**package** test;

**public** **class** UnionFind {

**int**[] parent;

**public** UnionFind(**int** n){

parent = **new** **int**[n];

**for**(**int** i = 0; i < n; i++)

parent[i] = -1;

}

**public** **int** find(**int** elem) {

**while**(parent[elem] != -1)

elem = parent[elem];

**return** elem;

}

**public** **void** union(**int** rootA, **int** rootB){

parent[rootB] = rootA;

}

}

**class** MinHeap {

**private** **int** count;

**private** **int** size;

**private** CompKey[] itemArray;

**public** MinHeap(){

count = 0;

size = 64;

itemArray = **new** CompKey[size];

}

**public** **void** insert(CompKey newItem){

**if**(count >= size -1){

System.*out*.println();

**return**;

}

count++;

**int** childLoc = count;

**int** parentLoc = childLoc / 2;

**while**(parentLoc != 0){

**if**(newItem.compareTo(itemArray[parentLoc]) >= 0){

itemArray[childLoc] = newItem;

**return**;

}**else**{

itemArray[childLoc] = itemArray[parentLoc];

childLoc = parentLoc;

parentLoc = childLoc /2;

}

}

itemArray[childLoc] = newItem;

}

**public** CompKey delete(){

**if**(count == 0){

**return** **null**;

}**else**{

**int** currentLoc;

**int** childLoc;

CompKey itemToMove;

CompKey deletedItem;

deletedItem = itemArray[1];

itemToMove = itemArray[count--];

currentLoc = 1;

childLoc = 2\*currentLoc;

**while**(childLoc <= count){

**if**(childLoc < count){

**if**(itemArray[childLoc +1].compareTo(itemArray[childLoc]) < 0)

childLoc++;

}

**if**(itemArray[childLoc].compareTo(itemToMove) < 0){

itemArray[currentLoc] = itemArray[childLoc];

currentLoc = childLoc;

childLoc = 2\*currentLoc;

}**else**{

itemArray[currentLoc] = itemToMove;

**return** deletedItem;

}

}

itemArray[currentLoc] = itemToMove;

**return** deletedItem;

}

}

**public** **int** numverElements(){

**return** count;

}

}

**interface** CompKey{

**public** **int** compareTo(Object o);

}

**class** Edge **implements** CompKey{

**int** head;

**int** tail;

**int** weight;

**public** Edge(**int** h, **int** t, **int** w){

head = h;

tail = t;

weight = w;

}

**public** **int** compareTo(Object value){

**int** a = **this**.weight;

**int** b = ((Edge) value).weight;

**return** a - b;

}

}

**class** Wgraph{

**int** n;

**int** e;

**int**[][] weight;

**public** Wgraph(**int** noOfVertices){

n = noOfVertices;

e = 0;

weight = **new** **int**[n][n];

**for**(**int** i = 0; i < n; i++){

**for**(**int** j = 0; j < n; j++){

**if**(i==j)

weight[i][j] = 0;

**else**

weight[i][j] = 9999;

}

}

}

**public** Wgraph(){

n = 0;

e = 0;

}

**public** **void** insertEdge(**int** i, **int** j, **int** w){

**if**(weight[i][j] == 9999)

e++;

weight[i][j] = w;

}

**public** **void** removeEdge(**int** i, **int** j){

**if**(weight[i][j] != 9999){

e--;

weight[i][j] = 9999;

}

}

**public** Edge[] spanningTree(){

Edge[] T = **new** Edge[n - 1];

**int** Tptr = -1;

MinHeap edgeList = **new** MinHeap();

**for**(**int** i = 0; i < n; i++){

**for**(**int** j = i+1; j < n; j++){

**if**(weight[i][j] != 9999)

edgeList.insert(**new** Edge(i, j, weight[i][j]));

}

}

UnionFind uf = **new** UnionFind(n);

**while**(Tptr < n-1 && edgeList.numverElements() > 0){

Edge edge = (Edge) (edgeList.delete());

**int** a = uf.find(edge.head);

**int** b = uf.find(edge.tail);

**if**(a != b){

Tptr++;

T[Tptr] = edge;

uf.union(a, b);

}

}

**return** T;

}

**void** print(){

**for**(**int** j = 0; j < n; j++){

**for**(**int** i = 0; i < n; i++){

**if**(weight[j][i] != 9999 && weight[j][i] != 0){

System.*out*.print("(" + j + "-"+weight[j][i]+ "-" +i+") ");

}

}

System.*out*.println();

}

}

}

**package** test;

**public** **class** WgraphTest {

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

Wgraph gr = **new** Wgraph(6);

gr.insertEdge(0, 1, 5);

gr.insertEdge(0, 2, 4);

gr.insertEdge(1, 2, 2);

gr.insertEdge(1, 3, 7);

gr.insertEdge(2, 3, 6);

gr.insertEdge(2, 4, 11);

gr.insertEdge(3, 4, 3);

gr.insertEdge(3, 5, 8);

gr.insertEdge(4, 5, 8);

System.*out*.println("입력된 가중치 그래프의 값(정점A-가중치-정점B)");

gr.print();

System.*out*.println("최소 비용 간성(정점A-가중치-정점B)");

