ACM 代码册

EDITED BY

PUFANYI

目录

第一章 搜索

1.1 Dancing Links

```
const int maxn = 505;
const int maxm = 6005;

struct Dancing_Links {
  int n, m, total, ans;

  struct Node {
    int up, down, left, right, row, column;
  } no[maxm];

  int siz[maxn];
  int first[maxn];
  int stk[maxn];

  void init(int n, int m) {
    ans = 0;
    this->n = n, this->m = m;
    memset(first, 0, sizeof(first));
    memset(siz, 0, sizeof(siz));
    for (int i = 0; i <= m; ++i) {
      no[i].left = i - 1, no[i].right = i + 1;
      no[i].up = no[i].down = i;
    }
    no[0].left = m, no[m].right = 0, total = m;
  }

  void insert(int row, int col) {
    total++, siz[col]++;
    no[total].row = row, no[total].column = col;
    no[total].down = col, no[total].up = no[col].up;
    no[col].up = total, no[no[total].up].down = total;
    if (!first[row]) {
      first[row] = no[total].left = no[total].right = total;
    } else {
      no[total].right = first[row], no[total].left = no[first[row]].left;
```

```
      no[no[total].left].right = no[first[row]].left = total;
    }
  }

  void remove(int col) {
    no[no[col].left].right = no[col].right;
    no[no[col].right].left = no[col].left;
    for (int i = no[col].down; i != col; i = no[i].down) {
      for (int j = no[i].right; j != i; j = no[j].right) {
        no[no[j].up].down = no[j].down;
        no[no[j].down].up = no[j].up;
        siz[no[j].column]--;
      }
    }
  }

  void recover(int col) {
    for (int i = no[col].up; i != col; i = no[i].up) {
      for (int j = no[i].left; j != i; j = no[j].left) {
        no[no[j].up].down = no[no[j].down].up = j;
        siz[no[j].column]++;
      }
    }
    no[no[col].left].right = no[no[col].right].left = col;
  }

  bool dance(int dep) {
    if (!no[0].right) {
      ans = dep - 1;
      return true;
    }
    int col = no[0].right;
    for (int i = no[0].right; i; i = no[i].right) {
      if (siz[i] < siz[col]) {
        col = i;
      }
    }
    remove(col);
    for (int i = no[col].down; i != col; i = no[i].down) {
      stk[dep] = no[i].row;
      for (int j = no[i].right; j != i; j = no[j].right) {
        remove(no[j].column);
      }
      if (dance(dep + 1)) {
        return true;
      }
      for (int j = no[i].left; j != i; j = no[j].left) {
```

```
        recover(no[j].column);
      }
    }
    recover(col);
    return false;
  }
} dlx;

int main() {
  int n, m, x;
  read(n), read(m);
  dlx.init(n, m);
  for (int i = 1; i <= n; ++i) {
    for (int j = 1; j <= m; ++j) {
      if (read(x) && x) {
        dlx.insert(i, j);
      }
    }
  }
  if (dlx.dance(1)) {
    for (int i = 1; i <= dlx.ans; ++i) {
      writesp(dlx.stk[i]);
    }
    puts("");
  } else {
    puts("No Solution!");
  }
  return 0;
}
```

1.2　$\alpha-\beta$ 剪枝

第二章 字符串

2.1 KMP

```
std::vector<int> kmp(std::string s) {
  int n = s.length();
  std::vector<int> pi(n);
  for (int i = 1; i < n; ++i) {
    int j = pi[i - 1];
    while (j && s[i] != s[j]) {
      j = pi[j - 1];
    }
    if (s[i] == s[j]) {
      j++;
    }
    pi[i] = j;
  }
  return pi;
}
```

2.2 Z-function

```
std::vector<int> z_function(std::string s) {
  int n = s.length();
  std::vector<int> z(n);
  z[0] = n;
  for (int i = 1, l = 0, r = 0; i < n; ++i) {
    if (i <= r && z[i - l] < r - i + 1) {
      z[i] = z[i - l];
    } else {
      z[i] = std::max(0, r - i + 1);
      while (i + z[i] < n && s[z[i]] == s[i + z[i]]) {
        z[i]++;
      }
    }
    if (i + z[i] - 1 > r) {
      l = i, r = i + z[i] - 1;
    }
```

```
  }
  return z;
}
```

