浙江大學

Shortest Path with Heaps

Advanced Data Structures and Algorithm Analysis
Research Project 2

April 8, 2018

Contents

Chapter 1

Introduction

Our group aimed at implementing the shortest path algorithm by using different heaps, such as Binomial heap, fibonacci heap, and then comparing their run times to find the best structure for Dijkstra's algorithm.

1.1 Purpose

The report contains the idea of our group about how to implement the Dijkstra's algorithm by using differnt data structures, such as fibonacci heap and binomial heap. The details about how we implement the data structures would also be illustrated in this report. To help you get a better understanding of our design of the Dijkstra's algorithm, we not only display the source code in the appendix but also the pseudo-code in the second part. To illustrate how we get to the conclusion that some specific data structrue is the best for implementing Dijkstra's algorithm, we display our test results based on the USA road networks for evalution data sets.

1.2 Background

The main algorithm in our project to compute the shortest paths is the Dijkstra's algorithm. The Dijkstra (Dijkstra) algorithm is a typical algorithm used to solve the shortest path. It is also an example in many tutorials, proposed by computer scientist Dijkstra of Holland in 1959 to get the shortest path from the starting point to all other points. The greedy algorithm is used to find the nearest point from each point, and because it can not be used to solve the graph with negative edges. Most of the problems are solved in this way: there is an undirected graph G (V, E), and the weight of the edge E[i] is W[i], finding the shortest path from V[0] to V[i].

We get to know that a binomial heap is a heap similar to a binary heap but also supports quick merging of two heaps. This is achieved by using a special tree structure. It is important as an implementation of the mergeable heap abstract data type (also called meldable heap), which is a priority queue supporting merge operation. In wikipedia, we found that a set of binomial trees is how the binomial heap is implemented, it could be descripted recursively as follows:A binomial tree of order 0 is a single node A binomial tree of order k has a root node whose

children are roots of binomial trees of orders k−1, k−2, ..., 2, 1, 0 (in this order). A binomial tree of order k has 2k nodes, height k.

The Fibonacci pile is a loose two heap. The main difference between the two heap is that the Fibonacci tree is not two trees, and the root arrangement of these trees is disordered. (the root nodes of the two heaps are sorted from the left to the right by the number of nodes, not according to the size of the root nodes).

In this project we use above two heap data structure to implement the Dijkstra's algorithm and test the run times based on the USA road networks for evaluation data sets.

Chapter 2

Algorithm Specification

2.1 Fibonacci Heap

2.1.1 Definition

Fibonacci heap is assumbly of Minimum heap in order which is similar as binary heap but have better time complexity on average time.

1. All the subtree is the minimum heap, it should follow the rule of minimum heap.
2. Only entrance is the minimum node.
3. each node is connected by double-way chain. Like left and right, or parent and child.

Fibonacci-heap property

A Fibonacci heap is a collection of trees satisfying the minimum-heap property, that is, the key of a child is always greater than or equal to the key of parent. [1]

2.1.2 Operations

Searching

To find the specific node just do taversal of searching.

Algorithm 1 Search node

```
1: function SEARCH-FIBONACCIHEAP(number, *heap)
2:     if (number<heap->min)
3:     return
4:     traversal to find each tree node if (number == node->number)
5:     if(found) return *node
6:     else return null
7: end function
```

[1] Wikipedia, "Fibonacci_heap", https://en.wikipedia.org/wiki/Fibonacci_heap

Insertion

To insert a key k in the Fibonacci heap, join the node as bidirectional chain and judge whether it smaller than small one, if so change the min pointer of heap:

Delete Minimum

To delete a node z, is the most complex operation in the Fibonacci heap:

Algorithm 2 Delete Minimum Node

```
1: function DELTE_MIN(*heap)
2:     if(heap->min have child)
3:     move heap->min->child to the root chain
4:     else deleteheap->min
5:     heap->min = find another Minimum node number on the root chain
   MERGE(*heap)
6: end function
```

When performing Merge() inside the DELETE_MIN function it combine the same degree of the subtree in the Fibonacci heap.

