Name: Weiren Wang

UFID: 8198-8318

UF Email: weirenwang@ufl.edu

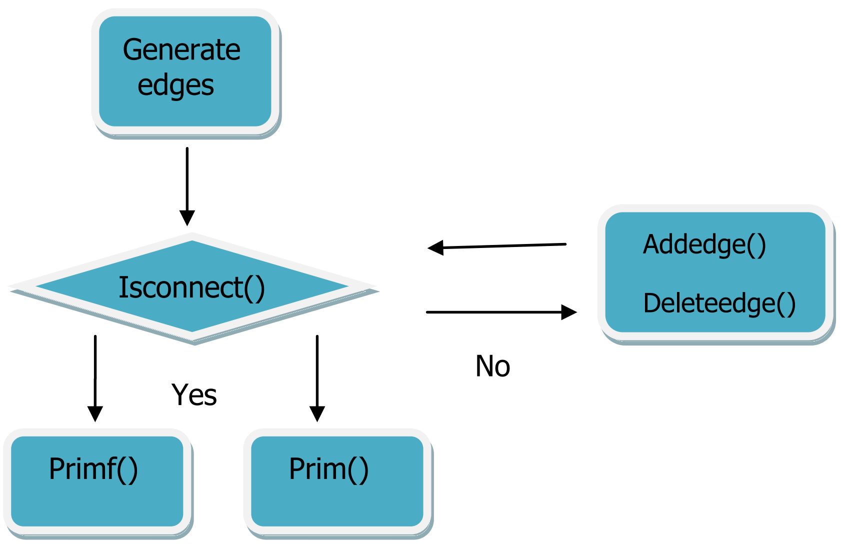
Part 1 Compile

Platform: Ubuntu C++

Compile steps: 1. c++ -g proj.cpp -o proj

```
2 ./proj  –r  n  d   (random mode)
  ./proj  –s  filename (simple scheme mode)
  ./proj  -f  filename(fibonacci scheme mode)
```

Part 2 Program Structure



The program will firstly generate random edges in the set. Then we use Depth First Search or Breadth First Search to judge whether the graph is connected. If not, we add a edge between two nodes(one is connected, the other is not), and delete a edge from the redundant edge set.(edges that do not influence connectivity).

Main function:

Generate edges: randomly generate edges

```
void randominit(int Gnum,int density,HeadNode hnode[])
```

Isconnect: judge whether the graph is connect `bool isconnect(bool isvisited[], int Gnum)`

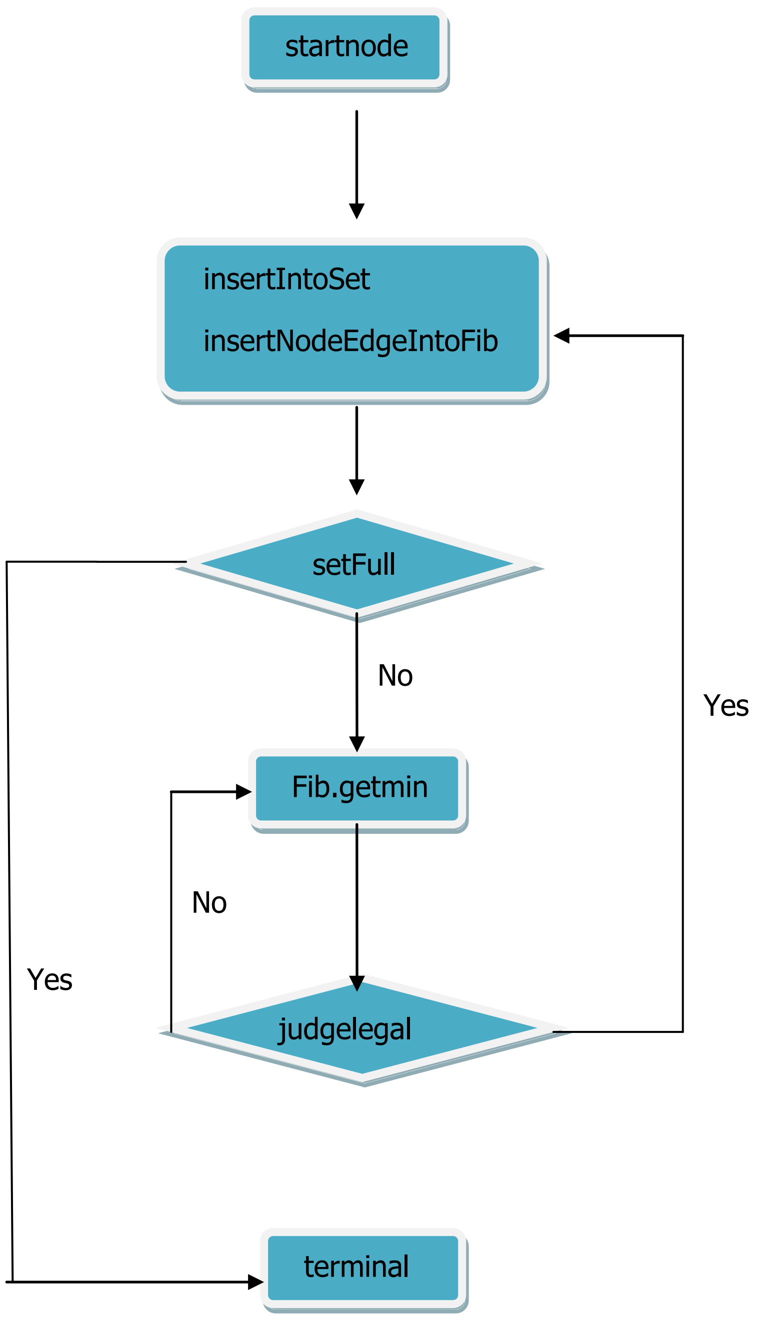
Primf: The funtion return the weight of the mst. The function call addnewedges and selectminedgenode untill the set<Edge> t has generated the mst.

