Национална програма
"Обучение за ИТ умения и кариера"
https://it-kariera.mon.bg

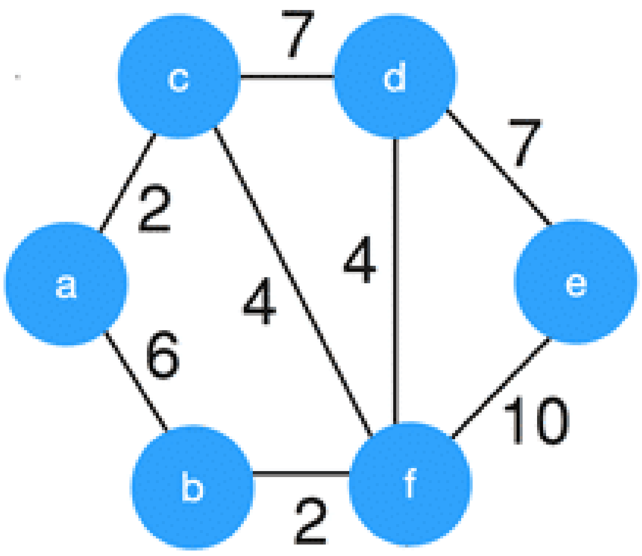
Министерството на
образованието и науката
https://www.mon.bg

Графи и алгоритми върху графи

Алгоритми и структури от данни

Съдържание

- Начини на представяне на графите. Компоненти на свързаност
  - Упражнения: намиране на компоненти на свързаност
- Топологично сортиране
  - Упражнения: топологично сортиране
- Пътища в граф, алгоритъм на Дейкстра
  - Упражнения: пътища в граф
- Други алгоритми върху графи
  - Упражнения: други алгоритми върху графи

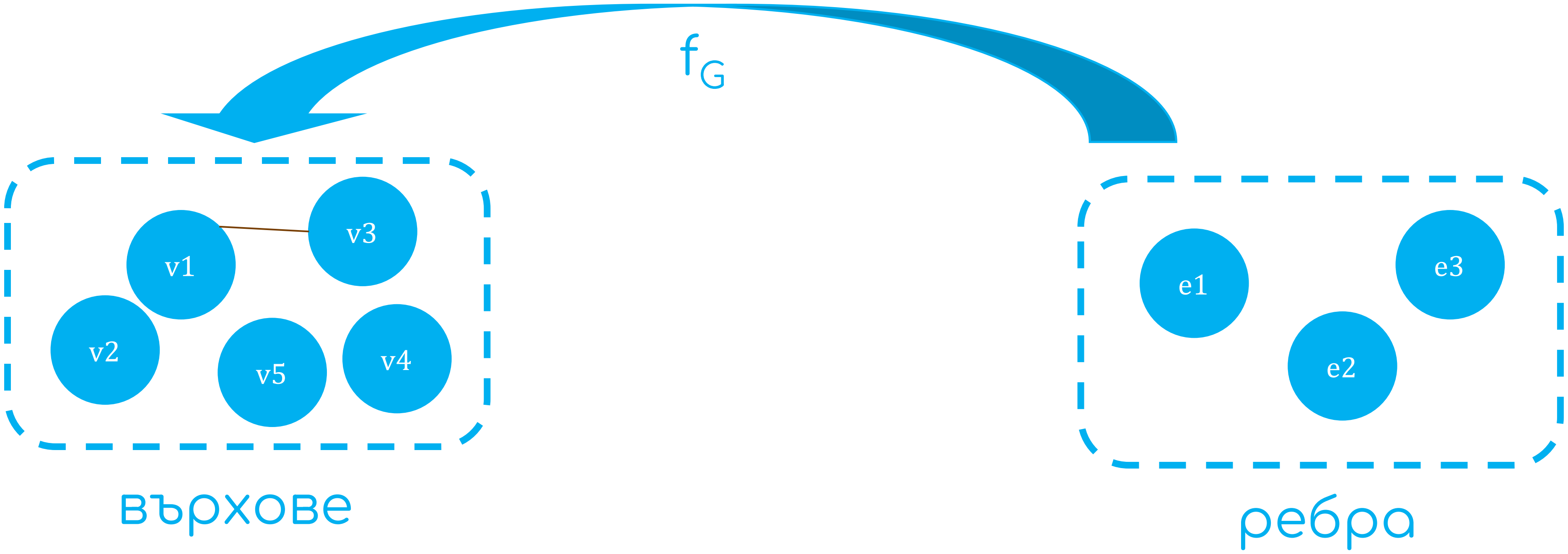


Определения и терминология

Представяне на графи

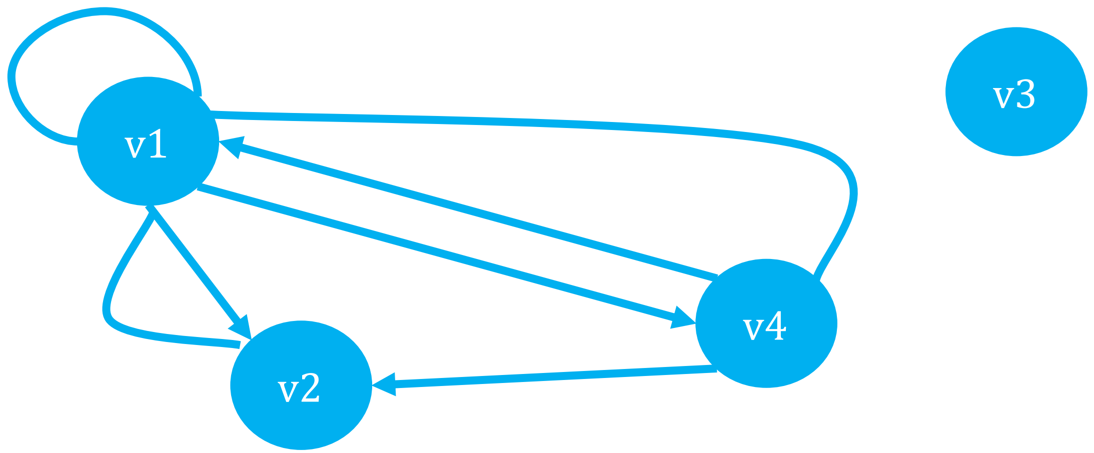
Ориентиран мултиграф [1/2]

- $V=(v_1,v_2,v_3,\ldots v_n)$ - множество на върховете
- $E=(e_1,e_2,e_3,e_4,\ldots e_m)$- множество на ребрата
- f_G: E -> VxV – свързваща функция, съпоставяща на всеки елемент на E наредена двойка върхове



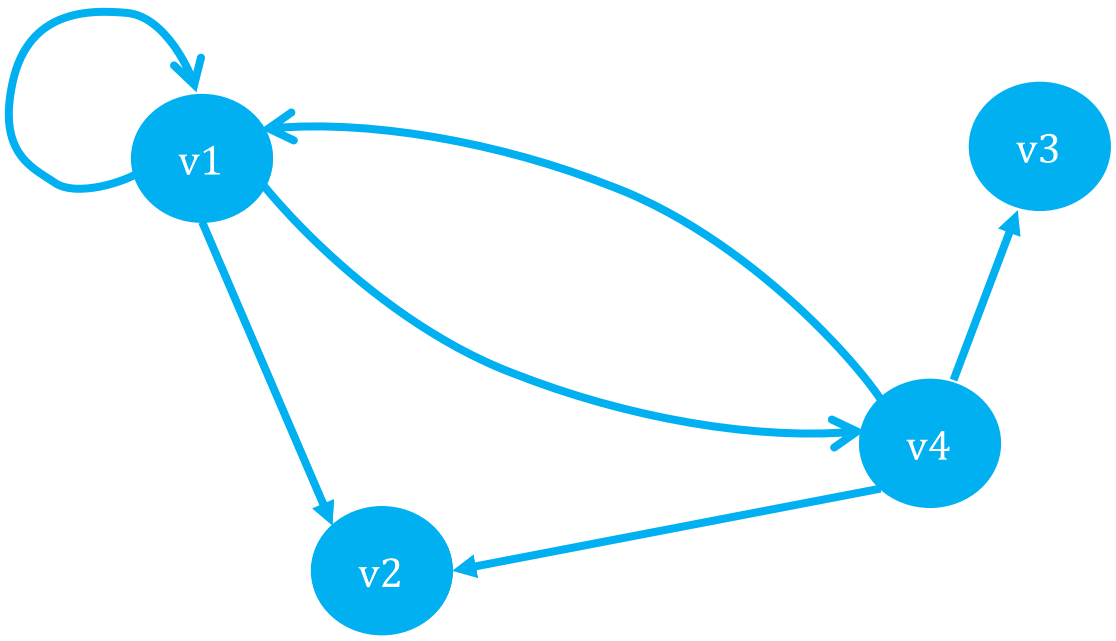
Ориентиран мултиграф [2/2]

- $G=\{V, E, f_G\}$ - краен ориентиран мултиграф
- v_i и v_j са върхове (точки) , които са свързани с ребра (стрелки). *Означава се* $e=(v_i,v_j)$.
- В един ориентиран мултиграф може да съществуват:
  - изолиран връх – в който не влизат и не излизат ребра
  - примка – ребро, чието начало и край съвпадат
  - повече от едно ориентирано ребро между два върха



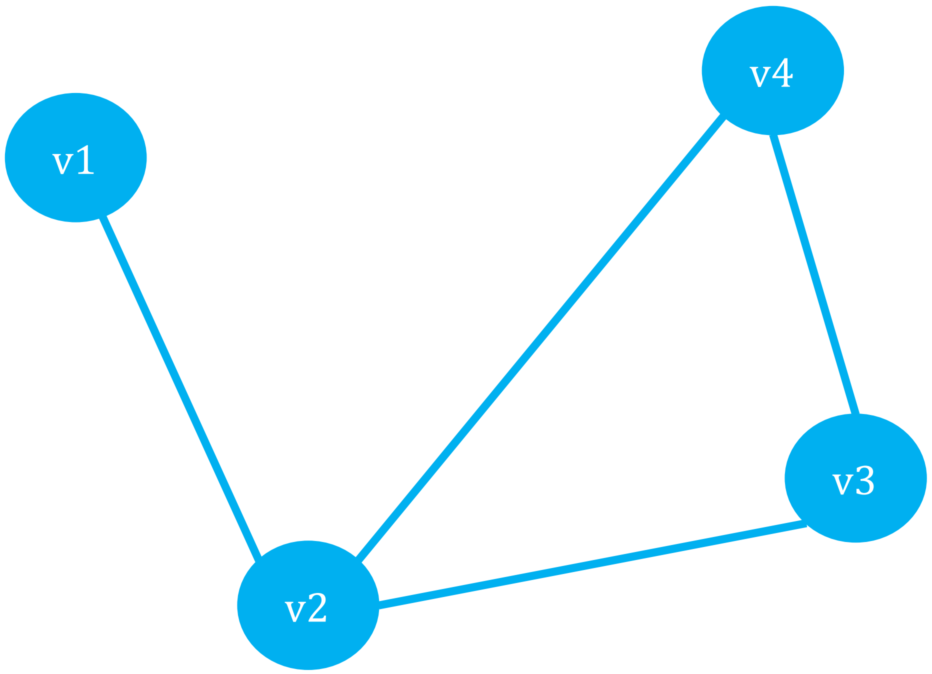
Ориентиран граф

- $G = \{V, E\}$ - краен ориентиран граф
- f_G е инективна (еднозначна)
- всяко ребро се определя еднозначно от съответната двойка върхове и има посока



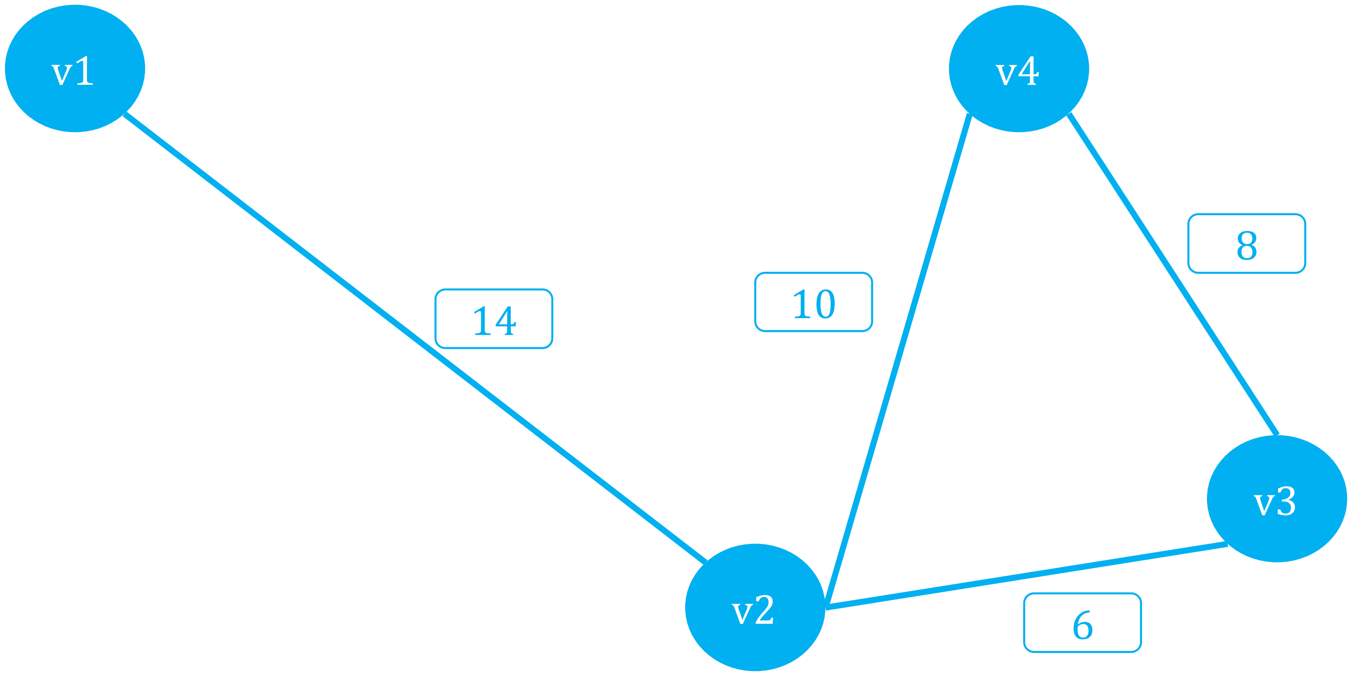
Неориентиран граф

- $G = \{V, E\}$ - краен ориентиран граф
- f_G е инективна (еднозначна)
- всяко ребро се определя еднозначно от съответната двойка върхове и няма посока



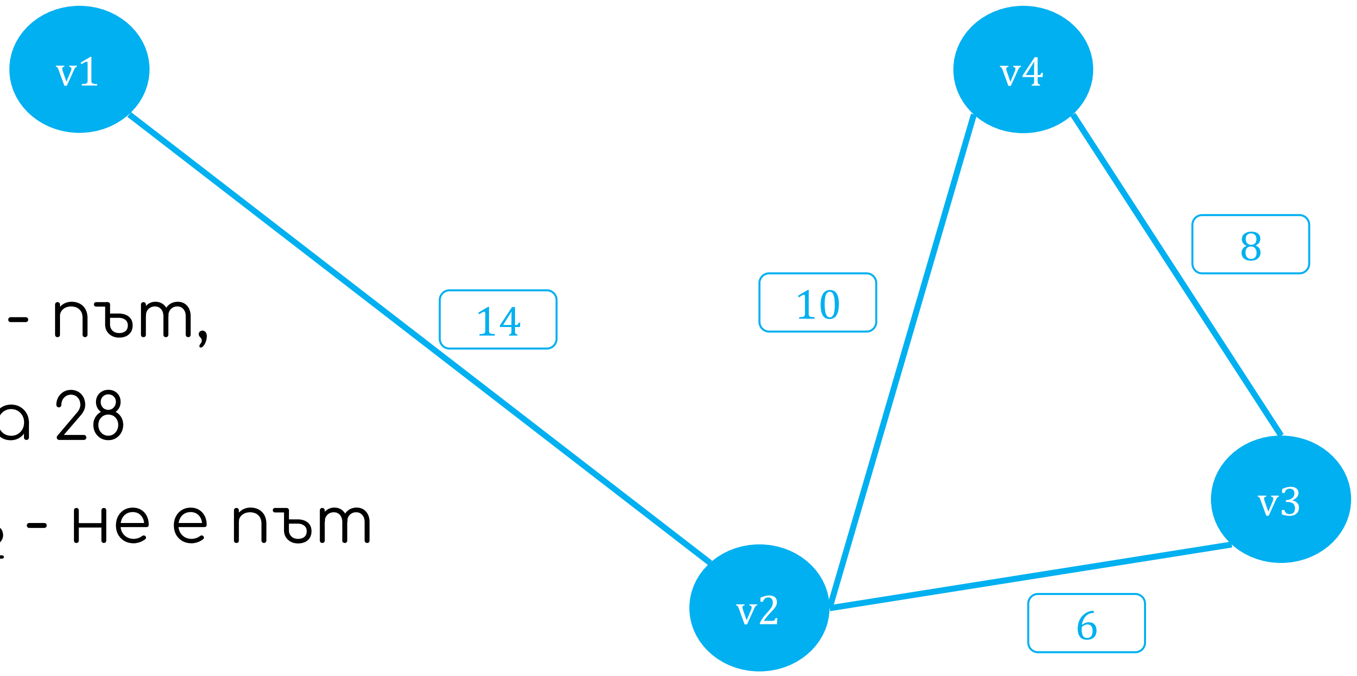
Претеглен граф

всяко ребро има тегло



Път в граф [1/2]

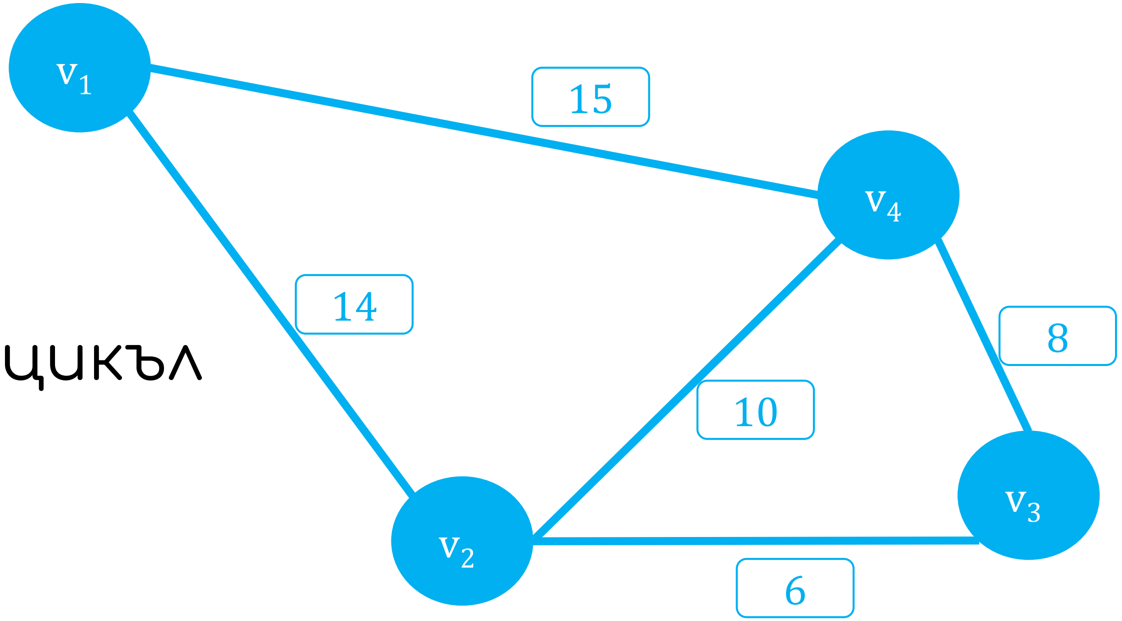
- Последователността от върхове $v_{i1}, v_{i2}, v_{i3}, \ldots v_{il}$, наричаме път, ако за всяко $j=1 \ldots l-1$, съществува $e \in E$, такова, че $f_G(e) = (v_{ij}, v_{ij+1})$.
- Естественото число l наричаме дължина на пътя.



