Vale + 2 x Ale ICPC Team Notebook

Contents

1 C++

1.1 C++ template

```
#include <bits/stdc++.h>

#define fi first
#define se second
#define forn(i,n) for(int i=0; i< (int)n; ++i)
#define for1(i,n) for(int i=1; i<= (int)n; ++i)
#define fore(i,l,r) for(int i=(int)l; i<= (int)r; ++i)
#define ford(i,n) for(int i=(int)(n) - 1; i>= 0; --i)
#define fored(i,l,r) for(int i=(int)r; i>= (int)l; --i)
#define pb push_back
#define el '\n'
#define d(x) cout<< #x<< " " << x<<el
#define ri(n) scanf("%d",&n)
#define sz(v) int(v.size())
#define all(v) v.begin(),v.end()

using namespace std;

typedef long long ll;
typedef double ld;
typedef pair<int,int> ii;
typedef pair<ll,ll> pll;
typedef tuple<int, int, int> iii;
typedef vector<int> vi;
typedef vector<ii> vii;
typedef vector<ll> vll;
typedef vector<ld> vd;

const int inf = 1e9;
const int nax = 1e5+200;
const ld pi = acos(-1);
const ld eps= 1e-9;

int dr[] = {1,-1,0, 0,1,-1,-1, 1};
int dc[] = {0, 0,1,-1,1, 1,-1,-1};

ostream& operator<<(ostream& os, const ii& pa) { //
    DEBUGGING
  return os << "("<< pa.fi << ", " << pa.se << ")";
}

int main(){
  ios_base::sync_with_stdio(false);
  cin.tie(NULL); cout.tie(NULL);
  cout << setprecision(20)<< fixed;
}
```

2 Graph algorithms

2.1 Dijkstra

```
// O ((V+E)*log V)
vector <ii> g[nax];
int d[nax], p[nax];
void dijkstra(int s, int n){
  forn(i, n) d[i] = inf, p[i] = -1;
  d[s] = 0;
  priority_queue <ii, vector <ii>,greater<ii> > q;
  q.push({0, s});
  while(sz(q)){
    auto [dist, u] = q.top();  q.pop();
    if(dist > d[u]) continue;
    for(auto& [v, w]: g[u]){
      if (d[u] + w < d[v]){
        d[v] = d[u] + w;
        p[v] = u;
        q.push(ii(d[v], v));
      }
    }
  }
}
vi find_path(int t){
```

```
    vi path;
    int cur = t;
    while(cur != -1){
      path.pb(cur);
      cur = p[cur];
    }
    reverse(all(path));
    return path;
  }
```

3 Flows

3.1 Hungarian Algorithm

```
const ld inf = 1e18; // To Maximize set "inf" to 0, and
    negate costs
inline bool zero(ld x){ return x == 0; } // For Integer/
    LL --> change to x == 0
struct Hungarian{
  int n; vector<vd> c;
  vi l, r, p, sn;  vd ds, u, v;
  Hungarian(int n): n(n), c(n, vd(n, inf)), l(n, -1), r(n
      , -1), p(n), sn(n), ds(n), u(n), v(n){}
  void set_cost(){ forn(i, n) forn(j, n) cin >> c[i][j];
      }
        ld assign() {
    set_cost();
                forn(i, n) u[i] = *min_element(all(c[i]))
                    ;
                forn(j, n){
      v[j] = c[0][j] - u[0];
      for1(i, n-1) v[j] = min(v[j], c[i][j] - u[i]);
    }
                int mat = 0;
                forn(i, n) forn(j, n) if(r[j] == -1 &&
                    zero(c[i][j] - u[i]  -v[j])){
      l[i] = j,  r[j] = i, ++mat; break;
    }
                for(; mat < n; ++mat){
      int s = 0, j = 0, i;
      while(l[s] != -1) ++s;
      forn(k, n) ds[k] = c[s][k] - u[s] - v[k];
      fill(all(p), -1),  fill(all(sn), 0);
      while(1){
        j = -1;
        forn(k, n) if(!sn[k] && (j == -1 || ds[k] < ds[j
            ])) j = k;
        sn[j] = 1,  i = r[j];
        if(i == -1) break;
        forn(k, n) if(!sn[k]){
          auto n_ds = ds[j] + c[i][k] - u[i] - v[k];
          if(ds[k] > n_ds) ds[k] = n_ds,  p[k] = j;
          }
        }
        forn(k, n) if(k != j && sn[k]){
          auto dif = ds[k] - ds[j];
          v[k] += dif, u[r[k]] -= dif;
        }
        u[s] += ds[j];
        while(p[j] >= 0) r[j] = r[p[j]],  l[r[j]] = j,  j =
            p[j];
        r[j] = s,  l[s] = j;
                }
                ld val = 0;
      forn(i, n) val += c[i][l[i]];
                return val;
          }
    void print_assignment(){ forn(i, n) cout << i+1 << " "
        << l[i]+1 << el; }
};
```

4 Data Structures

4.1 Disjoint Set Union

```
struct dsu{
  vi p, r; int comp;
  dsu(int n): p(n), r(n, 1), comp(n){iota(all(p), 0);}
  int find_set(int i){return p[i] == i ? i : p[i] =
      find_set(p[i]);}
  bool is_same_set(int i, int j){return find_set(i) ==
      find_set(j);}
  void union_set(int i, int j){
    if((i = find_set(i)) == (j = find_set(j))) return;
    if(r[i] > r[j]) swap(i, j);
    r[j] += r[i];  r[i] = 0;
    p[i] = j;  --comp;
  }
};
```

