ФАКУЛТЕТ ЗА ИНФОРМАТИЧКИ НАУКИ
И КОМПЈУТЕРСКО ИНЖЕНЕРСТВО

ГРАФОВИ

- втор дел -

АЛГОРИТМИ И ПОДАТОЧНИ СТРУКТУРИ

- предавања -

Детекција на циклус во граф

- (Ориентиран) Циклус во граф се дефинира како (ориентирана) непразна патека во која првото теме е исто со последното теме

- Прашање: Како да се детектира циклус во граф?

Детекција на циклус во граф

Можен пристап: Извршувај DFS теме по теме или додека не ги изминеш сите темиња или наидеш на теме кое е веќе посетено во моменталната патека која се разгледува

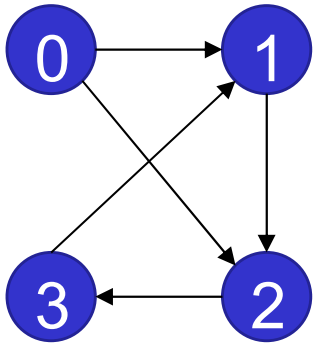
Детекција на циклус

1. Земи непосетено теме s, означи го како посетено, стави го на стек и означи дека е на стек (дел од моменталната патека)
2. Се додека стекот не е празен
2.1. Ако темето на врвот има сосед кој е посетен и е дел од моменталната патека КРАЈ: ИМА ЦИКЛУС
2.2. Ако темето на врвот има непосетен сосед означи го како посетен, стави го на стек и означи дека е на стек и врати се на 2
2.3. Извади го темето од врвот на стекот, означи дека не е на стек и врати се на 2
3. Ако има уште непосетени темиња во графот, врати се на 1, инаку НЕМА ЦИКЛУС

Детекција на циклус во граф

Детекција на циклус

1. Земи непосетено теме s, означи го како посетено, стави го на стек и означи дека е на стек (дел од моменталната патека)
2. Се додека стекот не е празен
2.1. Ако темето на врвот има сосед кој е посетен и е дел од моменталната патека КРАЈ: ИМА ЦИКЛУС
2.2. Ако темето на врвот има непосетен сосед означи го како посетен, стави го на стек и означи дека е на стек и врати се на 2
2.3. Извади го темето од врвот на стекот, означи дека не е на стек и врати се на 2
3. Ако има уште непосетени темиња во графот, врати се на 1, инаку НЕМА ЦИКЛУС



посетено

0	1	2	3

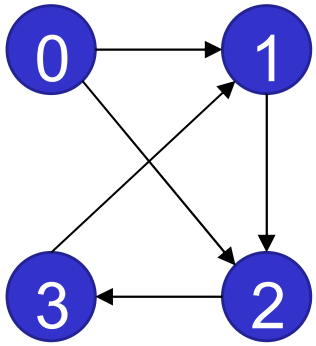
на стек

0	1	2	3

Детекција на циклус во граф

Детекција на циклус

1. Земи непосетено теме s, означи го како посетено, стави го на стек и означи дека е на стек (дел од моменталната патека)
2. Се додека стекот не е празен
2.1. Ако темето на врвот има сосед кој е посетен и е дел од моменталната патека КРАЈ: ИМА ЦИКЛУС
2.2. Ако темето на врвот има непосетен сосед означи го како посетен, стави го на стек и означи дека е на стек и врати се на 2
2.3. Извади го темето од врвот на стекот, означи дека не е на стек и врати се на 2
3. Ако има уште непосетени темиња во графот, врати се на 1, инаку НЕМА ЦИКЛУС



посетено

0	1	2	3
T			

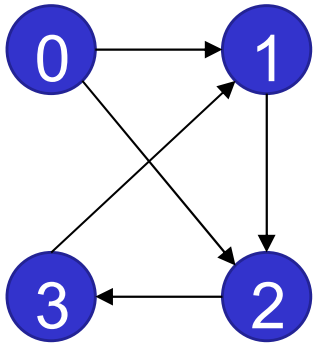
на стек

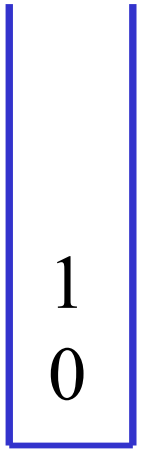
0	1	2	3
T			

Детекција на циклус во граф

Детекција на циклус

1. Земи непосетено теме s, означи го како посетено, стави го на стек и означи дека е на стек (дел од моменталната патека)
2. Се додека стекот не е празен
2.1. Ако темето на врвот има сосед кој е посетен и е дел од моменталната патека КРАЈ: ИМА ЦИКЛУС
2.2. Ако темето на врвот има непосетен сосед означи го како посетен, стави го на стек и означи дека е на стек и врати се на 2
2.3. Извади го темето од врвот на стекот, означи дека не е на стек и врати се на 2
3. Ако има уште непосетени темиња во графот, врати се на 1, инаку НЕМА ЦИКЛУС





посетено

0	1	2	3
T	T		

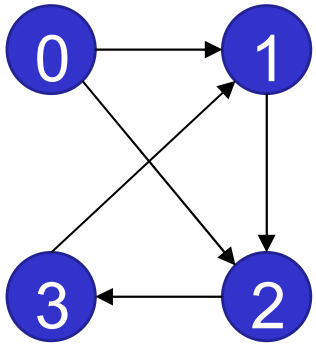
на стек

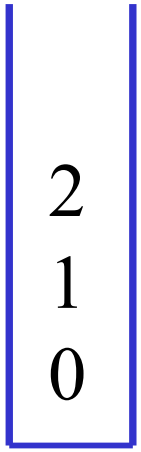
0	1	2	3
T	T		

Детекција на циклус во граф

Детекција на циклус

1. Земи непосетено теме s, означи го како посетено, стави го на стек и означи дека е на стек (дел од моменталната патека)
2. Се додека стекот не е празен
2.1. Ако темето на врвот има сосед кој е посетен и е дел од моменталната патека КРАЈ: ИМА ЦИКЛУС
2.2. Ако темето на врвот има непосетен сосед означи го како посетен, стави го на стек и означи дека е на стек и врати се на 2
2.3. Извади го темето од врвот на стекот, означи дека не е на стек и врати се на 2
3. Ако има уште непосетени темиња во графот, врати се на 1, инаку НЕМА ЦИКЛУС





посетено

0	1	2	3
T	T	T	

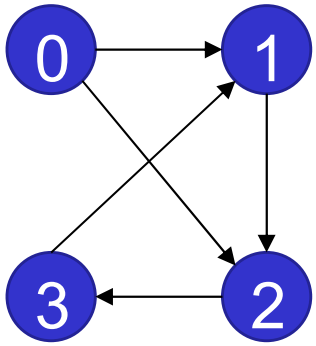
на стек

0	1	2	3
T	T	T	

Детекција на циклус во граф

Детекција на циклус

1. Земи непосетено теме s, означи го како посетено, стави го на стек и означи дека е на стек (дел од моменталната патека)
2. Се додека стекот не е празен
2.1. Ако темето на врвот има сосед кој е посетен и е дел од моменталната патека КРАЈ: ИМА ЦИКЛУС
2.2. Ако темето на врвот има непосетен сосед означи го како посетен, стави го на стек и означи дека е на стек и врати се на 2
2.3. Извади го темето од врвот на стекот, означи дека не е на стек и врати се на 2
3. Ако има уште непосетени темиња во графот, врати се на 1, инаку НЕМА ЦИКЛУС



3
2
1
0

посетено

0	1	2	3
T	T	T	T

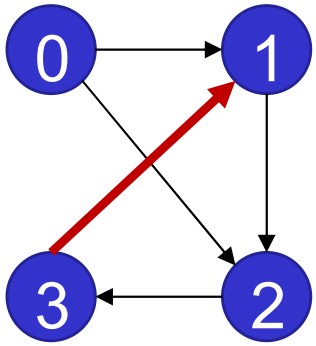
на стек

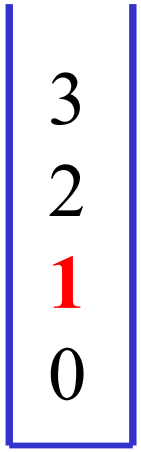
0	1	2	3
T	T	T	T

Детекција на циклус во граф

Детекција на циклус

1. Земи непосетено теме s, означи го како посетено, стави го на стек и означи дека е на стек (дел од моменталната патека)
2. Се додека стекот не е празен
2.1. Ако темето на врвот има сосед кој е посетен и е дел од моменталната патека КРАЈ: ИМА ЦИКЛУС
2.2. Ако темето на врвот има непосетен сосед означи го како посетен, стави го на стек и означи дека е на стек и врати се на 2
2.3. Извади го темето од врвот на стекот, означи дека не е на стек и врати се на 2
3. Ако има уште непосетени темиња во графот, врати се на 1, инаку НЕМА ЦИКЛУС





посетено

0	1	2	3
T	T	T	T

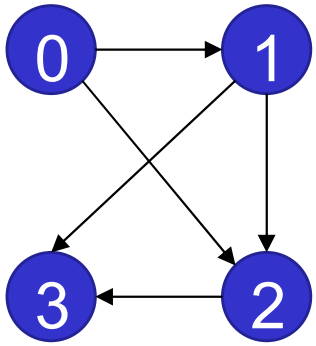
на стек

0	1	2	3
T	T	T	T

Детекција на циклус во граф

Детекција на циклус

1. Земи непосетено теме s, означи го како посетено, стави го на стек и означи дека е на стек (дел од моменталната патека)
2. Се додека стекот не е празен
2.1. Ако темето на врвот има сосед кој е посетен и е дел од моменталната патека КРАЈ: ИМА ЦИКЛУС
2.2. Ако темето на врвот има непосетен сосед означи го како посетен, стави го на стек и означи дека е на стек и врати се на 2
2.3. Извади го темето од врвот на стекот, означи дека не е на стек и врати се на 2
3. Ако има уште непосетени темиња во графот, врати се на 1, инаку НЕМА ЦИКЛУС



