Design and Practice of Software

Name:Yang Yi

Student Number:1130320118

Introduction

This is my report for the lesson “Design and Practice of Software” ’s homework.

All programs use c programming language. As well as they use file input and output , but sometimes the program only output a number , so I could not be bothered to write a file output function….I just print it to screen:P

All programs had never been optimized . I know how to do that , but I just busy for other course :(

Because it is too long after I complete these program, so when I explain these program to Mrs. Liu , I fogot something.

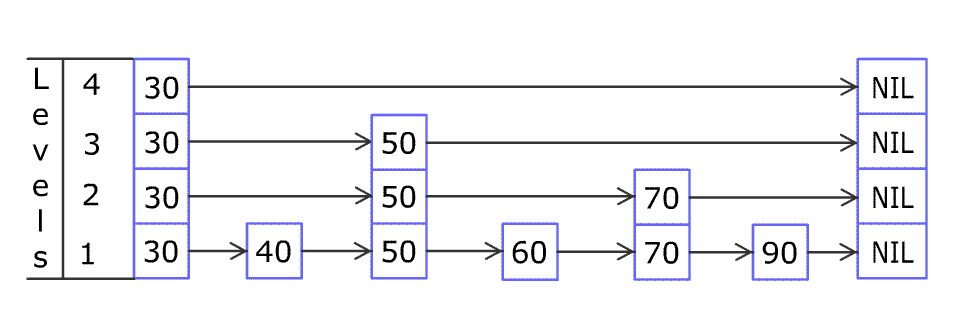
Procedure

1. Linear Structure

1. Skip List and Priority Queue

The Dean’s Office website always crash when we try to select course. So I try to make a server visit waiting queue to avoid the crash. As the students that never select their course should obviously be prior access than others. In order to do that , I use skip list in priority queue .

down here is the implement to insert 80 to the skip list



2. KMP algorithm

The KMP means *Knuth–Morris–Pratt* , because the algorithm was conceived in 1970 by Donald Knuth and Vaughan Pratt, and independently by James H. Morris. The three published it jointly in 1977.

The main idea of this algorithm is to avoid repeated compare.

2. Tree Structure

//I don’t do anything about this…….So I couldn’t say anything about it. The source code of it is not debug.

3. Graph Structure and Hashing

The algorithm down here is two algorithm for finding the shortest paths.

1. Dijkstra's algorithm

In my program , there are 3 inputs at first line. m and q means the initial node and the destination. n refer to how many edge there are.

Dijkstra's algorithm is the fastest algorithm to solve these problem . And it’s very gently , because it have only one key line. In my program , it looks like:

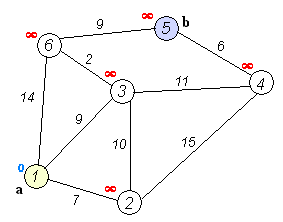
res[i] = c[x][i]+res[x];

//*the array c[][] is an adjacency matrix;*

//*res[x] is the shortest path from m to x;*

*P.S. I set zero to positive infinity in this program , because Dijkstra's algorithm can’t deal with negative edge , so I think , well, it is not important as it can’t deal with zero too :P*

The animation down here is the procedure of Dijkstra.



2. Floyd–Warshall algorithm

Although this algorithm is slower than Dijkstra’s algorithm . And it was published later than Dijkstra’s algorithm (Dijkstra was published in 1959 , but Floyd published in 1962) .

The most advantage of this algorithm is it can deal with negative edge , but Dijkstra can’t. And it is an successful example of dynamic programming.

The dynamic programming is gently too , the one key line (known as Bellman Equation) in my program is:

res[j][k] = res[j][i]+res[i][k];

//*it even don’t need to comment ! We all know what it says!*

*By the way , there’s one improve in this program than the Dijkstra one . As this algorithm can deal with negative , it is obviously that it can’t set zero to positive infinity . Instead, I plus a new input maX as the biggest number of edge and use it to initialize .*

Down here are two greedy algorithm to find a minimum spanning tree.

3.Prim's algorithm

This algorithm is involved in point . In my program , the initial point is the first point input , and always get the nearliest point from the tree.

The array *lst[]* record if this point is in the tree or not.

4.Kruskal's algorithm

Different from prim’s algorithm , the kruskal is involved in edge , the algorithm choose edge as small as possible . So we have to check whether the two point have already connected or not (in the tree).

I use coloring graph to do that:

1. At first , every point have a different color

2. If two point been connected , then change every point that connect with them’s color to the same one.

The array *lst[]* record the color.

4.Internal sort

1.Quicksort

Quicksort is a divide and conquer algorithm. In my program , I use a doubly linked list like this:



My key recursive function called :

“qs(node \*p, int num)” //*define at line 41 in the quicksort.c*

// *p* points to the first one of the section it should sort

// num means the number of elements it should sort

The steps are:

1. Choose the first one as a pivot

2.check others , if it is smaller than pivot , then move it to pivot’s left

3.recursively sort the elements at pivot’s left and right

It is not a stable sort.

The animation down here is perform a quicksort.

