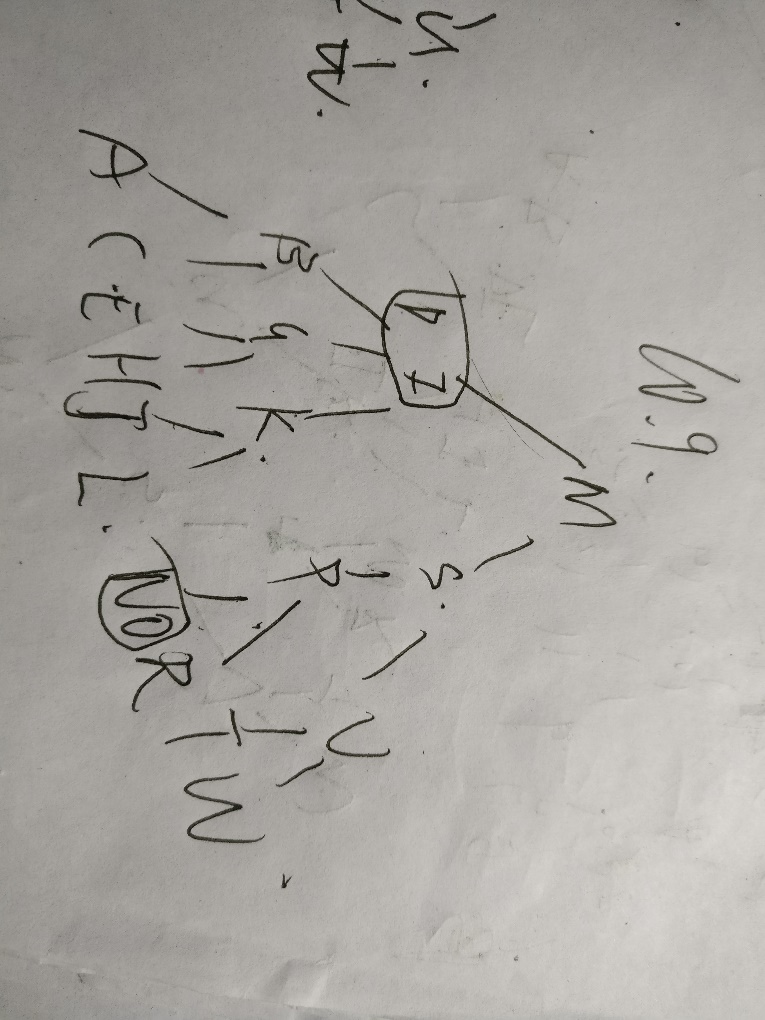
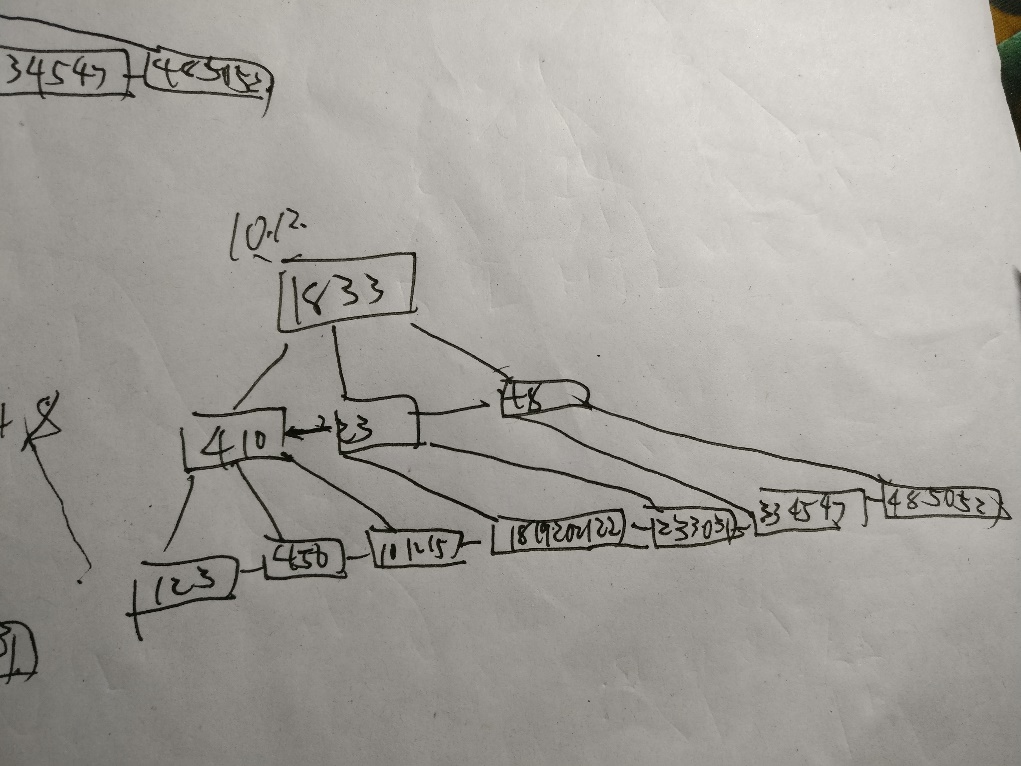