посетено

0	1	2	3
T			

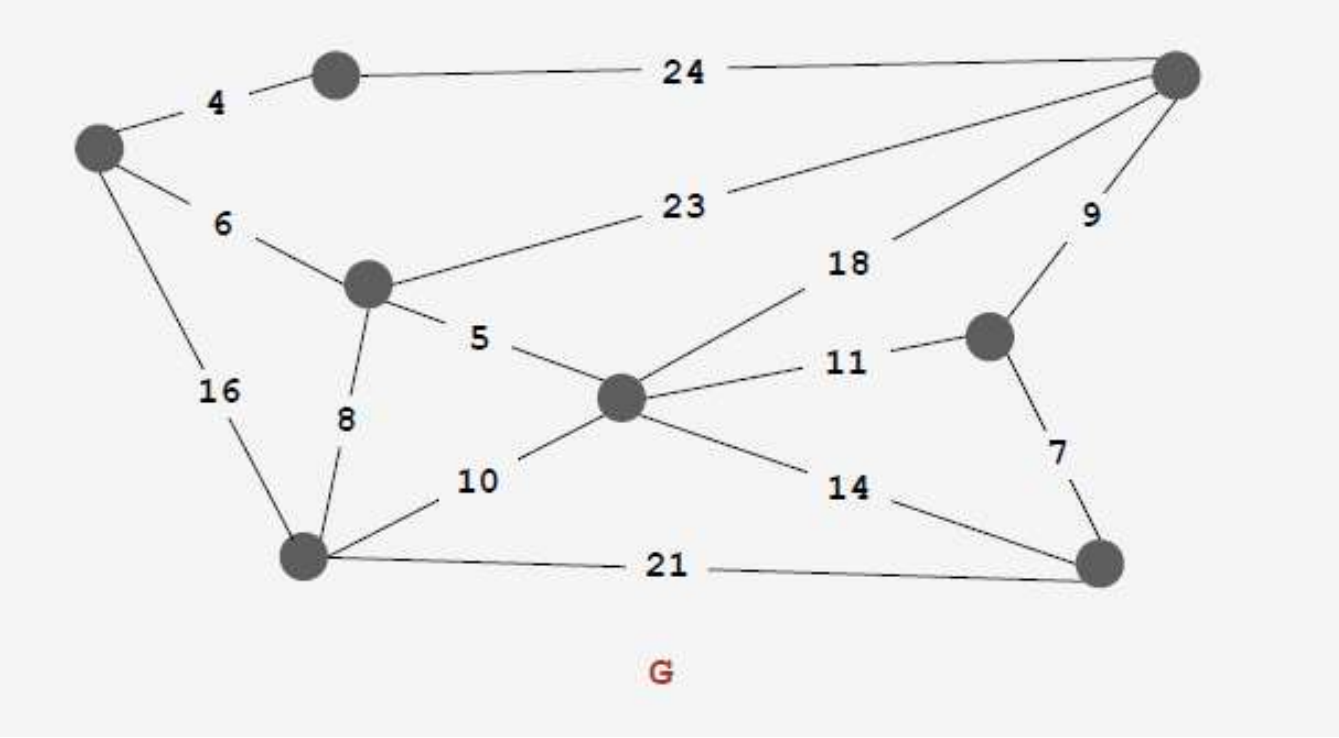
на стек

0	1	2	3
T			

Минимално распнувачко дрво (Minimum Spannig Tree - MST)

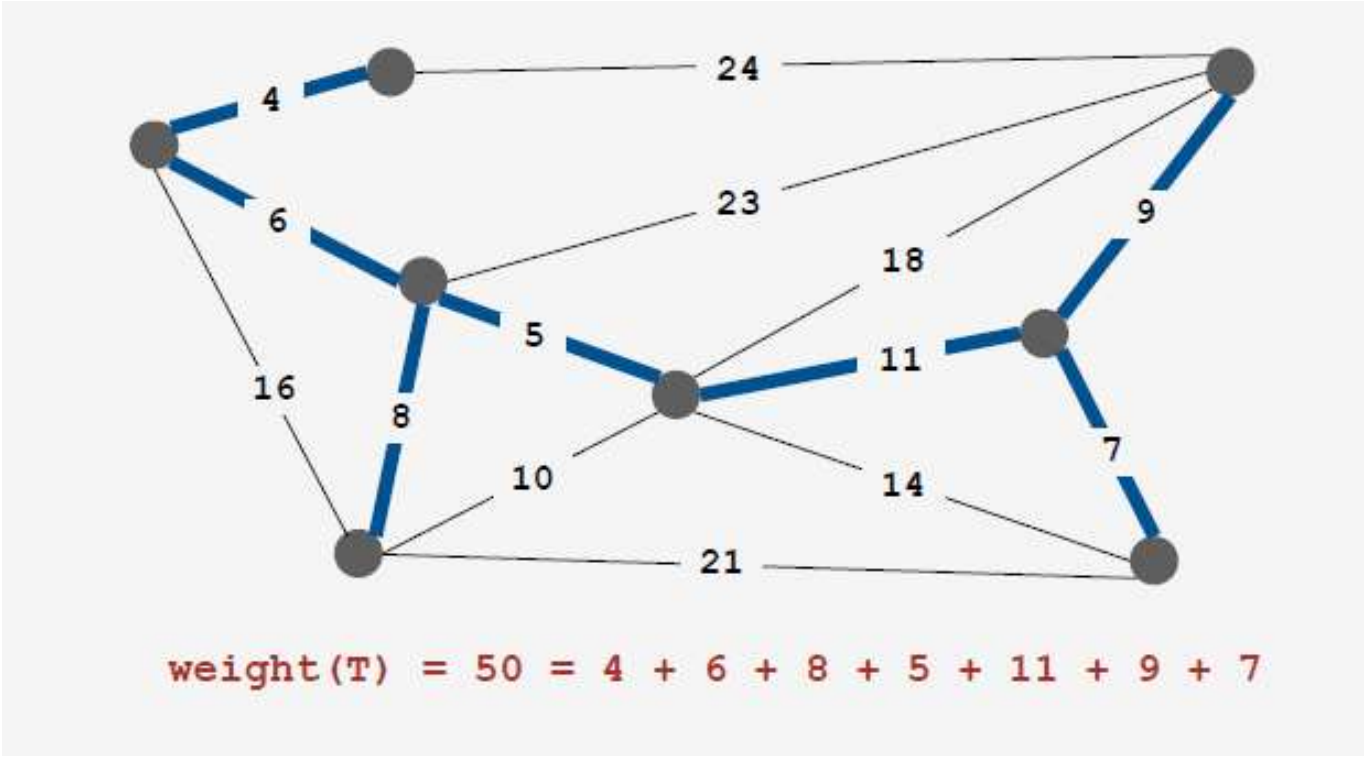
Дадено: Неориентиран граф G со позитивни тежини на неговите ребра (графот е поврзан).

Цел: Најди го минималното множество од ребра (збир од нивните тежини) кое што ги поврзува сите темиња



MST – Brute force

Brute force: Генерирај ги сите распнувачки дрва
- Проблем 1: Не е лесно за имплементација
- Проблем 2: Такви дрва има премногу



MST - примена

- Дизајнирање на мрежи (телефонски, електрични, компјутерски)

- Апроксимативни алгоритми за тешки проблеми (TSP, Steiner Tree)

- Автоматско конфигурирање на Ethernet мрежи да се избегнат Циклуси

- Анализа на кластери

- Пронаоѓање на патишта во сателитски слики

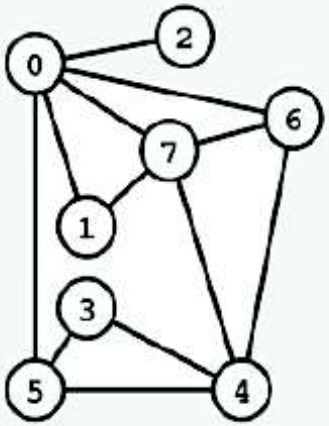
MST – Крускалов алгоритам

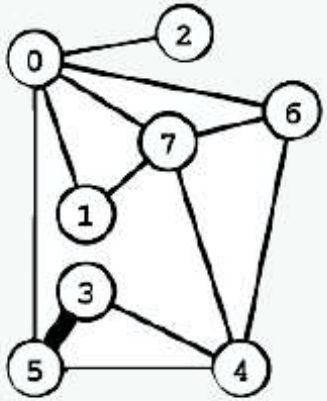
Разгледувај ги ребрата во растечки редослед, во однос на нивната тежина. Додај го реброто во решението само доколку неговото додавање не создава циклус.

