Graph Algorithm

1. Dijkstra's/Prim's Algorithm

```
#include <cstdio>
#include <vector>
#include <queue>
#define N (int)1e5
#define P pair<int, int>
#define F first
#define S second
#define INF 1<<30
using namespace std;
vector<P> g[N+2];
priority_queue<P> q;
int n, m, sum = 0;
int d[N+2];
void SSSP() {
    for(int i = 1; i <= n; i++)
        d[i] = INF;
    q.push( P(0, 1) );
    while( !q.empty() ) {
        int u = q.top().S, dis = -q.top().F;
        q.pop();
        if(d[u] <= dis)       continue;
        d[u] = dis;
        for(int i = 0; i < g[u].size(); i++) {
            int v = g[u][i].F, w = g[u][i].S;
            if(d[u] + w <= d[v])
                q.push( P(-d[u]-w, v) );
        }
    }
}
int MST() {
    int sum = 0;
    for(int i = 1; i <= n; i++)
        d[i] = INF;
    q.push( P(0, 1) );
    while( !q.empty() ) {
        int u = q.top().S, dis = -q.top().F;
        q.pop();
        if(d[u] <= dis)       continue;
        d[u] = dis;
        sum += dis;
        for(int i = 0; i < g[u].size(); i++) {
            int v = g[u][i].F, w = g[u][i].S;
            if(w <= d[v])
                q.push( P(-w, v) );
        }
    }
    return sum;
}
int main() {
    scanf("%d %d", &n, &m);
    while(m--) {
        int s, e, w;
        scanf("%d %d %d", &s, &e, &w);
        g[s].push_back( P(e, w) );
        g[e].push_back( P(s, w) );
    }
    return 0;
}
```

2. Union-find algorithm

```
#include <csdio>
#define N (int)1e5
int root[N+2];
int find(int x) {
    if(x != r[x])   r[x] = find(r[x]);
    return r[x];
}
void union(int x, int y) {
    int rootx = find(x), rooty = find(y);
    root[rootx] = rooty;
}
int main() {
    //Initialization
    for(int i = 1; i <= N; i++)
        root[i] = i;
    return 0;
}
```

3. Least Common Ancestor

```
#include <cstdio>
#include <queue>
#include <vector>
#include <algorithm>
#define L long long
#define P pair<L, int>
#define F first
#define S second
#define INF (L)1<<60
using namespace std;
vector<P> g[100002];
priority_queue<P> q;
int n, k;
int s, e, v;
int p[100002], l[1000002], dp[100002][20];
L w;
L d[100002];
void dfs(int u,int lvl) {
    l[u] = lvl;
    for(int i = 0; i < g[u].size(); i++) {
        v = g[u][i].S;
        if(p[u] == v)       continue;
        p[v] = u;
        dfs(v, lvl+1);
    }
}
int LCA(int x, int y) {
    if(l[x] < l[y])     swap(x, y);
    int lg;
    for(lg = 1; 1<<lg <= l[x]; lg++);
    lg--;
    for(int i = lg; i >= 0; i--)
        if(l[x] - (1<<i) >= l[y])
            x = dp[x][i];
    if(x == y)      return x;
    for(int i = lg; i >= 0; i--)
        if(dp[x][i] != -1 && dp[x][i] != dp[y][i])
            x = dp[x][i], y = dp[y][i];
    return p[x];
}
int main() {
```

```
    scanf("%d %d", &n, &k);
    for(int i = 1; i < n; i++) {
        scanf("%d %d %lld", &s, &e, &w);
        g[s].push_back( P(w, e) );
        g[e].push_back( P(w, s) );
    }
//Find parents for all nodes.
    dfs(0, 0);
//Find LCA using DP
    for(int i = 0; i < n; i++)
        for(int j = 0; 1<<j < n; j++)
            dp[i][j] = -1;
    for(int i = 0; i < n; i++)
        dp[i][0] = p[i];
    for(int j = 1; 1<<j < n; j++)
        for(int i = 0; i < n; i++)
            if(dp[i][j-1] != -1)
                dp[i][j] = dp[ dp[i][j-1] ][j-1];
    return 0;
}
```

