

Universidade Federal de Itajubá

ACM ICPC Reference

.++

Dêner José Ribeiro Eduardo Augusto de Oliveira Leonardo Furtado de Oliveira

> Coach Felipe Kallás Silva

Contents

1	Gra	phs and Trees	2
	1.1	Depth-First Search (DFS)	2
	1.2	Breadth-First Search (BFS)	2
	1.3	Dijkstra	3
	1.4	Bellman-Ford	3
	1.5	Floyd-Warshall	4
	1.6	Kruskal	4
	1.7	Ford-Fulkerson	5
	1.8	Tarjan	6
	1.9	Fenwick Tree	6
	1.10	Segment Tree	7
2	Dyn	namic Programming	8
	2.1	Merge-Sort	8
	2.2	Longest Increasing Subsequence (LIS)	8
	2.3	Longest Common Subsequence (LCS)	9
	2.4	Kadane	9
	2.5		10
	2.6	•	10
	2.7	Binary Search Bounds	11
3	Mat	th and Geometry	12
	3.1	· · · · · · · · · · · · · · · · · · ·	12
	3.2		$\frac{12}{12}$
	3.3	•	12
	3.4		12
	3.5		13
	3.6		13
	3.7	•	13
	3.8		13
	3.9		14^{-3}
	3.10		14
		· · · · · · · · · · · · · · · · · · ·	 14
			 14
			15^{-1}
		v o	15^{-3}
4	Tem	nplates	17
_			17
	4.2	Line	
	4.3		18
5	Mis	cellaneous 1	19
9			19
	V		

* The codes listed here use the following header.

```
#include <bits/stdc++.h>
   using namespace std;
   #define ii pair<int, int>
   #define mk make_pair
   #define vi vector<int>
   #define vii vector<ii>>
   #define vb vector<bool>
   #define vs vector<string>
10
   #define pb push_back
11
   #define 11 long long
12
   #define INF 1000000 // Large Number
13
   #define MOD 100000007
14
   #define MAXN 1000000 // Problem Limit
```

1 Graphs and Trees

1.1 Depth-First Search (DFS)

```
Time: O(V + E)
```

1.2 Breadth-First Search (BFS)

Time: O(V + E)

```
int n;
   vi graph[MAXN];
   vi bfs(int src, int w = 1) {
       queue<int> q;
       vi dist(n, INT_MAX);
       vb vis(n);
       q.push(src);
       dist[src] = 0;
10
       while (!q.empty()) {
11
           int u = q.front();
           q.pop();
13
           vis[u] = true;
14
           for (int v : graph[u]) {
15
               if (!vis[v] && dist[u] + w < dist[v]) {</pre>
16
                   dist[v] = dist[u] + w;
17
                   q.push(v);
           }
20
21
       return dist;
22
   }
```

1.3 Dijkstra

Time: $O(n^2 log n)$

```
int n;
   vii graph[MAXN];
   vi dijkstra(int src) {
       priority_queue<ii, vii, greater<ii>>> pq;
       vi dist(n, INT_MAX);
       vb vis(n, false);
       pq.push(mk(0, src));
       dist[src] = 0;
10
       while (!pq.empty()) {
11
12
           int u = pq.top().second;
           pq.pop();
13
           vis[u] = true;
14
           for (int i : graph[u] {
              int w = i.first;
16
               int v = i.second;
               if (!vis[v] && dist[v] > dist[u] + w) {
18
                  dist[v] = dist[u] + w;
19
                  pq.push(mk(dist[v], v));
20
               }
           }
22
       return dist;
24
   }
```

