Contents

1 Math

1.1 快速冪

```
/*快速冪*/
ll mypow(ll x, ll y, ll p) {
    long long ans = 1;
    while (y) {
        if (y & 1)    ans = ans * x % p;    //prime
        x = x * x % p;    //每次把自己平方
        y >>= 1;    //每次右移一格
    }
    return ans;
}
```

1.2 快速乘

```
/*快速乘(a * b) mod m 大數乘法取餘數*/
ll mul(ll x, ll y, ll mod) {
    ll ret = x * y - (ll)((long double)x / mod * y) *
        mod;
    // LL ret=x*y-(LL)((long double)x*y/mod+0.5)*mod;
    return ret < 0 ? ret + mod : ret;
}
```

1.3 快速乘法 karatsuba

```
/*karatsuba 快速乘法*/

// Get size of the numbers
int getSize(ll num)
{
    int count = 0;
    while (num > 0)
    {
        count++;
        num /= 10;
    }
    return count;
}

ll karatsuba(ll X, ll Y){
    // Base Case
    if (X < 10 && Y < 10)
        return X * Y;

    // determine the size of X and Y
    int size = fmax(getSize(X), getSize(Y));

    // Split X and Y
    int n = (int)ceil(size / 2.0);
    ll p = (ll)pow(10, n);
    ll a = (ll)floor(X / (double)p);
    ll b = X % p;
    ll c = (ll)floor(Y / (double)p);
    ll d = Y % p;

    // Recur until base case
    ll ac = karatsuba(a, c);
    ll bd = karatsuba(b, d);
    ll e = karatsuba(a + b, c + d) - ac - bd;

    // return the equation
    return (ll)(pow(10 * 1L, 2 * n) * ac + pow(10 * 1L,
         n) * e + bd);
}
```

1.4 GCD

```
/*GCD*/
ll gcd(ll a, ll b){
    return b == 0 ? a : gcd(b, a % b);
}
```

1.5 ax+by=gcd(a,b)

```
/*ax+by=gcd(a,b) 一組解*/
ll a, b, x, y;
ll exgcd(ll a, ll b, ll& x, ll& y) {
    if (b) {
        ll d = exgcd(b, a % b, y, x);
        return y -= a / b * x, d;
    }
    return x = 1, y = 0, a;
}
```

1.6 Chinese Remainder Theorem

```
/*Chinese remainder theorem*/
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;
}
```

1.7 模反元素 inverse

```
/*Chinese remainder theorem*/
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;
}
```

1.8 Sieve Prime

```
/*Sieve_Prime*/
const int N = 20000000;//質數表大小
bool sieve[N];
vector<int> prime;
void linear_sieve(){
    for (int i = 2; i < N; i++)
    {
        if (!sieve[i]) prime.push_back(i);
        for (int p : prime)
        {
            if (i * p >= N) break;
            sieve[i * p] = true;
            if (i % p == 0) break;
        }
    }
}
```

1.9 Miller Rabin

```
/*Miller_Rabin 質數判定*/
// n < 4,759,123,141        3 :  2, 7, 61
// n < 1,122,004,669,633    4 :  2, 13, 23, 1662803
// n < 3,474,749,660,383          6 :  pirmes <= 13
// n < 2^64                       7 :
// 2, 325, 9375, 28178, 450775, 9780504, 1795265022
// Make sure testing integer is in range [2, n-2] if
// you want to use magic.
ll magic[N] = {};
bool witness(ll a, ll n, ll u, int t) {
    if (!a) return 0;
    ll x = mypow(a, u, n);  //快速冪
    for (int i = 0; i < t; i++) {
        ll nx = mul(x, x, n);   //快速乘
        if (nx == 1 && x != 1 && x != n - 1) return 1;
        x = nx;
    }
    return x != 1;
}
bool miller_rabin(ll n) {
    int s = (magic number size);
        // iterate s times of witness on n
        if (n < 2) return 0;
    if (!(n & 1)) return n == 2;
    ll u = n - 1; int t = 0;
    // n-1 = u*2^t
    while (!(u & 1)) u >>= 1, t++;
    while (s--) {
        ll a = magic[s] % n;
        if (witness(a, n, u, t)) return 0;
    }
    return 1;
}
```

1.10 Prime factorization

```
/*質因數分解*/
list<int> breakdown(int N) {
  list<int> result;
  for (int i = 2; i * i <= N; i++) {
    if (N % i == 0) {  // 如果 i 能够整除 N，说明 i 为
        N 的一个质因子。
      while (N % i == 0) N /= i;
      result.push_back(i);
    }
  }
  if (N != 1) {  // 说明再经过操作之后 N 留下了一个素数
    result.push_back(N);
  }
  return result;
}
```

1.11 Fibonacci

```
/*Fibonacci*/
int Fib[100005];
int F(int n) {
    Fib[0] = 0; Fib[1] = 1;

    for (int i = 2; i <= n; i++)
        Fib[i] = Fib[i - 1] + Fib[i - 2];

    return Fib[n];
}
```

