# Codebook

Jin, Lai, Lam from YZU

September 27, 2020

## Contents



# 1 Environment

## 1.1 .vimrc

```
set nu " set number
set mouse=a " set mouse=a
set sw=4 " set shiftwidth=4
set st=4 " set tabstop=4
set ai " set autoindent
set ci " set cindent
set cul " set cursorline
set t_Co=256
filetype indent on
colorscheme slate
syntax on
```

## 1.2 compile

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

## 1.3 copy

```
# copy template file
#!bin/bash
for name in {A..M};
do
  cp template.cpp $name.cpp
  echo 0 > $name.in
  echo 0 > $name.out
done
```

## 1.4 template

```
//  template to code in C++
#include <bits/stdc++.h>
using namespace std;

int main(){

  return 0;
}
```

# 2 Structure

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

//Graph (adjacent matrix)
matrix[row][col]
distance[row][col]
visited[row][col]
m = row_i, n = col_j
```

# 3 Code Note

```
int GCD(int a, int b){
  return b == 0 ? a : GCD(b, a % b);
}

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

// Find shortest path
// Queue
void BFS(Graph, visited, FirstNode){
  queue Q
  Q.push(FirstNode)
  while(!Q.empty){
    currentNode = Q.pop()
    if(currentNode == targetNode)break // find target
    if(!visited[currentNode]){
      visited[currentNode] = true
      for(all nextNode){
        if(!visited[nextNode])
          Q.push(nextNode)
      }
    }
  }
}

// Find connectivity
// Stack
// Change BFS queue to stack
```

```
#define INF 0xFFFFFFFF
void FloydWarshall(Graph){
  for i = 0 to size
    for j = 0 to size
      for k = 0 to size
        Graph[i][j] = min(Graph[i][j], Graph[i][k] + Graph[k][j]);
  print Graph[x][y] //get shortest path from x to y
}

bool NeagtiveCycle(){
  for(int i = 0; i < V; ++i){
    if(dist[i][i] < 0)
      return ture;
  }
  return false;
}

// Dijkstra
struct node{
  int id,len;
  node(int n, int weight) : id(n), len(weight){};
  bool operator<(const node &right) const{ return id > right.id;}
};

// Using a priority queue
void Dijkstra(){
  best[E] = 0;
  que.push(node(E, 0));
  while(!que.empty()){
    node cur = que.top();
    que.pop();
    for(int i = 0; i < num[cur.id]; ++i){
      if(best[path[cur.id][i]] > cur.len + w[cur.id][path[cur.id][i]]){
        best[path[cur.id][i]] = cur.len + w[cur.id][path[cur.id][i]];
        que.push(node(path[cur.id][i], best[path[cur.id][i]]));
      }
    }
  }
}

void BellmanFord(){
  e = edge.size();
  fill(best, best+e, NIL);
  best[0] = 0;
  for(int i = 0; i < n; ++i)
    for(int j = 0; j < e; ++j)
      if(best[edge[j].to] > best[edge[j].from] + edge[j].weight)
        best[edge[j].to] = best[edge[j].from] + edge[j].weight;
  for(int j = 0; j < e; ++j)
    if(best[edge[j].to] > best[edge[j].from] + edge[j].weight){
      indefinitely = false;
      break;
    }
}

int CeilIndex(int A[], int tail[], int low, int high, int key){
  while(high-low > 1){
    int mid = (high + low) / 2;
    if(A[tail[mid]] >= key)
      high = mid;
    else
      low = mid;
  }
  return high;
}

int LongestIncreasingSubsequence(int A[], int n){
  if(n == 0) return 0;
  int *tail = new int[n+1];
  int *prev = new int[n+1];

  int length = 1;
  tail[1] = 1;
  for(int i = 2; i <= n; ++i){
    if(A[i] < A[tail[1]])
      tail[1] = i;
    else if(A[i] > A[tail[length]]){
      prev[i] = tail[length];
      tail[++length] = i;
    }
    else{
      int position = CeilIndex(A, tail, 1, length, A[i]);
      prev[i] = tail[position-1];
      tail[position] = i;
    }
  }
}

