



UNIVERSIDADE FEDERAL DE ITAJUBÁ

ACM ICPC Reference

.++

Dêner José Ribeiro
Eduardo Augusto de Oliveira
Leonardo Furtado de Oliveira

Coach
Felipe Kallás Silva

Agosto / 2019

Contents

1	Graphs 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	Dynamic 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	Knapsack	10
2.6	Coin Change	10
2.7	Binary Search Bounds	11
3	Math and Geometry	12
3.1	Greatest Common Divisor	12
3.2	Least Common Multiple	12
3.3	Extended Euclidean Algorithm	12
3.4	N Choose R	12
3.5	Modular Multiplication	13
3.6	Modular Exponentiation	13
3.7	Modular Inverse	13
3.8	Matrix Multiplication	13
3.9	Sieve Of Eratosthenes	14
3.10	Factoring	14
3.11	Divisors	14
3.12	Amount of Divisors	14
3.13	Polygon Area	15
3.14	Convex Hull	15
4	Templates	17
4.1	Point	17
4.2	Line	18
4.3	Circle	18
5	Miscellaneous	19
5.1	ASCII Table	19

* The codes listed here use the following header.

```

1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 #define ii pair<int, int>
6 #define mk make_pair
7 #define vi vector<int>
8 #define vii vector<ii>
9 #define vb vector<bool>
10 #define vs vector<string>
11 #define pb push_back
12 #define ll long long
13 #define INF 1000000 // Large Number
14 #define MOD 1000000007
15 #define MAXN 1000000 // Problem Limit

```

1 Graphs and Trees

1.1 Depth-First Search (DFS)

Time: $O(V + E)$

```

1 int n;
2 vi graph[MAXN];
3 bool vis[MAXN];
4
5 void dfs(int u) {
6     vis[u] = true;
7     for (int v : graph[u])
8         if (!vis[v])
9             dfs(v);
10 }

```

1.2 Breadth-First Search (BFS)

Time: $O(V + E)$

```

1 int n;
2 vi graph[MAXN];
3
4 vi bfs(int src, int w = 1) {
5     queue<int> q;
6     vi dist(n, INT_MAX);
7     vb vis(n);
8
9     q.push(src);
10    dist[src] = 0;
11    while (!q.empty()) {
12        int u = q.front();
13        q.pop();
14        vis[u] = true;
15        for (int v : graph[u]) {
16            if (!vis[v] && dist[u] + w < dist[v]) {
17                dist[v] = dist[u] + w;
18                q.push(v);
19            }
20        }
21    }
22    return dist;
23 }

```

1.3 Dijkstra

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

```

1  int n;
2  vii graph[MAXN];
3
4  vi dijkstra(int src) {
5      priority_queue<ii, vii, greater<ii>> pq;
6      vi dist(n, INT_MAX);
7      vb vis(n, false);
8
9      pq.push(mk(0, src));
10     dist[src] = 0;
11     while (!pq.empty()) {
12         int u = pq.top().second;
13         pq.pop();
14         vis[u] = true;
15         for (int i : graph[u] {
16             int w = i.first;
17             int v = i.second;
18             if (!vis[v] && dist[v] > dist[u] + w) {
19                 dist[v] = dist[u] + w;
20                 pq.push(mk(dist[v], v));
21             }
22         }
23     }
24     return dist;
25 }
```

1.4 Bellman-Ford

Time: $O(V * E)$

```

1  struct edge {
2      int u, v, w;
3      edge() {};
4      edge(int _u, int _v, int _w) {
5          u = _u, v = _v, w = _w;
6      }
7  };
8
9  int n, m;
10 vector<edge> graph;
11
12 bool bellman(int src) {
13     vi dist(n, INT_MAX);
14
15     dist[src] = 0;
16     for (int i = 0; i < n - 1; i++)
17         for (edge e : graph)
18             if (dist[e.u] != INT_MAX)
19                 dist[e.v] = min(dist[e.u] + e.w, dist[e.v]);
20     for (edge e : graph)
21         if (dist[e.u] != INT_MAX && dist[e.u] + e.w < dist[e.v])
22             return true;
23     return false;
24 }
```