1.4 Bellman-Ford

Time: O(V * E)

```
struct edge {
       int u, v, w;
       edge() {};
       edge(int _u, int _v, int _w) {
           u = _u, v = _v, w = _w;
5
6
   };
   int n, m;
10
   vector<edge> graph;
11
12
   bool bellman(int src) {
       vi dist(n, INT_MAX);
13
14
       dist[src] = 0;
15
       for (int i = 0; i < n - 1; i++)</pre>
16
           for (edge e : graph)
17
               if (dist[e.u] != INT_MAX)
18
                   dist[e.v] = min(dist[e.u] + e.w, dist[e.v]);
19
       for (edge e : graph)
20
           if (dist[e.u] != INT_MAX && dist[e.u] + e.w < dist[e.v])</pre>
21
               return true;
22
       return false;
23
   }
24
```

1.5 Floyd-Warshall

```
Time: O(n^3)
```

1.6 Kruskal

Time: O(nlogn)

```
struct edge {
       int u, v, w;
        edge() {};
       edge(int _u, int _v, int _w) {
           u = _u, v = _v, w = _w;
       }
       bool operator < (const edge su) const {</pre>
           return w < su.w;</pre>
       }
9
   };
10
11
   int n, m, root[MAXN];
12
    vector<edge> graph;
13
14
15
   int findset(int u) {
       return root[u] == u ? u : root[u] = findset(root[u]);
16
   }
   void initset() {
19
       for (int i = 0; i < n; i++) root[i] = i;</pre>
20
21
22
   int kruskal() {
23
       initset();
24
       sort(graph.begin(), graph.end());
25
       vi tree[n];
26
27
       int total = 0;
       for (int i = 0; i < m; i++) {</pre>
28
           int u = graph[i].u;
29
           int v = graph[i].v;
30
           int w = graph[i].w;
31
32
           int fu = findset(u);
33
           int fv = findset(v);
34
           if (fu != fv) {
35
               root[fu] = fv;
36
               total += w;
               tree[u].pb(v);
           }
39
       }
40
       return total;
41
   }
42
```

1.7 Ford-Fulkerson

Time: $O(EV^3)$

```
int n, flow[MAXN][MAXN], parent[MAXN];
   vector<int> graph[MAXN];
   bool bfs(int src, int snk) {
       memset(parent, -1, sizeof parent);
       queue<int> q;
6
       parent[src] = -2;
       q.push(src);
       while (!q.empty()) {
10
          int u = q.front();
11
           q.pop();
12
           for (int v : graph[u]) {
13
              if (parent[v] == -1 && flow[u][v]) {
14
                  parent[v] = u;
                  if (v == snk) return true;
16
                  q.push(v);
18
           }
19
20
21
       return false;
22
   int ford(int src, int snk) {
24
       int max_flow = 0;
25
       while (bfs(src, snk)) {
26
           int path_flow = INT_MAX;
27
           for (int v = snk; v != src; v = parent[v])
28
              path_flow = min(path_flow, flow[parent[v]][v]);
29
30
           for (int v = snk; v != src; v = parent[v]) {
31
               flow[parent[v]][v] -= path_flow;
32
               flow[v][parent[v]] += path_flow;
33
           }
34
35
           max_flow += path_flow;
36
       return max_flow;
37
   }
38
```

1.8 Tarjan

Time: O(V + E)

```
int n, disc[MAXN], low[MAXN], scc[MAXN], ct, id;
   vi graph[MAXN];
   stack<int> st;
   bool onstack[MAXN];
   void dfs(int u) {
       disc[u] = low[u] = ct++;
       st.push(u);
       onstack[u] = true;
10
       for (int v : graph[u]) {
11
           if (disc[v] == -1) {
12
               dfs(v);
13
               low[u] = min(low[u], low[v]);
14
           } else if (onstack[v])
               low[u] = min(low[u], disc[v]);
16
       if (low[u] == disc[u]) {
18
           int v;
19
           do {
20
21
               v = st.top();
22
               st.pop();
               onstack[v] = false;
               scc[v] = id;
           } while (u != v);
           id++;
26
       }
27
   }
28
29
   void tarjan() {
30
       ct = id = 0;
31
       memset(disc, -1, sizeof disc);
32
       for (int i = 0; i < n; i++)</pre>
           if (disc[i] == -1)
34
               dfs(i);
35
36
```

