Contents

1 Basic

1.1 vimrc

```
colo ron
syn on
se ai ar nu rnu
se mouse=a bs=2 ts=4 sw=4 ttm=100
se makeprg=g++\ -Wall\ -Wshadow\ -O2\ -std=c++0x\ -o\
    %<\ %
au BufNewFile *.cpp 0r ~/default.cpp | :1,$-7 fo
filetype indent on

map <F7> <ESC>:wa<CR>:make!<CR>
imap <F7> <ESC>:wa<CR>:make!<CR>
map <F8> :cope <CR>
map <S-F8> :ccl <CR>
map <F9> :!./%< <CR>
map <C-F9> :!./%< < %<.in <CR>
map <C-F7> <ESC>:tabe %<.in<CR>
```

1.2 Default Code

```cpp
#include<bits/stdc++.h>
#define F first
#define S second
#define PB push_back
#define MP make_pair
#define ALL(x) begin(x),end(x)
#define SZ(x) ((int)(x).size())
#define FZ(n) memset((n),0,sizeof(n))
using namespace std;
typedef long long LL;
typedef double D;
typedef long double LDB;
typedef pair<int, int> PII;
typedef pair<LL, LL> PLL;
#define rep(i, n) for(int i = 0; i < n; i ++)
#define rep1(i, a, b) for(int i = a; i < b; i ++)
#define per(i, a, b) for(int i = a; i >= b; i --)
#define IOS ios_base::sync_with_stdio(0); cin.tie(0)

int main(){

}
```

2 Data Structure

2.1 Bigint

```cpp
struct Bigint{
  static const int LEN = 60;
  static const int BIGMOD = 10000;

  int s;
  int vl, v[LEN];
  //  vector<int> v;
  Bigint() : s(1) { vl = 0; }
  Bigint(long long a) {
    s = 1; vl = 0;
    if (a < 0) { s = -1; a = -a; }
    while (a) {
      push_back(a % BIGMOD);
      a /= BIGMOD;
    }
  }
  Bigint(string str) {
    s = 1; vl = 0;
    int stPos = 0, num = 0;
    if (!str.empty() && str[0] == '-') {
      stPos = 1;
      s = -1;
    }
    for (int i=SZ(str)-1, q=1; i>=stPos; i--) {
      num += (str[i] - '0') * q;
      if ((q *= 10) >= BIGMOD) {
        push_back(num);
        num = 0; q = 1;
      }
    }
    if (num) push_back(num);
    n();
  }

  int len() const {
    return vl;
    //    return SZ(v);
  }
  bool empty() const { return len() == 0; }
  void push_back(int x) {
    v[vl++] = x;
    //    v.PB(x);
  }
  void pop_back() {
    vl--;
    //    v.pop_back();
  }
  int back() const {
    return v[vl-1];
    //    return v.back();
  }
  void n() {
    while (!empty() && !back()) pop_back();
  }
  void resize(int nl) {
    vl = nl;
    fill(v, v+vl, 0);
    //    v.resize(nl);
    //    fill(ALL(v), 0);
  }

  void print() const {
    if (empty()) { putchar('0'); return; }
    if (s == -1) putchar('-');
    printf("%d", back());
    for (int i=len()-2; i>=0; i--) printf("%.4d",v[i]);
  }
  friend std::ostream& operator << (std::ostream& out,
      const Bigint &a) {
    if (a.empty()) { out << "0"; return out; }
    if (a.s == -1) out << "-";
    out << a.back();
    for (int i=a.len()-2; i>=0; i--) {
      char str[10];
      snprintf(str, 5, "%.4d", a.v[i]);
      out << str;
    }
    return out;
  }

  int cp3(const Bigint &b)const {
    if (s != b.s) return s - b.s;
    if (s == -1) return -(-*this).cp3(-b);
    if (len() != b.len()) return len()-b.len();//int
    for (int i=len()-1; i>=0; i--)
      if (v[i]!=b.v[i]) return v[i]-b.v[i];
    return 0;
  }

  bool operator < (const Bigint &b)const{ return cp3(b)
      <0; }
  bool operator <= (const Bigint &b)const{ return cp3(b
      )<=0; }
  bool operator == (const Bigint &b)const{ return cp3(b
      )==0; }
  bool operator != (const Bigint &b)const{ return cp3(b
      )!=0; }
  bool operator > (const Bigint &b)const{ return cp3(b)
      >0; }
  bool operator >= (const Bigint &b)const{ return cp3(b
      )>=0; }

  Bigint operator - () const {
    Bigint r = (*this);
    r.s = -r.s;
    return r;
  }
  Bigint operator + (const Bigint &b) const {
    if (s == -1) return -(-(*this)+(-b));
    if (b.s == -1) return (*this)-(-b);
    Bigint r;
    int nl = max(len(), b.len());
    r.resize(nl + 1);
    for (int i=0; i<nl; i++) {
      if (i < len()) r.v[i] += v[i];
      if (i < b.len()) r.v[i] += b.v[i];
      if(r.v[i] >= BIGMOD) {
        r.v[i+1] += r.v[i] / BIGMOD;
        r.v[i] %= BIGMOD;
      }
    }
    r.n();
    return r;
  }
  Bigint operator - (const Bigint &b) const {
    if (s == -1) return -(-(*this)-(-b));
    if (b.s == -1) return (*this)+(-b);
    if ((*this) < b) return -(b-(*this));
    Bigint r;
    r.resize(len());
    for (int i=0; i<len(); i++) {
      r.v[i] += v[i];
      if (i < b.len()) r.v[i] -= b.v[i];
      if (r.v[i] < 0) {
        r.v[i] += BIGMOD;
        r.v[i+1]--;
      }
    }
    r.n();
    return r;
  }
  Bigint operator * (const Bigint &b) {
    Bigint r;
    r.resize(len() + b.len() + 1);
    r.s = s * b.s;
    for (int i=0; i<len(); i++) {
      for (int j=0; j<b.len(); j++) {
        r.v[i+j] += v[i] * b.v[j];
        if(r.v[i+j] >= BIGMOD) {
          r.v[i+j+1] += r.v[i+j] / BIGMOD;
          r.v[i+j] %= BIGMOD;
        }
      }
    }
    r.n();
    return r;
  }
  Bigint operator / (const Bigint &b) {
    Bigint r;
    r.resize(max(1, len()-b.len()+1));
    int oriS = s;
    Bigint b2 = b; // b2 = abs(b)
    s = b2.s = r.s = 1;
    for (int i=r.len()-1; i>=0; i--) {
      int d=0, u=BIGMOD-1;
      while(d<u) {
        int m = (d+u+1)>>1;
        r.v[i] = m;
        if((r*b2) > (*this)) u = m-1;
        else d = m;
      }
      r.v[i] = d;
    }
    s = oriS;
    r.s = s * b.s;
    r.n();
    return r;
  }
  Bigint operator % (const Bigint &b) {
    return (*this)-(*this)/b*b;
  }
};
```