Алчен алгоритам

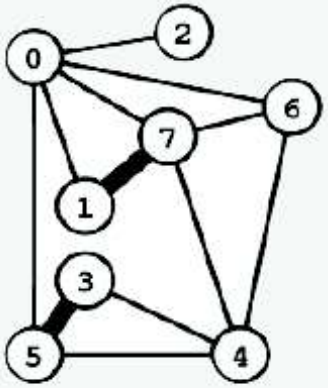
Крускалов алгоритам

0	1	2	3	4	5	6	7

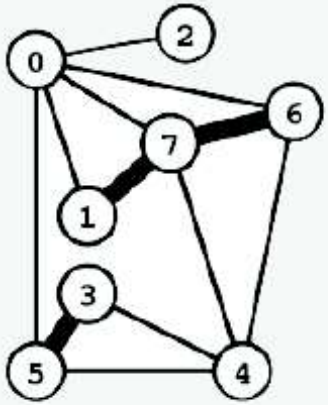




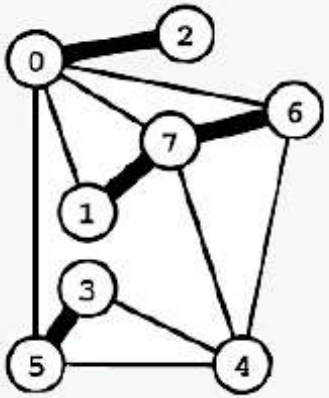
3-5



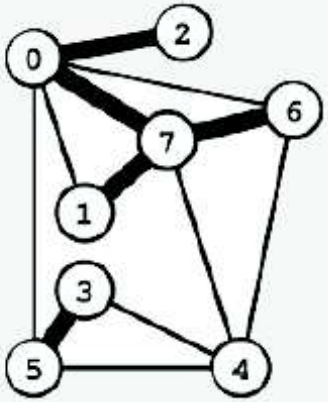
1-7



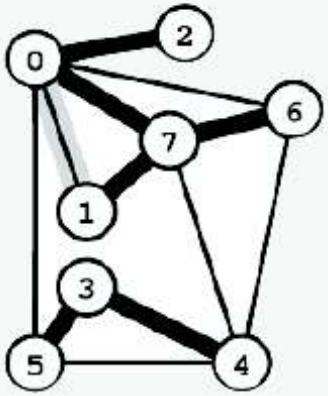
6-7



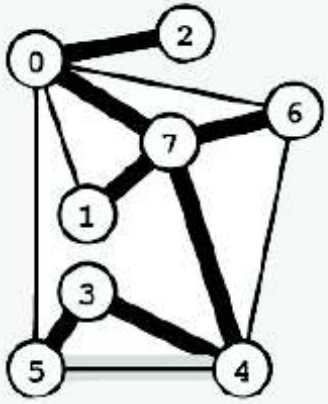
0-2



0-7



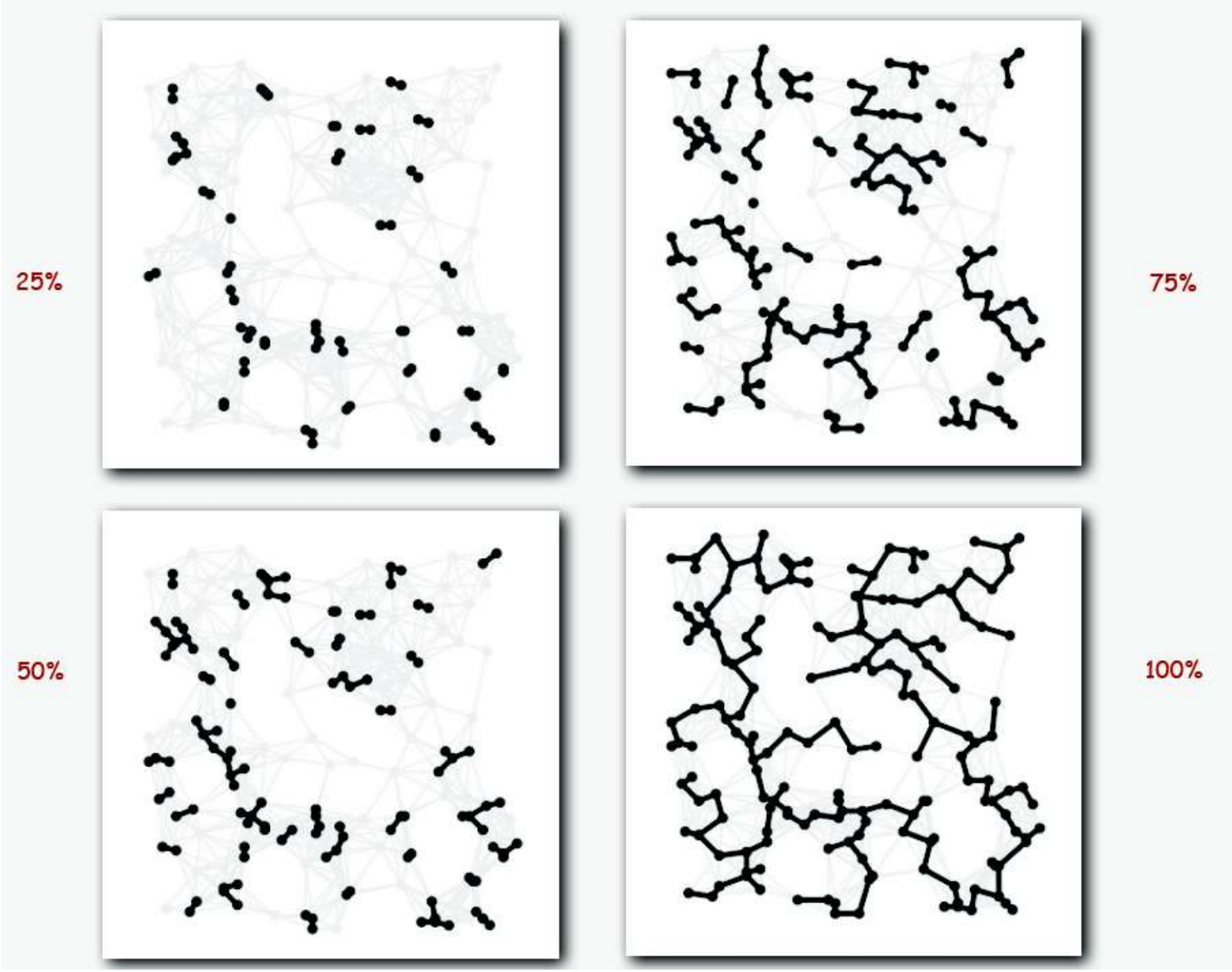
0-1 3-4



4-5 4-7

3-5 0.18
1-7 0.21
6-7 0.25
0-2 0.29
0-7 0.31
0-1 0.32
3-4 0.34
4-5 0.40
4-7 0.46
0-6 0.51
4-6 0.51
0-5 0.60

Крускалов алгоритам

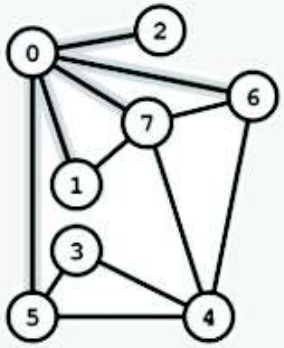


MST – Примов алгоритам

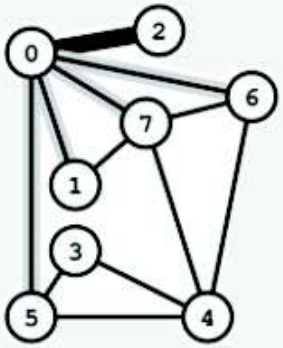
Започни од било кое теме s и од него расти дрво T.
Во секој чекор додади ребро во T кое што има точно едно теме во T, а ако има повеќе такви, додади го тоа со најмала тежина.

Алчен алгоритам

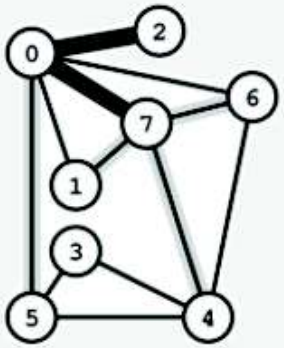
Примов алгоритам



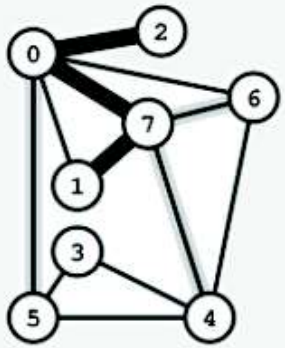
0-2 0-7 0-1 0-6 0-5



0-7 0-1 0-6 0-5

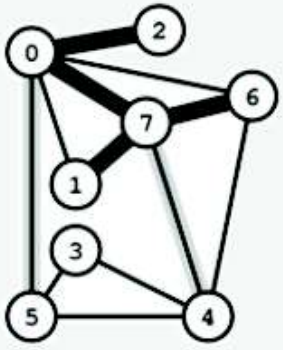


7-1 7-6 0-1 7-4 0-6 0-5

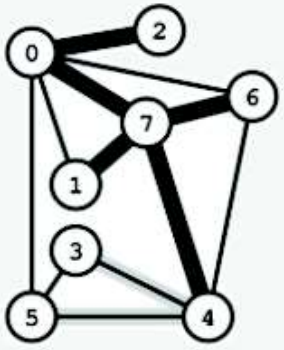


7-6 7-4 0-6 0-5

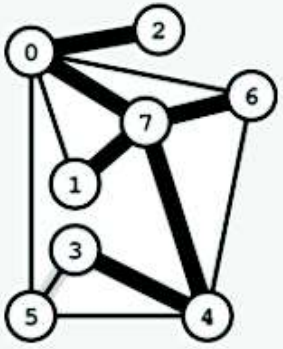
ребрата со точно едно теме во T, сортирани по нивната тежина