1.9 Fenwick Tree

Time: O(log n)

```
int n, bit[MAXN], v[MAXN];
   int query(int i) {
       int sum = 0;
       for (; i > 0; i -= i & (-i))
5
           sum += bit[i];
6
       return sum;
7
   }
8
   void update(int i, int value) {
       for (; i <= n; i += i & (-i))</pre>
11
           bit[i] += value;
12
13
14
   void build() {
15
       memset(bit, 0, sizeof bit);
16
       for (int i = 1; i <= n; i++)</pre>
17
           update(i, v[i-1]);
18
   }
```

1.10 Segment Tree

Build Time: O(n) | Query Time: O(log n) | Update Time: O(log n)

```
int v[MAXN], st[4 * MAXN], /* lazy[4 * MAXN] */;
   void build(int 1, int r, int no = 0) {
       if (1 > r) return;
       if (1 == r) {
           st[no] = v[1];
           return;
       }
       int mid = 1 + (r - 1) / 2;
       build(1, mid, no *2 + 1);
       build(mid + 1, r, no * 2 + 2);
       st[no] = st[no * 2 + 1] + st[no * 2 + 2];
13
   }
14
    /* void prop(int 1, int r, int no) {
       if (lazy[no] != 0) {
16
           st[no] += (r - 1 + 1) * lazy[no];
           if (1 != r) {
18
               lazy[no * 2 + 1] += lazy[no];
               lazy[no * 2 + 2] += lazy[no];
20
21
22
           lazy[no] = 0;
23
   } */
24
25
   int query(int 1, int r, int qs, int qe, int no = 0) {
26
       if (1 > r || 1 > qe || r < qs) return 0;</pre>
       // prop(1, r, no);
28
       if (1 >= qs && r <= qe) return st[no];</pre>
29
       int mid = 1 + (r - 1) / 2;
30
       return query(1, mid, qs, qe, 2 * no + 1) + query(mid + 1, r, qs, qe, 2 * no + 2);
31
   }
32
33
   void update(int 1, int r, int value, int pos, int no = 0) {
34
35
       if (1 > pos || r < pos) return;</pre>
       st[no] += value;
36
       if (1 == r) return;
37
       int mid = 1 + (r - 1) / 2;
38
       update(1, mid, value, pos, no * 2 + 1);
39
       update(mid + 1, r, value, pos, no * 2 + 2);
40
41
42
    /* void lazyUpdate(int 1, int r, int qs, int qe, int value, int no = 0) {
43
       if (1 > r || 1 > qe || r < qs) return;
44
       prop(l, r, no);
45
       if (1 \ge qs \&\& r \le qe) {
46
           st[no] += (r - 1 + 1) * value;
47
           if (1 != r) {
48
               lazy[no * 2 + 1] += value;
49
               lazy[no * 2 + 2] += value;
50
           }
51
           return;
52
       }
53
       int mid = 1 + (r - 1) / 2;
54
       lazyUpdate(1, mid, qs, qe, value, no * 2 + 1);
55
       lazyUpdate(mid + 1, r, qs, qe, value, no * 2 + 2);
56
       st[no] = st[no * 2 + 1] + st[no * 2 + 2];
57
   } */
58
```

^{*} Lazy propagation commented.

2 Dynamic Programming

2.1 Merge-Sort

Time: O(nlogn)

```
int merge(int v[], int l, int r) {
        if (r == 1)
            return 0;
        int invs = 0, mid = 1 + (r - 1) / 2;
        invs += merge(v, 1, mid);
        invs += merge(v, ++mid, r);
        int i = 1, j = mid;
        queue<int> temp;
10
        while (i <= mid - 1 && j <= r) {</pre>
            if (v[i] <= v[j])</pre>
13
                temp.push(v[i++]);
14
15
                temp.push(v[j++]);
16
                invs += (mid - i);
            }
        }
19
        while (i <= mid - 1)</pre>
20
           temp.push(v[i++]);
21
        while (j <= r)</pre>
22
           temp.push(v[j++]);
23
        for (i = 1; i <= r; i++) {</pre>
24
            v[i] = temp.front();
25
            temp.pop();
26
        return invs;
   }
```

