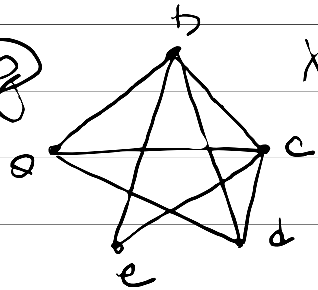
def 1. • Хамилтонов път – минаващ точно веднъж през всеки връх на графа (първи ≠ последен).

• Хамилтонов цикъл – хамилтонов път, в който първи = последен.

зад.

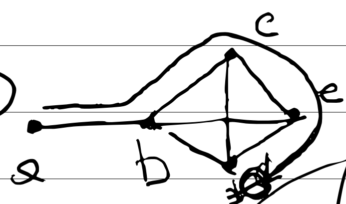


Хамилтонов ли е графа?

a, c, e, b, d, a

a, b, e, c, d, a

зад.



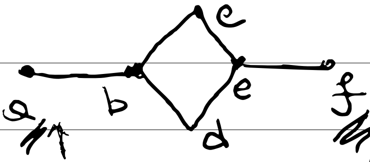
a) Хамилтонов ли е?

b) Има ли хамил. път?

> Определено не.

> a, b, c, e, d

зад



a) Хамилтонов ли е?

b) Има ли хамил. път?

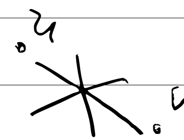
> Не е Хамилтонов.

> Няма и Хамилтонов път.

Th/ на Dirac

Ако $G = \langle V, E \rangle$ е граф, т.ч. $n = |V| \geq 3$ и всеки връх $v \in V$, то $d(v) \geq \frac{n}{2}$, то G е Хамилтонов

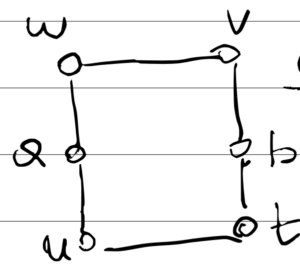
Th/ на Ore



Ако $G = \langle V, E \rangle$ е граф, т.ч. $n = |V| \geq 2$ и за всяка двойка несъседни върхове u и $v \in V$ е изпълнено, че $d(u) + d(v) \geq n$, то G е Хамилтонов.

зад) Условието $d(u)+d(v) \geq n$ е достатъчно, но не е необходимо, за да бъде графа Хамилтонов.

прим1



$\underline{d(u)+d(v)} = 2+2 = 4 \ngeq 6 = n$

b Също $(\forall v' \in V)[d(v') = 2 \ngeq 3]$

$3 = \frac{6}{2}$

Th на Dirac също не става

Но има Хамилтонов цикъл!

(303) На прием били поканени 20 гости. Всеки от тях познава поне 10 от другите. Док., че можем да ги наредим на масата, така че всеки да си познава двамата съседа.

Неориен. граф: • V - гостите

• E - познанства

$(\forall v \in V) d(v) \geq 10$, $|V| \geq 3$, $|V| \geq 2$

↳ По Оре G е Хамилтонов

$(\forall v \in V)(\forall u \in V)[\{u,v\} \notin E \Rightarrow d(u)+d(v) \geq 20]$

↳ По Dirac G е Хамилтонов.

Хамилтонов цикъл → става да ги наредим на масо.

u - v - w ...

def | Път – не може да се повтарят върхове и ребра.

Верига – не може да се повтаря ребро, но може да се повтарят върхове.

def | Ойлерови вериги

Минават през всяко ребро точно веднъж. Има два вида: отворени и затворени.

Th | Нека **G е свързан мултиграф**.
Ако броят на върховете от нечетна степен е:

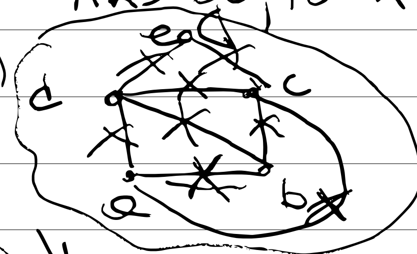
a) 0, то има затворена Ойлерова верига и не същ. отворена.

b) 2, то същ. отворена Ойлерова верига с крайща-двата върха от нечетна степен и не същ. затворена

c) ≠ 0 и 2, то не същ. нито отворена, нито затворена Ойлерова верига.

