ЛАБОРАТОРНАЯ РАБОТА № 8
ИССЛЕДОВАНИЕ АЛГОРИТМОВ ПОИСКА КРАТЧАЙШЕГО ПУТИ НА ГРАФЕ

ЦЕЛЬ РАБОТЫ: программно реализовать и исследовать эффективность алгоритмов поиска кратчайшего пути на графе с использованием функций библиотеки MPI.

1 ТЕОРЕТИЧЕСКОЕ СВЕДЕНИЯ

Математические модели в виде графов широко используются при моделировании разнообразных явлений, процессов и систем. Как результат, многие теоретические и реальные прикладные задачи могут быть решены при помощи тех или иных процедур анализа графовых моделей. Среди множества этих процедур может быть выделен некоторый определенный набор типовых алгоритмов обработки графов. Рассмотрению вопросов теории графов, алгоритмов моделирования, анализу и решению задач на графах посвящено достаточно много различных изданий.

1.1 Общие понятия теории графов

Ориентированный граф обозначается как $G = (V, R)$, где V — множество вершин графа, R — множество дуг.

Вершина $v_j \in V$, где $j = \overline{1,n}$, а $n = |V|$

Дуга $r_{jk} = (v_j, v_k) \in R \subset V^2, m = |R|$

В общем случае дугам графа могут быть составлены числовые характеристики (веса) $W_{jk} = W(r_{jk}) = W(v_j, v_k)$.

Матрица смежности $A = \{ a_{jk} | \, j = \overline{1,n} \ , k = \overline{1,n} \}$ - способ представления графа .

Элемент a_{jk} определяется следующим образом :

$$\begin{Bmatrix} W(v_j, v_k), (v_j, v_k) \in R \\ 0, j = k \text{ и } (v_j, v_k) \notin R \\ \infty, (v_j, v_k) \in R \text{ и } j \neq R \end{Bmatrix}$$

т.е. отсутствие дуги – бесконечно большое значение.

1.2 Задача о кратчайшем пути на графе

Задан ориентированный граф $G = (V, R)$ с весовой функцией $W: R \to R^+$(в общем виде $W: R \to R$, где R- действительная ось).

Путь между вершиной v_0 и некоторой вершиной v_j следующим образом:

$\mathrm{p} = < \mathrm{v}_0 , \mathrm{v}_1 , \dots , \mathrm{v}_\mathrm{j} >$ либо в виде $\mathrm{v}_0 \xrightarrow{p} \mathrm{v}_\mathrm{j}$.

Вес пути – суммарный вес всех ребер, в него входящих:

$$\mathrm{W(P)} = \sum_{\mathrm{j=1}}^{\mathrm{k}} \mathrm{w}(\mathrm{v}_{\mathrm{j}-1} , \mathrm{v}_\mathrm{j})$$

Вес кратчайшего пути из вершины v_j

$$\delta(v_j , v_k) = \begin{cases} \min_P W(P) & \exists\, \mathrm{v}_\mathrm{j} \longrightarrow^{\mathrm{p}} \mathrm{v}_\mathrm{k} \\ \infty , \nexists\, \mathrm{v}_\mathrm{j} \longrightarrow^{\mathrm{p}} \mathrm{v}_\mathrm{k} \end{cases}$$

Формулировка задачи о кратчайшем пути:

Для заданного ориентированного взвешенного графа $\mathrm{G} = (\mathrm{V}, \mathrm{R})$ требуется найти кратчайшие пути $\mathrm{v}_0 \to \mathrm{v}_j$ от вершины $\mathrm{v}_0 \in V$ до всех вершин $\mathrm{v}_j \in V$.

Атрибуты для вершины графа (вершины v_i])

– атрибут “предшественник” $\pi[\mathrm{v}]$, где v_j- рассматриваемая вершина, тогда $\pi[\mathrm{v_i}]$ позволяет определить путь, обратный кратчайшему пути ,из рассматриваемой вершины v_j до исходной вершины v_j

– атрибут “оценка кратчайшего пути” $\mathrm{d}[v_i]$ – значение веса $\delta(v_0 , v_1)$, который обладает кратчайший путь из v_0 в v_i.

