върхове от V', което е точно деф. на клика. Противоречие.

Сложност $\theta(1) + \theta(n) + \theta(1) + \theta\left(k^2 * \textit{полиномиална проверка за принадлежност на ребро към } E\right)$. Тоест полиномиална сложност.

2) K–Clique $\in$ NP–hard

3–SAT $\in$ NP–complete $\Rightarrow$ 3–SAT $\in$ NP–hard. Ще докажем, че 3–SAT $\leq_m^p$ K–Clique, откъдето ще следва, че K–Clique $\in$ NP–hard.

Нека $x = c_1 \wedge c_2 \wedge c_3 \wedge \ldots \wedge c_n$ е вход за 3–SAT. Тоест x_i е дизюнкция от точно 3 литерали: $c_i = l_{i,1} \vee l_{i,2} \vee l_{i,3}$, $i \in \{1, \ldots, n\}$. Тоест общия вид на x е следния: $x = (l_{1,1} \vee l_{1,2} \vee l_{i,3}) \wedge (l_{2,1} \vee l_{2,2} \vee l_{2,3}) \wedge \ldots \wedge (l_{n,1} \vee l_{n,2} \vee l_{n,3})$. Искаме да докажем, че 3–SAT $\leq_m^p$ K–Clique.

Гледаме дефиницията на m-сводимост за полиномиално време. Трябва да конструираме изчислима тотална функция f, такава че:

· $f(x)$ е вход за K–Clique
· изходите на 3–SAT и K–Clique за входове съответно x и $f(x)$ съвпадат
· $f(x)$ се изчислява чрез детерминиран алгоритъм за полиномиално време

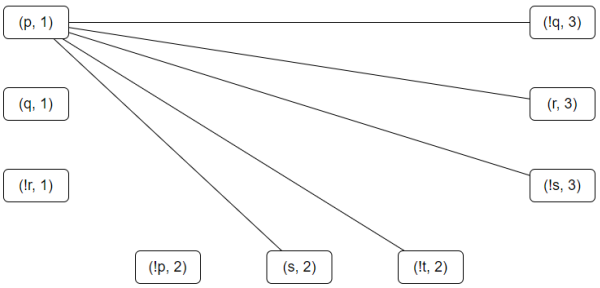
Искаме $f(x)$ да ни върне наредена двойка от неориентиран граф $G = \langle V, E \rangle$ и естествено число k.

Стъпка 1:

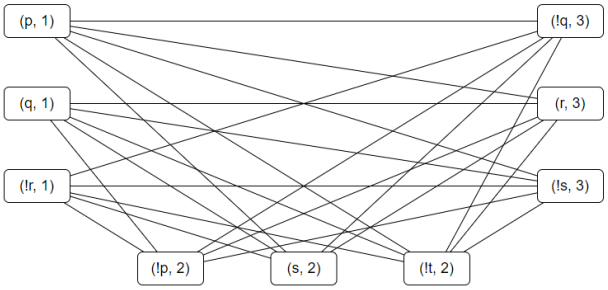
Ще съставим функция f_V, която по вход $x = c_1 \wedge \ldots \wedge c_n = (l_{1,1} \vee l_{1,2} \vee l_{i,3}) \wedge (l_{2,1} \vee l_{2,2} \vee l_{2,3}) \wedge \ldots \wedge (l_{n,1} \vee l_{n,2} \vee l_{n,3})$ за 3–SAT ни връща множеството V. За фиксираното x имаме: $V = f_V(x) = \{\langle l_{i,j}, i \rangle \mid i \in \{1, \ldots, n\}, j \in \{1, 2, 3\}\}$. Тоест за конкретно $x = (p \vee q \vee \neg r) \wedge (\neg p \vee s \vee \neg t) \wedge (\neg q \vee r \vee \neg s)$ ще имаме $f_V(x) = \{\langle p, 1\rangle, \langle q, 1\rangle, \langle \neg r, 1\rangle, \langle \neg p, 2\rangle, \langle s, 2\rangle, \langle \neg t, 2\rangle, \langle \neg q, 3\rangle, \langle r, 3\rangle, \langle \neg s, 3\rangle\}$.