`int primf(HeadNode hnode[], int Gnumm)`

This function is the prim algorithm for the basic scheme.The function return the total weight of mst. we first insert a start node into the set<HeadNode> tv. Then get a minmum edge,whose two end nodes, one is in tv set, the other is not. When the set<Edge> t size is equal to Gnumm, which is size of mst. The algorithm stop.

`int prim(HeadNode hnode[],int Gnumm)`

Primf



The algorithm begin at a startnode, then we insert the node into set tv. In the same time, we insert all edges of the node into a Fibonacci heap. If the tv set is node full, then we getmin from the fibnacci heap, judge whether this edge connect two nodes already in set. If it is not illegal, we continue getmin until a edge connect a node that not in the set tv. Then we add the new node into the tv set and continue. If the set tv is full , the algorithm end.

Function:

This is to print each node in a node set. It's used to print results and debuging.

```
void printtv(set<HeadNode> &tv)
```

The function is to compute the new edges which need to be added to the Fibnonacci heap. The program get each edge of addnode(begin),and judge the whether the end of edge belong to tv. tv is the node set that already joined the spanining tree. If the end node of edge has already in tv, get the next edge. Otherwise, insert the edge into fibonacci heap.

```
void addnewedges(HeadNode hnode[], HeadNode addnode, Fibonacci<Edge> &h,
set<HeadNode> &tv)
```

This function is to get the min edge from the fibonacci heap. Then the function judge whether this edge connects two nodes already in a tv set. If one of them is node in the tv set. Then this edge is part of MST. The function insert the edge into set<Edge> t. Then return the new node, which will join the tv set later. Otherwise, we simply ignore this edge and get another edge, untill the edge's one node is not in the tv set.

```
HeadNode selectminedgenode(HeadNode hnode[], Fibonacci<Edge> &h, set<HeadNode>
&tv, set<Edge> &t, set<Edge> &rmedge)
```

This function is add a new node, whose info is end, value is weight,to the HeadNode addnode. This function is used for building the adjacency list

```
void add(HeadNode &addnode,int end, int weight)
```

This function is to delete a node from a Headnode, the node info is end. This function is used fo building the adjacency list.

```
void dlt(HeadNode &dltnode,int end)
```

This function is to judge whether the HeadNode hnode[begin] has a adjacancy node end

```
bool isexist(HeadNode hnode[],int begin,int end)
```

This function is to judge whether the nodes are all visited.This function it used for bfs algorithm.

```
bool isconnect(bool isvisited[], int Gnum)
```

This function is a depth first search algorithm. The function is to judge whether the graph is connected. If it is connected, return true. Otherwise, return false.

```
bool bfs(HeadNode hnode[], bool isvisited[], int Gnum,set<Edge> &redundant,int startnum)
```

This function is to find a node that hasn't been visited, return the node's id

```
int unconnectnode(bool isvisited[], int Gnum)
```

This funciton is to find a node that has been visited.return the node's id.

```
int connectnode(bool isvisited[], int Gnum)
```

This function is to print out the adjacency list(graph) that the program generated.

```
void printgraph(HeadNode hnode[],int Gnum)
```

This function is to make every edge be undirected.The method is to find a (begin,end,weight) edge, insert a (end,begin,weight) edge to adjacency list

```
void undirectgraph(HeadNode hnode[],int Gnum)
```

Fibonacci

This is constructor of Fibonacci heap, the function generate a new FNode to store value

`Fibonacci(V value)`

This is destructor of Fibonacci heap.

