

Codebook

1	Basic
1.1	.vimrc
2	Number
3	Matrix
4	Graph
4.1	Bridge And Cut
4.2	BCC
4.3	Two Sat
5	Path
5.1	Kth Shortest
5.2	EulerCircuit
6	Flow
6.1	Dinic
6.2	StoerWanger
6.3	Mixed Euler
7	Match
7.1	KM
7.2	BiMatch
7.3	General Match
8	MST
8.1	Restricted MST
8.2	MDST
8.3	MRST

1 Basic

1.1 .vimrc

```

1
1
2
3
4 1 set nocompatible           " be iMproved, required
4 2 filetype off               " required
4 3
4 4 set rtp+=~/vim/bundle/vundle/
4 5 call vundle#rc()
4 6 Plugin 'gmarik/vundle'
4 7 Plugin 'tpope/vim-fugitive'
4 8 Plugin 'L9'
4 9 Plugin 'Lokaltog/vim-easymotion'
10 Plugin 'rstacruz/sparkup', {'rtp': 'vim/'}
11 Plugin 'valloric/YouCompleteMe'
12 Plugin 'scrooloose/nerdtree'
13 Plugin 'jistr/vim-nerdtree-tabs'
14 Plugin 'bling/vim-airline'
15 Plugin 'terryma/vim-multiple-cursors'
16 filetype plugin indent on    " required
17
18 "airline config
19 set laststatus=2
20 let g:airline_powerline_fonts=1
21 let g:airline#extensions#tabline#enabled=1
22 let g:airline#extensions#tabline#buffer_nr_show=1
23
24 "youcompletememe config
25 let g:ycm_global_ycm_extra_conf = '~/vim/.
   ycm_extra_conf.py'
26 let g:ycm_enable_diagnostic_signs = 0
27 let g:ycm_key_invoke_completion = '<c->'
28 set completeopt=menuone
29
30 "vim-easymotion config
31 map / <Plug>(easymotion-sn)
32 omap / <Plug>(easymotion-tn)
33 map n <Plug>(easymotion-next)
34 map N <Plug>(easymotion-prev)
35
36 set t_Co=256
37 set term=screen-256color
38 set number
39 map <F5> :NERDTreeTabsToggle <CR>
40
41 set tabstop=4
42 set shiftwidth=4
43 set softtabstop=4
44 set expandtab
45
46
47 set wrap
48 set showcmd
49 colorscheme torte
50 map <F2> :w <CR> :call OP() <CR>
51 map! <F2> <ESC> :w <CR> :call OP() <CR> <ESC>
52 map <F9> :w <CR> :call CP_R() <CR> <ESC>
53 map! <F9> <ESC> :w <CR> :call CP_R() <CR> <ESC>
54 map <HOME> ^
55 map! <HOME> <ESC>^i
56 map <ESC>OH <HOME>
57 map! <ESC>OH <HOME>
58 map <END> $
59 map <ESC>OF <END>
60 map! <ESC>OF <ESC><END>a
61 function CP_R()
62
63   if( &ft == 'cpp' )
64     let cpl = 'g++ -w -o "%:r.exe" -std=c++11 "%" |
       let exc = '"./%:r.exe"'
65   elseif( &ft == 'c' )
66     let cpl = 'gcc -w -o "%:r" -std=c99 "%" | let exc
       = '"./%:r"'
67   elseif( &ft == 'java' )
68     let cpl = 'javac "%" | let exc = 'java "%:r"'
69   elseif( &ft == 'python' )
70     let exc = 'python "%'
71   elseif( &ft == 'sh' )
72     let exc = 'sh "%'
73   endif
74

```

```

75 let pause = 'printf "Press any key to continue..." &&
    read -n 1 && exit'
76 if !exists('exc')
77     echo 'Can't compile this filetype...'
78     return
79 endif
80 if exists('cpl')
81     let cp_r = cpl . ' && time ' . exc
82 else
83     let cp_r = 'time ' . exc
84 endif
85 "execute '!'$COLORTERM -x bash -c ' ' . cp_r . ' '; ' .
    pause . ' ;exec bash'''
86 execute '!' clear && ' ' . cp_r . ' && ' . pause
87 endfunction
88
89 function OP()
90     execute '!'$COLORTERM -x gedit ' . "%" . " ;"
91 endfunction