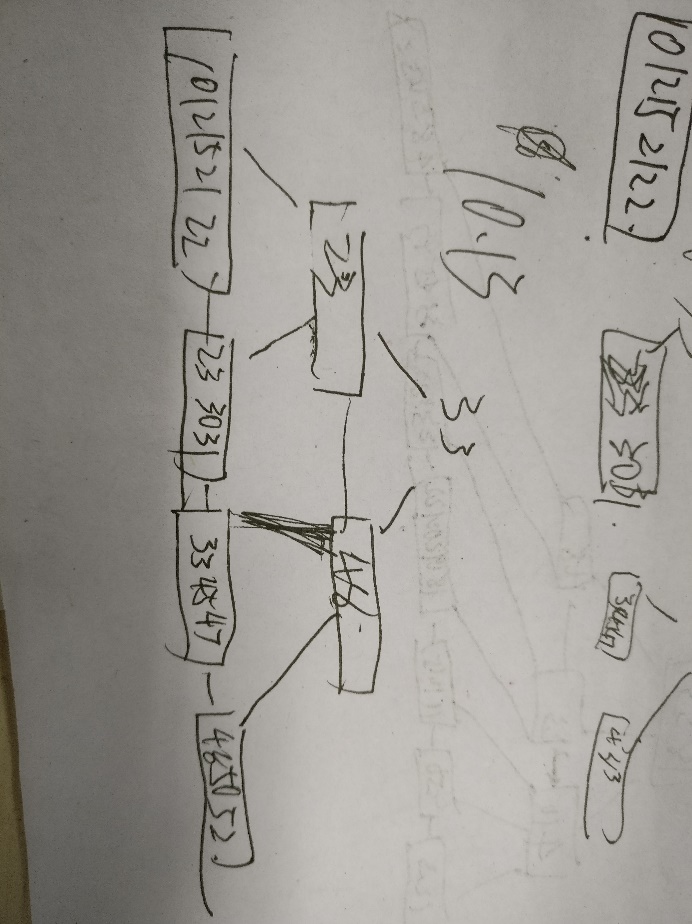
10.9



