NAT: Nostalgic Alien Trespassers — TCR NWERC 2015

Magnus Selin

March 15, 2015

Contents

Import:
Import:

```
# mseln-algo-bookl
```

File: gcd.cpp
Author: Magnus Selin
Updated: 2014-12-07
Import:
Import:

```
int gcd(int a, int b){
  int t;
  while (b != 0){
    t = b;
    b = a % b;
    a = t;
  }
  return a;
}
```

File: vec3.cpp
Author: Magnus Selin
Updated: 2014-12-07
Description: Rational number class
Import:
Import: gcd.cpp.

```
using namespace std;

template<class T>
class Q{
private:
  T p, q;
public:
  Q(){}
  Q(T a, T b){
    p = a; q = b;
    if(q < 0){p = -p; q = -q;}
    if(p == 0) q = 1;
    if(q == 0){
      // Denominator == 0 -> Handle error
      q = 1;
    }
    T g = gcd(p, q);
    p /= g; q /= g;
  }
  Q operator+(Q a){ return Q(a.p*q + p*a.q, a.q*q); }
  Q operator-(Q a){ return Q(p*a.q - a.p*q, a.q*q); }
  Q operator*(Q a){ return Q(a.p*p, a.q*q); }
  Q operator/(Q a){ return Q(p*a.q, q*a.p); }
  void operator=(Q a){p = a.p; q = a.q;}
  bool operator==(Q a){
    Q f = *this
    Q s = Q(a.p, a.q);
    return (f.p == s.p and f.q == s.q);
  }
};
```

File: lis.cpp
Author: Magnus Selin
Updated: 2015-03-15
Import:
Import:

```
vi lis(vi a){
  vi lis;   // Longest increasing subsequnece
  vi s;    // s[i] min number of last number in lis of length i
  vi ind;   // Index of s[i] in a
  vi pre;   // Index to previous in subsequence

  s.push_back(a[0]);
  ind.push_back(0);
  pre.resize(a.size(), -1);

  for(int i = 1; i < a.size(); i++){
    // Start by looking at a list of length 1, and then increase by 1 every
    // step. i=1; a[0] is known to be in lis.
    if(a[i] > s[s.size()-1]){
      // Append at end
      s.push_back(a[i]);
      ind.push_back(i);
      pre[i] = ind[ind.size()-2];
    }
    else{
      vii mit = upper_bound(s.begin(), s.end(), a[i]);
      int m = distance(s.begin(), mit);
      // Ex: {1 (3) 5 7 9} <-- 2 :: replace 2 and 3

      s[m] = a[i];
      ind[m] = i;
      pre[i] = ind[m-1];
      if(m == 0) pre[i] = -1;
    }
  }

  int i = ind[ind.size()-1];
  while(i != -1){
    lis.push_back(a[i]);
    i = pre[i];
  }
  reverse(lis.begin(), lis.end());

  return lis;
}
```

File: segment_segment_intersection.cpp
Author: Magnus Selin
Updated: 2014-12-14
Import:
Import:

```
typedef vec2<double> vd2;
bool segment_segment_intersection(vd2 p, vd2 p1, vd2 q, vd2 q1){
  vd2 r = p1 - p;
  vd2 s = q1 - q;

  double t = ((q-p)%s)/(r%s);
  double u = ((q-p)%r)/(r%s);

  if(r%s == 0 and (q-p)%r == 0){
    if((0 <= (q-p)*r and (q-p)*r <= r*r) or
      (0 <= (p-q)*s and (p-q)*s <= s*s))
      return true;
      // Collinear overlapping
    else
      return false;
      // Collinear not overlapping
  }
  else if(r%s == 0 and (q-p)%r != 0){
    return false;
    // Parallel
  }
  else if(0 <= t and t <= 1 and 0 <= u and u <= 1){
    return true;
```

```cpp
    // Intersecting at p + tr = q + us
  }
  else{
    return false;
    // Neither parallel nor do they not intersect
  }
}
```

File: three_points_to_circle.cpp
Author: Magnus Selin
Updated: 2014-12-11
Input: Three arbitrary points in 2d space.
Output: Middle point and radius of the only circle going through the three points.
Import:
Import: datastructures/vec3.cpp, datastructures/homogenous_coord.cpp.

```cpp
vector<double> three_points_to_circle(vec3<double> v1, vec3<double> v2, vec3<double> v3){
  vec3<double> m1 = (v2 - v1)/2 + v1;
  vec3<double> m2 = (v3 - v2)/2 + v2;

  vec3<double> n1 = vec3<double>(v2[1]-v1[1], -(v2[0]-v1[0]), 1);
  vec3<double> n2 = vec3<double>(v3[1]-v2[1], -(v3[0]-v2[0]), 1);

  vec3<double> a1 = n1 + m1;
  vec3<double> a2 = n2 + m2;

  h<double> h1(m1[0],m1[1],1);
  h<double> h2(a1[0],a1[1],1);
  h<double> h3(m2[0],m2[1],1);
  h<double> h4(a2[0],a2[1],1);

  h<double> l1 = h1*h2;
  h<double> l2 = h3*h4;

  h<double> pm = (l1*l2).norm();

  double r = vec3<double>(pm.x-v1[0], pm.y-v1[1], 0).abs();

  vector<double> ret_val;
  ret_val.push_back(pm.x);
  ret_val.push_back(pm.y);
  ret_val.push_back(r);

  printf("%lf␣%lf␣%lf\n", pm.x, pm.y, r);

  return ret_val;
}
```

