

# Codebook

Jin, Lai, Lam from YZU

September 30, 2019

## Contents

### 1 Environment

1.1	.vimrc	1
1.2	compile	1
1.3	copy	1
1.4	template	1

### 2 Data Structure

2.1	Binary Tree	1
2.2	Graph	1

### 3 Algorithm

3.1	GCD	1
3.2	LCM	1
3.3	DFS	1
3.4	BFS	1
3.5	Floyd-Warshall Algorithm	1
3.6	Dijkstra's Algorithm	1

### 4 Container

4.1	vector	1
4.2	stack	1
4.3	queue	1
4.4	priority_queue	1
4.5	set	1
4.6	map	1
4.7	list	1

### 5 C++ Library

5.1	algorithm	1
5.2	bitset	1
5.3	cmath	1

5.4	iomanip	1
5.5	cstdio	1

### 6 Note

6.1	Preparing	1
6.2	Response Message	1

## 1 Environment

### 1.1 .vimrc

```
1 set number
1 2 set mouse=a
1 3 set shiftwidth=4
1 4 set tabstop=4
1 5 set autoindent
1 6 set cindent
1 7 filetype indent on
1 8 set cursorline
1 9 set t_Co=256
1 10 colorscheme slate
1 11 syntax on
```

### 1.2 compile

```
1 #shell script to compile program and
2 execute
2 2 #!/bin/bash
2 3 g++ -Wall -O2 -std=c++14 -static -pipe -o
2 $1 $1.cpp && ./ $1 < $1.in > $1.out | cat
2 ./ $1.out
```

### 1.3 copy

```
1 #copy template file
2 #!/bin/bash
3 for name in {A..M};
3 do
3 5 cp template.cpp $name.cpp
3 6 done
```

### 1.4 template

```
1 //template to code in C++
2 #include <bits/stdc++.h>
3 typedef unsigned long long ull;
4 4 typedef long long ll;
4 5 using namespace std;
4 6
4 7 int main(){
8
9     return 0;
10 }
```

## 2 Data Structure

### 2.1 Binary Tree

```
1 //Binary Tree (array)
2 Array[]
3 rootNode = Array[0]
4 fatherNode = p
5 leftChildNode = Array[2 * p] + 1
6 rightChildNode = Array[2 * p] + 2
```

### 2.2 Graph

```
1 //Graph (adjacent matrix)
2 matrix[row][col]
3 distance[row][col]
4 visited[row][col]
5 m = row_i, n = col_j
```

## 3 Algorithm

### 3.1 GCD

```
1 int GCD(int a, int b){
2     if(b == 0)
3         return a;
4     return GCD(b, a%b);
5 }
```

### 3.2 LCM

```
1 int LCM(int a, int b){
2     return a / GCD(b, a%b) * b;
3 }
```

### 3.3 DFS

```

1 void DFS(){
2   Graph[][]
3   visited[][] = {}
4   FirstNode
5   stack S
6   S.push(FirstNode)
7   while(!S.empty){
8     currentNode = S.pop()
9     if(currentNode == targetNode)break //
10    find target
11    if(!visited[currentNode]){
12      visited[currentNode] = true
13      for(all nextNode){
14        if(nextNode && !visited[nextNode])
15          S.push(nextNode)
16      }
17    }
18  }

```

### 3.4 BFS

```

1 void BFS(){
2   Graph[][]
3   visited[][] = {}
4   FirstNode
5   queue Q
6   Q.push(FirstNode)
7   while(!Q.empty){
8     currentNode = Q.pop()
9     if(currentNode == targetNode)break //
10    find target
11    if(!visited[currentNode]){
12      visited[currentNode] = true
13      for(all nextNode){
14        if(nextNode && !visited[nextNode])
15          Q.push(nextNode)
16      }
17    }
18  }

```

### 3.5 Floyd-Warshall Algorithm

```

1 void Floyd_Warshall(){
2   INF
3   int Graph[][] //edge length
4
5   for(all i, j)
6     if(i == j)
7       Graph[i][j] = 0
8   else
9     Graph[i][j] = INF
10  read Graph
11  for(all i, j, k)
12    Graph[i][j] = min(Graph[i][j], Graph[i][k] + Graph[k][j])
13
14  print Graph[x][y] //get shortest path
15  from x to y

```

### 3.6 Dijkstra's Algorithm

```

1 void Floyd_Warshall(){
2   INF
3   int Graph[][] //edge length
4   int distance[]
5   bool visit[]
6
7   for(all i, j)
8     if(i == j)
9       Graph[i][j] = 0
10  else
11    Graph[i][j] = INF
12  read Graph
13  read keypoint
14  for(all i)
15    distance[i] = e[keypoint][i];
16
17  visit[keypoint] = true
18  for(all i){
19    minimum = INF
20    int u
21    for(all j){
22      if(!visit[j] && distance[j] < min){

```

```

23      min = distance[j];
24      u = j
25    }
26  }
27  visit[u] = true;
28  for(all v){
29    if(Graph[u][v] < INF && distance[v] >
30      distance[u] + Graph[u][v])
31      distance[v] = distance[u] + Graph[u][v]
32  }
33
34  print distance[x] //get shortest path
35  from keypoint to x

```

## 4 Container

### 4.1 vector

```

1 //template
2 template <class value_type>
3 //init
4 vector <value_type>
5 //iterator
6 iterator begin()
7 iterator end()
8 //capacity
9 size_type size()
10 void reserve(size_type)
11 bool empty()
12 //access
13 reference operator[](size_type)
14 reference at(size_type)
15 //modifiers
16 void push_back(value_type)
17 void pop_back()
18 iterator insert(const_iterator, value_type)
19 iterator erase(const_iterator)