int max1DRangeSum(int column[]{
  int globalMax = column[1];
  int localMax = column[1];
  for(int i = 2; i <= m; i++){
    localMax = max(column[i], localMax + column[i]);
    if(globalMax < localMax)
      globalMax = localMax;
  }
  return globalMax;
}

int max2DRangeSum(int A[][n+1]){
  for(int l = 1; l < n; ++left){
    memset(rowSum, 0, sizeof(rowSum));
    for(int r = left; r <= n; ++right){
      for(int i = 1; i <= m; ++i)
        rowSum[i] += A[i][r];
      localMax = max1DRangeSum(rowSum);
      if(globalMax < localMax)
        globalMax = localMax;
    }
  }
}

int find(int a){
  return a = (p[a] == a) ? a : (p[a] = find (p[a]));
}

void Union(int a, int b){
```

```
  p[find(a)] = find(b);
}
```

# 4 Algorithm Note

**Algorithm 1:** ArticulationPoints($G$)

```
foreach vertex u ∈ G.V do
    u.cut = false
end
foreach vertex u ∈ G.V do
    if u.π == NIL then
        if u.numChildren > 1 then
            u.cut = true
    else
        foreach v ∈ G.Adj[u] do
            if v.π == u then
                if v.low >= u.d then
                    u.cut = true
        end
    end
end
```

**Algorithm 2:** Biconnect($G$)

```
time = time + 1
u.d = time
u.low = time
foreach v ∈ G.Adj[u] do
    if v.d == 0 then
        v.π = u
        Push((u, v), S)
        Biconnect(G, v)
        u.low = min(u.low, v.low)
        if v.low >= u.d then
            start new component
            do
                (x1, x2) = Pop(S)
                put (x1, x2) in current component
            while (x1, x2! = (u, v)andx1.d >= v.d);
        end
    else if v! = u.π then
        Push((u, v), S)
        u.low = min(u.low, v.d)
    end
end
```

**Algorithm 3:** TopologicalSort($G$)

```
foreach vertex uinG.V do
    u.color = WHITE
end
foreach vertex uinG.V do
    if u.color = WHITE then
        DFS_Visit(G, u)
    end
end
```

**Algorithm 4:** DFS_Visit($G, u$)

```
u.color = GRAY
foreach v ∈ G.Adj[u] do
    if v.color == WHITE then
        DFS_Visit(G, u)
    end
end
u.color = BLACK insert u onto the front of a linked
  list
```

# 5 C++ Library

```
#include <bits/stdc++.h>

// algorithm (c++)
template <class InputIterator, class
    value_type>
InputIterator find(InputIterator first,
    InputIterator last, value_type val)

template <class RandomAccessIterator>
void sort(RandomAccessIterator first,
    RandomAccessIterator last)

template <class RandomAccessIterator, class
     Compare>
void sort(RandomAccessIterator first,
    RandomAccessIterator last, Compare comp)

template <class ForwardIterator, class
    value_type>
bool binary_search(ForwardIterator first,
    ForwardIterator last, value_type val)

template <class BidirectionalIterator>
bool next_permutation(BidirectionalIterator
     first, BidirectionalIterator last)

// cmath
double cos(double)
double acos(double) //PI = acos(0.0)*2.0
double exp(double) //exponential
double log(double)
double log10(double)
double log2(double)
double pow(double, double)
double sqrt(double)
double cbrt(double)
double ceil(double) //round up
double floor(double) //round down
double round(double) //round
double abs(double)
```

```
// cstdio
int printf(char *format, ...)
int sprintf(char *str, char *format, ...)
int scanf(char *format, ...)
int sscanf(char *str, char *format, ...)

