**New Things Learnt:**

1. Reviewed Union Find alg
2. Segmented Tree, with reference to <http://community.topcoder.com/tc?module=Static&d1=tutorials&d2=lowestCommonAncestor>
3. Fenwich Tree, with reference to:<http://community.topcoder.com/tc?module=Static&d1=tutorials&d2=binaryIndexedTrees>

**Some Written Exercise:**

* 2.2.2.1: Use Binary Search Tree.
  1. Insertion and deletion cost: O(log n)
  2. Looping cost: O(n)

* 2.2.2.2: List can work; array also OK. Use insertion sort of eliminate the repeated strings

* 2.3.1.1:

Adjacency Matrix:

0 1 2 3 4 5 6 7

0 1 1 0 0 0 0 0 0

1 0 1 0 1 0 0 0 0

2 0 1 1 0 0 0 0 0

3 0 0 1 1 1 0 0 0

4 0 0 0 0 1 1 0 0

5 0 0 0 0 0 1 0 1

6 0 0 0 0 1 0 1 0

7 0 0 0 0 0 0 1 1

Adjacency List:

0: 1

1: 3

2: 1

3: 2 4

4: 5

5: 7

6: 4

7: 6

Edge List:

0 1, 1 3, 2 1, 3 2, 3 4, 4 5, 5 7, 6 4, 7 6

* 4.3.1.2:

Sample Code for three of them here, the rest are similar:

#include <iostream>

#include <vector>

#include <utility>

using namespace std;

typedef vector<int> vi;

typedef vector<vi> vii;

typedef pair<int,int> pii;

typedef pair<int,pii> pipii;

typedef vector<pii > vpii;

typedef vector<vpii> vvpii;

typedef vector<pipii> vpipii;

void trans\_AM\_to\_AL(const vii adjMat, vvpii adjList){

adjList.clear();

for(int i=0;i<adjMat.size();i++){

adjList.push\_back( \*(new vpii()) );

for(int j=0;j<adjMat.size();j++){

if(i == j) continue;

if(adjMat[i][j] > 0){

adjList[i].push\_back(make\_pair(j,adjMat[i][j]));

}

}

}

return;

}

void trans\_AL\_to\_AM(const vvpii adjList, vii adjMat){

adjMat.clear();

for(int i=0;i<adjList.size();i++){

adjMat.push\_back(\*(new vi(0,adjList.size())));

adjMat[i][i] = 1;

}

for(int i=0;i<adjList.size();i++){

for(int j=0;j<adjList.size();j++){

adjMat[i][adjList[i][j].first] = adjList[i][j].second;

}

}

return;

}

void trans\_AM\_to\_EL(const vii adjMat, vpipii edgeList){

edgeList.clear();

for(int i=0;i<adjMat.size();i++){

for(int j=0;j<adjMat.size();j++){

if(i == j) continue;

if(adjMat[i][j] > 0) edgeList.push\_back(make\_pair(i,make\_pair(j,adjMat[i][j])));

}

}

}

* 2.3.1.3: to transpose the adjList, rebuilt by making the j to be i, and i to be j.

* 2.3.2.1: Extended Union Find Alg:

#include <iostream>

#include <vector>

using namespace std;

// union find alg

// use negative number to note # of elements inside a set

vector<int> v;

int numDisjointSets;

void init(int n){

v.assign(n,-1);

numDisjointSets = n;

}

int findSet(int i){

return (v[i] < 0)?i:(v[i] = findSet(v[i]));

}

bool isSameSet(int i, int j){

return findSet(i) == findSet(j);

}

void unionSet(int i, int j){

if(i < j){

if(!isSameSet(i,j)){

v[findSet(i)] += v[findSet(j)];

numDisjointSets--;

}

v[j] = findSet(i);

} else{

if(!isSameSet(i,j)){

v[findSet(j)] += v[findSet(i)];

numDisjointSets--;

}

v[i] = findSet(j);

}

}

int elesNumInSet(int i){

return -v[findSet(i)];

}

**Anything want to contribute to the book:**

The tutorial on top-coder site is quite good. I've read through the complete discuss on Segmented Tree and Fenwick Tree, from there I can see the involving of an algorithm.