2.2 unordered_map

```
struct Key {
  int first,second;
  Key () {}
  Key (int _x, int _y) : first(_x),  second(_y) {}
  bool operator == (const Key &b) const {
    return tie(F,S) == tie(b.F,b.S);
  }
};
struct KeyHasher {
  size_t operator()(const Key& k) const {
    return k.first + k.second*100000;
  }
};
```

```
typedef unordered_map<Key,int,KeyHasher> map_t;

int main(int argc, char** argv){
  map_t mp;
  for (int i=0; i<10; i++)
    mp[Key(i,0)] = i+1;
  for (int i=0; i<10; i++)
    printf("%d\n", mp[Key(i,0)]);

  return 0;
}
```

2.3 extc_heap

```
#include <bits/extc++.h>
typedef __gnu_pbds::priority_queue<int> heap_t;
heap_t a,b;

int main() {
  a.clear();
  b.clear();
  a.push(1);
  a.push(3);
  b.push(2);
  b.push(4);
  assert(a.top() == 3);
  assert(b.top() == 4);
  // merge two heap
  a.join(b);
  assert(a.top() == 4);
  assert(b.empty());

  return 0;
}
```

2.4 extc_balance_tree

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;
typedef tree<int,null_type,less<int>,rb_tree_tag,
     tree_order_statistics_node_update> set_t;
typedef cc_hash_table<int,int> umap_t;

int main()
{
  // Insert some entries into s.
  set_t s;
  s.insert(12);
  s.insert(505);

  // The order of the keys should be: 12, 505.
  assert(*s.find_by_order(0) == 12);
  assert(s.find_by_order(2) == end(s));

  // The order of the keys should be: 12, 505.
  assert(s.order_of_key(12) == 0);
  assert(s.order_of_key(505) == 1);

  // Erase an entry.
  s.erase(12);

  // The order of the keys should be: 505.
  assert(*s.find_by_order(0) == 505);

  // The order of the keys should be: 505.
  assert(s.order_of_key(505) == 0);
}
```

2.5 Disjoint Set

```
struct DisjointSet {
  // save() is like recursive
  // undo() is like return
  int n, fa[MXN], sz[MXN];
  vector<pair<int*,int>> h;
  vector<int> sp;
  void init(int tn) {
    n=tn;
    for (int i=0; i<n; i++) {
      fa[i]=i;
      sz[i]=1;
    }
    sp.clear(); h.clear();
  }
  void assign(int *k, int v) {
    h.PB({k, *k});
    *k=v;
  }
  void save() { sp.PB(SZ(h)); }
  void undo() {
    assert(!sp.empty());
    int last=sp.back(); sp.pop_back();
    while (SZ(h)!=last) {
      auto x=h.back(); h.pop_back();
      *x.F=x.S;
    }
  }
  int f(int x) {
    while (fa[x]!=x) x=fa[x];
    return x;
  }
  void uni(int x, int y) {
    x=f(x); y=f(y);
    if (x==y) return ;
    if (sz[x]<sz[y]) swap(x, y);
    assign(&sz[x], sz[x]+sz[y]);
    assign(&fa[y], x);
  }
}djs;
```

2.6 Treap

```
const int MEM = 16000004;
struct Treap {
  static Treap nil, mem[MEM], *pmem;
  Treap *l, *r;
  char val;
  int size;
  Treap () : l(&nil), r(&nil), size(0) {}
  Treap (char _val) :
    l(&nil), r(&nil), val(_val), size(1) {}
} Treap::nil, Treap::mem[MEM], *Treap::pmem = Treap::
    mem;

int size(const Treap *t) { return t->size; }
void pull(Treap *t) {
  if (!size(t)) return;
  t->size = size(t->l) + size(t->r) + 1;
}
Treap* merge(Treap *a, Treap *b) {
  if (!size(a)) return b;
  if (!size(b)) return a;
  Treap *t;
  if (rand() % (size(a) + size(b)) < size(a)) {
    t = new (Treap::pmem++) Treap(*a);
    t->r = merge(a->r, b);
  } else {
    t = new (Treap::pmem++) Treap(*b);
    t->l = merge(a, b->l);
  }
  pull(t);
  return t;
}
void split(Treap *t, int k, Treap *&a, Treap *&b) {
  if (!size(t)) a = b = &Treap::nil;
  else if (size(t->l) + 1 <= k) {
    a = new (Treap::pmem++) Treap(*t);
    split(t->r, k - size(t->l) - 1, a->r, b);
    pull(a);
  } else {
    b = new (Treap::pmem++) Treap(*t);
    split(t->l, k, a, b->l);
    pull(b);
  }
}

int nv;
Treap *rt[50005];

void print(const Treap *t) {
  if (!size(t)) return;
  print(t->l);
  cout << t->val;
  print(t->r);
}

int main(int argc, char** argv) {
  IOS;
  rt[nv=0] = &Treap::nil;
  Treap::pmem = Treap::mem;
  int Q, cmd, p, c, v;
  string s;
  cin >> Q;
  while (Q--) {
    cin >> cmd;
    if (cmd == 1) {
      // insert string s after position p
      cin >> p >> s;
      Treap *tl, *tr;
      split(rt[nv], p, tl, tr);
      for (int i=0; i<SZ(s); i++)
        tl = merge(tl, new (Treap::pmem++) Treap(s[i]));
      rt[++nv] = merge(tl, tr);
    } else if (cmd == 2) {
      // remove c characters starting at position
      Treap *tl, *tm, *tr;
      cin >> p >> c;
      split(rt[nv], p-1, tl, tm);
      split(tm, c, tm, tr);
      rt[++nv] = merge(tl, tr);
    } else if (cmd == 3) {
      // print c characters starting at position p, in version v
      Treap *tl, *tm, *tr;
      cin >> v >> p >> c;
      split(rt[v], p-1, tl, tm);
      split(tm, c, tm, tr);
      print(tm);
      cout << "\n";
    }
  }
  return 0;
}
```

3 Graph

3.1 BCC Edge