`virtual ~Fibonacci()`

This function is to merge two FNodes.

`FNode<V>* merge(FNode<V>* aheap, FNode<V>* bheap)`

This Function is to generate a new node to store the value then return the node pointer

`FNode<V>* initnode(V value)`

This function is to insert a new value into the Fibonacci heap

`FNode<V>* insert(V value)`

This function is to reture the minmimue V value in the fibonacci heap.

`V getmin()`

This function is to add a child to a parent node.Note: In this funtion, the parent->value is smaller than child->value

`FNode<V>* _addchild(FNode<V> *parent, FNode<V> *child)`

This function is to set a node list parent and childcut to NUll and False.

`void _unMarkAndParentAll(FNode<V> *n)`

This function is to remove the min element in the fibonacci heap. details are discussed in the function.

```
void removemin()
```

This part is for the cut function in the Fibonacci heap

```
FNode<V>* _cut(FNode<V> *n)
```

This function is to decrease a node's value by value. node-> value=node->value->value

```
FNode<V>* decreasekey(FNode<V>* n, V value)
```

Part 3 Result

Expectation:

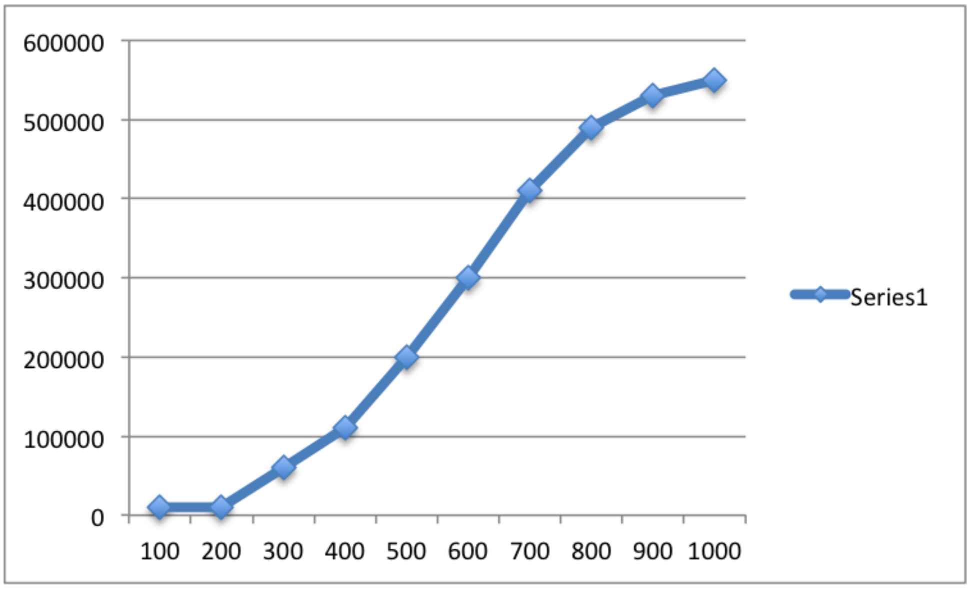
1.In the Fibonacci heap implementation, the complexity should be O(|E| + |V| log |V|)

2.In the adjancency list implementation, the complexity should be O($|V|^2$),

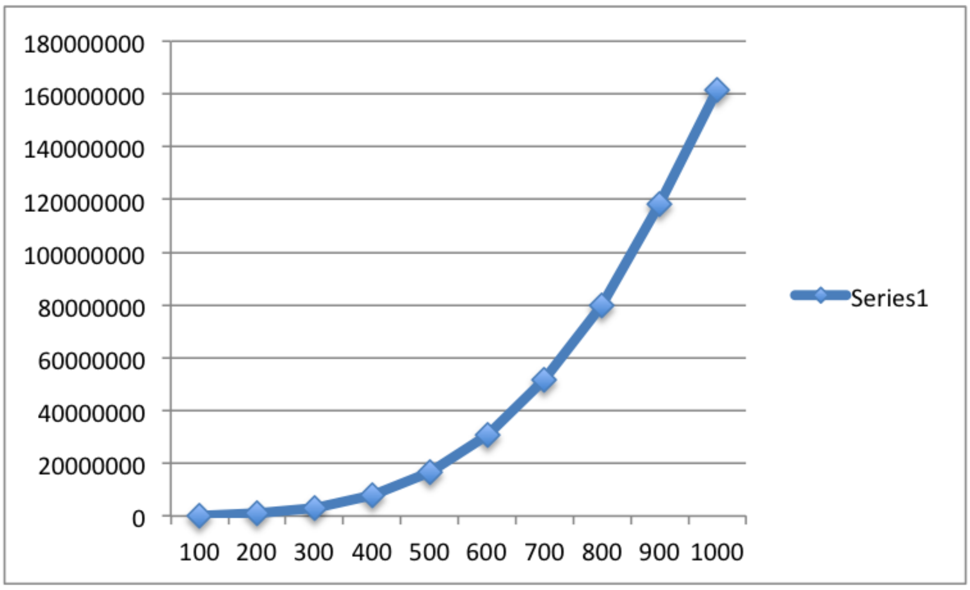
3.In the Fibonacci heap implementation, the density increase will influence the complexity in linear.