1.12 josephus

```
/*約瑟夫問題：n個人圍成一桌，數到m的人出列*/
int josephus(int n, int m) {  //n人每m次
    int ans = 0;
    for (int i = 1; i <= n; ++i)
        ans = (ans + m) % i;
    return ans;
}
```

1.13 MOD

```
/*MOD*/
/// _fd(a,b)  floor(a/b).
/// _rd(a,m)  a-floor(a/m)*m.
/// _pv(a,m,r) largest x s.t x<=a && x%m == r.
/// _nx(a,m,r) smallest x s.t x>=a && x%m == r.
/// _ct(a,b,m,r) |A| , A = { x : a<=x<=b && x%m == r }.
int _fd(int a, int b) { return a < 0 ? (-~a / b - 1) :
    a / b; }
int _rd(int a, int m) { return a - _fd(a, m) * m; }
int _pv(int a, int m, int r) {
    r = (r % m + m) % m;
    return _fd(a - r, m) * m + r;
}
```

```
int _nt(int a, int m, int r) {
    m = abs(m);
    r = (r % m + m) % m;
    return _fd(a - r - 1, m) * m + r + m;
}
int _ct(int a, int b, int m, int r) {
    m = abs(m);
    a = _nt(a, m, r);
    b = _pv(b, m, r);
    return (a > b) ? 0 : ((b - a + m) / m);
}
```

1.14 Epsilon

```
/*精準度(Epsilon)*/
void Equal(float a, float b)    //判斷相等
{
    float eps = 1e-8;
    if ((fabs(a - b)) < eps)
        printf("Yes\n");
    else printf("No\n");
}
void NEqual(float a, float b)   //判斷不相等
{
    float eps = 1e-8;
    if ((fabs(a - b)) > eps)
        printf("Yes\n");
    else printf("No\n");
}
void Less(float a, float b) //判斷小於
{
    float eps = 1e-8;
    if ((a - b) < -eps)
        printf("Yes\n");
    else printf("No\n");
}
void Greater(float a, float b)  //判斷大於
{
    float eps = 1e-8;
    if ((a - b) > eps)
        printf("Yes\n");
    else printf("No\n");
}
```

1.15 取整函數 floor-ceil

```
/*floor向下取整，ceil向上取整*/
int floor(int a,int b){ return a/b - (a%b and a<0^b<0);
     }
int ceil (int a,int b){ return a/b + (a%b and a<0^b>0);
     }
```

1.16 Big number

```
/*大數(Big Number)*/
void add(int a[100], int b[100], int c[100])    //加法
{
    int i = 0, carry = 0;
    for (i = 0; i < 100; ++i) {
        c[i] = a[i] + b[i] + carry;
        carry = c[i] / 10;
        c[i] %= 10;
    }
}
void sub(int a[100], int b[100], int c[100])    //減法
{
    int i = 0, borrow = 0;
    for (i = 0; i < 100; ++i) {
        c[i] = a[i] - b[i] - borrow;
        if (c[i] < 0) {
            borrow = 1;
            c[i] += 10;
        }
        else
            borrow = 0;
    }
}
void mul(int a[100], int b[100], int c[100])    //乘法
{
    int i = 0, j = 0,carry = 0;
    for (i = 0; i < 100; ++i) {
        if (a[i] == 0) continue;
        for (j = 0; j < MAX; ++j)
            c[i + j] += a[i] * b[i];
    }

    for (i = 0; i < MAX; ++i) {
        carry = c[i] / 10;
        c[i] %= 10;
    }
}
void div(int a[100], int b[100], int c[100])    //除法
{
    int t[100];

    for (i = 100 - 1; i >= 0; i--) {
        for (int k = 9; k > 0; k--) // 嘗試商數
        {
            mul(b + i, k, t);
            if (largerthan(a + i, t))
            {
                sub(a + i, t, c + i);
                break;
            }
        }
    }
}
```

1.17 GaussElimination

```
/*GaussElimination*/
// by bcw_codebook

const int MAXN = 300;
const double EPS = 1e-8;

int n;
double A[MAXN][MAXN];

void Gauss() {
  for(int i = 0; i < n; i++) {
    bool ok = 0;
    for(int j = i; j < n; j++) {
      if(fabs(A[j][i]) > EPS) {
        swap(A[j], A[i]);
        ok = 1;
        break;
      }
    }
    if(!ok) continue;

    double fs = A[i][i];
    for(int j = i+1; j < n; j++) {
      double r = A[j][i] / fs;
      for(int k = i; k < n; k++) {
        A[j][k] -= A[i][k] * r;
      }
    }
  }
}
```

