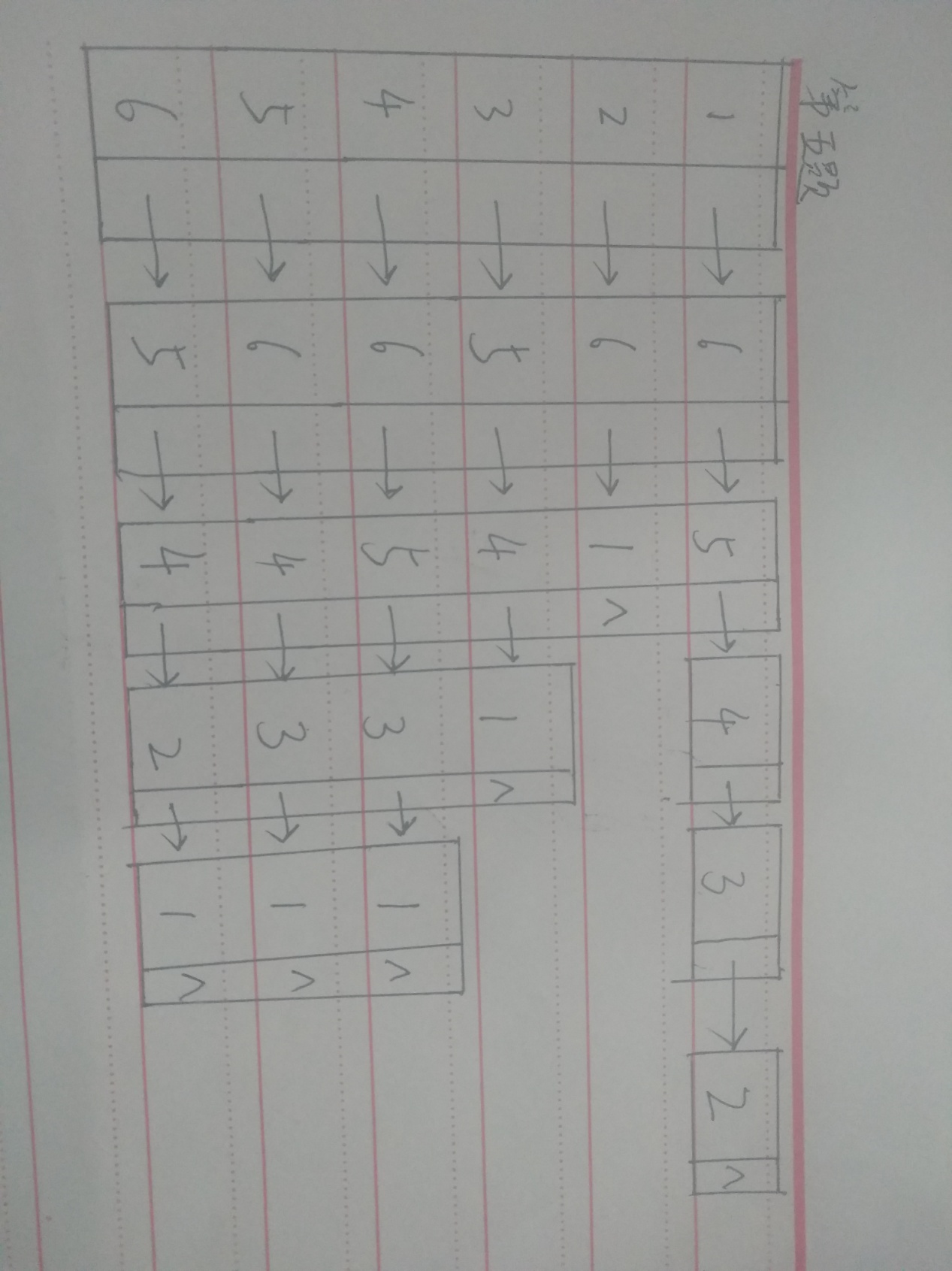
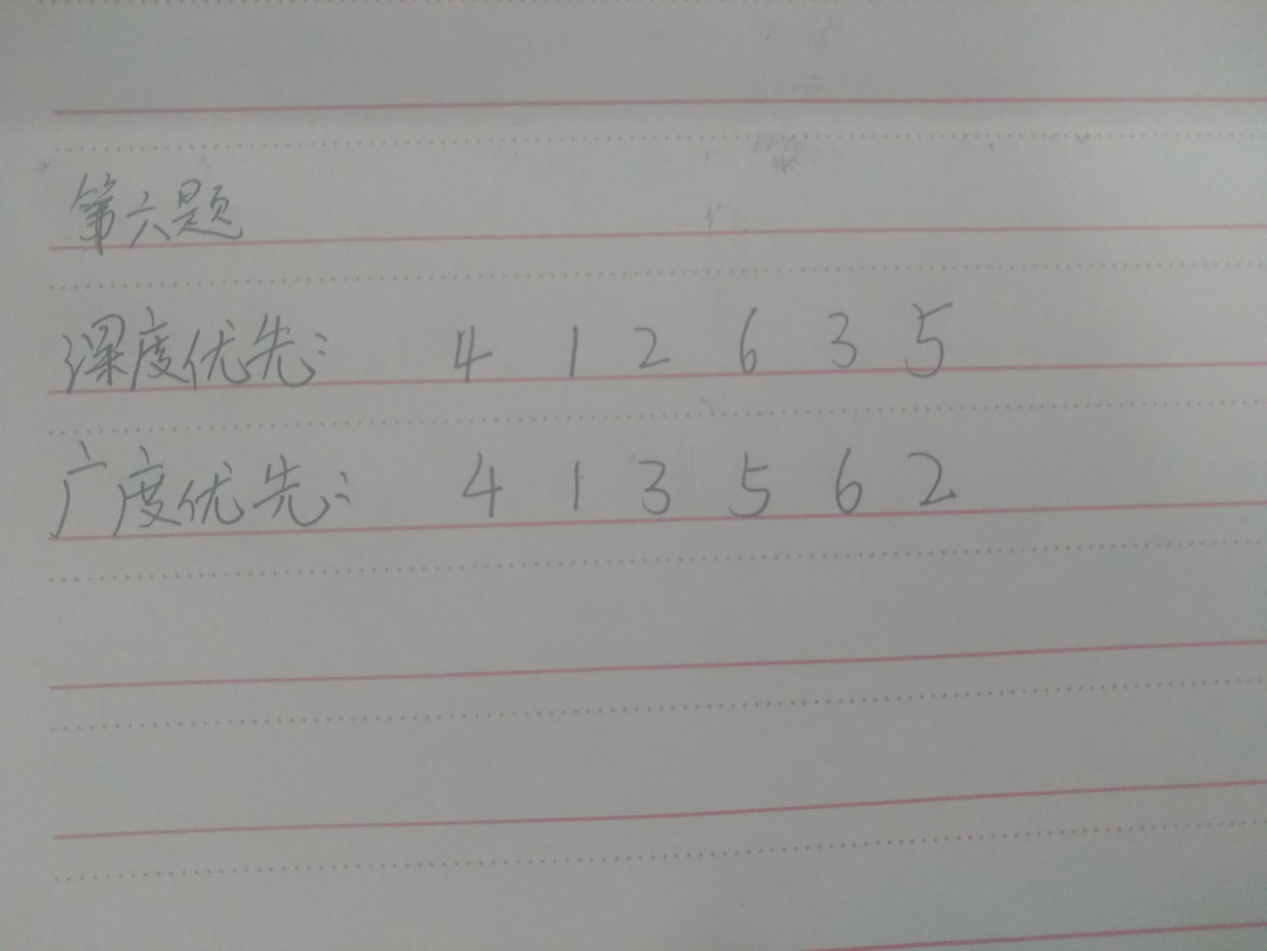
第五题：



