**ИТМО Кафедра Информатики и прикладной математики**

Отчет по лабораторной работе №2

«Поиск минимальных путей в графе»   
Вариант 7

**Выполнил: студент группы P3217**

**Плюхин Дмитрий**

**Преподаватель: Зинчик А. А.**

**2017 год**

1. **Задание**

1. Написать программу, реализующую алгоритм Декстры на основе d-кучи (d=2) и алгоритм Дейкстры, использующий метки.

2. Написать программу, реализующую алгоритмы, для проведения экспериментов, в которых можно выбирать:

* число n вершин и число m ребер графа,
* натуральные числа q и r, являющиеся соответственно нижней и верхней границей для весов ребер графа.

Выходом данной программы должно быть время работы ТА алгоритма Декстры на основе d-кучи (d=2) и время работы ТВ алгоритма алгоритм Дейкстры, использующего метки, в секундах.

3. Провести эксперименты на основе следующих данных:

* n = 104+1
* m = 0, … ,107 с шагом 105
* q = 1
* r = 106
* нарисовать графики функций TА(m) и ТВ(m);

4. Сформулировать и обосновать вывод о том, в каких случаях целесообразно применять алгоритм А, а в каких алгоритм В.

1. **Исходный код**

int findShortestWays(DWORD numberOfVertexes, DWORD queueWidth, DWORD startVertex, GraphAdjRecordPointer graphAdj, //asymptotically O((n+m)logn)

LPDWORD distances, LPDWORD vertexes, LPDWORD indexes, PriorityQueueRecordPointer priorityQueue){

for (int i = 0; i < numberOfVertexes; i++){

distances[i] = 0xffff;

vertexes[i] = numberOfVertexes;

indexes[i] = i; // vertex with name i at the index i (on it's place)

priorityQueue[i].name = i;

priorityQueue[i].key = 0xffff;

}

priorityQueue[startVertex].key = 0; // current evaluation of shortest path to the start vertex gives 0 length (because it's beginning of a route)

DWORD numberOfUncheckedVertexes = numberOfVertexes; // no one vertex checked at the moment

makePriorityQueue(numberOfVertexes, queueWidth, priorityQueue, indexes); // move start vertex to the top (after that vertex with name s has index 0)

PriorityQueueRecord minRecord; // here we will extract vertex with shortest path to that

DWORD nameOfMinimalVertex; // here we will keep the name of the vertex with shortest path to that

GraphAdjRecordPointer linkedVertexAdjRecord; // temporary variable for keeping vertex linked with minimal

DWORD nameOfLinkedVertex; // temporary variable for keeping linked vertex's name

DWORD indexOfLinkedVertexInPriorityQueue; // index in priority queue and name are different (for example, vertex s now has index 0, so index[s]=0)

while (numberOfUncheckedVertexes > 0){ // while there are unchecked vertexes (gives O(n))

//showArr(indexes, numberOfVertexes);

//showPriorityQueue(priorityQueue,numberOfUncheckedVertexes,queueWidth);

minRecord = takeMininum(numberOfUncheckedVertexes, queueWidth, priorityQueue, indexes); // extract vertex with minimal path to that from the s (gives O(dlogn))

//showArr(indexes, numberOfVertexes);

//showPriorityQueue(priorityQueue,numberOfUncheckedVertexes,queueWidth);

numberOfUncheckedVertexes--;

nameOfMinimalVertex = minRecord.name; // save name of minimal vertex in a temporary variable

//cout << "Select " << nameOfMinimalVertex << " and " << minRecord.key << endl;

distances[nameOfMinimalVertex] = minRecord.key; // update shortest path to minimal vertex in result array

linkedVertexAdjRecord = &graphAdj[nameOfMinimalVertex];// take first linked vertex in list of vertex with minimal path to that from the s

if (linkedVertexAdjRecord->weight == 0) continue;

while (linkedVertexAdjRecord != NULL){ // while there are linked vertexes (gives O(m/n))

nameOfLinkedVertex = linkedVertexAdjRecord->name; // save name of linked vertex

//cout << "Watch " << nameOfLinkedVertex << endl;

indexOfLinkedVertexInPriorityQueue = indexes[nameOfLinkedVertex]; // save index of linked vertex in priority queue

//cout << indexOfLinkedVertexInPriorityQueue << endl;