2.2 Longest Increasing Subsequence (LIS)

Time: O(n)

```
int n, v[MAXN];
   int lis() {
       if (!n)
          return 0;
      vi tail(n), prev(n, -1);
       int len = 1;
       for (int i = 1; i < n; i++) {</pre>
          if (v[i] < v[tail[0]])</pre>
10
              tail[0] = i;
          12
13
              prev[i] = tail[len - 1];
              tail[len++] = i;
14
          } else {
15
              int pos = distance(tail.begin(), upper_bound(tail.begin(), tail.begin() + len - 1, v[i]));
16
              prev[i] = tail[pos - 1];
17
              tail[pos] = i;
18
19
20
21
       return len;
```

2.3 Longest Common Subsequence (LCS)

Time: O(n * m)

```
string s, t;
   int lcs() {
       int n = s.size(), m = t.size();
       int dp[n + 1][m + 1];
       for (int i = 0; i <= n; i++) {</pre>
           for (int j = 0; j \le m; j++) {
               if (!i || !j)
                   dp[i][j] = 0;
10
               else if (s[i - 1] == t[j - 1])
11
                   dp[i][j] = dp[i - 1][j - 1] + 1;
12
13
                   dp[i][j] = max(dp[i - 1][j], dp[i][j - 1]);
14
           }
15
       }
16
       return dp[n][m];
18
       /* int i = n, j = m, index = dp[n][m];
19
       string seq(n + 1, ' ');
20
       while (i > 0 \&\& j > 0) {
21
           if (s[i - 1] == t[j - 1]) {
22
               i--; j--; index--;
               seq[index] = s[i];
           } else if (dp[i - 1][j] > dp[i][j - 1])
               i--;
26
           else
27
               j--;
28
29
       return seq; */
30
31
```

2.4 Kadane

Time: O(n)

```
int v[MAXN];
// Max interval sum
int kadane() {
   int max = INT_MIN, temp = 0;
   for (int i : v) {
       temp += i;
       if (max < temp) max = temp;
       if (temp < 0) temp = 0;
   }
   return max;
}</pre>
```

^{*} Sequence building commented.

2.5 Knapsack

Time: O(n * w)

```
int value[MAXN], weight[MAXN];
   // Minimum sum of elements in which the sum of weights is less than or equal to w
   int knapsack(int n, int w) {
      int dp[n + 1][w + 1];
      for (int i = 0; i <= n; i++) {</pre>
           for (int j = 0; j \le w; j++) {
               if (!i || !j)
                   dp[i][j] = 0;
               else if (weight[i - 1] <= j)</pre>
                   dp[i][j] = max(value[i - 1] + dp[i - 1][j - weight[i - 1]], dp[i - 1][j]);
11
                   dp[i][j] = dp[i - 1][j];
12
           }
13
       }
       return dp[n][w];
16
   // With repetitions
17
   int unbKnapsack(int n, int w) {
18
       vi dp(w + 1);
19
       for (int i = 1; i <= w; i++)</pre>
20
           for (int j = 0; j < n; j++)
21
               if (weight[j] <= i)</pre>
22
                   dp[i] = max(dp[i - weight[j]] + value[j], dp[i]);
       return dp[w];
   }
```