4. Topological Sort

```
#include <cstdio>
#include <cstring>
#include <vector>
#include <queue>
#include <stack>
#define N (int)1e5
#define P pair<int, int>
using namespace std;
vector<int> g[40010];
vector<int> topo;
stack<int> zero;
int n, m, t;
int deg[40010];     //in-degree,index
bool mark[40010];   //is visited?
bool DAG() {
    for(int i = 1; i <= n; i++) {
        queue<int> q;
        memset(mark, false, sizeof(mark));
        q.push(i);
        while(!q.empty()) {
            int now = q.front();
            q.pop();
            mark[now] = true;
            for(int j = 0; j < g[now].size(); j++) {
                if(g[now][j] == i)      return false;
                if(!mark[g[now][j]])
q.push(g[now][j]);
            }
        }
    }
    return true;
}
int main() {
    scanf("%d %d", &n, &m);
    while(m--) {
        int str, end;
        scanf("%d %d", &str, &end);
        g[str].push_back(end);
        deg[end]++;
    }
    if(!DAG())  printf("no topological order\n");
    else {
```

```
        for(int i = 1; i <= n; i++)
            if(!deg[i])     zero.push(i);
        while(!zero.empty()) {
            int now = zero.top();
            zero.pop();
            topo.push_back(now);
            for(int i = 0; i < g[now].size(); i++) {
                deg[g[now][i]]--;
                if(!deg[g[now][i]])
                    zero.push(g[now][i]);
            }
        }
        for(int i = 0; i < topo.size(); i++)
            printf("%d ", topo[i]);
        printf("\n");
    }
    return 0;
}
```

5. Bipartite Coloring

```
#include <cstdio>
#include <cstring>
#include <vector>
#define N (int)1e5
using namespace std;
vector<int> adj[N+2];
int mark[N+2];
bool dfs(int now, int color) {
    mark[now] = color;
    for(int i = 0; i < adj[now].size(); i++) {
        if(mark[adj[now][i]] == color)   return
false;
        if(!mark[adj[now][i]]) {
            if(!dfs(adj[now][i], -color)    return
false;
        }
    }
    return true;
}
int main() {
    int n, m;
    scanf("%d %d", &n, &m);
    while(m--) {
        int str, end;
        scanf("%d %d",&str,&end);

adj[str].push_back(end),adj[end].push_back(str);
    }
    for(int i = 1; i <= n; i++) {
        if(!mark[i]) {
            if(!dfs(i,1)){
                printf("G is not bipartite\n");
                break;
            }
        }
        if(i == n)    printf("G is bipartite\n");
    }
    return 0;
}
```

Data structures

1. Fenwick Tree

```
#include <cstdio>
#define N (int)1e5
int ft[N+2];
int ft2[N+2];
void update(int *ft, int pos, int val) {
    for(; pos <= N; pos += -pos & pos)
        ft[pos] += val;
}
int query(int *ft, int pos) {
    int sum = 0;
    for(; pos > 0; pos -= -pos & pos)
        sum += ft[pos];
    return sum;
}
//In case of range-update & range-query, add these
functions.
void range_update(int from, int to, int val) {
    update(ft, from, val);      update(ft, to+1, -
val);
    update(ft2, from, val*(from-1));
    update(ft2, to+1, -val*to);
}
int query(int pos) {
    return query(ft, pos) * pos - query(ft2, pos);
}
int range_query(int from, int to) {
    return query(to) - query(from-1);
}
```

2. Segment Tree

```
#include <cstdio>
#define N (int)1e5
int st[6*N+2];
void build(int s, int e, int nw){
    if(s == e) {
        //Depends on problems.
        return;
    }
    int m = (s+e)>>1;
    build(s, m, nw<<1);     bd(m+1, e, nw<<1|1);
    //Depends on problems.
}
void update(int s, int e, int nw, int in, int val) {
    if(s == e){
        //Depends on problems.
        return;
    }
    int m = (s+e)>>1;
    if(in <= m)     update(s, m, nw<<1, in, val);
    else    update(m+1, e, nw<<1|1, in, val);
    //Depends on problems.
}
int query(int s, int e, int nw, int l, int r) {
    if(s > e || e < l || s > r)     return 0;
    if(l <= s && e <= r)
        return //Depends on problems.
    int m = (s+e)>>1;
    return //Depends on problems, basically calls
    //qr(s, m, nw<<1, l, r); and
```

```
    //qr(m+1, e, nw<<1|1, l, r);

}
```

