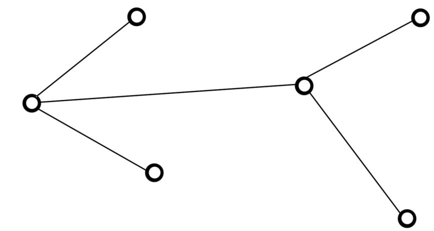
ЛЕКЦІЯ 12

Дерева, ліс
Остовні дерева, остовний ліс
Пошук остовних дерев з мінімальною вагою
Обхід графа

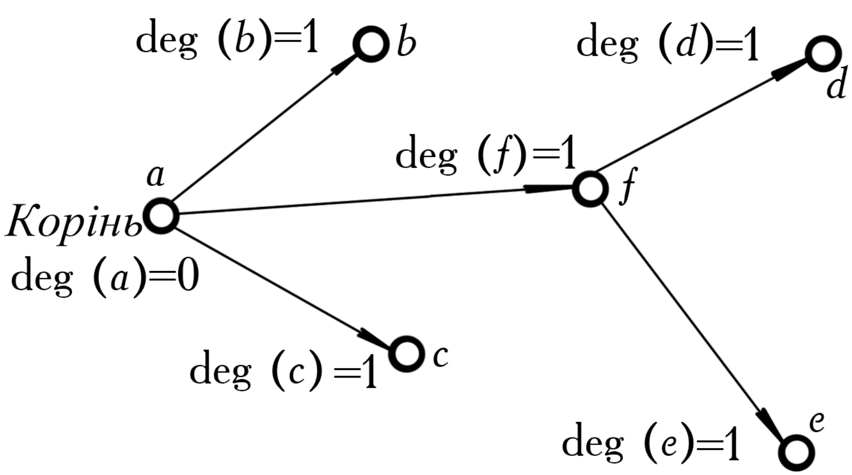
Визначення дерева. Властивості дерев

Неорієнтованим деревом називають зв'язний неорієнтований граф без циклів.



Кореневим деревом називають таке дерево, у якому існує виділена вершина, що має назву *кореня.*

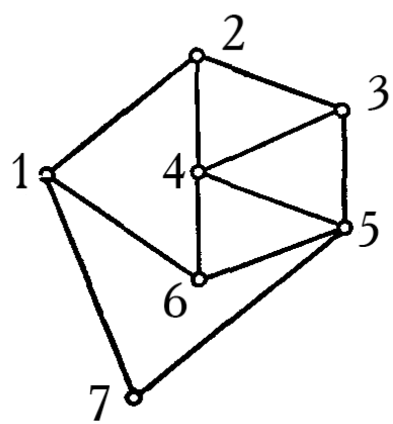
Корінь у неорієнтованому графі – це одна з вершин, обрана за бажанням спостерігача.



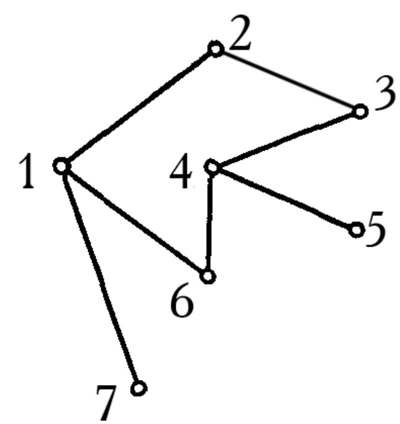
Орієнтованим деревом називають зв'язний орієнтований граф без циклів, у якому напівстепінь входу кожної вершини, за винятком кореневої, дорівнює одиниці, а напівстепінь входу кореневої вершини дорівнює 0.

Остовним підграфом називають такий підграф, у якому множина його вершин збігається з множиною вершин самого графа.

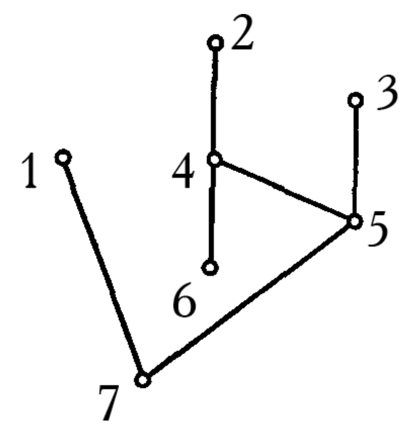
Остовним деревом графа G називають **остовний підграф** графа G, який є деревом.