Стъпка 2:

Ще съставим функция f_E, която по вход $x = c_1 \wedge \ldots \wedge c_n = (l_{1,1} \vee l_{1,2} \vee l_{i,3}) \wedge (l_{2,1} \vee l_{2,2} \vee l_{2,3}) \wedge \ldots \wedge (l_{n,1} \vee l_{n,2} \vee l_{n,3})$ за 3–SAT ни връща множеството E. За фиксираното x имаме: $E = f_E(x) = \{\{\langle l_1, j_1\rangle, \langle l_2, j_2\rangle\} \subseteq V \mid 1 \leq j_1 < j_2 \leq n \;\&\; l_1 \neq \neg\, l_2\}$. Тоест за конкретно $x = (p \vee q \vee \neg r) \wedge (\neg p \vee s \vee \neg t) \wedge (\neg q \vee r \vee \neg s)$ ще имаме (с картинка за по-четимо):



Всеки връх (засега само един) го свързваме със всички върхове с различна втора координата и с не противоположна първа. Довършваме за останалите върхове:



Стъпка 3:

Нека $x = c_1 \wedge \ldots \wedge c_n$. Тогава $f(x) = \langle\langle f_V(x), f_E(x)\rangle, n\rangle$. Да не забравяме, че K–Clique има два параметъра - граф и размер на клика. Сега трябва да докажем следното:

Твърдение (локално)

Нека $x = c_1 \wedge \ldots \wedge c_n$ е вход за 3–SAT. Тогава изходите на 3–SAT и K–Clique за вход съответно x и $f(x)$ съвпадат.

Доказателство

($\Rightarrow$)

Нека изхода на 3–SAT за вход x е TRUE. Ще покажем, че изхода на K–Clique за вход $f(x) = \langle G = \langle f_V(x), f_E(x)\rangle, n\rangle$ е TRUE, където n е броя на клаузите в $x = c_1 \wedge \ldots \wedge c_n$. Тъй като x = TRUE, то от всяка клауза има поне по един литерал, който е истина. Нека Б.О.О. това са $L = \{l_1, \ldots, l_n\}$. Знаем, че всички литерали от L са истина $\Rightarrow$ няма контрарни литерали. Също така всички литерали са от различни клаузи $\xrightarrow{\text{от деф. на } f_V \text{ и } f_E}$ има клика с размер n. (Разбира се може мнооого по-детайлно доказателство)

($\Leftarrow$)

Нека изхода на K–Clique за вход $f(x) = \langle G = \langle f_V(x), f_E(x)\rangle, n\rangle$ е TRUE, където n е броя на клаузите в $x = c_1 \wedge \ldots \wedge c_n$. Ще докажем, че изхода на 3–SAT за вход x е TRUE. Довършете за упражнение (аналогично е на правата посока). □

Остана да покажем 3-тото изискване за функцията f и по-конкретно, че тя се изчислява чрез детерминиран алгоритъм за полиномиално време. Ще дадем кратка обосновка за сложността на f.

Ясно е, че V го съставяме за линейно време спрямо големината на входа. От друга страна E го съставяме за квадратично време спрямо големината на входа. Оттук следва, че в най-лошия случай имаме квадратично време спрямо големината на входа. Също така алг. за съставяне на V и E са очевидно детерминирани. Тоест има дет. полиномиален алгоритъм за f.

Тоест доказахме, че имаме f - тотална изчислима функция която за всеки вход x на 3–SAT:

- $f(x)$ е вход за K–Clique
- изходите на 3–SAT и K–Clique за входове съответно x и $f(x)$ съвпадат
- $f(x)$ се изчислява чрез детерминиран алгоритъм за полиномиално време (даже квадратично)

Оттук заключваме, че 3–SAT $\leq_m^p$ K–Clique $\xrightarrow{3\text{–SAT} \in \text{NP–hard}}$ K–Clique $\in$ NP–hard.

(4) K–VertexCover ∈ NP–Complete
$\begin{cases} \text{Вход : } G = \langle V, E\rangle - \text{неор. граф; } k \in \mathbb{N} \\ \text{Изход : има ли } V' \subseteq V : \left(|V'| = k \;\&\; \forall \{u, v\} \in E \text{ е изп. } \{u, v\} \cap V' \neq \emptyset \right)? \end{cases}$

Зад. 3

Докажете, че K–VertexCover ∈ NP–complete. Може да използвате наготово факта, че K–Clique ∈ NP–complete.

1) K–VertexCover ∈ NP

```
1. K–VertexCover(G = ⟨V, E⟩, k) : // V = {v_1, …, v_n}
2.     V' ← {}
3.     for i ← 1 to n
4.         if choose({TRUE, FALSE}) then
5.             V' ← V' ⋃ {v_i}
6.     if |V'| ≠ k then
7.         return FALSE
8.     for each u ∈ V' do
9.         for each {u, v} ∈ E do
10.            E ← E\{u, v}
11.    return (E = ∅)
```

Коректност Трябва да докажем: $\begin{cases} \text{поне едно изпълнение на функцията връща TRUE} & \text{, има покритие с размер } k \\ \text{всички изпълнения на функцията връщат FALSE} & \text{, няма покритие с размер } k \end{cases}$. Упражнете се!

Сложност $\theta(1) + \theta(n) + \theta(1) + O\left(n^{2*} \textit{ полиномиална проверка за принадлежност на ребро към } E\right)$. Тоест полиномиална.