```
struct BccEdge {
  static const int MXN = 100005;
  struct Edge { int v,eid; };
  int n,m,step,par[MXN],dfn[MXN],low[MXN];
  vector<Edge> E[MXN];
  DisjointSet djs;
  void init(int _n) {
    n = _n; m = 0;
    for (int i=0; i<n; i++) E[i].clear();
    djs.init(n);
  }
  void add_edge(int u, int v) {
    E[u].PB({v, m});
    E[v].PB({u, m});
    m++;
  }
  void DFS(int u, int f, int f_eid) {
    par[u] = f;
    dfn[u] = low[u] = step++;
    for (auto it:E[u]) {
      if (it.eid == f_eid) continue;
      int v = it.v;
      if (dfn[v] == -1) {
        DFS(v, u, it.eid);
        low[u] = min(low[u], low[v]);
      } else {
        low[u] = min(low[u], dfn[v]);
      }
    }
  }
  void solve() {
    step = 0;
    memset(dfn, -1, sizeof(int)*n);
    for (int i=0; i<n; i++) {
      if (dfn[i] == -1) DFS(i, i, -1);
    }
    djs.init(n);
    for (int i=0; i<n; i++) {
      if (low[i] < dfn[i]) djs.uni(i, par[i]);
    }
  }
}graph;
```

3.2 BCC Vertex

```
struct BccVertex {
  int n,nBcc,step,root,dfn[MXN],low[MXN];
  vector<int> E[MXN], ap;
  vector<PII> bcc[MXN];
  int top;
  PII stk[MXN];
  void init(int _n) {
    n = _n;
    nBcc = step = 0;
    for (int i=0; i<n; i++) E[i].clear();
  }
  void add_edge(int u, int v) {
    E[u].PB(v);
    E[v].PB(u);
  }
  void DFS(int u, int f) {
    dfn[u] = low[u] = step++;
    int son = 0;
    for (auto v:E[u]) {
      if (v == f) continue;
      if (dfn[v] == -1) {
        son++;
        stk[top++] = {u,v};
        DFS(v,u);
        if (low[v] >= dfn[u]) {
          if(v != root) ap.PB(v);
          do {
            assert(top > 0);
            bcc[nBcc].PB(stk[--top]);
          } while (stk[top] != PII(u,v));
          nBcc++;
        }
        low[u] = min(low[u], low[v]);
      } else {
        if (dfn[v] < dfn[u]) stk[top++] = PII(u,v);
        low[u] = min(low[u],dfn[v]);
      }
    }
    if (u == root && son > 1) ap.PB(u);
  }
  // return the edges of each bcc;
  vector<vector<PII>> solve() {
    vector<vector<PII>> res;
    for (int i=0; i<n; i++) {
      dfn[i] = low[i] = -1;
    }
    ap.clear();
    for (int i=0; i<n; i++) {
      if (dfn[i] == -1) {
        top = 0;
        root = i;
        DFS(i,i);
      }
    }
    for(int i = 0; i < nBcc; i ++){
    res.PB(bcc[i]);
    bcc[i].clear();
  }
    return res;
  }
}graph;
```

3.3 Strongly Connected Components

```
struct Scc{
  int n, nScc, vst[MXN], bln[MXN];
  vector<int> E[MXN], rE[MXN], vec;
  void init(int _n){
    n = _n;
    for (int i=0; i<n; i++){
      E[i].clear();
      rE[i].clear();
    }
  }
  void add_edge(int u, int v){
    E[u].PB(v);
    rE[v].PB(u);
  }
  void DFS(int u){
    vst[u]=1;
    for (auto v : E[u])
      if (!vst[v]) DFS(v);
    vec.PB(u);
  }
  void rDFS(int u){
    vst[u] = 1;
    bln[u] = nScc;
    for (auto v : rE[u])
      if (!vst[v]) rDFS(v);
  }
  void solve(){
    nScc = 0;
    vec.clear();
    for (int i=0; i<n; i++) vst[i] = 0;
    for (int i=0; i<n; i++)
      if (!vst[i]) DFS(i);
    reverse(vec.begin(),vec.end());
    for (int i=0; i<n; i++) vst[i] = 0;
    for (auto v : vec){
      if (!vst[v]){
        rDFS(v);
        nScc++;
      }
    }
  }
};
```

3.4 Maximum Clique

```
class MaxClique {
public:
    static const int MV = 210;

    int V;
    int el[MV][MV/30+1];
    int dp[MV];
    int ans;
    int s[MV][MV/30+1];
    vector<int> sol;

    void init(int v) {
        V = v; ans = 0;
        FZ(el); FZ(dp);
    }

    /* Zero Base */
    void addEdge(int u, int v) {
        if(u > v) swap(u, v);
        if(u == v) return;
        el[u][v/32] |= (1<<(v%32));
    }

    bool dfs(int v, int k) {
        int c = 0, d = 0;
        for(int i=0; i<(V+31)/32; i++) {
            s[k][i] = el[v][i];
            if(k != 1) s[k][i] &= s[k-1][i];
            c += __builtin_popcount(s[k][i]);
        }
        if(c == 0) {
            if(k > ans) {
                ans = k;
                sol.clear();
                sol.push_back(v);
                return 1;
            }
            return 0;
        }
        for(int i=0; i<(V+31)/32; i++) {
            for(int a = s[k][i]; a ; d++) {
                if(k + (c-d) <= ans) return 0;
                int lb = a&(-a), lg = 0;
                a ^= lb;
                while(lb!=1) {
                    lb = (unsigned int)(lb) >> 1;
                    lg ++;
                }
                int u = i*32 + lg;
                if(k + dp[u] <= ans) return 0;
                if(dfs(u, k+1)) {
                    sol.push_back(v);
                    return 1;
                }
            }
        }
        return 0;
    }

    int solve() {
        for(int i=V-1; i>=0; i--) {
            dfs(i, 1);
            dp[i] = ans;
        }
        return ans;
    }
};
```

3.5 MinimumMeanCycle

```
/* minimum mean cycle */
const int MAXE = 1805;
const int MAXN = 35;
const double inf = 1029384756;
const double eps = 1e-6;
struct Edge {
  int v,u;
  double c;
};
int n,m,prv[MAXN][MAXN], prve[MAXN][MAXN], vst[MAXN];
Edge e[MAXE];
vector<int> edgeID, cycle, rho;
double d[MAXN][MAXN];
inline void bellman_ford() {
  for(int i=0; i<n; i++) d[0][i]=0;
  for(int i=0; i<n; i++) {
    fill(d[i+1], d[i+1]+n, inf);
    for(int j=0; j<m; j++) {
      int v = e[j].v, u = e[j].u;
      if(d[i][v]<inf && d[i+1][u]>d[i][v]+e[j].c) {
        d[i+1][u] = d[i][v]+e[j].c;
        prv[i+1][u] = v;
        prve[i+1][u] = j;
      }
    }
  }
}
double karp_mmc() {
  // returns inf if no cycle, mmc otherwise
  double mmc=inf;
  int st = -1;
  bellman_ford();
  for(int i=0; i<n; i++) {
    double avg=-inf;
    for(int k=0; k<n; k++) {
      if(d[n][i]<inf-eps) avg=max(avg,(d[n][i]-d[k][i])
          /(n-k));
      else avg=max(avg,inf);
    }
    if (avg < mmc) tie(mmc, st) = tie(avg, i);
  }
  for(int i=0; i<n; i++) vst[i] = 0;
  edgeID.clear(); cycle.clear(); rho.clear();
  for (int i=n; !vst[st]; st=prv[i--][st]) {
    vst[st]++;
    edgeID.PB(prve[i][st]);
    rho.PB(st);
  }
  while (vst[st] != 2) {
    int v = rho.back(); rho.pop_back();
    cycle.PB(v);
    vst[v]++;
  }
  reverse(ALL(edgeID));
  edgeID.resize(SZ(cycle));
  return mmc;
}
```

4 Flow

4.1 Dinic