第六题：



第七题：

#include<stdio.h>

#define MAX\_VERTAX\_SIZE 20

#define INFINITE 65535

#define OK 1

#define ERROR 0

typedef int Status;

typedef int VertaxElemType;

typedef struct GraphAM{

VertaxElemType VertaxArray[MAX\_VERTAX\_SIZE];

int AdjacencyMatrix[MAX\_VERTAX\_SIZE][MAX\_VERTAX\_SIZE];

int vertaxNum;

int eageNum;

}GraphAM;

typedef struct MinCostEage{

VertaxElemType vertax;

int lowCost;

}MinCostEage;

int LocateVertax(GraphAM G, VertaxElemType c){

int i;

for( i = 0; i < G.vertaxNum; i++ ){

if( c == G.VertaxArray[i] )

return i;

}

return -1;

}

Status CreateUDG(GraphAM\* G){

int i,j,index\_v,index\_w, weight;

VertaxElemType v,w;

printf(" Greate UndiGraph with Cost\n");

printf("Please enter the number of Vertax and Eage:");

scanf("%d %d%\*c", &(G->vertaxNum), &(G->eageNum));

printf("ok, please enter the value of the vertaxes:\n");

for( i = 0; i < G->vertaxNum; i++ ){

scanf("%d%\*c", &(G->VertaxArray[i]));

}

for( i = 0; i < G->vertaxNum; i++ )

for( j = 0; j < G->vertaxNum; j++ )