2) K–VertexCover ∈ NP–hard

K–Clique ∈ NP–complete ⇒ K–Clique ∈ NP–hard. Ще докажем, че K–Clique $\leq_m^p$ K–VertexCover, откъдето ще следва, че K–VertexCover ∈ NP–hard.

Нека $x = \langle G = \langle V, E\rangle, k\rangle$ е вход за K–Clique. Искаме да докажем, че 3–SAT $\leq_m^p$ K–Clique. Гледаме дефиницията на m-сводимост за полиномиално време. Трябва да конструираме изчислима тотална функция f, такава че:

- $f(x)$ е вход за K–VertexCover
- изходите на K–Clique и K–VertexCover за входове съответно x и $f(x)$ съвпадат
- $f(x)$ се изчислява чрез детерминиран алгоритъм за полиномиално време

Искаме $f(x)$ да ни върне наредена двойка от неориентиран граф $G = \langle V, E\rangle$ и естествено число k'.

Идея (Умни сме били сетили сме се):

Дефинираме $f(\langle\langle V, E\rangle, k\rangle) = \left\langle\left\langle V, \overline{E}\right\rangle, n - k\right\rangle$, където $n = |V|$.

Твърдение (локално)

Нека $x = \langle G = \langle V, E\rangle, k\rangle$ е вход за K–Clique. Тогава K–Clique(G, k) ↔ K–VertexCover$\left(\overline{G}, n - k\right)$, където $\overline{G} = \left\langle V, \overline{E}\right\rangle$ и $n = |V|$.

Доказателство

(⇒)

Нека K–Clique(G, k) = TRUE. Нека $V = \{v_1, \ldots, v_n\}$. Нека Б.О.О. k-кликата е $V' = \{v_1, \ldots, v_k\}$. Твърдим, че $V \setminus V' = \{v_{k+1}, \ldots, v_n\}$ е $(n - k)$-покритие за $\overline{G}$. Ясно е, че няма ребра между v_i, v_j, $i, j \in \{v_1, \ldots, v_k\}$ в $\overline{G}$. Тоест ако v_i, $i \in \{1, \ldots, k\}$ е свързан с ребро в $\overline{G}$, то той е свързан с връх v_j, $j \in \{k + 1, \ldots, n\}$. Но всички върхове v_j, $j \in \{k + 1, \ldots, n\}$ са покрити. Следователно всяко ребро с край от v_i, $i \in \{1, \ldots, k\}$ е покрито. Очевидно всяко ребро с край от v_j, $j \in \{k + 1, \ldots, n\}$ е покрито, тъй като сме покрили всички v_j, $j \in \{k + 1, \ldots, n\}$. Тоест всички ребра са покрити. (Под покрито ребро се разбира ребро, което поне единия му край е покрит връх).

(⇐)

Нека K–VertexCover$\left(\overline{G}, n - k\right)$ = TRUE. Нека $V = \{v_1, \ldots, v_n\}$. Нека Б.О.О. покритите върхове са $V' = \{v_{k+1}, \ldots, v_n\}$. Тогава е ясно,

че няма ребра между v_i и v_j за $i, j \in \{1, \ldots, k\}$. Тогава в $\overline{\overline{G}} = G$ ще има между всички върхове с индекси $\{1, \ldots, k\}$. Тоест в G ще имаме клика с размер k. □

Ясно е, че f изпълнява трите условия за m-сводимост за полиномиално време $\Rightarrow$ K–Clique $\leq_m^p$ K–VertexCover. Следователно K–VertexCover $\in$ NP–hard.

(5) SubsetSum ∈ NP–Complete
{ Вход : множество от естествени числа A; $k \in \mathbb{N}$
{ Изход : има ли $A' \subseteq A : (\sum A' = k)$

Зад. 4

Докажете, че SubsetSum ∈ NP–complete. Може да използвате наготово факта, че:

a) 3–SAT ∈ NP–Complete

b) K–VertexCover ∈ NP–Complete

1) SubsetSum ∈ NP

```
1. SubsetSum(A, k) : // A = {a_1, ..., a_n}
2.     A' ← {}
3.     for i ← 1 to n
4.         if choose({TRUE, FALSE}) then
5.             A' ← A' ⋃ {a_i}
6.     sum ← 0
7.     for each a ∈ A' do
8.         sum ← sum + a
9.     return (sum = k)
```

Коректност Трябва да докажем: { поне едно изпълнение на функцията връща TRUE , има такова разбиване / всички изпълнения на функцията връщат FALSE , няма такова разбиване . Упражнете се!

Сложност $\theta(1) + \theta(n) + \theta(1) + \theta(n)$. Тоест полиномиална.

2) SubsetSum ∈ NP–hard

a) За упражнение

b) За упражнение