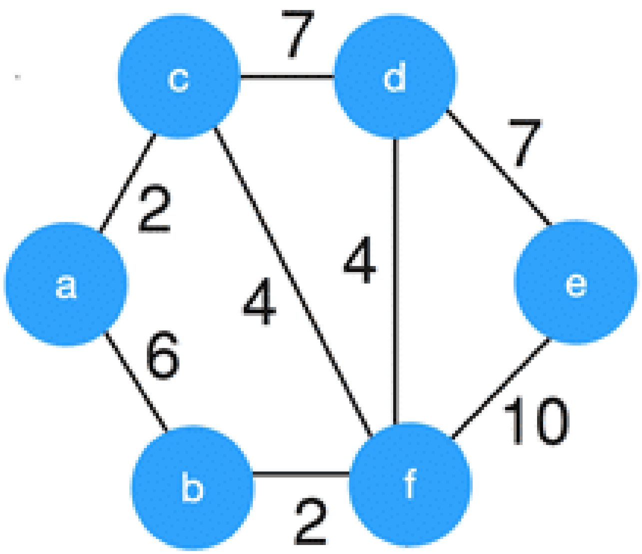
- v_1, v_2, v_3, v_4 - път, с дължина 28
- v_1, v_4, v_3, v_2 - не е път

Път в граф [2/2]

- Ако $v_{i1} = v_{il}$, пътя се нарича цикъл.
- Граф, съдържащ поне един цикъл наричаме цикличен, в противен случай казваме, че е ацикличен.
- Графа $G=\{V, E\}$ ще наричаме **свързан**, ако за всяка двойка върхове $v_i, v_j \in V$ съществува път от v_i до v_j.



- v_1, v_2, v_4, v_1 - цикъл

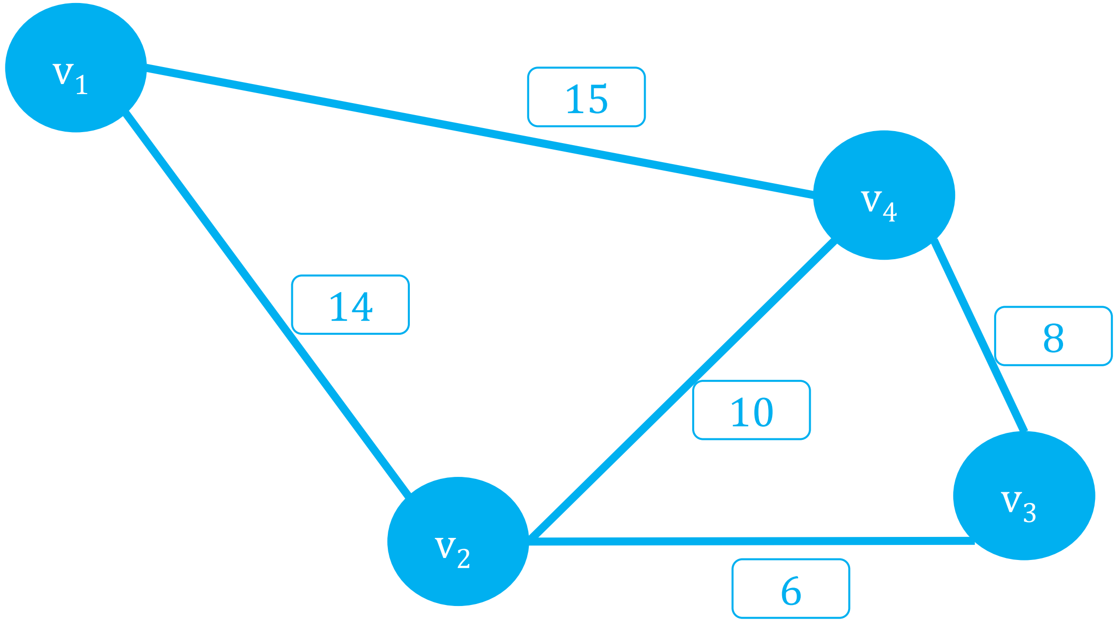


Представяне на граф

Графи

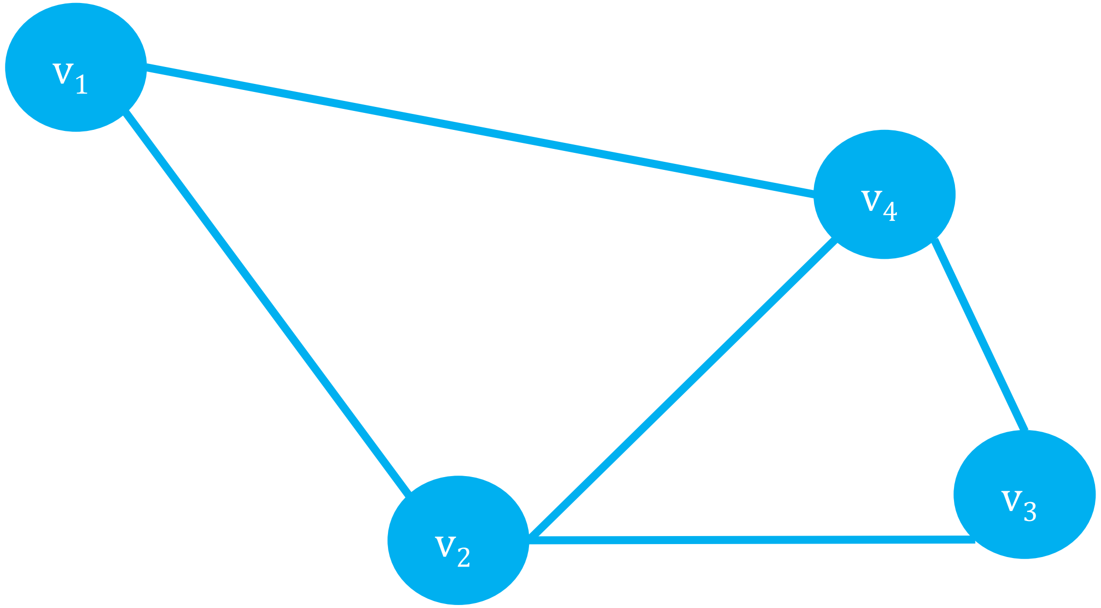
Представяне на граф [1/4]

- Списък на съседите
- Всеки връхсъдържа списък на своите съседи
- v_1 -> { v_2, v_4 }
- v_2 -> { v_1, v_4, v_3 }
- v_3 -> { v_2, v_4 }
- v_4 -> { v_1, v_2, v_3 }



Представяне на граф [2/4]

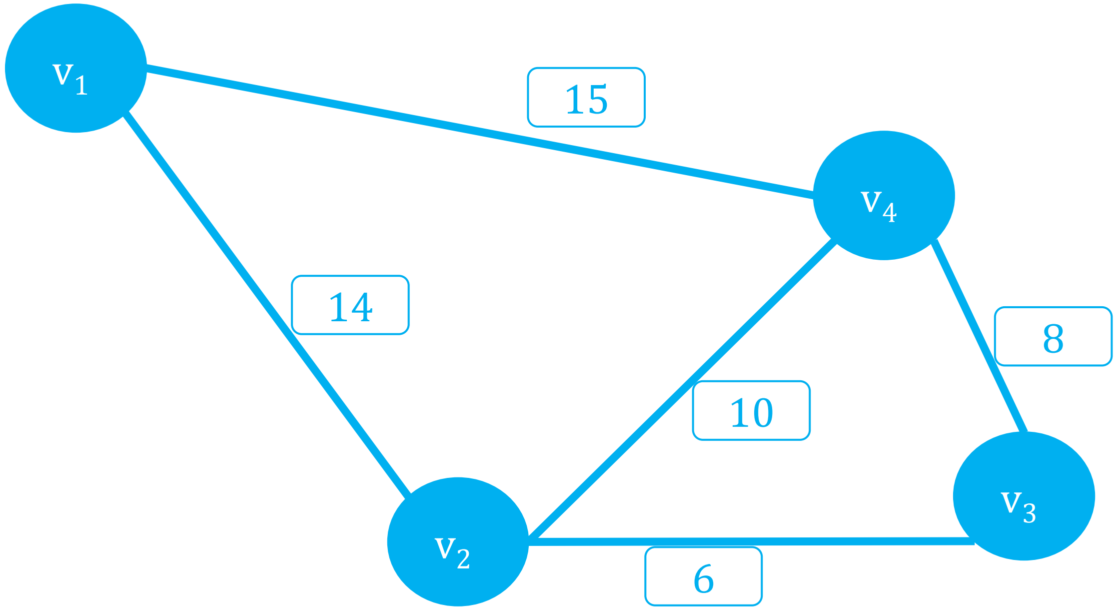
- Матрица на свързаност
- 1 - ако има свързващо ребро
- 0 - ако няма свързващо ребро



връх	v_1	v_2	v_3	v_4
v_1	0	1	0	1
v_2	1	0	1	1
v_3	0	1	0	1
v_4	1	1	1	0

Представяне на граф [3/4]

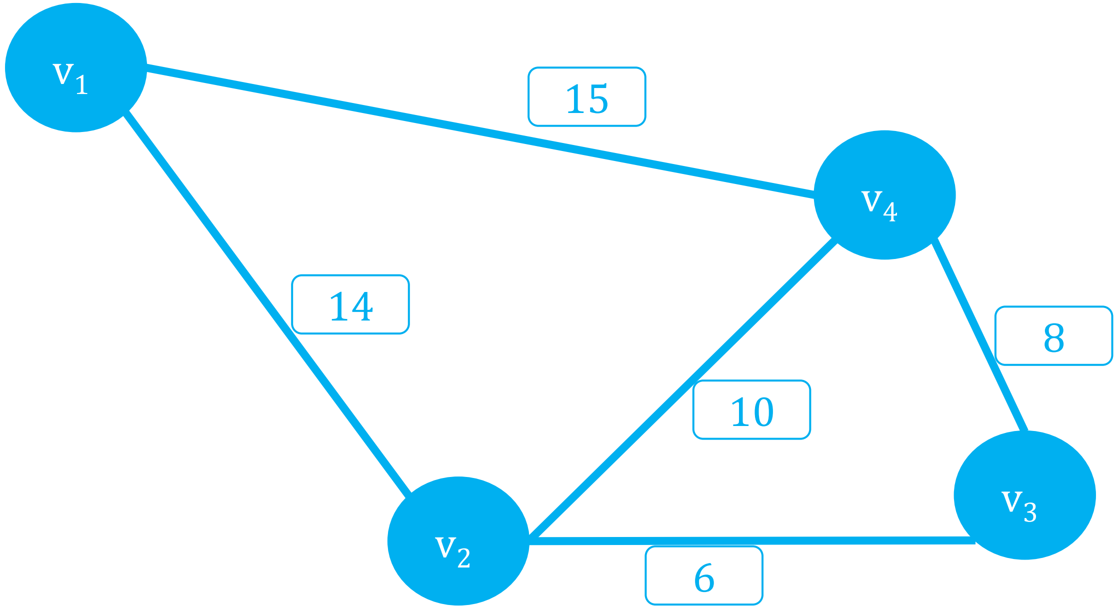
- Матрица на свързаност
- Стойността на теглото - ако има свързващо ребро
- 0 - ако няма свързващо ребро

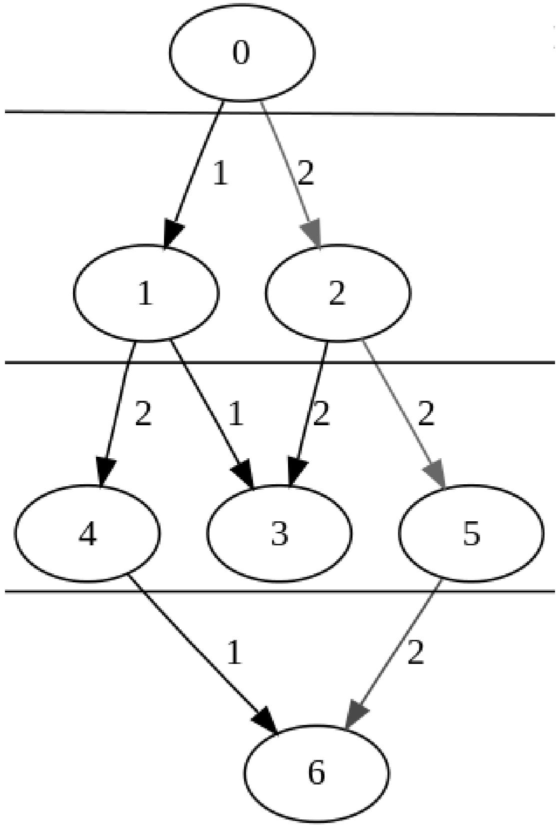


връх	v_1	v_2	v_3	v_4
v_1	0	14	0	15
v_2	14	0	6	10
v_3	0	6	0	8
v_4	15	10	8	0

Представяне на граф [4/4]

- Списък на ребрата
- Изброяват се всички ребра, прекарани в графа
- $\{ v_1, v_2 \}$
- $\{ v_1, v_4 \}$
- $\{ v_2, v_4 \}$
- $\{ v_2, v_3 \}$
- $\{ v_3, v_4 \}$





Топологично сортиране

Графи

Топологично сортиране

Топологично сортиране (подреждане) на ориентиран граф

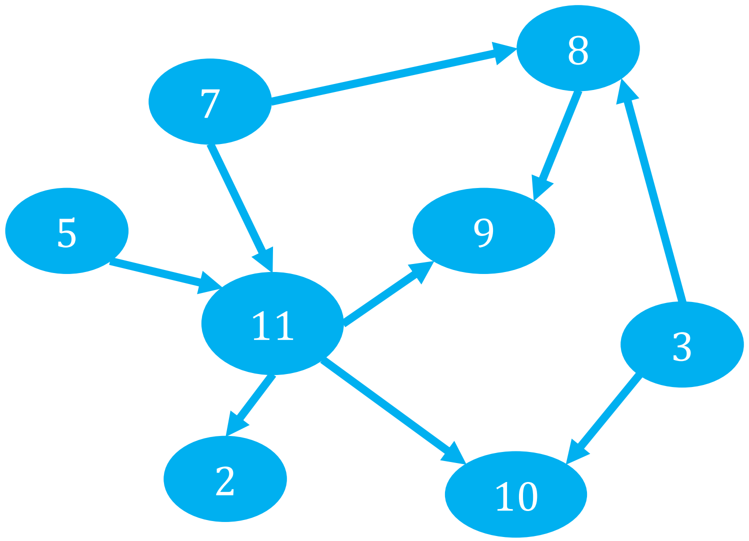
Линейно подреждане на върховете му, така че за всяко насочено ребро от върха u до връх v, u идва преди v в подреждането.

Пример:

7 → 5 → 3 → 11 → 8 → 2 → 9 → 10

3 → 5 → 7 → 8 → 11 → 2 → 9 → 10

5 → 7 → 3 → 8 → 11 → 10 → 9 → 2

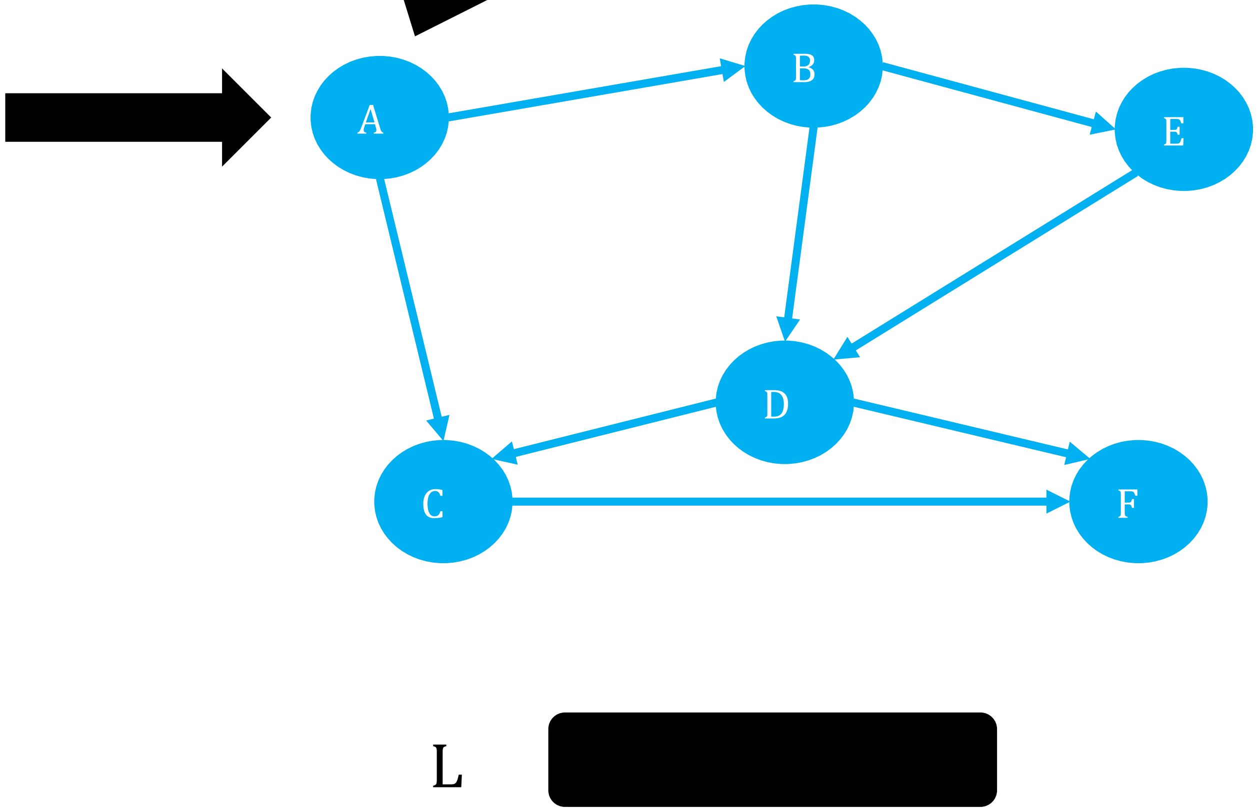


Правила при топологично сортиране

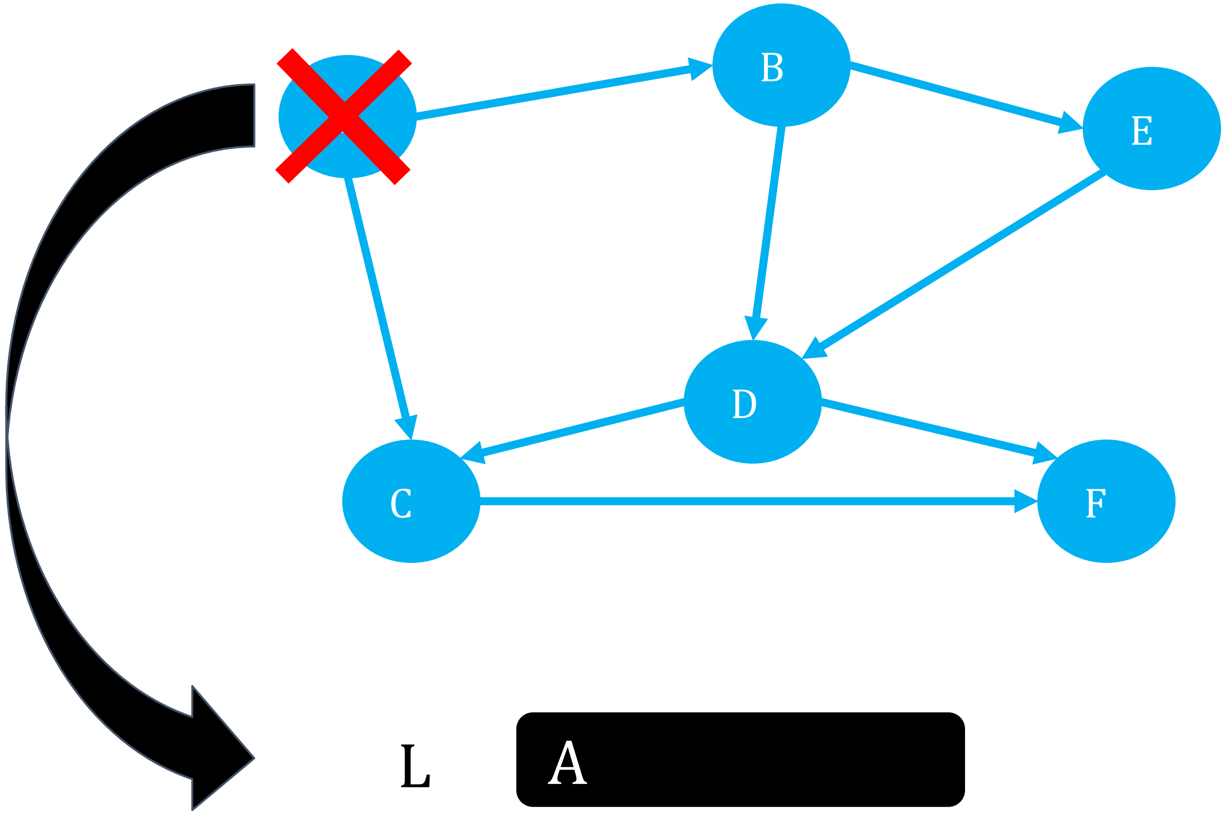
- Топологично сортиране не може да бъде направено при:
  - неориентиран граф
  - цикличен граф
- Сортирането не е уникално
- Съществуват различни сортирания и те дават различни резултати

