

Data Structures and Algorithms for competitive programming

Eoin Davey

November 18, 2018

Contents

1	Graphs	2	6	Maths	9
	Traversal	2		Miller Rabin	9
	BFS	2		Binomial Coefficients	9
	Dijkstras	2			
	Trees	2			
	MST	2			
	LCA	2			
	Centroid Decomposition	3			
	SCC Tarjans	3			
	AP & Bridges	3			
	Network Flow	3			
	Edmond Karp Max Flow	3			
	Ford Fulkerson Max Flow	4			
2	Data Structures	5			
	Fenwick Tree	5			
	UFDS	5			
	Sparse Table	5			
	Segment Tree	5			
3	Geometry	6			
	Convex Hull	6			
	Geometry Axioms	6			
4	Strings	7			
	Suffix Array	7			
	Trie	7			
	KMP	7			
5	Algorithms	8			
	NlogN LIS	8			
	RectInHist	8			

1 Graphs

Traversal

BFS

```
1  int dist[MXN];
2  vector<int> adjList[MXN];
3
4  int main(){
5      for(int i = 0; i < MXN; i++)
6          dist[i]=INF;
7      queue<int> q;
8      q.push(0);
9      dist[0] = 0;
10     while(!q.empty()){
11         int u = q.front(); q.pop();
12         int d = dist[u];
13         for(int i : adjList[u]){
14             if(dist[i]==INF){
15                 dist[i]=d+1;
16                 q.push(i);
17             }
18         }
19     }
20     return 0;
21 }
```

Dijkstras

```
1  struct path {
2      int u,d;
3      bool operator < (const path& p) const {
4          return d > p.d;
5      }
6  };
7  for(int i =0; i < N; ++i)
8      dist[i] = INF;
9  dist[S] = 0;
10 priority_queue<path> q;
11 q.push(path{S,0});
12 while(!q.empty()){
13     path p = q.top(); q.pop();
14     u = p.u,d = p.d;
15     if(dist[u] < d)
16         continue;
17     for(auto v : adjList[u]){
18         nd = d + v.second;
19         if(nd < dist[v.first]){
20             dist[v.first] = nd;
21             q.push(path{v.first ,nd});
22         }
23     }
24 }
```

Trees

MST

```
1  struct edge {
2      int x,y,w;
3      bool operator < (edge e) const {
4          return w < e.w;
5      }
6  };
7
8  int main(){
9      vector<edge> eList; //Input
10     for(int i = 0; i < N; i++)// Set up UFDS
11         p[i]=i;
12     vector<ii> treeList;
13     sort(eList.begin(),eList.end());
14     int cost = 0;
15     int sz=N;
16     int u,v,w;
17     for(const auto &i : eList){
18         v=i.x; u=i.y; w=i.w;
19         if(!connected(u,v)){
20             join(u,v);
21             treeList.push_back({min(u,v),max(u,v)});
22             sz--;
23             cost+=w;
24         }
25     }
26     if(sz!=1)
27         puts("Impossible");
28 }
```

LCA

```
1  /*
2   * H[u] is first visit of u
3   * E[x] is vertex at time x
4   * L[x] is depth at time x
5   */
6  void vis(int u, int d){
7      H[u]=vind;
8      E[vind] = u;
9      L[vind++] = d;
10     for(auto i : adjList[u]){
11         if(H[i]!=-1)
12             continue;
13         vis(i,d+1);
14         E[vind] = u;
15         L[vind++] = d;
16     }
17 }
18
19 int LCA(int u, int v){
20     if(H[u] > H[v]){
```

```

21         int t = u;
22         u = v;
23         v = t;
24     }
25     //run some range min query on L
26     //between H[u] and H[v]
27     int ind = rmq(H[u], H[v]);
28     return E[ind];
29 }
30
31 int dist(int u, int v){
32     int a = H[u];
33     int b = H[v];
34     int ind = LCA(u, v);
35     return abs(L[H[ind]] - L[a])
36         + abs(L[H[ind]] - L[b]);
37 }

```

Centroid Decomposition

```

1 void fill_sz(int u, int p){
2     sz[u] = 1;
3     for(int v : adjList[u]){
4         if(v==p || mkd[v])
5             continue;
6         fill_sz(v, u);
7         sz[u] += sz[v];
8     }
9 }
10
11 int get_centroid(int u, int n, int p){
12     for(int v : adjList[u]){
13         if(v==p || mkd[v])
14             continue;
15         if(sz[v] > n/2)
16             return get_centroid(v, n, u);
17     }
18     return u;
19 }
20
21 int decomp(int u){
22     fill_sz(u, -1);
23     int cent = get_centroid(u, sz[u], -1);
24     mkd[cent] = true;
25     for(int v : adjList[cent]){
26         if(mkd[v])
27             continue;
28         int r = decomp(v);
29         centP[r] = cent;
30     }
31     return cent;
32 }