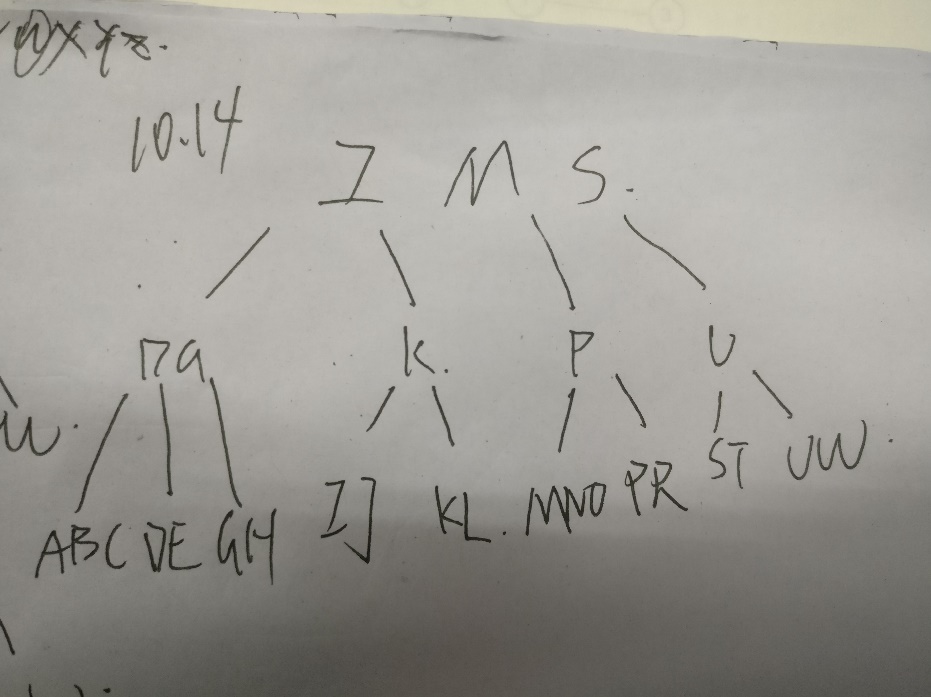
10.12



10.13



10.14



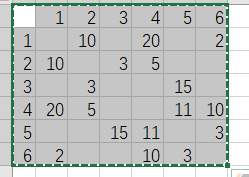
10.15

|  |  |  |
| --- | --- | --- |
|  | 至少 | 最多 |
| 1 | 0 | 15 |
| 2 | 16 | 1500 |
| 3 | 800 | 150000 |
| 4 | 40000 | 15000000 |
| 5 | 2000000 | 1500000000 |

10.16

|  |  |  |
| --- | --- | --- |
|  | 至少 | 最多 |
| 1 | 0 | 50 |
| 2 | 50 | 2500 |
| 3 | 1250 | 125000 |
| 4 | 31250 | 6250000 |
| 5 | 781250 | 312500000 |

11.3

<a>

<b>1->2(10)->4(20)->6(2)

2->1(10)->3(3)->4(5)

3->2(3)->5(15)

4->1(20)->2(5)->5(11)->6(10)

5->3(15)->4(11)->6(3)

6->1(2)->4(10)->5(3)

<c>相邻矩阵需要36\*2=72个字节,邻接表需要24\*4+18\*2+18\*2=168个字节,此时用邻接表占空间更多

<d>相邻矩阵需要36\*2=72个字节,邻接表需要24\*4+18\*1+18\*2=150个字节,此时用邻接表占空间更多

11.4

1->2->3->5->4->6

11.5

void DFS(Graphl\* G, int v) {

cout << v << "->";

G->setMark(v, VISITED);

for (int w = G->first(v); w<G->n(); w = G->next(v, w))

if (G->getMark(w) == UNVISITED)

DFS(G, w);

}

11.6

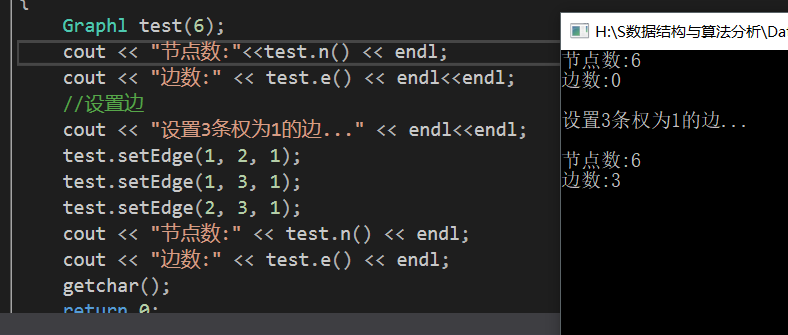
1->2->3

->4->5

->6

额外

1. 图的邻接表实现



#include "Graph.h"

#include "LList.h"

#include "Edge.h"

#include <assert.h>

#define UNVISITED 0

#define VISITED 1

class Graphl : public Graph {

private:

List<Edge>\*\* vertex; // List headers

int numVertex, numEdge; // Number of vertices, edges

int \*mark; // Pointer to mark array

public:

Graphl(int numVert)

{

Init(numVert);

}

//˜Graphl()

//{ // Destructor

// delete[] mark; // Return dynamically allocated memory

// for (int i = 0; i<numVertex; i++) delete[] vertex[i];

// delete[] vertex;

//}

void Init(int n) {

int i;

numVertex = n;

numEdge = 0;

mark = new int[n]; // Initialize mark array

for (i = 0; i<numVertex; i++) mark[i] = UNVISITED;

// Create and initialize adjacency lists

vertex = (List<Edge>\*\*) new List<Edge>\*[numVertex];

for (i = 0; i<numVertex; i++)

vertex[i] = new LList<Edge>();

}

int n() { return numVertex; } // Number of vertices

int e() { return numEdge; } // Number of edges

int first(int v) { // Return first neighbor of "v"

if (vertex[v]->length() == 0)

return numVertex; // No neighbor

vertex[v]->moveToStart();

Edge it = vertex[v]->getValue();

return it.vertex();

}

// Get v’s next neighbor after w

int next(int v, int w) {

Edge it;

if (isEdge(v, w)) {

if ((vertex[v]->currPos() + 1) < vertex[v]->length()) {

vertex[v]->next();

it = vertex[v]->getValue();

return it.vertex();

