Codebook

Jin, Lai, Lam from YZU

October 13, 2020

Contents

1 Environment

1.1 .vimrc

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

1.2 compile

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

1.3 copy

```
1 # copy template file
2 #!/bin/bash
3 for name in {A..M};
4 do
5   cp template.cpp $name.cpp
6   echo 0 > $name.in
7   echo 0 > $name.out
8 done
```

1.4 template

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

2 Structure

```
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
7
8 //Graph (adjacent matrix)
9 matrix[row][col]
10 distance[row][col]
11 visited[row][col]
12 m = row_i, n = col_j
```

3 Code Note

```
1 int GCD(int a, int b){
2   return b == 0 ? a : GCD(b, a % b);
3 }
4
5 int LCM(int a, int b){
6   return a / GCD(b, a % b) * b;
7 }
8
9 // Find shortest path
10 void BFS(Graph, visited, FirstNode){
11   queue Q
12   Q.push(FirstNode)
13   while(!Q.empty){
14     currentNode = Q.pop()
15     if(currentNode == targetNode)break //
     find target
16     if(!visited[currentNode]){
17       visited[currentNode] = true
18       for(all nextNode){
19         if(!visited[nextNode])
20           Q.push(nextNode)
21       }
22     }
23   }
24 }
25
26 void BellmanFord(){
27   e = edge.size();
28   fill(best, best+e, NIL);
```

```
29   best[0] = 0;
30   for(int i = 0; i < n; ++i)
31     for(int j = 0; j < e; ++j)
32       if(best[edge[j].to] > best[edge[j].
   from] + edge[j].weight)
33         best[edge[j].to] = best[edge[j].
   from] + edge[j].weight;
34   for(int j = 0; j < e; ++j)
35     if(best[edge[j].to] > best[edge[j].from
   ] + edge[j].weight){
36       indefinitely = false;
37       break;
38     }
39 }
40
41 int CeilIndex(int A[], int tail[], int low,
     int high, int key){
42   while(high-low > 1){
43     int mid = (high + low) / 2;
44     if(A[tail[mid]] >= key)
45       high = mid;
46     else
47       low = mid;
48   }
49   return high;
50 }
51
52 // Dijkstra
53 struct node{
54   int id,len;
55   node(int n, int weight) : id(n), len(
   weight){};
56   bool operator<(const node &right) const{
   return id > right.id;}
57 };
58
59 // Using a priority queue
60 void Dijkstra(){
61   best[E] = 0;
62   que.push(node(E, 0));
63   while(!que.empty()){
64     node cur = que.top();
65     que.pop();
66     for(int i = 0; i < num[cur.id]; ++i){
67       if(best[path[cur.id][i]] > cur.len +
   w[cur.id][path[cur.id][i]]){
68         best[path[cur.id][i]] = cur.len + w
   [cur.id][path[cur.id][i]];
69         que.push(node(path[cur.id][i], best
   [path[cur.id][i]]));
70       }
71     }
72   }
73 }
74
75 #define INF 0xFFFFFFFF
76 void FloydWarshall(Graph){
77   for i = 0 to size
78     for j = 0 to size
79       for k = 0 to size
80         Graph[i][j] = min(Graph[i][j],
   Graph[i][k] + Graph[k][j]);
81   print Graph[x][y] //get shortest path
     from x to y
82 }
83
84 bool NeagtiveCycle(){
85   for(int i = 0; i < V; ++i){
86     if(dist[i][i] < 0)
87       return ture;
88   }
89   return false;
90 }
91
92 /*
93   0/1 Knapsack Problem
94   recursive function  : c(n, w) = max(c(n
     -1, w), c(n-1, w-weight[n] + cost[n]))
95   c(n, w) : knapsack problem answer
96   n : from item 0_th to n_th