Процедура минимизации параметров $\pi[\mathrm{v_i}], \mathrm{d}[v_i]$.

$$\mathrm{INIT}(v_0) \forall\, v_i \in V \begin{bmatrix} \mathrm{d}[v_i] \leftarrow \infty \\ \pi[v_i] \leftarrow \text{"}\text{"} \end{bmatrix}$$
$$d[v_0 \leftarrow 0]$$

Если вершина v_i - текущая рассматриваемая, тогда для нее должен быть выполнен процесс обновления кратчайшего пути для вершины v_i .

Процесс (процедура) обновления пути реализуется для некоторого ребра $(v_j, v_i) \in R$

Процедура обновления предполагает проверку наличия возможности улучшения найденного кратчайшего пути к вершине v_i и в последующем обновлении атрибутов $\mathrm{d}[v_i]$ и $\pi[v_i]$ при наличии возможности улучшения .

В соответствии с реализацией процедуры обновления веса кратчайшего пути значение $\mathrm{d}[v_i]$ для предшествующей вершины v_j должно быть известно (т.е. фактически, кратчайший путь от v_0 до v_j должен быть известен ($v_j -$ предшественник v_i)).

Псевдокод процедуры обновления веса кратчайшего пути рассматриваемой вершины v_i

OBN (v_i, v_j) /*обновление веса вершины

Если $\mathrm{d}[v_i] > \mathrm{d}[v_j] + W(v_j, v_i)$, то $\begin{bmatrix} \mathrm{d}[v_j] \leftarrow \mathrm{d}[v_j] + W(v_j, v_i) \\ \pi[v_i] \leftarrow v_j \end{bmatrix}$

1.2.1 Алгоритм Дейкстры определения кратчайшего пути

Реализация алгоритма предполагает формирование множества вершин $S \subseteq V$ для которых вычислены окончательные веса $\delta(v_0, v_j)$ кратчайших путей к ним из вершины v_0.

В алгоритм последовательно (на каждом последующем шаге) выбирается вершина $v_j \in V\backslash S$, для которой $\mathrm{d}[v_j]$ является минимальным .

После добавления вершины v_j в множество S выполняется обновление весов дуг , исходящих из нее в множество вершин $v_i \in V\backslash S$.

Псевдокод процедуры:

INIT (v_0)

$S \leftarrow \emptyset$

$|V|$ раз выполнить : [определить $v_j \in V\backslash S : d\ [v_j = min]$] /*добавление в S вершины v из которой $d(v)$

$\forall v_i \in V\backslash S : (v_j, v_i \in R)$

OBN (v_j, v_i) /*определение новых $d\ [v_j]$;

Пример реализации процедуры алгоритма Дейкстры. Исходный граф представлен на рисунке 1.1.

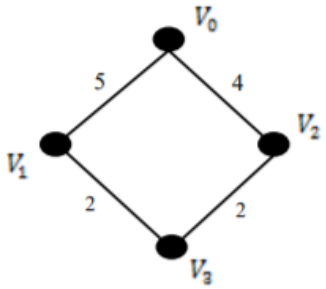

Рис. 1.1 — Исходный граф

1. $S = \emptyset$;
$d(V_0) = 0, V_0 \in V\backslash S$;
$d(V_0) = \min \forall\ V_j \in V\backslash S$;
$S = S \cup \{V_0\}$;
$\forall\ V_j \in V\backslash S: (V_0, V_i) \in R$;
$OBN(V_0, V_i)$;
Результат
$d(V_1) = 5;\ \pi[V_1] = V_0$;
$d(V_2) = 4;\ \pi[V_2] = V_0$;
$V = V\backslash\{V_0\}$;
2. $S = \{V_0\}$;
$d(V_2) = 4 = \min \forall\ V_j \in V\backslash S$;
$\mathrm{S} = \mathrm{S} \cup \{V_2\}$;
$\forall\ V_j \in V\backslash S: (V_2, V_i) \in R$;
$OBN(V_2, V_i)$;

Результат
$d(V_3) = 6;\ \pi[V_3] = V_2;$
$V = V \backslash \{V_2\};$
3. $S = \{V_0, V_2\}$
$d(V_1) = 5 = \min\ \forall V_i \in V \backslash S;$
$S = S \cup \{V_1\};$
$\forall V_j \in V \backslash S: (V_1, V_i) \in R;$
$OBN(V_1, V_i);$
Результат
$d(V_3) = 6;\ \pi[V_3] = V_2;$
$V = V \backslash \{V_1\};$
4. $S = \{V_0, V_2, V_1\}$
$d(V_3) = 6 = \min\ \forall V_i \in V \backslash S;$
$S = S \cup \{V_3\};$
$\forall V_j \in V \backslash S: (V_3, V_i) \in R;$
$OBN(V_3, V_i);$
Результат
$d(V_3) = 4;\ \pi[V_3] = V_0;$
$d(V_1) = 5;\ \pi[V_1] = V_0;$
Результирующий граф кратчайших путей представлен на рисунке 1.2.

