Contents

1 準備

1.1 init.el

linum は emacs24 のみ

```
;key
(keyboard-translate ?\C-h ?\C-?)
(global-set-key "\M-g" 'goto-line)

;tab
(setq-default indent-tabs-mode nil)
(setq-default tab-width 4)
(setq indent-line-function 'insert-tab)

;line number
(global-linum-mode t)
(setq linum-format "%4d ")
```

1.2 tpl.cpp

```
#include <bits/stdc++.h>
using namespace std;

#define rep(i,a) for(int i = 0;i < (a); i++)
#define repi(i,a,b) for(int i = (a); i < (b); i++)
#define repd(i,a,b) for(int i = (a); i >= (b); i--)
#define repit(i,a) for(__typeof((a).begin()) i = (a).begin(); i != (a).end(); i++)
#define all(u) (u).begin(),(u).end()
#define rall(u) (u).rbegin(),(u).rend()
#define UNIQUE(u) (u).erase(unique(all(u)),(u).end())
#define pb push_back
#define mp make_pair
const int INF = 1e9;
const double EPS = 1e-8;
const double PI = acos(-1.0);

typedef long long ll;
typedef vector<int> vi;
typedef vector<vi> vvi;
typedef pair<int,int> pii;

int main(){
}
```

2 文字列

2.1 Aho-Corasick 法

$O(N + M)$

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

```cpp
    set_union(all(a), all(b), back_inserter(res));
    return res;
}
// patternからパターンマッチングオートマトンの生成
PMA *buildPMA(vector<string> pattern){
    PMA *root = new PMA, *now;
    root->next[0] = root;
    rep(i, patter.size()){
        now = root;
        rep(j, pattern[i].size()){
            if(now->next[(int)pattern[i][j]] == 0)
                now->next[(int)pattern[i][j]] = new PMA;
            now = now->next[(int)pattern[i][j]];
        }
        now->matched.push_back(i);
    }
    queue<PMA*> que;
    repi(i,1,256){
        if(!root->next[i]) root->next[i] = root;
        else {
            root->next[i]->next[0] = root;
            que.push(root->next[i]);
        }
    }
    while(!que.empty()){
        now = que.front(); que.pop();
        repi(i,1,256){
            if(now->next[i]){
                PMA *next = now->next[0];
                while(!next->next[i]) next = next->next[0];
                now->next[i]->next[0] = next->next[i];
                now->next[i]->matched = set_union(now->next[i]->matched, next->next[i]->
                    matched);
                que.push(now->next[i]);
            }
        }
    }
    return root;
}
void match(PMA* &pma, const string s, vi &res){
    rep(i,s.size()){
        int c = s[i];
        while(!pma->next[c])
            pma = pma->next[0];
        pma = pma->next[c];
        rep(j,pma->matched.size())
            res[pma->matched[j]] = 1;
    }
}
```

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.

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

void visit(int v, int u){
    low[v] = num[v] = ++t;
    repit(e,G[v]){
        int w = *e;
        if (num[w] == 0) {
            visit(w, v);
            low[v] = min(low[v], low[w]);
            if ((num[v] == 1 && num[w] != 2) ||
                (num[v] != 1 && low[w] >= num[v])) art.pb(v);
        }
        else low[v] = min(low[v], num[w]);
    }
}
void art_point(){
    memset(low, 0, sizeof(low));
    memset(num, 0, sizeof(num));
    art.clear();
    rep(u,V) if (num[u] == 0) {
        t = 0;
        visit(u, -1);
    }
    /*
    sort(all(art));
    UNIQUE(art);
    */
}
```

3.1.2 橋

$O(V+E)$

```cpp
vi G[MAX];
vector<pii> brdg; // brdgに橋のリストが入る
stack<int> roots, S;
int num[MAX], inS[MAX], t, V;

void visit(int v, int u){
    num[v] = ++t;
    S.push(v); inS[v] = 1;
    roots.push(v);
    repit(e, G[v]){
        int w = *e;
        if(!num[w]) visit(w, v);
        else if(u != w && inS[w])
            while(num[roots.top()] > num[w])
                roots.pop();
    }
    if(v == roots.top()){
        int tu = u, tv = v;
        if(tu > tv) swap(tu, tv);
        brdg.pb(pii(tu, tv));
        while(1){
            int w = S.top(); S.pop();
            inS[w] = 0;
            if(v == w) break;
        }
        roots.pop();
    }
}

void bridge(){
    memset(num, 0, sizeof(num));
    memset(inS, 0, sizeof(inS));
    brdg.clear();
    while(S.size()) S.pop();
```

```
    while(roots.size()) roots.pop();
    t = 0;
    rep(u,V) if(num[u] == 0){
        visit(u,V);
        brdg.pop_back();
    }
}
```

3.1.3　強連結成分分解

$O(V+E)$

```
vi G[MAX];
vvi scc; // ここに強連結成分分解の結果が入る
stack<int> S;
int inS[MAX], low[MAX], num[MAX], t, V;

void visit(int v){
    low[v] = num[v] = ++t;
    S.push(v); inS[v] = 1;
    repit(e,G[v]){
        int w = *e;
        if(num[w] == 0){
            visit(w);
            low[v] = min(low[v], low[w]);
        }
        else if(inS[w]) low[v] = min(low[v], num[w]);
    }
    if(low[v] == num[v]){
        scc.pb(vi());
        while(1){
            int w = S.top(); S.pop();
            inS[w] = 0;
            scc.back().pb(w);
            if(v == w) break;
        }
    }
}

void stronglyCC(){
    t = 0;
    scc.clear();
    memset(num, 0, sizeof(num));
    memset(low, 0, sizeof(low));
    memset(inS, 0, sizeof(inS));
    while(S.size()) S.pop();
    rep(u,V) if(num[u] == 0) visit(u);
}
```

3.2　フロー

3.2.1　最大流

$O(EV^2)$