2.3 AC 自动机

```
const int maxn = 200005;

int ans[maxn];

struct Aho_Corasick {
  std::vector<int> id[maxn];
  int son[maxn][26];
  int fail[maxn];
  int val[maxn];
  int cnt;

  Aho_Corasick() {
    cnt = 0;
    memset(son, 0, sizeof(son));
    memset(fail, 0, sizeof(fail));
    memset(val, 0, sizeof(val));
  }

  void insert(std::string s, int _id) {
    int now = 0;
    for (auto c : s) {
      const int x = c - 'a';
      if (!son[now][x]) {
        son[now][x] = ++cnt;
      }
      now = son[now][x];
    }
    id[now].push_back(_id);
  }

  std::vector<int> fas[maxn];

  void build() {
    std::queue<int> q;
    for (int i = 0; i < 26; ++i) {
      if (son[0][i]) {
        q.push(son[0][i]);
      }
    }
    while (!q.empty()) {
```

```
      int now = q.front();
      q.pop();
      for (int i = 0; i < 26; ++i) {
        if (son[now][i]) {
          fail[son[now][i]] = son[fail[now]][i];
          q.push(son[now][i]);
        } else {
          son[now][i] = son[fail[now]][i];
        }
      }
    }
  }

  void getval(std::string s) {
    int now = 0;
    for (auto c : s) {
      now = son[now][c - 'a'];
      val[now]++;
    }
  }

  void build_fail_tree() {
    for (int i = 1; i <= cnt; ++i) {
      fas[fail[i]].push_back(i);
    }
  }

  void dfs(int now = 0) {
    for (auto x : fas[now]) {
      dfs(x);
      val[now] += val[x];
    }
    if (!id[now].empty()) {
      for (auto x : id[now]) {
        ans[x] = val[now];
      }
    }
  }
};

Aho_Corasick ac;

int n;

int main() {
  std::cin >> n;
  for (int i = 1; i <= n; ++i) {
```

```
    std::string s;
    std::cin >> s;
    ac.insert(s, i);
  }
  ac.build();
  std::string s;
  std::cin >> s;
  ac.getval(s);
  ac.build_fail_tree();
  ac.dfs();
  for (int i = 1; i <= n; ++i) {
    std::cout << ans[i] << std::endl;
  }
  return 0;
}
```

第三章 数学

3.1 快速幂

```
template <class T>
T ksm(T a, T b, T mod) {
  T ans = 1;
  for (; b; b >>= 1, a = (LL) a * a % mod) {
    if (b & 1) {
      ans = (LL) ans * a % mod;
    }
  }
  return ans;
}
```

3.2 位运算

3.2.1 Gray 码

```
int g(int n) {
  return n ^ (n >> 1);
}

int rev_g(int g) {
  int n = 0;
  for (; g; g >>= 1) {
    n ^= g;
  }
  return n;
}
```

3.3 数论

3.3.1 最大公约数

3.3.2 欧几里得算法

```cpp
template <class T>
T gcd(T a, T b) {
  while (b) {
    int t = a % b;
    a = b;
    b = t;
  }
  return a;
}
```

3.3.3 筛法

Eratosthenes 筛法

Euler 筛法

```cpp
void Euler(const int n = 100000) {
  np[1] = true;
  int cnt = 0;
  for (int i = 2; i <= n; ++i) {
    if (!np[i]) {
      prime[++cnt] = i;
    }
    for (int j = 1; j <= cnt && (LL) i * prime[j] <= n; ++j) {
      np[i * prime[j]] = true;
      if (!(i % prime[j])) {
        break;
      }
    }
  }
}
```

3.3.4 EXCRT

```cpp
LL CRT(int k, LL* a, LL* r) {
  LL n = 1, ans = 0;
  for (int i = 1; i <= k; i++) n = n * r[i];
  for (int i = 1; i <= k; i++) {
    LL m = n / r[i], b, y;
    exgcd(m, r[i], b, y);  // b * m mod r[i] = 1
    ans = (ans + a[i] * m * b % mod) % mod;
  }
  return (ans % mod + mod) % mod;
}
```

