

Летищни връзки

Проект по СДА-практикум, Андрей Стоев, ф.н. 62369

ПЪТНИЦИ БИЗНЕС ЛЕТИЩЕ СОФИЯ КАРИЕРИ ИНФОРМАЦИОННА ПОЛИТИКА



БЪЛГАРСКИ

ENGLISH



ПОЛЕТНА
ИНФОРМАЦИЯ



ОТ И ДО
ЛЕТИЩЕТО



ПОЛЕЗНА
ИНФОРМАЦИЯ



УСЛУГИ



КОНТАКТИ

КАЦАЩИ ИЗЛИТАЩИ ТЪРСЕНЕ ПО ДЕСТИНАЦИЯ АВИОКОМПАНИИ НОВИ ДЕСТИНАЦИИ



КАЦАЩИ



ИЗЛИТАЩИ

ЧАС	ОТ	ПОЛЕТ	ТЕРМИНАЛ	СТАТУС	
13:15	Франкфурт	LH1426	T2	Кацнал	▶
13:15	Франкфурт	AC9487	T2	Кацнал	▶
13:35	Лондон Лутън	W64302	T1	Очакван	▶
13:50	Киев Бориспол	7W755	T1	Очакван	▶
13:50	Киев Бориспол	FB1511	T1	Очакван	▶
14:00	Амстердам	FB462	T2	Очакван	▶

ЧАС	ДО	ПОЛЕТ	ТЕРМИНАЛ	СТАТУС	
10:50	Атина	A3981	T2	Излетял	▶
10:50	Атина	FB1507	T2	Излетял	▶
14:00	Франкфурт	LH1427	T2	Регистрация	▶
14:00	Франкфурт	AC9520	T2	Регистрация	▶
14:00	Франкфурт	ET1596	T2	Регистрация	▶
14:00	Франкфурт	SK3733	T2	Регистрация	▶

Коледа наближава и авиокомпания AlgoAir иска да предложи на клиентите си полет, който да ги отведе до вкъщи за празниците. За тази задача AlgoAir ни е наела, за да определим минималния брой еднопосочни полети, които трябва да предложи **допълнително**, за да може произволен клиент да стигне до всяко летище в даден списък от летища, започвайки от летище София.

Ще получим списък с летища (трибуковени кодове като „JFK“), списък с маршрути (единопосочни полети от едно летище до друго като [„JFK“, „SOF“]) и начално летище. От нас се иска да напишем програма, която връща минималния брой чартърни полети, които трябва да бъдат добавени, за да може произволен клиент да стигне до всяко летище в списъка, започвайки от началното летище. (В нашата имплементация дори ще добавим опция, която да извежда кои са тези полети - за удобство.)

Имайте предвид, че връзките не трябва да бъдат директни; добре ще е, ако до летището може да се стигне от началното летище, като е възможно първо да се спре на други летища. С други думи клиентите не ги интересува дали и колко прикачвания ще имат, а само това да имат възможност да се приберат във вкъщи за празниците от началното подадено летище (което в нашия случай ще е летище София).

Примерен списък на летища:

[DOH, DEL, ICN, HND, JFK, LGA, ORD, SAN, TLV, BOD, SIN, BUD, SOF]

(DOH-Hamad International Airport, Doha (Qatar), BUD-Budapest Liszt Ferenc, Budapest (Hungary), TLV-Ben Gurion Airport, Tel Aviv (Israel), JFK-John F. Kennedy International Airport и т.н. <https://airportcod.es/>)

Примерен списък на предлаганите от летището полети за даден ден:

[[HND, SOF], (има чартърен полет от Хондурас до София)
[SAN, BOD], (има чартърен полет от Сан Диего до Бодрум)
[ICN, HND],
[LGA, HND],
[BUD, SOF],
[TLV, SAN],
[DEL, DOH],
[HND, DEL],
[DOH, ICN],
[BOD, SAN],
[SIN, DEL],
[JFK, ICN],
[ORD, JFK],
[SAN, TLV],
[JFK, LGA],
[LGA, ORD],
[ORD, BUD],
[HND, SAN]]

(Този списък означава, че авиокомпанията има полет от първото до второто летище. За конкретния пример авиокомпанията може да лети **от HND до SOF** или **от ICN до HND** и т.н.)

Начално летище: **SOF**

Входните данни ще се четат от файл input.txt, а изходните ще се печатат на файл output.txt в удобен за преглед формат.

