Team Notebook

October 24, 2019

Contents

1 Algorithms

1.1 Hungarian

```
/*
    This will take a matrix a[N][N] and choose one item for
         each row such that the sum of all items
    is minimized.
    O(n^3)
*/
#define N 107

ll INF = 1000000000000000000ll;
int n;

ll a[N][N];

void hungarian(){
    vector<ll> u(n+1), v(n+1), p(n+1), way(n+1);
    for(int i=1; i<=n; i++){
        p[0] = i;
        ll j0 = 0;
        vector<ll> minv(n+1, INF);
        vector<char> used(n+1, false);
        do {
            used[j0] = true;
            ll i0 = p[j0], delta = INF, j1;
            for(int j=1; j<=n; j++){
                if(!used[j]) {
                    ll cur = a[i0][j]-u[i0]-v[j];
                    if(cur < minv[j])
                        minv[j] = cur, way[j] = j0;
                    if(minv[j] < delta)
                        delta = minv[j], j1 = j;
                }
            }
            for(int j=0; j<=n; j++){
                if(used[j])
                    u[p[j]] += delta, v[j] -= delta;
                else
                    minv[j] -= delta;
            }
            j0 = j1;
        } while(p[j0] != 0);
        do {
            ll j1 = way[j0];
            p[j0] = p[j1];
            j0 = j1;
        } while(j0);
    }
    vector<int> ans(n+1);
    for(int j=1; j<=n; j++)
        ans[p[j]] = j;

}
```

1.2 TernarySearch

```
double ternary_search(double l, double r) {
    double eps = 1e-9;              //set the error limit here
    while (r - l > eps) {
        double m1 = l + (r - l) / 3;
        double m2 = r - (r - l) / 3;
        double f1 = f(m1);      //evaluates the function at m1
        double f2 = f(m2);      //evaluates the function at m2
        if (f1 < f2)
            l = m1;
        else
            r = m2;
    }
    return f(l);                    //return the maximum of f(x)
          in [l, r]
}
```

1.3 UnionFind

```
/**
 Union find algorithm
 Complexity O(log n) for Join or Find.
*/

int pai[N];

void init(int n){
 for(int i=1; i<=n; i++){
  pai[i]=i;
 }
}

int find(int i){
 if(pai[i]==i)return i;
 return pai[i]=find(pai[i]);
}

int join(int a, int b){
 a=find(a);
 b=find(b);
 pai[a]=pai[b];
}
```

2 DP

2.1 ConvexHullTrick

```
/**
 * Source: Simon Lindholm
 * Description: Container where you can add lines of the
      form kx+m, and query maximum values at points x.
 *  Useful for dynamic programming.
 * Requires C++ 14
 * Use when dp[i] = max(m(j) * i + b(j)) where m and b are
      determined by some j < i
 * Negate everything to get min
 * Time: O(\log N)
 */

struct Line {
 mutable ll k, m, p;
 bool operator<(const Line& o) const { return k < o.k; }
 bool operator<(ll x) const { return p < x; }
};

struct LineContainer : multiset<Line, less<>> {
 // (for doubles, use inf = 1/.0, div(a,b) = a/b)
 const ll inf = LLONG_MAX;
 ll div(ll a, ll b) { // floored division
  return a / b - ((a ^ b) < 0 && a % b); }
 bool isect(iterator x, iterator y) {
  if (y == end()) { x->p = inf; return false; }
  if (x->k == y->k) x->p = x->m > y->m ? inf : -inf;
  else x->p = div(y->m - x->m, x->k - y->k);
  return x->p >= y->p;
 }
 void add(ll k, ll m) { //slope k, intercept m
  auto z = insert({k, m, 0}), y = z++, x = y;
  while (isect(y, z)) z = erase(z);
  if (x != begin() && isect(--x, y)) isect(x, y = erase(y));
  while ((y = x) != begin() && (--x)->p >= y->p)
   isect(x, erase(y));
 }
 ll query(ll x) { //max value at point x
  assert(!empty());
  auto l = *lower_bound(x);
  return l.k * x + l.m;
 }
};
```

3 Graph

3.1 Flow

3.1.1 Dinic

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

using namespace std;

const int MAXN = 2002; //XXX
//Set to the number of nodes in the flow graph.
const int MAXE = 2100012; //XXX
//Number of edges in the flow graph.

