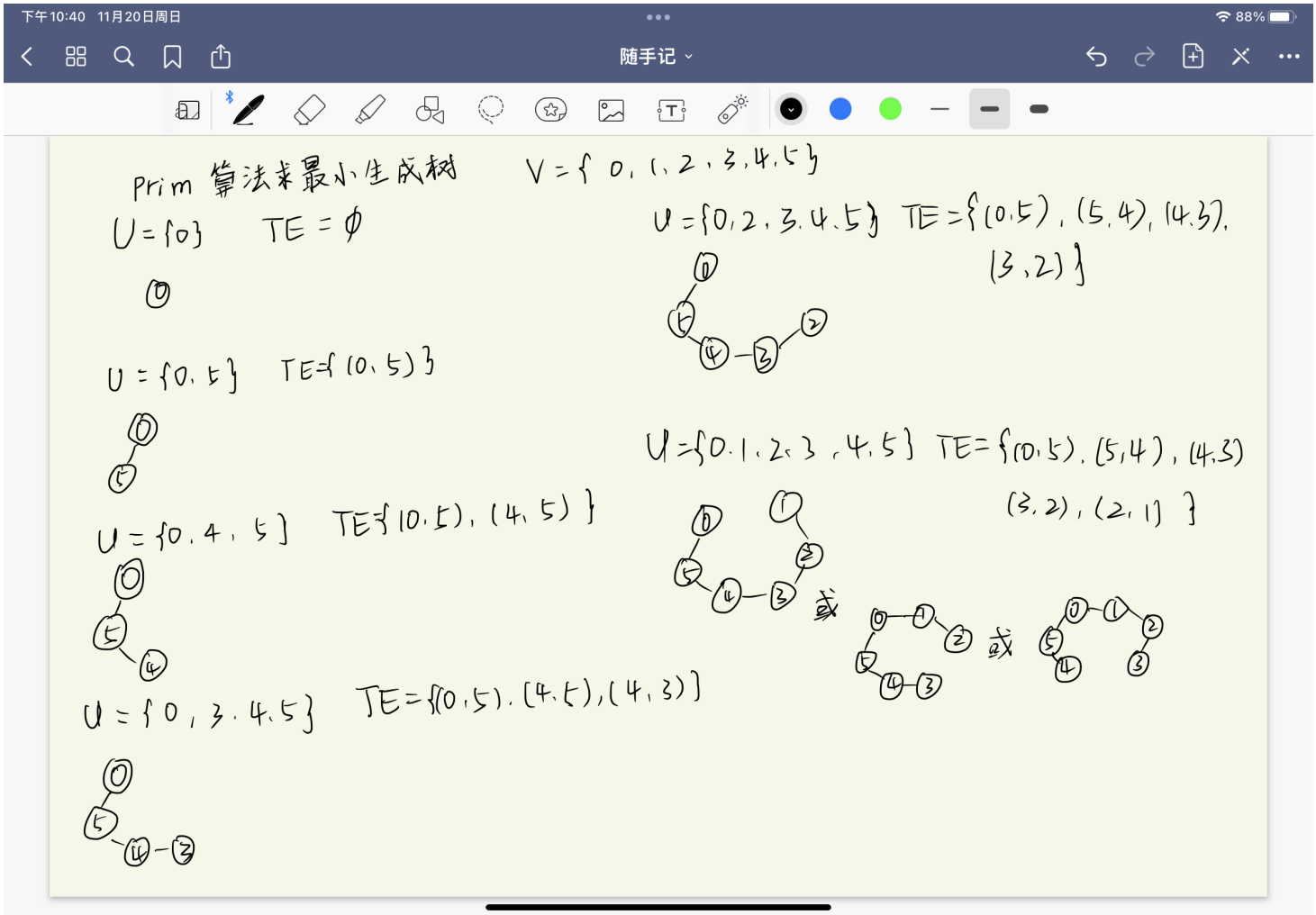
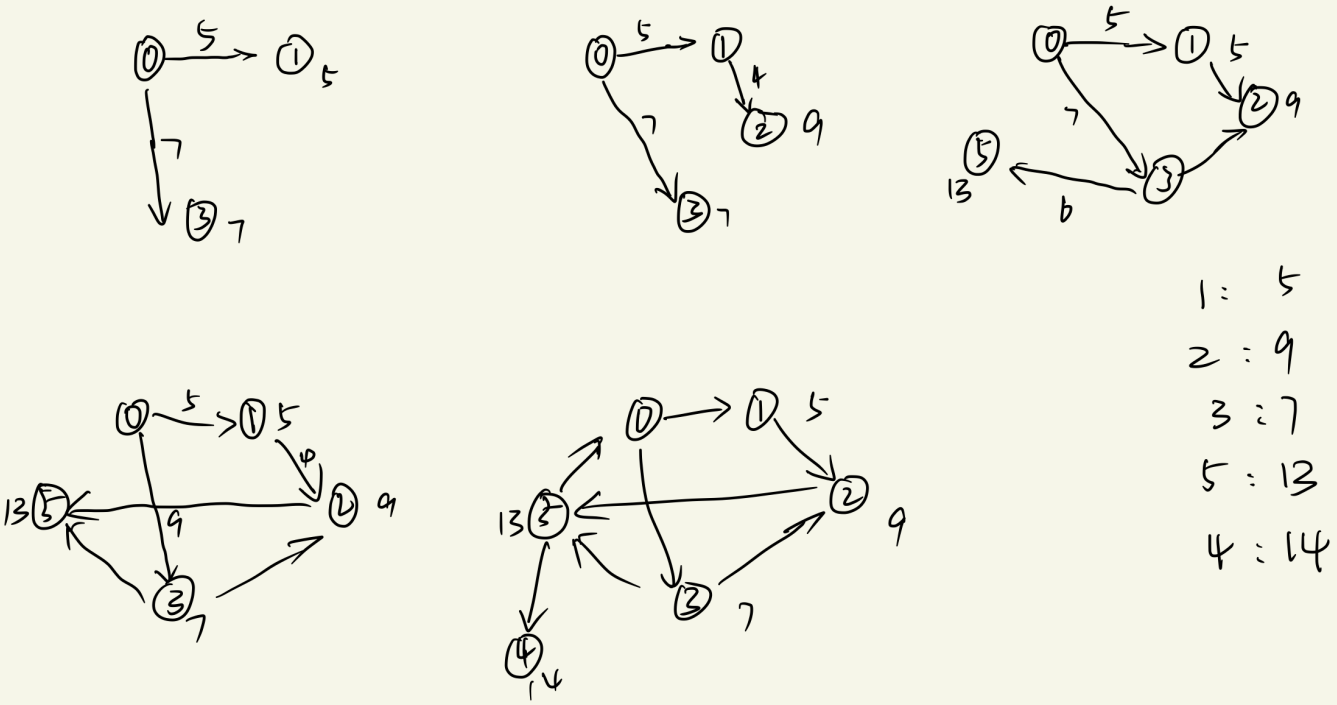
Prim算法 和 Dijkstra算法 实验报告