392 Има ли Ойлерова верига?
Ако да, то каква е тя?

a)

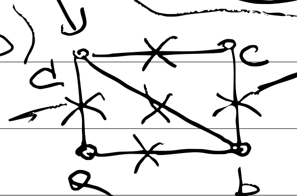


$d(a)=3$, $d(b)=3$

Имаме отворена

a d e c b a c d b

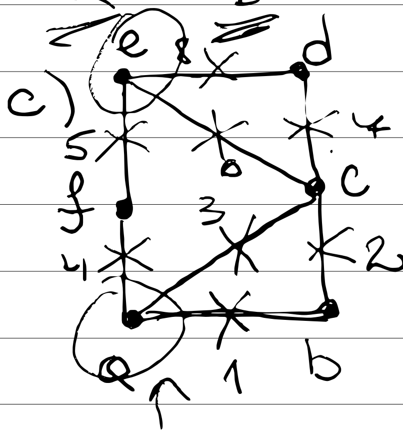
b)



Има точно 2 от нечетна

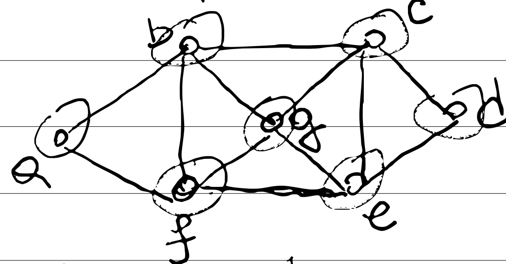
d c b d a b

c)



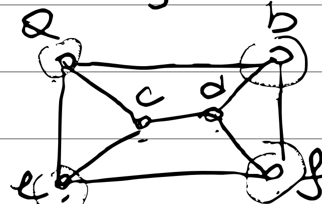
399 Какви са следните графи?

a) Ойлеров ли е? Имаме затворено. Аз Ойлеров.



Няма никаква верига

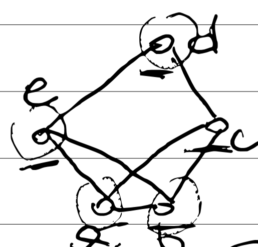
b) Ойлеров или Хамилтонов? $n=6$



$(\forall v \in V)\left[d(v) \geq \frac{n}{2}\right]$

Хамилтонов е. Дирак.

c) Ойлеров или Хамилтонов?

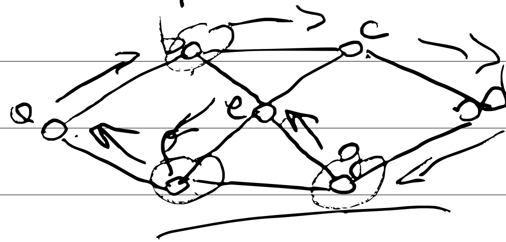


Няма вериги. $n=5$

$(\forall v \in V)(\forall u \in V)$[u и v не са съседни $\Rightarrow d(v)+d(u) \geq 5$]

Th Оре $\Rightarrow$ Хамилтонов е.

d) Ойлеров и Хамилтонов?

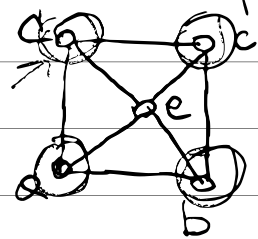


Няма верига.

$n=7$.

Хамилтонов: a b c d g e f a.

e) Хамилтонов, но не е Ойлеров

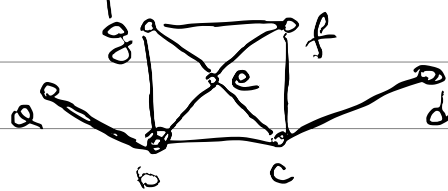


$n=5$

Th Ore -> Хамилтонов

a b c d e a

g) Не Ойлеров и не Хамилтонов



Покриващи дървета (ST)

(Минимални/максимални покриващи дървета OST)

def Покриващо дърво

$T = \langle V, \mathcal{E}' \rangle$ е покриващо дърво на $G = \langle V, \mathcal{E} \rangle$, ако:

- $\mathcal{E}' \subseteq \mathcal{E}$
- G е свързан (иначе T е гора);
- T е дърво.