```
struct Dinic{
  static const int MXN = 10000;
  struct Edge{ int v,f,re; };
  int n,s,t,level[MXN];
  vector<Edge> E[MXN];
  void init(int _n, int _s, int _t){
    n = _n; s = _s; t = _t;
    for (int i=0; i<n; i++) E[i].clear();
  }
  void add_edge(int u, int v, int f){
    E[u].PB({v,f,SZ(E[v])});
    E[v].PB({u,0,SZ(E[u])-1});
  }
  bool BFS(){
    for (int i=0; i<n; i++) level[i] = -1;
    queue<int> que;
    que.push(s);
    level[s] = 0;
    while (!que.empty()){
      int u = que.front(); que.pop();
      for (auto it : E[u]){
        if (it.f > 0 && level[it.v] == -1){
          level[it.v] = level[u]+1;
          que.push(it.v);
        }
      }
    }
    return level[t] != -1;
  }
  int DFS(int u, int nf){
    if (u == t) return nf;
    int res = 0;
    for (auto &it : E[u]){
      if (it.f > 0 && level[it.v] == level[u]+1){
        int tf = DFS(it.v, min(nf,it.f));
        res += tf; nf -= tf; it.f -= tf;
        E[it.v][it.re].f += tf;
        if (nf == 0) return res;
      }
    }
    if (!res) level[u] = -1;
    return res;
  }
  int flow(int res=0){
    while ( BFS() )
      res += DFS(s,2147483647);
    return res;
  }
}flow;
```

4.2 Cost Flow

```
typedef pair<long long, long long> pll;
struct CostFlow {
  static const int MXN = 205;
  static const long long INF = 102938475610293847LL;
  struct Edge {
    int v, r;
    long long f, c;
  };
  int n, s, t, prv[MXN], prvL[MXN], inq[MXN];
  long long dis[MXN], fl, cost;
  vector<Edge> E[MXN];
  void init(int _n, int _s, int _t) {
    n = _n; s = _s; t = _t;
    for (int i=0; i<n; i++) E[i].clear();
    fl = cost = 0;
  }
  void add_edge(int u, int v, long long f, long long c)
      {
    E[u].PB({v, SZ(E[v])  , f,  c});
    E[v].PB({u, SZ(E[u])-1, 0, -c});
  }
  pll flow() {
    while (true) {
      for (int i=0; i<n; i++) {
        dis[i] = INF;
        inq[i] = 0;
      }
      dis[s] = 0;
      queue<int> que;
      que.push(s);
      while (!que.empty()) {
        int u = que.front(); que.pop();
        inq[u] = 0;
        for (int i=0; i<SZ(E[u]); i++) {
          int v = E[u][i].v;
          long long w = E[u][i].c;
          if (E[u][i].f > 0 && dis[v] > dis[u] + w) {
            prv[v] = u; prvL[v] = i;
            dis[v] = dis[u] + w;
            if (!inq[v]) {
              inq[v] = 1;
              que.push(v);
            }
          }
        }
      }
      if (dis[t] == INF) break;
      long long tf = INF;
      for (int v=t, u, l; v!=s; v=u) {
        u=prv[v]; l=prvL[v];
        tf = min(tf, E[u][l].f);
      }
      for (int v=t, u, l; v!=s; v=u) {
        u=prv[v]; l=prvL[v];
        E[u][l].f -= tf;
        E[v][E[u][l].r].f += tf;
      }
      cost += tf * dis[t];
      fl += tf;
    }
    return {fl, cost};
  }
}flow;
```

4.3 Maximum Simple Graph Matching

```
struct GenMatch { // 1-base
  static const int MAXN = 514;
  int V;
  bool el[MAXN][MAXN];
  int pr[MAXN];
  bool inq[MAXN],inp[MAXN],inb[MAXN];
  queue<int> qe;
  int st,ed;
  int nb;
  int bk[MAXN],djs[MAXN];
  int ans;
  void init(int _V) {
    V = _V;
    for(int i = 0; i <= V; i++) {
      for(int j = 0; j <= V; j++) el[i][j] = 0;
      pr[i] = bk[i] = djs[i] = 0;
      inq[i] = inp[i] = inb[i] = 0;
    }
    ans = 0;
  }
  void add_edge(int u, int v) {
    el[u][v] = el[v][u] = 1;
  }
  int lca(int u,int v) {
    for(int i = 0; i <= V; i++) inp[i] = 0;
    while(1) {
      u = djs[u];
      inp[u] = true;
      if(u == st) break;
      u = bk[pr[u]];
    }
    while(1) {
      v = djs[v];
      if(inp[v]) return v;
      v = bk[pr[v]];
    }
    return v;
  }
  void upd(int u) {
    int v;
    while(djs[u] != nb) {
      v = pr[u];
      inb[djs[u]] = inb[djs[v]] = true;
      u = bk[v];
```

```
   if(djs[u] != nb) bk[u] = v;
  }
 }
 void blo(int u,int v) {
  nb = lca(u,v);
  for (int i=0; i<=V; i++) inb[i] = 0;
  upd(u); upd(v);
  if(djs[u] != nb) bk[u] = v;
  if(djs[v] != nb) bk[v] = u;
  for(int tu = 1; tu <= V; tu++)
   if(inb[djs[tu]]) {
    djs[tu] = nb;
    if(!inq[tu]){
     qe.push(tu);
     inq[tu] = 1;
    }
   }
 }
 void flow() {
  for(int i = 1; i <= V; i++) {
   inq[i] = 0;
   bk[i] = 0;
   djs[i] = i;
  }