Початковий граф



Остовний підграф



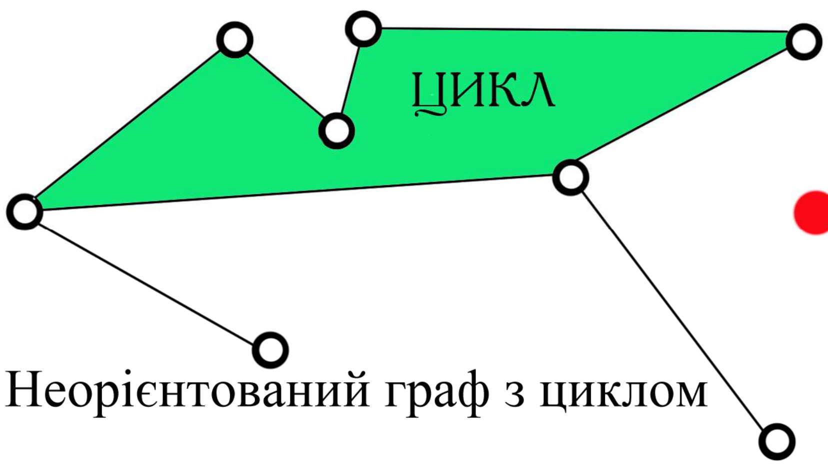
Остовне дерево

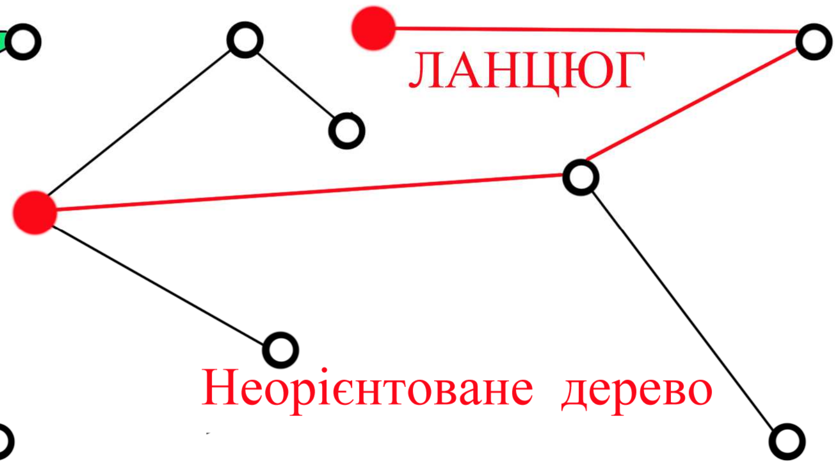
Теорема 1. Граф є деревом тоді і тільки тоді, коли будь-які дві його вершини зв'язані єдиним ланцюгом.

Доведення. Нехай граф є деревом. Якщо припустити існування більш ніж одного ланцюга, що зв'язує будь-які дві його вершини, то в такому графі існує цикл, тобто граф не може бути деревом.

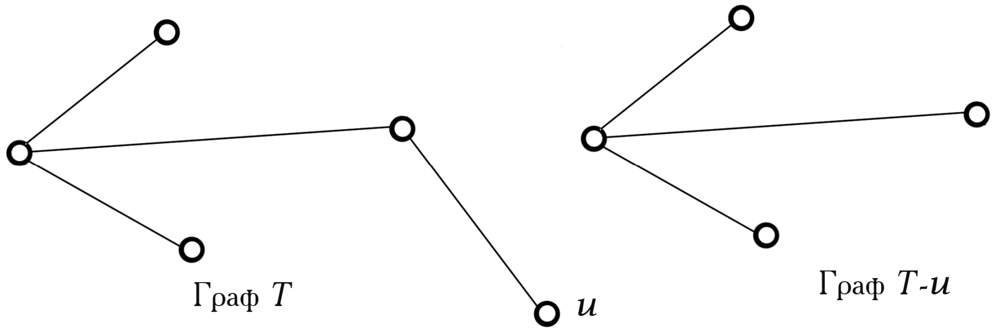
Навпаки, оскільки будь-які дві вершини графа з'єднані ланцюгом, то граф зв'язний, а в силу того, що цей ланцюг єдиний, він не має циклів. Отже, граф є деревом.

Теорема доведена.





Наслідок 1. Якщо T – дерево і u – його кінцева вершина, то граф $T-u$ *(T мінус u)* – дерево.
Дійсно, граф $T-u$ – підграф дерева T, для якого виконуються всі умови теореми.



Наслідок 2. Усяке непусте дерево має принаймні дві висячі вершини і одне висяче ребро.

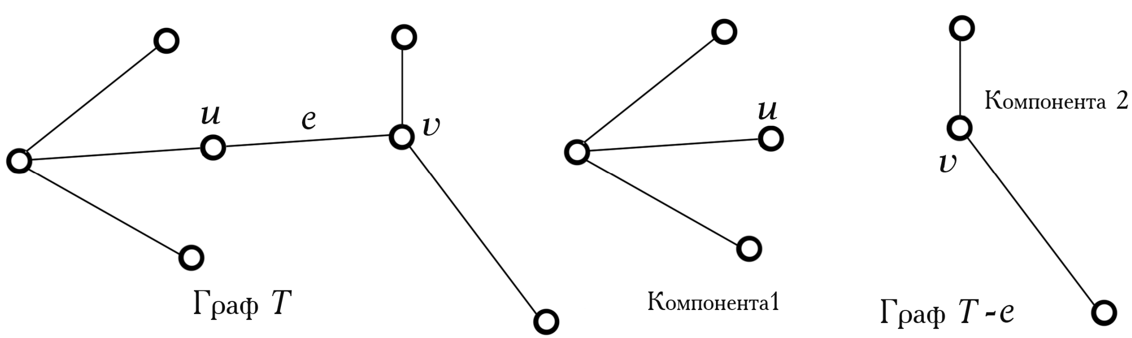
1. **Висяча вершина в неорієнтованому графі** – це вершина степеня 1.
2. **Висяча вершина в орграфі** – вершина з напівстепенем входу, що дорівнює 1, і напівстепенем виходу, що дорівнює 0.
3. **Висяче ребро** – це ребро, інцидентне вершині зі степенем 1.

Визначення. Ребро зв'язного графа називають *істотним*, якщо його видалення веде до порушення зв'язності цього графа.

Наслідок. У неорієнтованому графі істотним ребром є міст.

Теорема. У дереві кожне ребро істотне.

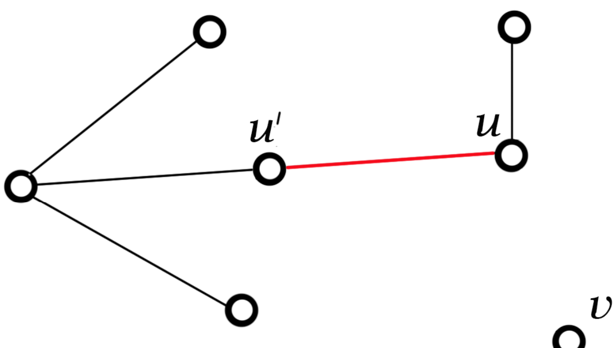
Доведення. Доведення випливає з того, що видалення ребра $e = \big(u, v\big)$ в дереві *T* через наявність єдиного ланцюга, який з'єднує вершини u і v, веде до появи двох компонент зв'язності: одна компонента містить вершину u, а інша – вершину v.