```
struct edge{int to, cap, rev;};
vector<edge> G[MAX];
int level[MAX], itr[MAX];

void add_edge(int from, int to, int cap){
    G[from].push_back((edge){to, cap, int(G[to].size())});
    G[to].push_back((edge){from, 0, int(G[from].size()-1)});
}

void bfs(int s, int t){
    memset(level, -1, sizeof(level));
    queue<int> que; que.push(s);
    level[s] = 0;
    while(!que.empty()){
        int v = que.front(); que.pop();
        if(v == t) return;
        for(int i = 0; i < G[v].size(); i++){
            edge &e = G[v][i];
            if(e.cap <= 0 or level[e.to] != -1) continue;
            que.push(e.to);
            level[e.to] = level[v]+1;
        }
    }
}

int dfs(int v, int t, int f){
    if(v == t) return f;
    for(int &i = itr[v] ; i < G[v].size(); i++){
        edge &e = G[v][i];
        if(level[e.to] <= level[v] or e.cap <= 0) continue;
        int d = dfs(e.to, t, min(f, e.cap));
        if(d > 0){
            e.cap -= d;
            G[e.to][e.rev].cap += d;
            return d;
        }
    }
    return 0;
}

int max_flow(int s, int t){
    int flow = 0, f;
    while(1){
        bfs(s, t);
        if(level[t] == -1) return flow;
        memset(itr, 0, sizeof(itr));
        while((f = dfs(s, t, INF)) > 0) flow += f;
    }
}
```

3.2.2　二部マッチング

$O(EV)$

```
int  V;
vector<int> G[MAX_V];
int match[MAX_V];
bool used[MAX_V];

void add_edge(int u, int v){
    G[u].push_back(v);
    G[v].push_back(u);
}

bool dfs(int v){
    used[v] = 1;
    rep(i,G[v].size()){
        int u = G[v][i], w = match[u];
        if(w < 0 || !used[w] && dfs(w)){
            match[v] = u;
            match[u] = v;
            return 1;
```

```
        }
    }
    return 0;
}

int bi_matching(){
    int res = 0;
    memset(match, -1, sizeof(match));
    rep(v,V) if(match[v] < 0){
        memset(used, 0, sizeof(used));
        if(dfs(v)) res++;
    }
    return res;
}
```

3.2.3 最小費用流

$O(FE \log V)$

```
struct edge{ int to, cap, cost, rev;};

int V;
vector<edge> G[MAX_V];
int h[MAX_V];
int dist[MAX_V];
int prevv[MAX_V], preve[MAX_V];

void add_edge(int from, int to, int cap, int cost){
    G[from].push_back((edge){to, cap, cost, int(G[to].size())});
    G[to].push_back((edge){from, 0, -cost, int(G[from].size() - 1)});
}

int min_cost_flow(int s, int t, int f){
    int res = 0;
    fill(h, h + V, 0);
    while(f > 0){
        priority_queue<pii, vector<pii>, greater<pii> > que;
        fill(dist, dist + V, inf);
        dist[s] = 0;
        que.push(pii(0, s));
        while(!que.empty()){
            pii p = que.top(); que.pop();
            int v = p.second;
            if(dist[v] < p.first) continue;
            rep(i,G[v].size()){
                edge &e = G[v][i];
                if(e.cap > 0 && dist[e.to] > dist[v] + e.cost + h[v] - h[e.to]){
                    dist[e.to] = dist[v] + e.cost + h[v] - h[e.to];
                    prevv[e.to] = v;
                    preve[e.to] = i;
                    que.push(pii(dist[e.to], e.to));
                }
            }
        }
        if(dist[t] == inf) return -1;
        rep(v,V) h[v] += dist[v];
        int d = f;
        for(int v = t; v != s; v = prevv[v])
            d = min(d, G[prevv[v]][preve[v]].cap);
        f -= d;
        res += d * h[t];
        for(int v = t; v != s; v = prevv[v]){
            edge &e = G[prevv[v]][preve[v]];
            e.cap -= d;
            G[v][e.rev].cap += d;
        }
    }
    return res;
}
```

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.

```
int minimum_steiner_tree(vi &T, vvi &g){
    int n = g.size(), t = T.size();
    if(t <= 1) return 0;
    vvi d(g);     // all-pair shortest
    rep(k,n)rep(i,n)rep(j,n)    //Warshall Floyd
        d[i][j] = min(d[i][j], d[i][k] + d[k][j]);

    int opt[1 << t][n];
    rep(S,1<<t) rep(x,n)
        opt[S][x] = INF;

    rep(p,t) rep(q,n)   // trivial case
        opt[1 << p][q] = d[T[p]][q];

    repi(S,1,1<<t){   // DP step
        if(!(S & (S-1))) continue;
        rep(p,n) rep(E,S)
            if((E | S) == S)
                opt[S][p] = min(opt[S][p], opt[E][p] + opt[S-E][p]);
        rep(p,n) rep(q,n)
            opt[S][p] = min(opt[S][p], opt[S][q] + d[p][q]);
    }

    int ans = INF;
    rep(S,1<<t) rep(q,n)
        ans = min(ans, opt[S][q] + opt[((1<<t)-1)-S][q]);
    return ans;
}
```