

Contents

1	準備	1
1.1	init.el	1
1.2	tpl.cpp	1
2	文字列	1
2.1	Aho-Corasick 法	1
3	グラフ	2
3.1	強連結成分分解	2
3.1.1	関節点	2
3.1.2	橋	2
3.1.3	強連結成分分解	3
3.2	フロー	3
3.2.1	最大流	3
3.2.2	二部マッチング	3
3.2.3	最小費用流	4
3.3	木	4
3.3.1	木の直径	4
3.3.2	最小シュタイナー木	4
4	数学	4
4.1	行列	4
4.1.1	単位行列	5
4.1.2	積	5
4.1.3	累乗	5
4.1.4	線形方程式の解 (Givens 消去法)	5
4.1.5	トレース	5

1 準備

1.1 init.el

linum は emacs24 のみ

```
1 ;key
2 (keyboard-translate ?\C-h ?\C-?)
3 (global-set-key "\M-g" 'goto-line)
4
5 ;tab
6 (setq-default indent-tabs-mode nil)
7 (setq-default tab-width 4)
8 (setq indent-line-function 'insert-tab)
9
10 ;line number
11 (global-linum-mode t)
12 (setq linum-format "%4d ")
```

1.2 tpl.cpp

```
1 #include <bits/stdc++.h>
2 using namespace std;
3
4 #define rep(i,a) for(int i = 0;i < (a); i++)
5 #define repi(i,a,b) for(int i = (a); i < (b); i++)
6 #define repd(i,a,b) for(int i = (a); i >= (b); i--)
7 #define repit(i,a) for(__typeof((a).begin()) i = (a).begin(); i != (a).end(); i++)
8 #define all(u) (u).begin(),(u).end()
9 #define rall(u) (u).rbegin(),(u).rend()
10 #define UNIQUE(u) (u).erase(unique(all(u)),(u).end())
11 #define pb push_back
12 #define mp make_pair
13 const int INF = 1e9;
14 const double EPS = 1e-8;
15 const double PI = acos(-1.0);
16
17 typedef long long ll;
18 typedef vector<int> vi;
19 typedef vector<vi> vvi;
20 typedef pair<int,int> pii;
21
22 int main(){
23 }
```

2 文字列

2.1 Aho-Corasick 法

$O(N + M)$

```
1 struct PMA{
2     PMA* next[256];    //0 is failure link
3     vi matched;
4     PMA(){memset(next, 0, sizeof(next));}
5     ~PMA(){rep(i,256) if(next[i]) delete next[i];}
6 };
7 vi set_union(const vi &a,const vi &b){
8     vi res;
```

```

9     set_union(all(a), all(b), back_inserter(res));
10    return res;
11 }
12 // patternからパターンマッチングオートマトンの生成
13 PMA *buildPMA(vector<string> pattern){
14     PMA *root = new PMA, *now;
15     root->next[0] = root;
16     rep(i, pattern.size()){
17         now = root;
18         rep(j, pattern[i].size()){
19             if(now->next[(int)pattern[i][j]] == 0)
20                 now->next[(int)pattern[i][j]] = new PMA;
21             now = now->next[(int)pattern[i][j]];
22         }
23         now->matched.push_back(i);
24     }
25     queue<PMA*> que;
26     rep(i, 1, 256){
27         if(!root->next[i]) root->next[i] = root;
28         else {
29             root->next[i]->next[0] = root;
30             que.push(root->next[i]);
31         }
32     }
33     while(!que.empty()){
34         now = que.front(); que.pop();
35         rep(i, 1, 256){
36             if(now->next[i]){
37                 PMA *next = now->next[i];
38                 while(!next->next[i]) next = next->next[0];
39                 now->next[i]->next[0] = next->next[i];
40                 now->next[i]->matched = set_union(now->next[i]->matched, next->next[i]->matched);
41                 que.push(now->next[i]);
42             }
43         }
44     }
45     return root;
46 }
47 void match(PMA* &pma, const string s, vi &res){
48     rep(i, s.size()){
49         int c = s[i];
50         while(!pma->next[c])
51             pma = pma->next[0];
52         pma = pma->next[c];
53         rep(j, pma->matched.size())
54             res[pma->matched[j]] = 1;
55     }
56 }