```

2 Number

```

1 #include <stdio.h>
2 #include <vector>
3 #include <math.h>
4 #include <complex>
5 #include <stdlib.h>
6 #include <time.h>
7 #include <iostream>
8 #include <algorithm>
9 using namespace std;
10 typedef long long ll;
11 #define EPS 1e-12
12 #define INF 1e15
13 typedef complex<double> Complex;
14 const double pi = acos(-1);
15
16 /* extended GCD */
17 ll ext_gcd(ll a, ll b, ll &x, ll &y){
18     ll d=a;
19     if(b!=0ll){
20         d=ext_gcd(b,a%b,y,x);
21         y--=(a/b)*x;
22     }
23     else x=1ll,y=0ll;
24     return d;
25 }
26
27 /*
28  * ax = b (mod n)
29  * return a set of answer(vector<ll>)
30  */
31 vector<ll> line_mod_equation(ll a, ll b, ll n){
32     ll x, y, d;
33     d = ext_gcd(a, n, x, y);
34     vector<ll> ans;
35     if(b%d==0ll){
36         x = (x%n + n) % n;
37         ans.push_back((x*(b/d))%(n/d));
38         for(ll i=1;i<d;i++){
39             ans.push_back((ans[0]+i*n/d)%n);
40         }
41         return ans;
42     }
43 }
44
45 /*
46  * find the inverse of n modular p
47  */
48 ll mod_inverse(ll n, ll p){
49     ll x, y;
50     ll d = ext_gcd(n, p, x, y);
51     return (p+x*p) % p;
52 }
53
54 /* P is the modular number */
55 #define P 24851
56 int fact[P+1];
57 /* called by Cmod */
58 int mod_fact(int n, int &e){
59     e = 0;
60     if(n == 0) return 1;

```

```

60 int res = mod_fact(n/P, e);
61 e += n / P;
62 if((n/P) % 2 == 0)
63     return res * (fact[n%P]%P);
64 return res * ((P-fact[n%P])%P);
65 }
66 /*
67  * return C(n, m) mod P
68  */
69 int Cmod(int n, int m){
70     /* this section only need to be done once */
71     fact[0] = 1;
72     for(int i=1;i<=P;i++){
73         fact[i] = fact[i-1] * i%P;
74     }
75     /* end */
76     int a1, a2, a3, e1, e2, e3;
77     a1 = mod_fact(n, e1);
78     a2 = mod_fact(m, e2);
79     a3 = mod_fact(n-m, e3);
80     if(e1 > e2 + e3) return 0;
81     return a1 * mod_inverse(a2 * (a3%P), P) % P;
82 }
83
84 /*
85  * solve the chinese remainder theorem(CRT)
86  * if a.size() != m.size(), return -1
87  * return the minimum positive answer of CRT
88  * x = a[i] (mod m[i])
89  */
90 int CRT(vector<int> a, vector<int> m) {
91     if(a.size() != m.size()) return -1;
92     int M = 1;
93     for(int i=0;i<(int)m.size();i++){
94         M *= m[i];
95     }
96     int res = 0;
97     for(int i=0;i<(int)a.size();i++){
98         res = (res + (M/m[i])*mod_inverse(M/m[i], m[i])
99             *a[i]) % M;
100     }
101     return (res + M) % M;
102 }
103
104 /* fast exponential */
105 ll pow_mod(ll x, ll N, ll M) {
106     ll res = 1;
107     x %= M;
108     while(N){
109         if(N&1ll) res = (res*x) % M;
110         x *= x; x %= M;
111         N >>= 1;
112     }
113     return res;
114 }
115
116 /* called by MillerRabin */
117 bool PrimeTest(ll n, ll a, ll d) {
118     if(n == 2 || n == a) return true;
119     if((n&1) == 0) return false;
120     while((d&1) == 0) d >>= 1;
121     ll t = pow_mod(a, d, n);
122     while((d!=n-1) && (t!=1) && (t!=n-1)){
123         t = (t*t) % n;
124         d <= 1;
125     }
126     return (t==n-1) || ((d&1)==1);
127 }
128
129 /* return true if n is a prime */
130 bool MillerRabin(ll n){
131     // test set
132     vector<ll> a = {2, 7, 61};
133     for(int i=0;i<(int)a.size();i++){
134         if(!PrimeTest(n, a[i], n-1)) return false;
135     }
136     return true;
137 }
138
139 /*
140  * gen phi from 1~MAXN
141  * store answer in phi
142  */
143 #define MAXN 100
144 int mindiv[MAXN], phi[MAXN], sum[MAXN];
145 void genphi(){