2.6 Coin Change

Time: O(n * w)

```
int coins[MAXN];
   int minCoins(int n, int w) {
       vi dp(w + 1, INT_MAX);
       dp[0] = 0;
6
       for (int i = 1; i <= w; i++) {</pre>
           for (int j = 0; j < n; j++) {
               if (coins[j] <= i) {</pre>
                   int sub_res = dp[i - coins[j]];
11
                   if (sub_res != INT_MAX && sub_res + 1 < dp[i])</pre>
12
                       dp[i] = sub_res + 1;
               }
           }
14
15
       return dp[w];
16
17
    // Total possibilities
18
    int count(int n, int w) {
19
        if (!w)
           return 1;
        if (w < 0)
22
           return 0;
        if (n <= 0 && w >= 1)
24
           return 0;
25
       return count(n - 1, w) + count(n, w - coins[n - 1]);
26
   }
```

2.7 Binary Search Bounds

Time: O(log(n))

```
int v[MAXN], n;
   int lower(int x) {
       int 1 = 0, r = n-1, mid;
       while (1 < r) {</pre>
           mid = (1+r+1)/2;
           if (v[mid] <= x)</pre>
               l = mid;
           else
               r = mid-1;
10
11
       return v[1] <= x ? 1 : -1;</pre>
12
   }
13
14
   int upper(int x) {
15
       int 1 = 0, r = n-1, mid;
16
       while (1 < r) {
17
           mid = (1+r)/2;
18
           if (v[mid] >= x)
19
               r = mid;
20
21
           else
               l = mid+1;
22
23
       return v[r] >= x ? r : -1;
```

3 Math and Geometry

3.1 Greatest Common Divisor

Time: O(log(min(x, y)))

```
int gcd(int x, int y) {
   return y ? gcd(y, x % y) : abs(x);
}
```

3.2 Least Common Multiple

Time: O(gcd(x, y))

```
int lcm(int x, int y) {
    if (x && y) return abs(x) / gcd(x, y) * abs(y);
    return abs(x | y);
}
```

3.3 Extended Euclidean Algorithm

Time: O(log(min(x, y)))

```
int egcd(int a, int b, int &x, int &y) {
       if (a == 0) {
           x = 0, y = 1;
           return b;
       }
       int xo, yo;
       int gcd = egcd(b % a, a, xo, yo);
       x = yo - (b / a) * xo;
       y = xo;
       if (gcd < 0)
10
11
          gcd = -gcd, x = -x, y = -y;
12
       return gcd;
   }
```

3.4 N Choose R

Time: $O(p^2 * log_p n)$

```
int ncrDp(int n, int r, int p) {
       vi dp(r + 1);
       dp[0] = 1;
       for (int i = 1; i <= n; i++)</pre>
           for (int j = min(i, r); j > 0; j--)
              dp[j] = (dp[j] + dp[j - 1]) % p;
       return dp[r];
   }
   int ncr(int n, int r, int p) {
10
       if (!r)
11
12
           return 1;
       return (ncr(n / p, r / p, p) * ncrDp(n % p, r % p, p)) % p;
13
   }
14
```

3.5 Modular Multiplication

Time: O(log n)

```
int mulmod(int a, int b, int p = 1e9+7) {
    int x = 0;
    a %= p;
    while (b > 0) {
        if (b & 1)
            x = (x + a) % p;
        a = (a * 2) % p;
        b >>= 1;
    }
    return x % p;
}
```

3.6 Modular Exponentiation

Time: O(log n)

```
int expmod(int a, int b, int p = 1e9+7) {
   int x = 1;
   a %= p;
   while (b > 0) {
       if (b & 1)
            x = (x * a) % p;
       a = (a * a) % p;
       b >>= 1;
   }
   return x;
}
```

3.7 Modular Inverse

Time: O(log(min(x, y)))

```
int invmod(int a, int p = 1e9+7) {
   int x, y;
   int gcd = egcd(a, p, x, y);
   if (gcd != 1) return -1;
   return x % p + ((x < 0) ? p : 0);
}</pre>
```

3.8 Matrix Multiplication

Time: $O(n^3)$

```
struct matrix {
       int mat[MAXN][MAXN];
   };
   int n;
   matrix matMul(matrix a, matrix b) {
       matrix c;
       int k;
       for (int i = 0; i < n; i++)</pre>
10
           for (int j = 0; j < n; j++)
11
               for (c.mat[i][j] = k = 0; k < n; k++)
12
                   c.mat[i][j] += a.mat[i][k] * b.mat[k][j];
13
14
       return c;
   }
```