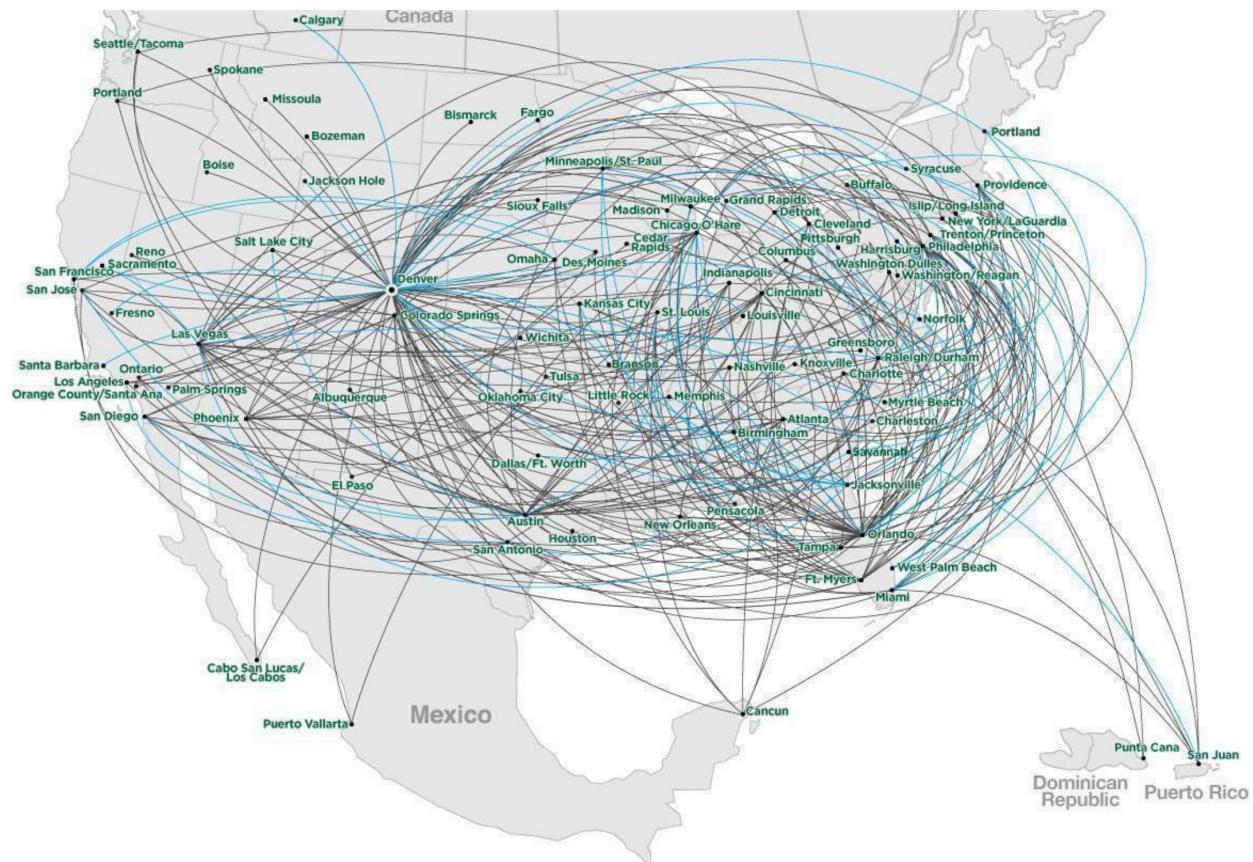
От примера разбираме, че AlgoAir **е възможно** дори **да не предлага** полети излитащи от София. Т.е. ние трябва да пресметнем колко **най-малко полети** ще са необходими да има от летище София до други дестинации, за да бъде изпълнено условието.

Забележка:

Забележете, че ако имаме две авиокомпании и обединим полетите им и дадем някакъв списък с летища, то лесно може да сведем задачата до такава с една авиокомпания.

Следователно по индукция, задачата може да се генерализира до *n* на брой авиокомпании заедно с техните полети и една стартова дестинация.

