Contents

1 Basic

1.1 .vimrc

```
syn on
se ai nu ru cul mouse=a
se cin et ts=2 sw=2 sts=2
so $VIMRUNTIME/mswin.vim
colo desert
se gfn=Monospace\ 14
noremap <buffer><F9> :! g++ -std=c++14 -O2 -Wall -
    Wshadow '%' -o '%<'<CR>
noremap <buffer><F5> :! './%<'<CR>
noremap <buffer><F6> :! './%<' < './%<.in'<CR>
noremap <buffer><F7> :! './%<' < './%<.in' > './%<.out'
    <CR>
```

2 Math

2.1 Euclidean's Algorithm

```
// a must be greater than b
vector< pair< int, int > > gcd( int a, int b ) {
  if ( b == 0 )
    return { 1, 0 };
  vector< pair< int, int > > q = gcd( b, b % a );
  return { q.second, q.first - q.second * ( a / b ) };
}
```

2.2 Big Integer

```
const int base = 1000000000;
const int base_digits = 9;

class Bigint {
  public:
    vector<int> a;
    int sign;

    Bigint(): sign(1) {}
    Bigint(long long v) { *this = v; }
    Bigint(const string &s) { read(s); }
    void operator=(const Bigint &v) { sign = v.sign; a
        = v.a; }
    void operator=(long long v) {
      sign = 1;
      if (v < 0)
        sign = -1, v = -v;
      for (; v > 0; v = v / base)
        a.push_back(v % base);
    }
    Bigint operator+(const Bigint &v) const {
      if (sign == v.sign) {
        Bigint res = v;
        for (int i = 0, carry = 0; i < (int) max(a.size
            (), v.a.size()) || carry; ++i) {
          if (i == (int) res.a.size())
            res.a.push_back(0);
          res.a[i] += carry + (i < (int) a.size() ? a[i
              ] : 0);
          carry = res.a[i] >= base;
          if (carry)
            res.a[i] -= base;
        }
        return res;
      }
      return *this - (-v);
    }
    Bigint operator-(const Bigint &v) const {
      if (sign == v.sign) {
        if (abs() >= v.abs()) {
          Bigint res = *this;
          for (int i = 0, carry = 0; i < (int) v.a.size
              () || carry; ++i) {
```

```
        res.a[i] -= carry + (i < (int) v.a.size() ?
            v.a[i] : 0);
        carry = res.a[i] < 0;
        if (carry)
          res.a[i] += base;
      }
      res.trim();
      return res;
    }
    return -(v - *this);
  }
  return *this + (-v);
}
void operator*=(int v) {
  if (v < 0)
    sign = -sign, v = -v;
  for (int i = 0, carry = 0; i < (int) a.size() ||
      carry; ++i) {
    if (i == (int) a.size())
      a.push_back(0);
    long long cur = a[i] * (long long) v + carry;
    carry = (int) (cur / base);
    a[i] = (int) (cur % base);
  }
  trim();
}
Bigint operator*(int v) const {
  Bigint res = *this;
  res *= v;
  return res;
}

friend pair<Bigint, Bigint> divmod(const Bigint &a1
    , const Bigint &b1) {
  int norm = base / (b1.a.back() + 1);
  Bigint a = a1.abs() * norm;
  Bigint b = b1.abs() * norm;
  Bigint q, r;
  q.a.resize(a.a.size());

  for (int i = a.a.size() - 1; i >= 0; i--) {
    r *= base;
    r += a.a[i];
    int s1 = r.a.size() <= b.a.size() ? 0 : r.a[b.a
        .size()];
    int s2 = r.a.size() <= b.a.size() - 1 ? 0 : r.a
        [b.a.size() - 1];