2.1.3 Pop

2.1.4 Merge

Each subtree save the message of degree, traversal every node to combine the same degree of subtree in the Fibonacci heap.

Algorithm 3 Merge

```
1: function MERGE(*heap)
2:     node = heap->min
3:     while(node)
4:     Insert(node->degree,hash_table[ ])
5:     if(hash_table has that degree already)
6:     newtree = binary-heap-merge(*subtree1,*subtree2)
7:     Insert(newtree, hash_table[ ])
8:     else node = node->R
9:     return *heap
10: end function
```

2.1.5 Decrese Key

This is the main advantage function in Fibonacci heap in this research program, which decrease the searching time complexity for Fibonacci heap.

Algorithm 4 Node Number Decrease

```
1: function NODE-NUMBER-DECREASE(*heap, number)
2:     node = search(number)
3:     decrease-number(node)
4:     mark(node->parent)
5:     while(node->parent already mark)
6:     move(node,main-chain)
7:     move(nodetree,main-chain)
8:     return *heap
9: end function
```

2.2 Binomial Queue

2.2.1 Definition

A binomial queue is not a heap-ordered tree, but rather a collection of heap-ordered trees, known as a forest. Each heap-ordered tree is a binomial tree. [2]

2.2.2 Operations

Search for Minimum

If a queue T is an Binominal queue, a searching minimun operation of T will be go through all the rootnode to find the minimum number of node.

Insertion

When inserting an element into a binominal queue, we add it like a binary calculation. The root is combination of the array of pointer which represent different degree of the binomial tree.

Algorithm 5 Insertion

```
1: function INSERTION(*queue, node)
2:     new queue =merge(node,last*queue)
3:     check(whether degree of new queue in the *queue)
4:     if(yes)
5:     merge(newqueue,same degree queue)
6:     check again
7:     else
8:     return *queue
9: end function
```

Minimum Deletion

The minimum deletion in Binomial queue is the process of divide the tree into many sub part of degree and merge all of it.

[2] M.A.Weiss 著、陈越改编，人民邮件出版社，2005,Data Structure and Algorithm Analysis in C (2nd Edition)

```
1: function MINIMUN DELETION(*queue)
2:     h = findMinNode()
3:     *newqueue = seperate(h)
4:     deleteMin(*queue)
5:     *queue = merge(*queue,*newqueue)
6:     return *queue
7: end function
```

2.2.3 Merge

Just judge which same degree of tree who has the smaller root than add another tree to her root as another child.

2.3 Dijkstra

2.3.1 Definition

Dijkstra's algorithm is a typical single source shortest path algorithm, which is used to calculate the shortest path of a node to all other nodes. It is an algorithm used among nodes in the shortest path graph, which could represent the road network. It was conceived by computer scientist Edsger w. Dijkstra in 1956. There are many variations of this algorithm. The original variant Dijkstra find the shortest path between two nodes, but more common variants as "source" to a single node, to find the shortest path from the source to all other nodes of the shortest path in the graph, producing a shortest-path tree. For a given source node, the algorithm finds the shortest path between the nodes. It can also be used for the shortest path from a node to a destination node. Once the shortest path of the destination node is determined, the algorithm is stopped.

2.3.2 algorithm properity

The main feature of the Dijkstra's algorithm is to expand from the starting point to the center, extending to the end until the end. The Dijkstra algorithm is a very representative shortest path algorithm, which is introduced as the basic content in many professional courses, such as data structure, graph theory, operational research and so on. When the graph contains negative weight edge, the dijkstra's algorithm is not appropriate any more.