1.18 FFT

```
/*FFT*/
```

```
// use llround() to avoid EPS
typedef double Double;
const Double PI = acos(-1);

// STL complex may TLE
typedef complex<Double> Complex;
#define x real()
#define y imag()

template<typename Iter> // Complex*
void BitReverse(Iter a, int n){
    for (int i=1, j=0; i<n; i++){
        for (int k = n>>1; k>(j^=k); k>>=1);
        if (i<j) swap(a[i],a[j]);
    }
}

template<typename Iter> // Complex*
void FFT(Iter a, int n, int rev=1){ // rev = 1 or -1
    assert( (n&(-n)) == n ); // n is power of 2
    BitReverse(a,n);
    Iter A = a;

    for (int s=1; (1<<s)<=n; s++){
        int m = (1<<s);

        Complex wm( cos(2*PI*rev/m), sin(2*PI*rev/m) );
        for (int k=0; k<n; k+=m){
            Complex w(1,0);
            for (int j=0; j<(m>>1); j++){
                Complex t = w * A[k+j+(m>>1)];
                Complex u = A[k+j];
                A[k+j] = u+t;
                A[k+j+(m>>1)] = u-t;
                w = w*wm;
            }
        }
    }

    if (rev==-1){
        for (int i=0; i<n; i++){
            A[i] /= n;
        }
    }
}
```

2 Data structure

2.1 Heap

```
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;
}
```

2.2 SparseTable

```
template<typename T = int, typename CMP = greater<T>>
struct SparseTable {
  int n;
  T st[__lg(MAXN) + 1][MAXN];
  CMP cmp;
  inline T max(T a, T b) { return cmp(a,b) ? a : b; }
  void init(int _n, auto data) {
    n = _n;
    for (int i = 0; i < n; ++i) st[0][i] = data[i];
    for (int i = 1, t = 2; t < n; t <<= 1, i++)
      for (int j = 0; j + t <= n; j++)
        st[i][j] = max(st[i-1][j], st[i-1][j + t/2]);
  }
  T query(int a, int b) { // [a,b]
    int t = __lg(b - a + 1);
    return max(st[t][a], st[t][b - (1 << t) + 1]);
  }
};
```

2.3 Treap

```
struct Treap {
  int data, sz;
  Treap *l, *r;
  Treap(int k) : data(k), sz(1), l(0), r(0) {}
};
inline int sz(Treap *o) { return o ? o->sz : 0; }
void pull(Treap *o) { o->sz = sz(o->l) + sz(o->r) + 1;
    }
void push(Treap *o) {}
Treap *merge(Treap *a, Treap *b) {
  if (!a || !b) return a ? a : b;
  if (randint(sz(a)+sz(b)) < sz(a))
    return push(a), a->r = merge(a->r, b), pull(a), a;
  return push(b), b->l = merge(a, b->l), pull(b), b;
}
void split(Treap *o, Treap *&a, Treap *&b, int k) {
  if (!o) return a = b = 0, void();
  push(o);
  if (o->data <= k)
    a = o, split(o->r, a->r, b, k), pull(a);
  else b = o, split(o->l, a, b->l, k), pull(b);
}
void split2(Treap *o, Treap *&a, Treap *&b, int k) {
  if (sz(o) <= k) return a = o, b = 0, void();
  push(o);
  if (sz(o->l) + 1 <= k)
    a = o, split2(o->r, a->r, b, k - sz(o->l) - 1);
  else b = o, split2(o->l, a, b->l, k);
  pull(o);
}
Treap *kth(Treap *o, int k) {
  if (k <= sz(o->l)) return kth(o->l, k);
  if (k == sz(o->l) + 1) return o;
  return kth(o->r, k - sz(o->l) - 1);
}
int Rank(Treap *o, int key) {
  if (o->data < key)
    return sz(o->l) + 1 + Rank(o->r, key);
  else return Rank(o->l, key);
}
bool erase(Treap *&o, int k) {
  if (!o) return 0;
  if (o->data == k) {
    Treap *t = o;
    push(o), o = merge(o->l, o->r);
    delete t;
    return 1;
  }
  Treap *&t = k < o->data ? o->l : o->r;
  return erase(t, k) ? pull(o), 1 : 0;
}
void insert(Treap *&o, int k) {
  Treap *a, *b;
  split(o, a, b, k);
  o = merge(a, merge(new Treap(k), b));
}
void interval(Treap *&o, int l, int r) {
  Treap *a, *b, *c;
  split2(o, a, b, l - 1), split2(b, b, c, r);
  // operate
  o = merge(a, merge(b, c));
}
```