5 Math

5.1 Sieve of Eratosthenes

```
// O(n)
// pr contains prime numbers
// lp[i] == i if i is prime
// else lp[i] is minimum prime factor of i
const int nax = 1e7;
int lp[nax+1];
vector<int> pr; // It can be sped up if change for an
    array
```

```cpp
void sieve(){
  fore(i,2,nax-1){
    if (lp[i] == 0) {
      lp[i] = i; pr.pb(i);
    }
    for (int j=0, mult= i*pr[j]; j<sz(pr) && pr[j]<=lp[i]
        && mult<nax; ++j, mult= i*pr[j])
      lp[mult] = pr[j];
  }
}
```

6 Dynamic Programming

6.1 Longest common subsequence

```cpp
const int nax = 1005;
int dp[nax][nax];
int lcs(const string &s, const string &t){
  int n = sz(s), m = sz(t);
  forn(j,m+1) dp[0][j] = 0;
  forn(i,n+1) dp[i][0] = 0;
  for1(i,n){
    for1(j,m){
      dp[i][j] = max(dp[i-1][j], dp[i][j-1]);
      if (s[i-1] == t[j-1]){
        dp[i][j] = max(dp[i][j], dp[i-1][j-1] + 1);
      }
    }
  }
  return dp[n][m];
}
```

7 Geometry

7.1 Point

```cpp
struct pt{
        ld x, y;
        pt(){}
        pt(ld x, ld y): x(x), y(y){}
        pt(ld ang): x(cos(ang)), y(sin(ang)){}  // Polar
            unit point: ang(randians)
  // ------- BASIC OPERATORS -------  //
  pt operator+(pt p){ return pt(x+p.x, y+p.y); }
        pt operator-(pt p){ return pt(x-p.x, y-p.y); }
        pt operator*(ld t){ return pt(x*t, y*t); }
        pt operator/(ld t){ return pt(x/t, y/t); }
        ld operator*(pt p){ return x*p.x + y*p.y; }
        ld operator%(pt p){ return x*p.y - y*p.x; }
  // ------- COMPARISON OPERATORS -------  //
  bool operator==(pt p){ return abs(x - p.x) <= eps &&
     abs(y - p.y) <= eps; }
        bool operator<(pt p)const{ // for sort, convex
            hull/set/map
                return x < p.x - eps || (abs(x - p.x) <=
                    eps && y < p.y - eps); }
        bool operator!=(pt p){ return !operator==(p); }
  // -------------- NORMS --------------  //
        ld norm2(){ return *this**this; }
        ld norm(){ return sqrt(norm2()); }
        pt unit(){ return *this/norm(); }
        // ------------ SIDE, LEFT------------ //
  ld side(pt p, pt q){ return (q-p) % (*this-p); }// C is
     : >0 L, ==0 on AB, <0 R
        bool left(pt p, pt q){ // Left of directed line
            PQ? (eps == 0 if integer)
                return side(p, q) > eps; } // (change to
                    >= -eps to accept collinear)
  // -------------- ANGLES -------------- //
  ld angle(){ return atan2(y, x); } // Angle from origin,
      in [-pi, pi]
  ld min_angle(pt p){ return acos(*this*p / (norm()*p.
     norm())); } // In [0, pi]
  ld angle(pt a, pt b, bool CW){ // Angle< AB(*this) > in
      direction CW
    ld ma = (a - b).min_angle(*this - b);
    return side(a, b) * (CW ? -1 : 1) <= 0 ? ma : 2*pi -
        ma; }
  bool in_angle(pt a, pt b, pt c, bool CW=1){ // Is pt
     inside infinite angle ABC
    return angle(a, b, CW) <= c.angle(a, b, CW); } //
        From AB to AC in CW direction
  // -------------- ROTATIONS -------------- //
        pt rot(pt p){ return pt(*this % p,*this * p); }//
           use ccw90(1,0), cw90(-1,0)
        pt rot(ld ang){ return rot(pt(sin(ang), cos(ang))
            ); } // CCW, ang (radians)
        pt rot_around(ld ang, pt p){ return p + (*this -
            p).rot(ang); }
  pt perp(){ return rot(pt(1, 0)); }
  // -------------- SEGMENTS -------------- //
  bool in_disk(pt p, pt q){ return (p - *this) * (q - *
     this) <= 0; }
  bool on_segment(pt p, pt q){ return side(p, q) == 0 &&
     in_disk(p, q); }
};
int sgn(ld x){
  if(x < 0) return -1;
  return x == 0 ? 0 : 1;
}
void segment_intersection(pt a, pt b, pt c, pt d, vector<
   pt>& out){ // AB y CD
  ld sa = a.side(c, d), sb = b.side(c, d);
```

```
ld sc = c.side(a, b), sd = d.side(a, b);            // proper cut
if(sgn(sa)*sgn(sb) < 0 && sgn(sc)*sgn(sd) < 0) out.pb((a*sb - b*sa) / (sb-sa));
for(pt p : {c, d}) if(p.on_segment(a, b)) out.pb(p);
    for(pt p : {a, b}) if(p.on_segment(c, d)) out.pb(p);
  }
```