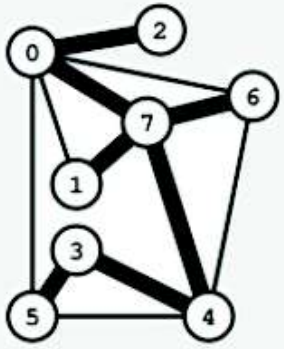
7-4 0-5 6-4



4-3 4-5 0-5

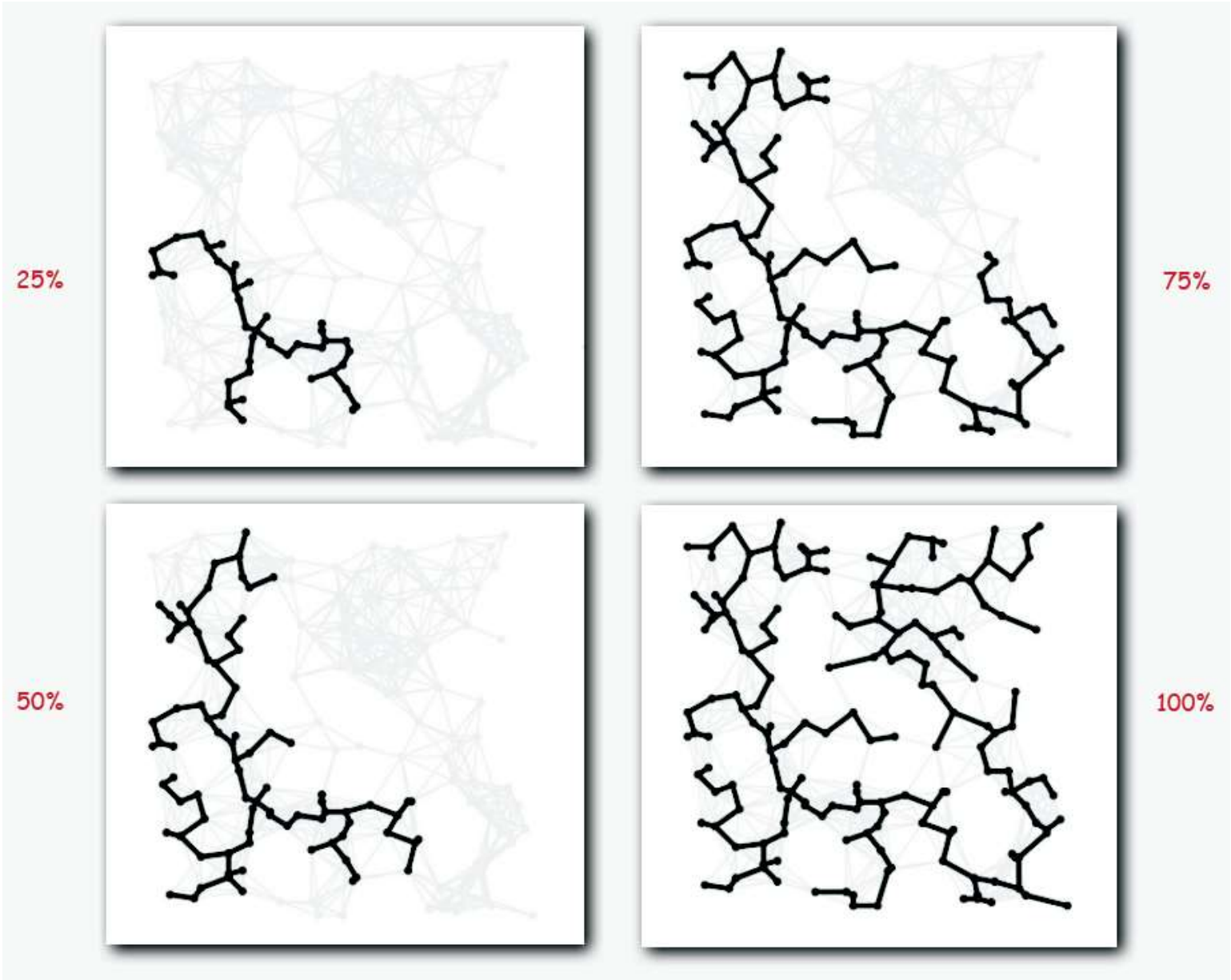


3-5 4-5 0-5



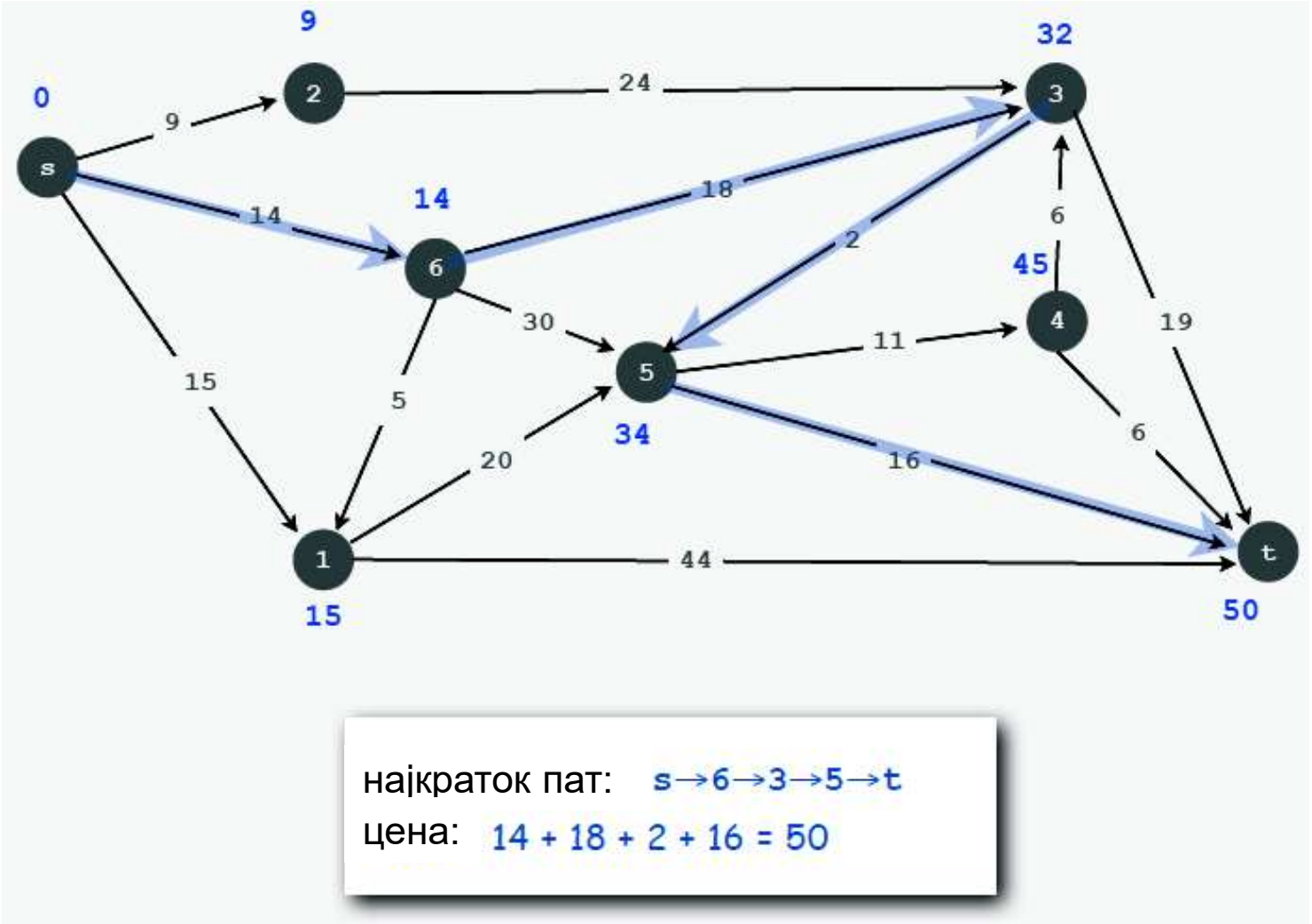

```
0-1  0.32
0-2  0.29
0-5  0.60
0-6  0.51
0-7  0.31
1-7  0.21
3-4  0.34
3-5  0.18
4-5  0.40
4-6  0.51
4-7  0.46
6-7  0.25
```

Примов алгоритам



Најкраток пат во тежински граф

Дадено: Ориентиран тежински граф G.
Цел: Најди го најкраткиот ориентиран пат од s до t



Најкраток пат - примена

- Мапи (Google Maps, Bing maps, Apple, viaMicheline, Opel)

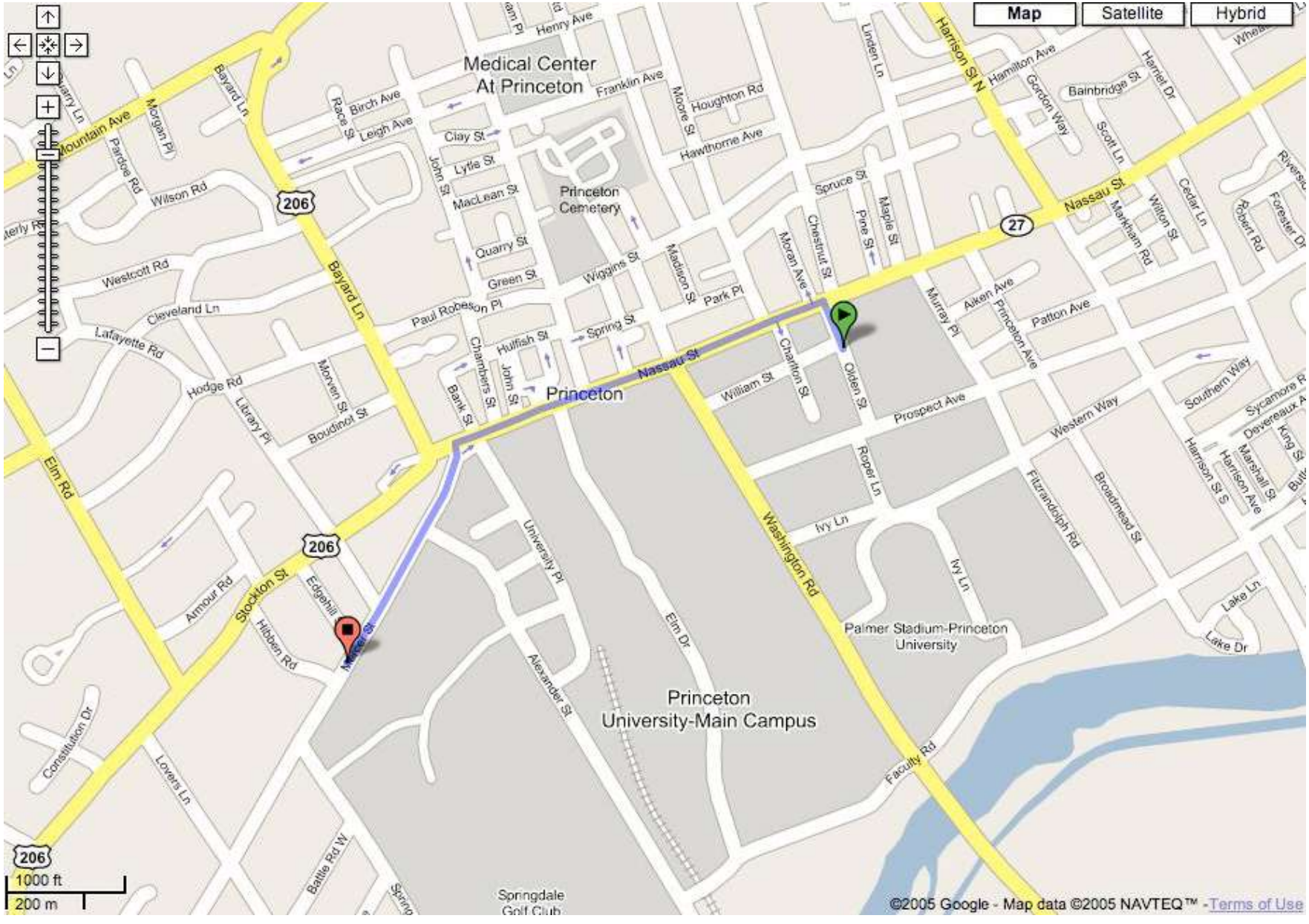
- Навигација на роботи

- Дизајн на урбанистички план

- Подалгоритми во покомплексни алгоритми

- Оптимално наведување во сообраќај
  со позната оптеретеност

Најкраток пат – Google Maps



Најкраток пат - верзии

Кои темиња ?

- Од едно теме до друго
- Од едно теме до сите останати
- Од секое теме до сите останати

Тежина на ребрата:

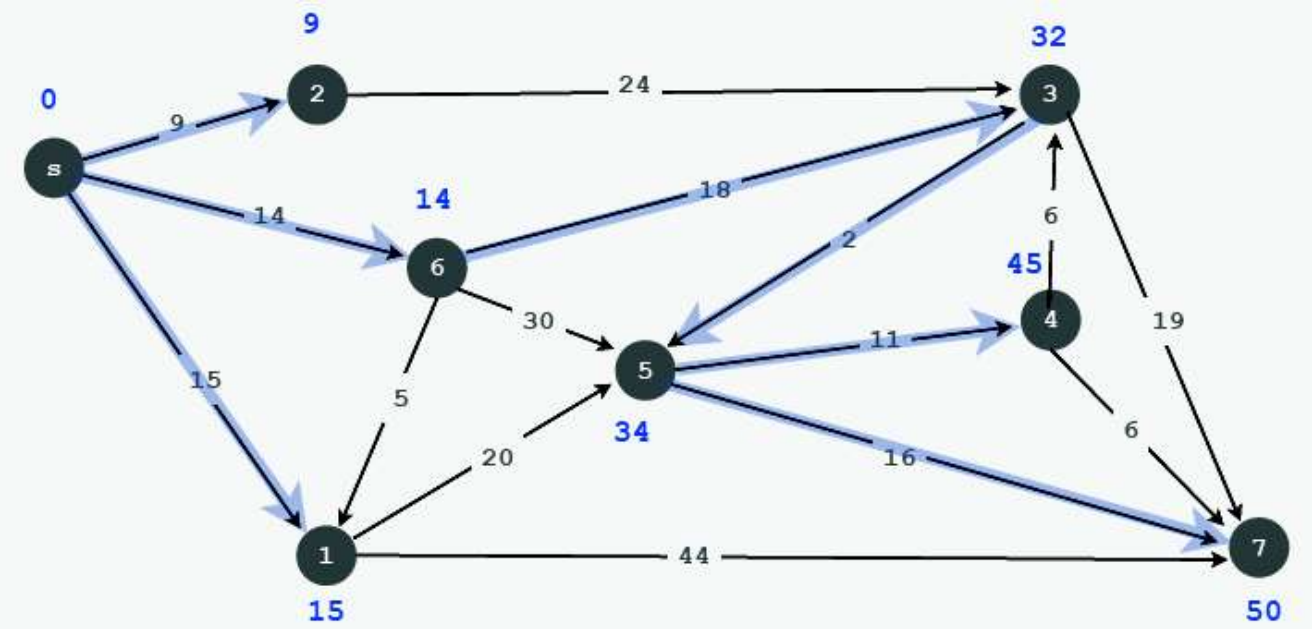
- Ненегативни тежини
- Произволни тежини
- Евклидски тежини

Најкраток пат од едно теме

Дадено: Ориентиран тежински граф G и почетно теме s

Цел: Најди го најкраткиот пат од s до сите останати темиња

Забелешка: користи parent[] (pred[]) за реконструкција на патиштата

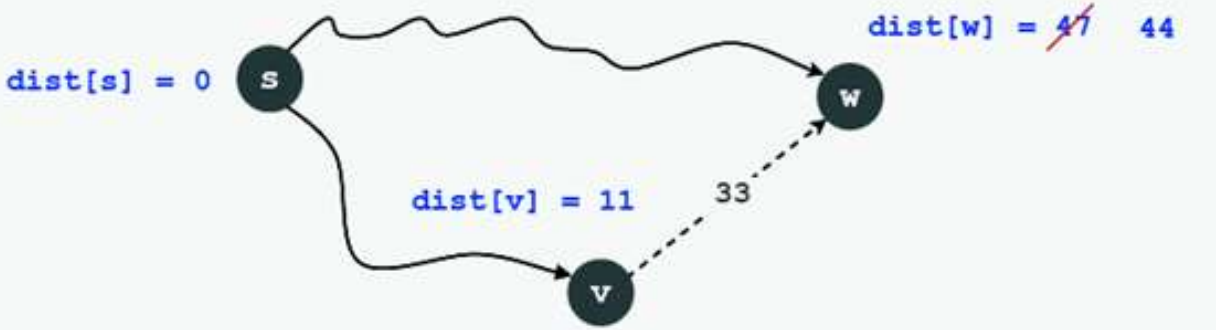


	s	1	2	3	4	5	6	7
dist[v]	0	15	9	32	45	34	14	50
pred[v]	-	0	0	6	5	3	0	5

Релаксација

Релаксација со ребро *e* од *v* до *w*:

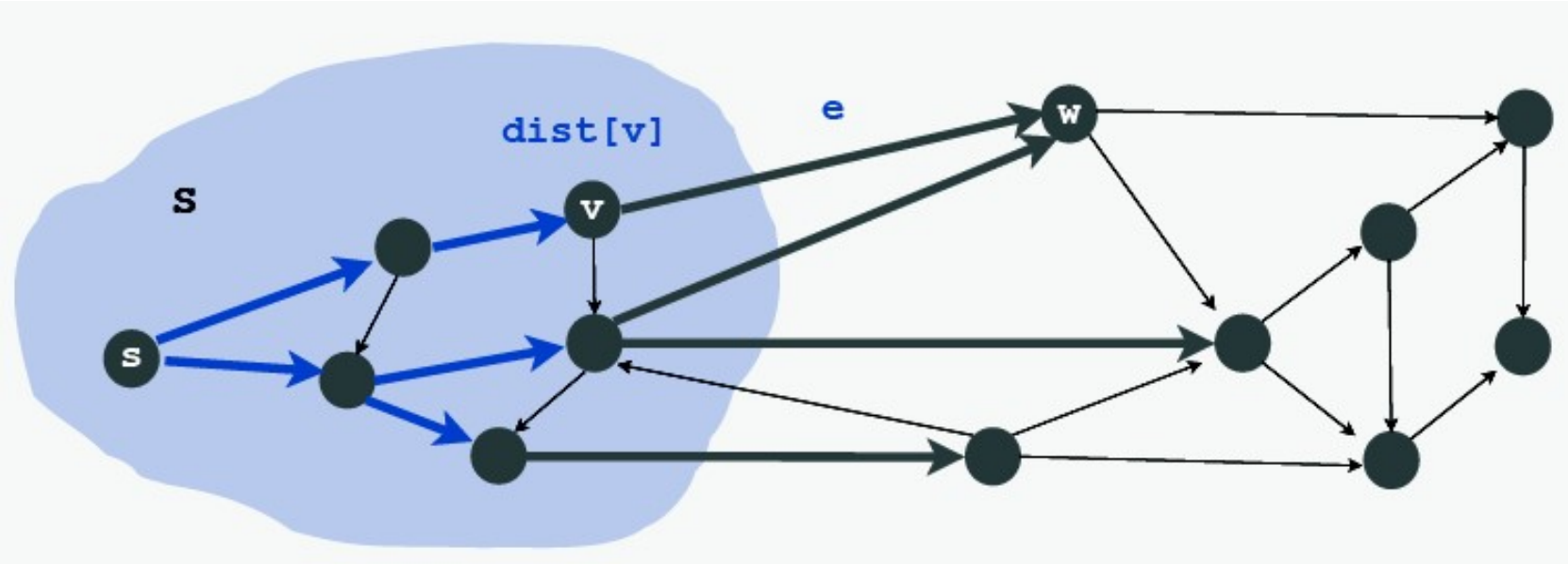
- нека dist[v] ***e*** должина на некој пат од ***s*** до ***v***
- нека dist[w] ***e*** должина на некој пат од ***s*** до ***w***
- Ако ***v***->***w*** креира пократок пат до ***w*** преку ***v***, обнови ги dist[***w***] и pred[***w***].




```
int v = e.from(), w = e.to();
if (dist[w] > dist[v] + e.weight())
{
    dist[w] = dist[v] + e.weight();
    pred[w] = v;
}
```

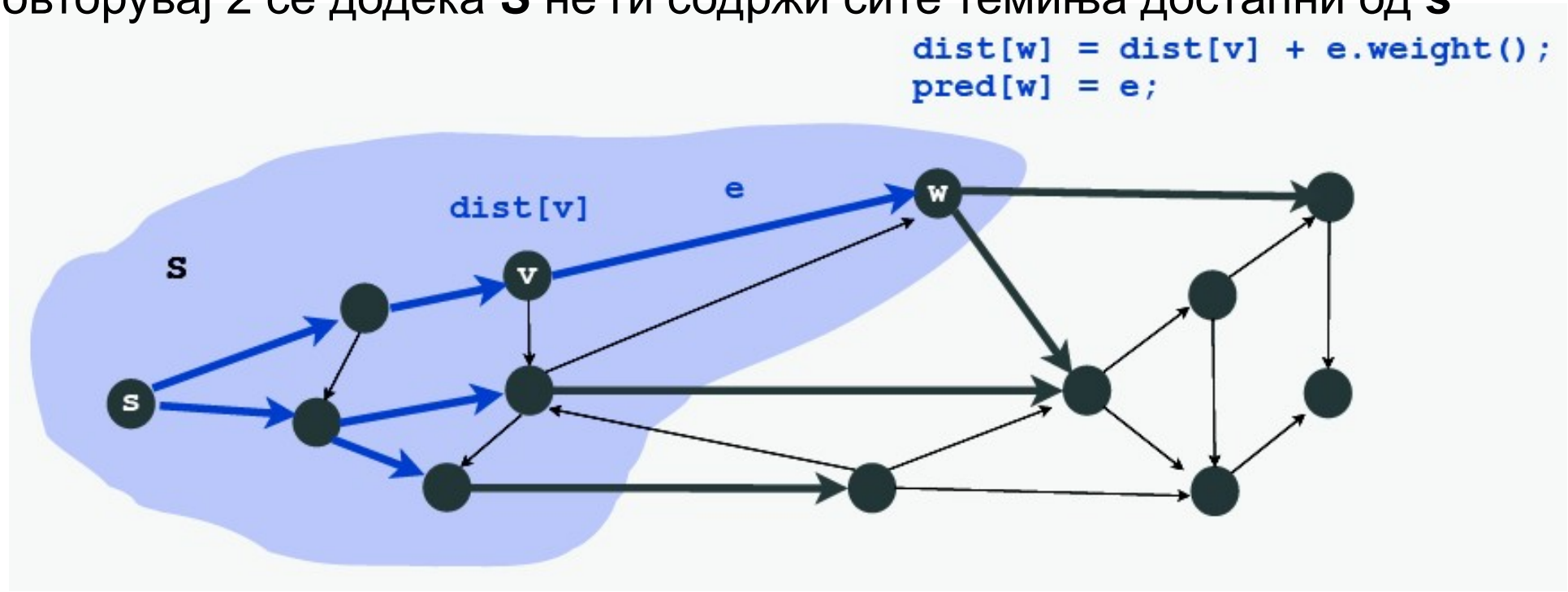
Алгоритам на Дијкстра

1. Иницијализирај го ***S*** со ***s***, dist[***s***] = 0, dist[***v***] = ∞ за останатите ***v,*** и постави го s за моментално теме
2. За моменталното теме:
   - пронајди ребро ***e***, со ***v*** во S и ***w*** не во S, што го минимизира dist[v] + ***e***.weight() и реклаксирај со реброто ***e***
   - Избери теме ***w*** што не е во S најблиску до **s** и стави го моментално
3. Повторувај 2 се додека ***S*** не ги содржи сите темиња достапни од ***s***



Алгоритам на Дијкстра

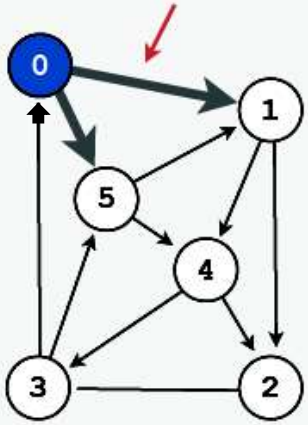
1. Иницијализирај го ***S*** со ***s***, dist[***s***] = 0, dist[***v***] = ∞ за останатите ***v,*** и постави го s за моментално теме
2. За моменталното теме:
   - пронајди ребро ***e***, со ***v*** во S и ***w*** не во S, што го минимизира dist[v] + ***e***.weight() и реклаксирај со реброто ***e***
   - Избери теме ***w*** што не е во S најблиску до **s** и стави го моментално
3. Повторувај 2 се додека ***S*** не ги содржи сите темиња достапни од ***s***



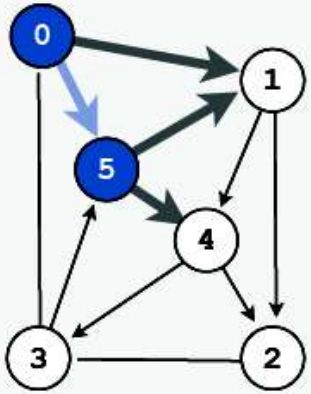
Алгоритам на Дијкстра - пример

Ребро со v во S и w не во S

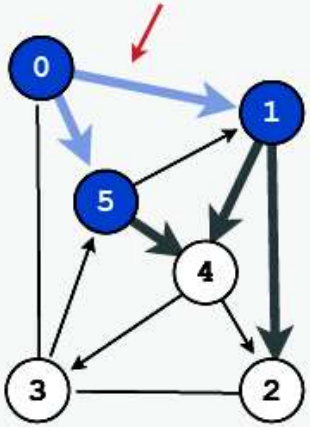
Ребро член на дрвото за најкратки патишта



0→5 (.29)
0→1 (.41)

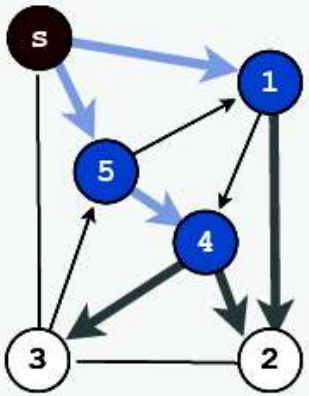


0→1 (.41)
5→4 (.50 = .29 + .21)
5→1 (.58 = .29 + .29)

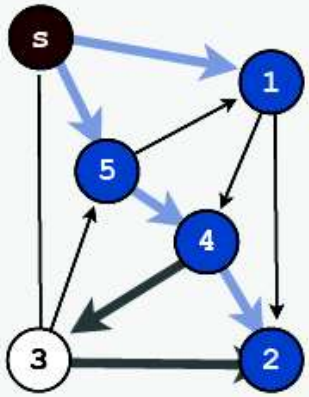


5→4 (.50)
1→4 (.73 = .41 + .32)
1→2 (.92 = .41 + .51)

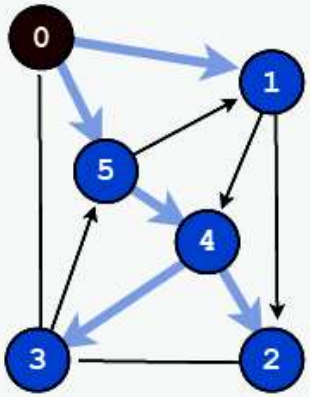
0→1	.41
0→5	.29
1→2	.51
1→4	.32
2→3	.50
3→0	.45
3→5	.38
4→2	.32
4→3	.36
5→1	.29
5→4	.21



4→2 (.82 = .50 + .32)
4→3 (.86 = .50 + .36)
1→2 (.92)