    int d = ((long long) base * s1 + s2) / b.a.back
        ();
    r -= b * d;
    while (r < 0)
      r += b, --d;
    q.a[i] = d;
  }

  q.sign = a1.sign * b1.sign;
  r.sign = a1.sign;
  q.trim();
  r.trim();
  return make_pair(q, r / norm);
}

Bigint operator/(const Bigint &v) const {
  return divmod(*this, v).first;
}

Bigint operator%(const Bigint &v) const {
  return divmod(*this, v).second;
}

void operator/=(int v) {
  if (v < 0)
    sign = -sign, v = -v;
  for (int i = (int) a.size() - 1, rem = 0; i >= 0;
      --i) {
    long long cur = a[i] + rem * (long long) base;
    a[i] = (int) (cur / v);
    rem = (int) (cur % v);
  }
  trim();
}
Bigint operator/(int v) const {
  Bigint res = *this;
  res /= v;
  return res;
}
int operator%(int v) const {
  if (v < 0)
    v = -v;
  int m = 0;
  for (int i = a.size() - 1; i >= 0; --i)
    m = (a[i] + m * (long long) base) % v;
  return m * sign;
}

void operator+=(const Bigint &v) { *this = *this +
    v; }
void operator-=(const Bigint &v) { *this = *this -
    v; }
void operator*=(const Bigint &v) { *this = *this *
    v; }
void operator/=(const Bigint &v) { *this = *this /
    v; }

bool operator<(const Bigint &v) const {
  if (sign != v.sign)
    return sign < v.sign;
  if (a.size() != v.a.size())
    return a.size() * sign < v.a.size() * v.sign;
  for (int i = a.size() - 1; i >= 0; i--)
    if (a[i] != v.a[i])
      return a[i] * sign < v.a[i] * sign;
  return false;
}

bool operator>(const Bigint &v) const { return v <
    *this; }
bool operator<=(const Bigint &v) const { return !(v
    < *this); }
bool operator>=(const Bigint &v) const { return !(*
    this < v); }
bool operator==(const Bigint &v) const { return !(*
    this < v) && !(v < *this); }
bool operator!=(const Bigint &v) const { return *
    this < v || v < *this; }

void trim() {
  while (!a.empty() && !a.back())
    a.pop_back();
  if (a.empty())
    sign = 1;
}
bool isZero() const {
  return a.empty() || (a.size() == 1 && !a[0]);
}
Bigint operator-() const {
  Bigint res = *this;
  res.sign = -sign;
  return res;
}
Bigint abs() const {
  Bigint res = *this;
  res.sign *= res.sign;
  return res;
}
long long longValue() const {
  long long res = 0;
  for (int i = a.size() - 1; i >= 0; i--)
    res = res * base + a[i];
  return res * sign;
}
friend Bigint gcd(const Bigint &a, const Bigint &b)
    {
  return b.isZero() ? a : gcd(b, a % b);
}
friend Bigint lcm(const Bigint &a, const Bigint &b)
    {
  return a / gcd(a, b) * b;
}
void read(const string &s) {
  sign = 1;
  a.clear();
  int pos = 0;
```

```
    while (pos < (int) s.size() && (s[pos] == '-' ||
        s[pos] == '+')) {
      if (s[pos] == '-')
        sign = -sign;
      ++pos;
    }
    for (int i = s.size() - 1; i >= pos; i -=
        base_digits) {
      int x = 0;
      for (int j = max(pos, i - base_digits + 1); j
          <= i; j++)
        x = x * 10 + s[j] - '0';
      a.push_back(x);
    }
    trim();
  }
  friend istream& operator>>(istream &stream, Bigint