}

}

return n(); // No neighbor

}

// Set edge (i, j) to "weight"

void setEdge(int i, int j, int weight) {

assert(weight>0, "May not set weight to 0");

Edge currEdge(j, weight);

if (isEdge(i, j)) { // Edge already exists in graph

vertex[i]->remove();

vertex[i]->insert(currEdge);

}

else { // Keep neighbors sorted by vertex index

numEdge++;

for (vertex[i]->moveToStart();

vertex[i]->currPos() < vertex[i]->length();

vertex[i]->next()) {

Edge temp = vertex[i]->getValue();

if (temp.vertex() > j) break;

}

vertex[i]->insert(currEdge);

}

}

void delEdge(int i, int j) { // Delete edge (i, j)

if (isEdge(i, j)) {

vertex[i]->remove();

numEdge--;

}

}

bool isEdge(int i, int j) { // Is (i,j) an edge?

Edge it;

for (vertex[i]->moveToStart();

vertex[i]->currPos() < vertex[i]->length();

vertex[i]->next()) { // Check whole list

Edge temp = vertex[i]->getValue();

if (temp.vertex() == j) return true;

}

return false;

}

int weight(int i, int j) { // Return weight of (i, j)

Edge curr;

if (isEdge(i, j)) {

curr = vertex[i]->getValue();

return curr.weight();

}

else return 0;

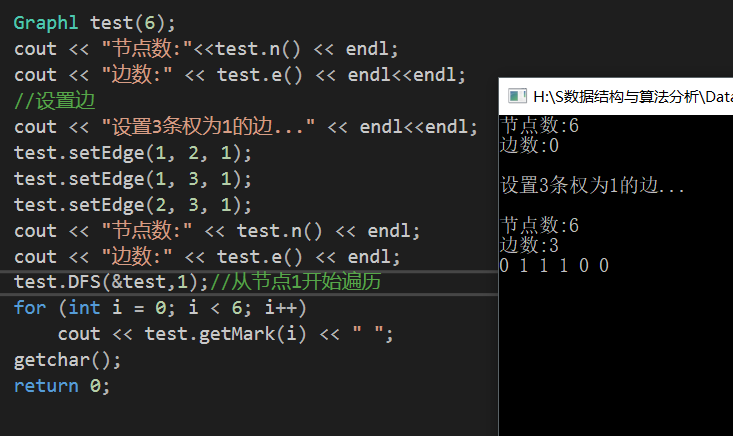
}

int getMark(int v) { return mark[v]; }

void setMark(int v, int val) { mark[v] = val; }

};

1. 深度优先遍历



void DFS(Graphl\* G, int v) { // Depth first search

//PreVisit(G, v); // Take appropriate action

G->setMark(v, VISITED);

for (int w = G->first(v); w<G->n(); w = G->next(v, w))

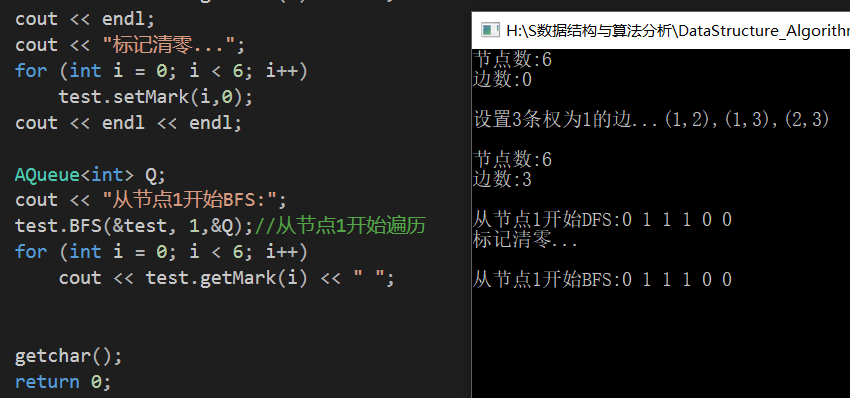
if (G->getMark(w) == UNVISITED)

DFS(G, w);

//PostVisit(G, v); // Take appropriate action

}

1. 广度优先遍历



void BFS(Graph\* G, int start, AQueue<int>\* Q) {

int v, w;

Q->enqueue(start); // Initialize Q

G->setMark(start, VISITED);

while (Q->length() != 0) { // Process all vertices on Q

v = Q->dequeue();

//PreVisit(G, v); // Take appropriate action

for (w = G->first(v); w<G->n(); w = G->next(v, w))

if (G->getMark(w) == UNVISITED) {

G->setMark(w, VISITED);

Q->enqueue(w);

}

}

}