int from[MAXE], to[MAXE], cap[MAXE], prv[MAXE], head[MAXN],
     pt[MAXN], ec;

void addEdge(int u, int v, int uv, int vu = 0){
 from[ec] = u, to[ec] = v, cap[ec] = uv, prv[ec] = head[u],
      head[u] = ec++;
 from[ec] = v, to[ec] = u, cap[ec] = vu, prv[ec] = head[v],
      head[v] = ec++;
}

int lv[MAXN], q[MAXN];
bool bfs(int source, int sink){
 memset(lv, 63, sizeof(lv));
 int h = 0, t = 0;
 lv[source] = 0;
 q[t++] = source;
 while (t-h){
  int v = q[h++];
  for (int e = head[v]; ~e; e = prv[e])
   if (cap[e] && lv[v] + 1 < lv[to[e]]){
    lv[to[e]] = lv[v] + 1;
    q[t++] = to[e];
   }
 }
 return lv[sink] < 1e8;
}

int dfs(int v, int sink, int f = 1e9){
 if (v == sink || f == 0)
  return f;
 int ret = 0;
 for (int &e = pt[v]; ~e; e = prv[e])
  if (lv[v]+1 == lv[to[e]]){
   int x = dfs(to[e], sink, min(f, cap[e]));
   cap[e] -= x;
   cap[e^1] += x;
   ret += x;
   f -= x;
   if (!f)
    break;
  }
 return ret;
}

int dinic(int source, int sink){
 int ret = 0;
 while (bfs(source, sink)){
  memcpy(pt, head, sizeof(head));
  ret += dfs(source, sink);
 }
 return ret;
}

int main(){
 memset(head, -1, sizeof(head));
 return 0;
}
```

3.2 ShortestPath

3.2.1 Dijkstra

```
typedef long long ll;
typedef pair<int, int> pii;

#define F first
#define S second

const int MAXN = 1e5 + 10;

//Distance will be saved in d[]
int n, m, d[MAXN];
vector <pii> adj[MAXN];
set<pii> st;

void update(int v){
 for (auto e:adj[v]){
  int u = e.F, w = e.S;
  if (d[v]+w < d[u]){
   st.erase({d[u], u});
   d[u] = d[v]+w;
   st.insert({d[u], u});
  }
 }
}

void dijk(int v){
 memset(d, 63, sizeof(d));
 d[v] = 0;
 st.insert({d[v], v});
 while (st.size()) {
  int v = st.begin()->S;
  st.erase(st.begin());
  update(v);
 }
}
```

4 Math

4.1 NT

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

using namespace std;

typedef long long ll;
typedef pair<int, int> pii;

#define F first
#define S second

int main() {

 return 0;
}
```

5 Python

5.1 InputArray

```
n = int(input())
a = list(map(int, input().split()));
ans = 1
v = []
```

```
for i in range(1, n):
 if(a[i] == 1):
  ans += 1
  v.append(a[i - 1])
v.append(a[n - 1])
print(len(v))
print(' '.join(map(str, v)));
```

6 Strings

6.1 SuffixArray

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

using namespace std;

typedef long long ll;
typedef pair<int, int> pii;

#define rank asdkflj
#define F first
#define S second

const int MAXN = 1e5 + 10;
const int LOG = 18;

string s;
int rank[LOG][MAXN], n, lg;
pair<pair<int, int>, int> sec[MAXN];
int sa[MAXN], lc[MAXN];

int lcp(int a, int b){
 int _a = a;
 for (int w = lg-1; ~w && max(a, b) < n; w--)
  if (max(a, b) + (1<<w) <= n && rank[w][a] == rank[w][b])
   a += 1<<w, b += 1<<w;
 return a - _a;
}

int cnt[MAXN];
pair<pii, int> gec[MAXN];
void srt() {
 memset(cnt, 0, sizeof(cnt));
 for (int i = 0; i < n; i++) cnt[sec[i].F.S+1]++;
 for (int i = 1; i < MAXN; i++) cnt[i] += cnt[i-1];
 for (int i = 0; i < n; i++) gec[--cnt[sec[i].F.S+1]] = sec[i];
 memset(cnt, 0, sizeof(cnt));
 for (int i = 0; i < n; i++) cnt[gec[i].F.F+1]++;
 for (int i = 1; i < MAXN; i++) cnt[i] += cnt[i-1];
 for (int i = n-1; ~i; i--) sec[--cnt[gec[i].F.F+1]] = gec[i];
}

