

Contents

1 Basic

1.1 vimrc

```
set nu ai si cin ts=4 sw=4 sts=4 mouse=a expandtab
syn on
imap {<CR> {<CR>}<Esc>ko
map <F5> :w<LF>:!g++ -O2 -std=c++11 % && echo "----
    Start-----" && ./a.out<LF>
map <F6> :w<LF>:!g++ -O2 -std=c++11 % && echo "----
    Start-----" && time ./a.out < input.in<LF>
map <F9> :t!e %<LF>
```

1.2 int128

```
__int128 parse(string &s) {
    __int128 ret = 0;
    for (int i = 0; i < (int)s.size(); i++)
        if ('0' <= s[i] && s[i] <= '9')
            ret = 10 * ret + s[i] - '0';
    return ret;
}
#define O ostream
O& operator << (O &out, __int128_t v) {
    O::sentry s(out);
    if (s) {
        __uint128_t uv = v < 0 ? -v : v;
        char buf[128], *d = end(buf);
        do {
            *(&d) = "0123456789"[uv % 10];
            uv /= 10;
        } while (uv != 0);
        if (uv < 0)
            *(&d) = '-';
        int len = end(buf) - d;
        if (out.rdbuf()->sputn(d, len) != len)
            out.setstate(ios_base::badbit);
    }
    return out;
}
#define I istream
I& operator >> (I &in, __int128_t &v) {
    string s; in >> s;
    v = parse(s);
    return in;
}
```

2 Flow

2.1 Dinic

```
struct Graph{
    struct Node; struct Edge;
    int V;
    struct Node : vector<Edge*>{
        iterator cur; int d;
        Node(){ clear(); }
    }_memN[MAXN], *node[MAXN];
    struct Edge{
        Node *u, *v;
        Edge *rev;
        LL c, f;
        Edge(){ }
        Edge(Node *u, Node *v, LL c, Edge *rev) : u(u), v(v),
            c(c), f(0), rev(rev){ }
    }_memE[MAXM], *ptrE;
```

```
Graph(int _V) : V(_V) {
    for (int i = 0; i < V; i++)
        node[i] = _memN + i;
    ptrE = _memE;
}
void addEdge(int _u, int _v, LL _c){
    *ptrE = Edge(node[_u], node[_v], _c, ptrE + 1);
    node[_u]->push_back(ptrE++);
    *ptrE = Edge(node[_v], node[_u], _c, ptrE - 1); //
        direction
    node[_v]->push_back(ptrE++);
}
Node *s, *t;
LL maxFlow(int _s, int _t){
    s = node[_s], t = node[_t];
    LL flow = 0;
    while (bfs()) {
        for (int i = 0; i < V; i++)
            node[i]->cur = node[i]->begin();
        flow += dfs(s, INF);
    }
    return flow;
}
bool bfs(){
    for (int i = 0; i < V; i++) node[i]->d = -1;
    queue<Node*> q; q.push(s); s->d = 0;
    while (q.size()) {
        Node *u = q.front(); q.pop();
        for (auto e : *u) {
            Node *v = e->v;
            if (!v->d && e->c > e->f)
                q.push(v), v->d = u->d + 1;
        }
    }
    return ~t->d;
}
LL dfs(Node *u, LL a){
    if (u == t || !a) return a;
    LL flow = 0, f;
    for (; u->cur != u->end(); u->cur++) {
        auto &e = *u->cur; Node *v = e->v;
        if (u->d + 1 == v->d && (f = dfs(v, min(a, e->c -
            e->f))) > 0) {
            e->f += f; e->rev->f -= f;
            flow += f; a -= f;
            if (!a) break;
        }
    }
    return flow;
}
};
```

2.2 MCMF

```
struct Graph {
    struct Node; struct Edge; int V;
    struct Node : vector<Edge*> {
        bool inq; Edge *pa; LL a, d;
        Node() { clear(); }
    }_memN[MAXN], *node[MAXN];
    struct Edge{
        Node *u, *v; Edge *rev;
        LL c, f, _c; Edge() {}
        Edge(Node *u, Node *v, LL c, LL _c, Edge *rev)
            : u(u), v(v), c(c), f(0), _c(_c), rev(rev) {}
    }_memE[MAXM], *ptrE;
    Graph(int _V) : V(_V) {
        for (int i = 0; i < V; i++)
            node[i] = _memN + i;
        ptrE = _memE;
    }
    void addEdge(int u, int v, LL c, LL _c) {
        *ptrE = Edge(node[u], node[v], c, _c, ptrE + 1);
        node[u]->push_back(ptrE++);
        *ptrE = Edge(node[v], node[u], 0, -_c, ptrE - 1);
```