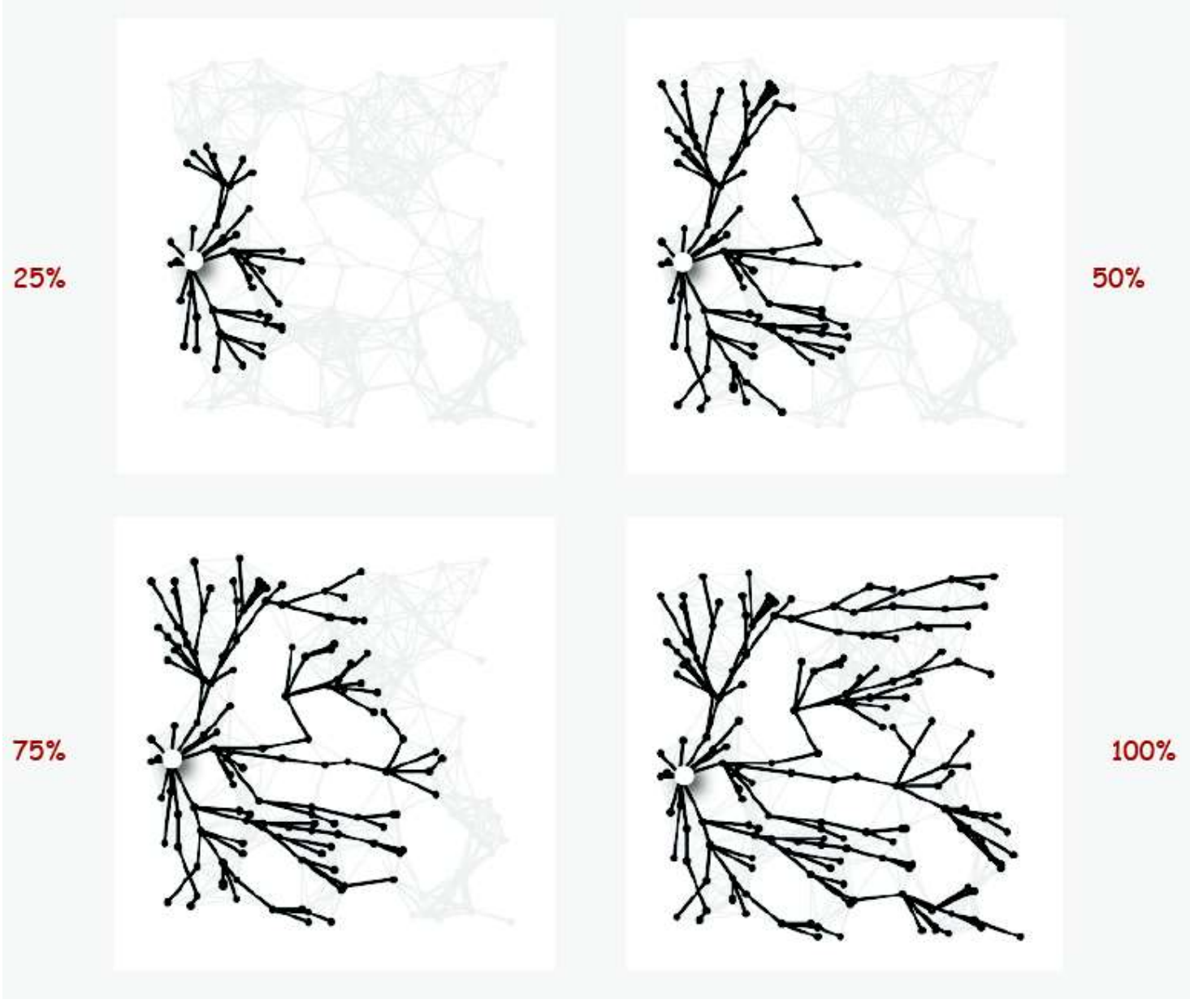


4→3 (0.86)
2→3 (1.32 = .82 + .50)



0	1	2	3	4	5
0					
-					

Алгоритам на Дијкстра



Алгоритам на Дијкстра - имплементација

Проблем: Најди ребро ***v***->***w*** со ***v*** во **S** и ***w*** не во **S**
што го минимизира dist[***v***] + **e**.weight()

Колку е тешко?

- Нерешливо
- O(E)
- O(V)
- O(log V)
- O(1)

Алгоритам на Дијкстра - имплементација

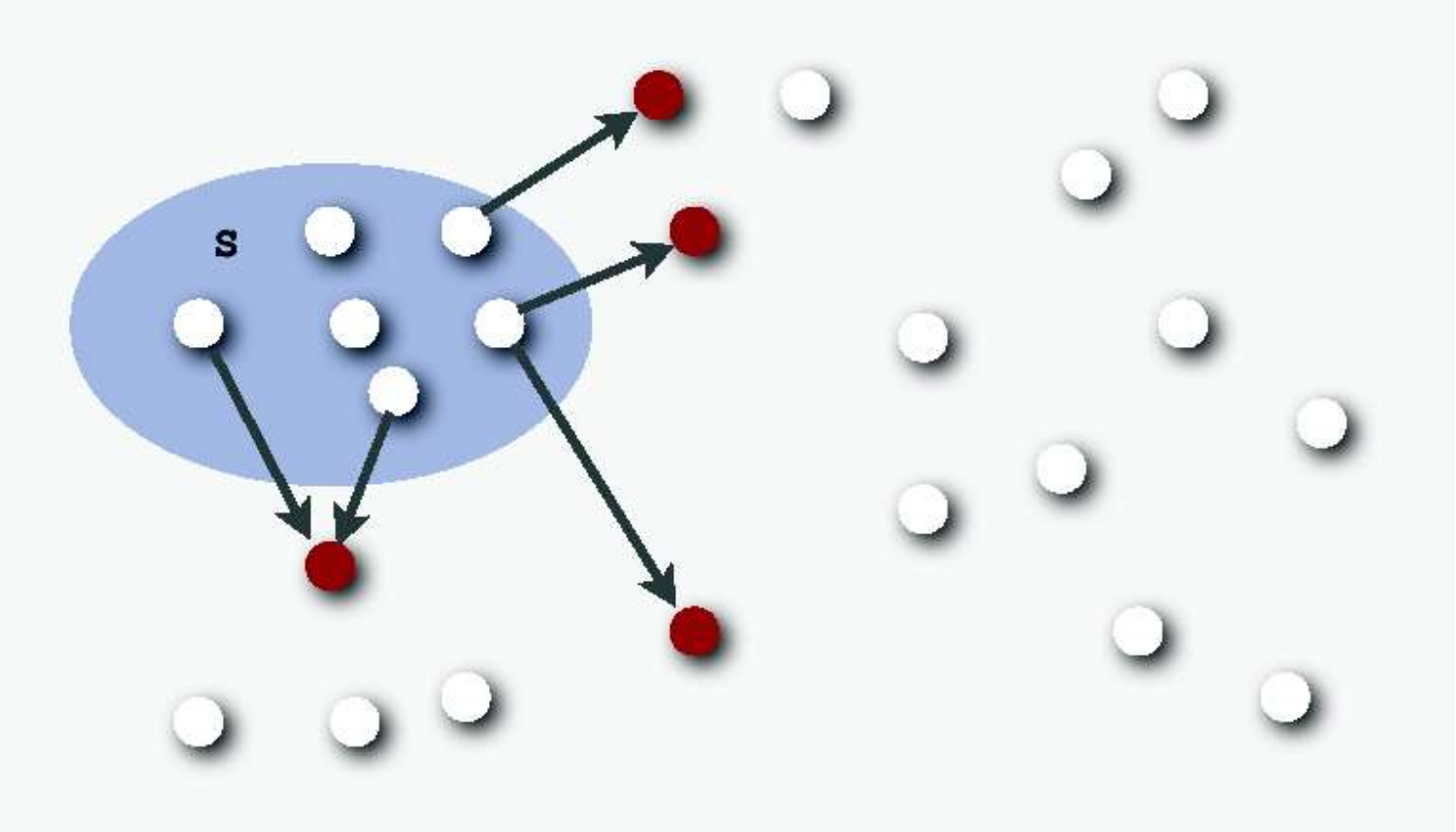
Проблем: Најди ребро со v во S и w не во S што го минимизира dist[v] + e.weight()

Колку е тешко?

- Нерешливо
- O(E) ← Разгледај ги сите ребра
- O(V) ← со низа, сортираме
- O(log V) ← со приоритетна редица (heap)
- O(1)

Алгоритам на Дијкстра - имплементација

Прашање : Што ќе се чува во приоритетната редица ?
Одговор : Темињата кои се на растојание од едно ребро
од темињата кои се веќе елементи на **S**.
Приоритет е нивната моментална оддалеченост од ***s***



Алгоритам на Дијкстра – со приоритетна редица (PQ)

Инваријанти:

- За секое ***v*** во **S**, dist[***v***] е "должината" на најкраткиот пат од ***s*** до ***v***.
- За секое ***w*** што не е во **S**, dist[***w***] го минимизира dist[***v***] + ***e***.weight() за сите ***e*** кои имаат теме во **S**.
- **PQ** ги содржи темињата ***w*** кои не се во **S**, со приоритет dist[***w***]

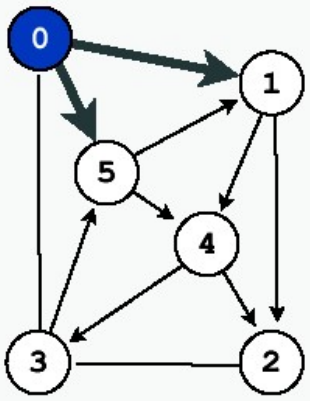
Импликација:

- Следниот елемент ***w*** кој треба да се додаде во **S**, е најмалиот во **PQ**
- За да се одржат инваријантите, обнови го dist[] со релаксација со сите ребра кои се поврзани на ***w***
- Обнови ја **PQ** ако некое теме се "доближи поблиску" до **S**.