3. Lazy propagation

```
#include <cstdio>
#define N (int)1e5
int st[6*N+2], lz[6*N+2];
void ud(int s, int e, int nw, int l, int r, int val)
{
    if(chk[nw]) {
        st[nw] = //depends on problems.
        if(s != e) {    //Lazy propagation
            lz[nw<<1] = lz[nw];
            chk[nw<<1] = true;
            lz[nw<<1|1] = lz[nw];
            chk[nw<<1|1] = true;
        }
        chk[nw] = false;
    }
    //Out of interval.
    if(l > e || r < s || s > e)     return;
    if(l <= s && e <= r){
        st[nw] = //depends on problems.
        if(s != e) {    //Lazy propagation
            lz[nw<<1] = val;    chk[nw<<1] = true;
            lz[nw<<1|1] = val;  chk[nw<<1|1] = true;
        }
        return;
    }
    int m = (s+e)>>1;
    ud(s ,m, nw<<1, l, r, val);     //Left-side
    ud(m+1, e, nw<<1|1, l, r, val); //Right-side
    st[nw] = //depends on problems.
}
L qr(int s, int e, int nw, int l, int r) {
    //Out of interval
    if(l > e || r < s || s > e)     return 0;
    if(chk[nw]) {
        st[nw] = //depends on problems.
        if(s != e) {    //Lazy propagation
            lz[nw<<1] = lz[nw];
            chk[nw<<1] = true;
            lz[nw<<1|1] = lz[nw];
            chk[nw<<1|1] = true;
        }
        chk[nw] = false;
    }
    if(l <= s && e <= r)    return st[nw];
    int m = (s+e)>>1;
    return //depends on problems, basically calls
    //qr(s, m, nw<<1, l, r); and
    //qr(m+1, e, nw<<1|1, l, r);
}
```

4. Convex Hull

```
#include <bits/stdc++.h>
#define L long long
#define N (int)1e5
using namespace std;
struct P {
    L x, y;
} a[N+2];
```

```
int n;
stack<P> s;
deque<P> ans;
//Find square of distance
L ds(P p1, P p2){
    return (p1.x - p2.x)*(p1.x - p2.x) +
           (p1.y - p2.y)*(p1.y - p2.y);
}
// To find orientation of ordered triplet (p, q, r).
// The function returns following values
// 0 --> p, q and r are co-linear
// 1 --> Clockwise
// 2 --> Counterclockwise
int chk(P p1, P p2, P p3){
    L t = (p3.y - p1.y)*(p2.x - p1.x) - (p3.x -
p1.x)*(p2.y - p1.y);
    if(t == 0)     return 0;
    if(t < 0)      return 1;
    return 2;
}
bool cmp(P p1, P p2){
    int t = chk(a[0], p1, p2);
    if(t == 0)
        return ds(a[0], p1) < ds(a[0], p2);
    return (t == 2);
}
P secTop() {
    P t = s.top();
    s.pop();
    P v = s.top();
    s.push(t);
    return v;
}
int main() {
    scanf("%d", &n);
    for(int i = 0; i < n; i++)
        scanf("%lld %lld", &a[i].x, &a[i].y);
    int mn = 0;
    //Find the left-most point or the bottom-most
    //point if equal
    for(int i = 1; i < n; i++)
        if(a[i].x < a[mn].x)      mn = i;
        else if(a[i].x == a[mn].x && a[i].y <
a[mn].y)  mn = i;
    swap(a[0], a[mn]);
    //Sort the remaining point
    sort(a+1, a+n, cmp);
    int m = 1; // Initialize size of modified array
    for(int i = 1; i < n; i++){
       // Keep removing i while angle of i and i+1 is
same
       // with respect to p0
        while(i < n-1 && chk(a[0], a[i], a[i+1]) !=
2)
            i++;
        a[m++] = a[i];
    }
    //Push first 3 points into stack
    for(int i = 0; i < 3; i++)
        s.push(a[i]);
    for(int i = 3; i < m; i++) {
        while(chk(secTop(), s.top(), a[i]) != 2)
            s.pop();
        s.push(a[i]);
```

```
    }
    while(!s.empty()){
        ans.push_front(s.top());
        s.pop();
    }
    if(ans[0].x == ans[ans.size()-1].x){
        P t = ans[ans.size()-1];
        ans.pop_back();
        ans.push_front(t);
    }
    printf("%d\n", ans.size());
    for(int i = 0; i < ans.size(); i++)
        printf("%lld %lld\n", ans[i].x, ans[i].y);
    return 0;
}
```