/*
  format

  print : %[flags][width][.precision][
    length]specifier
  scan  : %[*][width][length]specifier

  specifier:
  %c  : character
  %s  : string of characters
  %d  : signed decimal
  %u  : unsigned decimal
  %o  : unsigned octal
  %x  : unsigned hexadecimal
  %X  : unsigned hexadecimal (upper)
  %%  : %
*/

// iomanip
setfill(char_type)
setprecision(int)
setw(int)
setbase(int) //10, 8, 16

// STL
// bitset
template <class size_t>
bitset<size_t>(unsigned long long)
bitset<size_t>(string)
bitset<size_t>(char *)
bool operator[](size_t) const
ref operator[](size_t)
size_t count() // return the number of 1
size_t size() // size()-count() = the
    number of 0
bool any()
bool none()
ref set() // all
ref set(size_t, bool) // single
ref reset() // all
ref reset(size_t) // single
string to_string()
unsigned long to_ulong()
unsigned long long to_ullong()

// list
template <class value_type>
list <value_type>
iterator begin()
iterator end()
size_type size()
void reserve(size_type)
bool empty()
ref front(size_type)
ref back(size_type)
void remove(value_type)
void push_front(value_type)
void pop_front()
void push_back(value_type)
void pop_back()
iterator insert(const_interator, value_type
    )
iterator erase(const_interator)

// map
template <class key_type, class value_type>
typedef pair<key_type, value_type>
    instance_type
map <key_type, value_type>
iterator begin()
iterator end()
size_type size()
bool empty()
value_type& operator[](key_type)
map<key_type, value_type>::iterator->first
    //key value
map<key_type, value_type>::iterator->second
     // mapped value
iterator find(key_type)
size_type count(key_type)
pair<iterator, bool> insert(pair<key_type,
    value_type>(key_type, value_type))
size_type erase(key_type)

// priority_queue
template <class value_type>
priority_queue <value_type> //priority
    larger
priority_queue <value_type, vector<
    value_type>, greater<value_type> > //
    priority smaller
size_t size()
bool empty()
ref top()
void push(value_type)
void pop()

// queue
template <class value_type>
queue <value_type>
size_type size()
bool empty()
reference front()
reference back()
void push(value_type)
void pop()

// set
template <class value_type>
set <value_type>
iterator begin()
iterator end()
size_type size()
bool empty()
iterator find(value_type)
size_type count(value_type)
pair<iterator, bool> insert(value_type)
size_type erase(value_type)
size_type count(value_type) //return the
    number of element

// stack
template <class value_type>
stack <value_type>
size_type size()
bool empty()
```

```
reference top()
void push(value_type)
void pop()

// string
string
iterator begin()
iterator end()
size_type size()
void reserve(size_type)
bool empty()
reference operator[](size_type)
reference at(size_type)
string operator+= (string)
string insert(pos, string)
string erase(pos = 0, len)
string substr(pos = 0, len)
string to_string(value) // c++11
string str() // stringstream

// vector
template <class value_type>
vector <value_type>
iterator begin()
iterator end()
size_type size()
void reserve(size_type)
bool empty()
reference operator[](size_type)
reference at(size_type)
void push_back(value_type)
void pop_back()
iterator insert(const_interator, value_type
    )
iterator erase(const_interator)
```

# 6 Other Tool

## 6.1 gdb