Ако графът има тегла по ребрата, можем да говорим за <u>Optimal spanning tree</u> (OST) и то <u>максимални</u> или <u>минимални</u>.
Минимално покриващо дърво съдържа всички върхове на даден свързан тегловен <u>неориент.</u> граф и сбора от теглата на ребрата е <u>минимален</u>.

Основни алгоритми строящи покриващи дървета на <u>НГ/ОГ</u>.

- bfs – в ширина
- dfs – в дълбочина

Имаме теорема за НГ:

$\boxed{\text{Th} \mid \text{Всяко кореново дърво е дърво както и:}}$

Th | Графът G има покриващо дърво <-> G е свързан

bfs и dfs обхождат граф и строят покриващо дърво. С тях може да проверявате дали е свързан граф. С dfs е удобно да проверите дали граф съдържа цикъл. Bfs строи дърво на най-късите пътища (когато G няма тегло по ребрата) от даден начален връх до всички останали.

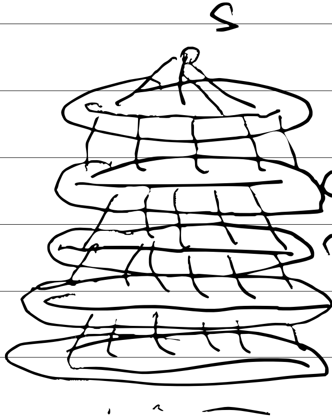
```
BFS(G, s)
  for all v ∈ V:
    dist(v) = ∞
    parent(v) = null
  dist(s) = 0 // тривиален път от s до s
  queue = initQueue(s) // в опашката
```

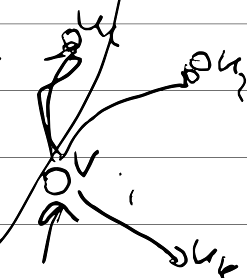
са сиви върхове, а извън нея са или бели, или черни. Белите имат dist = ∞ а черните dist < ∞

```
  while queue is not empty:
    v = dequeue(queue) // вадим главата на опашката
    for all edges {v,u} ∈ E:
      if dist(u) = ∞: // бял връх
        enqueue(queue, u)
        dist(u) = dist(v)+1
        parent(u) = v
```

масив - пази разстояние от s до останалите

масив пази покриващото дърво

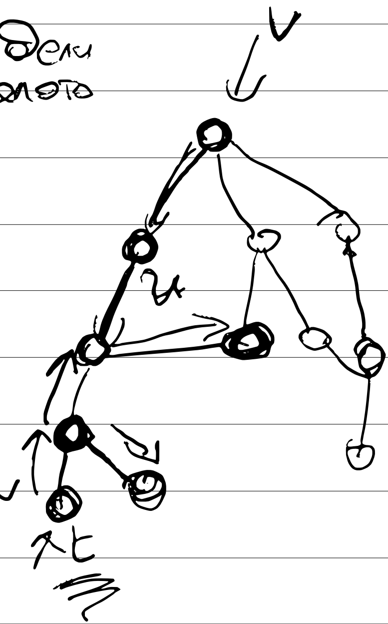




O(|V|+|E|)
сложност по време

```
DFS(G)
  forall v ∈ V:
      visited(v) = false   // само дели в началото
      parent(v) = null
  forall v ∈ V
      if not visited(v):
          explore(G, v)

explore(G, v)
  visited(v) = true   // сив връх е v
  for all edges {v, u} ∈ E:
      if not visited(u):
          parent(u) = v
          explore(u)
  // вече тук v е черен.
```



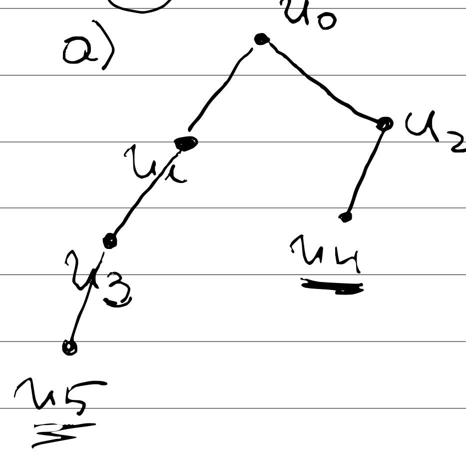
Има O(|V| + |E|) сложност по време. Всъщност такава е и сложността по памет.