Стъпка 1. Намираме възел без входящи ребра

Възелът A е единственият
възел без входящи ребра

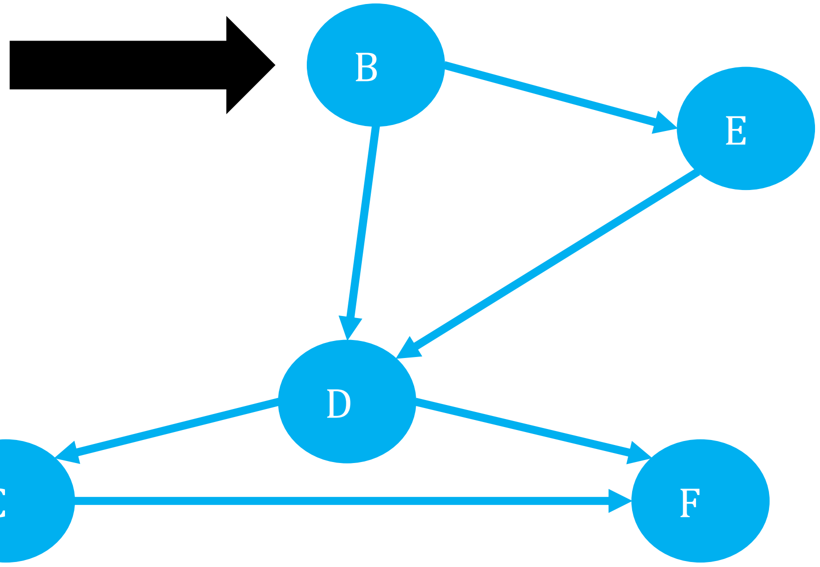


Стъпка 2. Премахваме възел A и съответните му ребра



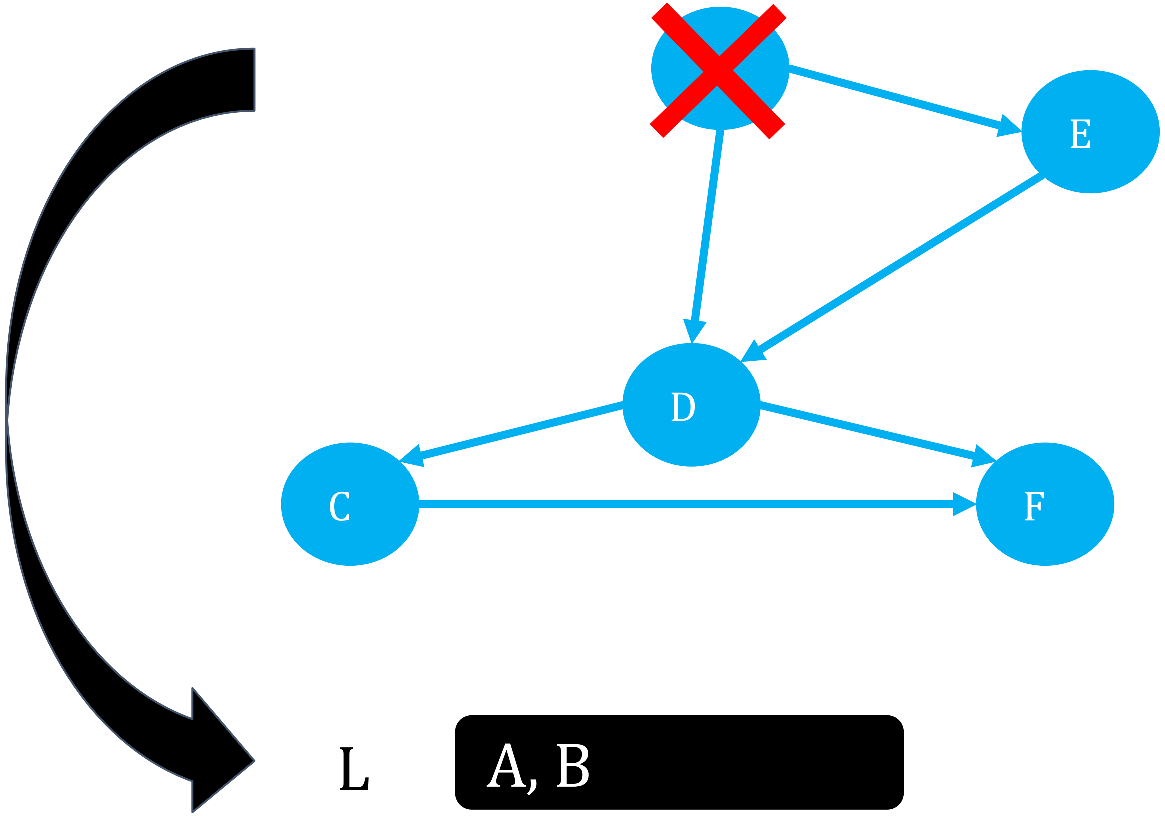
Стъпка 3. Намираме възел без входящи ребра

Възелът B е единственият
възел без входящи ребра



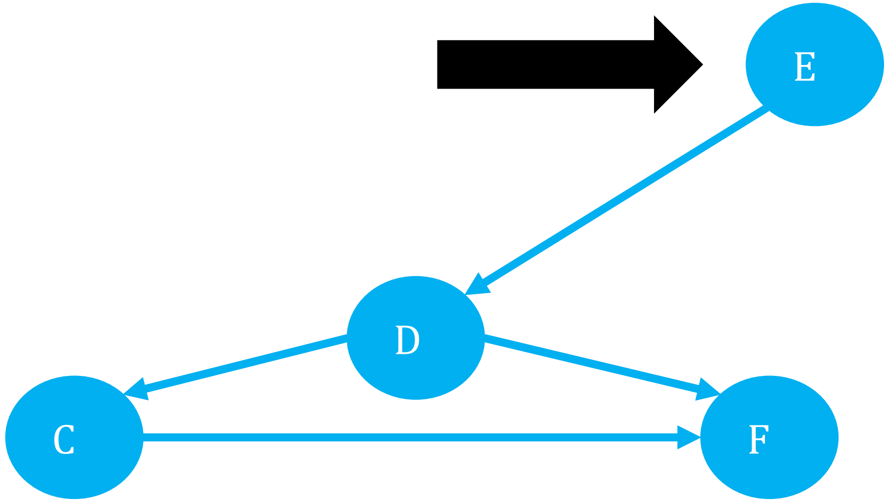
L A

Стъпка 4. Премахваме възел B и съответните му ребра



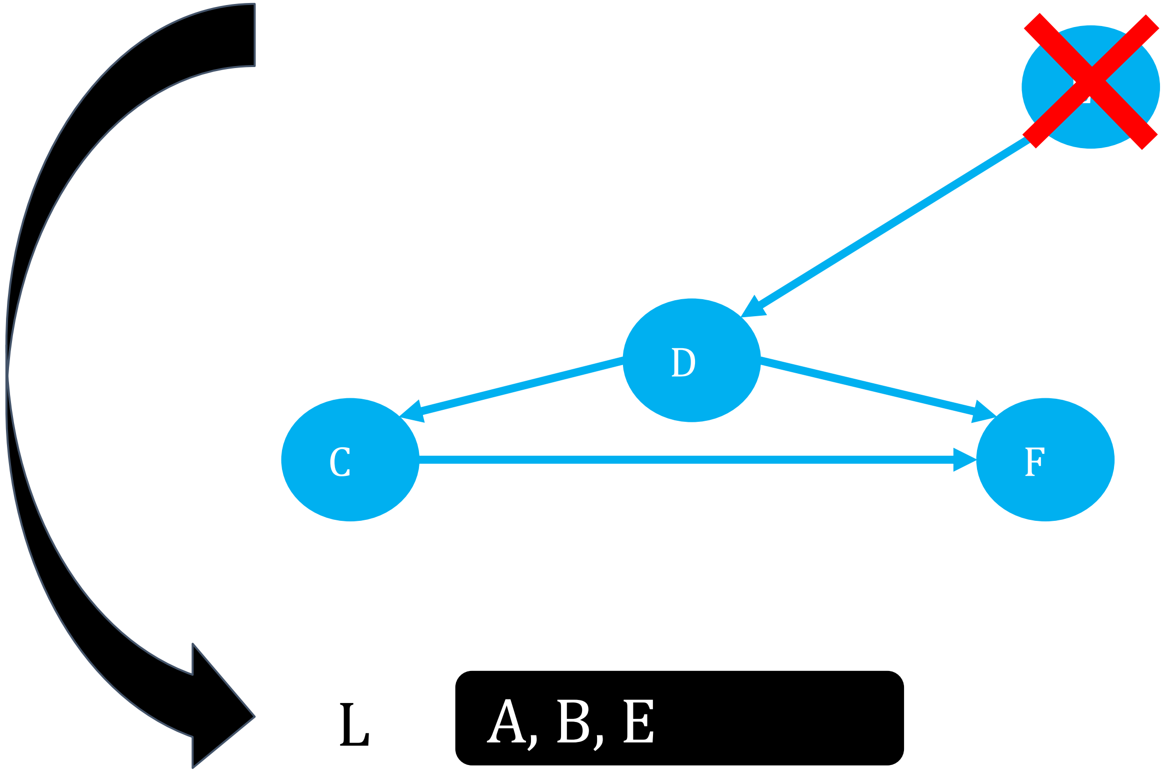
Стъпка 5. Намираме възел без входящи ребра

Възелът E е единственият
възел без входящи ребра

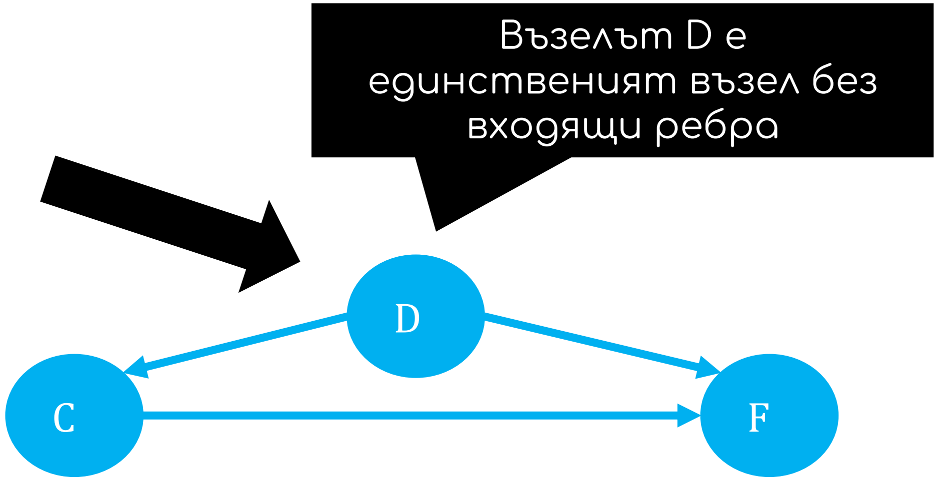


L A, B

Стъпка 6. Премахваме възел E и съответните му ребра

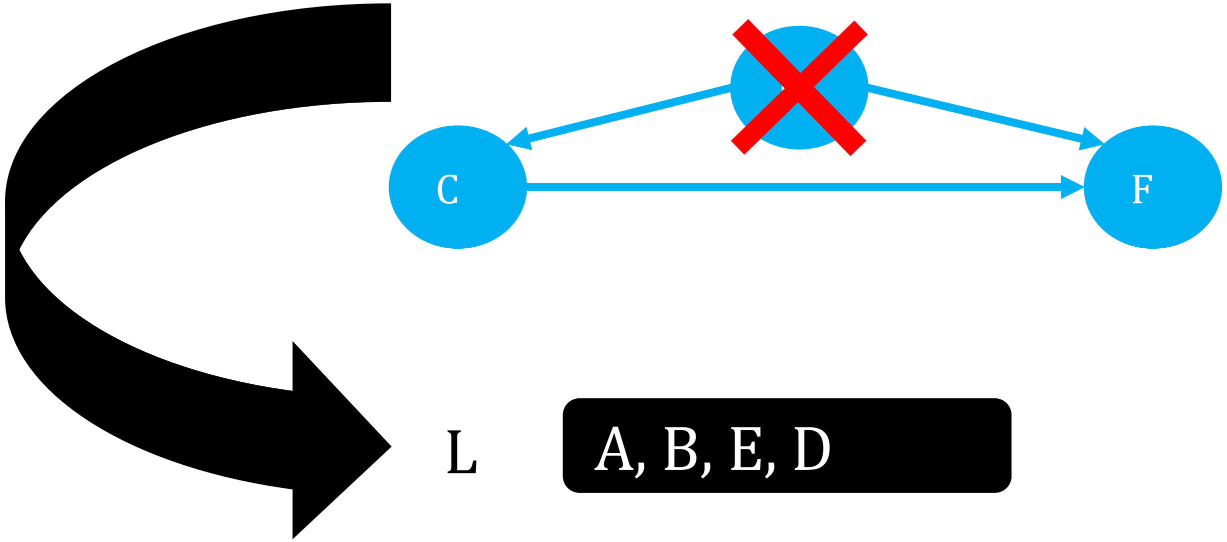


Стъпка 7. Намираме възел без входящи ребра

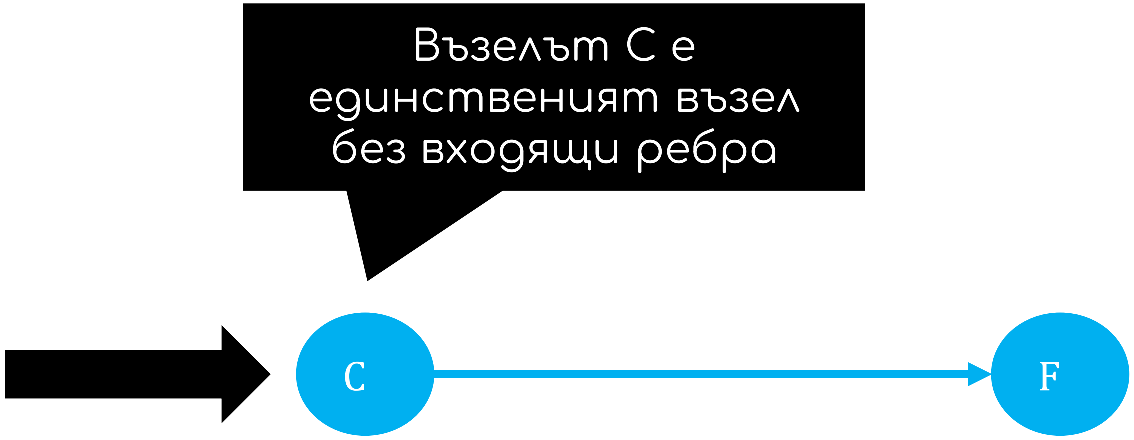


L A, B, E

Стъпка 8. Премахваме възел E и съответните му ребра

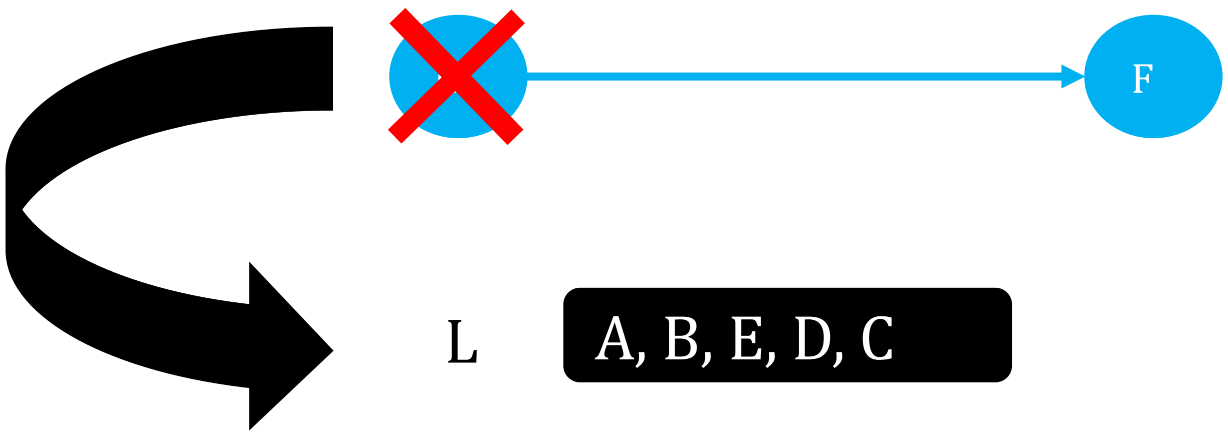


Стъпка 9. Намираме възел без входящи ребра



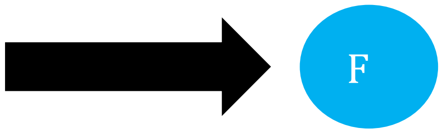
L A, B, E, D

Стъпка 10. Премахваме възел С и съответните му ребра



Стъпка 11. Намираме възел без входящи ребра

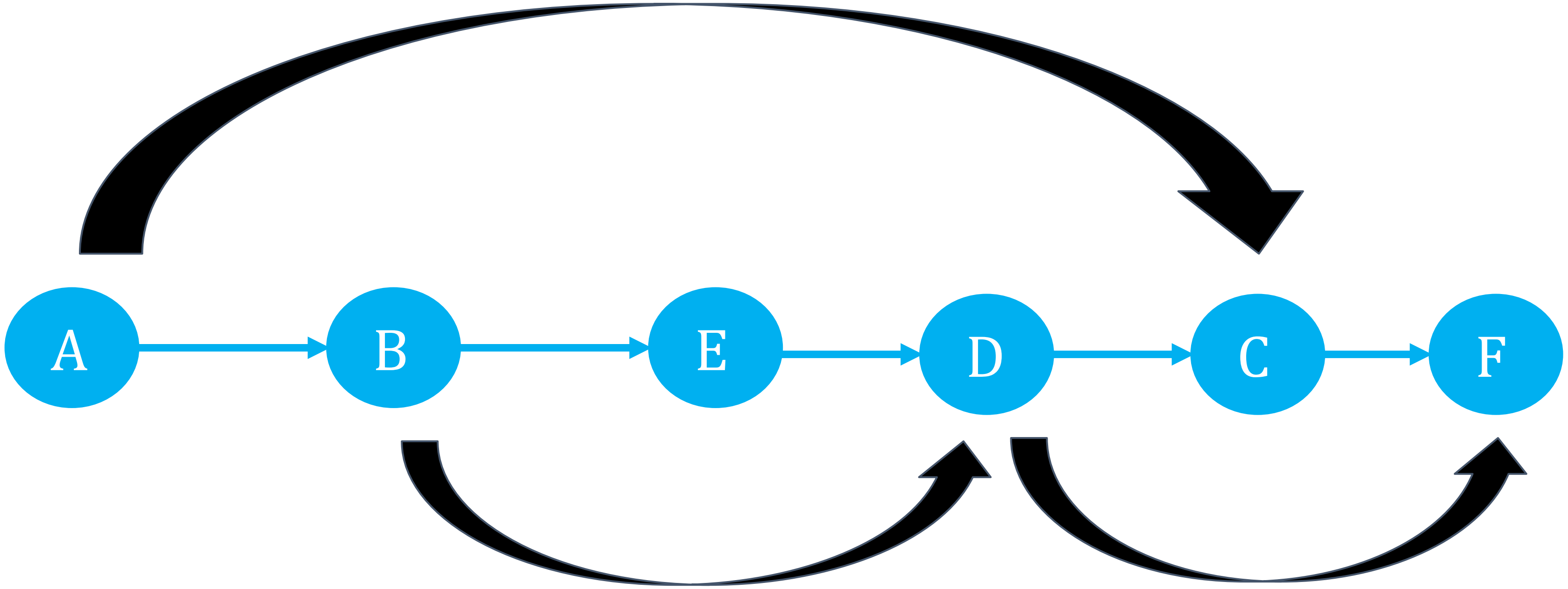
Възелът F е единственият възел без входящи ребра