G->AdjacencyMatrix[i][j] = INFINITE;

for( i = 0; i < G->eageNum; i++ ){

printf("ok,please enter the two Vertax and Wight of eage %d,\nNote:Seperated by Space: ", i+1);

scanf("%d %d %d%\*c", &v, &w, &weight);

if( LocateVertax(\*G, v) != -1 )

index\_v = LocateVertax(\*G, v);

else

return ERROR;

if( LocateVertax(\*G, w) != -1 )

index\_w = LocateVertax(\*G, w);

else

return ERROR;

G->AdjacencyMatrix[index\_v][index\_w] = G->AdjacencyMatrix[index\_w][index\_v] = weight;

}

return OK;

}

Status Prim(GraphAM G, VertaxElemType v){

int i,index\_v,min,min\_index,j;

index\_v = LocateVertax(G, v);

MinCostEage minCost[MAX\_VERTAX\_SIZE];

for( i = 0; i < G.vertaxNum; i++ ){

minCost[i].lowCost = G.AdjacencyMatrix[index\_v][i];

minCost[i].vertax = v;

}

minCost[index\_v].lowCost = 0;

printf("最小生成树:\n");

for( i = 1; i < G.vertaxNum; i++ ){

for( j = 0; j < G.vertaxNum; j++ ){

if( minCost[j].lowCost != 0 ){

min = minCost[j].lowCost;

min\_index = j;

break;

}

}

for( j = 0; j < G.vertaxNum; j++ ){

if( minCost[j].lowCost > 0 && minCost[j].lowCost < min ){

min = minCost[j].lowCost;

min\_index = j;

}

}

printf("(%d, %d)\t", minCost[min\_index].vertax, G.VertaxArray[min\_index]);

minCost[min\_index].lowCost = 0;

for( j = 0; j < G.vertaxNum; j++ ){

if( G.AdjacencyMatrix[min\_index][j] < minCost[j].lowCost ){

minCost[j].lowCost = G.AdjacencyMatrix[min\_index][j];

minCost[j].vertax = G.VertaxArray[min\_index];

}

}

}

}

int main(){

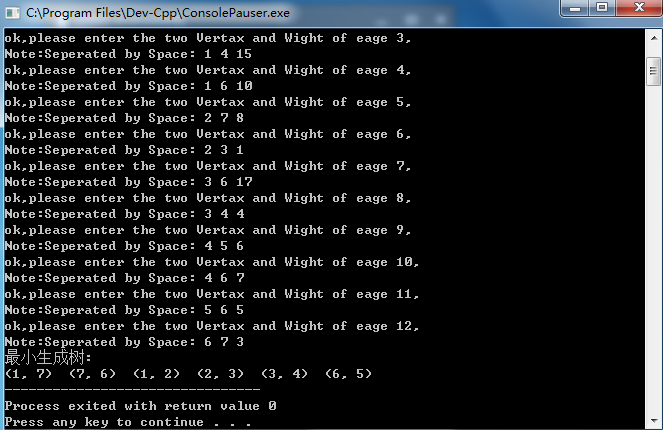
GraphAM G;

CreateUDG(&G);

Prim(G, 1);

return 0;

}



Kruskal算法：

#include<stdio.h>

#include<stdlib.h>

#define MAX\_VERTAX\_SIZE 20

#define MAX\_EAGE\_SIZE 50

#define OK 1

#define ERROR 0

typedef int Status;

typedef int VertaxElemType;

typedef struct EageNode{

int adjacentVertax;

int weight;

struct EageNode\* nextEage;

}EageNode,\*EageNodePtr;

typedef struct VertaxNode{

VertaxElemType data;

EageNodePtr firstEage;

}VertaxNode;

typedef struct GraphAL{

VertaxNode vertaxNodeArray[MAX\_VERTAX\_SIZE];

int vertaxNum;

int eageNum;

}GraphAL;

typedef struct EageTableNode{

int eageHead;

int eageTail;

int weight;

}EageTableNode;

typedef struct GraphET{

VertaxElemType vertaxArray[MAX\_VERTAX\_SIZE];

EageTableNode eageTable[MAX\_EAGE\_SIZE];

int eageNum;

int vertaxNum;

}GraphET;

int LocateVertax(GraphAL G, VertaxElemType v);

Status CreateGraphAL\_UDG\_S(GraphAL\* G);

Status FromAdjacencyListToEageTable(GraphAL G, GraphET\* G\_1);

void PrintEageTable(GraphET G);

void PrintAdjacencyList(GraphAL G);

int LocateVertax(GraphAL G, VertaxElemType v){

int i;

for( i = 0; i < G.vertaxNum; i++ ){

if( G.vertaxNodeArray[i].data == v )

return i;