3.3.5 Lucas

$$\binom{n}{m} \bmod p = \binom{\lfloor n/p \rfloor}{\lfloor m/p \rfloor} \cdot \binom{n \bmod p}{m \bmod p} \bmod p$$

3.3.6 Pollard-Rho

```
typedef unsigned long long ULL;
typedef long long LL;

std::set<int> ans;

inline ULL rnd() {
  static ULL seed = 2333;
  seed ^= seed << 40;
  seed ^= seed >> 23;
  seed ^= seed << 7;
  return seed;
}

template <typename T>
inline T gcd(T a, T b) {
  while (b) {
    T t = a % b;
    a = b;
    b = t;
  }
  return a < 0 ? -a : a;
}

template <typename T>
inline void add(T& x, T y, T mod) {
  x += y;
  if (x >= mod) {
    x -= mod;
  } else if (x < 0) {
    x += mod;
  }
}

inline LL cheng(LL a, LL b, LL mod) {
  LL tmp = ((long double) a * b + .5) / mod;
  return ((a * b - tmp * mod) % mod + mod) % mod;
}

inline LL ksm(LL a, LL b, LL mod) {
  LL ans = 1;
```

```
  for (; b; b >>= 1, a = cheng(a, a, mod)) {
    if (b & 1) {
      ans = cheng(ans, a, mod);
    }
  }
  return ans;
}

inline bool witness(LL a, LL n) {
  LL u = n - 1;
  int t = 0;
  while (!(u & 1)) {
    u >>= 1;
    t++;
  }
  LL x = ksm(a, u, n);
  for (int i = 1; i <= t; ++i) {
    LL lstx = x;
    x = cheng(x, x, n);
    if (x == 1 && lstx != 1 && lstx != n - 1) {
      return false;
    }
  }
  if (x != 1) {
    return false;
  }
  return true;
}

inline bool MR(LL n) {
  if (n == 2) {
    return true;
  }
  static const int s = 5;
  for (int i = 1; i <= s; ++i) {
    if (!witness(rnd() % (n - 1) + 1, n)) {
      return false;
    }
  }
  return true;
}

inline LL rho(LL n) {
  if (MR(n)) {
    return n;
  }
  LL x = rnd() % n;
```

```
  LL y = x;
  LL p = (n & 1) ? 1 : 2;
  while (p == 1) {
    LL cc = rnd() % n;
    while (true) {
      int bitt = 127;
      LL xx = 1;
      while (bitt--) {
        x = cheng(x, x, n);
        add(x, cc, n);
        y = cheng(y, y, n);
        add(y, cc, n);
        y = cheng(y, y, n);
        add(y, cc, n);
        if (x == y) {
          break;
        }
        LL tx = (__int128) xx * (y - x) % n;
        if (tx) {
          xx = tx;
        } else {
          break;
        }
      }
      LL d = gcd((LL) xx, n);
      if (d != 1 && d != n) {
        p = d;
        break;
      }
      if (x == y) {
        break;
      }
    }
  }
  return std::max(rho(p), rho(n / p));
}

inline void solve() {
  LL n;
  read(n);
  if (MR(n)) {
    puts("Prime");
  } else {
    writeln(rho(n));
  }
}
```