1.5 Floyd-Warshall

Time: $O(n^3)$

```

1  int n, graph[MAXN][MAXN];
2
3  void floyd() {
4      for (int k = 0; k < n; k++)
5          for (int i = 0; i < n; i++)
6              for (int j = 0; j < n; j++)
7                  graph[i][j] = min(graph[i][j], graph[i][k] + graph[k][j]);
8  }
```

1.6 Kruskal

Time: $O(n \log n)$

```

1  struct edge {
2      int u, v, w;
3      edge() {};
4      edge(int _u, int _v, int _w) {
5          u = _u, v = _v, w = _w;
6      }
7      bool operator < (const edge su) const {
8          return w < su.w;
9      }
10 };
11
12 int n, m, root[MAXN];
13 vector<edge> graph;
14
15 int findset(int u) {
16     return root[u] == u ? u : root[u] = findset(root[u]);
17 }
18
19 void initset() {
20     for (int i = 0; i < n; i++) root[i] = i;
21 }
22
23 int kruskal() {
24     initset();
25     sort(graph.begin(), graph.end());
26     vi tree[n];
27     int total = 0;
28     for (int i = 0; i < m; i++) {
29         int u = graph[i].u;
30         int v = graph[i].v;
31         int w = graph[i].w;
32
33         int fu = findset(u);
34         int fv = findset(v);
35         if (fu != fv) {
36             root[fu] = fv;
37             total += w;
38             tree[u].pb(v);
39         }
40     }
41     return total;
42 }
```

1.7 Ford-Fulkerson

Time: $O(EV^3)$

```
1 int n, flow[MAXN][MAXN], parent[MAXN];
2 vector<int> graph[MAXN];
3
4 bool bfs(int src, int snk) {
5     memset(parent, -1, sizeof parent);
6     queue<int> q;
7
8     parent[src] = -2;
9     q.push(src);
10    while (!q.empty()) {
11        int u = q.front();
12        q.pop();
13        for (int v : graph[u]) {
14            if (parent[v] == -1 && flow[u][v]) {
15                parent[v] = u;
16                if (v == snk) return true;
17                q.push(v);
18            }
19        }
20    }
21    return false;
22 }
23
24 int ford(int src, int snk) {
25     int max_flow = 0;
26     while (bfs(src, snk)) {
27         int path_flow = INT_MAX;
28         for (int v = snk; v != src; v = parent[v])
29             path_flow = min(path_flow, flow[parent[v]][v]);
30
31         for (int v = snk; v != src; v = parent[v]) {
32             flow[parent[v]][v] -= path_flow;
33             flow[v][parent[v]] += path_flow;
34         }
35         max_flow += path_flow;
36     }
37     return max_flow;
38 }
```

1.8 Tarjan

Time: $O(V + E)$

```

1  int n, disc[MAXN], low[MAXN], scc[MAXN], ct, id;
2  vi graph[MAXN];
3  stack<int> st;
4  bool onstack[MAXN];
5
6  void dfs(int u) {
7      disc[u] = low[u] = ct++;
8      st.push(u);
9      onstack[u] = true;
10
11     for (int v : graph[u]) {
12         if (disc[v] == -1) {
13             dfs(v);
14             low[u] = min(low[u], low[v]);
15         } else if (onstack[v])
16             low[u] = min(low[u], disc[v]);
17     }
18     if (low[u] == disc[u]) {
19         int v;
20         do {
21             v = st.top();
22             st.pop();
23             onstack[v] = false;
24             scc[v] = id;
25         } while (u != v);
26         id++;
27     }
28 }
29
30 void tarjan() {
31     ct = id = 0;
32     memset(disc, -1, sizeof disc);
33     for (int i = 0; i < n; i++)
34         if (disc[i] == -1)
35             dfs(i);
36 }

```

1.9 Fenwick Tree

Time: $O(\log n)$

```

1  int n, bit[MAXN], v[MAXN];
2
3  int query(int i) {
4      int sum = 0;
5      for (; i > 0; i -= i & (-i))
6          sum += bit[i];
7      return sum;
8  }
9
10 void update(int i, int value) {
11     for (; i <= n; i += i & (-i))
12         bit[i] += value;
13 }
14
15 void build() {
16     memset(bit, 0, sizeof bit);
17     for (int i = 1; i <= n; i++)
18         update(i, v[i-1]);
19 }