}

return -1;

}

Status CreateGraphAL\_UDG\_S(GraphAL\* G){

int i,index\_v,index\_w,weight;

VertaxElemType v,w;

EageNodePtr vPtr,wPtr;

printf(" Create UndiGraph with Adjacecy List\n");

printf("please enter the number of VERTAX and EAGE:");

scanf("%d %d%\*c", &(G->vertaxNum), &(G->eageNum));

printf("ok, please enter the value of vertaxes,Seperated by Space\n :");

for( i = 0; i < G->vertaxNum; i++ ){

scanf("%d%\*c", &(G->vertaxNodeArray[i].data));

G->vertaxNodeArray[i].firstEage = NULL;

}

for( i = 0; i < G->eageNum; i++ ){

printf("ok, please enter the two VERTAX and WEIGHT of EAGE %d\n note: Seperated by Space :", i+1);

scanf("%d %d %d%\*c", &v,&w,&weight);

index\_v = LocateVertax(\*G, v);

index\_w = LocateVertax(\*G, w);

vPtr = (EageNode\*)malloc(sizeof(struct EageNode));

if( !vPtr )

return ERROR;

vPtr->adjacentVertax = index\_w;

vPtr->weight = weight;

vPtr->nextEage = G->vertaxNodeArray[index\_v].firstEage;

G->vertaxNodeArray[index\_v].firstEage = vPtr;

}

return OK;

}

Status FromAdjacencyListToEageTable(GraphAL G, GraphET\* G\_1){

int i,j;

EageNodePtr p;

int counterOfEage = 0;

G\_1->eageNum = G.eageNum;

G\_1->vertaxNum = G.vertaxNum;

for( i = 0; i < G.vertaxNum; i++ )

G\_1->vertaxArray[i] = G.vertaxNodeArray[i].data;

for( i = 0; i < G.vertaxNum; i++ ){

p = G.vertaxNodeArray[i].firstEage;

while( p ){

G\_1->eageTable[counterOfEage].eageHead = i;

G\_1->eageTable[counterOfEage].eageTail = p->adjacentVertax;

G\_1->eageTable[counterOfEage].weight = p->weight;

counterOfEage++;

p = p->nextEage;

}

}

EageTableNode temp;

for( i = 0; i < G\_1->eageNum - 1; i++ ){

for( j = i; j < G\_1->eageNum; j++ ){

if( G\_1->eageTable[i].weight > G\_1->eageTable[j].weight ){

temp.eageHead = G\_1->eageTable[i].eageHead;

temp.eageTail = G\_1->eageTable[i].eageTail;

temp.weight = G\_1->eageTable[i].weight;

G\_1->eageTable[i].eageHead = G\_1->eageTable[j].eageHead;

G\_1->eageTable[i].eageTail = G\_1->eageTable[j].eageTail;

G\_1->eageTable[i].weight = G\_1->eageTable[j].weight;

G\_1->eageTable[j].eageHead = temp.eageHead;

G\_1->eageTable[j].eageTail = temp.eageTail;

G\_1->eageTable[j].weight = temp.weight;

}

}

}

if( counterOfEage == G\_1->eageNum ){

return OK;

}

else{

return ERROR;

}

}

void PrintAdjacencyList(GraphAL G){

int i;

EageNodePtr p;

for( i = 0; i < G.vertaxNum; i++ ){

printf(" %d %d : ",i+1, G.vertaxNodeArray[i].data);

p = G.vertaxNodeArray[i].firstEage;

while( p != NULL ){

printf("--->%d weight(%d)", p->adjacentVertax+1,p->weight);

p = p->nextEage;

}

printf("\n");

}

}

int getindex(int\* array, int index){

while( array[index] != 0 ){

index = array[index];

}

return index;

}

Status Kruskal(GraphET G){

int i;

int MSTed[MAX\_VERTAX\_SIZE];

int v,w;

for( i = 0; i < G.vertaxNum; i++ )

MSTed[i] = 0;

printf("利用Kruskal生成的最小生成树：\n");

for( i = 0; i < G.eageNum; i++ ){

v = getindex(MSTed, G.eageTable[i].eageHead);

w = getindex(MSTed, G.eageTable[i].eageTail);

if( v != w ){

printf("(%d, %d)\t", G.vertaxArray[G.eageTable[i].eageHead], G.vertaxArray[G.eageTable[i].eageTail], G.eageTable[i].weight);

MSTed[v] = w;

}

}

return OK;

}

int main(){

GraphAL G;

CreateGraphAL\_UDG\_S(&G);

GraphET G\_1;

FromAdjacencyListToEageTable(G, &G\_1);

Kruskal(G\_1);

return OK;

}