```

## 3 グラフ

### 3.1 強連結成分分解

#### 3.1.1 関節点

$O(E)$

ある関節点  $u$  がグラフを  $k$  個に分割するとき  $art$  には  $k-1$  個の  $u$  が含まれる. 不要な場合は unique を忘れないこと.

```

1 vi G[MAX], art; // artに関節点のリストが入る
2 int num[MAX], low[MAX], t, V;
3

```

```

4 void visit(int v, int u){
5     low[v] = num[v] = ++t;
6     repit(e, G[v]){
7         int w = *e;
8         if (num[w] == 0) {
9             visit(w, v);
10            low[v] = min(low[v], low[w]);
11            if ((num[v] == 1 && num[w] != 2) ||
12                (num[v] != 1 && low[w] >= num[v])) art.pb(v);
13        }
14        else low[v] = min(low[v], num[w]);
15    }
16 }
17 void art_point(){
18     memset(low, 0, sizeof(low));
19     memset(num, 0, sizeof(num));
20     art.clear();
21     rep(u, V) if (num[u] == 0) {
22         t = 0;
23         visit(u, -1);
24     }
25     /*
26     sort(all(art));
27     UNIQUE(art);
28     */
29 }

```

#### 3.1.2 橋

$O(V + E)$

```

1 vi G[MAX];
2 vector<pii> brdg; // brdgに橋のリストが入る
3 stack<int> roots, S;
4 int num[MAX], inS[MAX], t, V;
5
6 void visit(int v, int u){
7     num[v] = ++t;
8     S.push(v); inS[v] = 1;
9     roots.push(v);
10    repit(e, G[v]){
11        int w = *e;
12        if(!num[w]) visit(w, v);
13        else if(u != w && inS[w])
14            while(num[roots.top()] > num[w])
15                roots.pop();
16    }
17    if(v == roots.top()){
18        int tu = u, tv = v;
19        if(tu > tv) swap(tu, tv);
20        brdg.pb(pii(tu, tv));
21        while(1){
22            int w = S.top(); S.pop();
23            inS[w] = 0;
24            if(v == w) break;
25        }
26        roots.pop();
27    }
28 }
29
30 void bridge(){
31     memset(num, 0, sizeof(num));
32     memset(inS, 0, sizeof(inS));
33     brdg.clear();
34     while(S.size()) S.pop();
35 }

```

```

35 while(roots.size()) roots.pop();
36 t = 0;
37 rep(u,V) if(num[u] == 0){
38     visit(u,V);
39     brdg.pop_back();
40 }
41 }

```

### 3.1.3 強連結成分分解

$O(V + E)$

```

1 vi G[MAX];
2 vvi scc; // ここに強連結成分分解の結果が入る
3 stack<int> S;
4 int inS[MAX], low[MAX], num[MAX], t, V;
5
6 void visit(int v){
7     low[v] = num[v] = ++t;
8     S.push(v); inS[v] = 1;
9     repit(e,G[v]){
10         int w = *e;
11         if(num[w] == 0){
12             visit(w);
13             low[v] = min(low[v], low[w]);
14         }
15         else if(inS[w]) low[v] = min(low[v], num[w]);
16     }
17     if(low[v] == num[v]){
18         scc.pb(vi());
19         while(1){
20             int w = S.top(); S.pop();
21             inS[w] = 0;
22             scc.back().pb(w);
23             if(v == w) break;
24         }
25     }
26 }
27
28 void stronglyCC(){
29     t = 0;
30     scc.clear();
31     memset(num, 0, sizeof(num));
32     memset(low, 0, sizeof(low));
33     memset(inS, 0, sizeof(inS));
34     while(S.size()) S.pop();
35     rep(u,V) if(num[u] == 0) visit(u);
36 }

```

## 3.2 フロー

### 3.2.1 最大流

$O(EV^2)$

```

1 struct edge{int to, cap, rev;};
2 vector<edge> G[MAX];
3 int level[MAX], itr[MAX];
4
5 void add_edge(int from, int to, int cap){
6     G[from].push_back((edge){to, cap, int(G[to].size())});
7     G[to].push_back((edge){from, 0, int(G[from].size()-1)});