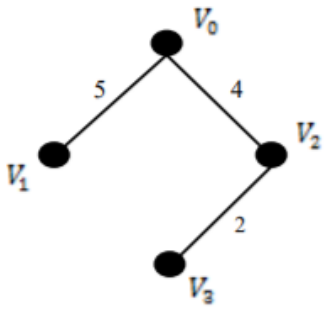

Рис. 1.2 — Итоговый граф кратчайшего пути

1.2.2 Параллельная реализация алгоритма Дейкстры

Параллельная реализация алгоритма предполагает обновление значений $d(V_i)$ и $\pi(V_i)$ для вершин $V_j \in V \backslash S$ таких, что:

$$(V_j, V_i) \in R \text{ и } V_j \in S$$

Таким образом, вершина V_i такая, что $d(V_j) = 6 = \min\ \forall V_j \in V \backslash S$, тогда $S = S \cup \{V_j\}$ и для V_j определяются те вершины V_i, для которых $(V_j, V_i) \in R$.

Тогда для всех $V_j \in V \backslash S$ таких, что $(V_j, V_i) \in R$ одновременно реализуется процедура $OBN(V_j, V_i)$.

1.2.3 Алгоритм построения кратчайшего пути Беллмана-Форда

Заданными являются: матрица весов *W* ориентированного графа *G(V, R)*, начальная вершина V_0.

Результат: кратчайшие расстояния от начальной вершины V_0 до каждой вершины $V_j \in V$, представленный в виде вектора $D[V_j]$, элемент $d(V_j)$ которого равен $d(V_0, V_j)$, где $d(V_0, V_j)$ – расстояние от V_0 до V_j.

Последовательность шагов алгоритма:

- Для вершины $V_j \in V$ реализуется присваивание $d(V_j) = W_0$ (в случае наличия $(V_j, V_i) \in R$), если $(V_j, V_i) \notin R$, тогда $d(V_j) = \infty$;
- Задание $k = 1$ (где k – количество итераций по определению кратчайшего пути от V_i до V_j);
- Выбор вершины $V_j \in V \backslash \{V_0\}$;
- Выбор вершины $V_i \in V$;
- Определение $d(V_j) = \min (d[V_j], d[V_i] + W_{ij})$;
- Если вершина V_i пробежала не все множество V, тогда выбор следующей вершины $V_i \in V$ и переход на шаг 5;
- Если вершина V_j пробежала не все множество вершин $V \backslash \{V_0\}$, тогда выбрать новую вершину V_j и переход на шаг 4;
- Если $k < n$ - 1 (т.е. должно быть выполнено (n - 1) итерация алгоритма), тогда $k = k + 1$ переход на шаг 3; при $k = n$ - 1 переход на шаг 9;
- Остановка алгоритма. Вектор $D[V_j]$ содержит расстояния $d(V_0, V_j)$ от V_0 до V_j.

Особенность алгоритма:

1. Шаг 5 – процедура обновления весов вершины(вес вершины – стоимость пути от V_0 до этой вершины V_j), процедура $OBN(V_j, V_i)$.

2. Последовательное выполнение шагов 6 и 7 обеспечивает рассмотрение всех дуг на графе $G(V, R)$.

Пример реализации алгоритма Беллмана-Форда для заданного графа.

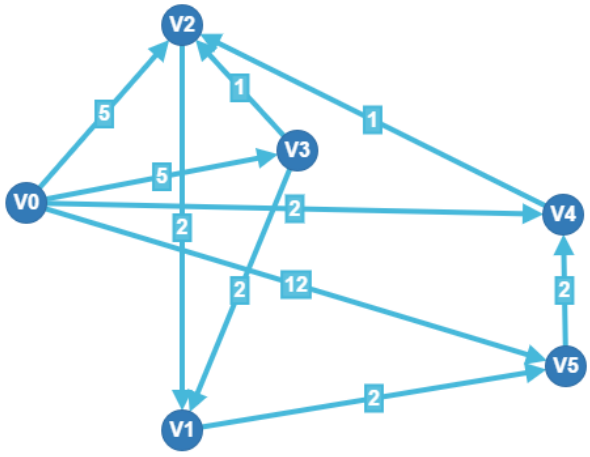

Рис. 1.3 — Исходный граф

D^0 – исходный вектор, элементы $d_i^0(d^0[v_j])$, которого на первом шаге алгоритма инициализируются следующим образом:

$d^0[v_j]=w_{0,j}$ при условии $(v_0,v_j)\in R$, если $(v_0, v_j)\notin R$ тогда $d^0[v_j]= \infty$.