```

1.10 Segment Tree

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

```

1  int v[MAXN], st[4 * MAXN], /* lazy[4 * MAXN] */;
2
3  void build(int l, int r, int no = 0) {
4      if (l > r) return;
5      if (l == r) {
6          st[no] = v[l];
7          return;
8      }
9      int mid = l + (r - l) / 2;
10     build(l, mid, no * 2 + 1);
11     build(mid + 1, r, no * 2 + 2);
12     st[no] = st[no * 2 + 1] + st[no * 2 + 2];
13 }
14
15 /* void prop(int l, int r, int no) {
16     if (lazy[no] != 0) {
17         st[no] += (r - l + 1) * lazy[no];
18         if (l != r) {
19             lazy[no * 2 + 1] += lazy[no];
20             lazy[no * 2 + 2] += lazy[no];
21         }
22         lazy[no] = 0;
23     }
24 } */
25
26 int query(int l, int r, int qs, int qe, int no = 0) {
27     if (l > r || l > qe || r < qs) return 0;
28     // prop(l, r, no);
29     if (l >= qs && r <= qe) return st[no];
30     int mid = l + (r - l) / 2;
31     return query(l, mid, qs, qe, 2 * no + 1) + query(mid + 1, r, qs, qe, 2 * no + 2);
32 }
33
34 void update(int l, int r, int value, int pos, int no = 0) {
35     if (l > pos || r < pos) return;
36     st[no] += value;
37     if (l == r) return;
38     int mid = l + (r - l) / 2;
39     update(l, mid, value, pos, no * 2 + 1);
40     update(mid + 1, r, value, pos, no * 2 + 2);
41 }
42
43 /* void lazyUpdate(int l, int r, int qs, int qe, int value, int no = 0) {
44     if (l > r || l > qe || r < qs) return;
45     prop(l, r, no);
46     if (l >= qs && r <= qe) {
47         st[no] += (r - l + 1) * value;
48         if (l != r) {
49             lazy[no * 2 + 1] += value;
50             lazy[no * 2 + 2] += value;
51         }
52         return;
53     }
54     int mid = l + (r - l) / 2;
55     lazyUpdate(l, mid, qs, qe, value, no * 2 + 1);
56     lazyUpdate(mid + 1, r, qs, qe, value, no * 2 + 2);
57     st[no] = st[no * 2 + 1] + st[no * 2 + 2];
58 } */

```

* Lazy propagation commented.

2 Dynamic Programming

2.1 Merge-Sort

Time: $O(n \log n)$

```

1  int merge(int v[], int l, int r) {
2      if (r == l)
3          return 0;
4      int invs = 0, mid = l + (r - l) / 2;
5
6      invs += merge(v, l, mid);
7      invs += merge(v, ++mid, r);
8
9      int i = l, j = mid;
10     queue<int> temp;
11
12     while (i <= mid - 1 && j <= r) {
13         if (v[i] <= v[j])
14             temp.push(v[i++]);
15         else {
16             temp.push(v[j++]);
17             invs += (mid - i);
18         }
19     }
20     while (i <= mid - 1)
21         temp.push(v[i++]);
22     while (j <= r)
23         temp.push(v[j++]);
24     for (i = l; i <= r; i++) {
25         v[i] = temp.front();
26         temp.pop();
27     }
28     return invs;
29 }
```

2.2 Longest Increasing Subsequence (LIS)

Time: $O(n)$

```

1  int n, v[MAXN];
2
3  int lis() {
4      if (!n)
5          return 0;
6      vi tail(n), prev(n, -1);
7      int len = 1;
8
9      for (int i = 1; i < n; i++) {
10         if (v[i] < v[tail[0]])
11             tail[0] = i;
12         else if (v[i] > v[tail[len - 1]]){
13             prev[i] = tail[len - 1];
14             tail[len++] = i;
15         } else {
16             int pos = distance(tail.begin(), upper_bound(tail.begin(), tail.begin() + len - 1, v[i]));
17             prev[i] = tail[pos - 1];
18             tail[pos] = i;
19         }
20     }
21     return len;
22 }
```