第四章　数据结构

4.1　动态树

4.1.1　Link-Cut Tree

```
#include <cstdio>
#include <iostream>
#include <algorithm>

using namespace std;

const int maxn = 300005;

class LCT {
  // node

 public:
  int sum[maxn], val[maxn];
  int s[maxn][2], fa[maxn];

 private:
  bool lzy_fan[maxn];

  void push_up(int x) {
    sum[x] = val[x] ^ sum[s[x][0]] ^ sum[s[x][1]];
  }

  bool nrt(int x) {
    return s[fa[x]][0] == x || s[fa[x]][1] == x;
  }

  void fan(int x) {
    swap(s[x][0], s[x][1]);
    lzy_fan[x] ^= 1;
  }

  void push_down(int x) {
    if (lzy_fan[x]) {
```

```
      if (s[x][0]) {
        fan(s[x][0]);
      }
      if (s[x][1]) {
        fan(s[x][1]);
      }
      lzy_fan[x] = 0;
    }
  }

  // splay
 private:
  void rotate(int x) {
    int y = fa[x], z = fa[y];
    int k = (s[y][1] == x), ss = s[x][!k];
    if (nrt(y)) {
      s[z][s[z][1] == y] = x;
    }
    fa[x] = z;
    s[x][!k] = y;
    fa[y] = x;
    s[y][k] = ss;
    if (ss) {
      fa[ss] = y;
    }
    push_up(y);
    push_up(x);
  }

  int sta[maxn];
  void splay(int x) {
    int K = x, top = 0;
    sta[++top] = K;
    while (nrt(K)) {
      sta[++top] = K = fa[K];
    }
    while (top) {
      push_down(sta[top--]);
    }
    while (nrt(x)) {
      int y = fa[x], z = fa[y];
      if (nrt(y)) {
        rotate(((s[y][0] == x) ^ (s[z][0] == y)) ? x : y);
      }
      rotate(x);
    }
  }
```

```

  // LCT
private:
  void access(int x) {
    for (int y = 0; x; x = fa[y = x]) {
      splay(x);
      s[x][1] = y;
      push_up(x);
    }
  }

  void make_root(int x) {
    access(x);
    splay(x);
    fan(x);
  }

  int find_root(int x) {
    access(x);
    splay(x);
    while (s[x][0]) {
      push_down(x);
      x = s[x][0];
    }
    splay(x);
    return x;
  }

  void split(int x, int y) {
    make_root(x);
    access(y);
    splay(y);
  }

public:
  void link(int x, int y) {
    make_root(x);
    if (find_root(y) != x) {
      fa[x] = y;
    }
  }

  void cut(int x, int y) {
    make_root(x);
    if (find_root(y) == x && fa[y] == x && !s[y][0]) {
      fa[y] = s[x][1] = 0;
      push_up(x);
```

```
    }
  }

  void change(int x, int y) {
    splay(x);
    val[x] = y;
    push_up(x);
  }

  int ask(int x, int y) {
    split(x, y);
    return sum[y];
  }
} tr;

int main() {
  int n, m;
  scanf("%d%d", &n, &m);
  for (int i = 1; i <= n; ++i) {
    scanf("%d", &tr.val[i]);
    tr.sum[i] = tr.val[i];
  }
  while (m--) {
    int cmd, x, y;
    scanf("%d%d%d", &cmd, &x, &y);
    switch (cmd) {
      case 0:
        printf("%d\n", tr.ask(x, y));
        break;
      case 1:
        tr.link(x, y);
        break;
      case 2:
        tr.cut(x, y);
        break;
      case 3:
        tr.change(x, y);
    }
  }
  return 0;
}
```

第五章　图论

5.1　生成树

5.1.1　矩阵树

假设给出图为 G，定义一个 $n \times n$ 的矩阵 $D(G)$ 表示 G 个点的度数，当 $i \neq j$ 时，$d_{i,j} = 0$，当 $i = j$ 时，$d_{i,j}$ 等于节点 i 的度数。再定义一个 $n \times n$ 的矩阵 A_G 表示 G 的邻接矩阵，$A_{i,j}$ 表示 i 到 j 的边数。然后我们定义基尔霍夫矩阵 $C(G) = D(G) - A(G)$。则 G 中生成树个数等于 $C(G)$ 中任意一个 $n-1$ 阶主子式的行列式的绝对值。所谓一个矩阵 M 的 $n-1$ 阶主子式就是对于两个整数 $r\,(1 \leq r \leq n)$，将 M 去掉第 r 行和第 r 列后形成的 $n-1$ 阶的矩阵，记作 M_r。

```
const int maxn = 13;

int n, m;

struct Matrix {
  double mt[maxn][maxn];

  inline double* operator [] (int x) {
    return mt[x];
  }

  inline void clear() {
    for (int i = 1; i <= n; ++i) {
      for (int j = 1; j <= n; ++j) {
        mt[i][j] = 0;
      }
    }
  }

  inline double getans() {
    int nn = n - 1;
    double ans = 1.;
    for (int i = 1; i <= nn; ++i) {
      int mx = i;
      for (int j = i + 1; j <= nn; ++j) {
        if (mt[mx][i] < mt[j][i]) {
          mx = j;
```

```
        }
      }
      if (i != mx) {
        ans *= -1;
        for (int j = i; j <= nn; ++j) {
          std::swap(mt[mx][j], mt[i][j]);
        }
      }
      if (mt[i][i] < 1e-10) {
        return 0.;
      }
      for (int j = i + 1; j <= nn; ++j) {
        double kk = mt[j][i] / mt[i][i];
        for (int k = i; k <= nn; ++k) {
          mt[j][k] -= kk * mt[i][k];
        }
      }
    }
    for (int i = 1; i <= nn; ++i) {
      ans *= mt[i][i];
    }
    return ans;
  }
} Kif;

void solve() {
  read(n), read(m);
  Kif.clear();
  for (int i = 1, u, v; i <= m; ++i) {
    read(u), read(v);
    Kif[u][u]++, Kif[v][v]++;
    Kif[u][v]--, Kif[v][u]--;
  }
  printf("%.0f\n", Kif.getans());
}
```