2.4 Link-Cut-Tree

```
// from bcw codebook

const int MXN = 100005;
const int MEM = 100005;

struct Splay {
  static Splay nil, mem[MEM], *pmem;
  Splay *ch[2], *f;
  int val, rev, size;
  Splay () : val(-1), rev(0), size(0) {
    f = ch[0] = ch[1] = &nil;
  }
  Splay (int _val) : val(_val), rev(0), size(1) {
    f = ch[0] = ch[1] = &nil;
  }
  bool isr() {
    return f->ch[0] != this && f->ch[1] != this;
  }
  int dir() {
    return f->ch[0] == this ? 0 : 1;
  }
  void setCh(Splay *c, int d) {
    ch[d] = c;
    if (c != &nil) c->f = this;
    pull();
  }
  void push() {
    if (rev) {
      swap(ch[0], ch[1]);
      if (ch[0] != &nil) ch[0]->rev ^= 1;
      if (ch[1] != &nil) ch[1]->rev ^= 1;
      rev=0;
    }
  }
  void pull() {
    size = ch[0]->size + ch[1]->size + 1;
    if (ch[0] != &nil) ch[0]->f = this;
    if (ch[1] != &nil) ch[1]->f = this;
  }
} Splay::nil, Splay::mem[MEM], *Splay::pmem = Splay::mem;
Splay *nil = &Splay::nil;

void rotate(Splay *x) {
  Splay *p = x->f;
  int d = x->dir();
  if (!p->isr()) p->f->setCh(x, p->dir());
  else x->f = p->f;
  p->setCh(x->ch[!d], d);
  x->setCh(p, !d);
  p->pull(); x->pull();
}

vector<Splay*> splayVec;
void splay(Splay *x) {
  splayVec.clear();
  for (Splay *q=x;; q=q->f) {
    splayVec.push_back(q);
    if (q->isr()) break;
  }
  reverse(begin(splayVec), end(splayVec));
  for (auto it : splayVec) it->push();
  while (!x->isr()) {
    if (x->f->isr()) rotate(x);
    else if (x->dir()==x->f->dir()) rotate(x->f),rotate(x);
    else rotate(x),rotate(x);
  }
}

Splay* access(Splay *x) {
  Splay *q = nil;
  for (;x!=nil;x=x->f) {
    splay(x);
    x->setCh(q, 1);
    q = x;
  }
  return q;
}
void evert(Splay *x) {
  access(x);
  splay(x);
  x->rev ^= 1;
  x->push(); x->pull();
}
void link(Splay *x, Splay *y) {
//  evert(x);
  access(x);
  splay(x);
  evert(y);
  x->setCh(y, 1);
}
void cut(Splay *x, Splay *y) {
//  evert(x);
  access(y);
  splay(y);
  y->push();
  y->ch[0] = y->ch[0]->f = nil;
}

int N, Q;
Splay *vt[MXN];

int ask(Splay *x, Splay *y) {
  access(x);
  access(y);
  splay(x);
  int res = x->f->val;
  if (res == -1) res=x->val;
  return res;
}
int main(int argc, char** argv) {
  scanf("%d%d", &N, &Q);
  for (int i=1; i<=N; i++)
    vt[i] = new (Splay::pmem++) Splay(i);
  while (Q--) {
    char cmd[105];
    int u, v;
    scanf("%s", cmd);
    if (cmd[1] == 'i') {
      scanf("%d%d", &u, &v);
      link(vt[v], vt[u]);
    } else if (cmd[0] == 'c') {
      scanf("%d", &v);
      cut(vt[1], vt[v]);
    } else {
      scanf("%d%d", &u, &v);
      int res=ask(vt[u], vt[v]);
      printf("%d\n", res);
    }
  }

  return 0;
}
```

3 Algorithm

3.1 Binary Search

```
/*Binary Search*/
```

```
int binary_search(int L,int R,int list[],int target,int
     mid){
    while(L<=R){
        mid=(L+R)/2;
        if(target==list[mid])
            return mid;
        else if(target<list[mid])
            R=mid-1;
        else
            L=mid+1;
    }
    return -1;
}
```

3.2 DFS

```
/*DFS*/

/*n皇后*/
//k為第幾行，a[k]為第幾列，n個皇后
int a[100], n, count;
void DFS(int k) {
    if (k > n) {//當k=n+1時找到解
        count++;
        printf("第%d個解\n", count);
        for (int i = 1; i <= n; i++) {//譜面輸出
            for (int j = 1; j < a[i]; j++)printf("0");
            printf("1");
            for (int j = a[i] + 1; j <= n; j++)printf("
                0");
            printf("\n");
        }
    }
    else {
        for (int i = 1; i <= n; i++) {//找不到合適的列
            (位置），回到上一行
            a[k] = i;  //存入皇后
            if (check(a, k))DFS(k + 1);//當前皇后的位置
                符合要求，則求下一個皇后(下一行)
        }
    }
}

/*交集法*/
//index=走訪位置，ans[]=答案，m為inp的序號
void DFS(int index, int m) {

    if (m == inp_size) {//等於最後一個
        for (int j = 0; j < n; j++) {   //check有重複出
            現的位置。
            ans[j] = ans[j] & tmp[j]; //位元運算
        }
    }

    else {
        while (index < n) {
            if (check(index, inp[m])) {  //判斷可不可以
                放進去。
                for (int j = 0; j < inp[m]; j++) {  //
                    放入方塊。
                    tmp[index + j] = 1;
                }
                DFS(index + inp[m], m + 1);    //進到下
                    一層，左子樹。
                for (int j = 0; j < inp[m]; j++) {  //
                    回復上一動，回節點。
                    tmp[index + j] = 0;
                }
            }
            index++;
        }
    }
}
```