L A, B, E, D, C

Стъпка 12. Премахваме възел F и съответните му ребра

Резултат от топологичното сортиране



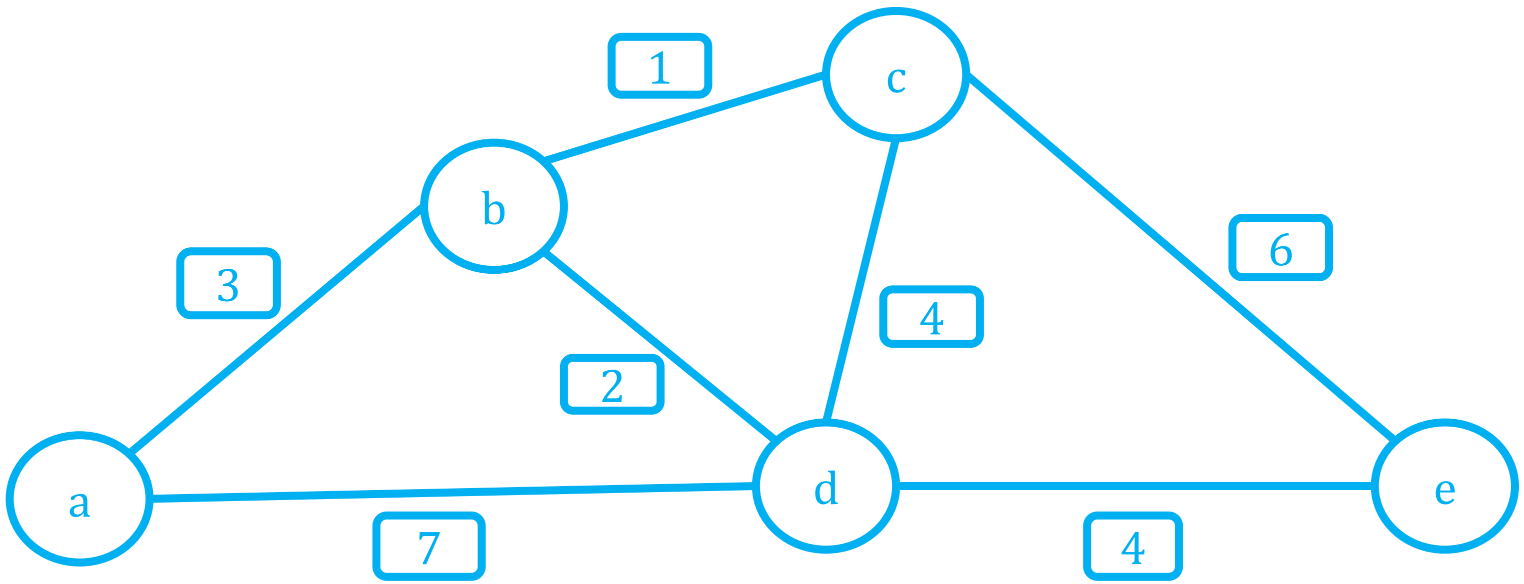
Топологично сортиране: DFS + цикъл [1/2]

```
//sortedNodes = { }  // свързан списък, който съдържа резултата
visitedNodes = { }  // набор от посетени възли
foreach node in graphNodes
   TopSortDFS(node)

TopSortDFS(node)
   if node ∉ visitedNodes
      visitedNodes ← node
      for each child c of node
         TopSortDFS(c)
      добави node възела в sortedNodes
```

Източник: https://www.cs.usfca.edu/~galles/visualization/TopoSortDFS.html

Топологично сортиране: DFS + цикъл [2/2]

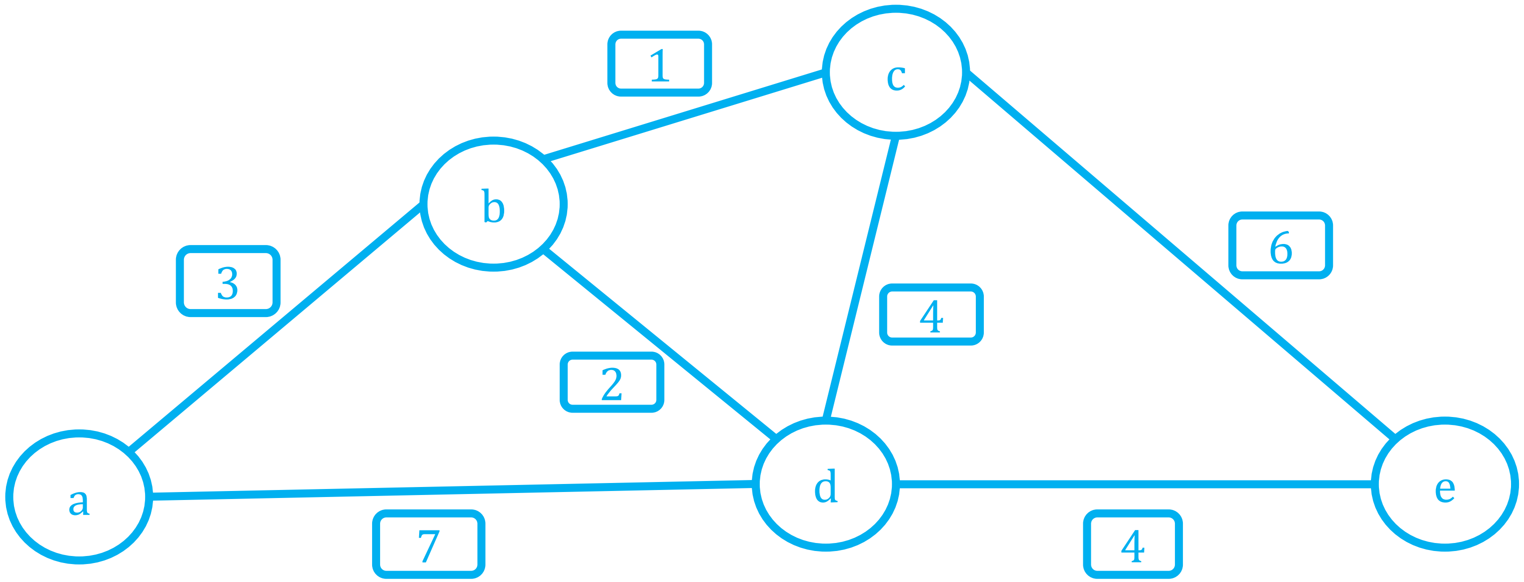
```
sortedNodes = { }  // свързан списък, съдържащ резултата
visitedNodes = { }  // списък от посетените възли
cycleNodes = { }  // набор от възли в настоящия цикъл от обхождането в дълбочина
foreach node in graphNodes
    TopSortDFS(node)

TopSortDFS(node)
    if node ∈ cycleNodes
        return "Грешка: намерен е цикъл"
    if node ∉ visitedNodes
        visitedNodes ← node
        cycleNodes ← node
        for each child c of node
            TopSortDFS(c)
        премахни node от cycleNodes
        добави node в sortedNodes
```



Алгоритъм на Дейкстра

Алгоритъм на Дейкстра [1/13]

Намиране на минимален път в претеглен граф с неотрицателни тегла



Алгоритъм на Дейкстра [2/13]

- Представяне на графа като матрица W, на която елементът Wij е равен на дължината на реброто, съединяващо i-тия и j-ия връх
- Означаваме с ∞ ребрата, чиито върхове не са инцидентни

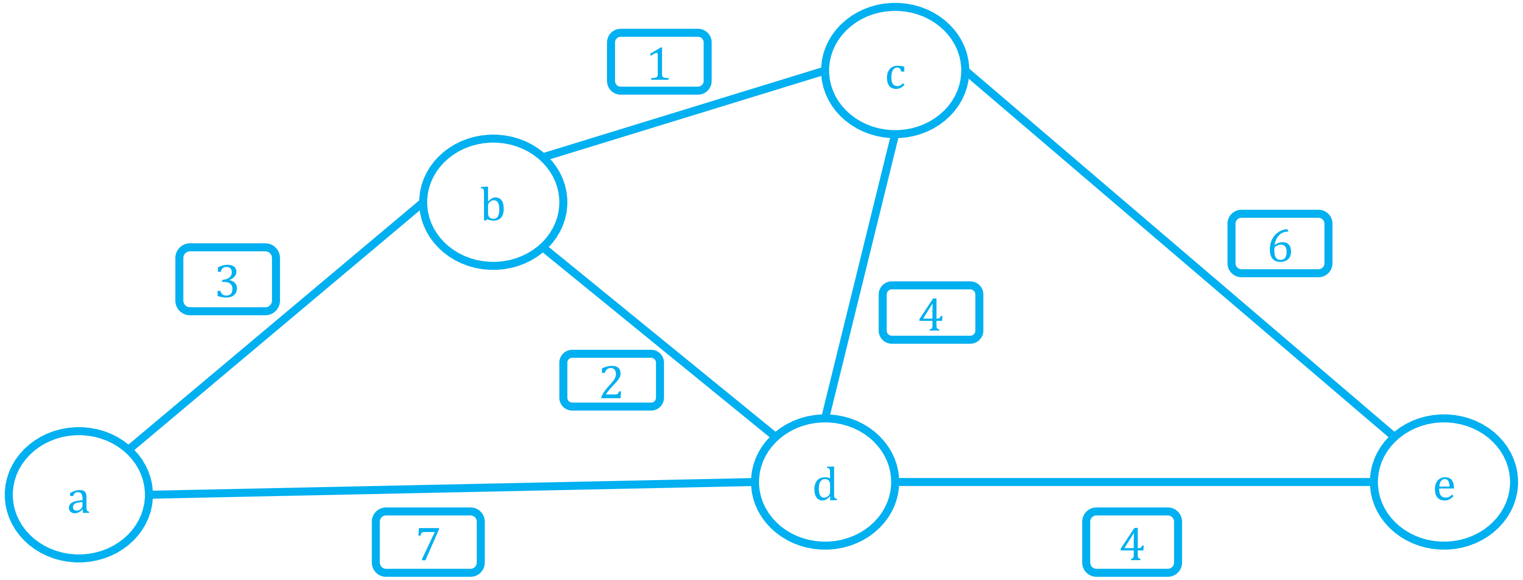
	a	b	c	d	e
a	0	3	∞	7	∞
b	3	0	1	2	∞
c	∞	1	0	4	6
d	7	2	4	0	4
e	∞	∞	6	4	0

Алгоритъм на Дейкстра [3/13]

- За всички върхове даваме стойност на масива L: l(i)=∞, освен този с номер u1, т.е. l(u1)=0.
- За всички върхове се дава стойност на масива H: h(i)=0, а за h(u1)=1;
- Започваме от връх *u1,* той е текущ и полагаме *p=u1;*
- За всички върхове *i,* за които *h(i)=0* и са инцидентни /съседни/ с върха *p* преизчисляваме по формулата *l(i)=min{l(i), l(t)+W[t,i]}*
- Измежду тях намираме такъв, за който *l(i)* е минимално, ако не е намерен такъв минимум, т.е. стойността на всички посетени върхове е безкрайност, то не съществува път и КРАЙ;
- Полагаме *p* да е равен на намерения връх с минимална стойност и правим *h[p]=1*;
- Ако p=u2, то е намерен път със стойност *l(u2)* и КРАЙ
    - иначе отиваме на стъпка 4.

Алгоритъм на Дейкстра [4/13]

- Търсим минимален път от връх a до връх e
- u1 = a, u2 = e;
- Полагаме l[a] = 0;
- Полагаме l[b] = … = l[e] = 100;

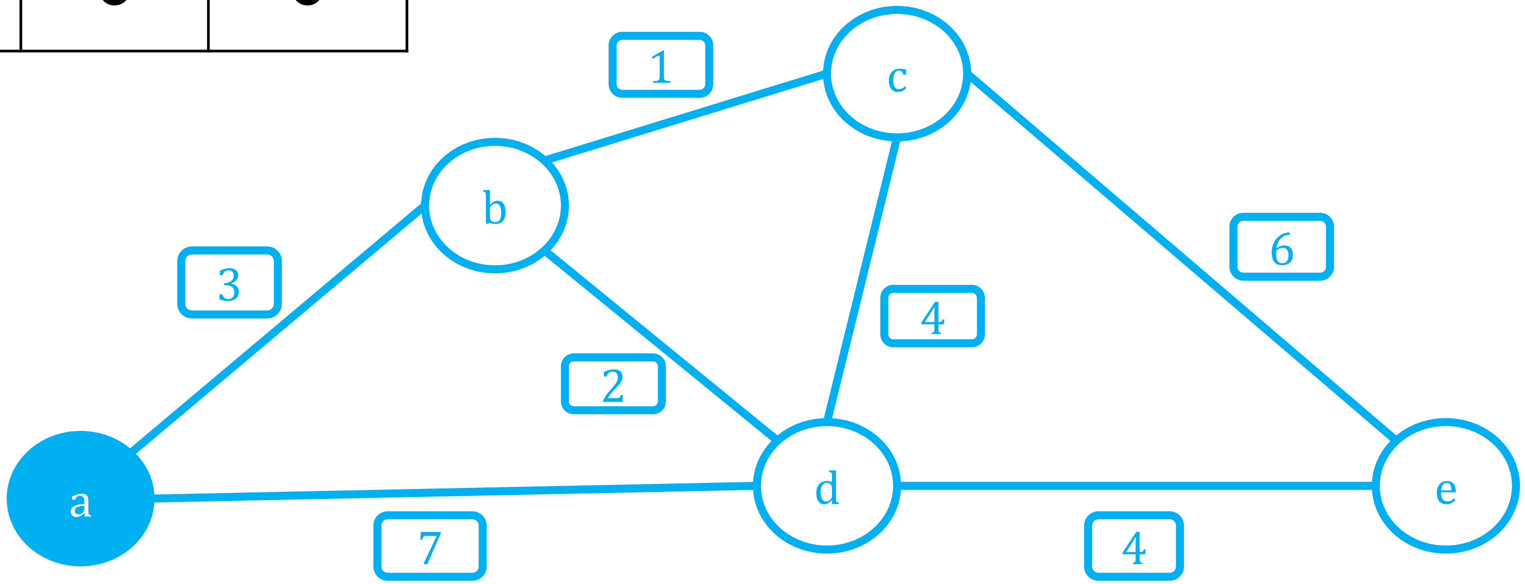


Предполагаме, че няма да надхвърлим път с дължина 100.

Алгоритъм на Дейкстра [5/13]

	a	b	c	d	e
L	0	100	100	100	100
H	1	0	0	0	0

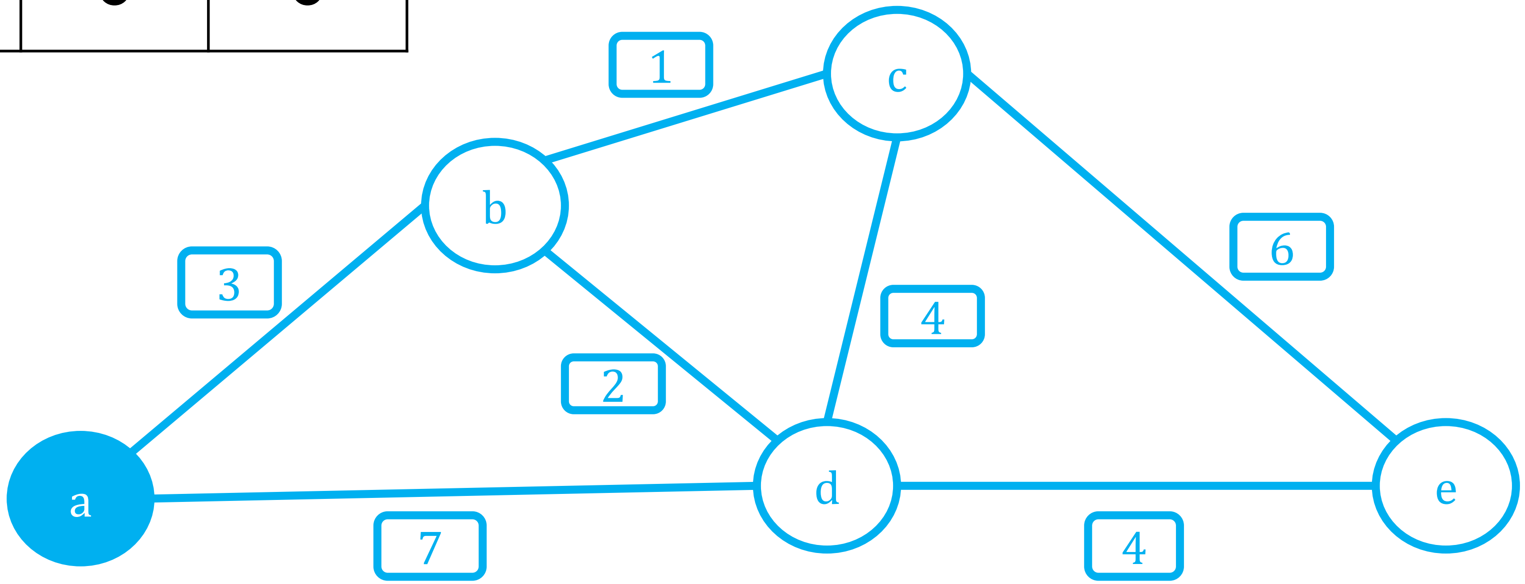
Полагаме p=a, h(a)=1;
Съседните на върха a са b, и d.



Алгоритъм на Дейкстра [6/13]

	a	b	c	d	e
L	0	3	100	7	100
H	1	0	0	0	0

l(b)=min{l(b), l(a)+3}=min{100, 0+3} =3
l(d)=min{l(d), l(a)+7}=min{100, 0+7} =7



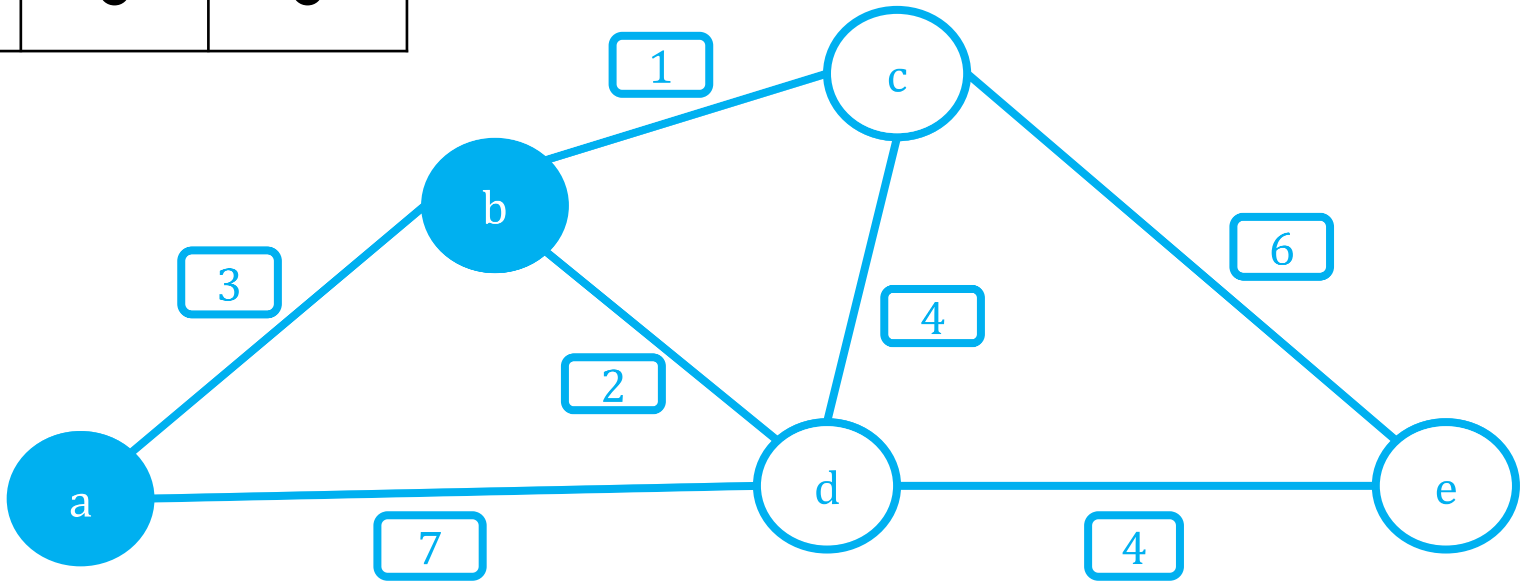
Алгоритъм на Дейкстра [7/13]

	a	b	c	d	e
L	0	3	100	7	100
H	1	1	0	0	0

Най-малката стойност е на l(b) и тя е 3.