2.3.3 Description

Set Graph = (Vertex, Edge) is a weighted directed graph, which divides the set of vertices of the graph into two groups, the first group is the set of vertices of the shortest path, which is expressed by S. At the beginning, there is only one source in S, and then every one of the shortest path will be added to the set S until all the vertices are added to S, the algorithm is The second set is the set of vertices of the remaining undetermined shortest paths (expressed in U), and the second groups of vertices are added to the S in order by the increasing order of the shortest path length. During the process of joining, the shortest path length of

every vertex from source point v to S is always less than the shortest path length from any source point v to U. In addition, each vertex corresponds to a distance, the distance from the vertex in the S is the shortest path length from the V to the vertex, and the distance from the vertex in the U is the shortest path length of the vertex from the V to the vertex of the S only.

2.3.4 Operations

At the beginning, the set S contains only the source point, namely S = v, and the distance from V is zero. U contains other vertices except V, if V has the edge of the vertex u in U, then <u, v> has a normal weight, if u is not a neighbour of a V, <u, v> weight is infinity.

Choose a vertex K which is the smallest distance from V from U, and adds K to S (the shortest distance between V and K).

Modifies the distance of each vertex in the U with the K as a new intermediate point. If the distance from the source point v to the vertex u is shorter than the original distance (without the vertex K), the distance value of the vertex u is modified, and the distance from the vertex K of the modified distance value is added to the upper edge.

Repeats steps "choose a vertex" and "modify the distance" until all vertices are included in S.

Function Implementation The details of our program to implement the Dijkstra's algorithm would be illustrated as follows.

Insert Insert an item into the queue. MergeTree Merge the tree.

MergeHeap Merge the heap.

Pop_Fix Pop the item (call function pop() to simply pop the item) and fix the distance and update the current shortest path. pop Pop the item.

cut Cut down the subtree.

de_fix Decrease the key of the item (call function() to simply decrease the key) and fix the structure and update the current shortest path.

de_key Decrease the key.

del Delete the specific vertex.

addpath Add the specific edge.

Chapter 3

Testing Results

3.1 Fibonacci Heap

3.1.1 Case 1

average(s)	Max(s)	Min(s)
18.9096	23.137	17.218

Figure 3.1: testing result of case 1

3.2 Binomial Queue

3.2.1 Case 1

average(s)	Max(s)	Min(s)
43.458	48.156	39.874

Figure 3.2: testing result of case 1

Chapter 4

Analysis and Comments

General speaking, in dijkstra algorithm and heap, heap have following functions.

1. Insert elements
2. Get heap head elements
3. Pop elements
4. Fix elements

For binomial heap: Time complexity

1. Insert :O(1)
2. Pop :O(logn)
3. Pop :O(1)
4. Fix :O(logn)

For binomial queue: Time complexity

1. Insert :O(1)
2. Pop :O(logn)
3. Pop :O(1)
4. can't fix

For binomial queue: Time complexity

1. Insert :O(1)
2. Pop :O(logn)
3. Pop :O(1)
4. fix :O(1)

In therom , the question of dijkstra is through relax the distanse so the element is frequently being fix in the structure.
For binomial queue, even has advantage on insert operation butthe maintain to fix the structure has bad time complexity.(In some unofficial test,binomial queue is not efficient than skew heap and mergable heap by have alo deletion operation). So in this problem there is no advantage for binomial queue, even if we want to fix the element in the heap, we need to add the same element in many times and use a mark array to prevent the logit fault.
For fibonacci heap: in theorom, not only has great efficient also have better time complexity. But the complexity of the structure himself make him not as good as we see. But in this question, it has really good advantage.

Because of Dijkstra is the algorithm of sigle root to find the minimum road. We unable to save N2 scale of distance array, so we redo Dijkstra every time in different data. Our original plan is to test binomial heap/skew heap/leftest heap/binomail queue/Fibonacci heap. But as the data scale is too bit, it nees 5 hours to go through one algorithm to finish the test. So we only do the complete test on Fibonacci heap and binomial queue.

Appendices

Appendix A

Author List, Declaration and Signatures

Author List

Declaration

We hereby declare that all the work done in this project titled "Binary Search Trees" is of our independent effort as a group.