3.9 Sieve Of Eratosthenes

Time: O(n)

```
// Primes from 2 to n
   vi sieve(int n) {
       vb prime(n + 1, true);
       for (int p = 2; p * p <= n; p++)</pre>
           if (prime[p])
               for (int i = p * 2; i <= n; i += p)</pre>
                   prime[i] = false;
       vi v;
       for (int p = 2; p <= n; p++)</pre>
10
           if (prime[p])
11
12
               v.pb(p);
       return v;
13
   }
```

3.10 Factoring

Time: $O(\sqrt{n})$

```
vi factorize(int n) {
    vi v;
    for(int i = 2; i * i <= n; i++) {
        if (n % i) continue;
        v.pb(i);
        n /= i--;
    }
    if (n > 1) v.pb(n);
    return v;
}
```

3.11 Divisors

Time: $O(\sqrt{n})$

```
vi divisors(int n) {
    int maxP = sqrt(n) + 2;
    vi div;
    for (int i = 1; i <= maxP; i++) {
        if (n % i == 0) {
            div.pb(i);
            div.pb(n / i);
        }
    }
    return div;
}</pre>
```

3.12 Amount of Divisors

Time: $O(n^2)$

```
// Amount of divisors of each integer from 0 to n
vi amount(int n) {
    vi v(n + 1);
    for (int i = 1; i <= n; i++)
        for (int j = i; j <= n; j += i)
            v[j]++;
    return v;
}</pre>
```

3.13 Polygon Area

Time: O(n)

```
int n;
double x[MAXN], y[MAXN];
// Points need to be given a clockwise or anti-clockwise manner.

double area() {
    double total = 0;
    int j = n - 1;
    for (int i = 0; i < n; i++) {
        total += (x[j] + x[i]) * (y[j] + y[i]);
        j = i;
    }
    return (total >= 0) ? (total / 2) : (-total / 2);
}
```

3.14 Convex Hull

Time: O(nlogn)

```
struct Point {
       int x, y;
       Point () {}
       Point (int _x, int _y) {
           x = _x, y = _y;
6
   };
7
   int n;
9
   Point points[MAXN], p0;
10
11
   Point nextToTop(stack<Point> &S) {
12
13
       Point p = S.top();
       S.pop();
       Point res = S.top();
15
       S.push(p);
16
       return res;
17
   }
18
19
   int distSq(Point p1, Point p2) {
20
       return pow(p1.x - p2.x, 2) + pow(p1.y - p2.y, 2);
21
   }
22
23
24
   int orientation(Point p, Point q, Point r) {
25
       int val = (q.y - p.y) * (r.x - q.x) - (q.x - p.x) * (r.y - q.y);
       if (val == 0) return 0;
26
       return (val > 0)? 1 : 2;
27
   }
28
29
   int compare(Point a, Point b) {
30
       int o = orientation(p0, a, b);
31
       if (!o)
32
           return (distSq(p0, b) >= distSq(p0, a));
       return (o == 2);
35
36
   double convexHull() {
37
       int ymin = points[0].y, min = 0;
38
       for (int i = 1; i < n; i++) {</pre>
39
           int y = points[i].y;
40
           if ((y < ymin) || (ymin == y && points[i].x < points[min].x))</pre>
41
               ymin = points[i].y, min = i;
42
       }
43
```

```
44
        swap(points[0], points[min]);
45
       p0 = points[0];
       sort(points + 1, points + n, compare);
       int m = 1;
50
       for (int i = 1; i < n; i++) {</pre>
51
           while (i < n-1 && !orientation(p0, points[i], points[i+1]))</pre>
52
               i++;
53
           points[m++] = points[i];
54
       }
55
56
       if (m < 3) return 0;</pre>
57
58
       stack<Point> S;
59
       S.push(points[0]);
60
       S.push(points[1]);
61
       S.push(points[2]);
62
63
       for (int i = 3; i < m; i++) {</pre>
64
           while (orientation(nextToTop(S), S.top(), points[i]) != 2)
65
               S.pop();
66
           S.push(points[i]);
67
       }
       double total = 0;
70
       Point ant = S.top(), prim = S.top();
71
       S.pop();
72
       while (!S.empty()) {
73
           total += sqrt(distSq(S.top(), ant));
74
           ant = S.top();
75
76
           S.pop();
77
       total += sqrt(distSq(ant, prim));
       return total;
79
   }
80
```