Полагаме p=b и този връх го маркираме.

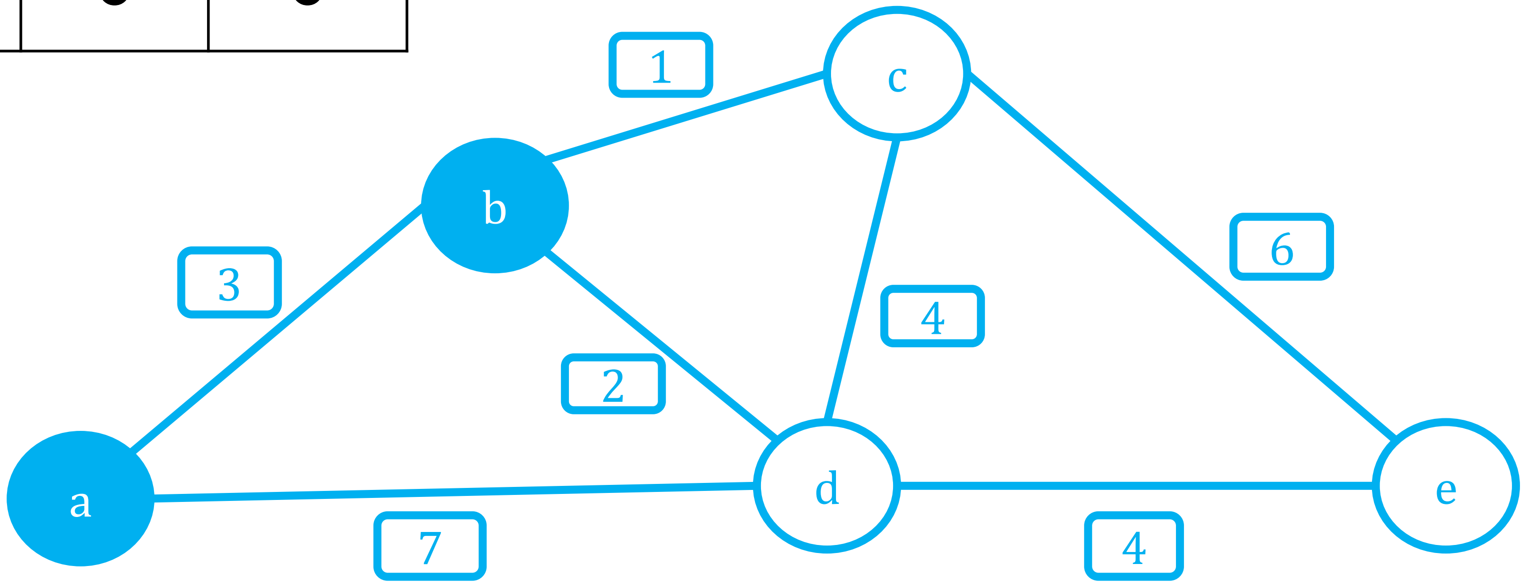
Полагаме h(b)=1;



Алгоритъм на Дейкстра [8/13]

	a	b	c	d	e
L	0	3	100	7	100
H	1	1	0	0	0

Съседните немаркирани върхове на b са c и d



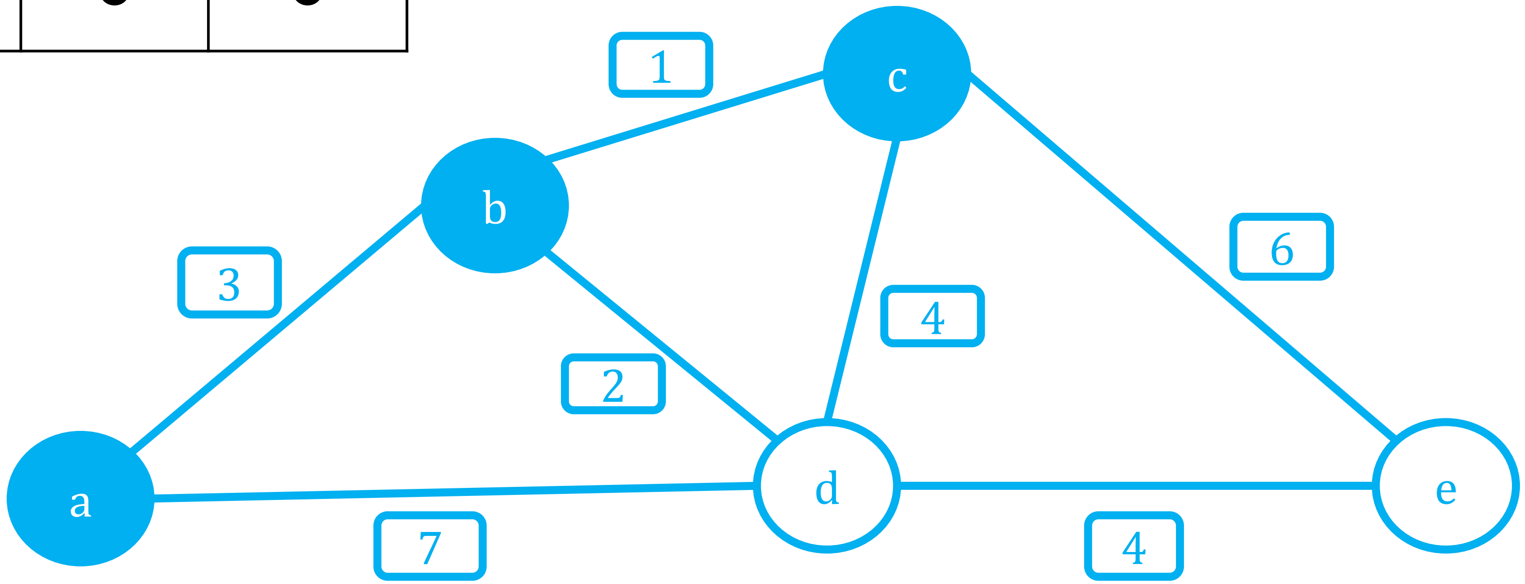
Алгоритъм на Дейкстра [9/13]

	a	b	c	d	e
L	0	3	4	7	100
H	1	1	1	0	0

Най-малката стойност е на l(c) и тя е 4.

Полагаме p=c и този връх го маркираме.

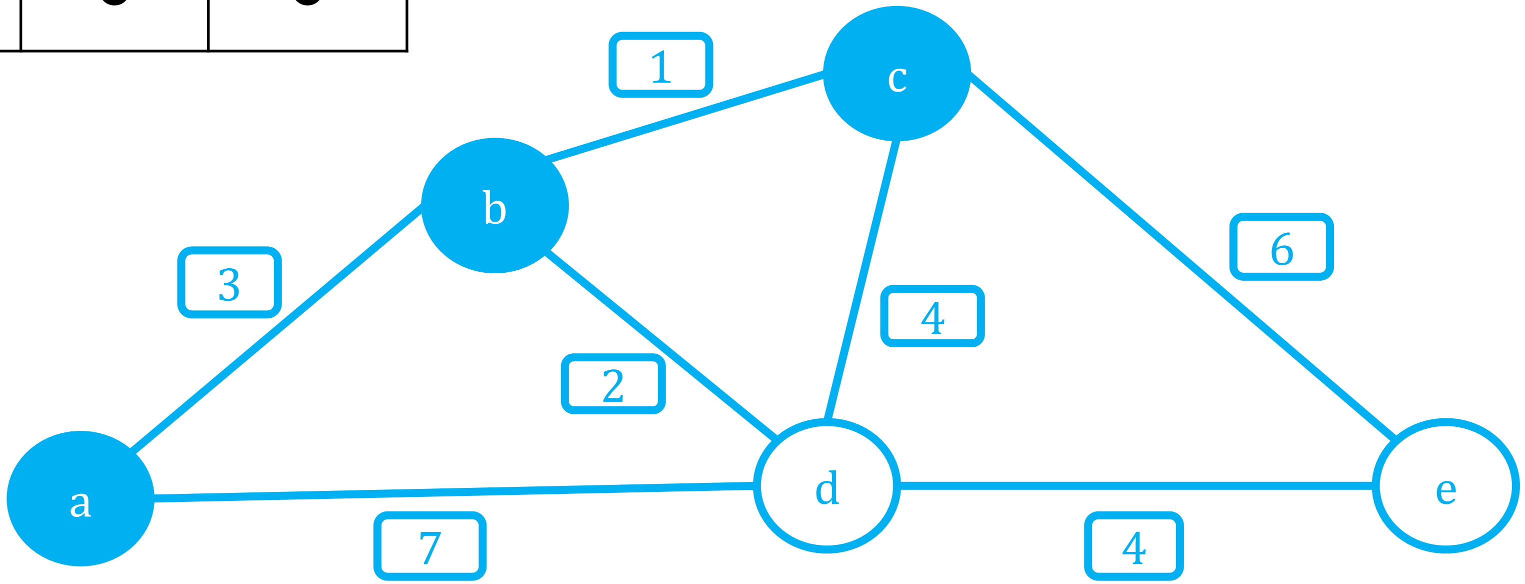
Полагаме h(c)=1;



Алгоритъм на Дейкстра [10/13]

	a	b	c	d	e
L	0	3	4	7	100
H	1	1	1	0	0

Съседните немаркирани върхове на c са d и e



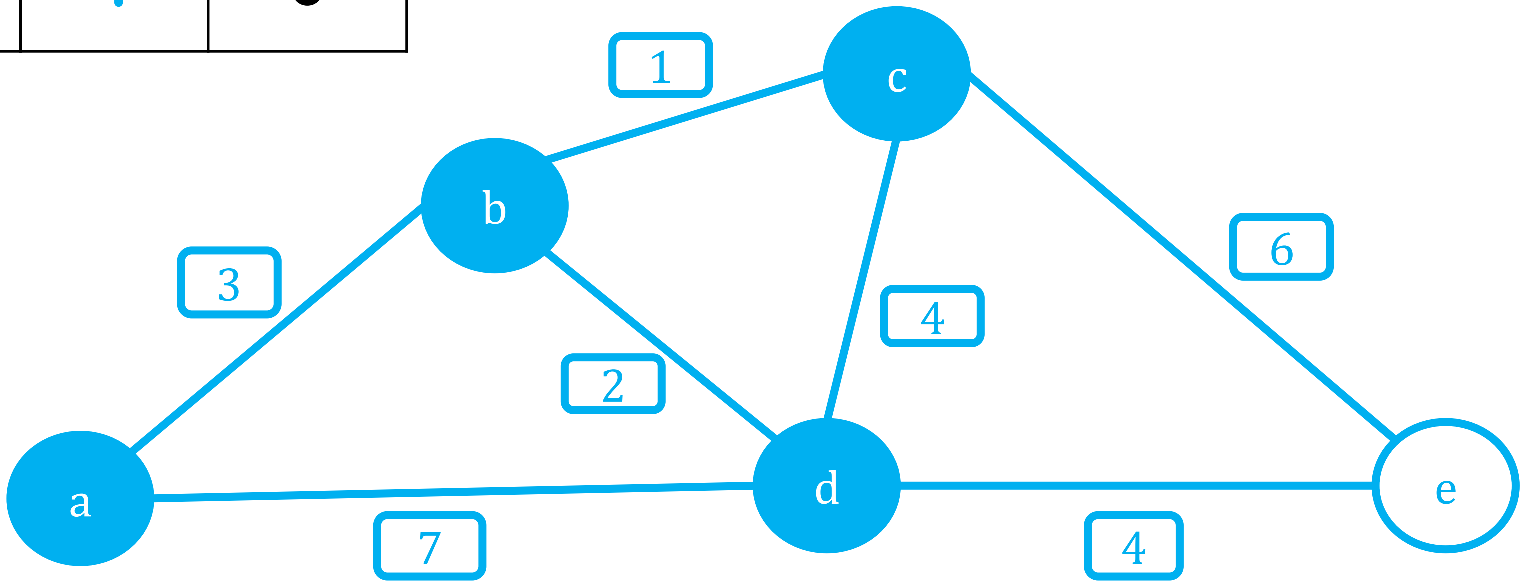
Алгоритъм на Дейкстра [11/13]

	a	b	c	d	e
L	0	3	4	5	100
H	1	1	1	1	0

Най-малката стойност е на l(d) и тя е 5.

Полагаме p=d и този връх го маркираме.

Полагаме h(d)=1;



Алгоритъм на Дейкстра [12/13]

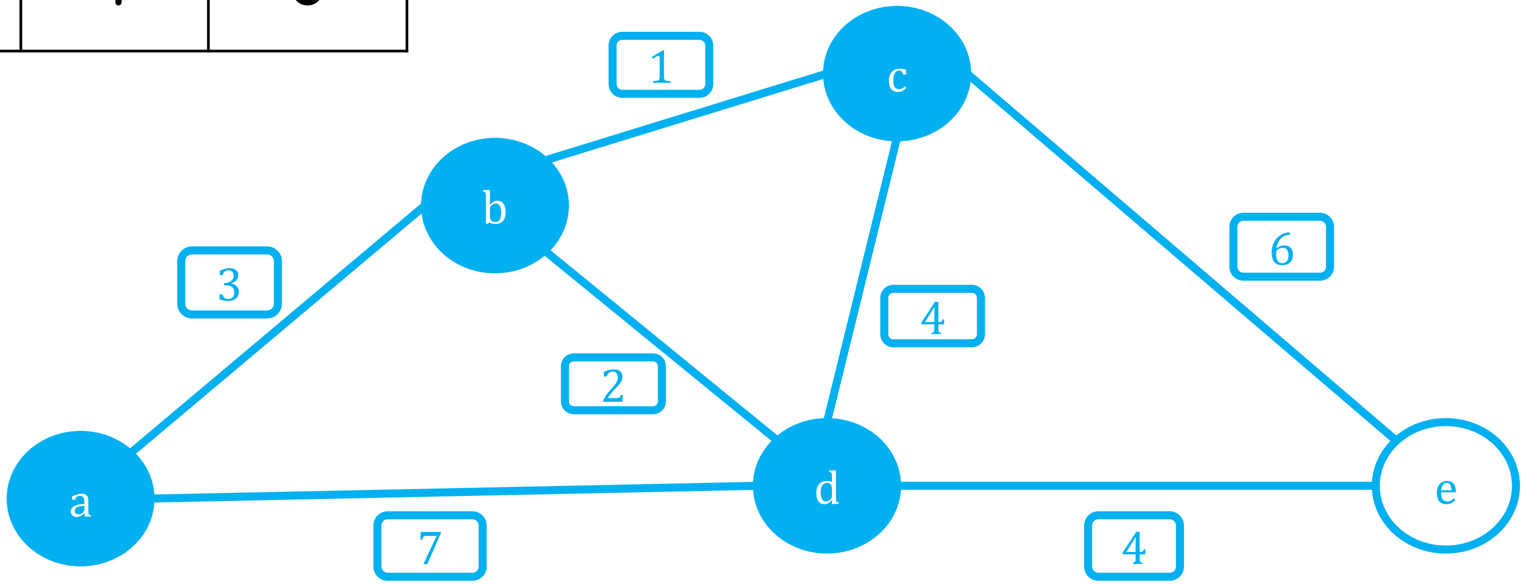
	a	b	c	d	e
L	0	3	4	5	9
H	1	1	1	1	0

Съседен, немаркиран връх на d е само е.

l(e)=min{l(e), l(d)+4}=min{10, 5+4} =9

Достигнахме u2=e;

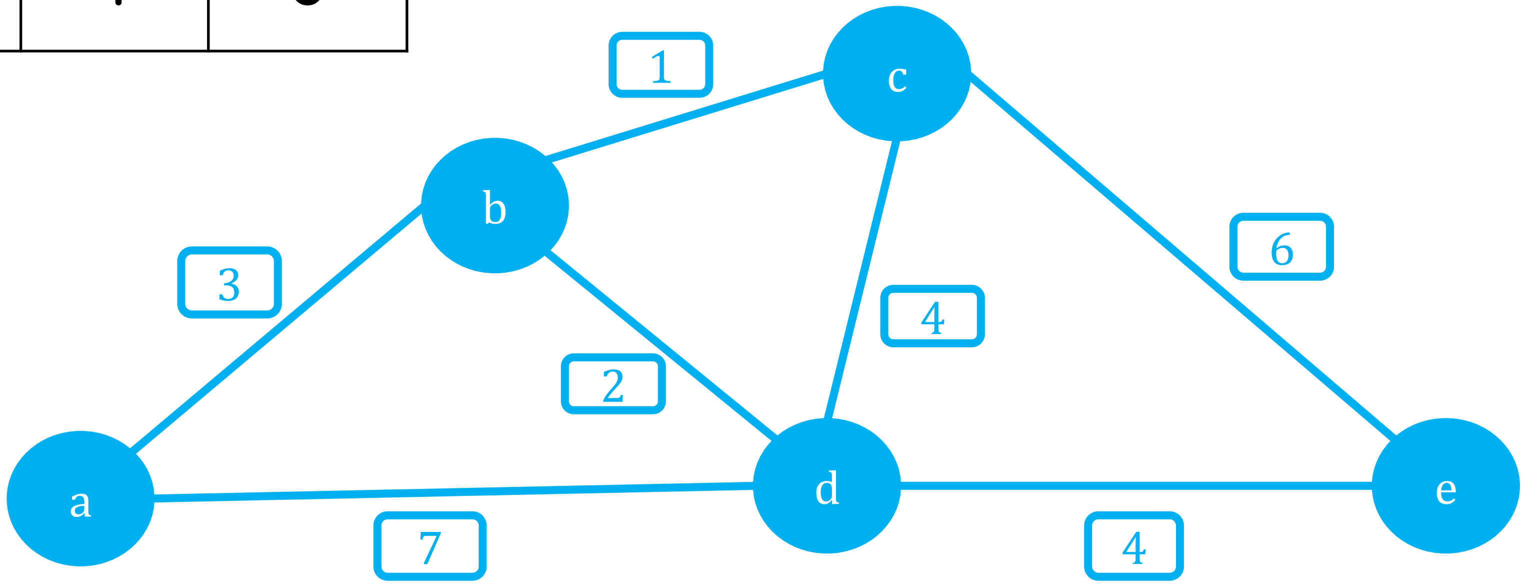
Край на алгоритъма.

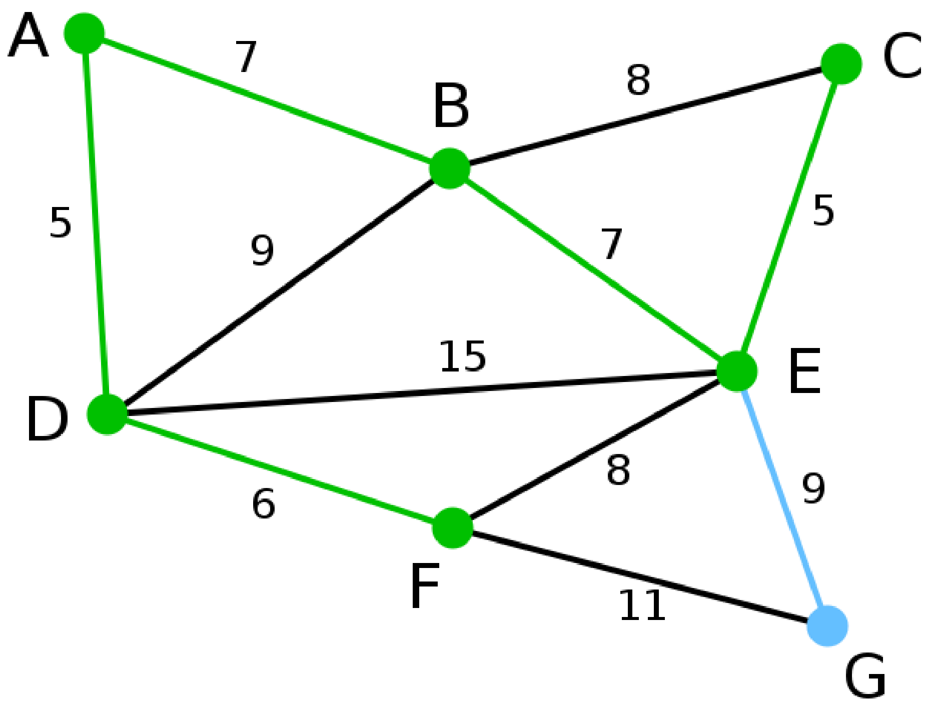


Алгоритъм на Дейкстра [13/13]

	a	b	c	d	e
L	0	3	4	5	9
H	1	1	1	1	0

Стойността на минималния път от връх *a* до връх *e* е 9.