File: vec3.cpp
Author: Magnus Selin
Updated: 2014-12-07
Description: Class for vectors of arbitrary type.
Import: iostream, cmath.
Import:

```cpp
template <class T>
class vec3 {
  private:
    T v[3];
  public:
    vec3(){}
    vec3(T a, T b, T c){
      v[0] = a; v[1] = b; v[2] = c;
    }

    T operator[](int i){ return v[i];}
    vec3 operator=(vec3 o){ v[0] = o[0]; v[1] = o[1]; v[2] = o[2]; }
    vec3 operator+(vec3 o){
      return vec3(v[0] + o[0], v[1] + o[1], v[2] + o[2]);
    }
    vec3 operator-(vec3 o){
      return vec3(v[0] - o[0], v[1] - o[1], v[2] - o[2]);
    }
    vec3 operator-(){
      return vec3(-v[0], -v[1], -v[2]);
    }
    double operator*(vec3 o){
      return v[0]*o[0] + v[1]*o[1] + v[2]*o[2];
    }
    vec3 operator*(double o){
      return vec3(v[0]*o, v[1]*o, v[2]*o);
    }
    vec3 operator/(double o){
      return vec3(v[0]/o, v[1]/o, v[2]/o);
    }
    vec3 operator%(vec3 o){
      return vec3(
          v[1]*o[2] - v[2]*o[1],
          v[2]*o[0] - v[0]*o[2],
          v[0]*o[1] - v[1]*o[0]);
    }

    void operator+=(vec3 o){ *this = *this + o; }
    void operator-=(vec3 o){ *this = *this - o; }
    void operator*=(double o){ *this = *this * o; }

    bool operator==(vec3 o){
      return v[0] == o[0] and v[1] == o[1] and v[2] == o[2];
    }
    bool operator!= (vec3 o){
      return !(*this == o);
    }

    double abs(){
      return sqrt(v[0]*v[0] + v[1]*v[1] + v[2]*v[2]);
    }
    double ang(vec3 o){
      if(*this != vec3(0,0,0) and o != vec3(0,0,0))
        return acos((*this * o) / (this->abs()*o.abs()));
      return 0;
    }
    vec3 norm(){
      if(*this != vec3(0,0,0))
        return (*this) / this->abs();
      return *this;
    }
};

template<typename T> inline std::ostream &operator<<(std::ostream &os, vec3<T> &v){
  os << "(" << v[0] << ",␣" << v[1] << ",␣" << v[2] << ")";
  return os;
}
```

File: homogenous_coord.cpp
Author: Magnus Selin
Updated: 2014-12-07
Description: Class handling homogeneous cordinates and lines
Import: cmath.
Import: ../numerical/gcd.cpp.

```cpp
template <class T>
class h{
  public:
    T x, y, z;
    h(T a, T b, T c){x=a; y=b; z=c;}
    h(T a, T d1, T b, T d2, T c){x=a*d2; y=b*d1; z=c*d1*d2;}
    h(){}

    h<T> operator*(h o){ return h<T>(y*o.z-o.y*z, z*o.x-o.z*x, o.x*y-x*o.y);}

    h<T> norm(){
      if(z!=0){
        if(x/z * z != x or y/z * z != y){
          int g = gcd(x, gcd(y,z));
          return h<T>(x/g, y/g, z/g);
        }
        else return h<T>(x/z, y/z, 1);
      }
      return h<T>(1,1,0);
    }
};
```

File: template.cpp
Author: Magnus Selin
Updated: 2014-12-07
Description: Standard template
Import: iostream, cstdlib, cstdio, cmath, vector, set, map, stack, queue, string, bitset, algorithm, cstring.
Import:

```
using namespace std;

#define rep(i, a, b) for(int i = (a); i < int(b); ++i)
#define trav(it, v) for(typeof((v).begin()) it = (v).begin(); it != (v).end(); ++it)

typedef double fl;
typedef long long ll;
typedef pair<int, int> pii;
typedef vector<int> vi;


bool solve(){

  return true;
}

int main(){
  int tc=1; //scanf("%d", &tc);
  rep(i, 0, tc) solve();

  return 0;
}
```