Паралельний алгоритм Флойда

Цей алгоритм більш загальний порівняно з алгоритмом Дейкстри, так як він знаходить найкоротші шляхи між будь-якими двома вузлами мережі. У цьому алгоритмі мережа представлена у вигляді квадратної матриці з n рядками і n стовпцями. Елемент (i, j) дорівнює відстані d_{ij} від вузла i до вузла j, яке має кінцеве значення, якщо існує дуга (i, j), і дорівнює нескінченності в іншому випадку.

Покажемо спочатку основну ідею методу Флойда. Нехай є три вузли i, j і k і задані відстані між ними (рис. 1). Якщо виконується нерівність $d_{ij} + d_{jk} < d_{ik}$, то доцільно замінити шлях i -> k шляхом i -> j -> k. Така заміна (далі її будемо умовно називати трикутним оператором) виконується систематично в процесі виконання алгоритму Флойда.

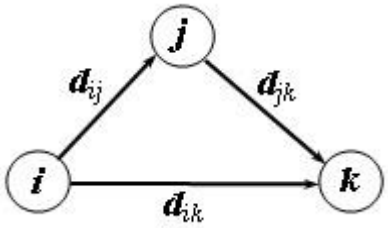


Рис.1. Трикутний оператор

Алгоритм Флойда вимагає виконання наступних дій.

Крок 0. Визначаємо початкову матрицю відстані D_0 і матрицю послідовності вузлів S_0. Діагональні елементи обох матриць позначаються знаком "-", що показує, що ці елементи в обчисленнях не беруть участь. Вважаємо k = 1:

$D_0 =$

	1	2	…	j	…	n
1	—	d_{12}	…	d_{1j}	…	d_{1n}
2	d_{21}	—	…	d_{2j}	…	d_{2n}
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
i	d_{i1}	d_{i2}	…	d_{ij}	…	d_{in}
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
n	d_{n1}	d_{n2}	…	d_{nj}	…	—

$S_0 =$

	1	2	…	j	…	n
1	—	2	…	j	…	n
2	1	—	…	j	…	n
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
i	1	2	…	j	…	n
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
n	1	2	…	j	…	—

Рис.2. Початкова ситуація

Основний крок k. Задаємо рядок k і стовпець k як провідний рядок і ведучий стовпець. Розглядаємо можливість застосування трикутного оператора до всіх елементів d_{ij} матриці D_{k-1}. Якщо виконується нерівність $d_{ik} + d_{kj} < d_{ij}$, (i <> k, j <> k, i <> j), тоді виконуємо наступні дії:

- створюємо матрицю D_k шляхом заміни в матриці D_{k-1} елемента d_{ij} на суму $d_{ik} + d_{kj}$,

- створюємо матрицю S_k шляхом заміни в матриці S_{k-1} елемента s_{ij} на k. Вважаємо k = k + 1 і повторюємо крок k.

Після реалізації n кроків алгоритму визначення за матрицями D_n і S_n найкоротшого шляху між вузлами i, j виконується за такими правилами.

- Відстань між вузлами i, j рівна елементу d_{ij} в матриці D_n.
- Проміжні вузли шляху від вузла i до вузла j визначаємо по матриці S_n. Нехай s_{ij} = k, тоді маємо шлях i -> k -> j. Якщо далі s_{ik} = k і s_{kj} = j, тоді вважаємо, що весь шлях визначений, так як знайдені всі проміжні вузли. В іншому випадку повторюємо описану процедуру для шляхів від вузла i до вузла k і від вузла k до вузла j.

Як приклад знайдемо для мережі, показаної на рисунку 3, найкоротші шляхи між будь-якими двома вузлами. Відстань між вузлами цієї мережі проставлені на малюнку біля відповідних ребер. Ребро (3, 5) орієнтоване, тому не допускається рух від вузла 5 до вузла 3. Всі інші ребра допускають рух в обидві сторони:

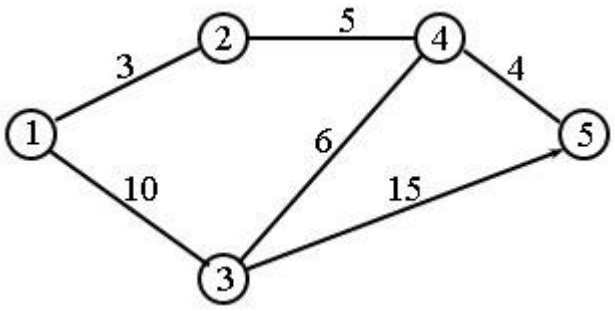


Рис.3. Приклад мережі

Крок 0. Початкові матриці D_0 і S_0 будуються безпосередньо за заданою схемою мережі. Матриця D_0 симетрична, за винятком пари елементів d_{35} і d_{53}, де d_{53} рівний нескінченності, оскільки неможливий перехід від вузла 5 до вузла 3:

D_0

	1	2	3	4	5
1	—	3	10	∞	∞
2	3	—	∞	5	∞
3	10	∞	—	6	15
4	∞	5	6	—	4
5	∞	∞	∞	4	—

S_0

	1	2	3	4	5
1	—	2	3	4	5
2	1	—	3	4	5
3	1	2	—	4	5
4	1	2	3	—	5
5	1	2	3	4	—

Рис.4. Початковий стан

Крок 1. У матриці D_0 виділені провідні рядок і стовпець (k = 1). Подвійною рамкою представлені елементи d_{23} і d_{32}, єдині серед елементів матриці D_0, значення яких можна поліпшити за допомогою трикутного оператора. Таким чином, щоб на основі матриць D_0 і S_0 отримати матриці D_1 і S_1, виконуємо наступні дії.

1. Замінюємо d_{23} на $d_{21} + d_{13} = 3 + 10 = 13$ і встановлюємо $s_{23} = 1$.

2. Замінюємо d_{32} на $d_{31} + d_{12} = 10 + 3 = 13$ і встановлюємо $s_{32} = 1$.

Матриці D_1 і S_1 мають такий вигляд:

D_1

	1	2	3	4	5
1	—	3	10	∞	∞
2	3	—	**13**	5	∞
3	10	**13**	—	6	15
4	∞	5	6	—	4
5	∞	∞	∞	4	—

S_1

	1	2	3	4	5
1	—	2	3	4	5
2	1	—	**1**	4	5
3	1	**1**	—	4	5
4	1	2	3	—	5
5	1	2	3	4	—

Рис.5. Матриці D_1 і S_1

Крок 2. Вважаємо k = 2; в матриці D_1 виділені провідні рядок і стовпець. Трикутний оператор застосовується до елементів матриці D_1 і S_1, виділеним подвійною рамкою. В результаті отримуємо матриці D_2 і S_2:

D_2

	1	2	3	4	5
1	—	3	10	**8**	∞
2	3	—	13	5	∞
3	10	13	—	6	15
4	**8**	5	6	—	4
5	∞	∞	∞	4	—

S_2

	1	2	3	4	5
1	—	2	3	**2**	5
2	1	—	1	4	5
3	1	1	—	4	5
4	**2**	2	3	—	5
5	1	2	3	4	—

Рис.6. Матриці D_2 і S_2

Крок 3. Вважаємо k = 3; в матриці D_2 виділені провідні рядок і стовпець. Трикутний оператор застосовується до елементів матриці D_2 і S_2, виділеним подвійною рамкою. В результаті отримуємо матриці D_3 і S_3:

D_3

	1	2	3	4	5
1	—	3	10	8	**25**
2	3	—	13	5	**28**
3	10	13	—	6	15
4	8	5	6	—	4
5	∞	∞	∞	4	—

S_3

	1	2	3	4	5
1	—	2	3	2	**3**
2	1	—	1	4	**3**
3	1	1	—	4	5
4	2	2	3	—	5
5	1	2	3	4	—

Рис.7. Матриці D_3 і S_3

Крок 4. Вважаємо k = 4, провідні рядок і стовпець у матриці D_3 виділені. Отримуємо нові матриці D_4 і S_4:

D_4

	1	2	3	4	5
1	—	3	10	8	**12**
2	3	—	**11**	5	**9**
3	10	**11**	—	6	**10**
4	8	5	6	—	4
5	**12**	**9**	**10**	4	—

S_4

	1	2	3	4	5
1	—	2	3	2	**4**
2	1	—	**4**	4	**4**
3	1	**4**	—	4	**4**
4	**2**	2	3	—	5
5	**4**	**4**	**4**	4	—

Рис.8. Матриці D_4 і S_4

Крок 5. Вважаємо k = 5, провідні рядок і стовпець у матриці D4 виділені. Ніяких дій на цьому етапі не виконує; обчислення закінчені.

Кінцеві матриці D_4 і S_4 містять всю інформацію, необхідну для визначення найкоротших шляхів між будь-якими двома вузлами мережі. Наприклад, найкоротша відстань між вузлами 1 і 5 одно $d_{15} = 12$.