(302) Да се номери покриващото дърво на графа:

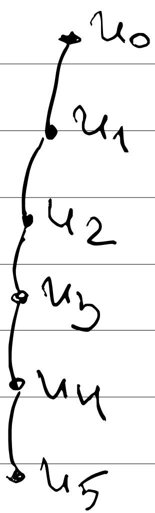
a) с bfs;

b) с dfs.

a)



b)



Optimal spanning tree

Kruskal(G, w):

$\mathcal{E}_T = \{\}$ $\qquad \mathcal{E}_T$

Sort edges $\mathcal{E}$ by inc. weight using w.

for all edges $\{u, v\} \in \mathcal{E}$:

if adding $\{u, v\}$ to $\mathcal{E}_T$ does not form a cycle:

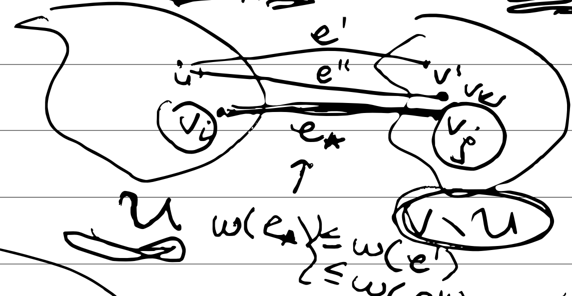
add edge to $\mathcal{E}_T$

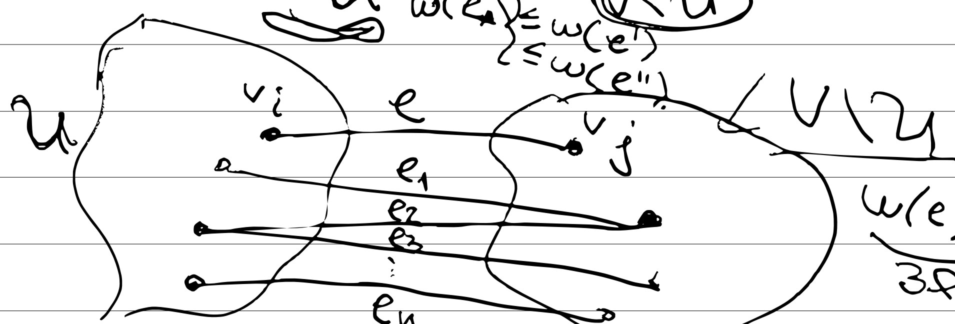
return $\mathcal{E}_T$

Най-проста имплементация.
Основава се на тн за съгласуваното мн-во:

Greedy algorithm

$\boxed{\text{Тм}}$ Нека $G = \langle V, E \rangle$ е НГ, свързан с ценова ф-ия ребрата $w: E \to \mathbb{R}$ и $\emptyset \neq U \subsetneq V$. Нека $e = \{v_i, v_j\} \in E$ е т.е. $v_i \in U$, $v_j \notin U$ и има мин. тегло измежду всички такива ребра. Тогава съществува MST $T_0 = \langle V, E_0 \rangle$, т.е. $e \in E_0$.



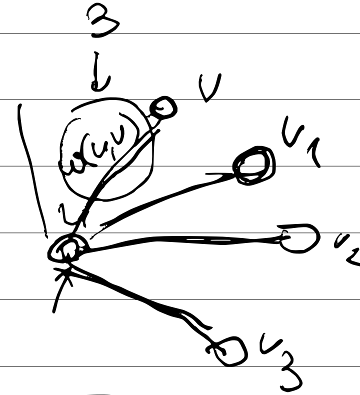


$w(e) \leq w(e_i)$ за $i = 1, \dots, n$

```
Prim-Jarnik(G, w, s)
  forall v ∈ V
    cost(v) = ∞
    parent(v) = null
  cost(s) = 0
  PQ = makePQ(V0, cost)   // строи приор. опашка спрямо cost
  while PQ is not empty:
    u = popMin(PQ)   // връща върха с най-малък ключ
    foreach (u,v) ∈ E:
      if cost(v) > w(u,v)
        cost(v) = w(u,v)
        parent(v) = u
        decreaseKey(PQ, v)
```

най-евтина цена за се стигне до върха v от кой да е негов съсед.

relax(u, v, w, cost, parent, PQ)