```
l (list)
b (breakpoint)
r (run)
p $value (print $value)
c (continue)
q quit
step
display $value
info
```

## 6.2 vim

# 7 Note

## 7.1 Preparing

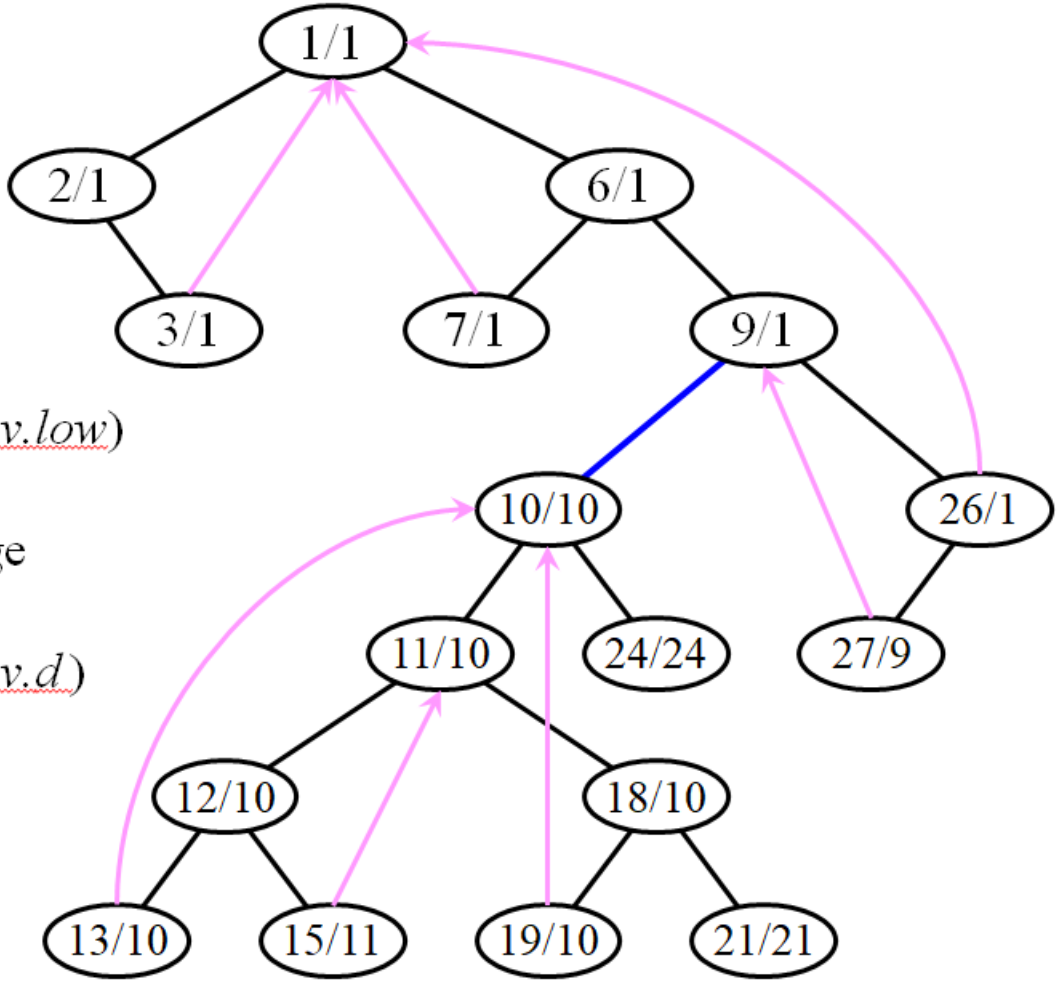
```
check keyboard
check mouse
check printer
check judge system
check response message
build environment(vim, g++, shell)
```

## 7.2 Response Message

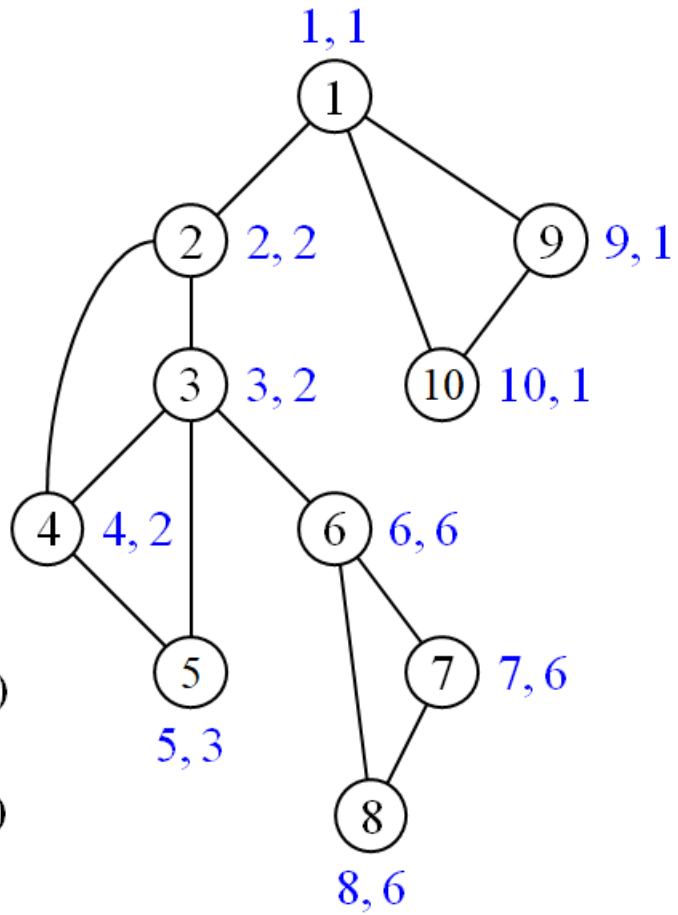
```
//for DOMjudge
CORRECT
COMPILER-ERROR
TIMELIMIT
RUN-ERROR
WRONG-ANSWER
```

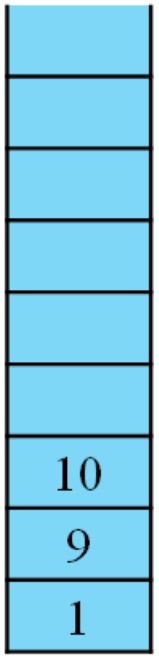
## 8 Image Note