void build() {
 n = s.size();
 {
  int cur = 1; lg = 0;
  while (cur < n){
   lg++;
   cur <<= 1;
  }
  lg++;
 }

 for (int i = 0; i < n; i++) rank[0][i] = s[i];
 for (int w = 1; w < lg; w++){
  for (int i = 0; i < n; i++)
   if (i + (1<<w-1) >= n)
    sec[i] = {{rank[w-1][i], -1}, i};
   else
    sec[i] = {{rank[w-1][i], rank[w-1][i+(1<<w-1)]}, i};
  //sort(sec, sec + n);
  srt();

  rank[w][sec[0].S] = 0;
  for (int i = 1; i < n; i++)
   if (sec[i].F == sec[i-1].F)
    rank[w][sec[i].S] = rank[w][sec[i-1].S];
   else
    rank[w][sec[i].S] = i;
 }
 for (int i = 0; i < n; i++)
  sa[rank[lg-1][i]] = i;
 for (int i = 0; i + 1 < n; i++)
  lc[i] = lcp(sa[i], sa[i+1]);
}

int main(){

 return 0;
}
```

7 geo

7.1 ConvexHull

```
bool cmp(pt a, pt b){return mk(a.y, a.x) < mk(b.y, b.x);}

vector<pt> convexhull(vector<pt> p){ //counterclockwise, no collinear points
 sort(p.begin(), p.end(), cmp);
 p.erase(unique(p.begin(), p.end()), p.end());
 vector<pt> up, dn;
 for(pt i : p){
  while(up.size() > 1 and orient(up[up.size() - 2], up.back(), i) >= 0) up.pop_back();
  while(dn.size() > 1 and orient(dn[dn.size() - 2], dn.back(), i) <= 0) dn.pop_back();
  up.pb(i);
  dn.pb(i);
 }
 for(int i = (int) up.size() - 2; i >= 1; i--)
  dn.pb(up[i]);
 return dn;
}
```

7.2 template

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

using namespace std;

#define ll long long int
#define pb push_back
#define mk make_pair
#define mt make_tuple
#define fi first
#define se second
#define ii pair<int, int>
#define all(x) (x).begin(), (x).end()
#define N 1000007 // 10e6 + 7


struct stableSum {
 /*
  Use stableSum to add (positive) elements that are doubles.
  It greatly reduces imprecision.
 */
 int cnt = 0;
 vector<double> v, pref{0};
 void operator+=(double a) {
  assert(a >= 0);
  int s = ++cnt;
  while (s % 2 == 0) {
   a += v.back();
   v.pop_back(), pref.pop_back();
   s /= 2;
  }
  v.push_back(a);
  pref.push_back(pref.back() + a);
 }
 double val() {return pref.back();}
};



int quadRoots(double a, double b, double c, pair<double,
     double> &out) {
 /*
  quadRoots will give the quadratic answer to equation
   x^2*a + x*b + c for a!=0
  Returns how many solutions, place them in out.
 */
 assert(a != 0);
 double disc = b*b - 4*a*c;
 if (disc < 0) return 0;
 double sum = (b >= 0) ? -b-sqrt(disc) : -b+sqrt(disc);
 out = {sum/(2*a), sum == 0 ? 0 : (2*c)/sum};
 return 1 + (disc > 0);
}

/*
 Tips:
  - Use Integers whenever possible.
  - Minimize division and square root operations.
  - Try to write code that handles many situations at once.
*/

/*
 ---------------------------------------- Points
      ----------------------------------------
*/

typedef double T;
typedef complex<T> pt;
#define x real()
#define y imag()

// abs(p) = sqrt(x*x + y*y)
```

```
T sq(pt p) {return p.x*p.x + p.y*p.y;}

pt translate(pt v, pt p) {
 // Translate a point p by a vector v.
 return p+v;
}

pt scale(pt c, double factor, pt p) {
 // Scale point p by factor around a center c.
 return c + (p-c)*factor;
}

pt rot(pt p, double a) {
 // Rotate point p by an angle a, counterclockwise.
 return p * polar(1.0, a);
}

pt perp(pt p) {
 // Rotate point p by 90 degrees, good for integer coords.
 return {-p.y, p.x};
}

T dot(pt v, pt w) {
 /*
  v*w = |v|*|w|*cos(angle)