Result:

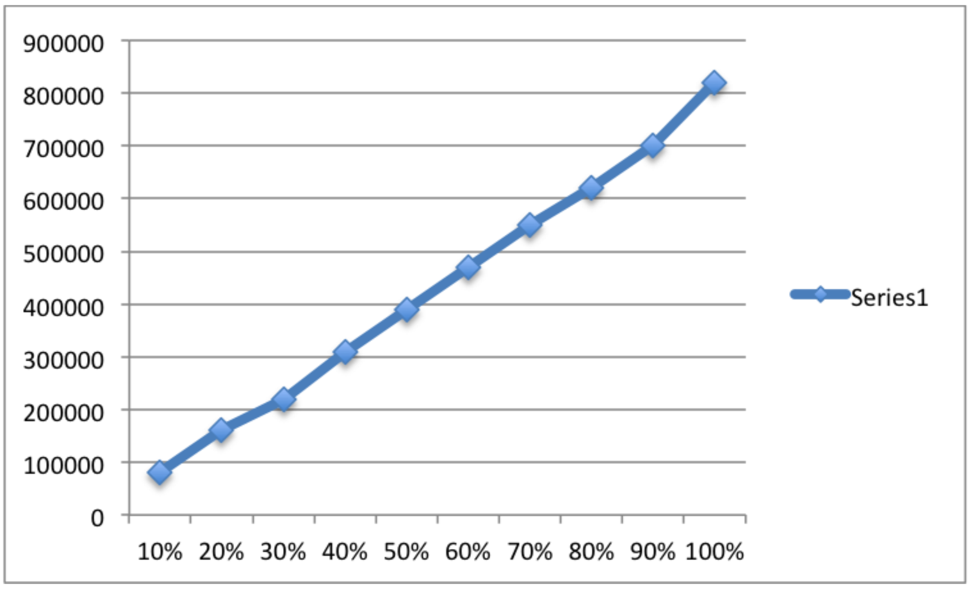
Fibonacci heap: nodes number increase density:70%



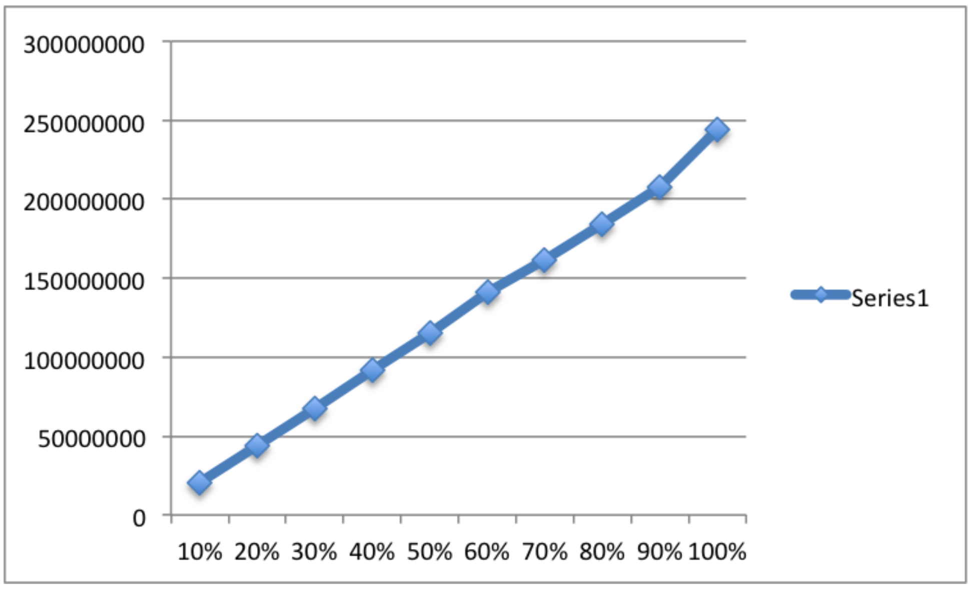
Simple scheme: nodes number increase density:70%



Fibonacci heap: density increase nodes num: 1000



Simple scheme: density increase nodes num: 1000


Series1