5. AVL Tree

```
#include<cstdio>
#define max(x,y) x>y? x:y
typedef struct AVL{
    int key,h;
    AVL *l,*r;
    AVL(int key) : key(key),l(NULL),r(NULL),h(1){}
}AVL;
AVL *root[100002];
int t,n,m,k,x,y,r[100002],cnt;
int findh(AVL* p){
    return p? p->h : 0;
}
int bfac(AVL* p){
    return findh(p->r) - findh(p->l);
}
void uph(AVL* p){
    p->h=max(findh(p->l),findh(p->r))+1;
}
AVL* rotateright(AVL* p){
    AVL *q=p->l;
    p->l=q->r;
    q->r=p;
    uph(p);     uph(q);
    return q;
}
AVL* rotateleft(AVL* p){
    AVL *q=p->r;
    q->r=p->l;
    p->l=q;
    uph(p);     uph(q);
    return q;
}
AVL* balance(AVL* p){
    uph(p);
    int fac=bfac(p);
    if(fac==2){
        if(bfac(p->r)<0)    p->r=rotateright(p->r);
        return rotateleft(p);
    }
    if(fac==-2){
        if(bfac(p->l)>0)    p->l=rotateleft(p->l);
        return rotateright(p);
    }
    return p;
}
```

```
AVL* insert(AVL* p,int k){
    if(!p)  return new AVL(k);
    if(k < p->key)  p->l=insert(p->l,k);
    else    p->r=insert(p->r,k);
    return balance(p);
}
void del(AVL* p){
    if(!p)  return ;
    del(p->l);      del(p->r);
    root[x]=insert(root[x],p->key);
    delete p;
}
void ans(AVL* p){
    if(!p)      return ;
    ans(p->r);
    cnt++;
    if(p->key==x)   return ;
    ans(p->l);
}
```

6. Min/Max deque

```
#include <cstdio>
#include <deque>
#define N (int)1e5
using namespace std;
deque<int> mn;
int n, m, a[N+2];
int main() {
    scanf("%d %d", &n, &m);
    for(int i = 1; i < m; i++) {
        scanf("%d", a+i);
        while(!mn.empty() && a[mn.back()] > a[i])
            mn.pop_back();
        mn.push_back(i);
    }
    for(int i = m; i <= n; i++) {
        scanf("%d", a+i);
        while(!mn.empty() && a[mn.back()] > a[i])
            mn.pop_back();
        mn.push_back(i);
        while(mn.front() <= i-m)  mn.pop_front();
        printf("%d\n", *mn.front());
    }
    return 0;
}
```

Algorithms

1. Longest Increasing Subsequence

```
#include <cstdio>
#include <algorithm>
#include <set>
using namespace std;
set<int> s;
set<int>::iterator it;
int n, x;
int main() {
    scanf("%d", &n);
    for(int i = 0; i < n; i++) {
        scanf("%d", &x);
        s.insert(x);
```

```
        it = upper_bound(s.begin(), s.end(), x);
        if(it != s.end())     s.erase(it);
    }
    printf("%d\n", s.size());
    return 0;
}
```

