Министерство образования Республики Беларусь
Учреждение образования
«Брестский государственный технический университет»
Кафедра ИИТ

Лабораторная работа №2
По дисциплине « ***Алгоритмы и структуры данных*** »
Тема: ***«Нахождение кратчайшего пути в связном графе.»***

Выполнил:
Студент 2 курса
Группы ПО-11(2)
Сымоник И.А
Проверила:
Глущенко Т.А

Брест 2023

Цель работы: Изучить алгоритмы Дейкстры и Флойда-Уоршелла.

Ход работы
Вариант 6

№ №	*Кол. вер-шин*	*Кол. ре-бер*	*Ребра и веса*
6.	*7*	*11*	*{1,2},{1,3},{2,4},{2,5},{2,6}.{2,7}, {2,5},{3,6}{4,6},{5,6},{6,7}; [5,4,3,6,6,8,5,7,4,4,3];*

Задание 1. Алгоритмом *Дейкстры* вычислить кратчайшие пути от вершины x_1 ко всем остальным вершинам графа, указав их длины и сам путь для каждой пары вершин (последовательность вершин). Для хранения длин кратчайших путей рекомендуется использовать бинарную кучу *(min-heap)*.

Исходный код:

```
std::list<std::pair<int, int>>* CreateMatrix(const std::vector<std::pair<int,
int>>& edges, const std::vector<int>& weights, int countOfVert)
{

	std::list<std::pair<int, int>> *edg = new std::list<std::pair<int,
int>>[countOfVert];

	int j = 0;
	for (auto& i : edges)
	{
		edg[i.first].push_back(std::make_pair(i.second, weights[j]));
		edg[i.second].push_back(std::make_pair(i.first, weights[j]));

		j++;
	}

	return edg;
}



void Dijkstra(std::vector<std::pair<int, int>>& edges, const std::vector<int>&
weights, const int firstVertex, const size_t countOfVertex)
{


	std::transform(std::begin(edges), std::end(edges), std::begin(edges),
[](std::pair<int, int> x) {return std::make_pair<int, int>(x.first - 1, x.second
- 1); });


	auto mat = CreateMatrix(edges, weights, countOfVertex);
```

```
	std::priority_queue<std::pair<int, int>, std::vector<std::pair<int, int>>,
std::greater<std::pair<int, int>>> pQueue;
	std::vector<std::pair<int, int>> dist;

	for (int i = 0; i < countOfVertex; i++)
	{
		dist.push_back(std::make_pair(INT_MAX, i));
	}

	pQueue.push(std::make_pair(firstVertex, 0));
	dist[firstVertex] = std::make_pair(firstVertex, 0);

	while (!pQueue.empty())
	{

		int u = pQueue.top().second;
		pQueue.pop();

		std::list<std::pair<int, int>>::iterator i;
		for (i = mat[u].begin(); i != mat[u].end(); ++i)
		{


			int v = (*i).first;
			int weight = (*i).second;

			if (dist[v].first > dist[u].first + weight)
			{
				dist[v].first = dist[u].first + weight;
				dist[v].second = u;

				pQueue.push(std::make_pair(dist[v].first, v));
			}
		}
	}


	for (size_t i = 0; i < dist.size(); i++)
	{
		std::cout << "Расстояние от узла " << firstVertex + 1 << " до узла "
<< i + 1 << " равно " << dist[i].first << std::endl;

		int currnode = i;
		std::cout << "Путь " << currnode + 1 ;
		while (currnode != firstVertex)
		{
			currnode = dist[currnode].second;
			std::cout << " <- " << currnode + 1;
		}
		std::cout << std::endl << std::endl;
	}

	delete[] mat;

}


int main()
{
	SetConsoleCP(1251);
	SetConsoleOutputCP(1251);
```

```
	std::vector<std::pair<int, int>> edges = { {1,2}, {1,3}, {2,4}, {2,5},
{2,6}, {2, 7},
		{ 2,5 }, { 3,6 }, {4, 6}, { 5,6 }, { 6,7 } };

	std::vector<int> weight = { 5,4,3,6,6,8,5,7,4,4,3 };

	Dijkstra(edges, weight, 0, 7);
}
```

Вывод программы:

Расстояние от узла 1 до узла 1 равно 0
Путь 1

Расстояние от узла 1 до узла 2 равно 5
Путь 2 <- 1

Расстояние от узла 1 до узла 3 равно 4
Путь 3 <- 1

Расстояние от узла 1 до узла 4 равно 8
Путь 4 <- 2 <- 1

Расстояние от узла 1 до узла 5 равно 10
Путь 5 <- 2 <- 1

Расстояние от узла 1 до узла 6 равно 11
Путь 6 <- 3 <- 1

Расстояние от узла 1 до узла 7 равно 13
Путь 7 <- 2 <- 1

Задание 2. Алгоритмом *Флойда-Уоршелла* вычислить кратчайшие пути между всеми парами вершин взвешенного графа, указав их длины и пути.

Исходный код:

```
void FloydWarshall(std::vector<std::vector<int>>& matrix)
{
	std::vector<std::vector<int>> dist;

	int n = matrix.size();
	for (int i = 0; i < n; i++)
	{
		std::vector<int> row;
		dist.push_back(row);
		for (int j = 0; j < n; j++)
		{
			dist[i].push_back(matrix[i][j]);
```

```
			}
		}


		for (size_t k = 0; k < matrix.size(); k++)
		{
			for (size_t i = 0; i < matrix.size(); i++)
			{
				for (size_t j = 0; j < matrix.size(); j++)
				{

					if (matrix[i][j] > matrix[i][k] + matrix[k][j])
						matrix[i][j] = matrix[i][k] + matrix[k][j];
				}
			}
		}
}


void PrintShortestPath(std::vector<std::vector<int>>& dist)
{
		for (size_t i = 0; i < dist.size(); i++)
		{
			for (size_t j = 0; j < dist[i].size(); j++)
			{
				std::cout << "Расстояние от узла " << i+1 << " до узла " <<
j+1 << " равно: " << dist[i][j] << std::endl;
			}
			std::cout << std::endl;
		}
}


std::vector<std::vector<int>> FormAdjMatrix(const std::vector<std::pair<int,
int>>& edges, const std::vector<int>& weigths, const int countOfVert)
{
		std::vector<std::vector<int>> adjMatrix;
		adjMatrix.resize(countOfVert);
		int j = 0;
		for (size_t i = 0; i < adjMatrix.size(); i++)
		{
			adjMatrix[i].resize(countOfVert);
			std::fill(adjMatrix[i].begin(), adjMatrix[i].end(), 1000000007);
			adjMatrix[i][j] = 0;
			j++;
		}

		for (size_t i = 0; i < edges.size(); i++)
		{
			adjMatrix[edges[i].first][edges[i].second] = weigths[i];
			adjMatrix[edges[i].second][edges[i].first] = weigths[i];
		}

		return adjMatrix;
}

int main()
{
		SetConsoleCP(1251);
		SetConsoleOutputCP(1251);

		std::vector<std::pair<int, int>> edges = { {1,2}, {1,3}, {2,4}, {2,5},
{2,6}, {2, 7},
```

```
                { 2,5 }, { 3,6 }, {4, 6}, { 5,6 }, { 6,7 } };

        std::vector<int> weight = { 5,4,3,6,6,8,5,7,4,4,3 };

        std::transform(std::begin(edges), std::end(edges), std::begin(edges),
[](std::pair<int, int> x) {return std::make_pair<int, int>(x.first - 1, x.second
- 1); });
        auto mat = FormAdjMatrix(edges, weight, 7);
        FloydWarshall(mat);
        PrintShortestPath(mat);
}
```

Вывод программы:

Расстояние от узла 1 до узла 1 равно: 0
Расстояние от узла 1 до узла 2 равно: 5
Расстояние от узла 1 до узла 3 равно: 4
Расстояние от узла 1 до узла 4 равно: 8
Расстояние от узла 1 до узла 5 равно: 10
Расстояние от узла 1 до узла 6 равно: 11
Расстояние от узла 1 до узла 7 равно: 13