```

```

8 }
9
10 void bfs(int s, int t){
11     memset(level, -1, sizeof(level));
12     queue<int> que; que.push(s);
13     level[s] = 0;
14     while(!que.empty()){
15         int v = que.front(); que.pop();
16         if(v == t) return;
17         for(int i = 0; i < G[v].size(); i++){
18             edge &e = G[v][i];
19             if(e.cap <= 0 or level[e.to] != -1) continue;
20             que.push(e.to);
21             level[e.to] = level[v]+1;
22         }
23     }
24 }
25
26 int dfs(int v, int t, int f){
27     if(v == t) return f;
28     for(int &i = itr[v]; i < G[v].size(); i++){
29         edge &e = G[v][i];
30         if(level[e.to] <= level[v] or e.cap <= 0) continue;
31         int d = dfs(e.to, t, min(f, e.cap));
32         if(d > 0){
33             e.cap -= d;
34             G[e.to][e.rev].cap += d;
35             return d;
36         }
37     }
38     return 0;
39 }
40
41 int max_flow(int s, int t){
42     int flow = 0, f;
43     while(1){
44         bfs(s, t);
45         if(level[t] == -1) return flow;
46         memset(itr, 0, sizeof(itr));
47         while((f = dfs(s, t, INF)) > 0) flow += f;
48     }
49 }

```

### 3.2.2 二部マッチング

$O(EV)$

```

1 int V;
2 vector<int> G[MAX_V];
3 int match[MAX_V];
4 bool used[MAX_V];
5
6 void add_edge(int u, int v){
7     G[u].push_back(v);
8     G[v].push_back(u);
9 }
10
11 bool dfs(int v){
12     used[v] = 1;
13     rep(i,G[v].size()){
14         int u = G[v][i], w = match[u];
15         if(w < 0 || !used[w] && dfs(w)){
16             match[v] = u;
17             match[u] = v;
18             return 1;

```

```

19     }
20 }
21 return 0;
22 }
23
24 int bi_matching(){
25     int res = 0;
26     memset(match, -1, sizeof(match));
27     rep(v,V) if(match[v] < 0){
28         memset(used, 0, sizeof(used));
29         if(dfs(v)) res++;
30     }
31     return res;
32 }

```

### 3.2.3 最小費用流

$O(FE \log V)$

```

1 struct edge{ int to, cap, cost, rev;};
2
3 int V;
4 vector<edge> G[MAX_V];
5 int h[MAX_V];
6 int dist[MAX_V];
7 int prevv[MAX_V], preve[MAX_V];
8
9 void add_edge(int from, int to, int cap, int cost){
10     G[from].push_back((edge){to, cap, cost, int(G[to].size())});
11     G[to].push_back((edge){from, 0, -cost, int(G[from].size() - 1)});
12 }
13
14 int min_cost_flow(int s, int t, int f){
15     int res = 0;
16     fill(h, h + V, 0);
17     while(f > 0){
18         priority_queue<pii, vector<pii>, greater<pii> > que;
19         fill(dist, dist + V, inf);
20         dist[s] = 0;
21         que.push(pii(0, s));
22         while(!que.empty()){
23             pii p = que.top(); que.pop();
24             int v = p.second;
25             if(dist[v] < p.first) continue;
26             rep(i,G[v].size()){
27                 edge &e = G[v][i];
28                 if(e.cap > 0 && dist[e.to] > dist[v] + e.cost + h[v] - h[e.to]){
29                     dist[e.to] = dist[v] + e.cost + h[v] - h[e.to];
30                     prevv[e.to] = v;
31                     preve[e.to] = i;
32                     que.push(pii(dist[e.to], e.to));
33                 }
34             }
35         }
36         if(dist[t] == inf) return -1;
37         rep(v,V) h[v] += dist[v];
38         int d = f;
39         for(int v = t; v != s; v = prevv[v])
40             d = min(d, G[prevv[v]][preve[v]].cap);
41         f -= d;
42         res += d * h[t];
43         for(int v = t; v != s; v = prevv[v]){
44             edge &e = G[prevv[v]][preve[v]];
45             e.cap -= d;
46             G[v][e.rev].cap += d;

```

```

47     }
48 }
49 return res;
50 }

```

## 3.3 木

### 3.3.1 木の直径

ある点 (どこでもよい) から一番遠い点 a を求める. 点 a から一番遠い点までの距離がその木の直径になる.