97   w : max_weight
98   weight[n] : weight of item n
99   cost[n] : cost of item n
100 */
101 //bottom up
102 void knapsack(int n, int w){
103   memset(c, 0, sizeof(c))
104   for(all i in n) //all item
105     for(all j in w) //all weight
106       if(j - weight[i] < 0)
107         c[i+1][j] = c[i][j]
108       else
109         c[i+1][j] = max(c[i+1][j], j-weight
   [i]+cost[i])
110   print c[n][w] //the highest value
111 }
112
113 /*
114   Coin Change Problem
115   recursive function  : c(n, m) = c(n-1, m)
     + c(n-1, m-price[n])
116   c(n, m) : coin change problem answer
117   n : from coin 0_th to n_th
118   m : target money
119   price[n]  : coin price
120 */
121 //bottom up
122 void coinChange(int m){
123   memset(c, 0, sizeof(c))
124   c[0] = 1;
125   for(all i in n) //all coin
126     for(all j from price[i] to m) //all
   target money
127       c[j] += c[j-price[i]]
128   print m //target money
129   print c[m] //kinds
130 }
131
132 /*
133   Knapsack/Coin Problem - Algorithm
134   first loop is item
135   Second loop is capacity (weight/value
     target)
136   Third loop(only appear in item limit case
     ) = max(number amount, now value/this
     value)
137
138   backpack structure:
139   Struct {weight,cost}
140
141   w-weight[n] meaning I push this I item
142   n-1 meaning look forward
```

```
143
144    Code:
145    c[i] = max (c[i],c[i - weight[n]] + cost[
        n]) - consider value
146    c[i] = max (c[i],c[i - weight[n]] +1) -
        consider amount of item
147    way[j] += way[j - weight[i]] - consider
        ways
148
149    Coin (like as backpack):
150    Code:
151    c[j] += c[j-price[i]]; << ways
152    c[j] = min(c[j], c[j-price[i]] + 1); -
        min amount of coin
153 */
154
155
156 int LongestIncreasingSubsequence(int A[],
       int n){
157   if(n == 0) return 0;
158   int *tail = new int[n+1];
159   int *prev = new int[n+1];
160
161   int length = 1;
162   tail[1] = 1;
163   for(int i = 2; i <= n; ++i){
164     if(A[i] < A[tail[1]])
165       tail[1] = i;
166     else if(A[i] > A[tail[length]]){
167       prev[i] = tail[length];
168       tail[++length] = i;
169     }
170     else{
171       int position = CeilIndex(A, tail, 1,
      length, A[i]);
172       prev[i] = tail[position-1];
173       tail[position] = i;
174     }
175   }
176 }
177
178 int max1DRangeSum(int column[]{
179   int globalMax = column[1];
180   int localMax = column[1];
```

```
181   for(int i = 2; i <= m; i++){
182     localMax = max(column[i], localMax +
       column[i]);
183     if(globalMax < localMax)
184       globalMax = localMax;
185   }
186   return globalMax;
187 }
188
189 int max2DRangeSum(int A[][n+1]){
190   for(int l = 1; l < n; ++left){
191     memset(rowSum, 0, sizeof(rowSum));
192     for(int r = left; r <= n; ++right){
193       for(int i = 1; i <= m; ++i)
194         rowSum[i] += A[i][r];
195       localMax = max1DRangeSum(rowSum);
196       if(globalMax < localMax)
197         globalMax = localMax;
198     }
199   }
200 }
201
202 int find(int a){
203   return a = (p[a] == a) ? a : (p[a] = find
       (p[a]));
204 }
205
206 void Union(int a, int b){
207   p[find(a)] = find(b);
208 }
```