```

```

141     for(int i=1;i<MAXN;i++)
142         mindiv[i] = i;
143     for(int i=2;i<MAXN;i++)
144         if(mindiv[i] == i)
145             for(int j=i*i;j<MAXN;j+=i)
146                 mindiv[j] = i;
147     phi[1] = 1;
148     for(int i=2;i<MAXN;i++){
149         phi[i] = phi[i/mindiv[i]];
150         if((i/mindiv[i])%mindiv[i] == 0)
151             phi[i] *= mindiv[i];
152         else phi[i] *= (mindiv[i]-1);
153     }
154 }
155
156 /*
157  * class of polynomial function
158  * coef is the coefficient
159  * f(x) = sigma(c[i]*x^i)
160  */
161 class Function {
162 public:
163     vector<double> coef;
164     Function(const vector<double> c=vector<double>()):
165         coef(c){}
166     double operator () (const double &rhs) const {
167         double res = 0.0;
168         double e = 1.0;
169         for(int i=0;i<(int)coef.size();i++,e*=rhs)
170             res += e * coef[i];
171         return res;
172     }
173     Function derivative() const {
174         vector<double> dc((int)this->coef.size()-1);
175         for(int i=0;i<(int)dc.size();i++)
176             dc[i] = coef[i+1] * (i+1);
177         return Function(dc);
178     }
179     int degree() const {
180         return (int)coef.size()-1;
181     }
182 };
183
184 /*
185  * calculate the integration of f(x) from a to b
186  * divided into n piece
187  * the bigger the n is, the more accurate the answer is
188  */
189 template<class T>
190 double simpson(const T &f, double a, double b){
191     double c = (a+b) / 2.0;
192     return (f(a)+4.0*f(c)+f(b)) * (b-a) / 6.0;
193 }
194
195 template<class T>
196 double simpson(const T &f, double a, double b, double
197     eps, double A){
198     double c = (a+b) / 2.0;
199     double L = simpson(f, a, c), R = simpson(f, c, b);
200     if(fabs(A-L-R) <= 15.0*eps) return L + R + (A-L-R)
201         / 15.0;
202     return simpson(f, a, c, eps/2, L) + simpson(f, c, b,
203         eps/2, R);
204 }
205
206 template<class T>
207 double simpson(const T &f, double a, double b, double
208     eps){
209     return simpson(f, a, b, eps, simpson(f, a, b));
210 }
211
212 /*
213  * called by find
214  * 1 = positive, -1 = negative, 0 = zero
215  */
216 int sign(double x){
217     return x < -EPS ? -1 : x > EPS;
218 }
219
220 /* called by equation */
221 template<class T>
222 double find(const T &f, double lo, double hi){
223     int sign_lo, sign_hi;
224     if((sign_lo=sign(f(lo))) == 0) return lo;
225     if((sign_hi=sign(f(hi))) == 0) return hi;
226     if(sign_hi * sign_lo > 0) return INF;
227     while(hi-lo>EPS){
228         double m = (hi+lo) / 2;
229         int sign_mid = sign(f(m));
230         if(sign_mid == 0) return m;
231         if(sign_lo * sign_mid < 0)
232             hi = m;
233         else lo = m;
234     }
235     return (lo+hi) / 2;
236 }
237
238 /*
239  * return a set of answer of f(x) = 0
240  */
241 template<class T>
242 vector<double> equation(const T &f){
243     vector<double> res;
244     if(f.degree() == 1){
245         if(sign(f.coef[1]))res.push_back(-f.coef[0]/f.
246             coef[1]);
247         return res;
248     }
249     vector<double> droot = equation(f.derivative());
250     droot.insert(droot.begin(), -INF);
251     droot.push_back(INF);
252     for(int i=0;i<(int)droot.size()-1;i++){
253         double tmp = find(f, droot[i], droot[i+1]);
254         if(tmp < INF) res.push_back(tmp);
255     }
256     return res;
257 }
258
259 /*
260  * called by FFT
261  * build the sequence of a that used to calculate FFT
262  * return a reversed sequence
263  */
264 vector<Complex> reverse(vector<Complex> a){
265     vector<Complex> res(a);
266     for (int i=1,j=0;i<(int)res.size();i++){
267         for(int k=((int)res.size())>>1;!(j^=k)&k;k
268             >>=1);
269         if(i > j) swap(res[i], res[j]);
270     }
271     return res;
272 }
273
274 /*
275  * calculate the FFT of sequence
276  * a.size() must be 2^k
277  * flag = 1 -> FFT(a)
278  * flag = -1 -> FFT-1(a)
279  * return FFT(a) or FFT-1(a)
280  */
281 vector<Complex> FFT(vector<Complex> a, int flag=1){
282     vector<Complex> res = reverse(a);
283     for(int k=2;k<=(int)res.size();k<=1){
284         double p0 = -pi / (k>1) * flag;
285         Complex unit_p0(cos(p0), sin(p0));
286         for(int j=0;j<(int)res.size();j+=k){
287             Complex unit(1.0, 0.0);
288             for(int i=j;i<j+k/2;i++,unit*=unit_p0){
289                 Complex t1 = res[i], t2 = res[i+k/2] *
290                     unit;
291                 res[i] = t1 + t2;
292                 res[i+k/2] = t1 - t2;
293             }
294         }
295     }
296     return res;
297 }
298
299 /*
300  * return the sequence of x-th of n!
301  * max(n) = 12
302  * 0 of 3! -> 123
303  * 5 of 3! -> 321
304  */
305 int factorial[] = {1, 1, 2, 6, 24, 120, 720, 5040,
306     40320, 362880, 3628800, 39916800, 479001600};
307
308 vector<int> idx2permutation(int x, int n){

