## АНАЛИЗ

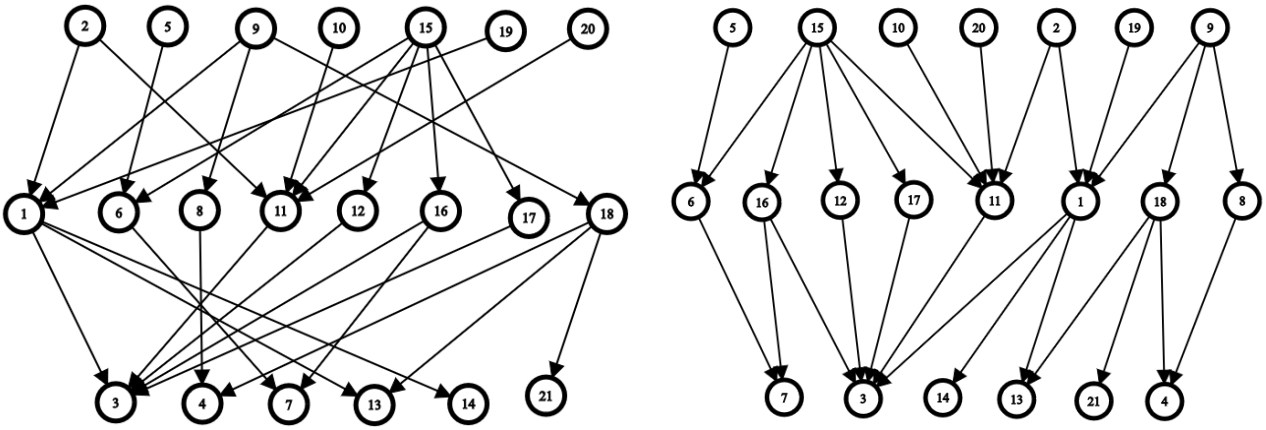
*Тагове: конструктивна задача, обхождания на графи, топологично сортиране, алчни стратегии, сортировки*

Формално в задачата е даден ориентиран ацикличен граф, в който върховете са камъните, изграждащи стената, и има ребро от един връх до друг, ако съществува изискване камъкът, представен чрез първия връх, да бъде разположен върху камъка, представен чрез втория. Освен това от начина, по който е дефиниран графът, следва, че той е планарен и върховете са разделени на нива – в първото ниво са върховете, в които не влизат ребра, а всяко следващо ниво включва върховете, към които има ребро от поне един връх в предишното ниво. Търсим подходяща наредба на върховете от всяко ниво, така че след като добавим ребрата върху схемата на графа, да няма двойка пресичащи се такива. Ще покажем как от всяка една такава наредба на върховете можем да конструираме стена с оптимално лице.

Първата стъпка от решението е да намерим всеки връх на кое ниво от графа се намира. Това може да стане с обхождане на графа в дълбочина или в широчина, като се започне от върховете от първото ниво. Значително по-трудно е да се намери една възможна наредба на камъните във всеки ред, която отговаря на условията. На следващите две схеми са показани:

**1)** граф, който е построен по реда на номерата на върховете в трите нива и съдържа взаимно пресичащи се ребра;

**2)** същия граф, в който върховете в трите нива са подредени по такъв начин, че да няма взаимно пресичащи се ребра.




Сега ще покажем алгоритъм за конструиране на такава наредба, в която на всеки връх ще поставим индекс, който да отговаря на поредността му сред върховете от неговото ниво. Върховете, които са практически неразличими (свързани са с едни и същи върхове в двете съседни нива), ще получават еднакви индекси. Примери за такива върхове са 10 и 20 от първото ниво, а също 12 и 17 от второто. За всяко ниво на графа от първото нататък изпълняване следните стъпки:

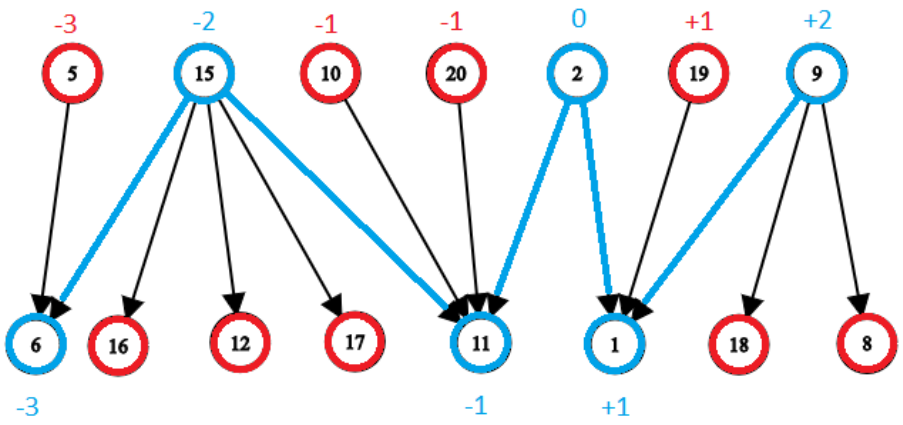
**1)** За всеки връх от текущото ниво образуваме двойки числа, състоящи се от най-малкия и най-големия индекс на връх в предишното ниво, от които има ребро към него. Ако предишно ниво няма (т.е. текущото е първото), считаме, че всички двойки са (0, 0). Сега, ако сортираме тези двойки в нарастващ ред (по приоритет на първото число и след това на второто), ще получим една предварителна наредба на върховете, базирана само на информацията, която имаме до момента от наредбите на върховете в предишните нива на графа. Ако няма следващо ниво (т.е. текущото е последно), тази наредба е достатъчна.

**2)** За да получим пълната наредба, ще трябва да вземем предвид и върховете от следващото ниво и ребрата към тях. За удобство ще изключим върховете, които имат само по едно инцидентно ребро, разположено между двете нива. Избираме си връх от първото ниво, който не сме изключили, и започваме обхождане от него, като се движим само по върхове от текущото и следващото ниво (забележете, че за целта на обхождането посоката на ребрата не ни интересува). В резултат на това обхождане получаваме верижка, в която началният връх се намира някъде в средата (възможно е да бъде и в края). Номерираме върховете от верижката спрямо позициите им – началният получава позиция 0, а тези в двете разклонения на верижката – последователни положителни и отрицателни числа, при което се взема предвид и предварителната наредба. С други думи не трябва да има два върха от текущото ниво, за които двойката индекси от предварителната наредба на първия е по-малка от тази на втория и същевременно той да получи по-голяма позиция от тази на втория. Изключените върхове копират позицията на единствения си съсед от следващото ниво.

**3)** Комбинираме двойките числа, с които получихме предварителната наредба, с позициите, получени след обхождането. Ако сортираме върховете по тези тройки (с приоритет на първото число, после второто и накрая третото), ще получим пълната наредба на върховете от текущото ниво.

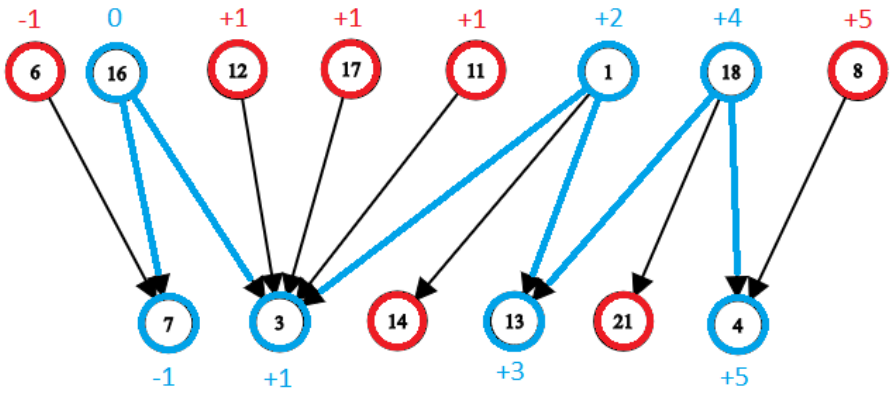
Сега, за да е по-ясно какво точно правим, ще приложим алгоритъма върху примера по-горе. Започваме с първото ниво:

| Номера на върховете от първо ниво | 2 | 5 | 9 | 10 | 15 | 19 | 20 |
|---|---|---|---|---|---|---|---|
| Двойки индекси от предишното ниво | 0; 0 | 0; 0 | 0; 0 | 0; 0 | 0; 0 | 0; 0 | 0; 0 |
| Изключени върхове (от първо ниво) | 5 | | 10 | | 19 | | 20 |
| Единствено инцидентно ребро | 5 → 6 | | 10 → 11 | | 19 → 1 | | 20 → 11 |
| Изключени върхове (от второ ниво) | 8 | | 12 | | 16 | 17 | 18 |
| Единствено инцидентно ребро | 9 → 8 | | 15 → 12 | | 15 → 16 | 15 → 17 | 9 → 18 |
| Един възможен начален връх | 2 | | | | | | |
| Получена верижка | 6 | ← | 15 | ← | 11 | ← | 2 | → | 1 | → | 9 |
| Позиции на върхове от верижката | -3 | | -2 | | -1 | | 0 | | +1 | | +2 |
| Изключени върхове (от първо ниво) | 5 | | 10 | | 19 | | 20 | | | | |
| Позиции на изключени върхове | -3 | | -1 | | +1 | | -1 | | | | |
| Номера на върхове в първото ниво | 2 | 5 | 9 | 10 | 15 | 19 | 20 | | | | |
| Тройки след добавяне на позициите | 0; 0; 0 | 0; 0; -3 | 0; 0; 2 | 0; 0; -1 | 0; 0; -2 | 0; 0; 1 | 0; 0; -1 | | | | |
| Нови индекси | 5 | 1 | 7 | 3 | 2 | 6 | 4 | | | | |
| Крайна наредба | 5 | 15 | 10 | 20 | 2 | 19 | 9 | | | | |



Продължаваме с второто ниво:

| Номера на върховете от второ ниво | 1 | 6 | 8 | 11 | 12 | 16 | 17 | 18 |
|---|---|---|---|---|---|---|---|---|
| Двойки индекси от предишното ниво | 5; 7 | 1; 2 | 7; 7 | 2; 5 | 2; 2 | 2; 2 | 2; 2 | 7; 7 |
| Изключени върхове (от второ ниво) | 6 | | 8 | | 11 | | 12 | 17 |
| Единствено инцидентно ребро | 6 → 7 | | 8 → 4 | | 11 → 3 | | 12 → 3 | 17 → 3 |
| Изключени върхове (от трето ниво) | 14 | | | | 21 | | | |
| Единствено инцидентно ребро | 1 → 14 | | | | 18 → 21 | | | |
| Един възможен начален връх | 16 | | | | | | | |
| Получена верижка | 7 | ← | 16 | → | 3 | → | 1 | → | 13 | → | 18 | → | 4 |
| Позиции на върхове от верижката | -1 | | 0 | | +1 | | +2 | | +3 | | +4 | | +5 |
| Изключени върхове (от второ ниво) | 6 | | 8 | | 11 | | 12 | 17 | | | | | |
| Позиции на изключени върхове | -1 | | +5 | | +1 | | +1 | +1 | | | | | |
| Номера на върхове във второто ниво | 1 | 6 | 8 | 11 | 12 | 16 | 17 | 18 |
| Тройки след добавяне на позициите | 5; 7; 2 | 1; 2; -1 | 7; 7; 5 | 2; 5; 1 | 2; 2; 1 | 2; 2; 0 | 2; 2; 1 | 7; 7; 4 |
| Нови индекси | 6 | 1 | 8 | 5 | 3 | 2 | 4 | 7 |
| Крайна наредба | 6 | 16 | 12 | 17 | 11 | 1 | 18 | 8 |




Остана само третото ниво:

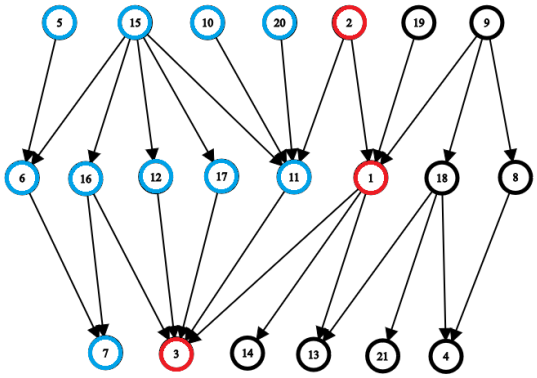
| Номера на върховете от трето ниво | 3 | 4 | 7 | 13 | 14 | 21 |
|---|---|---|---|---|---|---|
| Двойки индекси от предишното ниво | 2; 6 | 7; 8 | 1; 2 | 6; 7 | 6; 6 | 7; 7 |
| Нови индекси | 2 | 6 | 1 | 4 | 3 | 5 |
| Крайна наредба | 7 | 3 | 14 | 13 | 21 | 4 |