```
BRIDGE (G, u)
 1   time = time + 1
 2   u.d = time
 3   u.low = time
 4   for each v ∈ G.Adj[u]
 5       if v.d == 0
 6           v.π = u
 7           BRIDGE (G, v)
 8           u.low = min (u.low, v.low)
 9           if v.low > u.d
10               {u , v} is a bridge
11       else if v != u.π
12           u.low = min (u.low, v.d)
```



```
BRIDGECONNECT(G, u)
 1   time = time + 1
 2   u.d = time
 3   u.low = time
 4   PUSH(u, S)
 5   for each v ∈ G.Adj[u]
 6       if v.d == 0
 7           v.π = u
 8           BRIDGECONNECT(G, v)
 9           u.low = min(u.low, v.low)
10       else if v != u.π
11           u.low = min(u.low, v.d)
12   if u.low == u.d
13      start new component
14      do
15          w = POP(S)
16          put w in current component
17      while w != u
```




*S*

*Components*

1: 8, 7, 6
2: 5, 4, 3, 2
3: 10, 9, 1

```
class point {
public:
    double x, y;
    point(double corX = 0.0, double corY = 0.0) :x(corX), y(corY) {};
    point & operator = (const point &left)
    {
        x = left.x;
        y = left.y;
        return *this;
    }
};
point p[N];

bool cmpX(const point &left, const point &right)
{
    return left.x < right.x;
}

double dist(const point &left, const point &right)
{
    return sqrt((left.x - right.x)*(left.x - right.x) + (left.y - right.y)*(left.y - right.y));
}

double combine(const int &left, const int &right, const int mid, const double &midL, const double &midR)
{
    double d = min(midL, midR);
    //double line =  + d;
    double min_temp = d;
    for (int i = mid;(p[mid].x -  p[i].x)<= d && i >= left; --i)
        for (int j = mid + 1; (p[j].x - p[mid].x) <= d && j <= right; ++j)
        {
            min_temp = min(min_temp, dist(p[i], p[j]));
        }
    return min_temp;