А вече много на брой авиокомпании биха могли да образуват много гъста пътна мрежа. В такъв случай трудно бихме се справили с намирането на летище до което не може да се стигне от стартово летище, а пък за оптимизирането на броя на необходимите полети е немислимо. Но ако например сте обикновен човек, който иска да се приbere в Шотландия за Коледа - веднага ще установите дали има или не път до всички (но ако няма?). За наша радост, компютър би могъл светковично да се справи с този казус, от нас се иска само да открием и реализираме бърз алгоритъм.

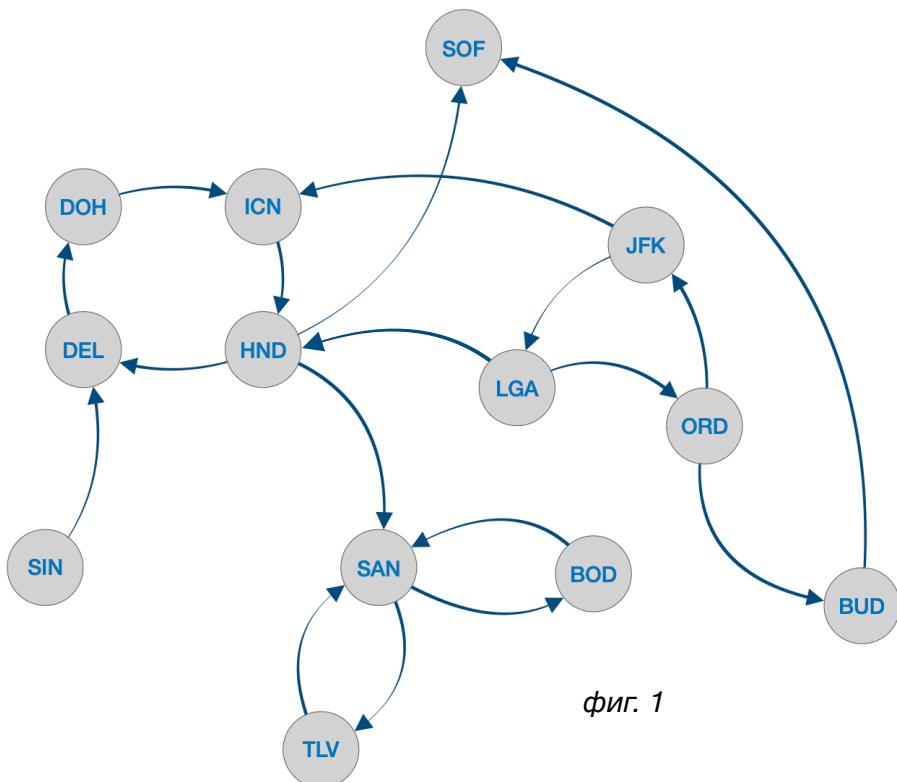


Тъй като имаме списък от еднопосочни връзки, то може да моделираме задачата чрез структурата от данни - **насочен мултиграф**.

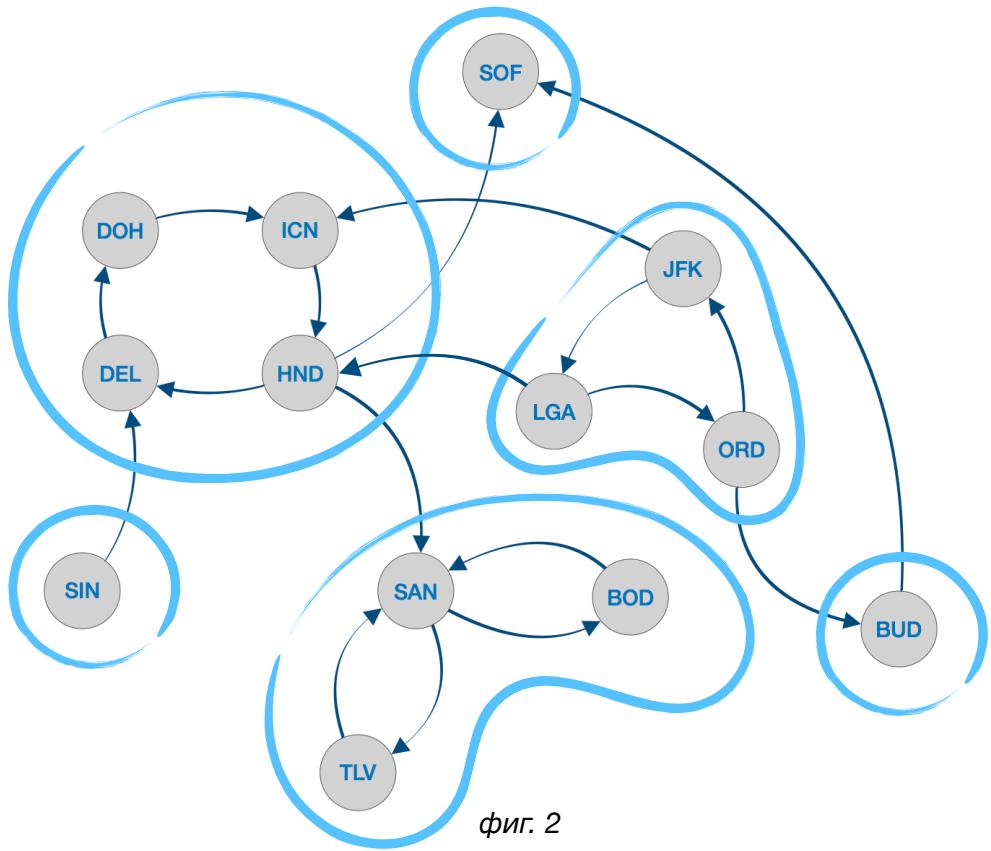
Дефиниция:

Насочен (мулти)граф се нарича **силно свързан**, ако има път във всяка посока между всяка двойка върхове на графа. Тоест, съществува път от първия връх в двойката към втория и друг път съществува от втория връх към първия.

Пример:

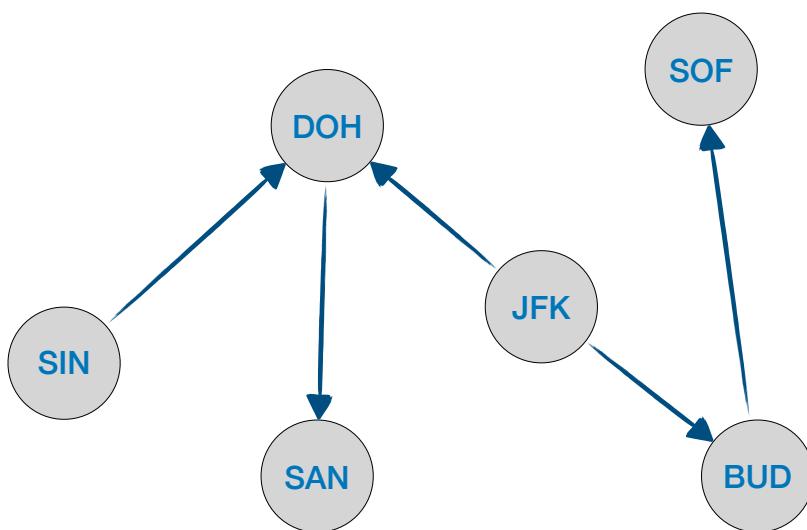


фиг. 1



фиг. 2

Компресиране на графа



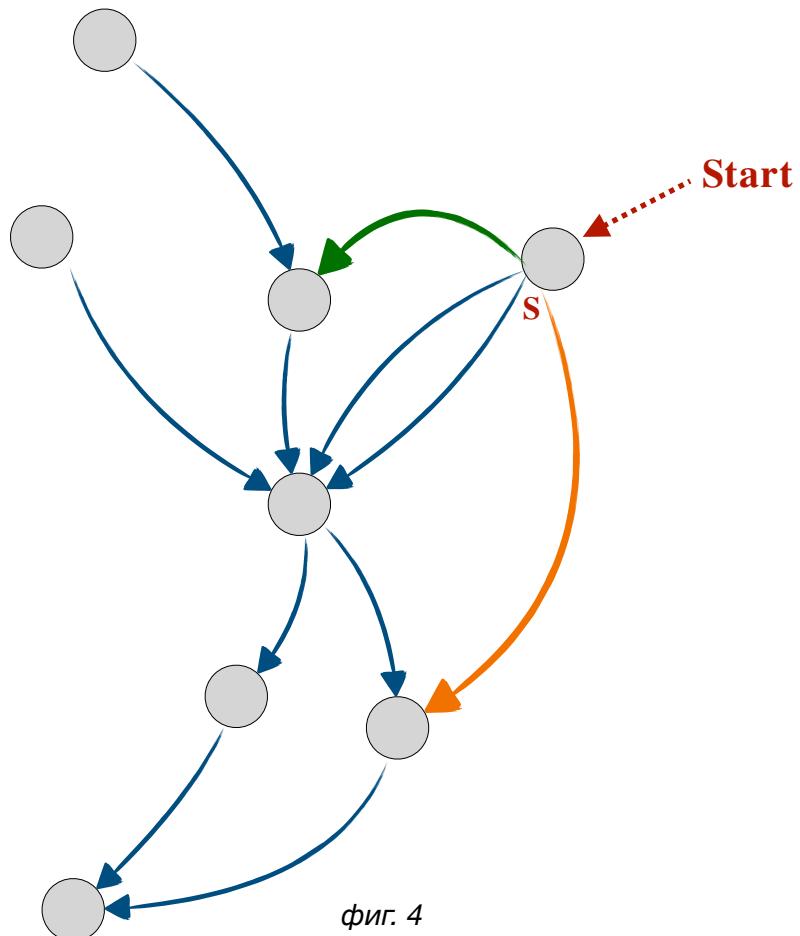
фиг. 3

Нека вземем произволен насочен граф (графа може дори да е разцепен на няколко компоненти на свързаност), например този от *фиг. 1*. Той репрезентира примера, но това не е съществено, тъй като разсъжденията, които ще правим ще са валидни за произволен насочен мулти граф.

Първото нещо което ще направим е да открием силно свързаните компоненти в графа. Те ще го разделят на няколко непресичащи се множества от върхове, тъй като ако допуснем че две силно свързани компоненти имат общ връх, то по дефиниция ще заключим, че това е една силно свързана компонента, което ще доведе до противоречие. Т.е. всеки връх ще участва в точно една силно свързана компонента от графа. Тогава, ако успеем да компресираме графа по такъв начин, че всяка силно свързана компонента да си има представител, то може да разглеждаме същата задача върху компенсирания граф, с печалбата че новия граф освен, че ще е с не-повече върхове от първоначалния ще е и **ацикличен** (*фиг. 2*) т.е.