1. Итерация – определение ∂^0.

$$d^1[u_j] = \min[d^0[u_1], d^0[u_j + w_{j1}] =$$
$$= \min d^1[u_j] = \min(d^0[u_1], d^0[u_2] + w_{21}; d^0[u_3] + w_{31};\ d^0[u_4]$$
$$+ w_{41};\ d^0[u_5] + w_{51}) = \min(\infty; 5+2; 5+2; 2+\infty; 12+\infty) = 7$$

$$d^1[u_2] = \min[d^0[u_2], d^0[u_j + w_{j2}] =$$
$$= \min(d^0[u_2], d^0[u_1] + w_{12}; d^0[u_3] + w_{32};\ d^0[u_4] + w_{42};\ d^0[u_5]$$
$$+ w_{52}) = \min(5; 7+\infty; 5+\infty; 2+1; 12+\infty) = 3$$

По аналогии:

$d^1[v_3] = 4$; $d^1[v_4] = 2$; $d^1[v_5] = 9$.

Повторяя подобные операции алгоритма, определяем векторы $\partial^{(2)}$, $\partial^{(3)}$, $\partial^{(4)}$ (всего n–1 итераций), которые сведены в таблицу

	v_0	v_1	v_2	v_3	v_4	v_5	∂^0	∂^1	∂^2	∂^3	∂^4
v_0	∞	∞	5	5	2	12	0	0	0	0	0
v_1	∞	∞	∞	∞	∞	2	∞	7	5	5	5
v_2	∞	2	∞	∞	∞	∞	5	3	3	3	3
v_3	∞	2	∞	∞	∞	∞	5	4	4	4	4
v_4	∞	∞	1	∞	∞	2	2	2	2	2	2
v_5	∞	∞	∞	∞	∞	∞	12	9	9	7	7

Понятно, что для каждой вершины при изменении ее стоимости (веса пути) $d[v_j]$ должен меняться параметр $\pi[v_j]$ — предшествующей вершины.

Особенности реализации алгоритма — при определении кратчайшего расстояния до одной из вершин реализуется определение кратчайшего расстояния до другой вершины.

Псевдокод алгоритма Беллмана-Форда:

INIT (v_0)

| V | – 1 раз повторить:

| $\forall(U_j, U_i) \in R$

| OBN (u_j, u_i);

$\forall(U_j, U_i) \in R$

| Если $d[v_i] > d[v_j] + W(v_j, v_i)$, то вернуть False и завершить работу.

Вернуть D;

1.2.4 Параллельная реализация алгоритма Беллмана-Форда

Исходные данные:

– наличие P процессов;

– $|R| >> 1$ (т.е. граф является плотно заполненным).

Плотно заполненный граф – число ребер близко к максимальному.

Матрица весов имеет вид ($W[v_i, v_j]$)

Разбиение матрицы весов $W[v_i, v_j]$ на подматрицы в соответствии с количеством процессов –> матрица $W[v_i, v_j]$ сеткой размера $(p+1)(p+1)$ разбивается на $(p+1)^2$ блоков.

Последовательность шагов параллельного алгоритма Беллмана-Форда.

1. С использованием P процессов выполняется обновление весов вершин $d[v_j]$ в верхних диагональных P блоках. (т.е. реализуется процедура OBN(v_j, v_i) для тех вершин, веса дуг которых входят в соответствующие блоки).

2. Обновление весов вершин $d[v_j]$ в нижнем диагональном блоке.

3. Для блоков каждой из вертикальных полос с P подматрицами (при исключении рассмотренного диагонального элемента) при использовании P вершин $d[v_j]$, веса дуг (v_j, v_i) для которых содержаться в выделенных подматрицах.