  while(qe.size()) qe.pop();
  qe.push(st);
  inq[st] = 1;
  ed = 0;
  while(qe.size()) {
   int u = qe.front(); qe.pop();
   for(int v = 1; v <= V; v++)
    if(el[u][v] && (djs[u] != djs[v]) && (pr[u] !=
      v)) {
     if((v == st) || ((pr[v] > 0) && bk[pr[v]] >
       0))
      blo(u,v);
     else if(bk[v] == 0) {
      bk[v] = u;
      if(pr[v] > 0) {
       if(!inq[pr[v]]) qe.push(pr[v]);
      } else {
       ed = v;
       return;
      }
     }
    }
  }
 }
 void aug() {
  int u,v,w;
  u = ed;
  while(u > 0) {
   v = bk[u];
   w = pr[v];
   pr[v] = u;
   pr[u] = v;
   u = w;
  }
 }
 int solve() {
  for(int i = 0; i <= V; i++) pr[i] = 0;
  for(int u = 1; u <= V; u++)
   if(pr[u] == 0) {
    st = u;
    flow();
    if(ed > 0) {
     aug();
     ans ++;
    }
   }
  return ans;
 }
}G;

int main() {
 G.init(V);
 for(int i=0; i<E; i++) {
  int u, v;
  cin >> u >> v;
  G.add_edge(u, v);
 }
 cout << G.solve() << endl;
}
```

4.4 Minimum Weight Matching (Clique version)

```
struct Graph {
 // Minimum General Weighted Matching (Perfect Match)
   0-base
 static const int MXN = 105;

 int n, edge[MXN][MXN];
 int match[MXN],dis[MXN],onstk[MXN];
 vector<int> stk;

 void init(int _n) {
  n = _n;
  for (int i=0; i<n; i++)
   for (int j=0; j<n; j++)
    edge[i][j] = 0;
 }
 void add_edge(int u, int v, int w) {
  edge[u][v] = edge[v][u] = w;
 }
 bool SPFA(int u){
  if (onstk[u]) return true;
  stk.PB(u);
  onstk[u] = 1;
  for (int v=0; v<n; v++){
   if (u != v && match[u] != v && !onstk[v]){
    int m = match[v];
    if (dis[m] > dis[u] - edge[v][m] + edge[u][v]){
     dis[m] = dis[u] - edge[v][m] + edge[u][v];
     onstk[v] = 1;
     stk.PB(v);
     if (SPFA(m)) return true;
     stk.pop_back();
     onstk[v] = 0;
    }
   }
  }
  onstk[u] = 0;
  stk.pop_back();
  return false;
 }

 int solve() {
  // find a match
  for (int i=0; i<n; i+=2){
   match[i] = i+1;
   match[i+1] = i;
  }
  while (true){
   int found = 0;
   for (int i=0; i<n; i++)
    dis[i] = onstk[i] = 0;
   for (int i=0; i<n; i++){
    stk.clear();
    if (!onstk[i] && SPFA(i)){
     found = 1;
     while (SZ(stk)>=2){
      int u = stk.back(); stk.pop_back();
      int v = stk.back(); stk.pop_back();
      match[u] = v;
      match[v] = u;
     }
    }
   }
   if (!found) break;
  }
  int ret = 0;
  for (int i=0; i<n; i++)
   ret += edge[i][match[i]];
  ret /= 2;
  return ret;
 }
}graph;
```

5 Math

5.1 ax+by=gcd

```
typedef pair<int, int> pii;
```

```
pii gcd(int a, int b){
  if(b == 0) return make_pair(1, 0);
  else{
    int p = a / b;
    pii q = gcd(b, a % b);
    return make_pair(q.second, q.first - q.second * p);
  }
}
```

5.2 Segmented Sieve

```
bool sieve[MXN];
void linear_sieve(){
  vector<int> prime;
  for(int i=2; i< MXN; ++i){
    if(!sieve[i]) prime.push_back(i);
    for(int j = 0; i*prime[j] < N; ++j){
      sieve[i*prime[j]] = true;
      if(i % prime[j] == 0) break;
    }
  }
}
```

5.3 Fast Fourier Transform

```
struct cp{
    double _a,_b;
    cp operator +(const cp &o)const {return (cp){_a+o.
        _a,_b+o._b};}
    cp operator -(const cp &o)const {return (cp){_a-o.
        _a,_b-o._b};}
    cp operator *(const cp &o)const {return (cp){_a*o.
        _a-_b*o._b,_b*o._a+_a*o._b};}
    cp operator *(const double &o)const {return (cp){_a
        *o,_b*o};}
    cp operator !() const{return (cp){_a,-_b};}
}w[MAX];
int pos[MAX];
void fft_init(int len){
    int j = 0;
    while((1<<j) < len) j++;
    j--;
    for(int i=0;i<len;i++)
        pos[i] = pos[i>>1]>>1 | ((i&1)<<j);
}
void fft(cp *_x, int len, int sta){
    for(int i = 0; i < len; i++)
        if(i < pos[i]) swap(_x[i], _x[pos[i]]);
    w[0] = (cp){1,0};
    for(unsigned i = 2; i <= len; i <<= 1){
        cp g=(cp){cos(2*PI/i),sin(2*PI/i)*sta};
        for(int j = i>>1; j >= 0; j -= 2) w[j] = w[j
            >>1];
        for(int j = 1; j < (i>>1); j += 2)w[j] = w[j
            -1]*g;
        for(int j = 0; j < len; j += i){
            cp *_a = _x+j, *_b = _a+(i>>1);
            for(int l = 0; l < (i>>1); l++){
                cp o = _b[l]*w[l];
                _b[l] = _a[l]-o;
                _a[l] = _a[l]+o;
            }
        }
    }
    if(sta == -1) for(int i = 0; i < len; i++) _x[i]._a
         /= len, _x[i]._b /= len;
}
cp _x[MAX],_y[MAX],_z[MAX];
void FFT(int _a, int _b, int l1, int l2, int _c){
    int len = 1;
    while(len <= (l1+l2)>>1)len <<= 1;
    fft_init(len);
    for(int i = l1>>1; i<len; i++) _x[i]._a = _x[i]._b
        = 0;
    for(int i = l2>>1; i<len; i++) _y[i]._a = _y[i]._b
        = 0;
    for(int i = 0; i < l1; i++) (i&1 ? _x[i>>1]._b : _x
        [i>>1]._a) = st[_a][i];
    for(int i = 0; i < l2; i++) (i&1 ? _y[i>>1]._b : _y
        [i>>1]._a) = st[_b][i];
    fft(_x, len, 1), fft(_y, len, 1);
    for(int i = 0; i < len>>1; i++){
        int j = len - 1&len - i;
        _z[i] = _x[i]*_y[i] - (_x[i]-!_x[j])*(_y[i]-!_y
            [j])*(w[i]+(cp){1,0})*0.25;
    }
    for(int i = len>>1; i < len; i++){
        int j = len - 1&len -i;
        _z[i] = _x[i]*_y[i] - (_x[i]-!_x[j])*(_y[i]-!_y
            [j])*((cp){1,0}-w[i^len>>1])*0.25;
    }
    fft(_z, len, -1);
    for(int i = 0; i < l1 + l2 - 1;i++)
        if(i&1) st[_c][i] = (LL)(_z[i>>1]._b+0.5);
        else st[_c][i] = (LL)(_z[i>>1]._a+0.5);
}
```

5.4 Theorem

5.4.1 Lucas' Theorem

For non-negative integer n, m and prime p, $\binom{m}{n} \equiv \prod_{i=0}^{k} \binom{m_i}{n_i} \pmod{p}$ where m_i is the i-th digit of m in base p.

5.4.2 Sum of Two Squares Thm (Legendre)

For a given positive integer n, let
$D_1 = (\#$ of positive integers d dividing N that $1 \equiv d \pmod{4})$
$D_3 = (\#$ of positive integers d dividing N that $3 \equiv d \pmod{4})$
then n can be written as a sum of two squares in exactly
$R(n) = 4(D_1 - D_3)$ ways.

5.4.3 Difference of D1-D3 Thm

let $n = 2^t \cdot (p_1^{e1} \cdot \ldots \cdot p_r^{er}) \cdots (q_1^{f1} \cdot \ldots \cdot q_s^{fs})$
where p_i, q_i are primes and $1 \equiv p_i \pmod{4}, 3 \equiv q_i \pmod{4}$

then $D_1 - D_3 = \begin{cases} (e_1+1)(e_2+1)...(e_r+1), & \text{if } f_i \text{ all even} \\ 0, & \text{if any } f_i \text{ is odd} \end{cases}$

5.4.4 Krush–Kuhn–Tucker Conditions

Stationarity
For maximizing $f(x)$: $\nabla f(x^*) = \sum_{i=1}^{m} \mu_i \nabla g_i(x^*) + \sum_{j=1}^{l} \lambda_j \nabla h_j(x^*)$
For minimizing $f(x)$: $-\nabla f(x^*) = \sum_{i=1}^{m} \mu_i \nabla g_i(x^*) + \sum_{j=1}^{l} \lambda_j \nabla h_j(x^*)$

Primal feasibility
$g_i(x^*) \le 0,$ for all $i = 1, \ldots, m$
$h_j(x^*) = 0,$ for all $j = 1, \ldots, l$

Dual feasibility
$\mu_i \ge 0,$ for all $i = 1, \ldots, m$

Complementary slackness
$\mu_i g_i(x^*) = 0,$ for all $i = 1, \ldots, m$

5.4.5 Chinese remainder theorem

$x \equiv r_i \mod p_i$
$N = \prod p_i$
$N_i = N/p_i$
$x \equiv \sum r_i N_i (N_i)_{p_i}^{-1} \mod N$

6 Geometry

6.1 Point operators