Доказателство за коректността на алгоритъма е фактът, че поредността на върховете от едно ниво зависи изцяло от предишното и следващото ниво, т.е. никога няма да се наложи да променяме наредбата на ниво, което вече сме преминали. Лека оптимизация на бързодействието може да се постигне, ако сортирането на тройките се прави с radix sort, вместо със стандартния алгоритъм за сортиране, реализиран в STL.

Сега остава по-лесната част от задачата (за която бяха предвидени 42 точки) – когато имаме поредността на върховете във всеки ред, да конструираме стена с минимално лице (т.е. с минимална широчина). За целта ще използваме алчен алгоритъм, който се явява лека модификация на итеративния алгоритъм за топологично сортиране. Необходимо ни е предварително да знаем за всеки връх кои са върховете с най-голям индекс от съседните нива, с които има общо ребро.

Нека началото на стената съответства на координата $x = 0$. Тогава първите камъни от всеки ред ще започнат от $x = 0$. На тази координата няма да има краища на камъни, защото това би противоречало на условието, че дължината им е поне единица. При $x = 1$ обаче част от камъните ще бъдат „затворени“ и по-конкретно тези, които имат най-много по един съсед в предишното и в следващото ниво (поне един такъв камък съществува). На тези редове, от координата $x = 1$ ще започне вторият камък (ако има такъв).

По-конкретно, алгоритъмът поддържа индексите на отворените камъни във всеки ред от стената (или ниво на графа). На всяка итерация увеличаваме текущата $x$-координата с 1 и проверяваме кои камъни могат да бъдат затворени, при което отваряме следващия на този ред. За да бъде затворен даден камък в общия случай, трябва да бъдат отворени и двата му съседа с най-големи индекси от съседните редове и разбира се – самият той. Можем да пазим броя оставащи камъни, които трябва да бъдат отворени, преди някой от отворените в момента да може да бъде затворен. При отварянето на някой камък трябва да актуализираме този брой за релевантните камъни и ако се получи 0, да пристъпим към затваряне на съответния камък на следващата итерация. Алгоритъмът приключва, когато бъде затворен и последният камък в стената, и $x$-координатата, на която това се случи, е широчината ѝ. Необходимо е да се направи корекция, ако на някои редове последните камъни са приключили на по-малка $x$-координата.





Отново ще покажем нагледно с един пример. Нека сме стигнали до координата $x = 11$ при строенето на стената. В този момент сме приключили с добавянето на камъни с номера 5, 15, 10 и 20 от първия ред; 6, 16, 12, 17 и 11 от втория ред и 7 от третия ред. Отворените камъни към момента са 2, 1 и 3 съответно на първи, втори и трети ред. На предишната итерация ($x = 10$) сме затворили камък 11 и сме отворили този с номер 1. При това ние знаем, че за съседът на връх 2 с най-голям индекс в следващото ниво на графа е именно 1, и също този на връх 3 от предишното ниво е именно 1. Тъй като съответстващият на връх 1 камък е вече отворен, нямаме повече неудовлетворени изисквания за връх 2 и за връх 3 и затова ги затваряме на текущата стъпка ($x = 11$). Това е съпроводено с отваряне на камъни 19 и 14. При желание читателят може да довърши стената до последната итерация ($x = 17$).

Необходимо и достатъчно условие за оптималността на стената е за всяка $x$-координата от нея да има поне един ръб между два камъка на някой ред, за който е вярно, че съществува ръб между друга двойка камъни на същия, предишния или следващия ред на координата $x + 1$. Посоченият алчен алгоритъм изпълнява това условие.

*Автори: Добрин Башев, Павел Петров*