5.1.2 最小生成树计数

然后是最小生成树计数。这个大概就是发现每个最小生成树每种边权的边数应该是一样的，且将这些边去掉后所得的连通块相同。

于是我们考虑建出一棵最小生成树，枚举边权然后把原来最小生成树上该边权的边删掉，然后跑矩阵树。

复杂度？假设离散之后边权 i 共有 a_i 条边，那么显然 $\sum a_i = m$。如果图没有重边，则 Kruscal 复杂度 $\mathcal{O}(m \log m)$，矩阵树复杂度为 $\mathcal{O}\left(\sum \left(n + m + \min(n, a_i)^3\right)\right)$，由于没有重边，前面的 $n + m$ 那一项卡满不过 $\mathcal{O}(m \times (n + m)) = \mathcal{O}(m^2) = \mathcal{O}(n^2 m)$，而后面那一项当每个 a_i 取

到 n 时最大，即 $\mathcal{O}\left(\frac{m}{n} \times n^3\right) = \mathcal{O}(n^2 m)$，所以总复杂度 $\mathcal{O}(n^2 m)$。

```
const int maxn = 105;
const int maxm = 1005;
const int mod = 31011;

int n, m;

struct Edge {
  int u, v, d;

  friend bool operator < (const Edge& a, const Edge& b) {
    return a.d < b.d;
  }
} e[maxm];

std::vector<std::pair<int, int>> v[maxn];

int col[maxn];

int fa[maxn];

inline int getfa(int x) {
  return fa[x] == x ? x : fa[x] = getfa(fa[x]);
}

inline void dfs(int now, int ccol, int bx) {
  col[now] = ccol;
  for (auto to : v[now]) {
    if (!col[to.first] && to.second != bx) {
      dfs(to.first, ccol, bx);
    }
  }
}

struct Matrix {
  int mt[maxn][maxn];

  inline void init(int n) {
    for (int i = 1; i <= n; ++i) {
      for (int j = 1; j <= n; ++j) {
        mt[i][j] = 0;
      }
    }
  }

  inline int* operator [] (int x) {
    return mt[x];
```

```
}

inline int solve(int n) {
  n--;
  if (!n) {
    return 1;
  }
  int ans = 1;
  for (int i = 1; i <= n; ++i) {
    int now = 0;
    for (int j = i; j <= n; ++j) {
      if (mt[j][i]) {
        now = i;
        break;
      }
    }
    if (!now) {
      return 0;
    } else if (now != i) {
      for (int j = i; j <= n; ++j) {
        std::swap(mt[i][j], mt[now][j]);
      }
      ans *= -1;
    }
    for (int j = i + 1; j <= n; ++j) {
      while (mt[j][i]) {
        int nowk = mt[i][i] / mt[j][i];
        for (int k = i; k <= n; ++k) {
          mt[i][k] -= mt[j][k] * nowk % mod;
          if (mt[i][k] < 0) {
            mt[i][k] += mod;
          } else if (mt[i][k] >= mod) {
            mt[i][k] -= mod;
          }
          std::swap(mt[i][k], mt[j][k]);
        }
        ans *= -1;
      }
    }
  }
  for (int i = 1; i <= n; ++i) {
    (ans *= mt[i][i]) %= mod;
  }
  if (ans <= mod) {
    ans += mod;
  }
  return ans;
```

```
  }
} mat;

inline int Main() {
  read(n), read(m);
  for (int i = 1; i <= m; ++i) {
    read(e[i].u), read(e[i].v), read(e[i].d);
  }
  std::sort(e + 1, e + m + 1);
  int cnt = 0, now = 0;
  for (int i = 1; i <= m; ++i) {
    if (now < e[i].d) {
      now = e[i].d;
      cnt++;
    }
    e[i].d = cnt;
  }
  for (int i = 1; i <= n; ++i) {
    fa[i] = i;
  }
  for (int i = 1; i <= m; ++i) {
    int fax = getfa(e[i].u);
    int fay = getfa(e[i].v);
    if (fax != fay) {
      fa[fax] = fay;
      v[e[i].u].emplace_back(e[i].v, e[i].d);
      v[e[i].v].emplace_back(e[i].u, e[i].d);
    }
  }
  int ans = 1;
  for (int i = 1; i <= cnt; ++i) {
    memset(col, 0, sizeof(col));
    int cntt = 0;
    for (int j = 1; j <= n; ++j) {
      if (!col[j]) {
        dfs(j, ++cntt, i);
      }
    }
    mat.init(cntt);
    for (int j = 1; j <= m; ++j) {
      if (e[j].d == i && col[e[j].u] != col[e[j].v]) {
        mat[col[e[j].u]][col[e[j].v]]--;
        mat[col[e[j].v]][col[e[j].u]]--;
        mat[col[e[j].u]][col[e[j].u]]++;
        mat[col[e[j].v]][col[e[j].v]]++;
      }
    }
```

```
    (ans *= mat.solve(cntt)) %= mod;
  }
  writeln(ans);
  return 0;
}
```