4 Graph

4.1 Disjoint Set(Union-Find)

```
/*Disjoint Set(Union-Find) 並查集*/
int f[N];    // 宣告父節點陣列 f
void init(int n) {
    for (int i = 0; i < n; i++)
        f[i] = i;
}
int find(int x) {
    return f[x] == x ? x : f[x] = find(f[x]);
}
void merge(int x, int y) {
    x = find(x), y = find(y);
    if (x != y)    f[y] = x;
}
```

4.2 Kruskal’s algorithm 最小生成樹

```
/*Kruskal’s algorithm 最小生成樹*/
struct Edge {
    int u, v, w;    // 點 u 連到點 v 並且邊權為 w
    friend bool operator<(const Edge& lhs, const Edge&
        rhs) {
        return lhs.w > rhs.w;    //兩條邊比較大小用邊權
            比較
    }
};

Edge graph[m];    // 宣告"邊"型態的陣列 graph
init(N);    //N個邊
sort(graph, graph + m); // 將邊照大小排序
int ans = 0;    //權重和
for (int i = 0; i < m; i++) {
    if (find(graph[i].u) != find(graph[i].v)) { // 如果
        兩點未聯通
        merge(graph[i].u, graph[i].v);    // 將兩點設成
            同一個集合
        ans += graph[i].w;    // 權重加進答案
    }
}

/*使用pq取代sort*/
int main() {
    int i, n, a, b, d;
    while (cin >> n) {

        if (n == 0)break;
        init(n);
        priority_queue<Edge> graph;// 宣告邊型態的陣列
            graph
        //priority_queue需改成return lhs.w > rhs.w;
        for (i = 0; i < n * (n - 1) / 2; i++) {
            cin >> a >> b >> d;
            graph.push(Edge{a,b,d});
        }

        int ans = 0;
        for (i = 0; i < n * (n - 1) / 2; i++) {
            if (find(graph.top().u) != find(graph.top()
                .v)) { // 如果兩點未聯通
                merge(graph.top().u, graph.top().v);
                    // 將兩點設成同一個集合
                ans += graph.top().w;    // 權重加進答
                    案
            }
            graph.pop();
        }
        cout << ans << "\n";
    }
    return 0;
}
```

4.3 Dijkstra’s Algorithm

```
#define MAX_V 100
#define INF 10000

struct Edge {
  int idx,w;
};
bool operator>(const Edge& a, const Edge& b) {
  return a.w > b.w;
}

int dist[MAX_V];
vector<vector<Edge> > adj(MAX_V);
void dijkstra(int vn, int s) {
  vector <bool> vis(vn, false);
  fill(dist, dist + vn, INF); dist[s] = 0;

  priority_queue <Edge, vector<Edge>, greater<Edge> >
      pq;
  Edge node;
  node.idx = s; node.w = 0;
  pq.emplace(node);
  while (!pq.empty()) {
    int u = pq.top().idx; pq.pop();
    if (vis[u])continue;
    vis[u] = true;
    for (auto v : adj[u]) {
      if (dist[v.idx] > dist[u] + v.w) {
        dist[v.idx] = dist[u] + v.w;
        node.w = dist[v.idx];
        node.idx = v.idx;
        pq.emplace(node);
      }

    }
  }
}

int main() {
  cin.tie(0);
  ios_base::sync_with_stdio(false);

  int start, u, v, w, i, j, ans;
  set <int> myset;

  //input
  cin >> start;
  Edge node;
  while (cin >> u >> v >> w) {
    node.idx = v; node.w = w;
    myset.insert(u);
    myset.insert(v);
    adj[u].push_back(node);
  }

  dijkstra(myset.size(), start);
  for (auto i : myset) {
    printf("%d: %d\n", i, dist[i]);
  }
  return 0;
}
```

4.4 Prim’s algorithm

```
#define N 100

struct Edge {
  int idx, w;
};
bool operator>(const Edge& lhs, const Edge& rhs) {
  return lhs.w > rhs.w;    //兩條邊比較大小用邊權比較
}

vector<vector<Edge> > adj(N, vector<Edge>(N));
priority_queue <Edge, vector<Edge>, greater<Edge> > pq;

int prim_pq(int vn, int start) {
  int tot = 0;

  vector<bool> vis(vn, false);
  vis[start] = true;

  for (auto v : adj[start]) pq.emplace(v);
  int times = 1;
  while (!pq.empty()) {
    Edge mn = pq.top(); pq.pop();
    if (vis[mn.idx])continue;
    vis[mn.idx] = true;
    tot += mn.w;

    int u;
    for (int i = 0;; i++) {
      if (adj[mn.idx][i].w == mn.w) {
        u = adj[mn.idx][i].idx;
        break;
      }
    }
    printf("%d: <%d,%d>\n",times++, u, mn.idx);

    for (auto v : adj[mn.idx]) pq.emplace(v);
  }
  return tot;
}

int main() {
  int start, u, v, w, i, j, ans, index, min, temp;
  //input
  cin >> start;
  int count = 0;
  Edge node;
  while (cin >> u >> v >> w) {
    node.idx = v; node.w = w;
    adj[u].push_back(node);
    node.idx = u; node.w = w;
    adj[v].push_back(node);
    count++;
  }

  ans = prim_pq(count, start);
  cout << "\nThe cost of minimum spanning tree: " <<
      ans << "\n";

  return 0;
}
```

