1

1.1 init.el

linum emacs24

```
 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

```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

2.1 Aho-Corasick

$O(N + M)$

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

```
10      return res;
11  }
12  // pattern
13  PMA *buildPMA(vector<string> pattern){
14      PMA *root = new PMA, *now;
15      root->next[0] = root;
16      rep(i, patter.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      repi(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          repi(i,1,256){
36              if(now->next[i]){
37                  PMA *next = now->next[0];
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, vector<int> &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]] = true;
55      }
56  }
```

3

3.1

$O(EV^2)$

```
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

$O(EV)$

```
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       for(int v = 0; v < V; v++){
28           if(match[v] < 0){
29               memset(used, 0, sizeof(used));
30               if(dfs(v)) res++;
31           }
32       }
33       return res;
34   }
```

3.3

$O(FE \log V)$

```
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   }
```

3.4

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

g   . T   .

```
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   }
```