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 ) };
}
```

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;
    }
  }
};
```

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;
  }
};
```