2.3 Longest Common Subsequence (LCS)

Time: $O(n * m)$

```

1 string s, t;
2
3 int lcs() {
4     int n = s.size(), m = t.size();
5     int dp[n + 1][m + 1];
6
7     for (int i = 0; i <= n; i++) {
8         for (int j = 0; j <= m; j++) {
9             if (!i || !j)
10                dp[i][j] = 0;
11            else if (s[i - 1] == t[j - 1])
12                dp[i][j] = dp[i - 1][j - 1] + 1;
13            else
14                dp[i][j] = max(dp[i - 1][j], dp[i][j - 1]);
15        }
16    }
17    return dp[n][m];
18
19    /* int i = n, j = m, index = dp[n][m];
20    string seq(n + 1, ' ');
21    while (i > 0 && j > 0) {
22        if (s[i - 1] == t[j - 1]) {
23            i--; j--; index--;
24            seq[index] = s[i];
25        } else if (dp[i - 1][j] > dp[i][j - 1])
26            i--;
27        else
28            j--;
29    }
30    return seq; */
31 }

```

* Sequence building commented.

2.4 Kadane

Time: $O(n)$

```

1 int v[MAXN];
2 // Max interval sum
3 int kadane() {
4     int max = INT_MIN, temp = 0;
5     for (int i : v) {
6         temp += i;
7         if (max < temp) max = temp;
8         if (temp < 0) temp = 0;
9     }
10    return max;
11 }

```

2.5 Knapsack

Time: $O(n * w)$

```

1  int value[MAXN], weight[MAXN];
2  // Minimum sum of elements in which the sum of weights is less than or equal to w
3  int knapsack(int n, int w) {
4      int dp[n + 1][w + 1];
5      for (int i = 0; i <= n; i++) {
6          for (int j = 0; j <= w; j++) {
7              if (!i || !j)
8                  dp[i][j] = 0;
9              else if (weight[i - 1] <= j)
10                 dp[i][j] = max(value[i - 1] + dp[i - 1][j - weight[i - 1]], dp[i - 1][j]);
11             else
12                 dp[i][j] = dp[i - 1][j];
13         }
14     }
15     return dp[n][w];
16 }
17 // With repetitions
18 int unbKnapsack(int n, int w) {
19     vi dp(w + 1);
20     for (int i = 1; i <= w; i++)
21         for (int j = 0; j < n; j++)
22             if (weight[j] <= i)
23                 dp[i] = max(dp[i - weight[j]] + value[j], dp[i]);
24     return dp[w];
25 }

```

2.6 Coin Change

Time: $O(n * w)$

```

1  int coins[MAXN];
2
3  int minCoins(int n, int w) {
4      vi dp(w + 1, INT_MAX);
5      dp[0] = 0;
6
7      for (int i = 1; i <= w; i++) {
8          for (int j = 0; j < n; j++) {
9              if (coins[j] <= i) {
10                 int sub_res = dp[i - coins[j]];
11                 if (sub_res != INT_MAX && sub_res + 1 < dp[i])
12                     dp[i] = sub_res + 1;
13             }
14         }
15     }
16     return dp[w];
17 }
18 // Total possibilities
19 int count(int n, int w) {
20     if (!w)
21         return 1;
22     if (w < 0)
23         return 0;
24     if (n <= 0 && w >= 1)
25         return 0;
26     return count(n - 1, w) + count(n, w - coins[n - 1]);
27 }

```

2.7 Binary Search Bounds

Time: $O(\log(n))$

```
1  int v[MAXN], n;
2
3  int lower(int x) {
4      int l = 0, r = n-1, mid;
5      while (l < r) {
6          mid = (l+r+1)/2;
7          if (v[mid] <= x)
8              l = mid;
9          else
10             r = mid-1;
11     }
12     return v[l] <= x ? l : -1;
13 }
14
15 int upper(int x) {
16     int l = 0, r = n-1, mid;
17     while (l < r) {
18         mid = (l+r)/2;
19         if (v[mid] >= x)
20             r = mid;
21         else
22             l = mid+1;
23     }
24     return v[r] >= x ? r : -1;
25 }
```

3 Math and Geometry

3.1 Greatest Common Divisor

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

```

1 int gcd(int x, int y) {
2     return y ? gcd(y, x % y) : abs(x);
3 }

```

3.2 Least Common Multiple

Time: $O(\gcd(x, y))$

```

1 int lcm(int x, int y) {
2     if (x && y) return abs(x) / gcd(x, y) * abs(y);
3     return abs(x | y);
4 }