      &v) {
    string s;
    stream >> s;
    v.read(s);
    return stream;
  }
  friend ostream& operator<<(ostream &stream, const
      Bigint &v) {
    if (v.sign == -1)
      stream << '-';
    stream << (v.a.empty() ? 0 : v.a.back());
    for (int i = (int) v.a.size() - 2; i >= 0; --i)
      stream << setw(base_digits) << setfill('0') <<
          v.a[i];
    return stream;
  }
  static vector<int> convert_base(const vector<int> &
      a, int old_digits, int new_digits) {
    vector<long long> p(max(old_digits, new_digits) +
        1);
    p[0] = 1;
    for (int i = 1; i < (int) p.size(); i++)
      p[i] = p[i - 1] * 10;
    vector<int> res;
    long long cur = 0;
    int cur_digits = 0;
    for (int i = 0; i < (int) a.size(); i++) {
      cur += a[i] * p[cur_digits];
      cur_digits += old_digits;
      while (cur_digits >= new_digits) {
        res.push_back(int(cur % p[new_digits]));
        cur /= p[new_digits];
        cur_digits -= new_digits;
      }
    }
    res.push_back((int) cur);
    while (!res.empty() && !res.back())
      res.pop_back();
    return res;
  }
  typedef vector<long long> vll;
  static vll karatsubaMultiply(const vll &a, const
      vll &b) {
    int n = a.size();
    vll res(n + n);
    if (n <= 32) {
      for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
          res[i + j] += a[i] * b[j];
      return res;
    }
    int k = n >> 1;
    vll a1(a.begin(), a.begin() + k);
    vll a2(a.begin() + k, a.end());
    vll b1(b.begin(), b.begin() + k);
    vll b2(b.begin() + k, b.end());

    vll a1b1 = karatsubaMultiply(a1, b1);
    vll a2b2 = karatsubaMultiply(a2, b2);

    for (int i = 0; i < k; i++)
      a2[i] += a1[i];
    for (int i = 0; i < k; i++)
      b2[i] += b1[i];

    vll r = karatsubaMultiply(a2, b2);
    for (int i = 0; i < (int) a1b1.size(); i++)
      r[i] -= a1b1[i];
    for (int i = 0; i < (int) a2b2.size(); i++)
      r[i] -= a2b2[i];

    for (int i = 0; i < (int) r.size(); i++)
      res[i + k] += r[i];
    for (int i = 0; i < (int) a1b1.size(); i++)
      res[i] += a1b1[i];
    for (int i = 0; i < (int) a2b2.size(); i++)
      res[i + n] += a2b2[i];
    return res;
  }
  Bigint operator*(const Bigint &v) const {
    vector<int> a6 = convert_base(this->a,
        base_digits, 6);
    vector<int> b6 = convert_base(v.a, base_digits,
        6);
    vll a(a6.begin(), a6.end());
    vll b(b6.begin(), b6.end());
    while (a.size() < b.size())
      a.push_back(0);
    while (b.size() < a.size())
      b.push_back(0);
    while (a.size() & (a.size() - 1))
      a.push_back(0), b.push_back(0);
    vll c = karatsubaMultiply(a, b);
    Bigint res;
    res.sign = sign * v.sign;
    for (int i = 0, carry = 0; i < (int) c.size(); i
        ++) {
      long long cur = c[i] + carry;
      res.a.push_back((int) (cur % 1000000));
      carry = (int) (cur / 1000000);
    }
    res.a = convert_base(res.a, 6, base_digits);
    res.trim();
    return res;
  }
};
```