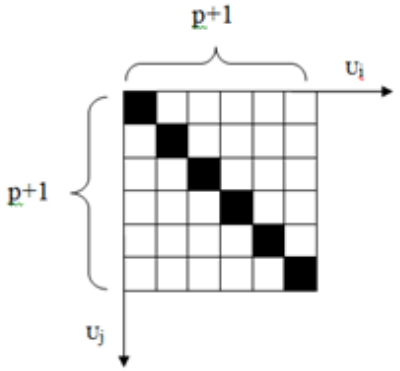

Рис. 1.4 — Сетка процессов и формирование подматриц весов

1.2.5 Параллельная реализация метода Беллмана для разрешенного графа

Разрешенный граф – граф, число дуг в котором близко к максимальному.

Если граф разрешенный, тогда q/б выполнено условие $|V|>>P$ (либо $n>>p$, где $n=|V|$).Тогда матрица смежности покрывается сеткой (n+1) (n+1).

Всего будет сформировано $(n+1)^2$ блоков.

Последовательность действий параллельного алгоритма.

1. Выполняется обновление весов вершин $d[v_j]$ в (n+1)-ом диагональном блоке.

2. Для каждой из горизонтальных полос параллельно выполняется обновление весов вершин для всех, внедиагональных квадратных подматриц (при условии $n>>p$).

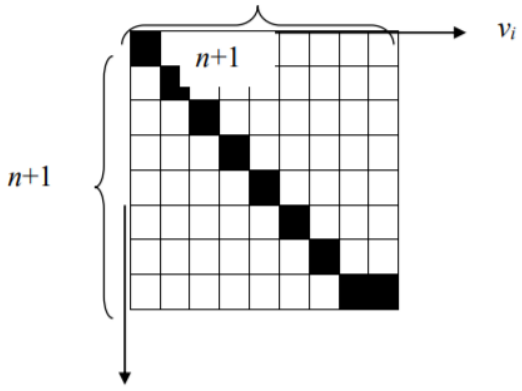

Рис. 1.5 — Разбиение матрицы смежности W(v_i, v_j) на подматрицы для разрешенного графа

2 ЗАДАНИЕ НА РАБОТУ

Выполнить разработку и отладку программы поиска кратчайшего пути с использованием вызовов требуемых функций библиотеки MPI для реализации алгоритма в соответствии с вариантом, указанным преподавателем. В качестве базового варианта также реализовать алгоритм последовательным методом. Получить результаты работы программы в виде протоколов сообщений, комментирующих параллельное выполнение процессов и их взаимодействие в ходе выполнения. Оценить эффективность параллельного алгоритма в сравнении с последовательным на том же наборе исходных данных.

2.1 Вариант № 1

Выполнить разработку программы, реализующей поиск кратчайшего пути на графе при помощи алгоритма Дейкстры. Применить разработанную процедуру к графу на рисунке 2.1 для поиска кратчайшего пути от вершины 0 к вершине 9.

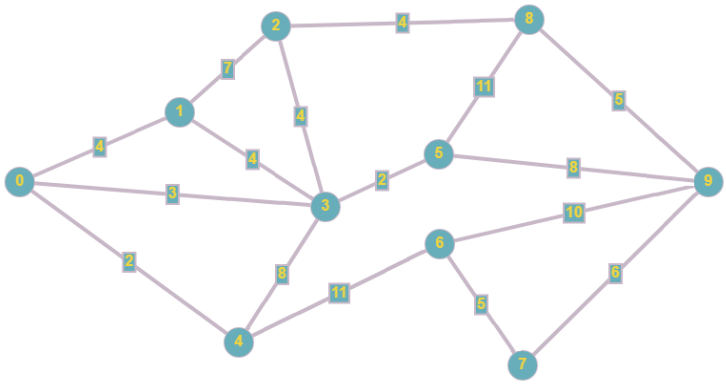

Рис. 2.1 – Исходный граф для реализации алгоритма Дейкстры

2.2. Вариант № 2

Выполнить разработку программы, реализующей поиск кратчайшего пути на графе при помощи алгоритма Белмана-Форда. Применить разработанную процедуру к графу на рисунке 2.2 для поиска кратчайшего пути от вершины 0 к вершине 9.

