Team Note of HYEA

hyea

Compiled on April 10, 2018

Contents

1 Graph

1.1 Hopcroft-Karp Bipartite Matching

Usage: Use `add_edge` to add edges, `match` to get matching, `getcover` to calculate vertex cover.

Time Complexity: $\mathcal{O}(E\sqrt{V})$

```
const int MAXN = 50005, MAXM = 50005;
vector<int> gph[MAXN];
int dis[MAXN], l[MAXN], r[MAXM], vis[MAXN];
void clear(){ for(int i=0; i<MAXN; i++) gph[i].clear();  }
void add_edge(int l, int r){ gph[l].push_back(r); }
bool bfs(int n){
  queue<int> que;
  bool ok = 0;
  memset(dis, 0, sizeof(dis));
  for(int i=0; i<n; i++){
    if(l[i] == -1 && !dis[i]){
      que.push(i);
      dis[i] = 1;
    }
  }
  while(!que.empty()){
    int x = que.front();
    que.pop();
    for(auto &i : gph[x]){
      if(r[i] == -1) ok = 1;
      else if(!dis[r[i]]){
        dis[r[i]] = dis[x] + 1;
        que.push(r[i]);
      }
    }
  }
  return ok;
}
bool dfs(int x){
  for(auto &i : gph[x]){
    if(r[i] == -1 || (!vis[r[i]] && dis[r[i]] == dis[x] + 1 &&
dfs(r[i]))){
      vis[r[i]] = 1; l[x] = i; r[i] = x;
      return 1;
    }
  }
  return 0;
}
int match(int n){
  memset(l, -1, sizeof(l));
```

```
  memset(r, -1, sizeof(r));
  int ret = 0;
  while(bfs(n)){
    memset(vis, 0, sizeof(vis));
    for(int i=0; i<n; i++) if(l[i] == -1 && dfs(i)) ret++;
  }
  return ret;
}
bool chk[MAXN + MAXM];
void rdfs(int x, int n){
  if(chk[x]) return;
  chk[x] = 1;
  for(auto &i : gph[x]){
    chk[i + n] = 1;
    rdfs(r[i], n);
  }
}
vector<int> getcover(int n, int m){ // solve min. vertex cover
  match(n);
  memset(chk, 0, sizeof(chk));
  for(int i=0; i<n; i++) if(l[i] == -1) rdfs(i, n);
  vector<int> v;
  for(int i=0; i<n; i++) if(!chk[i]) v.push_back(i);
  for(int i=n; i<n+m; i++) if(chk[i]) v.push_back(i);
  return v;
}
```

1.2 General Matching

Usage: Use `init` to init, `addEdge` to add edges, `match` to get matching, `Match` to find maximum matching. Vertices have 1-based index.

Time Complexity: $\mathcal{O}(VE)$

```
const int MAXN = 2020 + 1;
struct GM { // 1-based Vertex index
  int vis[MAXN], par[MAXN], orig[MAXN], match[MAXN], aux[MAXN], t,
N;
  vector<int> conn[MAXN];
  queue<int> Q;
  void addEdge(int u, int v) {
    conn[u].push_back(v); conn[v].push_back(u);
  }
  void init(int n) {
    N = n; t = 0;
    for(int i=0; i<=n; ++i) {
      conn[i].clear();
      match[i] = aux[i] = par[i] = 0;
    }
  }
  void augment(int u, int v) {
    int pv = v, nv;
    do {
      pv = par[v]; nv = match[pv];
      match[v] = pv; match[pv] = v;
      v = nv;
    } while(u != pv);
  }
  int lca(int v, int w) {
    ++t;
    while(true) {
      if(v) {
        if(aux[v] == t) return v; aux[v] = t;
        v = orig[par[match[v]]];
      }
      swap(v, w);
    }
  }
  void blossom(int v, int w, int a) {
    while(orig[v] != a) {
      par[v] = w; w = match[v];
      if(vis[w] == 1) Q.push(w), vis[w] = 0;
      orig[v] = orig[w] = a;
      v = par[w];
    }
  }
  bool bfs(int u) {
    fill(vis+1, vis+1+N, -1); iota(orig + 1, orig + N + 1, 1);
    Q = queue<int> (); Q.push(u); vis[u] = 0;
    while(!Q.empty()) {
      int v = Q.front(); Q.pop();
      for(int x: conn[v]) {
```

```
        if(vis[x] == -1) {
          par[x] = v; vis[x] = 1;
          if(!match[x]) return augment(u, x), true;
          Q.push(match[x]); vis[match[x]] = 0;
        }
        else if(vis[x] == 0 && orig[v] != orig[x]) {
          int a = lca(orig[v], orig[x]);
          blossom(x, v, a); blossom(v, x, a);
        }
      }
    }
    return false;
  }
  int Match() {
    int ans = 0;
    //find random matching (not necessary, constant improvement)
    vector<int> V(N-1); iota(V.begin(), V.end(), 1);
    shuffle(V.begin(), V.end(), mt19937(0x94949));
    for(auto x: V) if(!match[x]){
      for(auto y: conn[x]) if(!match[y]) {
        match[x] = y, match[y] = x;
        ++ans; break;
      }
    }
    for(int i=1; i<=N; ++i) if(!match[i] && bfs(i)) ++ans;
    return ans;
  }
};
```