3 Data Structure

3.1 Disjoint Set

```
class DisjointSet {
 public:
  static const int N = 1e5 + 10;
  int p[ N ];
  void Init( int x ) {
    for ( int i = 1 ; i <= x ; ++i )
      p[ i ] = i;
  }
  int Find( int x ) {
    return x == p[ x ] ? x : p[ x ] = Find( p[ x ] );
  }
  void Union( int x, int y ) {
    p[ Find( x ) ] = Find( y );
  }
};
```

3.2 Segement Tree with Lazy Tag

```
#define L(X) (X<<1)
#define R(X) ((X<<1)+1)
#define mid ((l+r)>>1)

class SegmentTree {
 public:
  static const int N = 1e5 + 10;
  int arr[ N ], st[ N << 2 ], lazy[ N << 2 ];

  inline void Pull( int now ) {
    st[ now ] = max( st[ L( now ) ], st[ R( now ) ] );
  }
```

```
  inline void Push( int now, int l, int r ) {
    if ( lazy[ now ] != 0 ) {
      if ( l != r ) {
        st[ L( now ) ] += lazy[ now ];
        st[ R( now ) ] += lazy[ now ];
        lazy[ L( now ) ] += lazy[ now ];
        lazy[ R( now ) ] += lazy[ now ];
      }
      lazy[ now ] = 0;
    }
  }
  void Build( int now, int l, int r ) {
    if ( l == r ) {
      st[ now ] = arr[ l ];
      return;
    }
    Build( L( now ), l, mid );
    Build( R( now ), mid + 1, r );
    Pull( now );
  }
  void Update( int ql, int qr, int value, int now, int
      l, int r ) {
    if ( ql > qr || l > qr || r < ql )
      return;
    Push( now, l, r );
    if ( l == ql && qr == r ) {
      st[ now ] += value;
      lazy[ now ] += value;
      return;
    }
    if ( qr <= mid ) Update( ql, qr, value, L( now ), l
        , mid );
    else if ( mid < ql ) Update( ql, qr, value, R( now
        ), mid + 1, r );
    else {
      Update( ql, mid, value, L( now ), l, mid );
      Update( mid + 1, qr, value, R( now ), mid + 1, r
          );
    }
    Pull( now );
  }
  int Query( int ql, int qr, int now, int l, int r ) {
    if ( ql > qr || l > qr || r < ql )
      return 0;
    Push( now, l, r );
    if ( l == ql && qr == r )
      return st[ now ];
    if ( qr <= mid )
      return Query( ql, qr, L( now ), l, mid );
    else if ( mid < ql )
      return Query( ql, qr, R( now ), mid + 1, r );
    else {
      int left = Query( ql, mid, L( now ), l, mid );
      int right = Query( mid + 1, qr, R( now ), mid +
          1, r );
      int ans = max( left, right );
      return ans;
    }
  }
};
```

3.3 Rope

```
#include<ext/rope>
using namespace __gnu_cxx;
// inserts c before p.
iterator insert(const iterator& p, charT c) :
// inserts n copies of c before p.
iterator insert(const iterator& p, size_t n, charT c) :
// inserts the character c before the ith element.
void insert(size_t i, charT c) :
// erases the element pointed to by p.
void erase(const iterator& p) :
// erases the range [f, l).
void erase(const iterator& f, const iterator& l) :
// Appends a C string.
void append(const charT* s) :
void replace(const iterator& f, const iterator& l,
    const rope& x)
void replace(const iterator& f, const iterator& l,
    const charT* s)
void replace(const iterator& f1, const iterator& l1,
    const charT* f2, const charT* l2)
void replace(const iterator& f1, const iterator& l1,
    const iterator& f2, const iterator& l2)
void replace(const iterator& p, const rope& x)
void replace(size_t i, size_t n, const rope& x)
void replace(size_t i, size_t n, charT c)
void replace(size_t i, size_t n, const charT* f, const
    charT* l)
void replace(size_t i, size_t n, const iterator& f,
    const iterator& l)
rope substr(iterator f, iterator l) const
rope substr(const_iterator f, const_iterator l) const
rope substr(size_t i, size_t n = 1) const
```

4 graph

4.1 Dijkstra's Algorithm

```
vector< pair< int, int > > v[ N ];

vector< int > Dijkstra( int s ) {
  // n: number of nodes
  vector< int > d( n + 1, 1e9 );
  vector< bool > visit( n + 1, false );
  d[ s ] = 0;

  priority_queue< pair< int, int >, vector< pair< int,
      int > >, greater< pair< int, int > > > pq;
  pq.push( make_pair( d[ s ], s ) );
  while ( 1 ) {
    int now = -1;
    while ( !pq.empty() and visit[ now = pq.top().
        second ] )
      pq.pop();
    if ( now == -1 or visit[ now ] )
      break;
    visit[ now ] = true;
    for ( int i = 0 ; i < v[ now ].size() ; ++i ) {
      int child = v[ now ][ i ].first;
      int w = v[ now ][ i ].second;
      if ( !visit[ child ] and ( d[ now ] + w ) < d[
          child ] ) {
        d[ child ] = d[ now ] + w;
        pq.push( make_pair( d[ child ], child ) );
      }
    }
  }
  return d;
}
```