```

SCC Tarjans

```

1 stack<int> scc;
2 int dfsCounter=1;
3 int sccIdx=1;
4 map<int, int> sccMap;
5
6 void tarjans(int u){
7     scc.push(u);
8     vis[u]=true;
9
10    dfs_low[u]=dfs_num[u]=dfsCounter++;
11
12    for(int i = 0; i < adjList[u].size(); i++){
13        int v = adjList[u][i];
14        if(dfs_num[v]==0){
15            tarjans(v);
16            dfs_low[u]=min(dfs_low[u], dfs_low[v]);
17        } else if(vis[v]){
18            dfs_low[u]=min(dfs_low[u], dfs_num[v]);
19        }
20    }
21    if(dfs_low[u]==dfs_num[u]){
22        while(1){
23            int v = scc.top(); scc.pop();
24            sccMap[v]=sccIdx;
25            vis[v]=false;
26            if(v==u)
27                break;
28        }
29        sccIdx++;
30    }
31 }

```

AP & Bridges

```

1 int dfs(int u, int p){
2     dfs_num[u] = dfs_low[u] = ++dfs_counter;
3     for(auto v : adjList[u]){
4         if(dfs_num[v]==0){
5             dfs(v, u);
6             if(dfs_low[v] >= dfs_num[u]){
7                 articulation[u]=true;
8             }
9             if(dfs_low[v] > dfs_num[u])
10                bridge = true;
11            dfs_low[u] = min(dfs_low[u], dfs_low[v]);
12        } else if(v!=p)
13            dfs_low[u] = min(dfs_low[u], dfs_num[v]);
14    }
15 }

```

Network Flow

Edmond Karp Max Flow

```

1 void aug(int u, int minE){
2     if(u==S){ f=minE; return; }
3     if(p[u]!=u){
4         aug(p[u], min(minE, res[p[u]][u]));
5         res[p[u]][u]-=f;
6         res[u][p[u]]+=f;
7     }
8 }
9
10 int main(){
11     int mf=0;
12     for(;;){
13         f=0; // Global
14         for(int i = 0; i < N; i++){
15             dist[i]=INF, p[i]==i;
16             dist[S]=0;
17             queue<int> q; q.push(S);
18             while(!q.empty()){
19                 int u = q.front(); q.pop();
20                 if(u==T) break;
21                 for(int i = 0; i < N; i++){
22                     if(res[u][i] > 0 && dist[i]==INF)
23                         dist[i]=dist[u]+1, p[i]=u, q.push(i);
24                 }
25                 aug(T, INF);
26                 if(f==0) break;
27                 mf+=f;
28             }
29             vector<ii> used;
30             for(int i = 0; i < N; i++){
31                 for(int j = 0; j < N; j++){
32                     if(graph[i][j] > 0 && res[i][j] < graph[i][j])
33                         used.push_back(make_pair(i, j));
34             }

```

Ford Fulkerson Max Flow

```

1 int ff(int u, int minE){
2     if(u==T)
3         return minE;
4     vis[u]=true;
5     for(auto i : adjList[u]){
6         if(!vis[i] && res[u][i] > 0){
7             if(int f = ff(i, min(minE, res[u][i]))){
8                 res[u][i] -= f;
9                 res[i][u] += f;
10                return f;
11            }
12        }
13    }
14    return 0;
15 }
16
17 int main(){
18     int mf = 0;

```

```

19     while(1){
20         memset(vis, 0, sizeof(vis));
21         int f = ff(S, INF);
22         if(f==0)
23             break;
24         mf+=f;
25     }
26     printf("%d\n", mf);
27 }

```

2 Data Structures

Fenwick Tree

```
1 int tree[MXN];
2 int N;
3 int lsOne(int i){ return i&(-i); }
4 void update(int k,int v){
5     for(; k<MXN; k+=lsOne(k))
6         tree[k]+=v;
7 }
8 int query(int k){
9     int cnt=0;
10    for(; k; k-=lsOne(k)){
11        cnt+=tree[k];
12    }
13    return cnt;
14 }
```