Для знаходження відповідних маршрутів нагадаємо, що сегмент маршруту (i, j) складається з ребра (i, j) тільки в тому випадку, коли $s_{ij} = j$. В іншому випадку вузли i, j пов'язані мінімум через один проміжний вузол. Наприклад, оскільки $s_{15} = 4$ і $s_{45} = 5$, спочатку найкоротший маршрут між вузлами 1 і 5 буде мати вигляд 1-> 4> 5. Але так як s_{14} не дорівнює 4, вузли 1 і 4 в певному шляху не пов'язані одним ребром (але у вихідній мережі вони можуть бути зв'язані безпосередньо). Далі слід визначити проміжний вузол (вузли) між першим і четвертим вузлами. Маємо $s_{14} = 2$ і $s_{24} = 4$, тому маршрут 1-> 4 замінюємо 1-> 2> 4. Оскільки $s_{12} = 2$ і $s_{24} = 4$, інших проміжних вузлів немає. Комбінуючи певні сегменти маршруту, остаточно отримуємо наступний найкоротший шлях від вузла 1 до вузла 5: 1-> 2> 4> 5. Довжина цього шляху дорівнює 12 кілометрам.

Поділ обчислень на незалежні частини

Як випливає із загальної схеми алгоритму Флойда, основне обчислювальне навантаження при вирішенні завдання пошуку найкоротших шляхів полягає у виконанні операції вибору мінімальних значень. Дана операція є досить простою, і її розпаралелювання не призведе до помітного прискорення обчислень. Більш ефективний спосіб організації паралельних обчислень може складатися в одночасному виконанні декількох операцій оновлення значень матриці D і S.

Покажемо коректність такого способу організації паралелізму. Для цього потрібно довести, що операції оновлення значень матриці D на одній і тій же ітерації зовнішнього циклу k можуть виконуватися незалежно. Іншими словами, слід показати, що на ітерації k не відбувається зміна елементів D_{ik} і D_{kj} ні для однієї пари індексів (i, j). Розглянемо вираз, за яким відбувається зміна елементів матриці D:

$D_{ij} <- \min(D_{ij}, D_{ik} + D_{kj})$.

Для i = k отримаємо

D_{kj} <- min(D_{kj}, D_{kk} + D_{kj}), але тоді значення D_{kj} не зміниться, тому що D_{kk} = 0.

Для j = k вираз перетвориться до виду D_{ik} <- min(D_{ik}, D_{ik} + D_{kk}), що також показує незмінність значень D_{ik}. Як результат, необхідні умови для організації паралельних обчислень забезпечені, і, тим самим, в якості базової підзадачі може бути використана операція оновлення елементів матриці D.

Виділення інформаційних залежностей

Виконання обчислень в підзадачах стає можливим тільки тоді, коли кожна підзадача (i, j) містить необхідні для розрахунків елементи D_{ij}, D_{ik}, D_{kj} матриці D. Для виключення дублювання даних розмістимо в підзадачі (i, j) єдиний елемент D_{ij}, тоді отримання всіх інших необхідних значень може бути забезпечено тільки за допомогою передачі даних. Таким чином, кожен елемент D_{kj} рядка k матриці D повинен бути переданий всім підзадачам (k, j), 1 <= j <= n, а кожен елемент D_{ik} стовпця k матриці D повинен бути переданий всім підзадачам (i, k), 1 < = i <= n, - рис. 9.

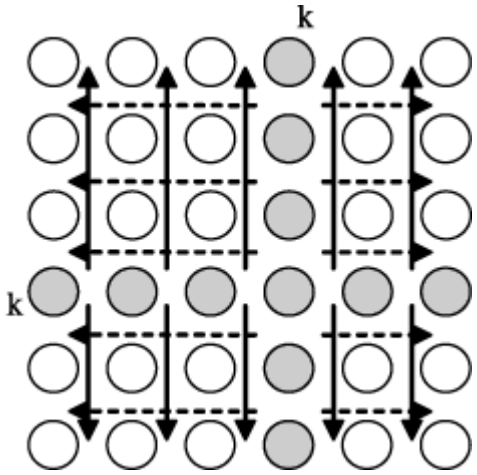


Рис.9. Інформаційна залежність базових підзадач (стрілками показані напрямки обміну значеннями на ітерації k)