Временска комплексност:

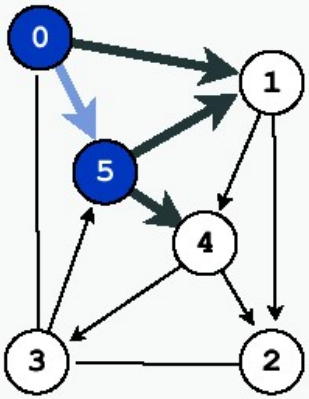
- Зависи од имплементацијата на **PQ**
- Точно **V** delmin() операции
- Точно **E** релаксации со ребро
- Имплементација со heap V*O(1) + E*O(logV) = O(E*log(V))

Алгоритам на Дијкстра - пример

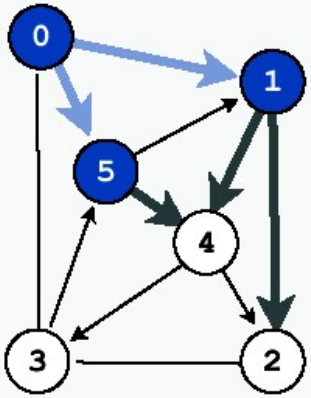


0→5 (.29)
0→1 (.41)

приоритетна листа

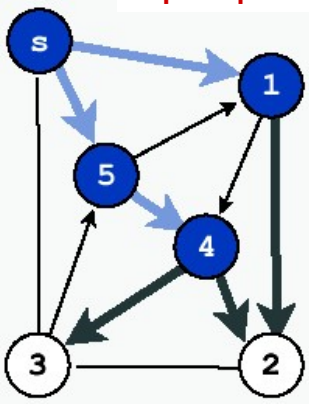


0→1 (.41)
5→4 (.50 = .29 + .21)
5→1 (.58 = .29 + .29)

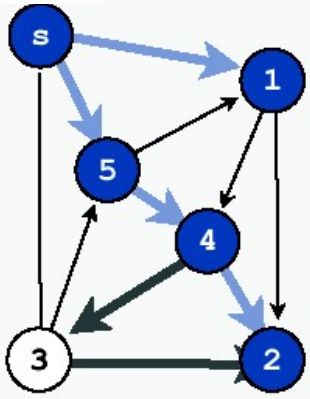


5→4 (.50)
1→4 (.73 = .41 + .32)
1→2 (.92 = .41 + .51)

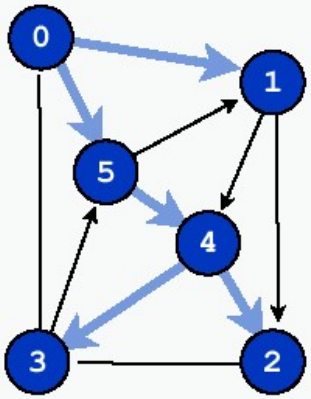
0→1	.41
0→5	.29
1→2	.51
1→4	.32
2→3	.50
3→0	.45
3→5	.38
4→2	.32
4→3	.36
5→1	.29
5→4	.21



1→4 (.73)
4→2 (.82 = .50 + .32)
4→3 (.86 = .50 + .36)
1→2 (.92)



4→3 (0.86)
2→3 (1.32 = .82 + .50)
1→2 (.92)



1→2 (.92)
3→5 (1.24 = .86 + .38)
3→0 (1.29 = .86 + .45)
2→3 (1.32)

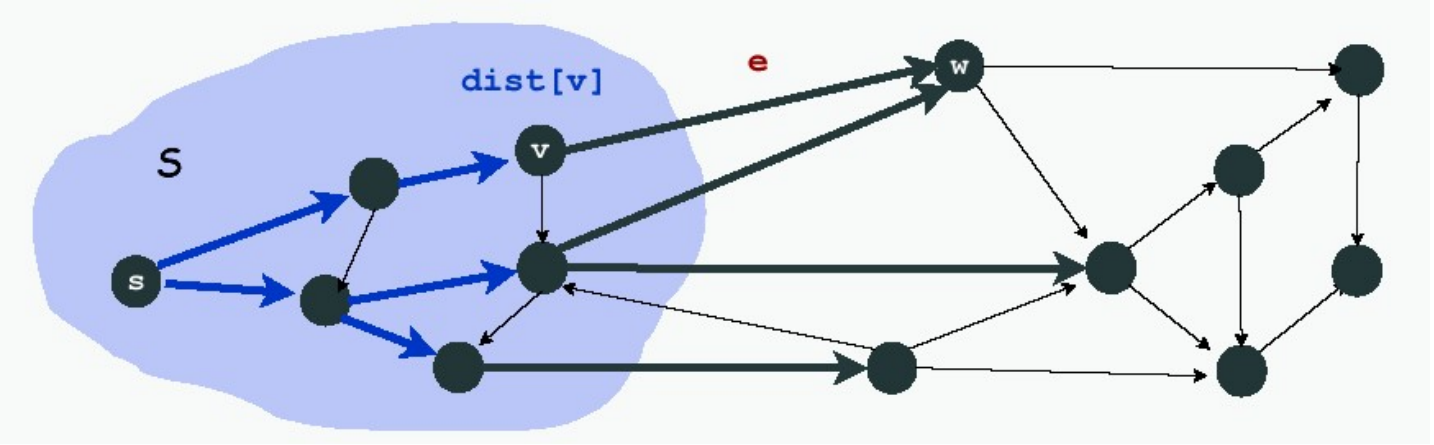
Алгоритам на Дијкстра - имплементација

```
for(v=0;v<V;v++)
  dist[v] <- INF
  marked[v] = False

PQ pq;
dist[s] <- 0
pq.put(dist[s], s)

while(NOT pq.isempty())
begin
  v = pq.delmin()
  if(marked[v]) continue
  marked[v] = True
  for(w:Sosedi(v))
    if(dist[w] > dist[v] + edge[v][w])
    begin
      dist[w] = dist[v] + edge[v][w]
      pred[w] = v
      pq.insert(dist[w], w)
    end
end
```

“Приоритетно” пребарување

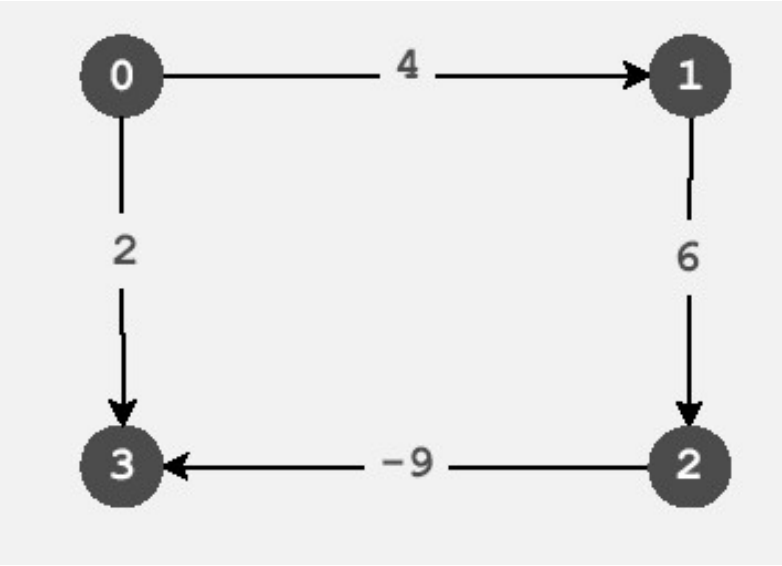
Забелешка: Сите досегашни алгоритми имаат заедничка стратегија:
- Одржувај множество **S** од обработени темиња
- Зголемувај го **S** со користење на ребра кои имаат еден крај во **S**.

DFS: Искористи ребро кое е прикачено на најновиот елемент на **S**
BFS: Искористи ребро кое е прикачено на најстариот елемент на **S**
Примов: Искористи ребро кое има најмала тежина
Дијкстра: Искористи ребро до теме кое е најблиску до **S**



Најкраток пат со негативни ребра

Дијкстра не работи со негативни тежини на ребра

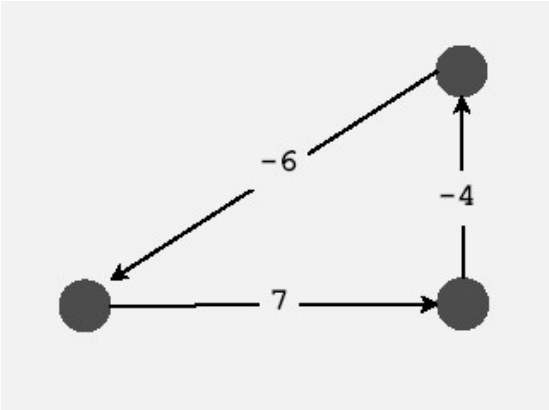


Дијкстра ќе го избере теме 3 веднаш после 0. НО! Најкраткиот пат од 0 до 3 е 0->1->2->3

Додавање на фиксна позитивна вредност на тежините исто така не работи (проблем се подолги патишта)

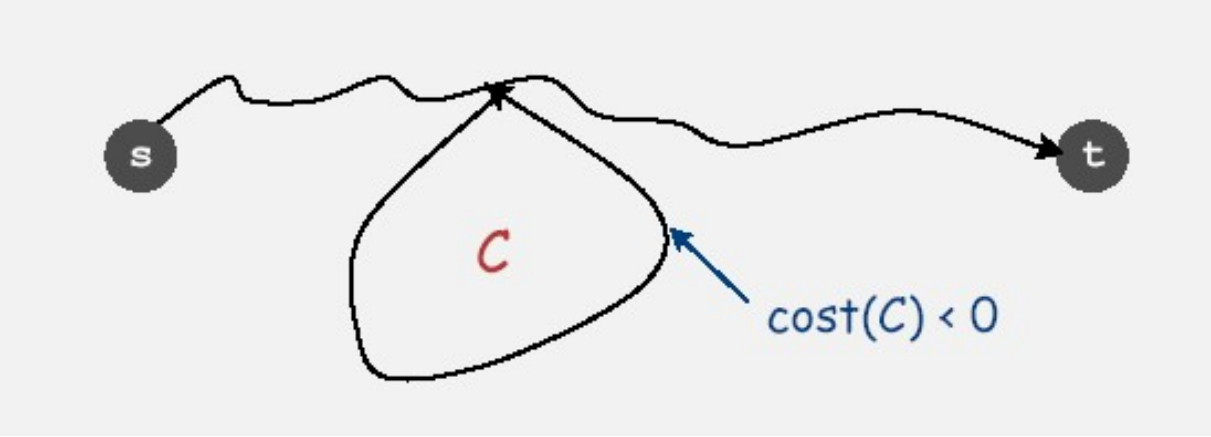
Најкраток пат со негативни ребра

Деф: Негативен циклус е ориентиран циклус чија сума на тежини на ребра е негативна



Забелешка - проблем:

Ако негативен циклус се појави на патот од s до t тогаш патот може да се направи произволно негативен со вртење околу циклусот



Негативни тежини на ребра

Дадено: Одреден број на валути и курсна листа.
Цел: Кој е најдобриот начин да се замени една унца злато во долари ?

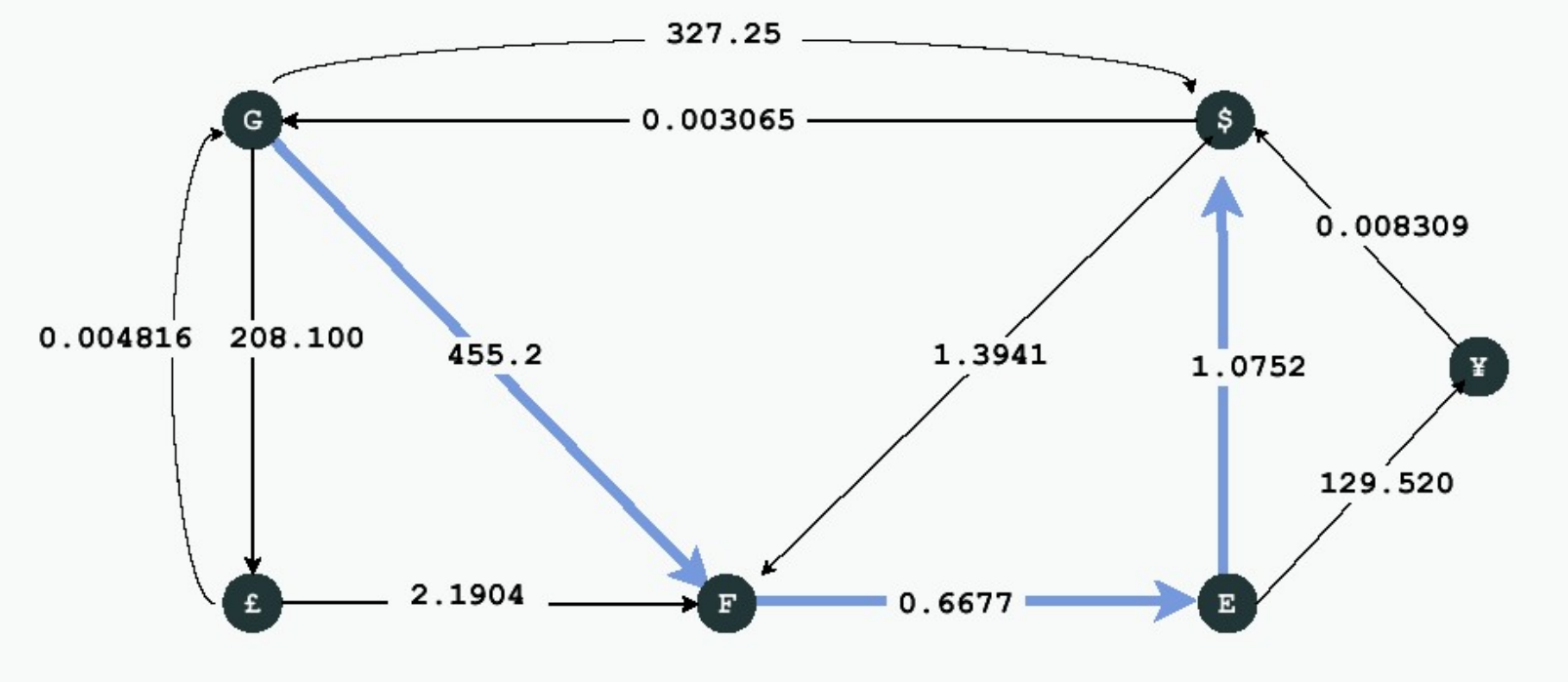
- 1 oz. gold ⇒ $327.25.
- 1 oz. gold ⇒ £208.10 ⇒ $327.00. [208.10 × 1.5714]
- 1 oz. gold ⇒ 455.2 Francs ⇒ 304.39 Euros ⇒ $327.28. [455.2 × .6677 × 1.0752]