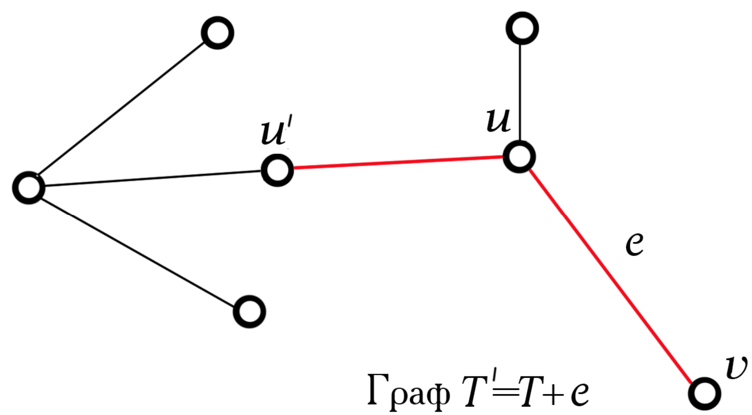
Отже, граф $T - e$ не є зв'язним.

Теорема 2. *Якщо* $T=\left(V,E\right)$ *– дерево і вершина* $v\notin V$, *то граф* $T'=\left(V\cup\left\{v\right\},E\cup\left\{\left(u,v\right)\right\}\right)$, *де* u *– довільна вершина з* V*, теж є деревом.*

Доведення. Оскільки T – дерево, то існує єдиний ланцюг, що з'єднує будь-яку вершину u' з вершиною u. Оскільки вершина $v\notin V$, то додавання одного кінцевого ребра $\left(u,v\right)$ приводить до того, що з кожної вершини u' маємо лише єдиний ланцюг, який з'єднує вершини u' і v. Виходячи з теореми 1 граф T' є деревом.



Граф T



Граф $T'=T+e$

Теорема 3. *Нехай граф* T *має* n *вершин. Тоді еквівалентними є такі твердження:*

1. *T є деревом.*

2. *T не має циклів і має* $n-1$ *ребро.*

3. *T – зв'язний граф і має* $n-1$ *ребро.*

4. *T – зв'язний граф і кожне його ребро є мостом.*

5. *Будь-які дві вершини графа* T *з'єднані тільки одним простим ланцюгом.*

6. T *не має циклів, але додавання будь-якого нового ребра в* T *сприяє виникненню тільки одного циклу.*

Визначення. Ліс – незв'язний n-граф без циклів.
1. Зв'язні компоненти лісу є деревами.
2. Будь-яка частина лісу або дерева також не має циклів.
3. Будь-яка частина лісу є лісом або деревом.

Теорема. Нехай ліс G містить n вершин і k компонентів. Тоді ліс G має $n-k$ ребер.

Доведення. З умови 2 теореми 3 випливає, що кожний компонент G_i має $\left(n_i - 1\right)$ ребро. Але тоді число ребер у G дорівнює

$$\left(n_1 - 1\right) + \left(n_2 - 1\right) + ... + \left(n_k - 1\right) = n_1 + n_2 + ... + n_k - k = n - k,$$

що і потрібно довести.

Теорема Келі. *Число різних дерев, які можна побудувати на n вершинах, дорівнює n^{n-2}.*

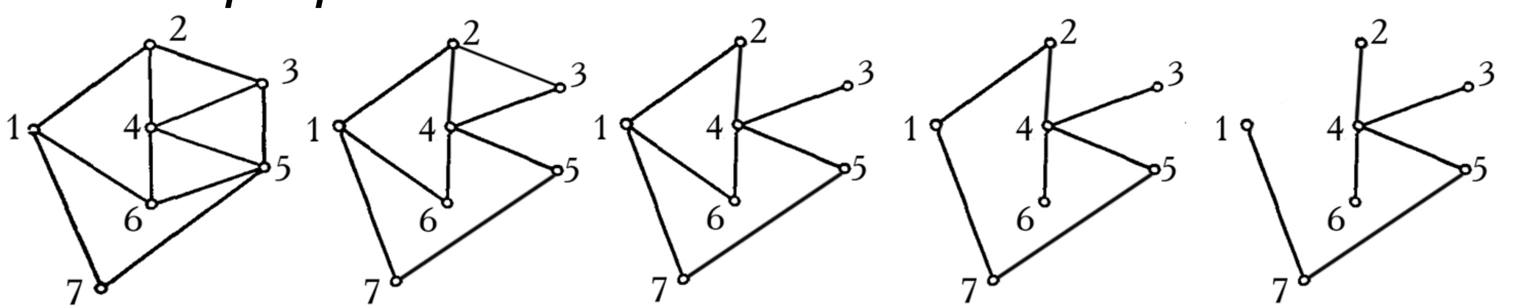
Процедура побудови остовного дерева

1. Видалення зі зв'язного графа G одного ребра, що належить деякому циклу, не порушує зв’язності графа G.

2. Застосуємо процедуру видалення ребра до одного з циклів у графі G.

3. Будемо повторювати видалення доти, поки в G не залишиться жодного циклу.

4. У результаті одержимо дерево, що містить усі вершини графа G. Це дерево називають *остовним деревом графа G.*



T $T_1 = T - (3,5)$ $T_2 = T_1 - (2,3)$ $T_3 = T_2 - (1,6)$ $T_4 = T_3 - (1,2)$

Процедура побудови остовного лісу

Нехай G являє собою граф з n вершинами, m ребрами і k компонентами.