UFDS

```
1 int find(int u){ return p[u] = (p[u] == u ? u : find(p[u])); }
2
3 inline void join(int a, int b){
4     pa = find(a);
5     pb = find(b);
6     if(pa!=pb){
7         if(rank[pa] < rank[pb]){
8             ni = pb;
9             pb = pa;
10            pa = ni;
11        }
12        p[pb] = pa;
13        if(rank[pa]==rank[pb])
14            rank[pa]++;
15    }
16 }
```

Sparse Table

```
1 inline int rmq(int u, int v){
2     if(u > v)
3         return -2000000000;
4     int k=(int) floor(log2((double)(v-u+1)));
5     if(r[mtable[u][k]] >
6        r[mtable[v-(1<<k)+1][k]])
7         return mtable[u][k];
8     return mtable[v-(1<<k) + 1][k];
9 }
10
11 for(int i = 0; i < N; i++)
12     mtable[i][0] = i;
13 for(int j = 1; (1 << j) <= N; j++)
```

```
14     for(int i = 0; i + (1<<j)-1 < N; ++i)
15         if(r[mtable[i][j-1]]
16            >r[mtable[i+(1<<(j-1))][j-1]])
17             mtable[i][j]= mtable[i][j-1];
18     else
19         mtable[i][j]=mtable[i+(1<<(j-1))][j-1];
```

Segment Tree

```
1 int tree[MXN*4 + 2];
2 int a[MXN];
3 int N;
4
5 void construct(int p, int L, int R){
6     if(L==R){
7         tree[p] = a[L];
8         return;
9     }
10    if(R<L)
11        return;
12    int md = (L+R)/2;
13    construct(2*p,L,md);
14    construct(2*p+1,md+1,R);
15    tree[p] = min(tree[2*p], tree[2*p+1]);
16 }
17
18 void update(int p, int L, int R, int ind,int v){
19     if(L==R){
20         a[ind] = v;
21         tree[p] = v;
22         return;
23     }
24    int md = (L+R)/2;
25    if(ind <= md)
26        update(2*p,L,md,ind,v);
27    else
28        update(2*p+1,md+1,R,ind,v);
29    tree[p] = min(tree[2*p], tree[2*p+1]);
30 }
31
32 int rmq(int p, int L, int R, int l, int r){
33     if(r < L || l > R)
34         return INF;
35     if(l>=L && r<=R)
36         return tree[p];
37     int md = (l+r)/2;
38     return min(rmq(2*p,L,R,l,md),rmq(2*p+1,L,R,md+1,r));
39 }
```

3 Geometry

Convex Hull

```
1 int main(){
2     for(int i = 0; i < N; i++){
3         perm[i]=i;
4     }
5     sort(perm,perm+N,
6         [](int a, int b){
7             const point &pa = V[a];
8             const point &pb = V[b];
9             if(real(pa)!=real(pb))
10                return real(pa) < real(pb);
11            return imag(pa) < imag(pb);
12        });
13     vector<int> L; vector<int> U;
14     for(int i = 0; i < N;){
15         int t = L.size();
16         if(t >= 2 && !ccw(V[L[t-2]],V[L[t-1]],V[perm[i]]))
17             L.pop_back();
18         else
19             L.push_back(perm[i++]);
20     }
21     for(int i = N-1; i >=0;){
22         int t = U.size();
23         if(t >= 2 && !ccw(V[U[t-2]],V[U[t-1]],V[perm[i]]))
24             U.pop_back();
25         else
26             U.push_back(perm[i--]);
27     }
28     vector<int> hull;
29     for(int i = 0; i < L.size()-1; ++i)
30         hull.push_back(L[i]);
31     for(int i = 0; i < U.size()-1; ++i)
32         hull.push_back(U[i]);
33     return 0;
34 }
```