4 Algorithm Note

Algorithm 1: ArticulationPoints(G)

---

1 **foreach** $vertex\ u \in G.V$ **do**
2 $u.cut =$ **false**
3 **end**
4 **foreach** $vertex\ u \in G.V$ **do**
5   **if** $u.\pi == NIL$ **then**
6   **if** $u.numChildren > 1$ **then**
7 $u.cut =$ **true**
8   **else**
9   **foreach** $v \in G.Adj[u]$ **do**
10   **if** $v.\pi == u$ **then**
11   **if** $v.low >= u.d$ **then**
12 $u.cut =$ **true**
13   **end**
14   **end**
15 **end**

Algorithm 2: Biconnect(G)

---

1 $time = time + 1$
2 $u.d = time$
3 $u.low = time$
4 **foreach** $v \in G.Adj[u]$ **do**
5   **if** $v.d == 0$ **then**
6 $v.\pi = u$
7 $Push((u,v), S)$
8 $Biconnect(G, v)$
9 $u.low = min(u.low, v.low)$
10   **if** $v.low >= u.d$ **then**
11   start new component
12   **do**
13 $(x_1, x_2) = Pop(S)$
14   put (x_1, x_2) in current component
15   **while** $(x_1, x_2! = (u, v)\ and\ x_1.d >= v.d)$;
16   **end**
17   **else if** $v! = u.\pi$ **then**
18 $Push((u,v), S)$
19 $u.low = min(u.low, v.d)$
20   **end**
21 **end**

Algorithm 3: Bridge(G, u)

1 $time = time + 1$
2 $u.d = time$
3 $u.low = time$
4 **foreach** $v \in G.Adj[u]$ **do**
5   **if** $v.d == 0$ **then**
6      $v.\pi = u$
7      $Bridge(G, v)$
8      $u.low = min(u.low, v.low)$
9   **if** $v.low > u.d$ **then**
10        $\{u, v\}$ is a bridge
11   **end**
12   **else if** $v \neq u.\pi$ **then**
13      $u.low = min(u.low, v.d)$
14   **end**
15 **end**

Algorithm 4: TopologicalSort(G)

1 **foreach** $vertex\ u \in G.V$ **do**
2    $u.color = $ WHITE
3 **end**
4 **foreach** $vertex\ u \in G.V$ **do**
5   **if** $u.color = WHITE$ **then**
6      $DFS_Visit(G, u)$
7   **end**
8 **end**

Algorithm 5: DFS_Visit(G, u)

1 $u.color = $ GRAY
2 **foreach** $v \in G.Adj[u]$ **do**
3   **if** $v.color == WHITE$ **then**
4      $DFS_Visit(G, u)$
5   **end**
6 **end**
7 $u.color = $ BLACK insert u onto the front of a linked list

5 C++ Library

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

```

```cpp
// 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, ...)
```

```cpp
/*
  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()
```

```
81 unsigned long long to_ullong()
82
83 // list
84 template <class value_type>
85 list <value_type>
86 iterator begin()
87 iterator end()
88 size_type size()
89 void reserve(size_type)
90 bool empty()
91 ref front(size_type)
92 ref back(size_type)
93 void remove(value_type)
94 void push_front(value_type)
95 void pop_front()
96 void push_back(value_type)
97 void pop_back()
98 iterator insert(const_interator, value_type
     )
99 iterator erase(const_interator)
100
101 // map
102 template <class key_type, class value_type>
103 typedef pair<key_type, value_type>
     instance_type
104 map <key_type, value_type>
105 iterator begin()
106 iterator end()
107 size_type size()
108 bool empty()
109 value_type& operator[](key_type)
110 map<key_type, value_type>::iterator->first
     //key value
111 map<key_type, value_type>::iterator->second
     // mapped value
112 iterator find(key_type)
113 size_type count(key_type)
114 pair<iterator, bool> insert(pair<key_type,
     value_type>(key_type, value_type))
115 size_type erase(key_type)
116
117 // priority_queue
118 template <class value_type>
119 priority_queue <value_type> //priority
```

```
     larger
120 priority_queue <value_type, vector<
     value_type>, greater<value_type> > //
     priority smaller
121 size_t size()
122 bool empty()
123 ref top()
124 void push(value_type)
125 void pop()
126
127 // queue
128 template <class value_type>
129 queue <value_type>
130 size_type size()
131 bool empty()
132 reference front()
133 reference back()
134 void push(value_type)
135 void pop()
136
137 // set
138 template <class value_type>
139 set <value_type>
140 iterator begin()
141 iterator end()
142 size_type size()
143 bool empty()
144 iterator find(value_type)
145 size_type count(value_type)
146 pair<iterator, bool> insert(value_type)
147 size_type erase(value_type)
148 size_type count(value_type) //return the
     number of element
149
150 // stack
151 template <class value_type>
152 stack <value_type>
153 size_type size()
154 bool empty()
155 reference top()
156 void push(value_type)
157 void pop()
158
159 // string
```

