

# Codebook

Jin, Lai, Lam from YZU

October 12, 2019

## Contents

### 1 Environment

1.1	.vimrc	1
1.2	compile	2
1.3	copy	3
1.4	template	4

### 2 Data Structure

2.1	Binary Tree	2
2.2	Graph	2

### 3 Algorithm

3.1	GCD	3
3.2	LCM	3
3.3	BFS	3
3.4	DFS	3
3.5	Floyd-Warshall Algorithm	3
3.6	Dijkstra's Algorithm	3
3.7	Infix, Postfix, Prefix	3
3.8	Knapsack Problem	3

### 4 Container

4.1	vector	4
4.2	stack	4
4.3	queue	4
4.4	priority_queue	4
4.5	set	4
4.6	map	4
4.7	list	4

### 5 C++ Library

5.1	algorithm	5
5.2	bitset	5
5.3	string	5
5.4	cmath	5
5.5	io manip	5
5.6	cstdio	5

### 6 Note

6.1	Preparing	6
6.2	Response Message	6

## 1 Environment

### 1.1 .vimrc

```
1
1
1 set number
1 set mouse=a
1 set shiftwidth=4
4 set tabstop=4
1 set autoindent
1 set cindent
1 filetype indent on
8 set cursorline
1 set t_Co=256
1 colorscheme slate
1 syntax on
```

### 1.2 compile

```
2
2
2 #shell script to compile program and
2 execute
2 #!/bin/bash
2 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

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

```
4 done
```

### 1.4 template

```
4
4
5 //template to code in C++
5 #include <bits/stdc++.h>
5 typedef unsigned long long ull;
4 typedef long long ll;
5 using namespace std;
5
5
5 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) return a;
3 return GCD(b, a%b);
4 }
```

### 3.2 LCM

```

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

```

### 3.3 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 }

```

### 3.4 DFS

```

1 //Stack
2 Change BFS queue to stack

```

### 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]
13   ][k] + Graph[k][j])
14
15  print Graph[x][y] //get shortest path
16  from x to y
17 }

```

### 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]
32        ][v]
33    }
34  }
35 }

```

```

34  print distance[x] //get shortest path
35  from keypoint to x
36 }

```

### 3.7 Infix, Postfix, Prefix

```

1 //equation from infix to postfix
2 void convertInfixToPostfix(){
3   setOperatorPriority() //0 is the largest
4   stack op
5   char input[]
6   char output[]
7   int index
8   for(all i in input){
9     if(input[i] == NUMBER){
10      output[index++] = input[i]
11    }
12    else if(input[i] == '(')
13      op.push(input[i])
14    else if(input[i] == ')'){
15      while(op.top() != '('){
16        output[index++] = op.pop()
17      }
18      op.pop();
19    }
20    else if(input[i] == OPERATOR){
21      if(op.empty()){
22        op.push(input[i])
23      }
24      else{
25        while(Priority[op.top()] < Priority
26        [input[i]]) //op.top >= input[i]
27        {
28          output[index++] = op.pop()
29        }
30        op.push(input[i])
31      }
32    }
33  }
34  while(!op.empty())
35    output[index++] = op.pop()
36 }

```

### 3.8 Knapsack Problem

```

1 /*
2  0/1 Knapsack Problem
3  recursive function : c(n, w) = max(c(n
4    -1, w), c(n-1, w-weight[n] + cost[n]))
5  c(n, w) : knapsack problem answer
6  n : from item 0_th to n_th
7  w : max_weight
8  weight[n] : weight of item n
9  cost[n] : cost of item n
10 */
11 //bottom up
12 void knapsack(int n, int w){
13     memset(c, 0, sizeof(c))
14     for(all i in n) //all item
15         for(all j in w) //all weight
16             if(j - weight[i] < 0)
17                 c[i+1][j] = c[i][j]
18             else
19                 c[i+1][j] = max(c[i+1][j], j-weight
20 [i]+cost[i])
21 print c[n][w] //the highest value
22 }
23 /*
24 Coin Change Problem
25 recursive function : c(n, m) = c(n-1, m)
26   + c(n-1, m-price[n])
27 c(n, m) : coin change problem answer
28 n : from coin 0_th to n_th
29 m : target money
30 price[n] : coin price
31 */
32 //bottom up
33 void change(int m){
34     memset(c, 0, sizeof(c))
35     c[0] = 1;
36     for(all i in n) //all coin
37         for(all j from price[i] to m) //all
38             target money
39             c[j] += c[j-price[i]]
40 print m //target money
41 print c[m] //kinds
42 }

```

## 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 )
20 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
5   larger
6 priority_queue <value_type, vector<
7   value_type>, greater<value_type> > //
8   priority smaller
9 //capacity
10 size_type size()
11 bool empty()
12 //access
13 reference top()
14 //modifiers
15 void push(value_type)
16 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 >
4 typedef pair<key_type, mapped_type>
5 value_type
6 //init
7 map <key_type, mapped_type>
8 //iterator
9 iterator begin()
10 iterator end()
11 //capacity
12 size_type size()
13 bool empty()
14 //access
15 mapped_type& operator[](key_type)
16 map<key_type, mapped_type>::iterator->first
17 //key value
18 map<key_type, mapped_type>::iterator->
19 second // mapped value
20 //operations
21 iterator find(key_type)
22 size_type count(key_type)
23 //modifiers
24 pair<iterator, bool> insert(pair<key_type,
25 mapped_type>(key_type, mapped_type))
26 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
2 value_type>
3 InputIterator find(InputIterator first,
4 InputIterator last, value_type val)
5
6 template <class RandomAccessIterator>
7 void sort(RandomAccessIterator first,
8 RandomAccessIterator last)
9
10 template <class RandomAccessIterator, class
11 Compare>
12 void sort(RandomAccessIterator first,
13 RandomAccessIterator last, Compare comp)
14
15 template <class ForwardIterator, class
16 value_type>
17 bool binary_search(ForwardIterator first,
18 ForwardIterator last, value_type val)
19
20 template <class BidirectionalIterator>
21 bool next_permutation(BidirectionalIterator
22 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
12 the number of 0
13 bool any()
14 bool none()
15 //operations
16 reference set() //all
17 reference set(size_t, bool) //single
18 reference reset() //all
19 reference reset(size_t) //single
20 string to_string()
21 unsigned long to_ulong()
22 unsigned long long to_ullong()

```

## 5.3 string

```

1 //init
2 string
3 //iterator
4 iterator begin()
5 iterator end()
6 //capacity
7 size_type size()
8 void reserve(size_type)
9 bool empty()
10 //access
11 reference operator[](size_type)
12 reference at(size_type)
13 //modifiers
14 string operator+= (string)
15 string insert(pos, string)
16 string erase(pos = 0, len)
17 //operation
18 string substr(pos = 0, len)
19 //function

```

```

20 string to_string(val)
21 //stringstream
22 string str()

```

## 5.4 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)

```

## 5.5 iomanip

```

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

```

## 5.6 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][
10         length]specifier
11  scan  : %[*][width][length]specifier
12
13  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

```