ще е представен от една или няколко дърводидни структури. Това е така, защото ако допуснем че в компресирания граф има цикъл, то този цикъл ще образува компонента на свързаност и би трявало да се е представила с единичен връх след компресацията.

Нека например графа след компресирането изглежда като този на *фиг. 4* и нека с червено сме отбелязали примерна стартова позиция **S**. Ако добавим ребро от **S** към някой от върховете, който и преди това се е достигал от **S** би било безсмислено (оранжеви ребра). Но ако добавяме ребра от **S** към недостижими от **S** върхове, то тогава ще има смисъл (зелени ребра). Въпроса е към колко най-малко такива върха трябва да добавим ребра?



Нека дефинираме още две понятия: вътрешна и външна степен на връх. Вътрешната степен на връх е броя на ребрата, чрез които се достига до дадения връх от различен от него връх. Външната степен е дуална на вътрешната, но за изходящите ребра и за нашата задача тя няма да ни е необходима. Забележете, че ако някой връх има вътрешна степен равна на 0, то той е недостижим. Следователно ще е необходимо да достигнем само до върховете с вътрешна степен 0, тъй като останалите все от някъде ще се достигат. Това ще е и отговора на задачата: броя на върховете в компресирания граф, които са с вътрешна степен равна на 0.

Описание на алгоритъма за решаване на задачата:

Нека N е броя на върховете, а M броя на ребрата в графа.