```
160 string
161 iterator begin()
162 iterator end()
163 size_type size()
164 void reserve(size_type)
165 bool empty()
166 reference operator[](size_type)
167 reference at(size_type)
168 string operator+= (string)
169 string insert(pos, string)
170 string erase(pos = 0, len)
171 string substr(pos = 0, len)
172 string to_string(value) // c++11
173 string str() // stringstream
174
175 // vector
176 template <class value_type>
177 vector <value_type>
178 iterator begin()
179 iterator end()
180 size_type size()
181 void reserve(size_type)
182 bool empty()
183 reference operator[](size_type)
184 reference at(size_type)
185 void push_back(value_type)
186 void pop_back()
187 iterator insert(const_interator, value_type
     )
188 iterator erase(const_interator)
```

6 Other Tool

6.1 gdb

```
1 l (list)
2 b (breakpoint)
3 r (run)
4 p $value (print $value)
5 c (continue)
6 q quit
7 step
8 display $value
9 info
```

6.2   vim

7   Note

7.1   Preparing

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

7.2   Response Message

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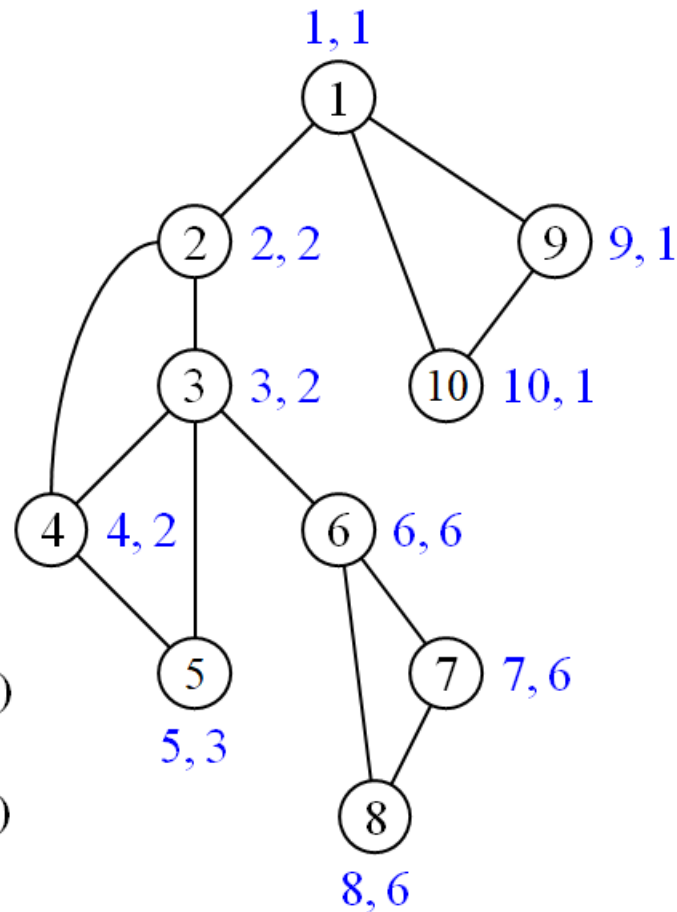
8   Image Note

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
```

Components

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

```cpp
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);
}
```

```cpp
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");
}
```

```cpp
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);
}
```

```c
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 $v \in G.Adj[u]$

6   **if** $v.d == 0$

7   STRONGCONNECT (G, v)

8           $u.low = \min(u.low, v.low)$

9   **else if** $v \in S$

10          $u.low = \min(u.low, v.d)$

11 **if** $u.low == u.d$

12   start new component

13   **do**

14          $w = \text{POP}(S)$

15   put w in current component

16   **while** $w \mathrel{!=} u$

Components

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