```
#define x first
#define y second

#define cpdd const pdd
struct pdd : pair<double, double> {
    using pair<double, double>::pair;

    pdd operator + (cpdd &p) const {
        return {x+p.x, y+p.y};
    }

    pdd operator - () const {
        return {-x, -y};
    }
    pdd operator - (cpdd &p) const {
        return (*this) + (-p);
    }

    pdd operator * (double f) const {
        return {f*x, f*y};
    }

    double operator * (cpdd &p) const {
        return x*p.x + y*p.y;
    }
};

double abs(cpdd &p) { return hypot(p.x, p.y); }
double arg(cpdd &p) { return atan2(p.y, p.x); }
double cross(cpdd &p, cpdd &q) { return p.x*q.y - p.y*q
    .x; }
double cross(cpdd &p, cpdd &q, cpdd &o) { return cross(
    p-o, q-o); }
pdd operator * (double f, cpdd &p) { return p*f; } //
    !! Not f*p !!
```

6.2 Intersection of two circles

```
using ld = double;
vector<pdd> interCircle(pdd o1, double r1, pdd o2,
    double r2) {
  ld d2 = (o1 - o2) * (o1 - o2);
  ld d = sqrt(d2);
  if (d < abs(r1-r2)) return {};
  if (d > r1+r2) return {};
  pdd u = 0.5*(o1+o2) + ((r2*r2-r1*r1)/(2*d2))*(o1-o2);
  double A = sqrt((r1+r2+d) * (r1-r2+d) * (r1+r2-d) *
      (-r1+r2+d));
  pdd v = A / (2*d2) * pdd(o1.S-o2.S, -o1.F+o2.F);
  return {u+v, u-v};
}
```

6.3 Intersection of two lines

```
const double EPS = 1e-9;

pdd interPnt(pdd p1, pdd p2, pdd q1, pdd q2, bool &res)
    {
  double f1 = cross(p2, q1, p1);
  double f2 = -cross(p2, q2, p1);
  double f = (f1 + f2);

  if(fabs(f) < EPS) {
    res = false;
    return {};
    }

  res = true;
  return (f2 / f) * q1 + (f1 / f) * q2;
}
```

6.4 Circle cover

```
typedef double type;
typedef pair<type,type> Pt;
typedef pair<Pt,Pt> Line;
typedef pair<Pt,type> Circle;
#define X first
#define Y second
#define O first
#define R second
Pt operator+( const Pt& p1 , const Pt& p2 ){
  return { p1.X + p2.X , p1.Y + p2.Y };
}
Pt operator-( const Pt& p1 , const Pt& p2 ){
  return { p1.X - p2.X , p1.Y - p2.Y };
}
Pt operator*( const Pt& tp , const type& tk ){
  return { tp.X * tk , tp.Y * tk };
}
Pt operator/( const Pt& tp , const type& tk ){
  return { tp.X / tk , tp.Y / tk };
}
type operator*( const Pt& p1 , const Pt& p2 ){
  return p1.X * p2.X + p1.Y * p2.Y;
}
type operator^( const Pt& p1 , const Pt& p2 ){
  return p1.X * p2.Y - p1.Y * p2.X;
}
type norm2( const Pt& tp ){
  return tp * tp;
}
double norm( const Pt& tp ){
  return sqrt( norm2( tp ) );
}
Pt perp( const Pt& tp ){
  return { tp.Y , -tp.X };
}