Geometry Axioms

```
1 typedef complex<double> pt;
2 typedef complex<double> vec;
3 typedef vector<pt> pgon;
4 typedef struct { pt p,q; } lseg;
5 double cross(const vec& a, const vec &b){
6     return x(a)*y(b)-y(a)*x(b);
7 }
8 //cross product of (b-a) and (c-b), 0 is collinear
9 int orientation(const pt& a,
10     const pt& b, const pt& c){
11     double v = cross(b-a,c-b);
12     if(abs(v-0.0)<EPS)
13         return 0;
```

```
14     return v > 0 ? 1 : 2;
15 }
16 //Line segment intersection
17 bool intersects(const lseg& a, const lseg& b){
18     if(a.q == b.p || b.q == a.p)
19         return false;
20     if(orientation(a.p,a.q,b.p)
21         !=orientation(a.p,a.q,b.q)
22         && orientation(b.p,b.q,a.p)
23         != orientation(b.p,b.q,a.q))
24         return true;
25     return false;
26 }
27 //Area of polygon
28 double area(const pgon& p){
29     double area = 0.0;
30     for(int i = 1; i < p.size(); ++i)
31         area+=cross(p[i-1],p[i]);
32     return abs(area)/2.0;
33 }
34 //If a->b->c is a counterclockwise turn
35 double ccw(const point& a, const point& b,
36     const point& c){
37     if(a==b || b==c || a==c)
38         return false;
39     point relA = b-a;
40     point relC = b-c;
41     return cross(relA,relC) >= 0.0;
42 }
43 //Returns if point p is in the polygon poly
44 bool inPoly(const pgon& poly, const pt& p){
45     for(int i = 0; i < poly.size()-1; i++){
46         if(!ccw(poly[i],p,poly[i+1]))
47             return false;
48     }
49     return true;
50 }
51 //Distance from p to line (a,b)
52 double distToLine(const pt& p, const pt& a,
53     const pt &b){
54     vec ap = p-a;
55     vec ab = b-a;
56     double u = dot(ap,ab)/dot(ab,ab);
57     //Ignore for non-line segment
58     if(u < 0.0) //Closer to a
59         return abs(a-p);
60     if(u > 1.0) //Closer to b
61         return abs(b-p);
62     pt c = a+ab*u; // This is the point
63     return abs(c-p);
64 }
```

4 Strings

Suffix Array

```
1 void countingSort(int k){
2     int i,sum,maxi=max(300,N);
3     memset(c,0,sizeof(c));
4     for(i = 0; i < N; i++){
5         c[i+k < N ? RA[i+k] : 0]++;
6     }
7     for(i=sum=0; i < maxi; i++){
8         int t = c[i];
9         c[i]=sum;
10        sum+=t;
11    }
12    for(i = 0; i < N; i++){
13        tempSA[c[SA[i]+k < N
14            ? RA[SA[i]+k]: 0]++] = SA[i];
15    }
16 }
17
18 int main(){
19     for(int i = 0; i < N; i++){
20         SA[i]=i,RA[i]=input[i];
21     }
22     int r;
23     for(int k = 1; k < N; k <= 1){
24         countingSort(k);
25         countingSort(0);
26         tempRA[SA[0]]=r=0;
27         for(int i = 1; i < N; i++){
28             tempRA[SA[i]]
29             =(RA[SA[i]]==RA[SA[i-1]]
30             && RA[SA[i]+k]==RA[SA[i-1]+k]
31             ? r:++r);
32         }
33         for(int i = 0; i < N; i++){
34             RA[i]=tempRA[i];
35         }
36     }
37     return 0;
38 }
```

Trie

```
1 struct node {
2     node * children[26];
3     int count;
4     node(){
5         memset(children,0,sizeof(children));
6         count=0;
7     }
8 };
9
10 void insert(node* nd, char *s){
11     if(*s){
```

```
12         if(!nd->children[*s-'a'])
13             nd->children[*s-'a']=new node();
14         insert(nd->children[*s-'a'],s+1);
15     }
16     nd->count++;
17 }
18
19 int count(node* nd, char *s){
20     if(*s){
21         if(!nd->children[*s-'a'])
22             return 0;
23         return count(nd->children[*s-'a'],s+1);
24     } else {
25         return nd->count;
26     }
27 }
```

KMP

```
1 vector<int> buildFailure(string s){
2     vector<int> T(n+1,0);
3     T[0]=-1;
4     int j = 0;
5     for(int i = 1; i < s.size();i++){
6         if(s[i]==s[j]){
7             T[i]=T[j];
8             j++;
9         } else{
10            T[i] = j;
11            j = T[j];
12            while(j >= 0 && s[i]!=s[j])
13                j = T[j];
14            j++;
15        }
16    }
17    T[s.size()] = j;
18    return T;
19 }
20 vector<int> search(string W, string S){
21     auto T=buildFailure(W);
22     vector<int> p;
23     int k = 0;
24     int j = 0;
25     while(j < S.size()){
26         if(W[k]==S[j]){
27             k++; j++;
28             if(k==W.size()){
29                 p.push_back(j-k);
30                 k = T[k];
31             }
32         } else{
33             k = T[k];
34             if(k < 0)
35                 j+=1, k+=1;
36         }
37     }
```

```

37     }
38     return p;
39 }