Други алгоритми в графи

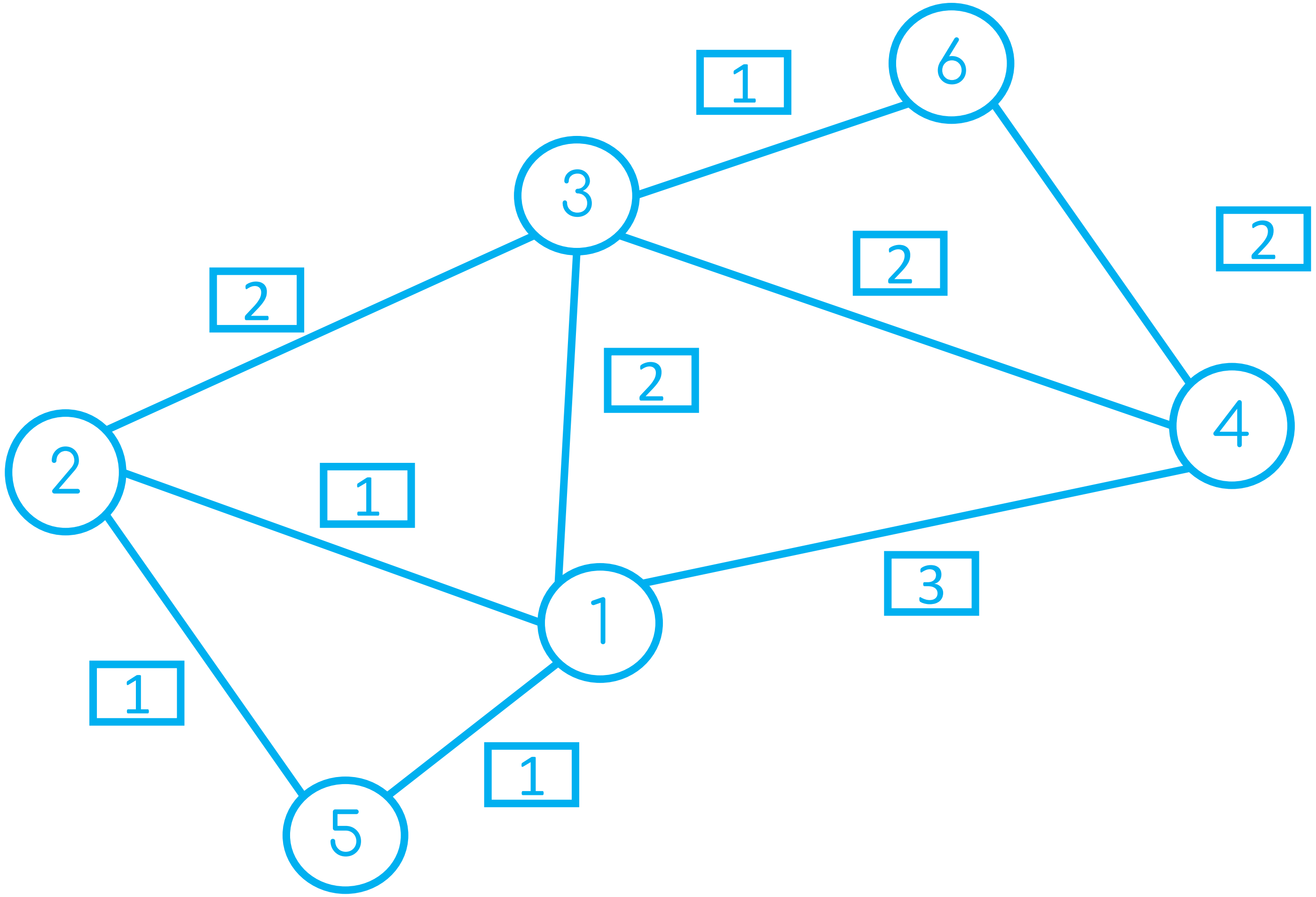
Алгоритъм на Прим

Задача

В община X има път между всеки две селища. Общинският съвет взел решение да асфалтира всички междуселищни пътища, но времето за изпълнение било малко, а финансите - недостатъчни. За целта решили да асфалтират само някои от тях, така, че да се стига от всяко селище до друго.

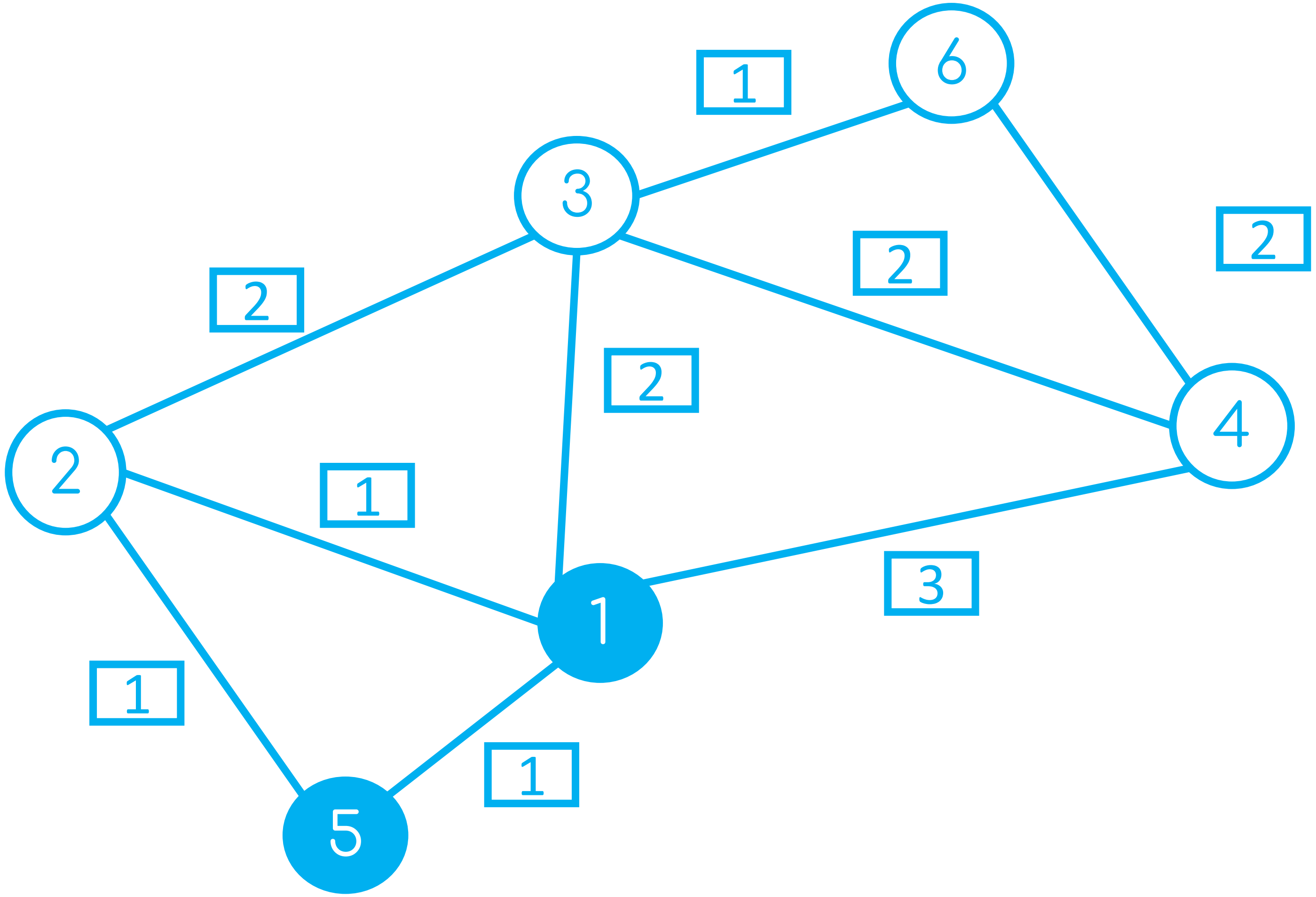
Алгоритъм на Прим [1/7]

Нека е даден граф с 6 върха и 9 ребра, със съответни тегла.



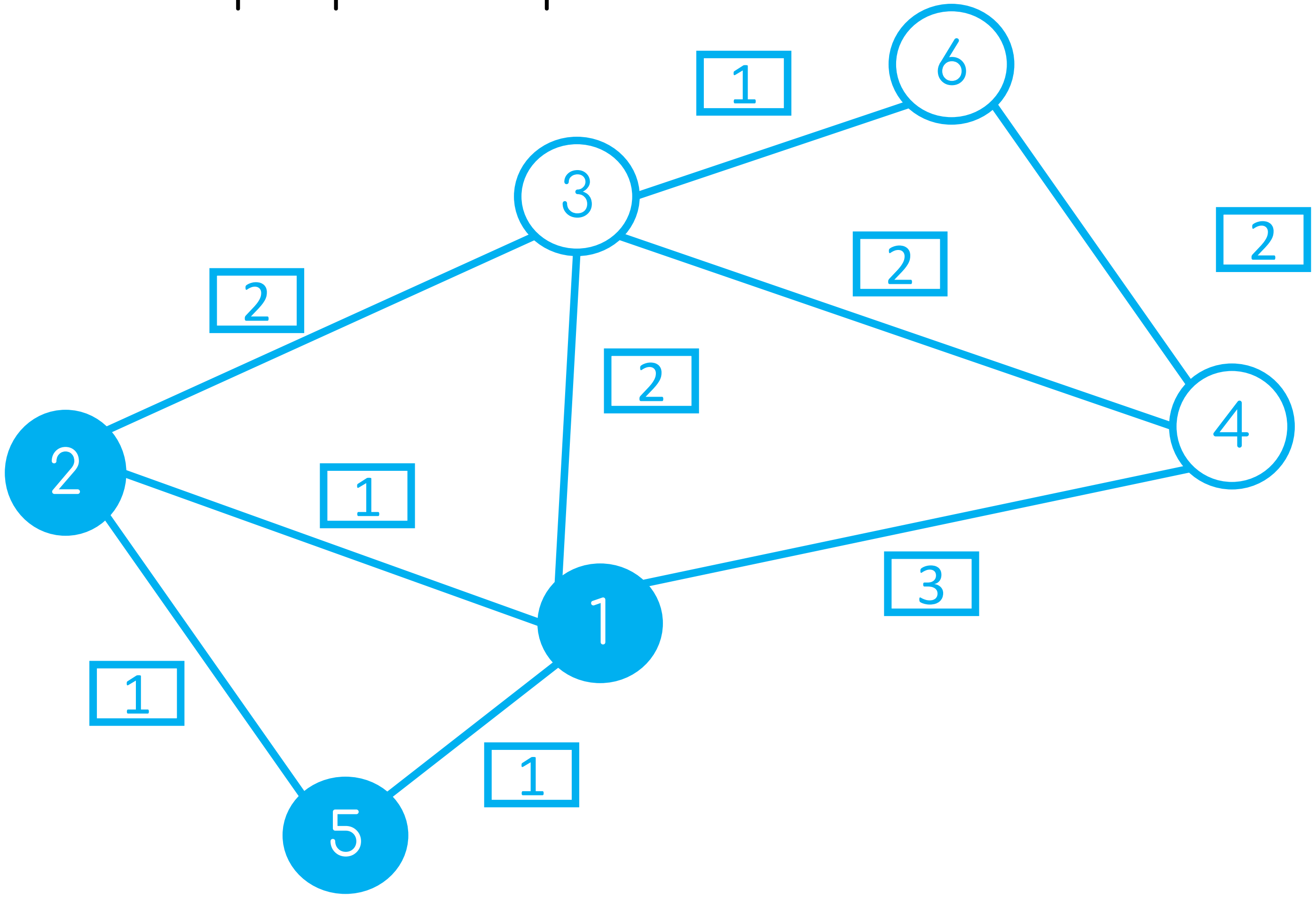
Алгоритъм на Прим [2/7]

Стъпка 1: Избираме един връх. Нека да е 1. Маркираме го. Търсим съседен на него, който не е маркиран и свързващото им ребро да е с минимално тегло. Това са 5 и 2. Нека изберем 5. Маркираме го.



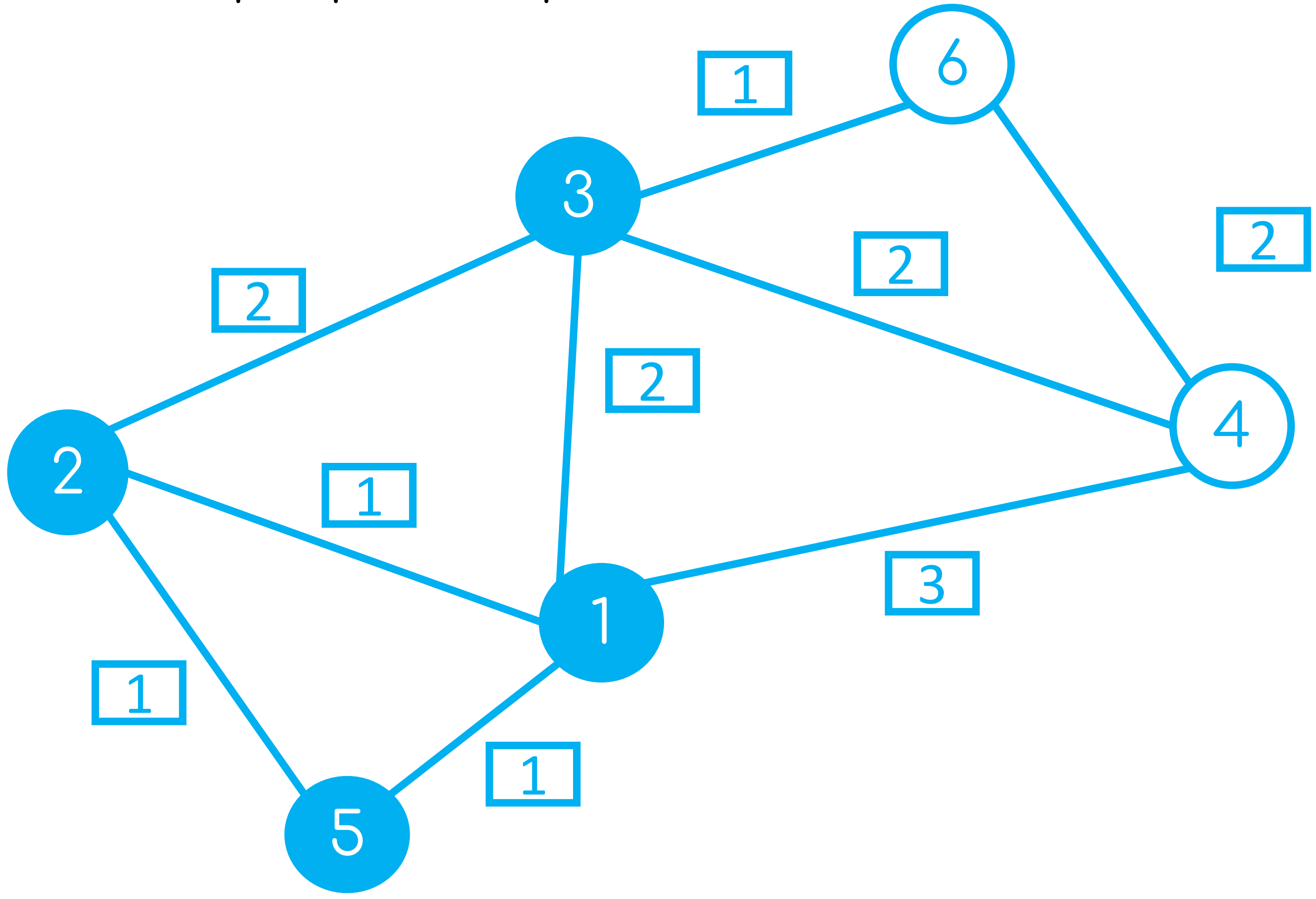
Алгоритъм на Прим [3/7]

Стъпка 2: Намираме следващото най-късо ребро, на което един от върховете е 1 или 5, а другият не е маркиран. Приемаме, че е (5,2). Маркираме не маркирания връх 2.



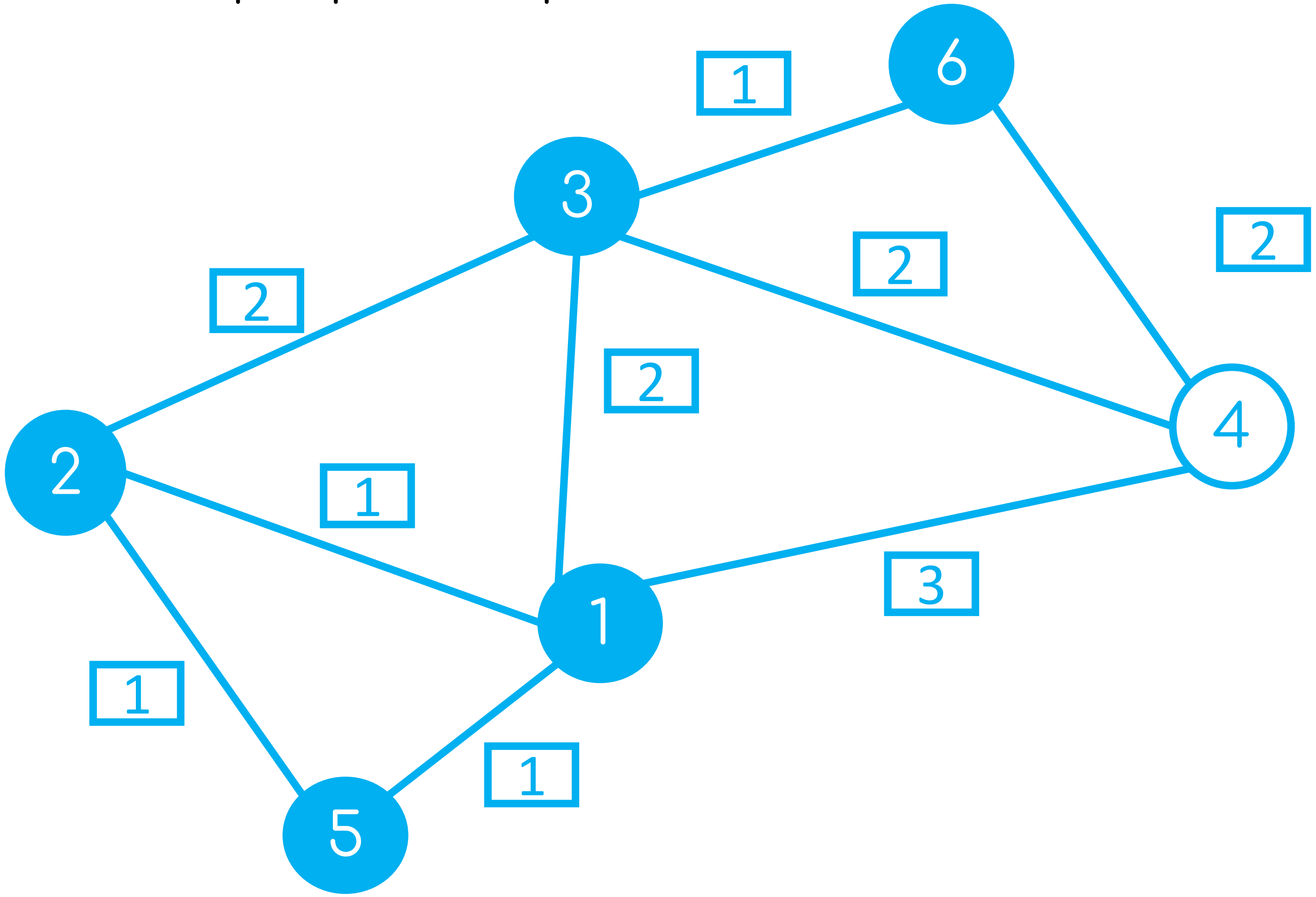
Алгоритъм на Прим [4/7]

Стъпка 3: Намираме следващото най-късо ребро, на което един от върховете е 1,2 или 5, а другият не е маркиран. Приемаме, че е (1,3). Маркираме не маркирания връх 3.