#define N 1021
struct CircleCover{
  int C; Circ c[ N ];
  bool g[ N ][ N ], overlap[ N ][ N ];
  // Area[i] : area covered by at least i circles
  D Area[ N ];
  void init( int _C ){ C = _C; }
  bool CCinter( Circ& a , Circ& b , Pt& p1 , Pt& p2 ){
    Pt o1 = a.O , o2 = b.O;
    D r1 = a.R , r2 = b.R;
    if( norm( o1 - o2 ) > r1 + r2 ) return {};
    if( norm( o1 - o2 ) < max(r1, r2) - min(r1, r2) )
        return {};
    D d2 = ( o1 - o2 ) * ( o1 - o2 );
    D d = sqrt(d2);
    if( d > r1 + r2 ) return false;
    Pt u=(o1+o2)*0.5 + (o1-o2)*((r2*r2-r1*r1)/(2*d2));
    D A=sqrt((r1+r2+d)*(r1-r2+d)*(r1+r2-d)*(-r1+r2+d));
    Pt v=Pt( o1.Y-o2.Y , -o1.X + o2.X ) * A / (2*d2);
    p1 = u + v; p2 = u - v;
    return true;
  }
  struct Teve {
    Pt p; D ang; int add;
    Teve() {}
    Teve(Pt _a, D _b, int _c):p(_a), ang(_b), add(_c){}
    bool operator<(const Teve &a)const
    {return ang < a.ang;}
  }eve[ N * 2 ];
  // strict: x = 0, otherwise x = -1
  bool disjuct( Circ& a, Circ &b, int x )
  {return sign( norm( a.O - b.O ) - a.R - b.R ) > x;}
  bool contain( Circ& a, Circ &b, int x )
  {return sign( a.R - b.R - norm( a.O - b.O ) ) > x;}
  bool contain(int i, int j){
    /* c[j] is non-strictly in c[i]. */
    return (sign(c[i].R - c[j].R) > 0 ||
            (sign(c[i].R - c[j].R) == 0 && i < j) ) &&
                contain(c[i], c[j], -1);
  }
  void solve(){
    for( int i = 0 ; i <= C + 1 ; i ++ )
      Area[ i ] = 0;
    for( int i = 0 ; i < C ; i ++ )
      for( int j = 0 ; j < C ; j ++ )
        overlap[i][j] = contain(i, j);
    for( int i = 0 ; i < C ; i ++ )
      for( int j = 0 ; j < C ; j ++ )
        g[i][j] = !(overlap[i][j] || overlap[j][i] ||
                    disjuct(c[i], c[j], -1));
    for( int i = 0 ; i < C ; i ++ ){
      int E = 0, cnt = 1;
      for( int j = 0 ; j < C ; j ++ )
        if( j != i && overlap[j][i] )
          cnt ++;
      for( int j = 0 ; j < C ; j ++ )
        if( i != j && g[i][j] ){
          Pt aa, bb;
          CCinter(c[i], c[j], aa, bb);
          D A=atan2(aa.Y - c[i].O.Y, aa.X - c[i].O.X);
          D B=atan2(bb.Y - c[i].O.Y, bb.X - c[i].O.X);
          eve[E ++] = Teve(bb, B, 1);
          eve[E ++] = Teve(aa, A, -1);
          if(B > A) cnt ++;
        }
      if( E == 0 ) Area[ cnt ] += pi * c[i].R * c[i].R;
      else{
        sort( eve , eve + E );
        eve[E] = eve[0];
        for( int j = 0 ; j < E ; j ++ ){
          cnt += eve[j].add;
          Area[cnt] += (eve[j].p ^ eve[j + 1].p) * .5;
          D theta = eve[j + 1].ang - eve[j].ang;
          if (theta < 0) theta += 2. * pi;
          Area[cnt] +=
            (theta - sin(theta)) * c[i].R*c[i].R * .5;
        }
      }
    }
  }
};
```

6.5 Half Plane Intersection

```
const double EPS = 1e-9;

pdd interPnt(Line l1, Line l2, bool &res){
    pdd p1, p2, q1, q2;
    tie(p1, p2) = l1;
    tie(q1, q2) = l2;
  double f1 = cross(p2, q1, p1);
    double f2 = -cross(p2, q2, p1);
  double f = (f1 + f2);

    if(fabs(f) < EPS) {
        res = false;
        return {0, 0};
    }

    res = true;
  return (f2 / f) * q1 + (f1 / f) * q2;
}

bool isin(Line l0, Line l1, Line l2) {
    // Check inter(l1, l2) in l0
    bool res;
    pdd p = interPnt(l1, l2, res);
    return cross(l0.S, p, l0.F) > EPS;
}

/* If no solution, check: 1. ret.size() < 3
 * Or more precisely, 2. interPnt(ret[0], ret[1])
 * in all the lines. (use (l.S - l.F).cross(p - l.F) >
     0
 */
vector<Line> halfPlaneInter(vector<Line> lines) {
    int sz = lines.size();
    vector<double> ata(sz), ord(sz);
    for (int i=0; i<sz; i++) {
        ord[i] = i;
        pdd d = lines[i].S - lines[i].F;
        ata[i] = atan2(d.y, d.x);
    }
    sort(ALL(ord), [&](int i, int j) {
        if (abs(ata[i] - ata[j]) < EPS) {
            return cross(lines[i].S, lines[j].S, lines[
                i].F) < 0;
        }
        return ata[i] < ata[j];
    });
    vector<Line> fin;
    for (int i=0; i<sz; i++) {
        if (!i or fabs(ata[ord[i]] - ata[ord[i-1]]) >
            EPS) {
            fin.PB(lines[ord[i]]);
        }
    }

    deque<Line> dq;
    for (int i=0; i<SZ(fin); i++) {
        while(SZ(dq) >= 2 and
              not isin(fin[i], dq[SZ(dq)-2], dq[SZ(dq)
                  -1])) {
            dq.pop_back();
        }
        while(SZ(dq) >= 2 and
              not isin(fin[i], dq[0], dq[1])) {
            dq.pop_front();
        }
        dq.push_back(fin[i]);
    }

    while (SZ(dq) >= 3 and
           not isin(dq[0], dq[SZ(dq)-2], dq[SZ(dq)-1]))
                {
        dq.pop_back();
    }

    while (SZ(dq) >= 3 and
           not isin(dq[SZ(dq)-1], dq[0], dq[1])) {
        dq.pop_front();
    }
    vector<Line> res(ALL(dq));
    return res;
}
```

6.6 dao point

```
typedef double Type;
struct Point{
  Type x, y;
  Point(){};
  Point(Type _x, Type _y){
    x = _x, y = _y;
  }
  void read(){
    scanf("%lf %lf", &x, &y);
  }
  Point operator +(const Point & P2){
    return Point(x + P2.x, y + P2.y);
  }
  Point operator -(const Point & P2){
    return Point(x - P2.x, y - P2.y);
  }
  Point operator *(const Type & Len){
    return Point(x*Len, y*Len);
  }
  Type operator *(const Point & P2){
    return x*P2.x + y*P2.y;
  }
  Type operator ^(const Point & P2){
    return x*P2.y - y*P2.x;
  }
  Type dis(){
    return x*x+y*y;
  }
};
struct Line{
  Point s, e;
  Line(){};
  Line(Point _s, Point _e){
    s = _s, e = _e;
  }
  void read(){
    s.read(); e.read();
  }
};
```

6.7 dao inter