### 3.3.2 最小シュタイナー木

$O(4^{|T|}V)$

g は無向グラフの隣接行列. T は使いたい頂点の集合.

```

1 int minimum_steiner_tree(vi &T, vvi &g){
2     int n = g.size(), t = T.size();
3     if(t <= 1) return 0;
4     vvi d(g); // all-pair shortest
5     rep(k,n)rep(i,n)rep(j,n) //Warshall Floyd
6         d[i][j] = min(d[i][j], d[i][k] + d[k][j]);
7
8     int opt[1<<t][n];
9     rep(S,1<<t) rep(x,n)
10         opt[S][x] = INF;
11
12     rep(p,t) rep(q,n) // trivial case
13         opt[1<<p][q] = d[T[p]][q];
14
15     repi(S,1,1<<t){ // DP step
16         if(!(S & (S-1))) continue;
17         rep(p,n) rep(E,S)
18             if((E | S) == S)
19                 opt[S][p] = min(opt[S][p], opt[E][p] + opt[S-E][p]);
20         rep(p,n) rep(q,n)
21             opt[S][p] = min(opt[S][p], opt[S][q] + d[p][q]);
22     }
23
24     int ans = INF;
25     rep(S,1<<t) rep(q,n)
26         ans = min(ans, opt[S][q] + opt[((1<<t)-1)-S][q]);
27     return ans;
28 }

```

## 4 数学

### 4.1 行列

C++11 だと array という名前では衝突するので arr にしている.

```

1 typedef double number;
2 typedef vector<number> arr;
3 typedef vector<arr> mat;

```

#### 4.1.1 単位行列

$O(N)$

```

1 mat identity(int n) {
2     mat A(n, arr(n));
3     rep(i,n) A[i][i] = 1;
4     return A;
5 }

```

#### 4.1.2 積

arr*arr	mat*arr	mat*mat
$O(N)$	$O(N^2)$	$O(N^3)$

```

1 number inner_product(const arr &a, const arr &b) {
2     number ans = 0;
3     rep(i,a.size()) ans += a[i] * b[i];
4     return ans;
5 }
6
7 arr mul(const mat &A, const arr &x) {
8     arr y(A.size());
9     rep(i,A.size()) rep(j,A[0].size())
10        y[i] = A[i][j] * x[j];
11     return y;
12 }
13
14 mat mul(const mat &A, const mat &B) {
15     mat C(A.size(), arr(B[0].size()));
16     rep(i,C.size()) rep(j,C[i].size()) rep(k,A[i].size())
17        C[i][j] += A[i][k] * B[k][j];
18     return C;
19 }

```

#### 4.1.3 累乗

$O(N^3 \log e)$

単位行列と積 (mat\*mat) が必要.

```

1 mat pow(const mat &A, int e) {
2     return e == 0 ? identity(A.size()) :
3     e % 2 == 0 ? pow(mul(A, A), e/2) : mul(A, pow(A, e-1));
4 }

```

#### 4.1.4 線形方程式の解 (Givens 消去法)

$O(N^3)$

```

1 #define mkrot(x,y,c,s) {double r = sqrt(x*x+y*y); c = x/r; s = y/r;}
2 #define rot(x,y,c,s) {double u = c*x+s*y; double v = -s*x+c*y; x = u; y = v;}
3 arr givens(mat A, arr b){
4     int n = b.size();
5     rep(i,n) repi(j,i+1,n){
6         double c, s;

```

```

7         mkrot(A[i][i], A[j][i], c, s);
8         rot(b[i], b[j], c, s);
9         repi(k,i,n) rot(A[i][k],A[j][k],c,s);
10    }
11    repd(i,n-1,0){
12        repi(j,i+1,n)
13            b[i] -= A[i][j] * b[j];
14        b[i] /= A[i][i];
15    }
16    return b;
17 }

```

#### 4.1.5 トレース

$O(N)$

```

1 number trace(const mat &A) {
2     number ans = 0;
3     rep(i,A.size()) ans += A[i][i];
4     return ans;
5 }

```