4 Templates

4.1 Point

```
struct Point {
       double x, y;
       Point () {}
       Point (double _x, double _y) {
          x = _x; y = _y;
       Point operator +(const Point & other) const {
          return Point (x + other.x, y + other.y);
10
       Point operator -(const Point & other) const {
11
          return Point (x - other.x, y - other.y);
12
13
       double operator ^(const Point & other) const {
14
          return x * other.y - y * other.x;
15
       }
16
       double operator *(const Point & other) const {
17
          return x * other.x + y * other.y;
18
19
       double operator ~() const {
20
          return x * x + y * y;
21
22
23
       double distanceToSegment2 (const Point s1, const Point s2) const {
24
          Point c = *this;
          if ((s2 - s1) * (c - s1) <= 0 || (s1 - s2) * (c - s2) <= 0) {
              return min(~(s1 - c), ~(s2 - c));
          } else {
              double area = (s2 - s1) ^ (c - s1);
29
              return (area * area) / (~(s2 - s1));
30
31
       }
32
   };
33
```

4.2 Line

```
struct Line {
       double a, b, c;
       Line () {}
       Line (double _a, double _b, double _c) {
           a = _a; b = _b; c = _c;
       bool parallel (const Line & other) {
10
          return (a * other.b == other.a * b);
       }
11
       Point intersect (const Line & other) {
          if (this -> parallel(other)) return Point(-INF, -INF);
13
           else {
14
              double det = a * other.b - other.a * b;
              double x = (b * other.c - other.b * c) / det;
16
              double y = (c * other.a - other.c * a) / det;
17
              return Point(x, y);
18
          }
19
       }
       Line perpendicular (Point p) {
           return Line(-b, a, b * point.x - a * point.y);
   };
```

4.3 Circle

```
struct Circle {
       Point o;
       double r;
       Circle () {}
       Circle (Point _o, double _r) {
           o = _o; r = _r;
       }
       Circle (Point a, Point b, Point c) {
9
           Line ab = Line(a, b);
10
           Line bc = Line(b, c);
11
           Point mAB = Point((a.x + b.x) * 0.5, (a.y + b.y) * 0.5);
12
           Point mBC = Point((b.x + c.x) * 0.5, (b.y + c.y) * 0.5);
13
           ab = ab.perpendicular(mAB);
14
           bc = bc.perpendicular(mBC);
15
           if (ab.parallel(bc)) {
17
               o = Point(-INF, -INF);
              r = -1.0;
19
           } else {
20
               o = ab.intersect(bc);
21
              r = (o - a);
22
           }
23
       }
   };
```