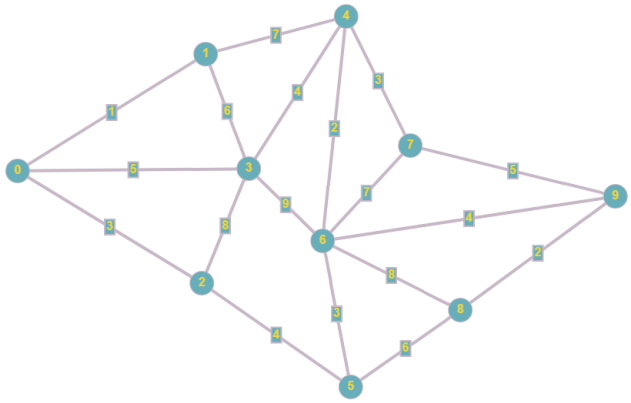

Рис. 2.2 – Исходный граф для реализации алгоритма Белмана-Форда

2.3. Вариант № 3

Выполнить разработку программы, реализующей поиск кратчайшего пути на графе при помощи алгоритма Белмана-Форда для разрешенного графа. Применить разработанную процедуру к графу на рисунке 2.3 для поиска кратчайшего пути от вершины 0 к вершине 13.

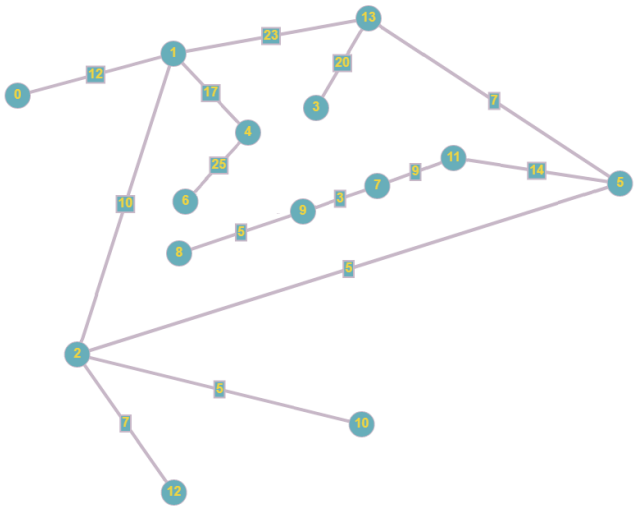

Рис. 2.3 – Исходный разрешенный граф для реализации алгоритма Белмана-Форда

3 КОНТРОЛЬНЫЕ ВОПРОСЫ

3.1. Выполнение каких базовых операций обеспечивает определение кратчайших путей на графе?

3.2. Каким образом организован последовательный алгоритм Дейкстры и каким образом он использует базовые операции?

3.3. В чем заключаются особенности алгоритма Дейкстры, допускающие его распараллеливание?

3.4. Каким образом организуется распараллеливание алгоритма Дейкстры?

3.5. Каким образом организован последовательный алгоритм Беллмана-Форда и каким образом он использует базовые операции?

3.6. В чем заключаются особенности алгоритма Беллмана-Форда, допускающие его распараллеливание?

3.7. Каким образом организуется распараллеливание алгоритма Беллмана-Форда?

ЛАБОРАТОРНАЯ РАБОТА № 9
ИССЛЕДОВАНИЕ АЛГОРИТМОВ ПОСТРОЕНИЯ ОСТОВНЫХ ДЕРЕВЬЕВ ГРАФА

ЦЕЛЬ РАБОТЫ: программно реализовать и исследовать эффективность алгоритмов построения остовных деревьев графа с использованием функций библиотеки MPI.

1. ТЕОРЕТИЧЕСКОЕ СВЕДЕНИЯ

1.1 Последовательный алгоритм Прима

Задан неориентированный граф G (V, R) с множеством вершин V = {1, 2, …, n} и множеством ребер R ($|R| = m$).

Остовный подграф графа G – связанный подграф $G_0 = (V, R_0)$ $R_0 \subseteq R$, то есть связный подграф содержащий все вершины графа G.

Остовное дерево в графе G – ациклический остовный подграф

Обозначим через $G_T(V_T, R_T)$ – часть сетевого дерева сформированную до текущей итерации алгоритма, тогда $V_T \subseteq V$ множество вершин включенных в остовное дерево, $R_T \subseteq R$ - множество дуг включенное в оставное дерево.

Через d_i $(1 \leq i \leq n)$ обозначим вес ребра минимальную длины соединяющего дугу из вершины $V_j \in V \backslash V_T$ с вершиной $V_j \in V_T$.