```

5 Algorithms

NlogN LIS

```

1  int ls[MXN];
2  int L[MXN];
3  int I[MXN];
4
5  void nlogn() {
6      for(int i = 1; i < N+1; ++i)
7          I[i]=INF;
8      I[0] = -INF;
9      int mx = 1;
10     for(int i = 0; i < N; ++i){
11         int ind = lower_bound(I, I+N+1, ls[i]) - I;
12         I[ind] = ls[i];
13         L[i] = ind;
14         mx = max(mx, ind);
15     }
16     int prv = INF;
17     vector<int> out;
18     for(int i = N-1; i >= 0; --i){
19         if(ls[i] < prv && L[i]==mx){
20             out.push_back(ls[i]);
21             prv = ls[i];
22             mx--;
23         }
24     }
25 }

```

RectInHist

```

1  int R,C;
2  char board[MXRC][MXRC];
3  int h[MXRC][MXRC];
4
5  int perim(int l, int w){
6      if(l==0 || w==0)
7          return 0;
8      return 2*l + 2*w;
9  }
10
11 int main(){
12     for(int i = 0; i < R; i++){
13         int run=0;
14         for(int j = 0; j < C; j++){
15             run = (board[i][j]=='.'?run+1:0);
16             h[i][j] = run;
17         }
18     }
19     int mx = 0;
20     for(int j = 0; j < C; j++){
21         stack<int> s;
22         for(int i = 0; i < R; i++){

```



```

23     if(s.empty()
24         ||h[i][j]>h[s.top()][j])
25         s.push(i);
26     else if(h[i][j]<h[s.top()][j]){
27         while(!s.empty()
28             &&h[i][j]<h[s.top()][j]){
29             int l = h[s.top()][j];
30             s.pop();
31             int pm = perim(l,
32                 (s.empty()?
33                     i:i-s.top()-1));
34             mx = max(mx,pm);
35         }
36         s.push(i);
37     } else if(h[i][j]==h[s.top()][j]){
38         s.pop();
39         s.push(i);
40     }
41 }
42 while(!s.empty()){
43     int l = h[s.top()][j]; s.pop();
44     int pm = perim(l, s.empty()? R : R - s.top()-1);
45     mx = max(mx,pm);
46 }
47 }
48 printf("%d\n",mx-1);
49 }

```

6 Maths

Miller Rabin

```

1 void factor(ll x, ll& e, ll& k){
2     while(x%2LL==0LL){
3         x/=2LL;
4         ++e;
5     }
6     k = x;
7 }
8
9 //increase x for higher certainty, 5 works well
10 bool is_prime(ll n, int x){
11     if(n&2LL==0 || n==1LL)
12         return false;
13     if(n==2 || n==3 || n==5 || n==7)
14         return true;
15     ll e, k;
16     factor(n-1,e,k);
17     while(x-->0){
18         ll a = (rand())%(n-5LL) + 2LL;
19         ll p = mod_exp(a,k,n);
20         if(p==1LL || p==n-1LL)
21             continue;
22         bool all_fail = true;
23         for(int i = 0; i < e-1; ++i){
24             p = mod_exp(p, 2, n);
25             if(p==n-1LL){
26                 all_fail = false;
27                 break;
28             }
29         }
30         if(all_fail)
31             return false;
32     }
33     return true;
34 }

```

Binomial Coefficients

```

1 ll ncrmem[MXN][MXN];
2
3 ll ncr(int n, int r){
4     if(n==0)
5         return r==0;
6     if(r==0)
7         return 1;
8     if(ncrmem[n][r] != -1)
9         return ncrmem[n][r];
10    return ncrmem[n][r] = ncr(n-1, r-1) + ncr(n-1, r);
11 }

```

```

1  void factor(ll x, ll& e, ll& k){
2      while(x%2LL==0LL){
3          x/=2LL;
4          ++e;
5      }
6      k = x;
7  }
8
9  //increase x for higher certainty, 5 works well
10 bool is_prime(ll n, int x){
11     if(n&2LL==0 || n==1LL)
12         return false;
13     if(n==2 || n==3 || n==5 || n==7)
14         return true;
15     ll e, k;
16     factor(n-1,e,k);
17     while(x-->0){
18         ll a = (rand())%(n-5LL) + 2LL;
19         ll p = mod_exp(a,k,n);
20         if(p==1LL || p==n-1LL)
21             continue;
22         bool all_fail = true;
23         for(int i = 0; i < e-1; ++i){
24             p = mod_exp(p, 2, n);
25             if(p==n-1LL){
26                 all_fail = false;
27                 break;
28             }
29         }
30         if(all_fail)
31             return false;
32     }
33     return true;
34 }

```