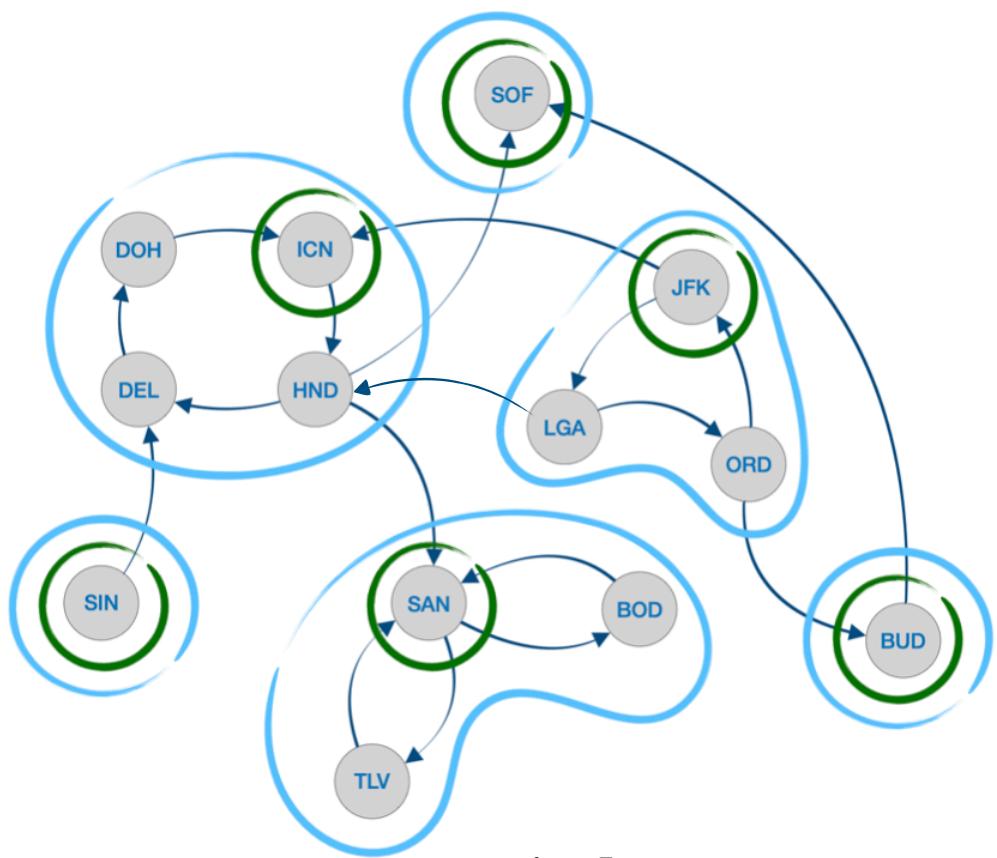
- Обработка на входа. Сложност $O(N + M)$
 - ще обходим списъка с летища и ще прикачим на всяко име **id** номер, който ще играе ролята на стойност за ключа - името. За целта ще използваме *unordered hash table*, тъй като не ни е обходима наредба а бързо намиране на стойност на даден ключ. След това ще създадем **списък на съседства** *vector < int > adj[n]* за съхраняване на чартерните полети и репрезентирането им като насочен мултиграф. Това е структура от данни, която за всеки връх съхранява върховете до които може да се достигне директно от този връх.
- Намираме силно свързаните компоненти на графа. $O(N + M)$
 - За тази част ще използваме алгоритъма на **Kosaraju-Sharir** за намиране на силно свързани компоненти. Реално този алгоритъм няма да ни даде нов списък на съседства с нови върхове и ребра, а просто ще **изльчи представител** за всяка компонента на свързаност в графа (фиг. 5 - *оградени със зелено върхове*). Това изльчване на представител ще бъде представено като масив *representativeAirports[]*.
- Компресираме графа на база на силно свързаните компоненти. $O(N + M)$
 - Тук вече ще използваме нов списък на съседства (имплицитно или експлицитно), който ще е от същата размерност както първоначалния списък (въпреки, че е възможно да е много по-разен). В него ще вземем предвид само тези върхове, които са представители в своята силно свързана компонента. Това ще са тези върхове, които отговарят на условието *representativeAirports[u] = u* за някой връх u . За ребра в този нов граф ще вземем само тези, които свързват два върха от различни компоненти на свързаност:

for each *edge(u, v) in adj do : adj2[re[u]].push(re[v])*

Забележете, че тук между две силно свързани компоненти може да има повече от едно ребро и ще ги преброим всички, което реално е дуплициране на ребра но не пречи на отговора и коректността на алгоритъма, нито на сложността му.

Тук също така ще се подгответим за финалната стъпка, като преброим вътрешните степени на всички върхове от новия списък на съседства (на силно свързаните компоненти) и това ще го направим още при създаването му.
- Намиране на броя на върховете с вътрешна степен равна на 0, които не са S в новия граф. $O(n)$
 - Преброяваме върховете, за които е изпълнено условието *representativeAirports[u] = u* и *inDegree[u] = 0* и *representativeAirports[u] ≠ S*.

Обща времева сложност $O(n + m)$. Сложност по памет, отново $O(n + m)$.



фиг. 5