4.5 BellmanFord algorithm (處理負環)

```
#define N 100
#define INF 1000

int dist[N][N];
vector<vector<int> > length(N,vector<int>(N));

void BellmanFord(int n, int v)
{ /* n為節點總數，計算單一起點v/所有終點的最短路徑，其
    中邊長允許是負值，length為adjacency matrix */
    for (int k = 0; k < n; k++)for (int i = 0; i < n; i
        ++)i == 0 ? dist[k][i] = 0 : dist[k][i] = INF;
        /* 對dist做初始化 */
    for (int i = 0; i < n; i++)if(length[v][i])dist[1][
        i] = length[v][i]; /* 對dist[1]做初始化 */

    for (int i = 0; i < n; i++) {
        dist[1][i] == INF ? cout << "i" : cout << dist
            [1][i];
        if (i != n - 1)cout << " ";
    }cout << "\n";

    for (int k = 2; k <= n - 1; k++) {
        for (int u = 0; u < n; u++) {
            for (int i = 0; i < length[u].size(); i++)
                {
                if (!length[u][i])continue;
                if (length[u][i] == INF)continue;
                if (dist[k][i] > dist[k-1][u] + length[
                    u][i])
                    dist[k][i] = dist[k-1][u] + length[
                        u][i];
            }
        }
        for (int i = 0; i < n; i++) {
            dist[k][i] == INF ? cout << "i" : cout <<
                dist[k][i];
            if (i != n - 1)cout << " ";
        }if (k != n - 1)cout << "\n";
    }
}
int main() {
    int i, u, v, w,s,vn;
    set<int> _set;

    while (cin >> u >> v >> w) {
        length[u][v] = w;
        _set.insert(u);
        _set.insert(v);
    }

    s = 0;
    vn = _set.size();
    BellmanFord(vn,s);

    return 0;
}
```

4.6 Adjacency list for DFS And BFS

```
/*Adjacency list for DFS And BFS*/

typedef struct node* nodePointer;  /*每一個節點裝入
    linkedlist*/
typedef struct node {
    int vertex;
    nodePointer link;
};

bool visited[MAX_VERTICES];  /* intitial: FALSE */
nodePointer graph[MAX_VERTICES];


void dfs(int v)
{/* 從一個點v開始對這個圖的深度優先搜尋 */
    nodePointer w;
    visited[v] = TRUE;
    cout << v;
    for (w = graph[v]; w; w = w->link)
        if (!visited[w->vertex]) {
            cout << " ";
            dfs(w->vertex);
        }
}

void bfs(int v)
{/* 從圖的頂點v開始做廣度(寬度)優先搜尋。
    全域陣列visited初始是0，佇列的運作和第四章的相似，
        front和rear是全域變數 */
    nodePointer w;
    queue <int> q;/* 佇列初始化 */
    cout << v;
    visited[v] = TRUE;
    q.push(v);
    while (q.size()) {
        v = q.front();
        q.pop();
        for (w = graph[v]; w; w = w->link)
            if (!visited[w->vertex]) {
                cout << " " << w->vertex;
                q.push(w->vertex);
                visited[w->vertex] = TRUE;
            }
    }
}

void create_node(int a, int b) {
    nodePointer temp = new(node);//將 b 接到 graph[a](
        list) 的最後。
    nodePointer current = graph[a];
    while (current->link) current = current->link;
    temp->vertex = b; current->link = temp; temp->link
        = NULL;

    nodePointer temp_a = new(node);//將 a 接到 graph[b
        ](list) 的最後。
    current = graph[b];
    while (current->link)current = current->link;
    temp_a->vertex = a; current->link = temp_a; temp_a
        ->link = NULL;
}
int main() {

    int a, b;
    for (int i = 0; i < MAX_VERTICES; i++) {//建立所有
        list
        nodePointer temp = new(node);
        temp->vertex = i;
        temp->link = NULL;
        graph[i] = temp;
    }
    while (cin >> a >> b) create_node(a, b);
    dfs(0); /* DFS */
  for (auto& it : visited)it = FALSE; /* 初始化 */
  bfs(0); /* BFS */
    return 0;
}
```

5 DP

5.1 背包問題

```
/*背包問題*/
// n,m,price,value
// 0/1
for (int j = m; j >= price; --j)
if (f[j - price] + value > f[j])
f[j] = f[j - price] + value;
// 完全
for (int j = 1; j <= price; ++j)
if (f[j - price] + value > f[j])
f[j] = f[j - price] + value;
```