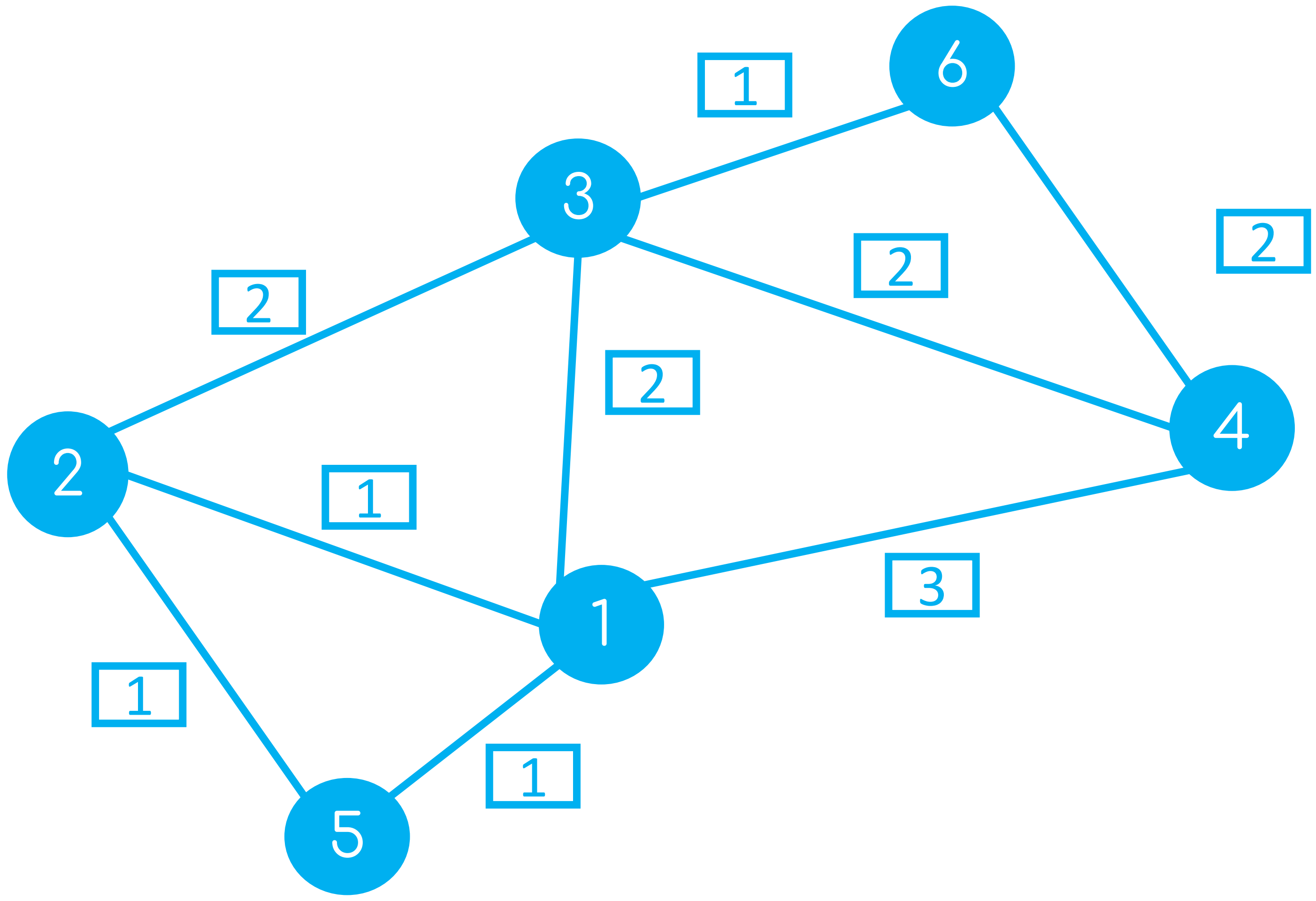
Алгоритъм на Прим [5/7]

Стъпка 4: Намираме следващото най-късо ребро, на което един от върховете е 1,2,3 или 5, а другият не е маркиран. Това е (3,6). Маркираме не маркирания връх 6.



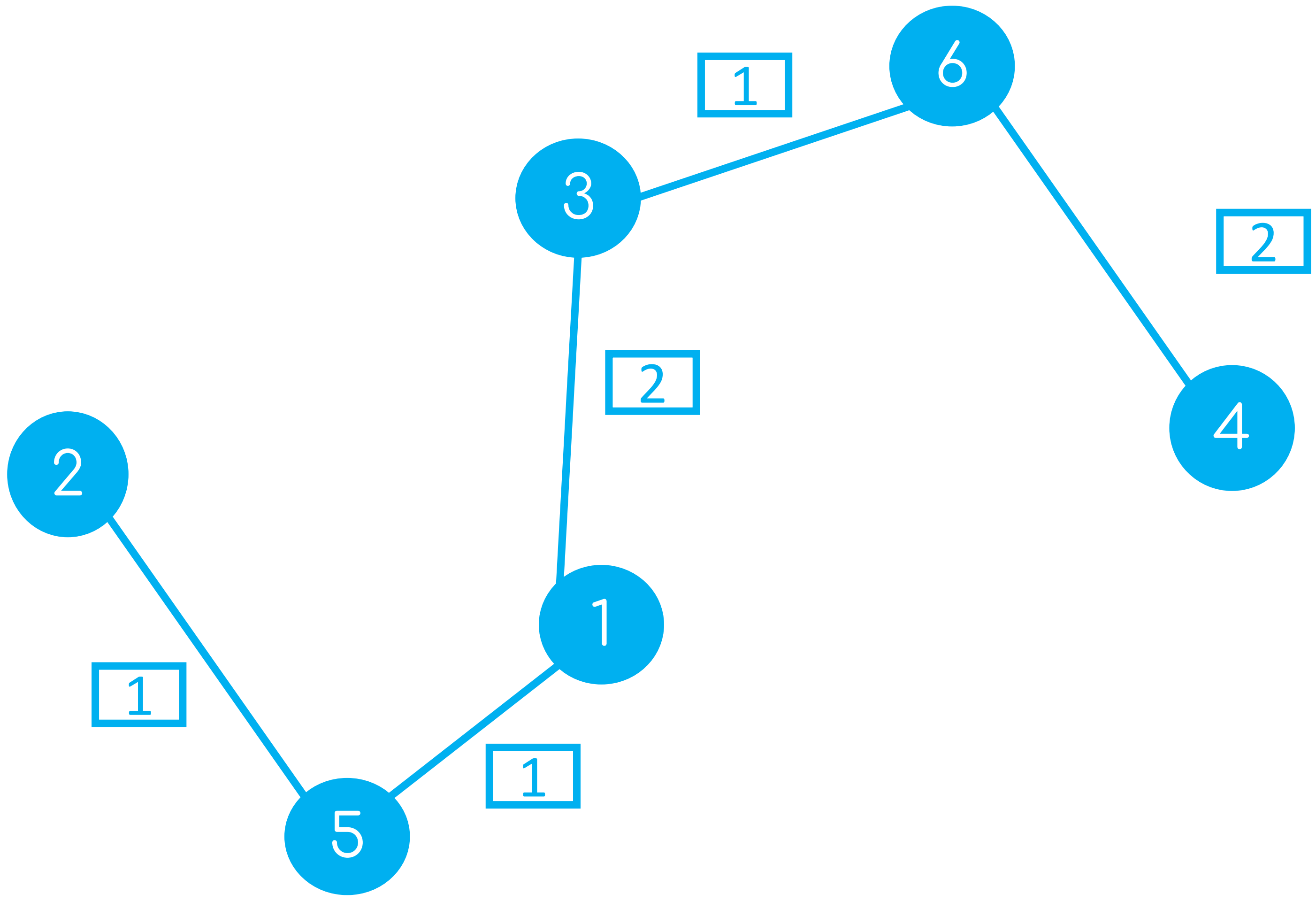
Алгоритъм на Прим [6/7]

Стъпка 5: Намираме следващото най-късо ребро, на което един от върховете е 1,2,3,5 или 6, а другият не е маркиран. Избираме който и да е от двата. Нека да е (6,4). Маркираме не маркирания връх 4.



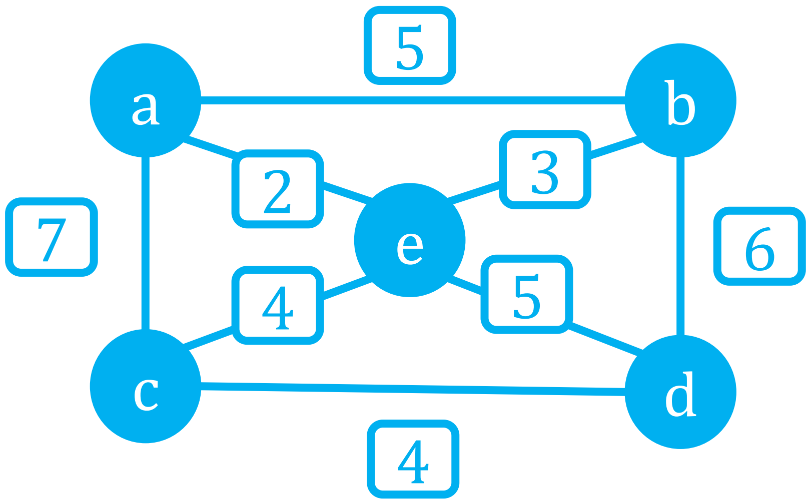
Алгоритъм на Прим [7/7]

Стъпка 6: Всички върхове са маркирани.
Край на алгоритъма.



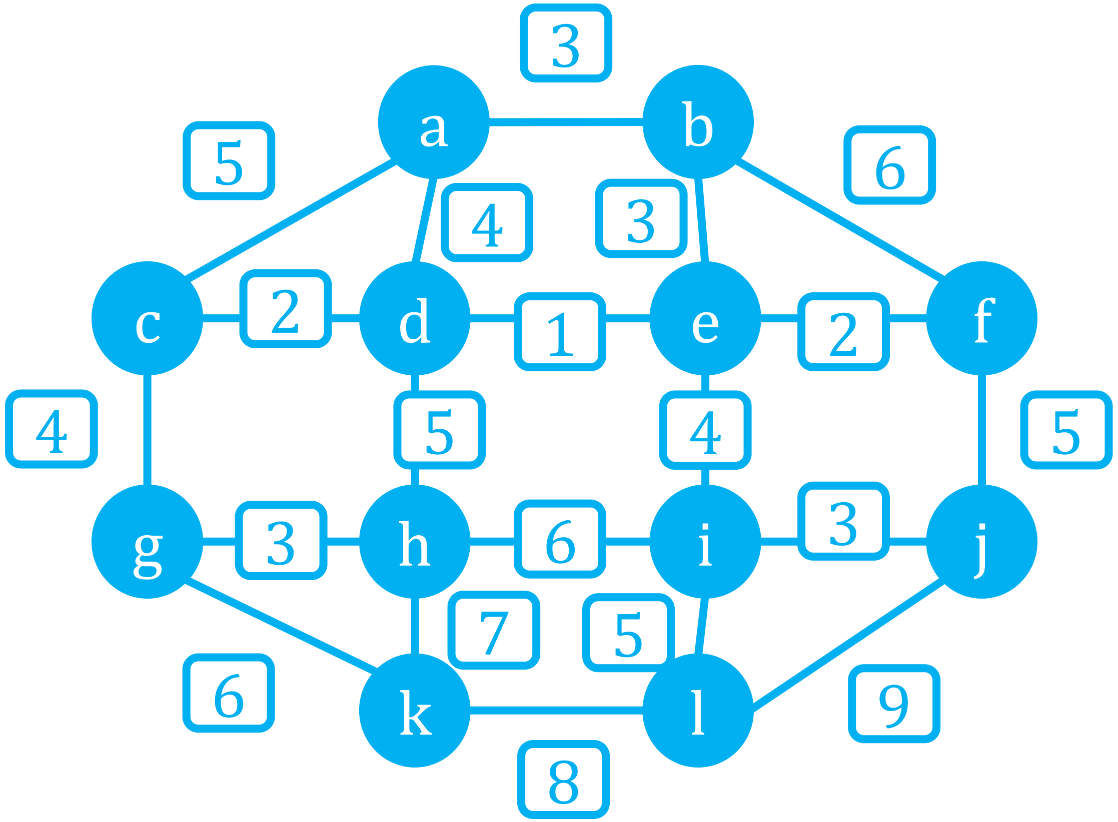
Упражнения: Prim's algorithm [1/2]

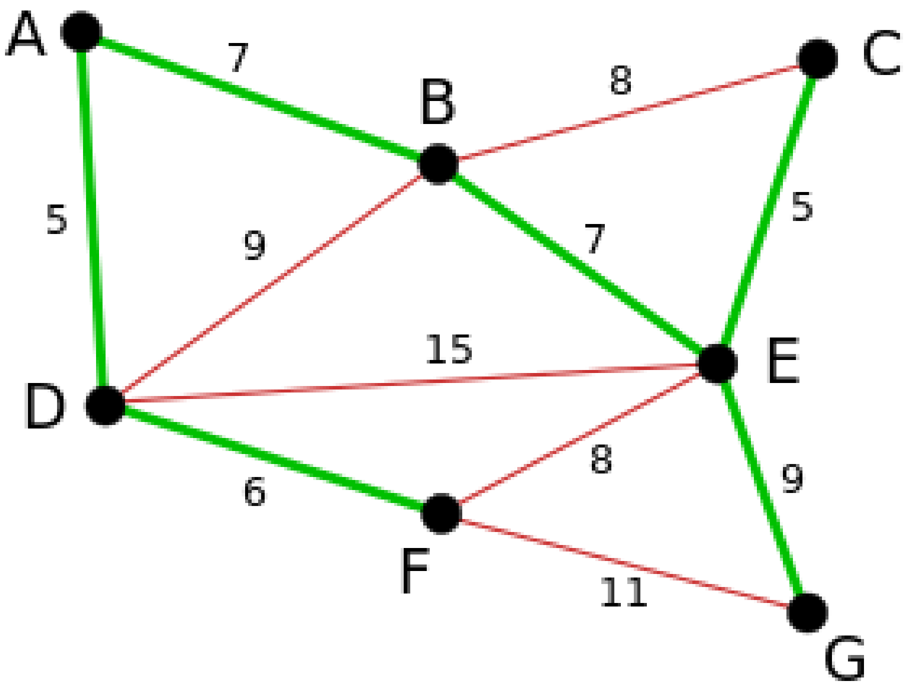
Приложете алгоритъма на Prim към дадения граф



Упражнения: Prim's algorithm [2/2]

Приложете алгоритъма на Prim към дадения граф





Други алгоритми в графи

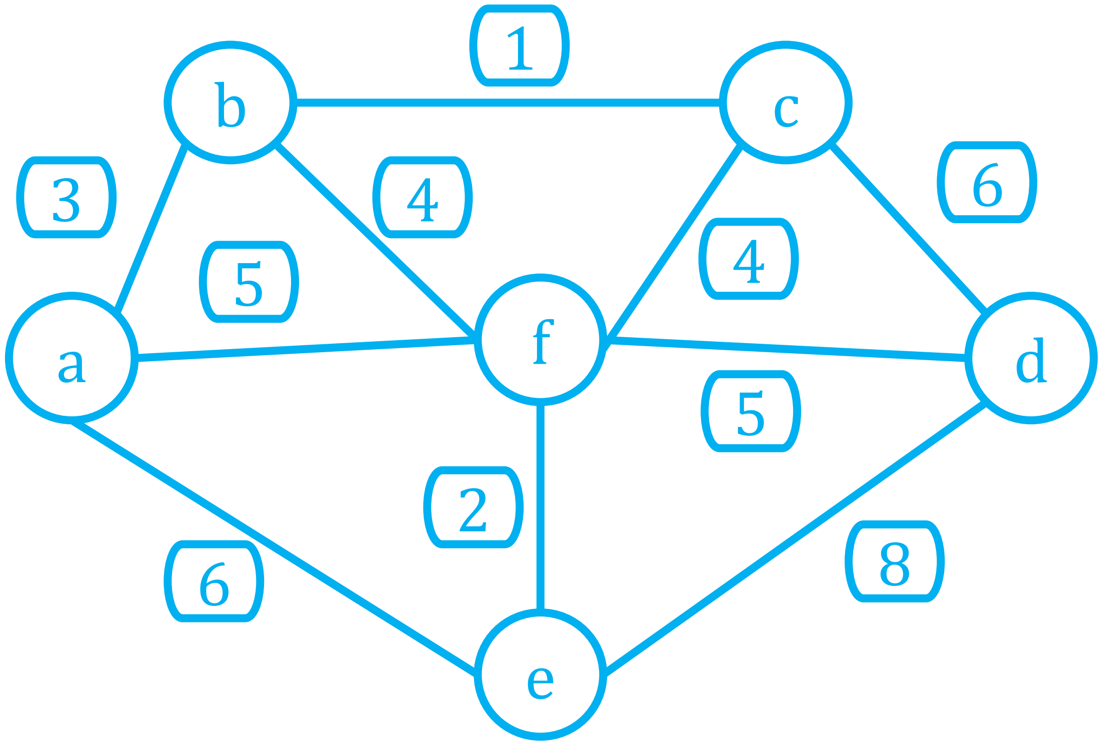
Алгоритъм на Крускал

Алгоритъм на Крускал [1/8]

Този алгоритъм реализира следната идея: Търси се минимално покриващо дърво в претеглен, свързан граф G = {V, E}, като ацикличен подграф с |V|-1 ребра, сумата от ребрата на който е минимална. В този случай дървото се разширява като подграфа е винаги ацикличен, но на междинните етапи не винаги е свързан.

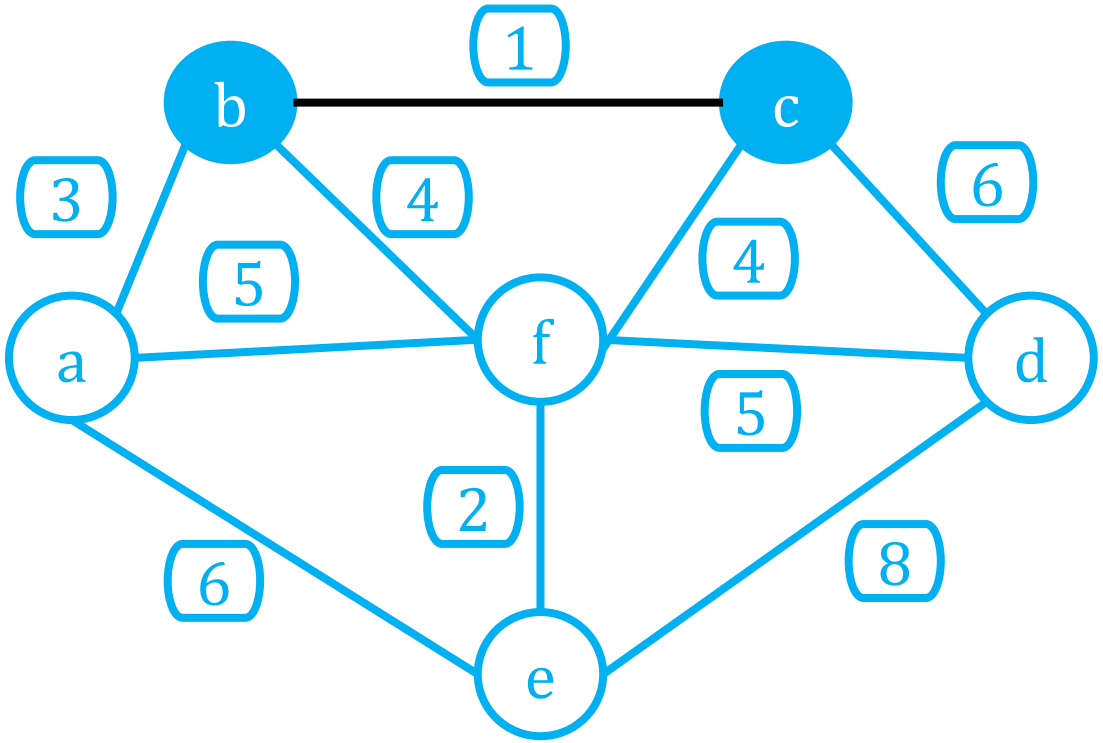
Алгоритъм на Кръскал [2/8]

Нека е даден граф с 6 върха и 10 ребра, със съответни тегла.