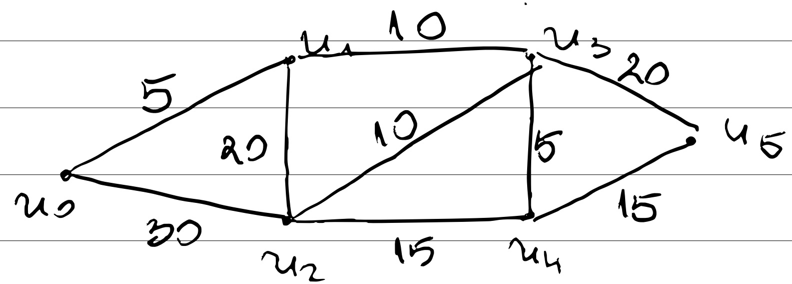


намерили сме връх, за който можем да подобрим цената му

~~5~~ 3

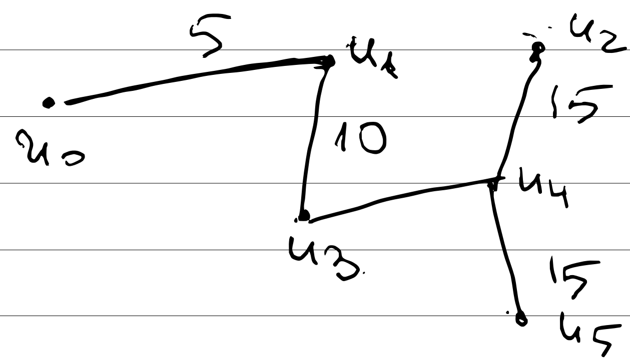
Използваме сигнал към опашката, че cost-ът на v е било намалено и предизвиква пренареждане на PQ спрямо обновения масив cost.

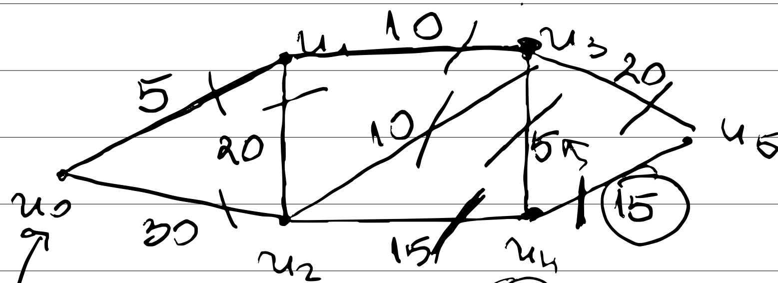
(302) Да се намери MST на графа.

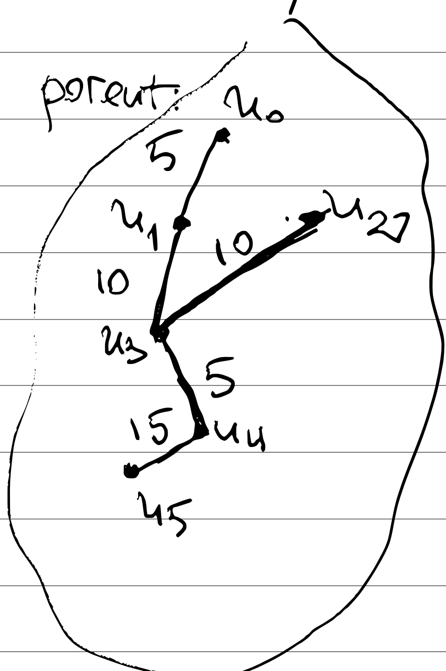


Kruskal:

~~$\{u_0, u_1\}: 5$~~; ~~$\{u_1, u_3\}: 5$~~; ~~$\{u_1, u_3\}: 10$~~; ~~$\{u_2, u_4\}: 15$~~;
~~$\{u_4, u_5\}: 15$~~; ~~$\{u_1, u_2\}: 20$~~; ~~$\{u_3, u_5\}: 20$~~; ~~$\{u_0, u_2\}: 30$~~







PQ :
cost :

~~u_0 0~~
~~u_1 ∞ 5~~
~~u_2 ∞ 30 20 10~~
~~u_3 ∞ 10~~
~~u_4 ∞ 5~~
~~u_5 ∞ 20 15~~

Prim-Jarnick