

Codebook

Jin, Lai, Lam from YZU

November 22, 2019

Contents

1 Environment

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

2 Data Structure

- 2.1 Binary Tree
- 2.2 Graph

3 Algorithm

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

4 Container

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

5 C++ Library

- 5.1 algorithm
- 5.2 bitset
- 5.3 string
- 5.4 cmath
- 5.5 iomanip
- 5.6 cstdio

6 Note

- 6.1 Preparing
- 6.2 Response Message

1 Environment

1.1 .vimrc

```
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
2 syntax on
```

1.2 compile

```
2 # shell script to compile program and
2 execute
2 # sh compile.sh $filename
2 #!/bin/bash
4 g++ -Wall -O2 -std=c++17 -static -pipe -o
3 $1 $1.cpp && ./$1 < $1.in > $1.out
3 wait
3 cat ./$1.out
```

1.3 copy

```
4 # copy template file
4 #!/bin/bash
4 for name in {A..M};
```

```
4 do
4 cp template.cpp $name.cpp
4 done
```

1.4 template

```
5 // template to code in C++
2 #include <bits/stdc++.h>
5 typedef unsigned long long ull;
5 typedef long long ll;
5 using namespace std;
6
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) 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(Graph, visited, FirstNode){
2     queue Q
3     Q.push(FirstNode)
4     while(!Q.empty()){
5         currentNode = Q.pop()
6         if(currentNode == targetNode) break // find target
7         if(!visited[currentNode]){
8             visited[currentNode] = true
9             for(all nextNode){
10                 if(nextNode && !visited[nextNode])
11                     Q.push(nextNode)
12             }
13         }
14     }
15 }
```

3.4 DFS

```
1 //Stack
2 Change BFS queue to stack
```

3.5 Floyd-Warshall Algorithm

```
1 #define INF 0xFFFFFFFF
2 void Floyd_Warshall(int Graph //edge length
3 ){
4     for(all i, j)
5         if(i == j)
6             Graph[i][j] = 0
7         else
8             Graph[i][j] = INF
9     read Graph
10    for(all i, j, k)
11        Graph[i][j] = min(Graph[i][j], Graph[i][k] + Graph[k][j])
```

```
12    print Graph[x][y] //get shortest path
13    from x to y
14 }
```

3.6 Dijkstra's Algorithm

```
1 #define INF 0xFFFFFFFF
2 void Floyd_Warshall(int Graph[][], /*edge
3     length*/, visit[][]){
4     int distance[]
5
6     for(all i, j)
7         if(i == j)
8             Graph[i][j] = 0
9         else
10            Graph[i][j] = INF
11    read Graph
12    read keypoint
13    for(all i)
14        distance[i] = Graph[keypoint][i];
15
16    visit[keypoint] = true
17    for(all i){
18        min = INF
19        int u
20        for(all j){
21            if(!visit[j] && distance[j] < min){
22                min = distance[j]
23                u = j
24            }
25        }
26        visit[u] = true;
27        for(all v){
28            if(Graph[u][v] < INF && distance[v] >
29                distance[u] + Graph[u][v])
30                distance[v] = distance[u] + Graph[u][v]
31        }
32    }
33
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(char input[]){
3     setOperatorPriority() //0 is the largest
4     stack op
5     char output[]
6     int index
7     for(all i in input){
8         if(input[i] == NUMBER){
9             output[index++] = input[i]
10        }
11        else if(input[i] == '(')
12            op.push(input[i])
13        else if(input[i] == ')'){
14            while(op.top() != '('){
15                output[index++] = op.pop()
16            }
17            op.pop();
18        }
19        else if(input[i] == OPERATOR){
20            if(op.empty()){
21                op.push(input[i])
22            }
23            else{
24                while(Priority[op.top()] < Priority
25                    [input[i]] //op.top >= input[i]
26                {
27                    output[index++] = op.pop()
28                }
29                op.push(input[i])
30            }
31        }
32    }
33    while(!op.empty())
34        output[index++] = op.pop()
35 }
```

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]))
```

```

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

```

```

45 target)
46 Third loop(only appear in item limit
47 case) = max(number amount, now value/
48 this value)
49
50 backpack structure:
51 Struct {weight, cost}
52
53 w-weight[n] meaning I push this I item
54 n-1 meaning look forward
55
56 Code:
57 c[i] = max (c[i], c[i - weight[n]] + cost[
58 n]) - consider value
59 c[i] = max (c[i], c[i - weight[n]] + 1) -
60 consider amount of item
61 way[j] += way[j - weight[i]] - consider
62 ways
63
64 Coin (like as backpack):
65 Code:
66 c[j] += c[j-price[i]]; << ways
67 c[j] = min(c[j], c[j-price[i]] + 1); -
68 min amount of coin
69 */

```

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

```

```

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 // oprations
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)
17
18 // multiset
19 size_type count(value_type) //return the
    number of element

```

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 //oprations
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 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 //opeartion
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:
14  %c : character
15  %s : string of characters
16  %d : signed decimal
17  %u : unsigned decimal
18  %o : unsigned octal
19  %x : unsigned hexadecimal
20  %X : unsigned hexadecimal (upper)
21  %% : %
22 */

```

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

```