Таким образом $d_i = \min (W(V_j, V_i): V_j \in V_T$ и $V_i \in V \backslash V_T)$. Либо $\forall V_i \in V \backslash V_T \Rightarrow d_i = \min (W(V_j, V_i);\ V_j \in V_T$ и $V_i \in V \backslash V_T, (V_j, V_i) \in R)$ и более точно $(V_j, V_i) \in R \backslash R_T$.

Тогда на критерий итерации алгоритма определяется ребро (V_j, V_i) минимального веса d_i связывающий $V_j \in V_T$ и $V_i \in V \backslash V_T$.

Соответствующая вершина V_i добавляется в дерево V_T ребро добавляется в множество R_T:

$$V_T = V_T \cup \{V_i\}, R_T = R_T \cup \{(V_j, V_i)\}.$$

В случае, когда $|V_T| = n$ алгоритм завершается.

Для реализации алгоритма каждой вершине $V_i \in V \backslash V_T$ приписывается пара параметров(α_i, β_i), где α_i – номер ближайшей (к исходной) вершине $V_j \in V_T$ (то есть $\alpha_i = j$). $\beta_i = (V_j, V_i)$ – соответствующие вершине V_j, V_i ребро.

При реализации алгоритма параметры α_i, β_i изменяются таким образом, чтобы соответствовать минимальному остовному дереву.

Пример реализации алгоритма минимального остовного дерева.

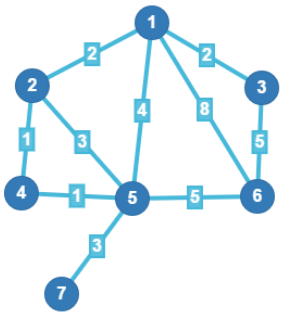

Рис. 1.1 — Исходный граф

Инициализация остовного дерева:

G_T=({v_1},∅)

Определение параметров α_i, β_i для соседних с v_1 вершин:

α_1=1, α_3=1, α_5=1, α_6=1;

β_3= 3, β_5=4, β_6=8

Итерация 1

Так как $\beta_2 = \beta_3$, тогда либо v_2 либо v_3 может быть добавлены в v_T:

G_T=({1, 2}, {(1, 2)})

Определение α_i, β_i для вершины v_4 (для v_5 параметры не пересчитываются, т.к. v_5 непосредственно к v_1 ближе, чем через вершину v_2). Имеем:

(α_3, β_3)=(1, 2), (α_4, β_4)=(2, 1), (α_5, β_5)=(1, 4), (α_6, β_6)=(7, 8)

Т.к. для вершины v_4 значение v_4 наименьшее, тогда вершина v_4 добавляется в V_T: G_T=({1, 2, 4}, {(1, 2),(2, 4)})

Т.к. (v_1, v_5) по стоимости равна $W_{V_1,V_2} + W_{V_2,V_4} + W_{V_4,V_5}$, тогда α_5 может быть модернизирована.

В итоге получаем: (α_3, β_3)=(1, 2), (α_5, β_5)=(4, 4), (α_6, β_6)=(1, 8) => вершина v_3 такая что W(v_1, v_3)=min(W(v_j, v_i), v_j∈ V_T, v_i∈ V/V_T).

Тогда V_T=({1, 2, 4, 3}, {(1, 2), (2, 4), (1, 3)})

Итоговый вид остовного дерева.

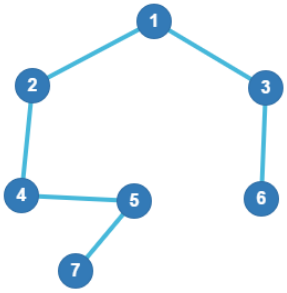

Рис. 1.2 — Результирующий граф

1.2 Последовательный алгоритм Прима

Алгоритм начинает работу с произвольной вершины графа, выбираемой в качестве корня дерева, и в ходе последовательно выполняемых итераций расширяет конструируемое дерево до МОД. Пусть V_T есть множество вершин, уже включенных алгоритмом в МОД, а величины d_i, 1<=i<=n, характеризуют дуги

минимальной длины от вершин, еще не включенных в дерево, до множества V_T, т.е.

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

(если для какой-либо вершины $i \notin V_T$ не существует ни одной дуги в V_T, значение d_i устанавливается равным ∞). В начале работы алгоритма выбирается корневая вершина МОД s и полагается V_T={s}, d_s=0.