2. Merge sort/Counting Inversion

```
#include <cstdio>
#define N (int)1e5
int n;
int a[N+2], t[N+2];
int merge(int s, int m, int e) {
    int l = s, r = m+1;
    int cnt = 0;
    for(int i = s; i <= e; i++) {
        if(l > m)           t[i] = a[r++];
        else if(r > e)      t[i] = a[l++];
        else if(a[l] <= a[r])   t[i] = a[l++];
        else {
            t[i] = a[r++];
            cnt += m-l+1;
        }
    }
    for(int i = s; i <= e; i++)
        a[i] = t[i];
    return cnt;
}
int merge_sort(int s, int e) {
    if(s >= e)      return 0;
    int m = (s+e)>>1;
    return merge_sort(s, m) + merge_sort(m+1, e)
        + merge(s, m, e);
}
int main() {
    scanf("%d", &n);
    for(int i = 0; i < n; i++)
        scanf("%d", a+i);
    merge_sort(0, n-1);
    return 0;
}
```

3. KMP Algorithm

```
#include <cstdio>
#define N (int)1e5
int a[N+2], b[N+2];
int t[N+2] = {-1}, n, m, cnt=0;
int main() {
    scanf("%d %d", &n, &m);
    for(int i = 0; i < n; i++)
        scanf("%d", a+i);
    for(int i = 0; i < m; i++)
        scanf("%d", b+i);
    int nowa = 0, nowb = 0, pos = 2, cnd = 0;
    while(pos < m) {
        if(b[pos-1] == b[cnd])    t[pos++] = ++cnd;
        else if(cnd > 0)    cnd = t[cnd];
        else    t[pos++] = 0;
    }
    while(nowa + nowb < n) {
        if(b[nowb] == a[nowa+nowb]) {
```

```
            if(nowb == m-1) {
                cnt++;
                nowa += nowb-t[nowb];
                nowb = t[nowb];
            }
            else    nowb++;
        }
        else {
            if(t[nowb] > -1) {
                nowa += nowb-t[nowb];
                nowb = t[nowb];
            }
            else {
                nowa++;     nowb=0;
            }
        }
    }
    printf("%d\n", cnt);
    return 0;
}
```

4. Closest Pair Problem

```
#include <cstdio>
#include <cmath>
#include <algorithm>
#include <vector>
#define INF 1e18
using namespace std;
struct pnt {
    double x, y;
};
typedef vector<pnt> point;
point a;    //List of points
bool cmpx(pnt a,pnt b) { return a.x < b.x;  }
bool cmpy(pnt a,pnt b) { return a.y < b.y;  }
double dis(pnt a,pnt b) {
    return pow(a.x - b.x, 2) + pow(a.y - b.y, 2);
}
double cls_strip(point p) {
    double Min = INF;
    for(int i = 0; i < p.size()-1; i++)
        for(int j = i+1; j < p.size() && (p[j].y-
p[i].y) < Min; j++)
            Min = min(Min, dis(p[i], p[j]) );
    return Min;
}
double cls(point px, point py) {
    if(px.size() <= 3)
        return cls_strip(px);
    int mid = px.size()/2;
    point pyl, pyr;
    for(int i = 0; i < py.size(); i++)
        if(px[i].x <= px[mid].x)
            pyl.push_back(py[i]);
        else    pyr.push_back(py[i]);
    point pxl = point(px.begin(), px.begin()+mid);
    point pxr = point(px.begin()+mid, px.end());
    double d = min(cls(pxl, pyl), cls(pxr, pyr));
    point tmp;
    for(int i = 0; i < px.size(); i++)
        if(abs(px[i].x-px[mid].x) < d)
            tmp.push_back(px[i]);
```

```
    return min(d, cls_strip(tmp));
}
double closest(point p) {
    point px = p, py = p;
    sort(px.begin(), px.end(), cmpx);
    sort(py.begin(), py.end(), cmpy);
    return cls(px, py);
}
```

Miscellaneous

1. Constructor & Bool operator overriding

```
struct S {
    //Variables
    S(//Parameters):
        t(t),x(x),y(y){}
    bool operator < (const S &a) const {
        //Overriding
    }
};
```

2. Standard Template Library

```
//Universal (Not recommended)
#include <bits/stdc++.h>
//Algorithm
#include <algorithm>
//Containers
#include <vector>
#include <queue>
#include <stack>
#include <set>
#include <map>
#include <list>
```

3. EOF Input

```
#include <cstdio>
int main() {
    while(scanf(/*parameters*/) != EOF) {
        //Source code
    }
    return 0;
}
```