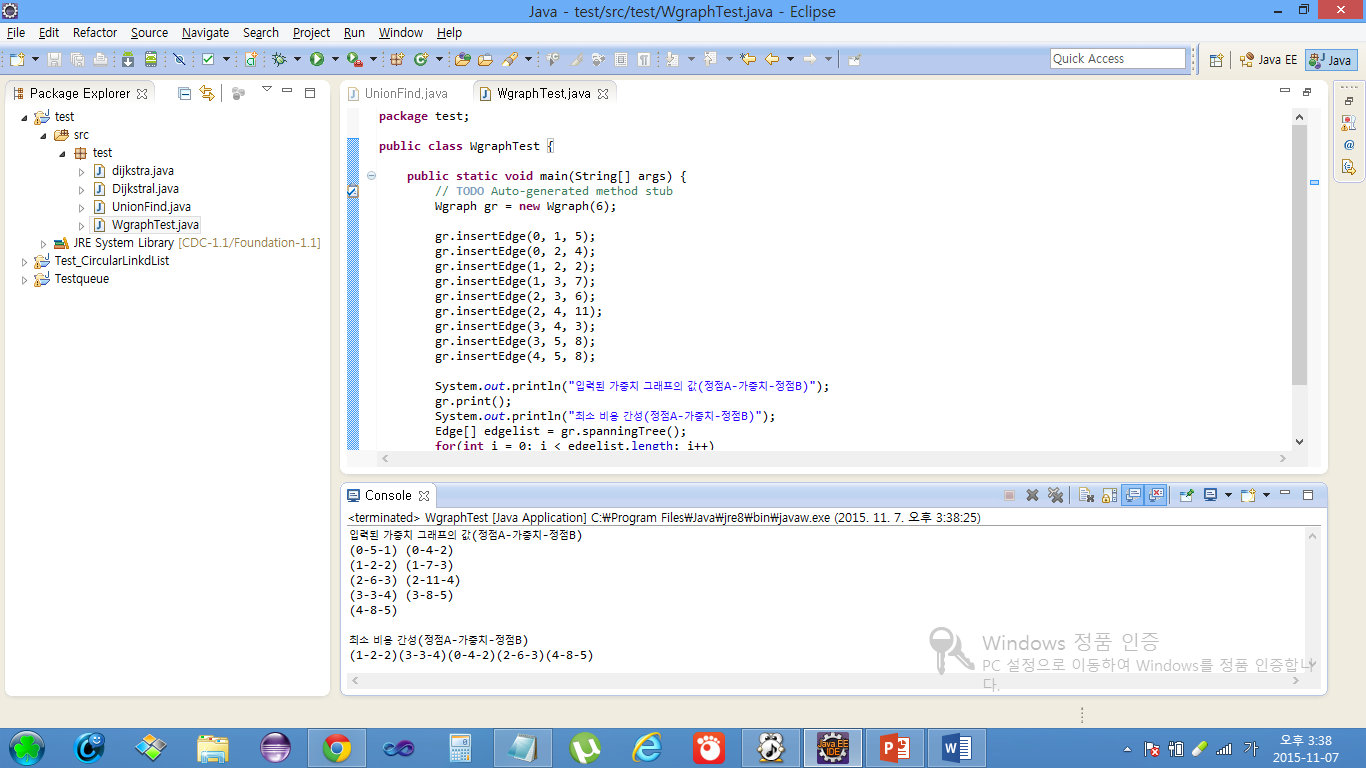
Edge[] edgelist = gr.spanningTree();

**for**(**int** i = 0; i < edgelist.length; i++)

System.*out*.print("("+edgelist[i].head+"-"+edgelist[i].weight+"-"+edgelist[i].tail+")");

}

}



**package** test;

**class** Dijkstral {

**int** weight[][] = **new** **int**[5][5];

**int** Dist[] = **new** **int**[5];

**int** num, u;

**boolean** S[] = **new** **boolean**[5];

**public** Dijkstral(**int** w[][],**int** nv){

num = nv;

**for**(**int** i = 0; i< num; i++){

**for**(**int** j = 0; j < num; j++){

weight[i][j] = w[i][j];

}

}

}

**public** **void** shortesPath(**int** v){

**for**(**int** i = 0; i<num; i++){

S[i] = **false**;

Dist[i] = weight[v][i];

}

S[v] = **true**;

Dist[v] = 0;

**for**(**int** i = 0;i<num-2;i++){

**int** mindist = 999;

**for**(**int** j=0;j<num;j++){

**if**(Dist[j]<mindist && S[j]==**false**){

u=j;

mindist=Dist[j];

}

}

S[u] = **true**;

**for**(**int** w = 0; w<num;w++){

**if**(S[w]==**false**){

**if**(Dist[w]>(Dist[u]+weight[u][w])){

Dist[w] = (Dist[u]+weight[u][w]);

}

}

}

}

}

**public** **void** print\_ds(){

**for**(**int** i = 0; i < num; i++)

System.*out*.print(S[i] + "\t");

System.*out*.println();

**for**(**int** i = 0; i< num; i++)

System.*out*.print(Dist[i] + "\t ");

System.*out*.println();

}

}

**package** test;

**public** **class** dijkstra {

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

**int** nv =5;

**int** weight[][] = {{0,2,5,999,3},{999,0,999,4,10},{999,999,0,6,2},{999,999,999,0,999},{999,999,1,2,0}};

Dijkstral ds = **new** Dijkstral(weight,nv);

ds.shortesPath(0);

ds.print\_ds();

}

}