currency	£	Euro	¥	Franc	$	Gold
UK pound	1.0000	0.6853	0.005290	0.4569	0.6368	208.100
Euro	1.45999	1.0000	0.007721	0.6677	0.9303	304.028
Japanese Yen	189.50	129.520	1.0000	85.4694	120.400	39346.7
Swiss Franc	2.1904	1.4978	0.01574	1.0000	1.3941	455.200
US dollar	1.5714	1.0752	0.008309	0.7182	1.0000	327.250
Gold (oz.)	0.004816	0.003295	0.0000255	0.002201	0.003065	1.0000

Конверзија на валути

Формулација со граф:

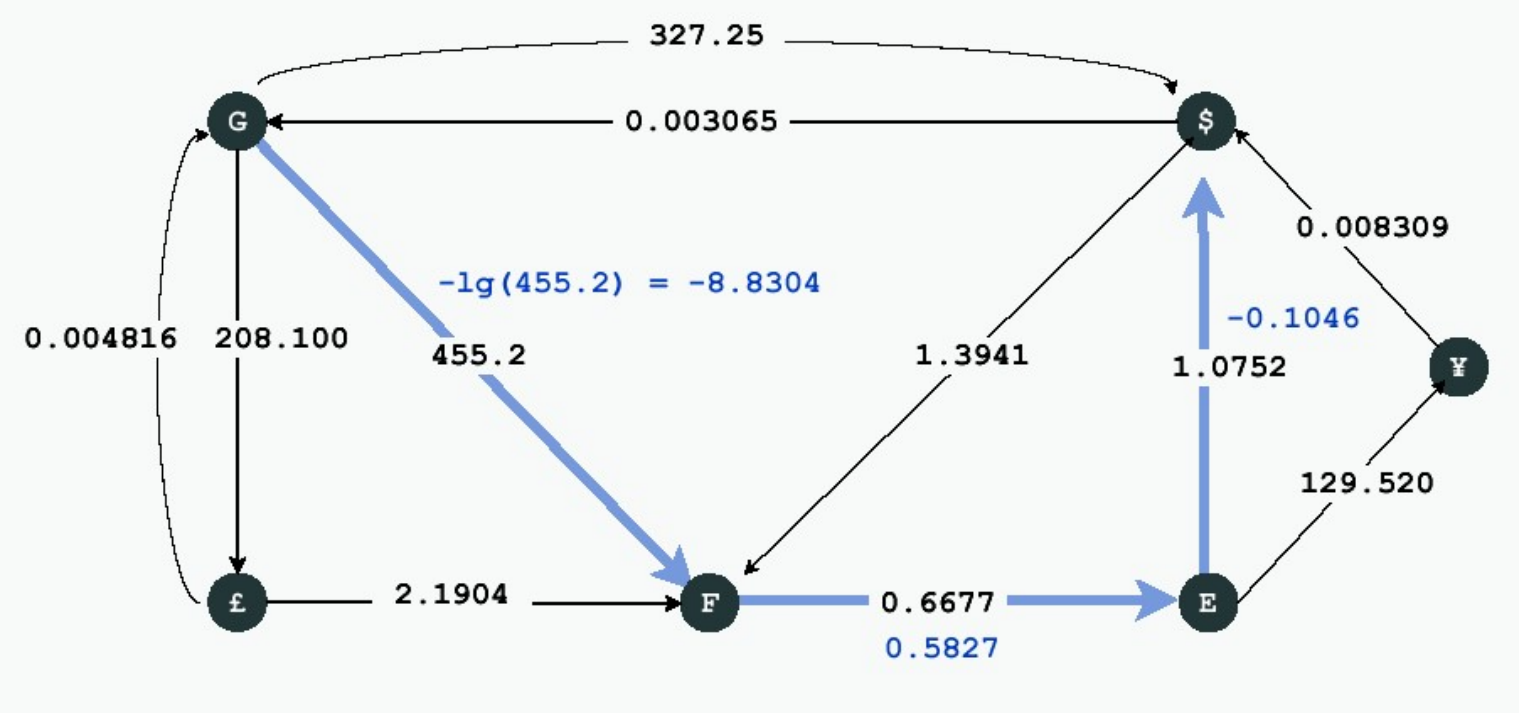
- Темиња: валута
- Ребра: конверзија , со тежина од курсна листа
- Најди пат кој го максимизира производот на тежините на патот



Конверзија на валути

Трансформација во најкраток пат, преку логаритми:

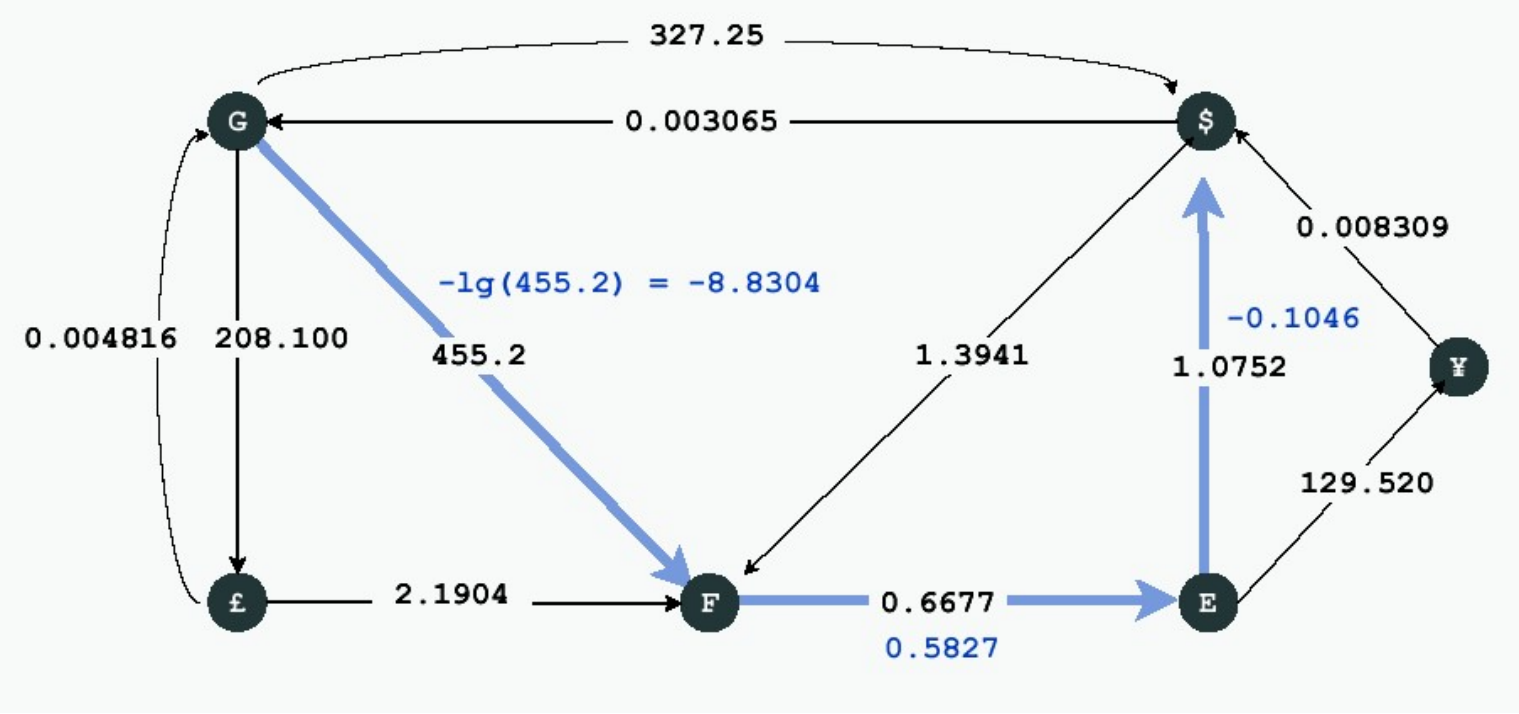
- Нека тежината на реброто v->w (t) се замени со $-\log_2(t)$
- Множењето се заменува со собирање
- Најкраткиот пат кореспондира на најдобрата секвенца од замена на валути



Конверзија на валути

Трансформација во најкраток пат, преку логаритми:

- Нека тежината на реброто v->w (t) се замени со $-\log_2(t)$
- Множењето се заменува со собирање
- Најкраткиот пат кореспондира на најдобрата секвенца од замена на валути



Проблем: Најкраток пат во граф со негативни тежини

Најкраток пат со негативни ребра – динамичко решение

```
for(v=0;v<V;v++)
  dist[v] = INF
dist[s]= 0.
ima = False

for (int phase = 1; phase <= V; i++)
  for (int v = 0; v < V; v++)
    for (w: Sosedi(v))
      if (dist[w] > dist[v] + e[v][w])
      begin
        dist[w] = dist[v] + e[v][w])
        pred[w] = v;
        if (phase == V) ima = True
      end
```

Најкраток пат со негативни ребра – динамичко решение

```
for(v=0;v<V;v++)
  dist[v] = INF
dist[s]= 0.
ima = False

for (int phase = 1; phase <= V; i++)
  for (int v = 0; v < V; v++)
    for (w: Sosedi(v))
      if (dist[w] > dist[v] + e[v][w])
      begin
        dist[w] = dist[v] + e[v][w])
        pred[w] = v;
        if (phase == V) ima = True
      end
```

Белман-Форд-Мур алгоритам:
Открива дали има негативен циклус. Ако dist[] не се менува после V-1 фаза тогаш не постои негативен циклус

Комплексност: O(EV) – забрзување со чување на редица од подобрени темиња