  Check sign of dot product to see if two vectors are going
     in the same dir.
   Positive if angle < pi/2, neg if >, 0 if =
 */
 return v.x*w.x + v.y*w.y;
}

bool isPerp(pt v, pt w) {return dot(v,w) == 0;}

double angle(pt v, pt w) {
 // Angle between two vectors.
 return acos(clamp(dot(v,w) / abs(v) / abs(w), -1.0, 1.0));
}

T cross(pt v, pt w) {
 /*
  v*w = |v|*|w|*sin(angle)
  Order of v, w matters! Angle is the ORIENTED angle between
       v and w.
   Positive if 0 < angle < pi, neg if -pi < angle < 0, zero
       if angle = 0 or pi.
 */
 return v.x*w.y - v.y*w.x;
}
```

```
T orient(pt a, pt b, pt c) {
 // I'll go from a to b to c. If turn left to c, positive.
      Right negative, straight zero.
 return cross(b-a,c-a);
}

double orientedAngle(pt a, pt b, pt c) {
 // Return the oriented angle between ab and ac, going from
      b to c.
 if (orient(a,b,c) >= 0)
  return angle(b-a, c-a);
 return 2*M_PI - angle(b-a, c-a);
}

bool inAngle(pt a, pt b, pt c, pt p) {
 // Use this to check if p lies in the angle that ab and ac
      form.
 assert(orient(a,b,c) != 0);
 if (orient(a,b,c) < 0) swap(b,c);
 return orient(a,b,p) >= 0 && orient(a,c,p) <= 0;
}

bool isConvex(vector<pt> p) {
 // To check if a polygon is convex, the orientation of all
      three consecutive
 // points should be the same.
 bool hasPos=false, hasNeg=false;
 for (int i=0, n=p.size(); i<n; i++) {
  int o = orient(p[i], p[(i+1)%n], p[(i+2)%n]);
  if (o > 0) hasPos = true;
  if (o < 0) hasNeg = true;
 }
 return !(hasPos && hasNeg);
}

bool half(pt p, pt v = {-1.0, 0.0}) { // true if in blue
    half
 // Modify v if you want a different starting angle.
 assert(p.x != 0 || p.y != 0); // the argument of (0,0) is
      undefined
 return cross(v,p) < 0 || (cross(v,p) == 0 && dot(v,p) < 0);
}

void polarSort(vector<pt> &v) {
 /*
  This will sort points according to their angle based on
      the origin.
  If I want to do the same thing but with a point not the
      origin, I have
  to subtract that point from all other points.
  If I want to add parameters in the sort such as magnitude, just add terms
 to the tuple.
 */
 sort(v.begin(), v.end(), [](pt v, pt w) {
  return make_tuple(half(v), 0) < make_tuple(half(w), cross(v,w));
 });
}

/*
 ---------------------------------------- Lines ----------------------------------------
*/

struct line {
 pt v; T c;
 // From direction vector v and offset c
 line(pt v, T c) : v(v), c(c) {}
 // From equation ax+by=c
 line(T a, T b, T c) : v(b,-a), c(c) {}
 // From points P and Q
 line(pt p, pt q) : v(q-p), c(cross(v,p)) {}

 // - these work with T = int
 T side(pt p);
 double dist(pt p);
 double sqDist(pt p);
 double slope();
 line perpThrough(pt p);
 bool cmpProj(pt p, pt q);
 line translate(pt t);
 // - these require T = double
 line shiftLeft(double dist);
 pt proj(pt p);
 pt refl(pt p);
};

T line::side(pt p) {
 // This says what side of the line a point is.
 // Positive side is on the left (remember the line has orientation).
 return cross(v,p)-c;
}

double line::dist(pt p) {
 // Dist point -> line
 return abs(side(p)) / abs(v);
}

double line::sqDist(pt p) {
 // Dist point -> line squared.
 return side(p)*side(p) / (double)sq(v);
}

double line::slope(){
 return v.y/v.x;
}