5.2 网络流

5.2.1 最大流

```
#include <cstdio>
#include <cstring>
#include <queue>

using namespace std;

#define fill(_a) memset(_a, 0x3f, sizeof(_a))
#define clr(_a) memset(_a, 0, sizeof(_a))
#define copy(_a, _b) memcpy(_a, _b, sizeof(_b))

const int inf = 0x3f3f3f3f;
const int maxn = 10005;
const int maxm = 100005;

int n, m, s, t;

struct EDGE
{
  int to, nxt;
  int dist;
} e[maxm<<1];

int first[maxn];
int tmp[maxn];
int _cnt = -1;

inline void init()
{
  _cnt = -1;
  memset(first, 0xff, sizeof(first));
}

inline void add_edge(int f, int t, int dist)
{
  e[++_cnt].to = t;
  e[_cnt].nxt = first[f];
```

```
  first[f] = _cnt;
  e[_cnt].dist = dist;
  e[++_cnt].to = f;
  e[_cnt].nxt = first[t];
  first[t] = _cnt;
  e[_cnt].dist = 0;
}

int deep[maxn];

inline bool bfs(int s, int t)
{
  copy(first, tmp);
  int now = s;
  queue<int> q;
  q.push(now);
  fill(deep);
  deep[s] = 0;
  while(!q.empty())
  {
    now = q.front();
    q.pop();
    for(int i = first[now]; ~i; i = e[i].nxt)
    {
      if(e[i].dist && deep[e[i].to] >= inf)
      {
        deep[e[i].to] = deep[now] + 1;
        q.push(e[i].to);
      }
    }
  }
  return deep[t] < inf;
}

inline int dfs(int now, int t, int limit)
{
  if(!limit || now == t)
    return limit;
  int flow = 0, f;
  for(int i = first[now]; ~i; i = e[i].nxt)
  {
    first[now] = i;
    if(deep[e[i].to] == deep[now]+1 && (f = dfs(e[i].to, t, min(limit, e[i].dist)))
      )))
    {
      flow += f;
      limit -= f;
```

```
      e[i].dist -= f;
      e[i^1].dist += f;
      if(!limit)
        break;
    }
  }
  return flow;
}

inline int Dinic(int s, int t)
{
  int maxflow = 0;
  while(bfs(s, t))
    maxflow += dfs(s, t, inf);
  return maxflow;
}

int main()
{
  scanf("%d%d%d%d", &n, &m, &s, &t);
  init();
  for(int i = 1, _f, _t, d; i <= m; ++i)
  {
    scanf("%d%d%d", &_f, &_t, &d);
    add_edge(_f, _t, d);
  }
  copy(tmp, first);
  printf("%d", Dinic(s, t));
  return 0;
}
```

```
#include <cstdio>
#include <cstring>
#include <queue>

using namespace std;

#define fill(_a) memset(_a, 0x3f, sizeof(_a))
#define clr(_a) memset(_a, 0, sizeof(_a))
#define copy(_a, _b) memcpy(_a, _b, sizeof(_b))

const int inf = 0x3f3f3f3f;
const int maxn = 10005;
const int maxm = 100005;

int n, m, s, t;