5.2 最長公共子序列 LCS

```
/*LCS 最長公共子序列*/
void LCS() {
    for (int i = 0; i <= n1; i++) length[i][0] = 0;
    for (int j = 0; j <= n2; j++) length[0][j] = 0;
    for (int i = 1; i <= n1; i++)
        for (int j = 1; j <= n2; j++)
            if (s1[i] == s2[j]) {
                length[i][j] = length[i - 1][j - 1] +
                    1;
                prev[i][j] = 0;  // 左上方
            }
            else {
                if (length[i - 1][j] < length[i][j -
                    1]) {
                    length[i][j] = length[i][j - 1];
                    prev[i][j] = 1;  // 左方
                }
                else {
                    length[i][j] = length[i - 1][j];
                    prev[i][j] = 2;  // 上方
                }
            }
    cout << "LCS的長度是" << length[n1][n2];
    cout << "LCS是";
    print_LCS(n1, n2);
}
void print_LCS(int i, int j) {
    if (i == 0 || j == 0) return;
    if (prev[i][j] == 0) {
        print_LCS(i - 1, j - 1);
        cout << s1[i];             // 印出LCS的元素
    }
    else if (prev[i][j] == 1)  // 左方
        print_LCS(i, j - 1);
    else if (prev[i][j] == 2)  // 上方
        print_LCS(i - 1, j);
}
```

5.3 最大非連續子序列和

```
/*最大非連續子序列和*/
int sub_max(int* list,int sub_len) { //子序列長度
    sub_len
    if (sub_len == 3) {
        return list[0] + list[2];
    }
    int temp[10005];
    for (int m = 0; m < sub_len; m++) {
        temp[m] = list[m];
    }
    temp[0] = list[0];
    temp[1] = list[1] > list[0] ? list[1] : list[0];
    for (int i = 2; i < sub_len; i++) {
        temp[i] = max(max(temp[i], temp[i - 1]), temp[i
             - 2] + list[i]);
    }
    return temp[sub_len - 1];
}
int main() {

    int n, m;
    int list[10005];

    cin >> n;

    for (m = 0; m < n; m++) {
        cin >> list[m];
    }
    sub_len = m;//list大小,global變數

    cout << sub_max(list, sub_len);

    return 0;
}
```

6 STL tool

6.1 常用工具

```
/*--------常用工具--------*/
swap(a,b);
min(a,b);
max({ a, b, c });

//math
abs(x);
pow(x);
sqrt(x);
__gcd(x, y);
__lg(x)  //以2為底數
log(x)   //以e為底數
log10(x) //以10為底數

//陣列處理
sort(arr,arr+n);
reverse(arr,arr+n);
*min_element(arr, arr+n); //value
min_element(arr, arr+n) - arr; //index
*lower_bound(arr, arr+4, c) << '\n'; //第一個大於等於 c
*upper_bound(arr, arr+4, c) << '\n'; //第一個大於 c
fill(arr, arr+3, 123); //取代 arr[0]=123 arr[1]=123 arr
    [2]=123

//輸出
cout << fixed << setprecision(10); //四捨五入 或是更高
    精度(int)10 * 位數 + 0.5
cout << setw(n) << setfill(c) << ; //寬度n 用char(c)填
    補

//迭代器
T.begin() //返回一個迭代器，它指向容器c的第一個元素
T.end() //返回一個迭代器，它指向容器c的最後一個元素的下
    一個位置
T.rbegin() //返回一個逆序迭代器，它指向容器c的最後一個
    元素
T.rend() //返回一個逆序迭代器，它指向容器c的第一個元素
    前面的位置
T.find() //可用於set,map的earse()。
```

6.2 Sort

```
/*--------sort--------*/

//cmp
struct T {int val, num;};
bool cmp(const T &a, const T &b) {
    return a.num < b.num;
}
sort(arr.begin(), arr.end(), cmp);

//operator
struct Point {
    int x, y;
    bool operator<(Point b) {
        if (x != b.x) return x < b.x;
        else return y < b.y;
    }
};
Point arr[n];
sort(arr, arr+n); //二維平面，從小到大排列。
```

6.3 Stack

```
/*--------stack--------*/
• push()
• pop()
• top()
• empty()
• size()
```

6.4 Queuet

```
/*--------queue--------*/
• push()
• pop()
• front()
• empty()
• size()
```

6.5 Priority Queue

```
/*--------priority_queue--------*/
• top()
• push()
• pop()
• emplace()
```

```
priority_queue<T> pq //預設由大排到小
priority_queue<int, vector<int>, less<int> > pq;
priority_queue<T, vector<T>, greater<T> > pq;  //改成由小排到大
priority_queue<T, vector<T>, cmp> pq; //自行定義 cmp 排序

struct cmp {
    bool operator()(node a, node b) {
        /*priority_queue優先判定為!cmp，所以「由大排到小」 需「反向」定義
         實現「最小值優先」*/
        return a.x < b.x;
    }
};
```