1. Застосуємо процедуру видалення ребра до одного з циклів у кожній компоненті зв'язності графа G.

2. Будемо повторювати видалення доти, поки в кожній компоненті G не залишиться жодного циклу.

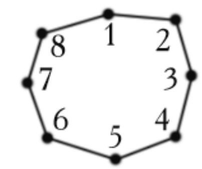
3. У результаті одержимо граф, який називають *остовним лісом.*

4. Число ребер, які при цьому видаляються, називають *цикломатичним числом* або *циклічним рангом графа* G і позначають $C(G)$. Таким чином, цикломатичне число є мірою зв'язності графа.

Властивості циклічного рангу

1. **Циклічний ранг дерева** дорівнює нулю.
2. **Циклічний ранг циклічного графа** дорівнює одиниці.

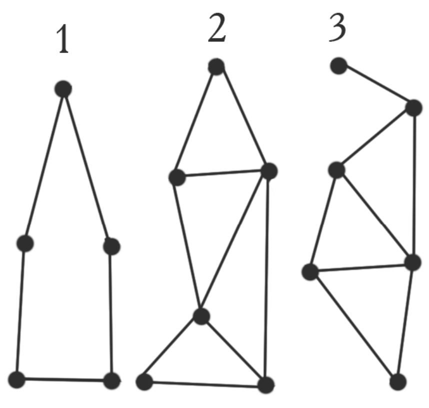
Циклічний граф – зв'язний регулярний граф степеня 2, єдина компонента зв’язності циклічного графа є простим циклом.



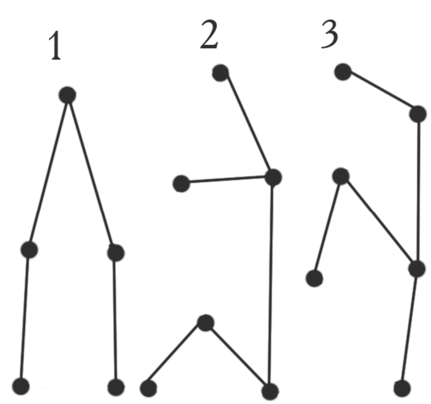
Остовне дерево графа – це дерево, що містить усі вершини графа.

Остовним лісом називають незв'язний граф, що складається з компонентів, які є остовними деревами.

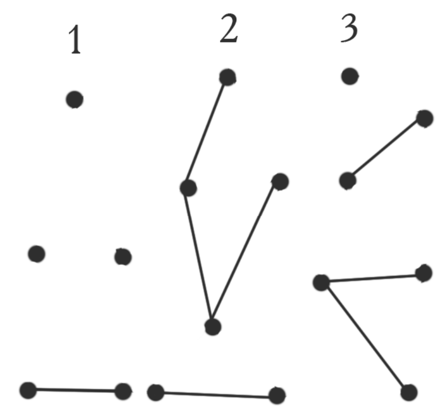
Теорема. Нехай *T* – остовний ліс графа G. Граф G', отриманий із графа G шляхом видалення всіх ребер графа T, називають *доповненням* остовного лісу T графа G.



Незв'язний граф G



Остовний ліс T

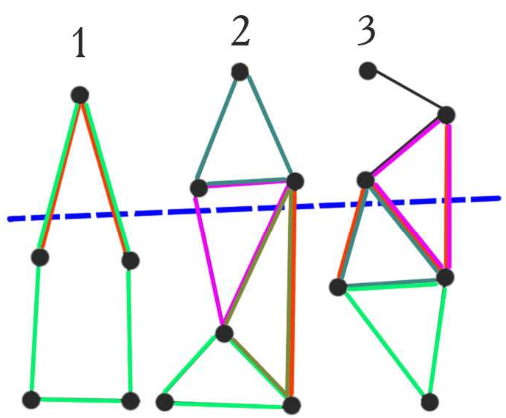


G'-доповнення остовного лісу T

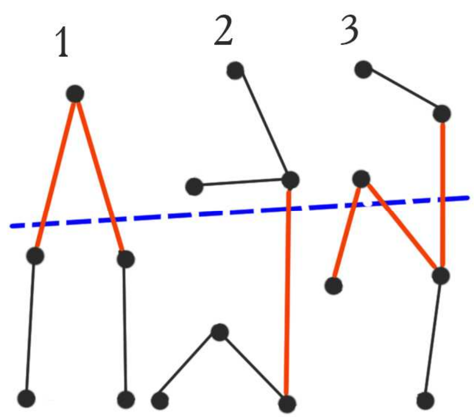
Теорема 4. Якщо T – остовний ліс графа G, то

а) усякий перетин в G має загальне ребро з T ;

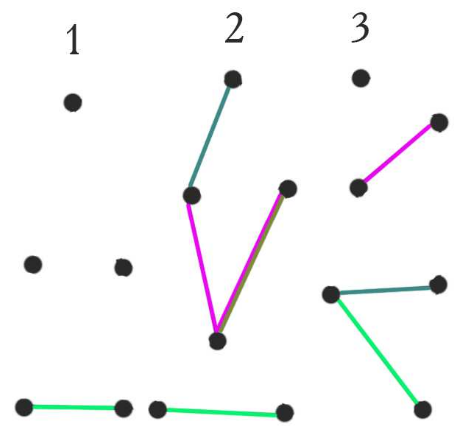
б) усякий цикл у G має загальне ребро з доповненням T .



Незв'язний граф G



Остовний ліс T



Доповнення остовного лісу G'

Фундаментальна система циклів графа

Нехай T – остовний ліс графа *G*.
Якщо додати до T будь-яке ребро графа G, що не входить до нього, то за п.6 теореми 3 одержимо єдиний цикл.