Масштабування і розподіл підзадач процесорам

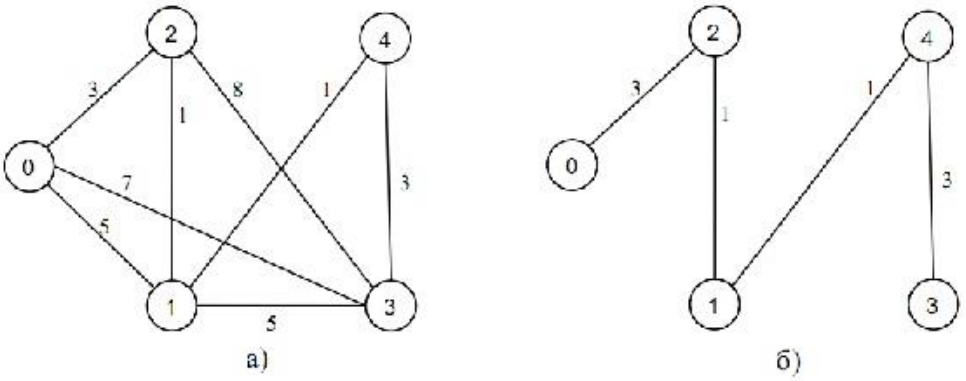
Як правило, число доступних процесорів p істотно менше, ніж число базових задач n^2 (p << n^2). Можливий спосіб укрупнення обчислень полягає у використанні стрічкової схеми розбиття матриці D - такий підхід відповідає об'єднанню в рамках однієї базової підзадачі обчислень, пов'язаних з оновленням елементів однієї або декількох рядків (горизонтальне розбиття) або стовпців (вертикальне розбиття) матриці A. Ці два типи розбиття практично рівноправні - з огляду на додатковий момент, що для алгоритмічної мови C масиви розташовуються по рядках, будемо розглядати далі тільки розбиття матриці A на горизонтальні смуги.

Слід зазначити, що при такому способі розбиття даних на кожній ітерації алгоритму Флойда потрібно передавати між підзадачами тільки елементи одного із рядків матриці D. Для оптимального виконання подібної комунікаційної операції топологія мережі повинна забезпечувати ефективне представлення структури мережі передачі даних у вигляді гіперкуба або повного графа.

Паралельний алгоритм Прима

Охоплючим деревом неорієнтованого графа G називається підграф T графа G, який є деревом і містить всі вершини з G. Визначивши ваги підграфа для зваженого графа як суму ваг, що входять в підграф дуг, під мінімальним охоплючим деревом або МОД T будемо розуміти охоплюче дерево мінімальної ваги.

Приклад неорієнтованого зваженого графа і відповідного йому МОД:



Алгоритм пошуку мінімального охоплює дерева в підвішеному графі носить назву методу Прима (Robert Prim).

Алгоритм починає роботу з довільної вершини графа, обраній в якості кореня дерева, і в ході послідовно виконуваних ітерацій розширює конструйоване дерево до МОД. Нехай V_T є множина вершин, вже включених алгоритмом в МОД, а величини d_i, $1 <= i <= n$, характеризують дуги мінімальної довжини від вершин, ще не включених в дерево, до множини V_T, тобто

$$\forall i \notin V_T \Rightarrow d_i = \min\{w(i,u) : u \in V_T, (i,u) \in R\}$$

(якщо для будь-якої вершини $i \notin V_T$ не існує жодної дуги в V_T, значення d_i встановлюється рівним ∞). На початку роботи алгоритму вибирається коренева вершина МОД s і встановлюється $V_T = \{s\}$, $d_s = 0$.

Дії, що виконуються на кожній ітерації алгоритму Прима, полягають у наступному:

- визначаються значення величин d_i для всіх вершин, ще не включених до складу МОД;
- вибирається вершина t графа G, що має дугу мінімальної ваги до множини $V_T t : d_t,\ i \notin V_T$ вершина t включається в V_T.

Після виконання n-1 ітерації методу МОД буде сформовано. Вага цього дерева може бути отримана за допомогою виразу

$$W_T = \sum_{i=1}^{n} d_i.$$

Трудомісткість знаходження МОД характеризується квадратичною залежністю від числа вершин графа $T_1 \sim n^2$.

Поділ обчислень на незалежні частини

Оцінимо можливості паралельного виконання розглянутого алгоритму знаходження мінімально охоплюючого дерева.