6.6 List

```
/*--------list--------*/
• push_back()
• pop_back()
• push_front()
• pop_front()
• back()
• front()
• insert(index, obj)
• erase()

//遍歷
for (auto iter = _list.begin(); iter != _list.end();
    iter++)
    cout << *iter << "\n";
```

6.7 Set

```
/*--------set--------*/
• insert()
• erase(l, r) //l與r皆為iterator
• erase()
• empty()
• clear()
• count() //元素是否存在

//遍歷
int mints[] = { 75,23,65,42,13,75,65 };
set<int> myset(myints, myints + 7);
for (auto it = myset.begin(); it != myset.end(); it++)
    cout << ' ' << *it;
```

6.8 Map

```
/*--------map--------*/
map<char, int> mymap;
mymap['b'] = 100, mymap['a'] = 200, mymap['c'] = 300;

//find
auto iter = mymap.find("a");
if (iter != mapStudent.end())
    cout << "Find, the value is" << iter->second <<
        endl;
else
    cout << "Do not Find" << endl;

//erase
auto iter = mymap.find("a");
mymap.erase(iter);

//遍歷
for (auto it = mymap.begin(); it != mymap.end(); it++)
cout << it->first << ", " << it->second << endl
```

6.9 Stringstream

```
/*--------stringstream--------*/
stringstream ss;
• getline(cin, str);
• ss.str("");
• ss.clear();
```

```
//實現"切割"以及"型態轉換"
//int_to_string
ss << n;
ss >> str;
//string_to_int
ss << str;
ss >> n;
```

```
//注意輸入時，cin後的快取問題
cin >> n;
getline(cin, str);  //str = endl
getline(cin, str);  //str = 目標str
```

```
//實現"進制轉換"
ss << oct << s;    //以8進制讀入流中
ss << hex << s;    //以16進制讀入流中
ss >> n;        //10進制int型輸出
ss >> s;        //x進制str型輸出
```

6.10 Bitset

```
/*--------bitset--------*/
//init
string s = "1001101";
bitset<10> b(s);
```

```
b.set();    //每個位元設 '1'
b.reset();  //每個位元設 '0'
b[pos] = 1;

//轉換
s = b.to_string();
unsigned long x = b.to_ulong();

//overload
b = !b0;
b = b0 & b1;
b = b0 | b1;
b = b0 ^ b1;

//shift
new_b = b << 2;
new_b = b >> 2;

//sum
b.any();//判別是否有 '1'
b.none();//判別是否沒 '1'
cnt = b.count();// 判別 '1' 之個數
cnt = b.size() - b.count();//判別 '0' 之個數
```

7 Other

7.1 前置作業

```
/*前置作業*/
#include <bits/stdc++.h>
#define ll long long
#define endl "\n"
using namespace std;

/*
include <bits/stdc++.h>
C:\Program Files\Microsoft Visual
Studio\2022\Community\VC\Tools\MSVC\14.30.30705\include\bits
*/

int main() {

 cin.tie(0); //取消強制flush
 ios_base::sync_with_stdio(false); //取消 iostream 與 stdio 的同步使用

 return 0;
}
```

7.2 Header

```
// C
#ifndef _GLIBCXX_NO_ASSERT
#include <cassert>
#endif
#include <cctype>
#include <cerrno>
#include <cfloat>
#include <ciso646>
#include <climits>
#include <clocale>
#include <cmath>
#include <csetjmp>
#include <csignal>
#include <cstdarg>
#include <cstddef>
#include <cstdio>
#include <cstdlib>
#include <cstring>
#include <ctime>

#if __cplusplus >= 201103L
#include <ccomplex>
#include <cfenv>
#include <cinttypes>
#include <cstdalign>
#include <cstdbool>
#include <cstdint>
#include <ctgmath>
#include <cwchar>
#include <cwctype>
#endif

// C++
#include <algorithm>
#include <bitset>
#include <complex>
#include <deque>
#include <exception>
#include <fstream>
#include <functional>
#include <iomanip>
#include <ios>
#include <iosfwd>
#include <iostream>
#include <istream>
#include <iterator>
#include <limits>
#include <list>
#include <locale>
#include <map>
#include <memory>
#include <new>
#include <numeric>
#include <ostream>
#include <queue>
#include <set>
#include <sstream>
#include <stack>
#include <stdexcept>
#include <streambuf>
#include <string>
#include <typeinfo>
#include <utility>
#include <valarray>
#include <vector>

#if __cplusplus >= 201103L
#include <array>
#include <atomic>
#include <chrono>
#include <condition_variable>
#include <forward_list>
#include <future>
#include <initializer_list>
#include <mutex>
#include <random>
#include <ratio>
#include <regex>
#include <scoped_allocator>
#include <system_error>
#include <thread>
#include <tuple>
#include <typeindex>
#include <type_traits>
#include <unordered_map>
#include <unordered_set>
#endif
```