2021100733
陈旭天

- 手工部分





- 实验源码
  - Source Code for Prim

```
#include<iostream>
#include<vector>
using namespace std;
void prim(vector<vector<int>> &VGraph, vector<int> &lowcost,
vector<int> &closest, vector<bool> &visited)
{
        int size = lowcost.size();
        visited[0] = true;
        for (int i = 1; i < size; i++)
        {
                lowcost[i] = VGraph[0][i];
                closest[i] = 0;
                visited[i] = false;
        }
        cout << "0";
        int weight = 0;
        for (int i = 0; i < size; i++)
        {
                int min = 99999;
                int index = 1;
                for (int j = 0; j < size; j++)
                {
                        if (lowcost[j] < min&&!visited[j])
                        {
                                min = lowcost[j];
                                index = j;
                        }
                }
                if (index == 1 && min == 99999)
                {
                        cout <<
            "\n the value of min shengcheng Tree is:"
            << weight<<endl;
                        return;
                }
                else
                {
                        weight += min;
                }
                cout << " -> " << index;
                visited[index] = true;
                for (int j = 1; j <size; j++)
                {
                        if ((VGraph[index][j]<lowcost[j]) && (!visited[j]))
                        {
                                lowcost[j] = VGraph[index][j];
                                closest[j] = index;
                        }
                }
        }
```

```
}
int main()
{
	int M, N;
	cin >>M>>N;
	vector<vector<int>> VGraph(M);
	vector<int> lowcost(M);
	vector<int> closest(M);
	vector<bool> visited(M);
	for (int i = 0; i < M; i++)
	{
		VGraph[i].resize(M);
	}
	for (int i = 0; i < M; i++)
	{
		for (int j = 0; j < M; j++)
		{
			VGraph[i][j] = 99999;
		}
	}
	for (int i = 0; i < N; i++)
	{
		int a, b;
		cin >> a >> b;
		int length;
		cin >> length;
		VGraph[a][b] = VGraph[b][a] = length;
	}
	prim(VGraph, lowcost, closest, visited);
}
```