}

double divide(const int &left,const int &right)
{
    if (left >= right)
        return INF;

    int mid = (left + right) / 2;
    double midL = divide(left, mid);
    double midR = divide(mid + 1, right);

    return combine(left, right, mid, midL, midR);
}



double closePair(const int &ptNum)
{
    sort(p, p + ptNum, cmpX);
    return divide(0, ptNum - 1);
}
```

```
struct point
{
    double x, y, d;
    point & operator= (const point &left) { x = left.x; y = left.y; d = left.d
};

point p[N], st[N];

double ans;

double cross(const point &O, const point &A, const point &B)
{
    return (A.x - O.x)*(B.y - O.y) - (A.y - O.y)*(B.x - O.x);
}

bool cmp1(const point &left, const point &right)
{
    return left.y < right.y || (left.y == right.y && left.x < right.x);
}

bool cmp2(const point &A, const point &B)
{
    double cp = cross(p[0], A, B);
    if (cp == 0) return A.d < B.d;
    return cp > 0;
}

double dist(const point &A, const point &B)
{
    return sqrt((A.x - B.x)*(A.x - B.x) + (A.y - B.y)*(A.y - B.y));
}

void convexhall(const int &ptNum)
{
    ans = 0;
    if (ptNum > 1)
    {
        sort(p, p + ptNum, cmp1);
        for (int i = 0; i < ptNum; ++i)
            p[i].d = dist(p[0], p[i]);

        sort(p + 1, p + ptNum, cmp2);
    }

    int stNum = 0;
    for (int i = 0; i < ptNum; ++i)
    {
        while (stNum > 1 && cross(st[stNum - 2], st[stNum - 1], p[i]) <= 0)
            stNum--;
        st[stNum++] = p[i];
    }

    st[stNum++] = p[0];
    for (int i = 0; i < stNum; ++i)
    {
        printf("(%.5f,%.5f)", st[i].x, st[i].y);
        if(i + 1 != stNum)
            printf(" ");
    }
    printf("\n");
}
```

```
bool bfs(int rGraph[V][V], int s, int t, int parent[])
{
    bool visited[V];
    memset(visited, 0, sizeof(visited));

    queue <int> q;
    q.push(s);
    visited[s] = true;
    parent[s] = -1;

    // Standard BFS Loop
    while (!q.empty())
    {
        int u = q.front();
        q.pop();

        for (int v=0; v<V; v++)
        {
            if (visited[v]==false && rGraph[u][v] > 0)
            {
                q.push(v);
                parent[v] = u;
                visited[v] = true;
            }
        }
    }

    return (visited[t] == true);
}
```

```
int fordFulkerson(int graph[V][V], int s, int t)
{
    int u, v;

    int rGraph[V][V];
    for (u = 0; u < V; u++)
        for (v = 0; v < V; v++)
            rGraph[u][v] = graph[u][v];

    int parent[V];

    int max_flow = 0;

    while (bfs(rGraph, s, t, parent))
    {
        int path_flow = INT_MAX;
        for (v=t; v!=s; v=parent[v])
        {
            u = parent[v];
            path_flow = min(path_flow, rGraph[u][v]);
        }

        for (v=t; v != s; v=parent[v])
        {
            u = parent[v];
            rGraph[u][v] -= path_flow;
            rGraph[v][u] += path_flow;
        }

        // Add path flow to overall flow
        max_flow += path_flow;
    }

    // Return the overall flow
    return max_flow;
}
```

```
STRONGCONNECT(G, u)
 1   time = time + 1
 2   u.d = time
 3   u.low = time
 4   PUSH(u, S)
 5   for each v ∈ G.Adj[u]
 6       if v.d == 0
 7           STRONGCONNECT(G, v)
 8           u.low = min(u.low, v.low)
 9       else if v ∈ S
10               u.low = min(u.low, v.d)
11   if u.low == u.d
12       start new component
13       do
14           w = POP(S)
15           put w in current component
16       while w != u
```

*Components*

1: 6
2: 7, 5, 4, 3
3: 8, 2, 1