Визначення. Множину всіх циклів, які одержують шляхом додавання окремо кожного ребра з G, що не входить в T, називають фундаментальною системою циклів, асоційованою з T.

Цикли даної фундаментальної системи будуть різними, але їх кількість дорівнює циклічному рангу графа G.

На рисунку показані графи, які показують фундаментальну систему циклів.

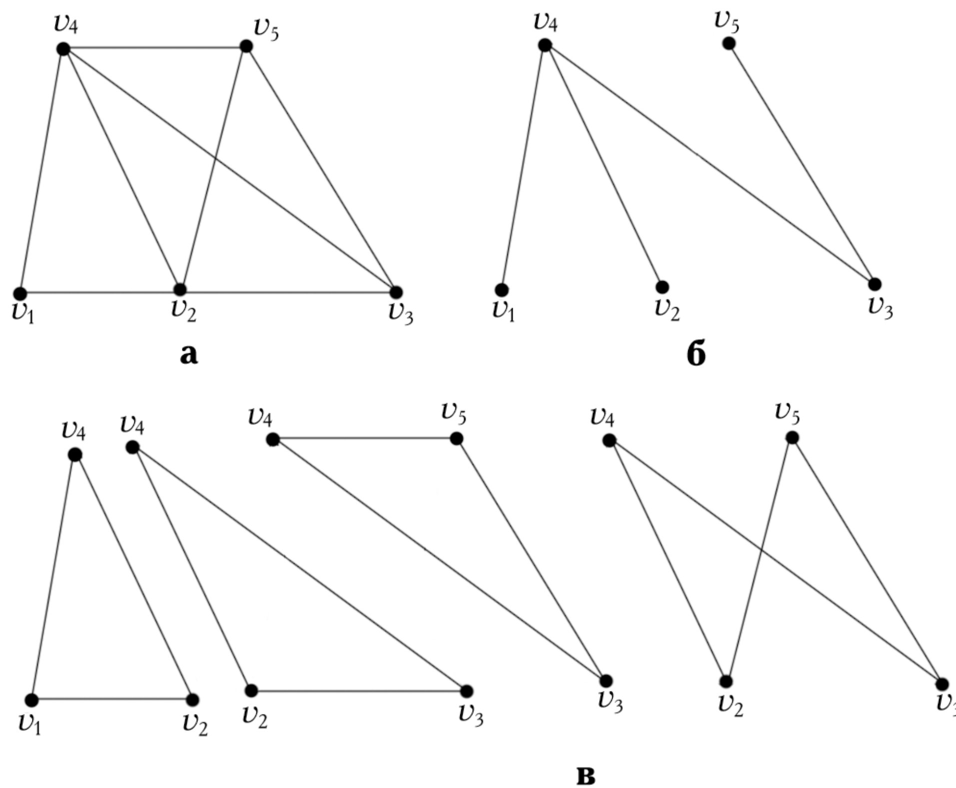


Рисунок: а – граф G; **б** – остовне дерево графа G; **в** – фундаментальна система циклів, асоційована з остовним деревом графа G.

Теорема 5. Нехай $G=\left(V,E\right)$ – довільний скінченний граф. Число ребер графа G, які необхідно вилучити для одержання остовного лісу *T*, не залежить від порядку їх видалення і дорівнює $C\left(G\right)=\left|E\right|-\left|V\right|+k$, де k – число компонент зв’язності графа G.

Наслідок. Граф *G* є остовним лісом тоді і тільки тоді, коли $C\left(G\right)=0$.

Наслідок. Граф *G* має єдиний цикл тоді і тільки тоді, коли $C\left(G\right)=1$.

Наслідок. Граф G, у якого число ребер перевищує число вершин, має цикл.

Наслідок. Будь-яке дерево порядку $n\geq 2$ має принаймні дві кінцеві вершини. 

Остов найменшої ваги

Постановка задачі

Нехай кожному ребру графа $G = (V, E)$ поставлене у відповідність деяке число

$$d_i = d(e_i), \quad e_i \in E, \quad i = 1, 2, \dots, |E|.$$

Необхідно в графі G знайти остовне дерево, сума чисел d_i у якому найменша.

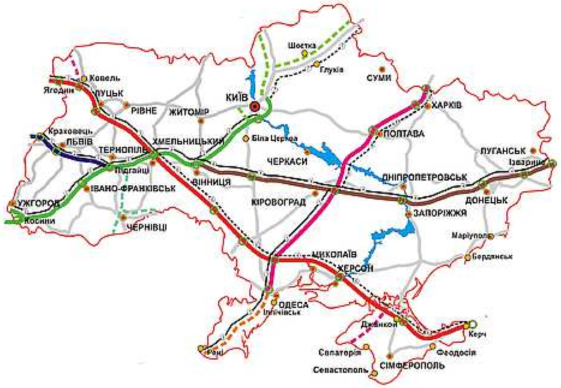
$$S = \min \sum_{e_i \in E} \left(d(e_i) \right)$$

Число d_i в цьому випадку називають *вагою ребра* e_i, а сам граф G – таким, що має *вагу*.

Отже, завдання полягає в тому, щоб знайти остовне дерево з найменшою вагою.

Практичні застосування задачі пошуку остовного дерева з мінімальною вагою

Ця задача виникає при проектуванні доріг, ліній електропередач, трубопроводів та ін.,




коли необхідно з'єднати задані точки деякою системою зв'язку так, щоб загальна довжина ліній зв'язку була мінімальною.

Розглянемо кілька методів розв'язування цієї задачі.

Розв'язок, що випливає з теореми Келі

1. Розглянемо всі можливі остовні дерева повного графа G з n вершинами.

Згідно з теоремою Келі число *різних дерев, які можна побудувати на* n *вершинах, дорівнює* n^{n-2}.