```

### 4.2 stack

```

1 //template
2 template <class value_type>
3 //init
4 stack <value_type>
5 //capacity
6 size_type size()
7 bool empty()
8 //access
9 reference top()
10 //modifiers
11 void push(value_type)
12 void pop()

```

### 4.3 queue

```

1 //template
2 template <class value_type>
3 //init
4 queue <value_type>
5 //capacity
6 size_type size()
7 bool empty()
8 //access
9 reference front()
10 reference back()
11 //modifiers
12 void push(value_type)
13 void pop()

```

### 4.4 priority\_queue

```

1 //template
2 template <class value_type>
3 //init
4 priority_queue <value_type> //priority
    larger
5 priority_queue <value_type, vector<
    value_type>, greater<value_type> > //
    priority smaller
6 //capacity
7 size_type size()
8 bool empty()
9 //access
10 reference top()
11 //modifiers

```

```

12 void push(value_type)
13 void pop()

```

### 4.5 set

```

1 //template
2 template <class value_type>
3 //init
4 set <value_type>
5 //iterator
6 iterator begin()
7 iterator end()
8 //capacity
9 size_type size()
10 bool empty()
11 //operations
12 iterator find(value_type)
13 size_type count(value_type)
14 //modifiers
15 pair<iterator, bool> insert(value_type)
16 size_type erase(value_type)

```

### 4.6 map

```

1 //template
2 template <class key_type, class mapped_type>
3 typedef pair<key_type, mapped_type>
    value_type
4 //init
5 map <key_type, mapped_type>
6 //iterator
7 iterator begin()
8 iterator end()
9 //capacity
10 size_type size()
11 bool empty()
12 //access
13 mapped_type& operator[](key_type)
14 map<key_type, mapped_type>::iterator->first
    //key value
15 map<key_type, mapped_type>::iterator->
    second // mapped value
16 //operations
17 iterator find(key_type)

```

```

18 size_type count(key_type)
19 //modifiers
20 pair<iterator, bool> insert(pair<key_type,
    mapped_type>(key_type, mapped_type))
21 size_type erase(key_type)

```

### 4.7 list

```

1 //template
2 template <class value_type>
3 //init
4 list <value_type>
5 //iterator
6 iterator begin()
7 iterator end()
8 //capacity
9 size_type size()
10 void reserve(size_type)
11 bool empty()
12 //access
13 reference front(size_type)
14 reference back(size_type)
15 //operations
16 void remove(value_type)
17 //modifiers
18 void push_front(value_type)
19 void pop_front()
20 void push_back(value_type)
21 void pop_back()
22 iterator insert(const_iterator, value_type)
23 iterator erase(const_iterator)

```

## 5 C++ Library

### 5.1 algorithm

```

1 template <class InputIterator, class
    value_type>
2 InputIterator find(InputIterator first,
    InputIterator last, value_type val)
3
4 template <class RandomAccessIterator>
5 void sort(RandomAccessIterator first,
    RandomAccessIterator last)

```

```

6
7 template <class RandomAccessIterator, class
    Compare>
8 void sort(RandomAccessIterator first,
    RandomAccessIterator last, Compare comp)
9
10 template <class ForwardIterator, class
    value_type>
11 bool binary_search(ForwardIterator first,
    ForwardIterator last, value_type val)
12
13 template <class BidirectionalIterator>
14 bool next_permutation(BidirectionalIterator
    first, BidirectionalIterator last);

```

## 5.2 bitset

```

1 //template
2 template <class size_t>
3 //init
4 bitset <size_t>(unsigned long long)
5 bitset <size_t>(string)
6 bitset <size_t>(char *)
7 //access
8 bool operator[](size_t) const
9 reference operator[](size_t)
10 size_t count() // return the number of 1
11 size_t size() // size()-count() = return
    the number of 0
12 bool any()
13 bool none()
14 //operations
15 reference set() //all
16 reference set(size_t, bool) //single
17 reference reset() //all
18 reference reset(size_t) //single
19 string to_string()
20 unsigned long to_ulong()
21 unsigned long long to_ullong()

```

## 5.3 cmath

```

1 double cos(double)
2 double acos(double) //PI = acos(0.0)*2.0
3 double exp(double) //exponential

```

```

4 double log(double)
5 double log10(double)
6 double log2(double)
7 double pow(double, double)
8 double sqrt(double)
9 double cbrt(double)
10 double ceil(double) //round up
11 double floor(double) //round down
12 double round(double) //round
13 double abs(double)

```

## 5.4 iomanip

```

1 setfill(char_type)
2
3 setprecision(int)
4
5 setw(int)
6
7 setbase(int) //10, 8, 16

```

## 5.5 cstdio

```

1 int printf(char *format, ...)
2 int sprintf(char *str, char *format, ...)
3 int scanf(char *format, ...)
4 int sscanf(char *str, char *format, ...)
5
6 /*
7     format
8
9     print : %[flags][width][.precision][
        length]specifier
10    scan  : %[*][width][length]specifier
11
12    specifier:
13    %c : character
14    %s : string of characters
15    %d : signed decimal
16    %u : unsigned decimal
17    %o : unsigned octal
18    %x : unsigned hexadecimal
19    %X : unsigned hexadecimal (upper)
20    %% : %
21 */

```

## 6 Note

### 6.1 Preparing

```

1 check keyboard
2 check mouse
3 build environment(vim, g++, shell)
4 check judge system
5 check response message

```

### 6.2 Response Message

```

1 //for DOMjudge
2 CORRECT
3 COMPILER-ERROR
4 TIMELIMIT
5 RUN-ERROR
6 WRONG-ANSWER

```