Ітерації методу повинні виконуватися послідовно і, тим самим, не можуть бути розпаралелені. З іншого боку, дії що виконуються на кожній ітерації алгоритму є незалежними і можуть реалізовуватися одночасно. Так, наприклад, визначення величин d_i може здійснюватися для кожної вершини графа окремо, знаходження дуги мінімальної ваги може бути реалізовано за каскадною схемою і т.д.

Розподіл даних між процесорами обчислювальної системи має забезпечувати незалежність перерахованих операцій алгоритму Прима. Зокрема, це може бути реалізовано, якщо кожна вершина графа розташовується на процесорі разом з усією пов'язаною з вершиною інформацією. Дотримання цього принципу призводить до того, що при рівномірному завантаженні кожен процесор p_j, $1 <= j <= p$, повинен містити:

набір вершин

$$V_j = \{\nu_{i_j+1}, \nu_{i_j+2}, \ldots, \nu_{i_j+k}\},\ i_j = k \cdot (j-1), k = \lceil n/p \rceil;$$

блок з k величин, що відповідає цьому набору

$$\Delta_j = \{d_{i_j+1}, d_{i_j+2}, \ldots, d_{i_j+k}\};$$

вертикальну смугу матриці суміжності графа G з k сусідніх стовпців

$$A_j = \{\alpha_{i_j+1}, \alpha_{i_j+2}, \ldots, \alpha_{i_j+k}\}$$

α_s є s-й стовпець матриці A;

загальну частину набору V_j і того, що формується в процесі обчислень множини вершин V_T.

Можемо зробити висновок, що базовою підзадачею в паралельному алгоритмі Прима може служити процедура обчислення блоку значень Δ_j для вершин V_j матриці суміжності A графа G.

Виділення інформаційних залежностей

З урахуванням вибору базових підзадач загальна схема паралельного виконання алгоритму Прима буде полягати в наступному:

- визначається вершина t графа G, що має дугу мінімальної ваги до множини V_T. Для вибору такої вершини необхідно здійснити пошук мінімуму в наборах величин d_i, наявних на кожному з процесорів, і виконати збірку отриманих значень на одному з процесорів;

- номер обраної вершини для включення в охоплююче дерево передається всім процесорам;
- оновлюються набори величин d_i з урахуванням додавання нової вершини.

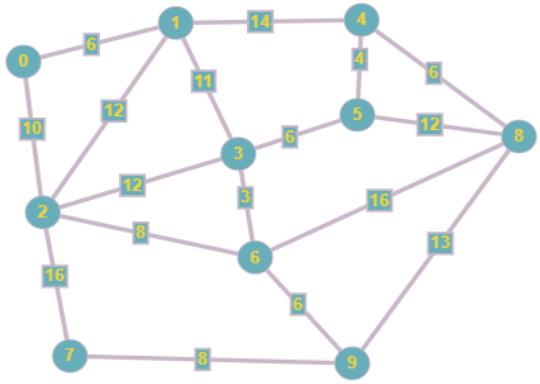
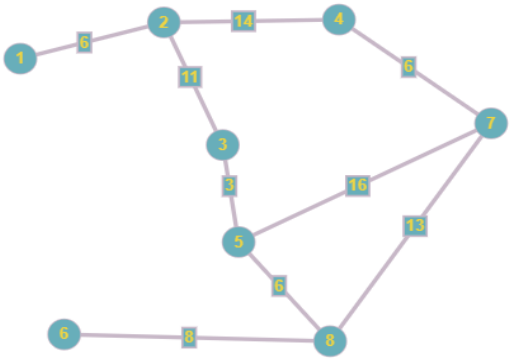
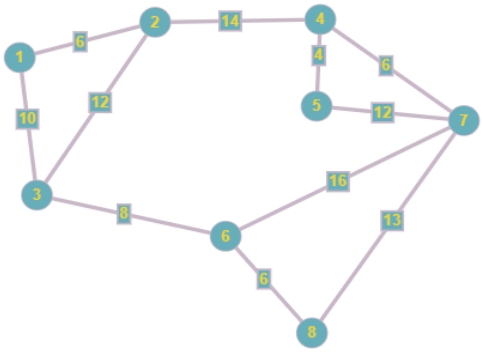
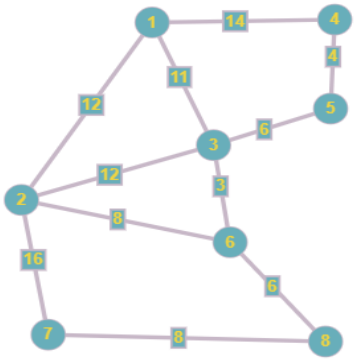
Таким чином, в ході паралельних обчислень між процесорами виконуються два типи інформаційних взаємодій: збір даних від усіх процесорів на одному з процесорів і передача повідомлень від одного процесора всім процесорам обчислювальної системи.