```
Point inter(Line l1, Line l2){
  Type v1 = (l1.s - l1.e) ^ (l2.s - l1.e);
  Type v2 = (l1.s - l1.e) ^ (l1.e - l2.e);
  Type v3 = (v1 + v2);
  if(v3 + eps > 0 && v3 - eps < 0) return Point(nan("")
      , nan(""));
  return l2.s*(v2/v3) + l2.e*(v1/v3);
}
```

6.8 dao 2D convex hull

```
int ori(Point s, Point e, Point P){
  Type val = (s - e)^(P - e);
  if(fabs(val) < eps) return 0;
  else if(val > 0) return 1;
  else return -1;
}
bool cmp(Point a, Point b){
  if(a.x != b.x) return a.x < b.x;
  return a.y < b.y;
}
vector<Point> convex_hull(vector<Point> pt){
  sort(pt.begin(), pt.end(), cmp);
  int top=0;
  vector<Point> stk(2*pt.size());
  for (int i=0; i<(int)pt.size(); i++){
    while (top >= 2 && ori(stk[top-2],stk[top-1],pt[i])
         >= 0)
      top--;
    stk[top++] = pt[i];
  }
  for (int i=pt.size()-2, t=top+1; i>=0; i--){
    while (top >= t && ori(stk[top-2],stk[top-1],pt[i])
         >= 0)
      top--;
    stk[top++] = pt[i];
  }
  stk.resize(top-1);
  return stk;
}
```

6.9 Minimum Covering Circle

```
struct Mcc{
  // return pair of center and r^2
  static const int MAXN = 1000100;
  int n;
  pdd p[MAXN],cen;
  double r2;

  void init(int _n, pdd _p[]){
    n = _n;
    memcpy(p,_p,sizeof(pdd)*n);
  }
  double sqr(double a){ return a*a; }
  double abs2(pdd a){ return a*a; }
  pdd center(pdd p0, pdd p1, pdd p2) {
    pdd a = p1-p0;
    pdd b = p2-p0;
    double c1=abs2(a)*0.5;
    double c2=abs2(b)*0.5;
    double d = a % b;
    double x = p0.x + (c1 * b.y - c2 * a.y) / d;
    double y = p0.y + (a.x * c2 - b.x * c1) / d;
    return pdd(x,y);
  }

  pair<pdd,double> solve(){
    random_shuffle(p,p+n);
    r2=0;
    for (int i=0; i<n; i++){
      if (abs2(cen-p[i]) <= r2) continue;
      cen = p[i];
      r2 = 0;
      for (int j=0; j<i; j++){
        if (abs2(cen-p[j]) <= r2) continue;
        cen = 0.5 * (p[i]+p[j]);
        r2 = abs2(cen-p[j]);
        for (int k=0; k<j; k++){
          if (abs2(cen-p[k]) <= r2) continue;
          cen = center(p[i],p[j],p[k]);
          r2 = abs2(cen-p[k]);
        }
      }
    }
    return {cen,r2};
  }
}mcc;
```

7 Stringology

7.1 Suffix Array

```
const int MAX = 1020304;
int ct[MAX], he[MAX], rk[MAX], sa[MAX], tsa[MAX], tp[
    MAX][2];

void suffix_array(char *ip){

  int len = strlen(ip);
  int alp = 256;
  memset(ct, 0, sizeof(ct));
  for(int i=0;i<len;i++) ct[ip[i]+1]++;
  for(int i=1;i<alp;i++) ct[i]+=ct[i-1];
  for(int i=0;i<len;i++) rk[i]=ct[ip[i]];

  for(int i=1;i<len;i*=2){
    for(int j=0;j<len;j++){
      if(j+i>=len) tp[j][1]=0;
      else tp[j][1]=rk[j+i]+1;

      tp[j][0]=rk[j];
    }
    memset(ct, 0, sizeof(ct));
    for(int j=0;j<len;j++) ct[tp[j][1]+1]++;
    for(int j=1;j<len+2;j++) ct[j]+=ct[j-1];
    for(int j=0;j<len;j++) tsa[ct[tp[j][1]]++]=j;

    memset(ct, 0, sizeof(ct));
    for(int j=0;j<len;j++) ct[tp[j][0]+1]++;
    for(int j=1;j<len+1;j++) ct[j]+=ct[j-1];
    for(int j=0;j<len;j++) sa[ct[tp[tsa[j]][0]]++]=tsa[
        j];

    rk[sa[0]]=0;
    for(int j=1;j<len;j++){
      if( tp[sa[j]][0] == tp[sa[j-1]][0] &&
        tp[sa[j]][1] == tp[sa[j-1]][1] )
        rk[sa[j]] = rk[sa[j-1]];
      else
        rk[sa[j]] = j;
    }
  }

  for(int i=0,h=0;i<len;i++){
    if(rk[i]==0) h=0;
    else{
      int j=sa[rk[i]-1];
      h=max(0,h-1);
      for(;ip[i+h]==ip[j+h];h++);
    }
    he[rk[i]]=h;
  }
}
```

7.2 KMP

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

void build_fail_function(string B, int *fail) {
    int len = B.length(), pos;
    pos = fail[0] = -1;
    for (int i = 1; i < len; i ++) {
        while (pos != -1 and B[pos + 1] != B[i])
            pos = fail[pos];
        if (B[pos + 1] == B[i]) pos ++;
        fail[i] = pos;
    }
}

void match(string A, string B, int *fail) {
    int lenA = A.length(), lenB = B.length();
    int pos = -1;
    for (int i = 0; i < lenA; i ++) {
        while (pos != -1 and B[pos + 1] != A[i])
            pos = fail[pos];

        if (B[pos + 1] == A[i]) pos ++;

        if (pos == lenB - 1) {
            // Match ! A[i - lenB + 1, i] = B
            pos = fail[pos];
        }
    }
}
```

7.3 Z value

```
void Zval(const char *s, int len, int *z) {
    z[0] = 0;
    for (int b=0, i=1; i<len; i++) {
        z[i] = max(min(z[i-b], z[b] + b - i), 0);
        while (s[i + z[i]] == s[z[i]]) z[i] ++;
        if (i+z[i] > b+z[b]) b=i;
    }
}
```

7.4 Lexicographically Smallest Rotation

```
string mcp(string s){
  int n = s.length();
  s += s;
  int i=0, j=1;
  while (i<n && j<n){
    int k = 0;
    while (k < n && s[i+k] == s[j+k]) k++;
    if (s[i+k] <= s[j+k]) j += k+1;
    else i += k+1;
    if (i == j) j++;
  }
  int ans = i < n ? i : j;
  return s.substr(ans, n);
}
```