4.2 Tarjan's Algorithm

```
// Build: O( V^2 ), Query: O( 1 )
// n: the number of nodes
int graph[ N ][ N ], lca[ N ][ N ];
vector< bool > visit( N, false );

void tarjan( int now ) {
  if ( visit[ now ] )
    return;
  visit[ now ] = true;

  for ( int i = 1 ; i <= n ; ++i )
    if ( visit[ i ] )
      lca[ now ][ i ] = lca[ i ][ now ] = st.Find( i );

  for ( int i = 1 ; i <= n ; ++i )
    if ( g[ now ][ i ] < 1e9 and !visit[ i ] ) {
      tarjan( i );
      st.Union( i, now );
    }
}
```

4.3 Jump Pointer Algorithm

```
int tin[ N ], tout[ N ], ancestor[ N ][ 20 ];
vector< int > v[ N ];

void dfs( int now, int pnow ) {
  tin[ now ] = ++now_time;

  ancestor[ now ][ 0 ] = pnow;
  for ( int i = 1 ; i < 20 ; ++i )
    ancestor[ now ][ i ] = ancestor[ ancestor[ now ][ i
        - 1 ] ][ i - 1 ];

  for ( auto child : v[ now ] )
    if ( child != pnow )
      dfs( child, now );

  tout[ now ] = ++now_time;
}
bool check_ancestor( int x, int y ) {
  return ( tin[ x ] <= tin[ y ] and tout[ x ] >= tout[
      y ] );
}
int find_lca( int x, int y ) {
  if ( check_ancestor( x, y ) ) return x;
  if ( check_ancestor( y, x ) ) return y;

  for ( int i = 19 ; i >= 0 ; --i )
    if ( !check_ancestor( ancestor[ x ][ i ], y ) )
      x = ancestor[ x ][ i ];
  return ancestor[ x ][ 0 ];
}
```

5 Flow

5.1 MinCostMaxFlow

```
// O( V^2 * F )
class MinCostMaxFlow{
 public:
  static const int MAXV = 2000;
  static const int INF  = 1e9;
  struct Edge{
    int v, cap, w, rev;
    Edge(){}
    Edge(int t2, int t3, int t4, int t5)
    : v(t2), cap(t3), w(t4), rev(t5) {}
  };
  int V, s, t;
  vector<Edge> g[MAXV];
  void Init(int n){
    V = n+4; // total number of nodes
    s = n+1, t = n+4; // s = source, t = sink
    for(int i = 1; i <= V; i++) g[i].clear();
  }
  // cap: capacity, w: cost
  void AddEdge(int a, int b, int cap, int w){
    g[a].push_back(Edge(b, cap, w, (int)g[b].size()));
    g[b].push_back(Edge(a, 0, -w, (int)g[a].size()-1));
  }
  int d[MAXV], id[MAXV], mom[MAXV];
  bool inqu[MAXV];
  int qu[2000000], ql, qr;
  //the size of qu should be much large than MAXV
  int MncMxf(){
    int INF = INF;
    int mxf = 0, mnc = 0;
    while(1){
      fill(d+1, d+1+V, INF);
      fill(inqu+1, inqu+1+V, 0);
      fill(mom+1, mom+1+V, -1);
      mom[s] = s;
      d[s] = 0;
      ql = 1, qr = 0;
      qu[++qr] = s;
      inqu[s] = 1;
      while(ql <= qr){
        int u = qu[ql++];
        inqu[u] = 0;
        for(int i = 0; i < (int) g[u].size(); i++){
          Edge &e = g[u][i];
          int v = e.v;
          if(e.cap > 0 && d[v] > d[u]+e.w){
            d[v] = d[u]+e.w;
            mom[v] = u;
            id[v] = i;
            if(!inqu[v]) qu[++qr] = v, inqu[v] = 1;
          }
        }
      }
      if(mom[t] == -1) break ;
      int df = INF;
      for(int u = t; u != s; u = mom[u])
        df = min(df, g[mom[u]][id[u]].cap);
      for(int u = t; u != s; u = mom[u]){
        Edge &e = g[mom[u]][id[u]];
        e.cap              -= df;
        g[e.v][e.rev].cap += df;
      }
      mxf += df;
      mnc += df*d[t];
    }
    return mnc;
  }
};
```