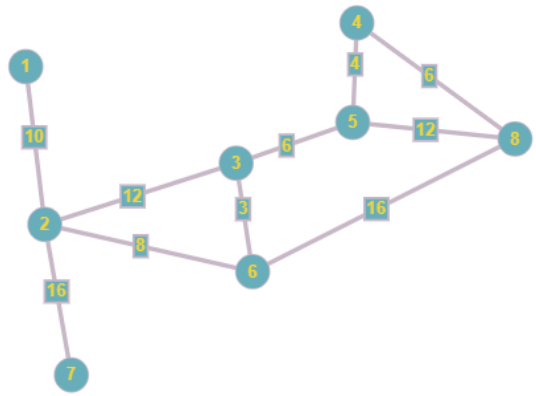
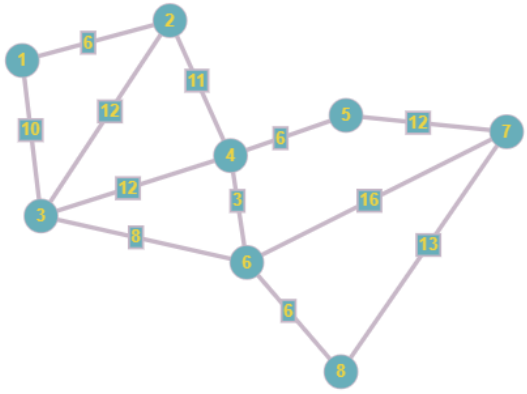
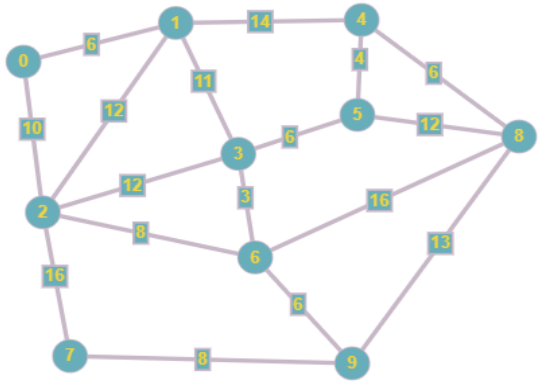
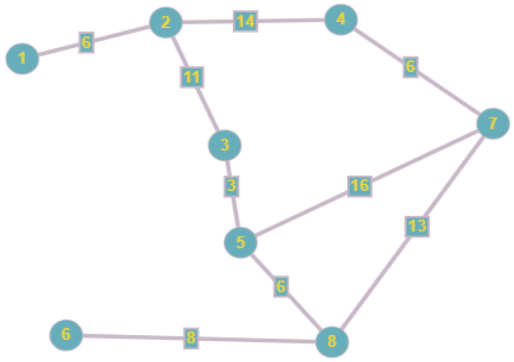
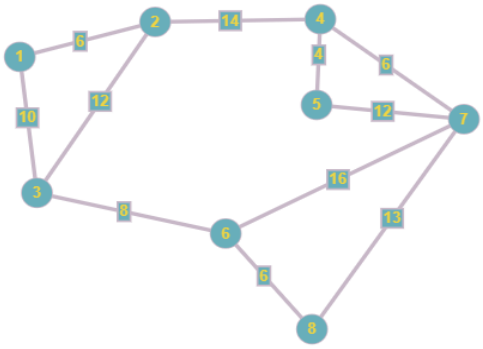
Масштабування і розподіл підзадач процесорам

За визначенням кількість базових підзадач завжди відповідає числу наявних процесорів, і, тим самим, проблема масштабування для паралельного алгоритму не виникає.

Розподіл підзадач між процесорами має враховувати характер виконуваних в алгоритмі Прима комунікаційних операцій. Для оптимальної реалізації необхідних інформаційних взаємодій між базовими підзадачами топологія мережі передачі даних повинна забезпечувати ефективне представлення у вигляді гіперкуба або повного графа.

Завдання

1. Алгоритм Флойда 	2. Алгоритм Флойда 
3. Алгоритм Флойда 	4. Алгоритм Флойда 

5. Алгоритм Флойда  	6. Алгоритм Прима 
7. Алгоритм Прима  	8. Алгоритм Прима 
9. Алгоритм Прима  	10. Алгоритм Прима