```

```

296     vector<bool> used(n+1, false);
297     vector<int> res(n);
298     for(int i=0;i<n;i++){
299         int tmp = x / factorial[n-i-1];
300         int j;
301         for(j=1;j<=n;j++){if(!used[j]){
302             if(tmp == 0) break;
303             tmp--;
304         }
305         res[i] = j, used[j] = true;
306         x %= factorial[n-i-1];
307     }
308     return res;
309 }
310 /*
311  * a is x-th og n!
312  * return x(0~n!)
313  * 123 of 3! -> 0
314  * 321 of 3! -> 5
315  */
316 int permutation2idx(vector<int> a){
317     int res = 0;
318     for(int i=0;i<(int)a.size();i++){
319         int tmp = a[i] - 1;
320         for(int j=0;j<i;j++){
321             if(a[j] < a[i]) tmp--;
322             res += factorial[(int)a.size()-i-1] * tmp;
323         }
324     }
325     return res;
326 }
327
328 int main(){
329 }

```

3 Matrix

4 Graph

4.1 Bridge And Cut

4.2 BCC

4.3 Two Sat

5 Path

5.1 Kth Shortest

5.2 EulerCircuit

6 Flow

6.1 Dinic

6.2 StoerWanger

6.3 Mixed Euler

7 Match

7.1 KM

7.2 BiMatch

7.3 General Match

8 MST

8.1 Restricted MST

8.2 MDST

8.3 MRST