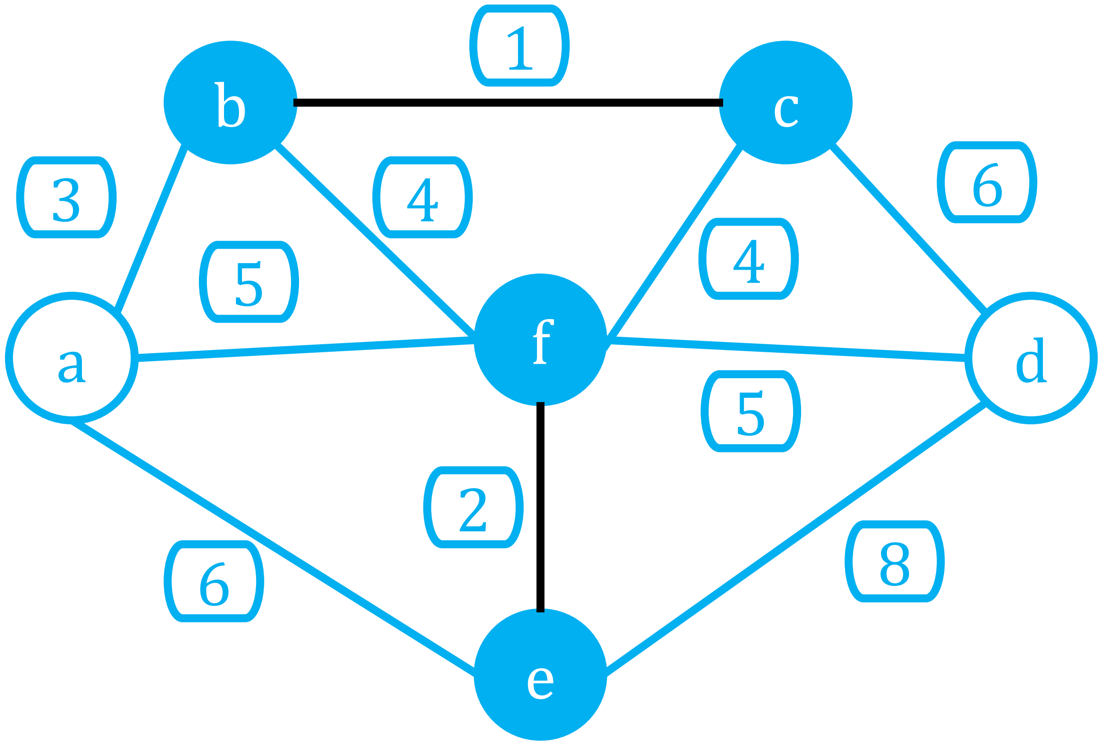
Алгоритъм на Крускал [3/8]

Стъпка 1. Избираме ребро с на-малко тегло. В случая това е реброто (b, c) с тегло 1. Маркираме го.



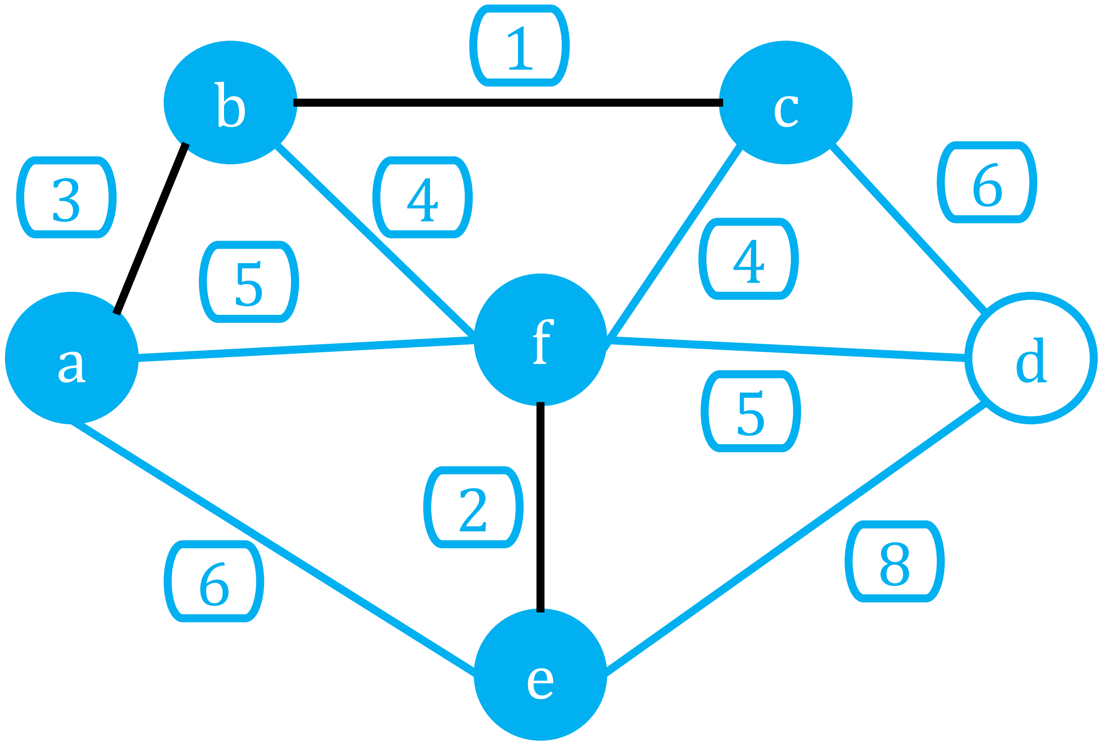
Алгоритъм на Крускал [4/8]

Стъпка 2. Избираме ребрата със следващото тегло, по-голямо от 1. Това е ребро (f,e) с тегло 2.



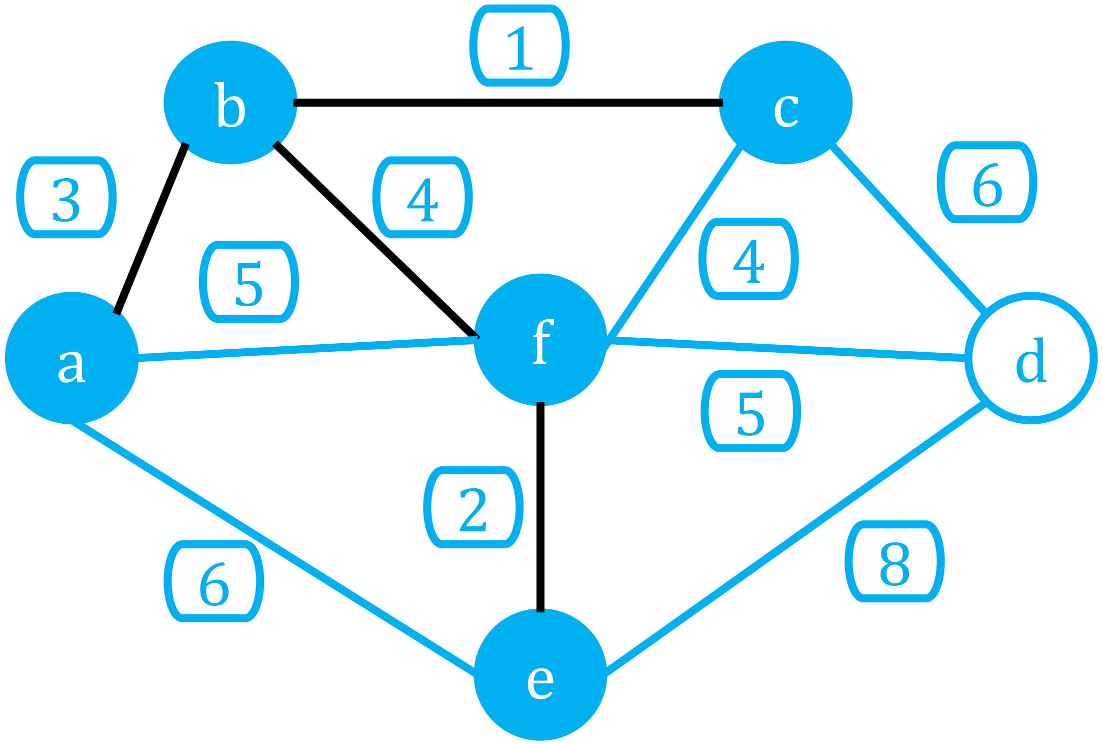
Алгоритъм на Крускал [5/8]

Стъпка 3. Избираме ребрата със следващото тегло, по-голямо от 2. Това е ребро (a,b) с тегло 3.



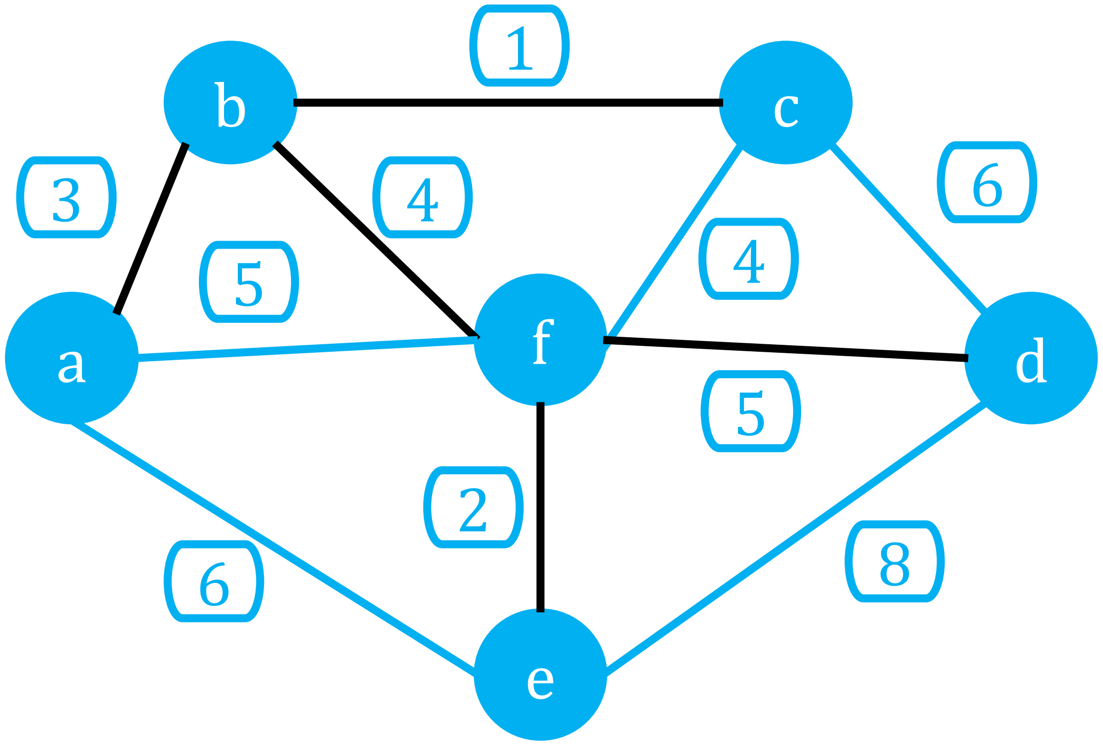
Алгоритъм на Крускал [6/8]

Стъпка 4. Избираме ребрата със следващото тегло, по-голямо от 3. Това е ребрата (b, f) и (c, f) с тегло 4. Търсим ацикличен подграф с |V|-1 ребра, сумата от ребрата на който е минимална . Това е реброто (b, f).



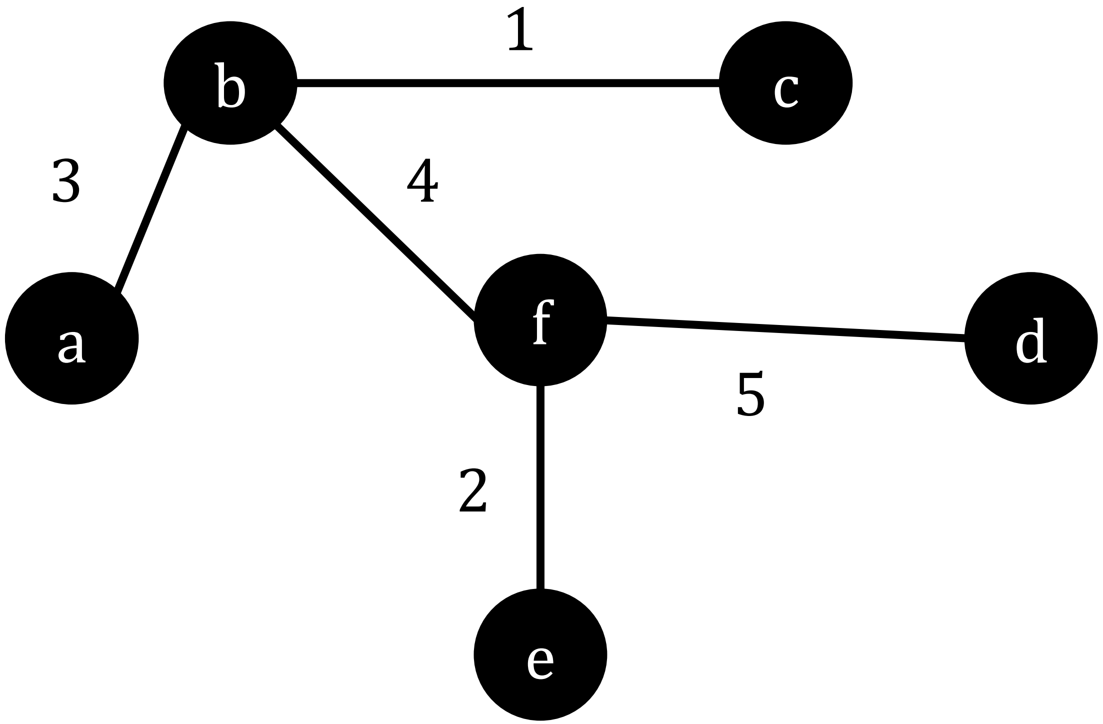
Алгоритъм на Крускал [7/8]

Стъпка 5. Избираме ребрата със следващото тегло, по-голямо от 4. Това са ребрата (a, f) и (d f) с тегло 5. Търсим ацикличен подграф с |V|-1 ребра, сумата от ребрата на който е минимална . Това е реброто (d, f).



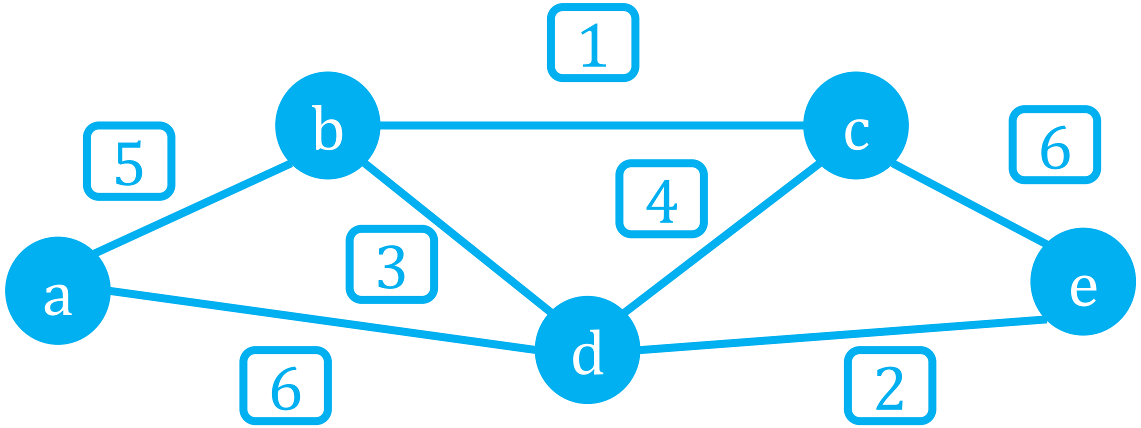
Алгоритъм на Крускал [8/8]

Стъпка 6. Край на алгоритъма.



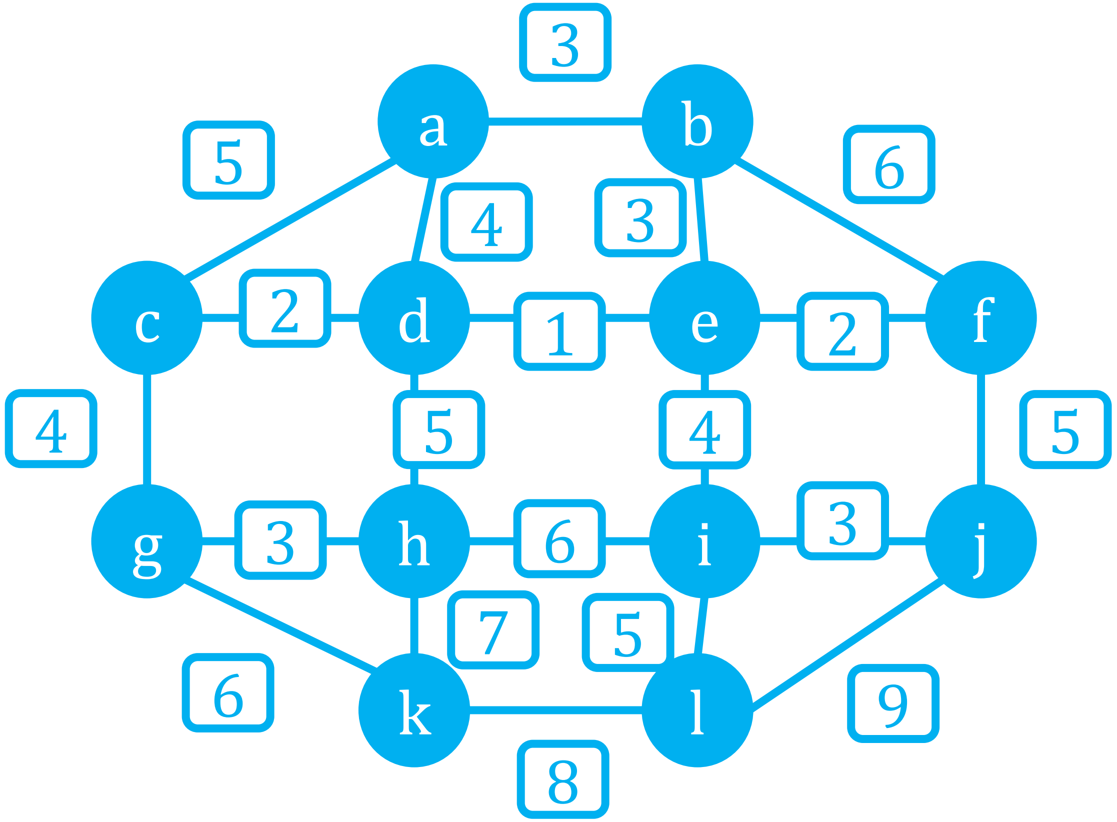
Упражнения: Kruskal's algorithm [1/2]

Приложете алгоритъма на Kruskal към дадения граф



Упражнения: Kruskal’s algorithm [2/2]

Приложете алгоритъма на Kruskal към дадения граф



Обобщение

- Представяне на графи
  - Списък на ребра
  - Матрица на свързаност
  - Списък на съседи
- Топологично сортиране
  - Подреждане на върховете на насочен, ацикличен граф
- Алгоритъм на Дейкстра
  - Намиране на минимален път в претеглен граф с неотрицателни тегла
- Други алгоритми върху графи
  - Алгоритъм на Прим
  - Алгоритъм на Крускал