line line::perpThrough(pt p) {
 // Line that is perpendicular to this line, and goes through p.
 return {p, p + perp(v)};
}

bool line::cmpProj(pt p, pt q) {
 // Use this if you want to sort points through a line.
 return dot(v,p) < dot(v,q);
}

line line::translate(pt t) {
 // Translate this line by vector t.
 return {v, c + cross(v,t)};
}

line line::shiftLeft(double dist) {
 // Shift this line to the left by dist. Note: you gotta substitute.
 return {v, c + dist*abs(v)};
}

bool inter(line l1, line l2, pt &out) {
 // Check if l1 and l2 intersect.
 T d = cross(l1.v, l2.v);
 if (d == 0) return false;
 out = (l2.v*l1.c - l1.v*l2.c) / d; // requires floating-point coordinates
 return true;
}

pt line::proj(pt p) {
 // Projects a point into a line.
 return p - perp(v)*side(p)/sq(v);
}
pt line::refl(pt p) {
 // This is the point that is the same distance from line as p, but on the other side.
 return p - perp(v)*2.0*side(p)/sq(v);
}

line bisector(line l1, line l2, bool interior) {
 // This returns the line that is between l1 and l2, dividing the angle in 2.
 assert(cross(l1.v, l2.v) != 0); // l1 and l2 cannot be parallel!
 double sign = interior ? 1 : -1;
 return {l2.v/abs(l2.v) + l1.v/abs(l1.v) * sign, l2.c/abs(l2.v) + l1.c/abs(l1.v) * sign};
}

/*
 ---------------------------------------- Segments ----------------------------------------
*/

bool inDisk(pt a, pt b, pt p) {
 // Pts a, b are the diameter of a disk, want to know if point p is inside.
 return dot(a-p, b-p) <= 0;
}

bool onSegment(pt a, pt b, pt p) {
 // Check if point p is in the segment formed by [a, b].
 return orient(a,b,p) == 0 && inDisk(a,b,p);
}

bool properInter(pt a, pt b, pt c, pt d, pt &out) {
 // Check if two segments [a, b], [c, d] incercept.
 // The proper interception is an interception that is a single point, but not an endpoint.
 double oa = orient(c,d,a),
 ob = orient(c,d,b),
 oc = orient(a,b,c),
 od = orient(a,b,d);
 // Proper intersection exists iff opposite signs
 if (oa*ob < 0 && oc*od < 0) {
  out = (a*ob - b*oa) / (ob-oa);
  return true;
 }
 return false;
}


// To create sets of points we need a comparison function
struct cmpX {
 bool operator()(const pt &a, const pt &b) const{
  return make_pair(a.x, a.y) < make_pair(b.x, b.y);
 }
};
```

```
set<pt,cmpX> inters(pt a, pt b, pt c, pt d) {
 // If |set| = 0, no interception.
 // If |set| = 1, point interception.
 // If |set| = 2, segment interception.
 pt out;
 if (properInter(a,b,c,d,out)) return {out};
 set<pt,cmpX> s;
 if (onSegment(c,d,a)) s.insert(a);
 if (onSegment(c,d,b)) s.insert(b);
 if (onSegment(a,b,c)) s.insert(c);
 if (onSegment(a,b,d)) s.insert(d);
 return s;
}

double segPoint(pt a, pt b, pt p) {
 // Dist of point p to segment [a, b]
 if (a != b) {
  line l(a,b);
  if (l.cmpProj(a,p) && l.cmpProj(p,b)) // if closest to
      projection
   return l.dist(p); // output distance to line
 }
 return min(abs(p-a), abs(p-b)); // otherwise distance to A
      or B
}

double segSeg(pt a, pt b, pt c, pt d) {
 // Dist of seg [a, b] to seg [c, d]
 pt dummy;
 if (properInter(a,b,c,d,dummy))
  return 0;
 return min({segPoint(a,b,c), segPoint(a,b,d), segPoint(c,d,
      a), segPoint(c,d,b)});
}

/*
 ------------------------------------------ Poligons
      ------------------------------------------