Расстояние от узла 2 до узла 1 равно: 5
Расстояние от узла 2 до узла 2 равно: 0
Расстояние от узла 2 до узла 3 равно: 9
Расстояние от узла 2 до узла 4 равно: 3
Расстояние от узла 2 до узла 5 равно: 5
Расстояние от узла 2 до узла 6 равно: 6
Расстояние от узла 2 до узла 7 равно: 8

Расстояние от узла 3 до узла 1 равно: 4
Расстояние от узла 3 до узла 2 равно: 9
Расстояние от узла 3 до узла 3 равно: 0
Расстояние от узла 3 до узла 4 равно: 11
Расстояние от узла 3 до узла 5 равно: 11
Расстояние от узла 3 до узла 6 равно: 7
Расстояние от узла 3 до узла 7 равно: 10

Расстояние от узла 4 до узла 1 равно: 8
Расстояние от узла 4 до узла 2 равно: 3
Расстояние от узла 4 до узла 3 равно: 11
Расстояние от узла 4 до узла 4 равно: 0
Расстояние от узла 4 до узла 5 равно: 8
Расстояние от узла 4 до узла 6 равно: 4
Расстояние от узла 4 до узла 7 равно: 7

Расстояние от узла 5 до узла 1 равно: 10
Расстояние от узла 5 до узла 2 равно: 5
Расстояние от узла 5 до узла 3 равно: 11
Расстояние от узла 5 до узла 4 равно: 8
Расстояние от узла 5 до узла 5 равно: 0
Расстояние от узла 5 до узла 6 равно: 4
Расстояние от узла 5 до узла 7 равно: 7

Расстояние от узла 6 до узла 1 равно: 11
Расстояние от узла 6 до узла 2 равно: 6
Расстояние от узла 6 до узла 3 равно: 7
Расстояние от узла 6 до узла 4 равно: 4
Расстояние от узла 6 до узла 5 равно: 4
Расстояние от узла 6 до узла 6 равно: 0
Расстояние от узла 6 до узла 7 равно: 3

Расстояние от узла 7 до узла 1 равно: 13
Расстояние от узла 7 до узла 2 равно: 8
Расстояние от узла 7 до узла 3 равно: 10
Расстояние от узла 7 до узла 4 равно: 7
Расстояние от узла 7 до узла 5 равно: 7
Расстояние от узла 7 до узла 6 равно: 3
Расстояние от узла 7 до узла 7 равно: 0

Задание 3. Вручную указать *3 итерации* прохождения алгоритмов (построить матрицы).

Для алгоритма Дейкстры

Итерация	S	w	Вершина 2	Вершина 3	Вершина 4	Вершина 5	Вершина 6	Вершина 7
Начало	{1}	-	5	4	∞	∞	∞	∞
1	{1,3}	3	5		∞	∞	7	∞
2	{1,3,2}	2			8	10	11	13
3	{1,3,2,4}	4				10	12	13

Для алгоритма Флойда-Уоршелла

Первая итерация:

	1	2	3	4	5	6	7
1	0	5	4	∞	∞	∞	∞
2	5	0	∞	3	5	6	8
3	4	∞	0	∞	∞	7	∞
4	∞	3	∞	0	∞	4	∞
5	∞	5	∞	∞	0	4	∞
6	∞	6	7	4	4	0	3
7	∞	8	∞	∞	∞	3	0

Первая итерация

	1	2	3	4	5	6	7
1	**0**	**5**	**4**	**∞**	**∞**	**∞**	**∞**
2	**5**	0	9	3	5	6	8
3	**4**	9	0	∞	∞	7	∞
4	**∞**	3	∞	0	∞	4	∞
5	**∞**	5	∞	∞	0	4	∞
6	**∞**	6	7	4	4	0	3
7	**∞**	8	∞	∞	∞	3	0

Вторая итерация:

	1	**2**	3	4	5	6	7
1	0	**5**	4	8	10	11	13
2	**5**	**0**	**9**	**3**	**5**	**6**	**8**
3	4	**9**	0	12	14	15	17
4	8	**3**	12	0	8	4	11

5	10	**5**	14	8	0	4	13
6	11	**6**	7	4	4	0	3
7	13	**8**	17	11	13	3	0

Задание 4. Решить задачу *238 Product of Array Except Self* на ресурсе *LeetCode*.