2. При виборі кожного дерева визначимо суму його ваг.

3. Застосувавши сортування за величиною суми ваг, на початку списку одержимо ті остовні дерева, які мають мінімальну суму ваг.

Але оскільки число n^{n-2} *навіть при невеликому* n *буде дуже великим, то такий шлях розв'язування даної задачі досить трудомісткий.*

Тому актуальним є використання більш ефективних алгоритмів.

Попередні зауваження

1. **Порядок графа** – число, яке дорівнює кількості вершин графа.
2. **Порожній граф** – граф, що не містить ребер або регулярний граф степеня 0.

Алгоритм Краскала

Алгоритм Краскала полягає у виконанні такої послідовності дій:

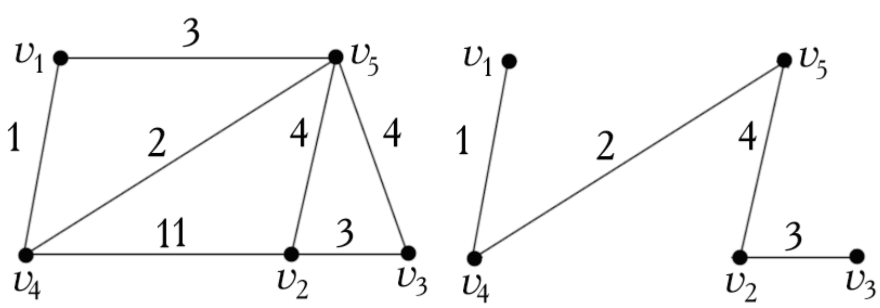
1. Беремо порожній граф O і будуємо граф $T_1 = O + e_1$, де e_1 – ребро графа $G = \left(V, E\right)$ мінімальної ваги.

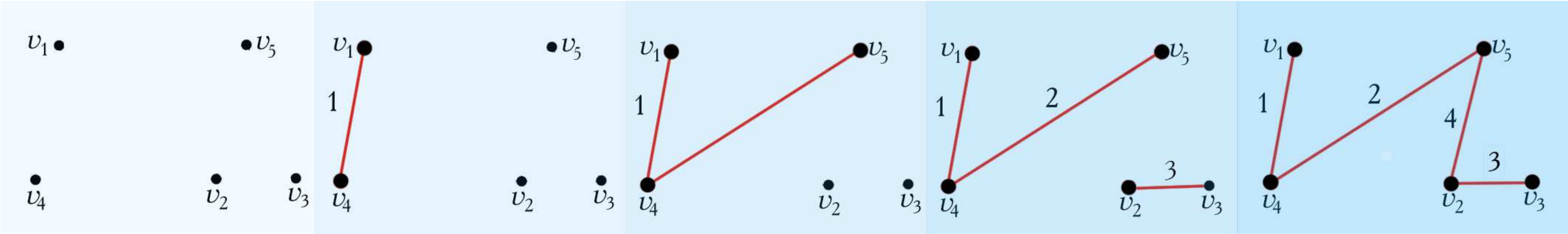
2. Якщо граф T_k уже побудований і $k < n - 1$, то будуємо граф $T_{k+1} = T_k + e_{k+1}$, де e_{k+1} – ребро мінімальної ваги серед ребер графа G, що не ввійшли в граф T_k, яке не становить циклу з ребрами графа T_k.

Цей алгоритм базується на спеціальній теоремі.

Теорема 7. Нехай G – зв'язний граф порядку n і T – підграф графа G, отриманий у результаті виконання кроків 1 і 2 алгоритму Краскала. Тоді підграф T – остовне дерево графа G мінімальної ваги.

Приклад. Нехай дано граф G, показаний на рисунку ліворуч. Необхідно знайти остовне дерево мінімальної ваги цього графа за допомогою алгоритму Краскала.
Розв'язок.





Властивості алгоритму Краскала

1. Алгоритм використовується для зважених неорієнтованих графів.

2. Алгоритм може бути використаний для побудови остовного лісу в багатокомпонентних незв'язних графах. У цьому випадку **структури даних**, що описують кожну з компонент зв’язності, **повинні бути окремими**.

3. Обчислювальна складність алгоритму Краскала $O\left(e \cdot \log e\right)$, де e – кількість ребер у даному графі.

Алгоритм Прима

Алгоритм Прима полягає у виконанні такої послідовності дій:

1. У порожньому графі O вибираємо довільну **вершину**.