Действия, выполняемые на каждой итерации алгоритма Прима, состоят в следующем:

a. определяются значения величин d_i для всех вершин, еще не включенных в состав МОД;

b. выбирается вершина t графа G, имеющая дугу минимального веса до множества $V_T t : d_t, i \notin V_T$;

- вершина t включается в V_T.

После выполнения n-1 итерации метода МОД будет сформировано. Вес этого дерева может быть получен при помощи выражения

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

Трудоемкость нахождения МОД характеризуется квадратичной зависимостью от числа вершин графа $T1 \sim n^2$.

1.3 Параллельный алгоритм Прима

Оценим возможности параллельного выполнения рассмотренного алгоритма нахождения минимально охватывающего дерева.

Итерации метода должны выполняться последовательно и, тем самым, не могут быть распараллелены. С другой стороны, выполняемые на каждой итерации алгоритма действия являются независимыми и могут реализовываться одновременно. Так, например, определение величин d_i может осуществляться для каждой вершины графа в отдельности, нахождение дуги минимального веса может быть реализовано по каскадной схеме и т.д.

Распределение данных между процессорами вычислительной системы должно обеспечивать независимость перечисленных операций алгоритма Прима. В частности, это может быть реализовано, если каждая вершина графа располагается на процессоре вместе со всей связанной с вершиной информацией. Соблюдение данного принципа приводит к тому, что при равномерной загрузке каждый процессор p_j, 1<=j<=p, должен содержать:

- набор вершин

$$V_j = \left\{v_{i_j+1}, v_{i_j+2}, \dots, v_{i_j+k}\right\}, i_j = k * (j-1), k = [n/p]$$

- соответствующий этому набору блок из k величин

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

- вертикальную полосу матрицы смежности графа G из k соседних столбцов

$$A_j = \{\alpha_{i_j+1}, \alpha_{i_j+2}, \dots, \alpha_{i_j+k}\} (\alpha_s \text{ есть } s - \text{й столбец матрицы A})$$

– общую часть набора V_j и формируемого в процессе вычислений множества вершин V_T.

Как итог можем заключить, что базовой подзадачей в параллельном алгоритме Прима может служить процедура вычисления блока значений Δ_j для вершин V_j матрицы смежности A графа G.

С учетом выбора базовых подзадач общая схема параллельного выполнения алгоритма Прима будет состоять в следующем:

– определяется вершина t графа G, имеющая дугу минимального веса до множества V_T. Для выбора такой вершины необходимо осуществить поиск минимума в наборах величин d_i, имеющихся на каждом из процессоров, и выполнить сборку полученных значений на одном из процессоров;

– номер выбранной вершины для включения в охватывающее дерево передается всем процессорам;

– обновляются наборы величин d_i с учетом добавления новой вершины.

Таким образом, в ходе параллельных вычислений между процессорами выполняются два типа информационных взаимодействий: сбор данных от всех процессоров на одном из процессоров и передача сообщений от одного процессора всем процессорам вычислительной системы.

По определению количество базовых подзадач всегда соответствует числу имеющихся процессоров, и, тем самым, проблема масштабирования для параллельного алгоритма не возникает.

Распределение подзадач между процессорами должно учитывать характер выполняемых в алгоритме Прима коммуникационных операций. Для оптимальной реализации требуемых информационных взаимодействий между базовыми подзадачами топология сети передачи данных должна обеспечивать эффективное представление в виде гиперкуба или полного графа.

Общий анализ сложности параллельного алгоритма Прима для нахождения минимального охватывающего дерева дает идеальные показатели эффективности параллельных вычислений:

$$S_p = \frac{n^2}{(n^2/p)} = p \text{ и } E_p = \frac{n^2}{p * (n^2/p)}$$

При этом следует отметить, что в ходе параллельных вычислений идеальная балансировка вычислительной нагрузки процессоров может быть нарушена. В зависимости от вида исходного графа G количество выбранных вершин в охватывающем дереве на разных процессорах может оказаться различным, и распределение вычислений между процессорами станет неравномерным (вплоть до отсутствия вычислительной нагрузки на отдельных процессорах). Однако такие предельные ситуации нарушения балансировки в общем случае возникают достаточно редко, а организация динамического перераспределения вычислительной нагрузки между процессорами в ходе вычислений является интересной, но одновременно и очень сложной задачей.