```

3.3 Extended Euclidean Algorithm

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

```

1 int egcd(int a, int b, int &x, int &y) {
2     if (a == 0) {
3         x = 0, y = 1;
4         return b;
5     }
6     int xo, yo;
7     int gcd = egcd(b % a, a, xo, yo);
8     x = yo - (b / a) * xo;
9     y = xo;
10    if (gcd < 0)
11        gcd = -gcd, x = -x, y = -y;
12    return gcd;
13 }

```

3.4 N Choose R

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

```

1 int ncrDp(int n, int r, int p) {
2     vi dp(r + 1);
3     dp[0] = 1;
4     for (int i = 1; i <= n; i++)
5         for (int j = min(i, r); j > 0; j--)
6             dp[j] = (dp[j] + dp[j - 1]) % p;
7     return dp[r];
8 }
9
10 int ncr(int n, int r, int p) {
11     if (!r)
12         return 1;
13     return (ncr(n / p, r / p, p) * ncrDp(n % p, r % p, p)) % p;
14 }

```

3.5 Modular Multiplication

Time: $O(\log n)$

```

1 int mulmod(int a, int b, int p = 1e9+7) {
2     int x = 0;
3     a %= p;
4     while (b > 0) {
5         if (b & 1)
6             x = (x + a) % p;
7         a = (a * 2) % p;
8         b >>= 1;
9     }
10    return x % p;
11 }

```

3.6 Modular Exponentiation

Time: $O(\log n)$

```

1 int expmod(int a, int b, int p = 1e9+7) {
2     int x = 1;
3     a %= p;
4     while (b > 0) {
5         if (b & 1)
6             x = (x * a) % p;
7         a = (a * a) % p;
8         b >>= 1;
9     }
10    return x;
11 }

```

3.7 Modular Inverse

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

```

1 int invmod(int a, int p = 1e9+7) {
2     int x, y;
3     int gcd = egcd(a, p, x, y);
4     if (gcd != 1) return -1;
5     return x % p + ((x < 0) ? p : 0);
6 }

```

3.8 Matrix Multiplication

Time: $O(n^3)$

```

1 struct matrix {
2     int mat[MAXN][MAXN];
3 };
4
5 int n;
6
7 matrix matMul(matrix a, matrix b) {
8     matrix c;
9     int k;
10    for (int i = 0; i < n; i++)
11        for (int j = 0; j < n; j++)
12            for (c.mat[i][j] = k = 0; k < n; k++)
13                c.mat[i][j] += a.mat[i][k] * b.mat[k][j];
14    return c;
15 }

```

3.9 Sieve Of Eratosthenes

Time: $O(n)$

```

1 // Primes from 2 to n
2 vi sieve(int n) {
3     vb prime(n + 1, true);
4
5     for (int p = 2; p * p <= n; p++)
6         if (prime[p])
7             for (int i = p * 2; i <= n; i += p)
8                 prime[i] = false;
9
10    vi v;
11    for (int p = 2; p <= n; p++)
12        if (prime[p])
13            v.pb(p);
14    return v;
15 }
```

3.10 Factoring

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

```

1 vi factorize(int n) {
2     vi v;
3     for(int i = 2; i * i <= n; i++) {
4         if (n % i) continue;
5         v.pb(i);
6         n /= i--;
7     }
8     if (n > 1) v.pb(n);
9     return v;
10 }
```

3.11 Divisors

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

```

1 vi divisors(int n) {
2     int maxP = sqrt(n) + 2;
3     vi div;
4     for (int i = 1; i <= maxP; i++) {
5         if (n % i == 0) {
6             div.pb(i);
7             div.pb(n / i);
8         }
9     }
10    return div;
11 }
```

3.12 Amount of Divisors

Time: $O(n^2)$

```

1 // Amount of divisors of each integer from 0 to n
2 vi amount(int n) {
3     vi v(n + 1);
4     for (int i = 1; i <= n; i++)
5         for (int j = i; j <= n; j += i)
6             v[j]++;
7     return v;
8 }
```

3.13 Polygon Area

Time: $O(n)$

```

1  int n;
2  double x[MAXN], y[MAXN];
3  // Points need to be given a clockwise or anti-clockwise manner.
4  double area() {
5      double total = 0;
6      int j = n - 1;
7      for (int i = 0; i < n; i++) {
8          total += (x[j] + x[i]) * (y[j] + y[i]);
9          j = i;
10     }
11     return (total >= 0) ? (total / 2) : (-total / 2);
12 }
```