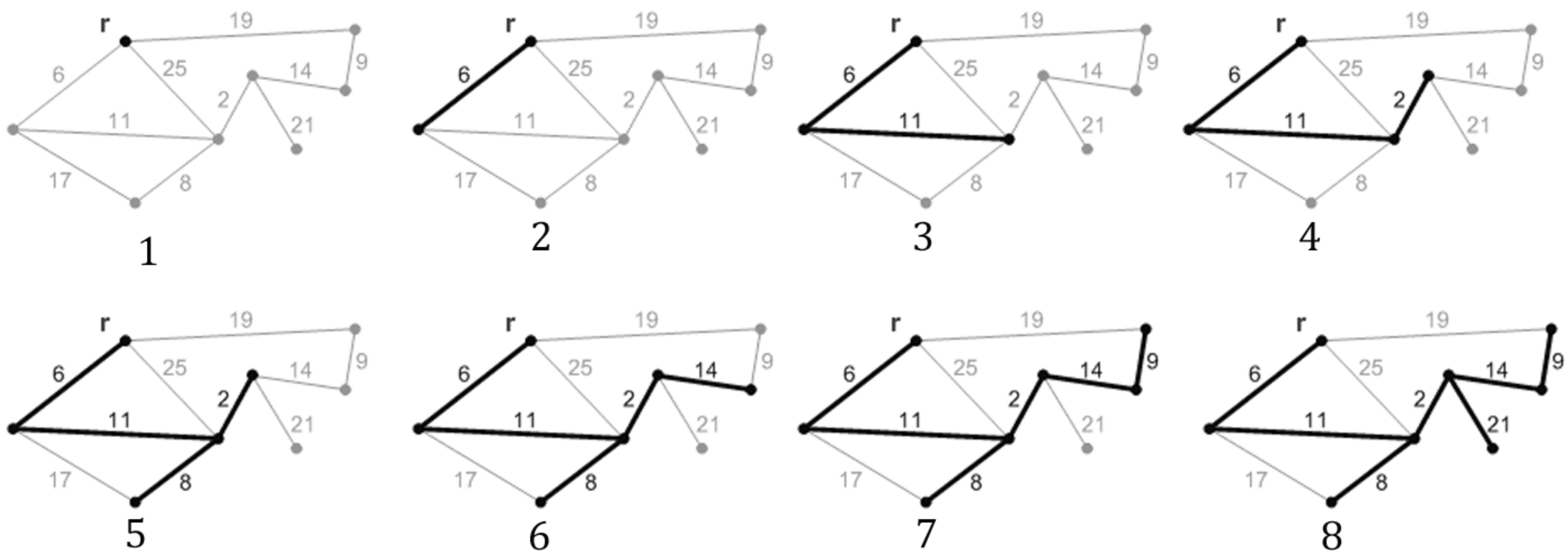
2. Вибираємо ребро e_1 мінімальної ваги до суміжної вершини і будуємо підграф $T_1 = O + e_1,$.

3. Якщо граф T_k уже побудований і $k < n - 1$, то будуємо граф $T_{k+1} = T_k + e_{k+1}$, де e_{k+1} – ребро мінімальної ваги, що з'єднує вершину графа T_k із суміжною вершиною, яка не включена в множину вершин графа T_k.

Приклад. Нехай даний граф G. Необхідно знайти остовне дерево мінімальної ваги цього графа за допомогою алгоритму Прима.

Розв'язок. 1.Вибираємо довільну вершину r і проводимо інцидентне ребро мінімальної ваги.

2. Потім, переглядаючи інцидентні ребра на кожній з кінцевих вершин, знаходимо ребро мінімальної ваги.



Властивості алгоритму Прима

1. Алгоритм використовується для зважених **неорієнтованих зв'язних** графів.

2. Обчислювальна складність алгоритму Прима $O\left(n^2\right)$, де n – кількість вершин графа. Якщо значення n достатньо **велике**, то використовувати цей алгоритм **не раціонально**.

3. Якщо кількість ребер e **значно менша, ніж** n^2, то алгоритм Краскала кращий, але якщо e **близька до** n^2, то рекомендують застосовувати алгоритм Прима.

Структура алгоритму Прима

Даний граф $G(V, E)$, де множина вершин $V = \{1, 2, ..., i, ..., n\}$

$U = \varnothing$- множина вершин остовного дерева.

$T = \varnothing$- множина ребер остовного дерева.

```
procedure Prim (G: граф; var T: множина ребер;
var U: множина вершин;
  u,v: вершина;
begin
 T := ∅; U := {i};
 while U ≠ V do
 begin
  Знаходження ребра (u,v) найменшої вартості і
  такого, що u∈U і v∈V\U
  T:= T ∪ {(u,v)};
  U:= U ∪ {v}
 end
end.
```

Обхід графів. Основні положення

Обійти граф – це значить побувати у всіх вершинах точно по одному разу.

Алгоритми обходу складаються з:

1. Послідовного відвідування вершин. При цьому:

   1.1. Вершину, яка ще **не відвідана, називають новою**.

   1.2. **Факт відвідування вершини запам'ятовується**, так що з моменту відвідування і до кінця роботи алгоритму вона вважається відвіданою.

   1.3. **У результаті відвідування** вершина **стає відкритою** і залишається такою, поки не будуть досліджені всі інцидентні їй ребра.

   1.3. **Після дослідження** всіх інцидентних **ребер** вона перетворюється ***в закриту***.

2. Дослідження ребер.

Які саме дії виконуються при відвідуванні вершини і дослідженні ребра – залежить від конкретної задачі

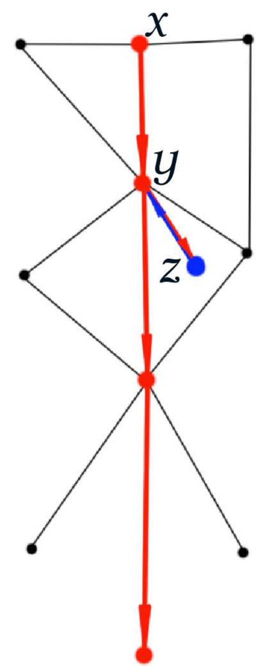
Алгоритми обходу графа: - обхід графа в глибину

-обхід графа в ширину.

Обхід у глибину

Обхід у глибину – це обхід графа за такими правилами:

1. Перебуваючи у вершині x, потрібно рухатися **в будь-яку іншу** y, раніше не відвідану вершину (якщо така знайдеться), одночасно **запам'ятовуючи ребро**, по якому ми вперше потрапили в дану вершину.



2. Якщо з вершини z ми **не можемо потрапити** в раніше не відвідану вершину або такої немає, то ми **повертаємося у вершину** y, з якої вперше потрапили в z, і **продовжуємо обхід** у глибину з вершини y.

При виконанні обходу графа за цими правилами ми прагнемо проникнути "углиб" графа так далеко, як тільки це можливо, потім відступаємо на крок назад і знову прагнемо пройти вперед, і так далі.