2 Data Structure

2.1 Randomized Meldable Heap

Usage: Min-heap `H` is declared as `Heap<T> H`. You can use `push`, `size`, `empty`, `top`, `pop` as `std::priority_queue`. Use `H.meld(G)` to meld contents from `G` to `H`.

Time Complexity: $\mathcal{O}(logn)$

```
namespace Meldable {
mt19937 gen(0x94949);
template<typename T>
struct Node {
  Node *l, *r;
  T v;
  Node(T x): l(0), r(0), v(x){}
};
template<typename T>
Node<T>* Meld(Node<T>* A, Node<T>* B) {
  if(!A) return B; if(!B) return A;
  if(B->v < A->v) swap(A, B);
  if(gen()&1) A->l = Meld(A->l, B);
  else A->r = Meld(A->r, B);
  return A;
}
template<typename T>
struct Heap {
  Node<T> *r; int s;
  Heap(): r(0), s(0){}
  void push(T x) {
    r = Meld(new Node<T>(x), r);
    ++s;
  }
  int size(){ return s; }
  bool empty(){ return s == 0;}
  T top(){ return r->v; }
  void pop() {
    Node<T>* p = r;
    r = Meld(r->l, r->r);
    delete p;
    --s;
  }
  void Meld(Heap x) {
    s += x->s;
    r = Meld(r, x->r);
  }
};
}
```

3 Geometry

3.1 Smallest Enclosing Circle

Usage: Use `solve` with `vector<Point>`. It returns `Circle c`, `c.p` is center, `c.r` is radius.

Time Complexity: $\mathcal{O}(n)$

```
namespace cover_2d{
  double eps = 1e-9;
  using Point = complex<double>;
  struct Circle{ Point p; double r; };
  double dist(Point p, Point q){ return abs(p-q); }
  double area2(Point p, Point q){ return (conj(p)*q).imag();}
  bool in(const Circle& c, Point p){ return dist(c.p, p) < c.r +
eps; }
  Circle INVAL = Circle{Point(0, 0), -1};
  Circle mCC(Point a, Point b, Point c){
    b -= a; c -= a;
    double d = 2*(conj(b)*c).imag(); if(abs(d)<eps) return INVAL;
    Point ans = (c*norm(b) - b*norm(c)) * Point(0, -1) / d;
    return Circle{a + ans, abs(ans)};
  }
  Circle solve(vector<Point> p) {
    mt19937 gen(0x94949); shuffle(p.begin(), p.end(), gen);
    Circle c = INVAL;
    for(int i=0; i<p.size(); ++i) if(c.r<0 ||!in(c, p[i])){
      c = Circle{p[i], 0};
      for(int j=0; j<=i; ++j) if(!in(c, p[j])){
        Circle ans{(p[i]+p[j])*0.5, dist(p[i], p[j])*0.5};
        if(c.r == 0) {c = ans; continue;}
        Circle l, r; l = r = INVAL;
        Point pq = p[j]-p[i];
        for(int k=0; k<=j; ++k) if(!in(ans, p[k])) {
          double a2 = area2(pq, p[k]-p[i]);
          Circle c = mCC(p[i], p[j], p[k]);
          if(c.r<0) continue;
          else if(a2 > 0 && (l.r<0||area2(pq, c.p-p[i]) > area2(pq,
l.p-p[i]))) l = c;
          else if(a2 < 0 && (r.r<0||area2(pq, c.p-p[i]) < area2(pq,
r.p-p[i]))) r = c;
        }
        if(l.r<0&&r.r<0) c = ans;
        else if(l.r<0) c = r;
        else if(r.r<0) c = l;
        else c = l.r<=r.r?l:r;
      }
    }
    return c;
  }
};
```