Lab 6
Sets and Union/Find Structures

Michael Morikawa & James Daza

April 16, 2020

Lab Questions

Question 1 Describe the template method pattern.
It is a design pattern that uses a generic way to solve a problem, and then redefined certain steps of that process to fit the specifics of the problam that you are trying to solve.

Question 2 What are some applications for the find/union partition structures?
Some applications for this data structure is to keep track of connected components of a graph and is also used in other graph related algorithms.

Source Code

Merge.hpp

```
#pragma once
#include <list>

//Code provided by the book with slight format changes

template <typename E>
class Merge
{                                       // generic Merge
public:                                 // global types
    typedef std::list<E> List;          // list type
    void merge(List &A, List &B, List &C); // generic merge function
protected:                              // local types
    typedef typename List::iterator Itor;  // iterator type
                                        // overridden functions
    virtual void fromA(const E &a, List &C) = 0;
    virtual void fromBoth(const E &a, const E &b, List &C) = 0;
    virtual void fromB(const E &b, List &C) = 0;
};

template <typename E> // generic merge
void Merge<E>::merge(List &A, List &B, List &C)
{
    Itor pa = A.begin(); // A's elements
    Itor pb = B.begin(); // B's elements
    while (pa != A.end() && pb != B.end())
    { // main merging loop
```

```
        if (*pa < *pb)
        {
            fromA(*pa++, C); // take from A
        }
        else if (*pa == *pb)
        {
            fromBoth(*pa++, *pb++, C); // take from both
        }
        else
        {
            fromB(*pb++, C); // take from B
        }
    }
    while (pa != A.end()) // take rest from A
    {
        fromA(*pa++, C);
    }
    while (pb != B.end()) // take rest from B
    {
        fromB(*pb++, C);
    }
}

template <typename E> // set intersection
class IntersectMerge : public Merge<E>
{
protected:
    typedef typename Merge<E>::List List;
    virtual void fromA(const E &a, List &C)
    {
    } // ignore
    virtual void fromBoth(const E &a, const E &b, List &C)
    {
        C.push_back(a);
    } // add a only
    virtual void fromB(const E &b, List &C)
    {
    } // ignore
};

template <typename E> // set union
class UnionMerge : public Merge<E>
{
protected:
    typedef typename Merge<E>::List List;
    virtual void fromA(const E &a, List &C)
    {
        C.push_back(a);
    } // add a
    virtual void fromBoth(const E &a, const E &b, List &C)
    {
        C.push_back(a);
    } // add a only
    virtual void fromB(const E &b, List &C)
```

```
    {
        C.push_back(b);
    } // add b
};

template <typename E> // set subtraction
class SubtractMerge : public Merge<E>
{
protected:
    typedef typename Merge<E>::List List;
    virtual void fromA(const E &a, List &C)
    {
        C.push_back(a);
    } // add a
    virtual void fromBoth(const E &a, const E &b, List &C)
    {
    } // ignore
    virtual void fromB(const E &b, List &C)
    {
    } // ignore
};
```

main.cpp

```
#include <iostream>
#include <list>
#include "Merge.hpp"

void printSet(const std::list<int> &S)
{
    std::cout << "( ";
    for (int i : S)
    {
        std::cout << i << ' ';
    }
    std::cout << ")\n";
}

int main()
{
    std::list<int> set1({2, 3, 4, 5, 6, 7, 8});
    std::list<int> set2({6, 7, 8, 9, 10, 11, 12});
    std::list<int> set1_Union_set2;
    std::list<int> set1_Subtract_set2;
    std::list<int> set1_Intersect_set2;
    IntersectMerge<int> setIntersect;
    UnionMerge<int> setUnion;
    SubtractMerge<int> setSubtract;

    std::cout << "Set 1\n";
    printSet(set1);
    std::cout << "\nSet 2\n";
    printSet(set2);
    std::cout << "\nUnion of set 1 and set 2\n";
```

```
    setUnion.merge(set1, set2, set1_Union_set2);
    printSet(set1_Union_set2);
    std::cout << "\nSet 1 subtract Set 2\n";
    setSubtract.merge(set1, set2, set1_Subtract_set2);
    printSet(set1_Subtract_set2);
    std::cout << "\nSet 1 intersect Set 2\n";
    setIntersect.merge(set1, set2, set1_Intersect_set2);
    printSet(set1_Intersect_set2);
}
```

Output