//showPriorityQueue(priorityQueue,numberOfVertexes,queueWidth);

if (priorityQueue[indexes[nameOfLinkedVertex]].key > distances[nameOfMinimalVertex] + linkedVertexAdjRecord->weight){

// if the known minimal length of path to the linked vertex more than one which we can make if will go through current minimal vertex

priorityQueue[indexOfLinkedVertexInPriorityQueue].key = distances[nameOfMinimalVertex] + linkedVertexAdjRecord->weight; // decrease length of path

moveUp(numberOfUncheckedVertexes, queueWidth, indexOfLinkedVertexInPriorityQueue, priorityQueue, indexes); // and move it upper if possibly (gives O(logn))

vertexes[nameOfLinkedVertex] = nameOfMinimalVertex;

}

linkedVertexAdjRecord = linkedVertexAdjRecord->next; // take next linked vertex

}

}

return 0;

}

int findShortestWaysByMarks(DWORD numberOfVertexes, DWORD queueWidth, DWORD startVertex, GraphAdjRecordPointer graphAdj, LPDWORD distances, LPDWORD vertexes, DWORD graphDensity){

LPDWORD concludings = (LPDWORD)malloc(numberOfVertexes\*sizeof(DWORD)); //concludings[i] shows is shortest path builded for the vertex i

for (int i = 0; i < numberOfVertexes; i++){

distances[i] = 0xffffffff;

vertexes[i] = numberOfVertexes;

concludings[i] = 0;

}

distances[startVertex] = 0;

DWORD numberOfUncheckedVertexes = numberOfVertexes; // no one vertex checked at the moment

DWORD nameOfFirstUncheckedVertex;

DWORD nameOfOneUncheckedVertex;

DWORD nameOfAnotherUncheckedVertex;

DWORD nameOfLinkedVertex;

GraphAdjRecordPointer linkedVertexAdjRecord;

while (numberOfUncheckedVertexes > 0){ // while there are unchecked vertexes (gives O(n))

nameOfFirstUncheckedVertex = 0;

while (concludings[nameOfFirstUncheckedVertex] != 0) nameOfFirstUncheckedVertex++;

nameOfOneUncheckedVertex = nameOfFirstUncheckedVertex;

for (DWORD nameOfAnotherUncheckedVertex = nameOfOneUncheckedVertex + 1; nameOfAnotherUncheckedVertex < numberOfVertexes; nameOfAnotherUncheckedVertex++){

if (concludings[nameOfAnotherUncheckedVertex] == 0){

if (distances[nameOfOneUncheckedVertex] > distances[nameOfAnotherUncheckedVertex]){

nameOfOneUncheckedVertex = nameOfAnotherUncheckedVertex;

}

}

}

//after O(n) got name of unchecked vertex having minimal distance

concludings[nameOfOneUncheckedVertex] = 1;

numberOfUncheckedVertexes--;

linkedVertexAdjRecord = &graphAdj[nameOfOneUncheckedVertex];// take first linked vertex in list of vertex with minimal path to that from the s

if (linkedVertexAdjRecord->weight == 0) continue;

while (linkedVertexAdjRecord != NULL){

nameOfLinkedVertex = linkedVertexAdjRecord->name;

if (concludings[nameOfLinkedVertex] == 0){

if (distances[nameOfLinkedVertex] > distances[nameOfOneUncheckedVertex] + linkedVertexAdjRecord->weight){

distances[nameOfLinkedVertex] = distances[nameOfOneUncheckedVertex] + linkedVertexAdjRecord->weight;

vertexes[nameOfLinkedVertex] = nameOfOneUncheckedVertex;

}

}

linkedVertexAdjRecord = linkedVertexAdjRecord->next;

}

}

return 0;

}

1. **Результаты экспериментов**
2. **Вывод**

Итак, касательно рассмотренных алгоритмов поиска кратчайших путей в графе можно сделать вывод, что алгоритм Дейкстры на основе d-кучи является предпочтительным в случае больших объемов данных, а также в тех случаях, когда является предпочтительным выделение большего количества памяти ради улучшения быстродействия. Алгоритм Дейкстры, использующий метки, в отличие от предыдущего варианта, требует меньше памяти для своей работы (не нуждается в структуре для размещения очереди и в массиве для хранения индексов), однако работает медленнее алгоритма, использующего кучу, поэтому может быть использован в условиях ограниченного количества доступной памяти.