5 Miscellaneous

5.1 ASCII Table

Dec	Bin	Нех	Char	Dec	Bin	Hex	Char	Dec	Bin	Hex	Char	Dec	Bin	Hex	Char
0	0000 0000	00	[NUL]	32	0010 0000	20	space	64	0100 0000	40	0	96	0110 0000	60	*
1	0000 0001	01	[SOH]	33	0010 0001	21	1	65	0100 0001	41	A	97	0110 0001	61	a
2	0000 0010	02	[STX]	34	0010 0010	22	H .	66	0100 0010	42	В	98	0110 0010	62	b
3	0000 0011	03	[ETX]	35	0010 0011	23	#	67	0100 0011	43	C	99	0110 0011	63	C
4	0000 0100	04	[EOT]	36	0010 0100	24	\$	68	0100 0100	44	D	100	0110 0100	64	d
5	0000 0101	05	[ENQ]	37	0010 0101	25	ક	69	0100 0101	45	E	101	0110 0101	65	е
6	0000 0110	06	[ACK]	38	0010 0110	26	&	70	0100 0110	46	F	102	0110 0110	66	f
7	0000 0111	07	[BEL]	39	0010 0111	27	1	71	0100 0111	47	G	103	0110 0111	67	g
8	0000 1000	80	[BS]	40	0010 1000	28	(72	0100 1000	48	H	104	0110 1000	68	h
9	0000 1001	09	[TAB]	41	0010 1001	29)	73	0100 1001	49	I	105	0110 1001	69	i
10	0000 1010	0 A	[LF]	42	0010 1010	2A	*	74	0100 1010	4A	J	106	0110 1010	6A	j
11	0000 1011	0В	[VT]	43	0010 1011	2B	+	75	0100 1011	4B	K	107	0110 1011	6B	k
12	0000 1100	0C	[FF]	44	0010 1100	2C	,	76	0100 1100	4C	L	108	0110 1100	6C	1
13	0000 1101	0D	[CR]	45	0010 1101	2D	-	77	0100 1101	4D	M	109	0110 1101	6D	m
14	0000 1110	0E	[SO]	46	0010 1110	2E		78	0100 1110	4E	N	110	0110 1110	6E	n
15	0000 1111	0F	[SI]	47	0010 1111	2F	/	79	0100 1111	4F	0	111	0110 1111	6F	0
16	0001 0000	10	[DLE]	48	0011 0000	30	0	80	0101 0000	50	P	112	0111 0000	70	p
17	0001 0001	11	[DC1]	49	0011 0001	31	1	81	0101 0001	51	Q	113	0111 0001	71	q
18	0001 0010	12	[DC2]	50	0011 0010	32	2	82	0101 0010	52	R	114	0111 0010	72	r
19	0001 0011	13	[DC3]	51	0011 0011	33	3	83	0101 0011	53	S	115	0111 0011	73	s
20	0001 0100	14	[DC4]	52	0011 0100	34	4	84	0101 0100	54	T	116	0111 0100	74	t
21	0001 0101	15	[NAK]	53	0011 0101	35	5	85	0101 0101	55	U	117	0111 0101	75	u
22	0001 0110	16	[SYN]	54	0011 0110	36	6	86	0101 0110	56	v	118	0111 0110	76	v
23	0001 0111	17	[ETB]	55	0011 0111	37	7	87	0101 0111	57	W	119	0111 0111	77	W
24	0001 1000	18	[CAN]	56	0011 1000	38	8	88	0101 1000	58	X	120	0111 1000	78	x
25	0001 1001	19	[EM]	57	0011 1001	39	9	89	0101 1001	59	Y	121	0111 1001	79	У
26	0001 1010	1A	[SUB]	58	0011 1010	3 A	:	90	0101 1010	5 A	\mathbf{z}	122	0111 1010	7 A	\mathbf{z}
27	0001 1011	1B	[ESC]	59	0011 1011	3в	;	91	0101 1011	5B	[123	0111 1011	7в	{
28	0001 1100	1C	[FS]	60	0011 1100	3C	<	92	0101 1100	5C	λ	124	0111 1100	7C	1
29	0001 1101	1 D	[GS]	61	0011 1101	3D	=	93	0101 1101	5D	1	125	0111 1101	7D	}
30	0001 1110	1E	[RS]	62	0011 1110	3E	>	94	0101 1110	5E	^	126	0111 1110	7E	~
31	0001 1111	1F	[US]	63	0011 1111	3 F	?	95	0101 1111	5 F	_	127	0111 1111	7 F	[DEL]