```

```
struct EDGE
{
  int to, nxt;
  int dist;
} e[maxm<<1];

int first[maxn];
int tmp[maxn];
int _cnt = -1;

inline void init()
{
  _cnt = -1;
  memset(first, 0xff, sizeof(first));
}

inline void add_edge(int f, int t, int dist)
{
  e[++_cnt].to = t;
  e[_cnt].nxt = first[f];
  first[f] = _cnt;
  e[_cnt].dist = dist;
  e[++_cnt].to = f;
  e[_cnt].nxt = first[t];
  first[t] = _cnt;
  e[_cnt].dist = 0;
}

int deep[maxn];

inline bool bfs(int s, int t)
{
  copy(first, tmp);
  int now = s;
  queue<int> q;
  q.push(now);
  fill(deep);
  deep[s] = 0;
  while(!q.empty())
  {
    now = q.front();
    q.pop();
    for(int i = first[now]; ~i; i = e[i].nxt)
    {
      if(e[i].dist && deep[e[i].to] >= inf)
      {
        deep[e[i].to] = deep[now] + 1;
```

```
        q.push(e[i].to);
      }
    }
  }
  return deep[t] < inf;
}

inline int dfs(int now, int t, int limit)
{
  if(!limit || now == t)
    return limit;
  int flow = 0, f;
  for(int i = first[now]; ~i; i = e[i].nxt)
  {
    first[now] = i;
    if(deep[e[i].to] == deep[now]+1 && (f = dfs(e[i].to, t, min(limit, e[i].dist)
        )))
    {
      flow += f;
      limit -= f;
      e[i].dist -= f;
      e[i^1].dist += f;
      if(!limit)
        break;
    }
  }
  return flow;
}

inline int Dinic(int s, int t)
{
  int maxflow = 0;
  while(bfs(s, t))
    maxflow += dfs(s, t, inf);
  return maxflow;
}

int main()
{
  scanf("%d%d%d%d", &n, &m, &s, &t);
  init();
  for(int i = 1, _f, _t, d; i <= m; ++i)
  {
    scanf("%d%d%d", &_f, &_t, &d);
    add_edge(_f, _t, d);
  }
  copy(tmp, first);
```

```
  printf("%d", Dinic(s, t));
  return 0;
}
```

第六章　其他

6.1　读入输出优化

```cpp
inline char gc() {
  static const int L = 23333;
  static char sxd[L], *sss = sxd, *ttt = sxd;
  if (sss == ttt) {
    ttt = (sss = sxd) + fread(sxd, 1, L, stdin);
    if (sss == ttt) {
      return EOF;
    }
  }
  return *sss++;
}

#ifdef Debug
#define dd c = getchar()
#else
#define dd c = gc()
#endif
template <class T>
inline bool read(T& x) {
  x = 0;
  char dd;
  bool flg = false;
  for (; !isdigit(c); dd) {
    if (c == '-') {
      flg = true;
    } else if (c == EOF) {
      return false;
    }
  }
  for (; isdigit(c); dd) {
    x = (x * 10) + (c ^ 48);
  }
  if (flg) {
    x = -x;
  }
```

```
  return true;
}
#undef dd

template <class T>
inline void write(T x) {
  if (x < 0) {
    x = -x;
    putchar('-');
  }
  if (x > 9) {
    write(x / 10);
    x %= 10;
  }
  putchar(x | 48);
}

template <class T>
inline void writeln(T x) {
  write(x);
  puts("");
}

template <class T>
inline void writesp(T x) {
  write(x);
  putchar(' ');
}
```