- `Source Code for Dijkstra`

```cpp
#include<stdlib.h>
#include<vector>
#include<cstdio>
#include<iostream>

#define NumVertex 6
#define NumEdge 10
#define NotAVertex (-1)
#define Infinity 10000

using namespace std;



struct edge
{
    int from;
    int to;
    int value;
    /* data */
};

typedef struct edge Graph[NumEdge];

struct vertex
{
    vector<pair<int,int>> List;
    bool known;
    int dist;
    int path;
    /* data */
};

typedef struct  vertex Table[NumVertex];

// typedef int vertex;

void ReadGraph(Graph G,Table T){
    for(int i = 0;i < NumEdge;++i){
        T[G[i].from].List.push_back(make_pair(G[i].to,G[i].value));
    }
}

void InitTable(int start,Graph G,Table T){
    ReadGraph(G,T);
    for(int i = 0;i < NumVertex; ++i){
        T[i].known = false;
        T[i].dist = Infinity;
        T[i].path = NotAVertex;
    }
```

```
    T[start].dist = 0;
}

void PrintPath(int V,Table T){

    if(T[V].path != NotAVertex){
        PrintPath(T[V].path, T);
         //recursion to print shortest path to V.
        printf("  to");
    }
    printf("%d",V);
}

void Dijkstra( Table T){
    int V = NumVertex - 1, W;
    int cnt = 1;
    for(;;){
        for(int i = 0;i < NumVertex;++i){
            if(!T[i].known && T[i].dist < Infinity){
                if(T[V].dist > T[i].dist)V = i;
            }
        }
        T[V].known = true;
        cnt++;
        for(pair<int,int> a:T[V].List){
            W = a.first;
            if(!T[W].known){
                if(T[V].dist + a.second < T[W].dist){
                    T[W].dist = T[V].dist + a.second;
                     // update the shortest path for each W
                    T[W].path = V;
                             // update the closest vertex for W
                }

            };

        }
        V = W;    // refresh the value of V is quite crucial.
        if(cnt == NumVertex){
            break;
        }
    }
}

int main(void){
    Graph G = {{0,1,5},{0,3,7},{1,2,4},
                {3,2,5},{3,5,6},{2,0,8},
                {2,5,9},{5,0,3},{5,4,1},
                {4,3,5}};
    Table T;
    InitTable(0,G,T);
```

```
    Dijkstra(T);
    // T[5].path = 4;
    cout << "the path to 1: ";
    PrintPath(1,T);
    printf("\n");
    cout << "the path to 2: ";
    PrintPath(2,T);
     printf("\n");
     cout << "the path to 3: ";
    PrintPath(3,T);
     printf("\n");
     cout << "the path to 4: ";
    PrintPath(4,T);
     printf("\n");
     cout << "the path to 5: ";
    PrintPath(5,T);
    return 0;
}
```

- 实验程序输出验证
  - Prim

```
0 1 5
0 3 7
1 2 4
3 2 5
3 5 6
2 0 8
2 5 9
5 0 3
5 4 1
4 3 5
0 -> 5 -> 4 -> 1 -> 2 -> 3
 the value of min shengcheng Tree is:18
```

  - Dijkstra

```
PS C:\Users\36126\c语言\DataStructureEX> cd "c:\
jkstra } ; if ($?) { .\Dijkstra }
the path to 1: 0  to1
the path to 2: 0  to1  to2
the path to 3: 0  to3
the path to 4: 0  to3  to5  to4
the path to 5: 0  to3  to5
```