*/

double areaTriangle(pt a, pt b, pt c) {
 return abs(cross(b-a, c-a)) / 2.0;
}

double areaPolygon(vector<pt> p) {
 double area = 0.0;
 for (int i = 0, n = p.size(); i < n; i++) {
  area += cross(p[i], p[(i+1)%n]); // wrap back to 0 if i ==
      n-1
 }
 return abs(area) / 2.0;
}

bool above(pt a, pt p) {
 // True if P at least as high as A (blue part).
 return p.y >= a.y;
}


bool crossesRay(pt a, pt p, pt q) {
 // Check if [PQ] crosses ray from A.
 return (above(a,q) - above(a,p)) * orient(a,p,q) > 0;
}

bool inPolygon(vector<pt> p, pt a, bool strict = true) {
 // Check if point a is in polygon p.
 // If strict, returns false when A is on the boundary.

 int numCrossings = 0;
 for (int i = 0, n = p.size(); i < n; i++) {
  if (onSegment(p[i], p[(i+1)%n], a))
   return !strict;
  numCrossings += crossesRay(a, p[i], p[(i+1)%n]);
 }
 return numCrossings & 1; // inside if odd number of
      crossings
}


/*
 ------------------------------------------ Circle
      ------------------------------------------
*/

pt circumCenter(pt a, pt b, pt c) {
 // Gives the center of the circle that goes though a, b, c.
 b = b-a, c = c-a; // consider coordinates relative to A
 assert(cross(b,c) != 0); // no circumcircle if A,B,C
      aligned
 return a + perp(b*sq(c) - c*sq(b))/cross(b,c)/2.0;
}

template <typename T> int sgn(T k) {
 // Return -1, 0, 1 depending on sign of k.
 return (T(0) < k) - (k < T(0));
}

int circleLine(pt o, double r, line l, pair<pt,pt> &out) {
 // Circle-Line intercection (0, 1, 2).
 // If only 1 intercection, out.fi == out.se.
 double h2 = r*r - l.sqDist(o);
 if (h2 >= 0) { // the line touches the circle
  pt p = l.proj(o); // point P
  pt h = l.v*sqrt(h2)/abs(l.v); // vector parallel to l, of
      length h
  out = {p-h, p+h};
 }
 return 1 + sgn(h2);
}

int circleCircle(pt o1, double r1, pt o2, double r2, pair<pt
     ,pt> &out) {
 // Circle-Circle intercection (0, 1, 2, inf).
 // Similar to circleLine.
 pt d=o2-o1; double d2=sq(d);
 if (d2 == 0) {assert(r1 != r2); return 0;} // concentric
      circles
 double pd = (d2 + r1*r1 - r2*r2)/2; // = |O_1P| * d
 double h2 = r1*r1 - pd*pd/d2; // = h2
 if (h2 >= 0) {
  pt p = o1 + d*pd/d2, h = perp(d)*sqrt(h2/d2);
  out = {p-h, p+h};
 }
 return 1 + sgn(h2);
}

int tangents(pt o1, double r1, pt o2, double r2, bool inner,
      vector<pair<pt,pt>> &out) {
 // There can be (0, 1, 2) tangents.
 // If 2 tangents, there are two pairs (p1, p2) of points of
      that tangent on the circles.
 // If 1 tangent, pairs are equal.
 if (inner) r2 = -r2;
 pt d = o2-o1;
 double dr = r1-r2, d2 = sq(d), h2 = d2-dr*dr;
 if (d2 == 0 || h2 < 0) {assert(h2 != 0); return 0;}
 for (double sign : {-1,1}) {
  pt v = (d*dr + perp(d)*sqrt(h2)*sign)/d2;
  out.push_back({o1 + v*r1, o2 + v*r2});
 }
 return 1 + (h2 > 0);
}


int main(int argc, char const *argv[]){
 pt p = {3.4, 2.1};
 cout << p << endl;
 return 0;
}
```

8 misc

8.1 template

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

using namespace std;
typedef long long ll;
typedef pair<int, int> pii;

#define F first
#define S second
#define se second
#define fi first
#define pb push_back
#define eb emplace_back
#define mk make_pair

#define N 1000007 //10e6 +7

int main(){
 ios::sync_with_stdio(false);

}
```