Профиль: https://leetcode.com/DOXECEES/

```
class Solution {
public:
    vector<int> productExceptSelf(vector<int>& nums) {


        std::vector<int> ans;
        ans.push_back(1);

        int temp = 1;
        int temp2;
        for(int i = 1; i < nums.size(); ++i)
        {
            temp2 = nums[i - 1] * temp;
            ans.push_back(temp2);
            temp = std::move(temp2);
        }

        temp = 1;
        for(int i = nums.size() - 1; i > 0; i--)
        {
            temp2 = nums[i] * temp;
            ans[i-1] = ans[i-1] * temp2;
            temp = std::move(temp2);
        }
        return ans;
    }
};
```

Runtime
18ms
Beats 63.59%of users with C++

Memory
25.41MB

Beats 19.51%of users with C++

Вопросы к лабораторной работе:

1.Что такое *«жадный»* алгоритм и какой из указанных алгоритмов является *«жадным»*? Указать *«О большое»* для обоих алгоритмов.

Жадные алгоритмы — это алгоритмы, которые, на каждом шагу принимают локально оптимальное решение, не заботясь о том, что будет дальше.

Алгоритм Дейкстры является жадным, потому что он всегда отмечает ближайшую вершину графа.

Сложность алгоритма Дейкстры – $O(n^2)$, если вершины хранятся в простом массиве или O(m*logn), если же используется двоичная куча.

Алгоритм Уоршелла-Флойда – $O(n^3)$

2. Почему классический алгоритм *Дейкстры* не работает для графов с отрицательными весами?

Алгоритм Дейкстры не работает с отрицательными весами, потому что он основан на принципе выбора кратчайшего пути на основе накопленной стоимости. Когда вес ребра отрицательный, возникает проблема, называемая "циклом отрицательного веса". В таких случаях алгоритм Дейкстры может зациклиться, так как он будет постоянно обновлять стоимость пути, стремясь найти еще более короткий путь. Это делает алгоритм неприменимым для графов с отрицательными весами.

3. Как влияет на эффективность алгоритма *Дейкстры* использование *бинарной кучи* (кучи *Фибоначчи*, *2-4* кучи и т. д.)

Используя двоичную кучу можно выполнять операции извлечения минимума и обновления элемента за O(logn) Тогда время работы алгоритма Дейкстры составит O(m * logn)

Используя Фибоначчиевы кучи можно выполнять операции извлечения минимума за O(logn) и обновления элемента за O(1)
Таким образом, время работы алгоритма составит O(n*logn + m)

4. Какой из алгоритмов построен на принципе *динамического программирования* и что это за принцип?

Динамическое программирование — это подход к решению задач, который основывается на том, что исходная задача разбивается на более мелкие подзадачи, которые проще решить. На принципе динамического программирования построен алгоритм Уоршелла-Флойда.

5. Почему алгоритм *Флойда-Уоршелла* не работает для графов с *отрицательными циклами*? Можно ли использовать алгоритм для проверки наличия *отрицательных циклов* в графе.

Если в графе существует отрицательный цикл, то алгоритм Флойда-Уоршелла может зациклиться и продолжать обновлять значения расстояний бесконечное количество раз. Это связано с тем, что отрицательный цикл позволяет бесконечно уменьшать значения расстояний на каждой итерации алгоритма. В результате, алгоритм не сможет завершить работу и не сможет корректно определить кратчайшие пути.

Да, можно. Если после окончания работы алгоритма существует вершина i такая, что dp[i][i]<0, то она входит в отрицательный цикл.

6. Какие алгоритмы нахождения кратчайших путей вы еще знаете?

Алгоритм поиска A* находит маршрут с наименьшей стоимостью от одной вершины (начальной) к другой (целевой, конечной), используя алгоритм поиска по первому наилучшему совпадению на графе.

Алгоритм Флойда — Уоршелла находит кратчайшие пути между всеми вершинами взвешенного ориентированного графа[6].

Алгоритм Джонсона находит кратчайшие пути между всеми парами вершин взвешенного ориентированного графа.

Алгоритм Ли (волновой алгоритм) основан на методе поиска в ширину.

Вывод: изучили алгоритмы Дейкстры и Флойда-Уоршелла для поиска кратчайшего пути.