# Fredman-Tarjan.cpp Documentation

Harman Kumar
2013CS10224

September 17, 2015

## 1 The Union Find:

The class UF is an implementation of the Union Find data structure. Union by rank and path compression heuristics were added to make the implementation efficient.

```
class UF
{
    int cnt;  //count of the number of disjoint sets.
    unordered_map<string, int> sz;  // Size of a set.
    unordered_map<string, string> mapping;  // For parent relation.

public:
    UF(set<string> str) // Create an union find data structure with
     N isolated sets.

    ~UF() // Destructor.

    string find(string p) // Return the id of component
    corresponding to object p.

    void merge(string x, string y)  // Replace sets containing x
    and y with their union.

    bool connected(string x, string y)   // Are objects x and y in
    the same set?

    int count()   // Return the number of disjoint sets.

};
```

## 2 FibonacciHeap.h

In order to implement the Fredman-Tarjan algorithm, the fibonacci heap data structure was implemented.

Following is the structure of a node of the fibonacci heap:

```
template <class V> struct node
{

public:
  // Data members.

  node<V>* prev;  // Pointer to previous sibling
  node<V>* next;  // Pointer to the next sibling.
  node<V>* child; // Pointer to a child.
  node<V>* parent;  // Pointer to the parent.
  V value;  // Key, Value pair in the node.
  int degree; //Degree of the node.
  bool marked;  // For cascading cuts, to check whether the node is
     marked or not.

  // Get functions.

  node<V>* getPrev() {return prev;}
  node<V>* getNext() {return next;}
  node<V>* getChild() {return child;}
  node<V>* getParent() {return parent;}
  V getValue() {return value;}
  bool isMarked() {return marked;}
  bool hasChildren() {return child;}
  bool hasParent() {return parent;}
};
```

Following is the structure of the fibonacci heap (only user accesible functions are mentioned here):

```
template <class V> class FibonacciHeap {
protected:
  node<V>* heap;  //Pointer to the min node.

public:

  FibonacciHeap() // Constructor
  {
    heap = NULL;
  }

  node<V>* emplace(V value) // Inserts a node in the heap with
    value V.
  {

  }

  void merge_lazy(FibonacciHeap& other) // Links the two heaps
    together and updates the min pointer.
  {

  }

  bool isEmpty() // Tells whether the heap is empty.
  {

```

```
25  }
26
27  V top() // Returns the value of the smallest element of the heap.
28  {
29
30  }
31
32  V pop() // Removes the smallest element of the heap
33  {
34
35  }
36
37  void decreaseKey(node<V>* n, V value) // Modifies the value of
      the node pointed by n to the new value.
38  {
39
40  }
```

# 3 fredman_tarjan(UF disjoint_set)

1. One iteration of the fredman-tarjan algorithm takes as input the set of "super-vertices" on which the MST has to be calculated.
2. It performs an iteration of the algorithm on the vertices and stores the edges that have been added to the MST.
3. Finally a contraction is performed on the set of edges and vertices.

# 4 I/O specification

The graph is read from **RandomGraph.txt**, stored in the form of an adjacency list and the MST is written in **tarjan_time.txt**

# 5 Requirements and Execution

The system must have c++11 and boost libraries on the system.

To execute the code:
g++ -o0 -I <path to boost libraries> -std=c++11 fredman-tarjan_fibonacci_heap.cpp -o tarjan