VE477
Lab2 Report
Wang Yichao
517370910011

Source code for kruskal and prim can be seen in main2.c and main.c

Use make to compile

1. This is for prim

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef struct _Edge
{
	int start, end, weight;
}Edge;

typedef struct _Graph
{
	// V number of vertices, E number of edges
	int V, E;
	Edge* edge;
}Graph;

Graph* createGraph(int V, int E)
{
	Graph* graph = (Graph*) malloc(sizeof(Graph));
	graph->V = V;
	graph->E = E;
	graph->edge = (Edge*) malloc(E * sizeof(Edge));
	return graph;
}

typedef struct _Subset
{
	int parent;
	int rank;
}Subset;

//for qsort
int Comp(const void* a, const void* b)
{
	Edge* a1 = (Edge*)a;
	Edge* b1 = (Edge*)b;
	return a1->weight - b1->weight;
}

int Comp2(const void* a, const void* b) // only for joj format
{
	Edge* a1 = (Edge*)a;
	Edge* b1 = (Edge*)b;
	if (a1->start == b1->start){
		return a1->end - b1->end;
	}
	return a1->start - b1->start;
}
```

```
void PrimMST(Graph *graph){

        int V = graph->V;
        Edge result[V];
        int e = 0;

   qsort(graph->edge,graph->E,sizeof(Edge),Comp);
   int *connected = malloc(graph->E*sizeof(int));

   for (int i = 0; i < V; i++)
        connected[i] = 0;

   connected[graph->edge[0].start] = 1;
   for (int i = 0; i < V; i++){
      for (int j = 0; j < graph->E; j++){
         if (connected[graph->edge[j].start] + connected[graph->edge[j].end] == 1){
            result[i] = graph->edge[j];
            e++;
            connected[result[i].start] = 1;
            connected[result[i].end] = 1;
            break;
         }
      }
   }

   qsort(result, e, sizeof(Edge), Comp2);
        for (int i = 0; i < e; i++)
                printf("%d--%d\n", result[i].start, result[i].end);
   free(connected);
}

int main()
{
        int V = 0; // Number of vertices in graph
        int E = 0; // Number of edges in graph
        scanf("%d", &E);
        scanf("%d", &V);
        Graph* graph = createGraph(V, E);
        //printf("%d %d\n", V, E);
        for(int i = 0; i < E; i++){
                int a;
                int b;
                scanf("%d %d %d", &a, &b, &graph->edge[i].weight);
                if (a<b){
                        graph->edge[i].start = a;
                        graph->edge[i].end = b;
                }
                else{
                        graph->edge[i].start = b;
                        graph->edge[i].end = a;
                }
                //printf("%d %d %d\n ", a, b, graph->edge[i].weight);
        }
        PrimMST(graph);
        free(graph);

        return 0;
}
```

2. This is for kruskal

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef struct _Edge
{
	int start, end, weight;
}Edge;

typedef struct _Graph
{
	// V number of vertices, E number of edges
	int V, E;
	Edge* edge;
}Graph;

Graph* createGraph(int V, int E)
{
	Graph* graph = (Graph*) malloc(sizeof(Graph));
	graph->V = V;
	graph->E = E;
	graph->edge = (Edge*) malloc(E * sizeof(Edge));
	return graph;
}

typedef struct _Subset
{
	int parent;
	int rank;
}Subset;

int find(Subset* subsets, int i) //find root
{
	if (subsets[i].parent != i)
		subsets[i].parent = find(subsets, subsets[i].parent);
	return subsets[i].parent;
}

void Union(Subset* subsets, int x, int y)
{
	int xroot = find(subsets, x);
	int yroot = find(subsets, y);

	// three outs
	if (subsets[xroot].rank > subsets[yroot].rank)
		subsets[yroot].parent = xroot;
	else if (subsets[xroot].rank < subsets[yroot].rank)
		subsets[xroot].parent = yroot;
	else
	{
		subsets[yroot].parent = xroot;
		subsets[xroot].rank++;
	}
}

//for qsort
int Comp(const void* a, const void* b)
{
	Edge* a1 = (Edge*)a;
	Edge* b1 = (Edge*)b;
	return a1->weight - b1->weight;
}

int Comp2(const void* a, const void* b) // only for joj format
{
	Edge* a1 = (Edge*)a;
	Edge* b1 = (Edge*)b;
	if (a1->start == b1->start){
		return a1->end - b1->end;
	}
	return a1->start - b1->start;
}

void KruskalMST(Graph* graph)
{
	int V = graph->V;
	Edge result[V];
	int e = 0;
	int i = 0;

	qsort(graph->edge, graph->E, sizeof(graph->edge[0]), Comp);

	Subset *subsets = (Subset*) malloc( V * sizeof(Subset) );

	for (int v = 0; v < V; v++)
	{
		subsets[v].parent = v;
		subsets[v].rank = 0;
	}

	while (e < V - 1 && i < graph->E)
	{
		Edge next_edge = graph->edge[i++];
		int x = find(subsets, next_edge.start);
		int y = find(subsets, next_edge.end);

		if (x != y)
		{
			result[e++] = next_edge;
			Union(subsets, x, y);
		}
	}

	qsort(result, e, sizeof(Edge), Comp2);
	for (i = 0; i < e; ++i)
		printf("%d--%d\n", result[i].start, result[i].end);
	return;
}

int main()
{
	int V = 0; // Number of vertices in graph
	int E = 0; // Number of edges in graph
	scanf("%d", &E);
	scanf("%d", &V);
	Graph* graph = createGraph(V, E);
	//printf("%d %d\n", V, E);
	for(int i = 0; i < E; i++){
		int a;
```

```
                int b;
                scanf("%d %d %d", &a, &b, &graph->edge[i].weight);
                if (a<b){
                        graph->edge[i].start = a;
                        graph->edge[i].end = b;
                }
                else{
                        graph->edge[i].start = b;
                        graph->edge[i].end = a;
                }
                //printf("%d %d %d\n ", a, b, graph->edge[i].weight);
        }
        KruskalMST(graph);
        free(graph);

        return 0;
}
```

The time complexity for kruskal is $O(V\ \log V + E\ \log V) = O(E\ \log E)$, prim is the same. But using Fibonacci Heap, the prim can be improved to $O(E + \log V)$. So according to the formula and experiment, prim is better when we have a dense graph(which means many edges). Kruskal is better in sparse graph(less edges).

2. Python

With is used in exception handling to make the code cleaner and much more readable

Decorators allow us to wrap another function in order to extend the behavior of wrapped function, without permanently modifying it.

Iterators are objects that can be iterated upon.

Generator functions allow you to declare a function that behaves like an iterator, i.e. it can be used in a for loop.