Приклад обходу графа в глибину

Пошук у глибину починаємо з будь-якої вершини. Рекурсивно застосуємо до всіх вершин, у які можна потрапити з поточної, таку послідовність дій.

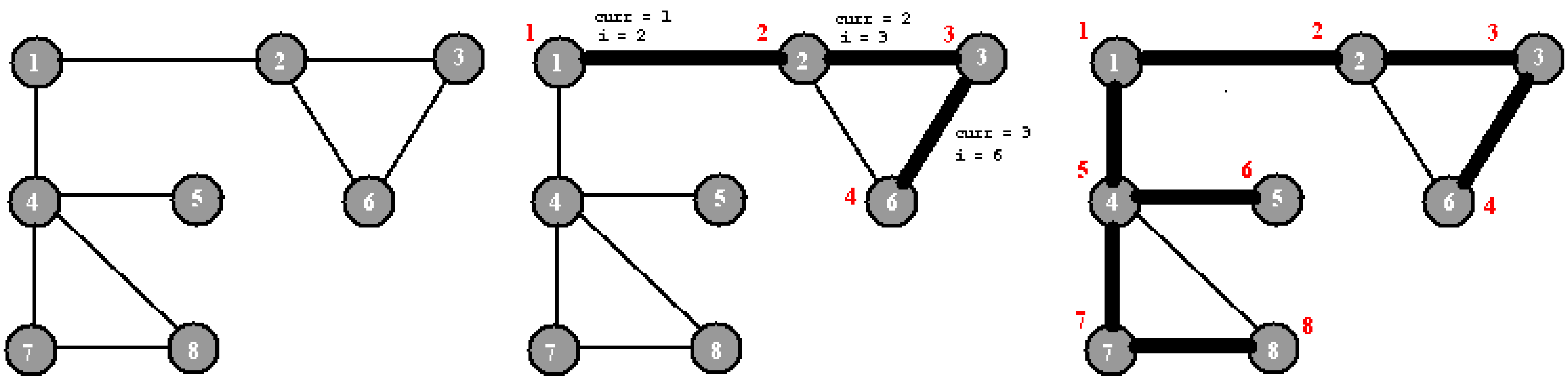
1. Задамо граф з N вершин матрицею суміжності A розміром $N \times N$ у вигляді масиву $A[N, N]$.

2. Задамо одномірний масив $Visited[N]$ і заповнимо його нулями, уважаючи, що жодна вершина до початку виконання алгоритму не була відвідана.

3. Задамо рекурсивну процедуру обходу графа в глибину.

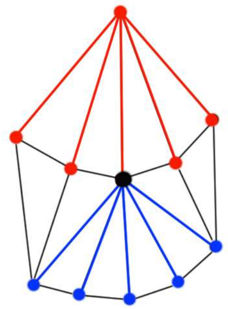
```
Program Depth
Procedure Go(Curr:Integer);
begin
 Visited[Curr]:=1;  {Позначаємо поточну верш. як пройдену}
 For i=1 to N do
 begin
  If Visited[i]=0 AND (A[Curr,i]=1) then Go(i);
 end;
end;

Begin Go(Start) end.
```



Обхід в ширину

Обхід в ширину – це обхід графа за наступними правилами:



2. Нехай ми перебуваємо у початковій вершині a.

3. Із цієї вершини відбувається перегляд відразу всіх ще не переглянутих вершин, суміжних вершині a. Таким чином, пошук ведеться у всіх можливих напрямках одночасно.

4. Потім проглядаються вершини, що перебувають від a на відстані 2, і т.д.

Чим ближче вершина до стартової вершини, тем раніше вона буде відвідана.

Обхід в ширину шукає найкоротший можливий шлях.

Приклад обходу графа в ширину

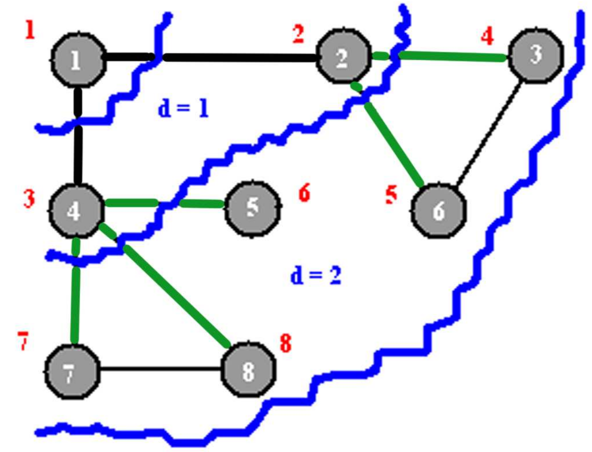
При пошуку в ширину замість стека рекурсивних викликів **використовується черга**, у яку записуються вершини в порядку видалення від початкової.

1. **Задамо масив** $Visited[N]$ і заповнимо його нулями, вважаючи, що жодна вершина до початку виконання алгоритму не була відвідана.

2. **Створимо чергу** для зберігання вершин у вигляді масиву $Queue[N]$. У початок черги запишемо початкову вершину.

3. **Змінна** r вказує на позицію в черзі, з якої ми **читаємо** дані.

4. **Змінна** w вказує на позицію в черзі, куди ми будемо **писати** дані.
5. Задамо процедуру обходу в ширину.




```
r:= 0, w:=1;

While (r < w) do
begin
  r:=r+1;
  Curr = queue[r]; {вибрали активну вершину}
  For i:= 1 to N do {відвідуємо всі суміжні вершини}
  begin
   if ( (Visited[i]=0) AND (A[curr,i]=1) do
   begin
    Visited[i]:= 1; {відзначаємо відвідану вершину}
    w:=w+1;
    queue[w]:= i; {ставимо її в чергу активних вершин}
   end;
  end;
end;
```