3.14 Convex Hull

Time: $O(n \log n)$

```

1  struct Point {
2      int x, y;
3      Point () {}
4      Point (int _x, int _y) {
5          x = _x, y = _y;
6      }
7  };
8
9  int n;
10 Point points[MAXN], p0;
11
12 Point nextToTop(stack<Point> &S) {
13     Point p = S.top();
14     S.pop();
15     Point res = S.top();
16     S.push(p);
17     return res;
18 }
19
20 int distSq(Point p1, Point p2) {
21     return pow(p1.x - p2.x, 2) + pow(p1.y - p2.y, 2);
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);
26     if (val == 0) return 0;
27     return (val > 0)? 1 : 2;
28 }
29
30 int compare(Point a, Point b) {
31     int o = orientation(p0, a, b);
32     if (!o)
33         return (distSq(p0, b) >= distSq(p0, a));
34     return (o == 2);
35 }
36
37 double convexHull() {
38     int ymin = points[0].y, min = 0;
39     for (int i = 1; i < n; i++) {
40         int y = points[i].y;
41         if ((y < ymin) || (ymin == y && points[i].x < points[min].x))
42             ymin = points[i].y, min = i;
43     }
```



```
44
45     swap(points[0], points[min]);
46
47     p0 = points[0];
48     sort(points + 1, points + n, compare);
49
50     int m = 1;
51     for (int i = 1; i < n; i++) {
52         while (i < n-1 && !orientation(p0, points[i], points[i+1]))
53             i++;
54         points[m++] = points[i];
55     }
56
57     if (m < 3) return 0;
58
59     stack<Point> S;
60     S.push(points[0]);
61     S.push(points[1]);
62     S.push(points[2]);
63
64     for (int i = 3; i < m; i++) {
65         while (orientation(nextToTop(S), S.top(), points[i]) != 2)
66             S.pop();
67         S.push(points[i]);
68     }
69
70     double total = 0;
71     Point ant = S.top(), prim = S.top();
72     S.pop();
73     while (!S.empty()) {
74         total += sqrt(distSq(S.top(), ant));
75         ant = S.top();
76         S.pop();
77     }
78     total += sqrt(distSq(ant, prim));
79     return total;
80 }
```

4 Templates

4.1 Point

```
1 struct Point {
2     double x, y;
3     Point () {}
4     Point (double _x, double _y) {
5         x = _x; y = _y;
6     }
7
8     Point operator +(const Point & other) const {
9         return Point (x + other.x, y + other.y);
10    }
11    Point operator -(const Point & other) const {
12        return Point (x - other.x, y - other.y);
13    }
14    double operator ^(const Point & other) const {
15        return x * other.y - y * other.x;
16    }
17    double operator *(const Point & other) const {
18        return x * other.x + y * other.y;
19    }
20    double operator ~() const {
21        return x * x + y * y;
22    }
23
24    double distanceToSegment2 (const Point s1, const Point s2) const {
25        Point c = *this;
26        if ((s2 - s1) * (c - s1) <= 0 || (s1 - s2) * (c - s2) <= 0) {
27            return min(~(s1 - c), ~(s2 - c));
28        } else {
29            double area = (s2 - s1) ^ (c - s1);
30            return (area * area) / ~(s2 - s1));
31        }
32    }
33};
```

4.2 Line

```

1 struct Line {
2     double a, b, c;
3
4     Line () {}
5     Line (double _a, double _b, double _c) {
6         a = _a; b = _b; c = _c;
7     }
8
9     bool parallel (const Line & other) {
10         return (a * other.b == other.a * b);
11     }
12     Point intersect (const Line & other) {
13         if (this -> parallel(other)) return Point(-INF, -INF);
14         else {
15             double det = a * other.b - other.a * b;
16             double x = (b * other.c - other.b * c) / det;
17             double y = (c * other.a - other.c * a) / det;
18             return Point(x, y);
19         }
20     }
21     Line perpendicular (Point p) {
22         return Line(-b, a, b * point.x - a * point.y);
23     }
24 };

```

4.3 Circle

```

1 struct Circle {
2     Point o;
3     double r;
4
5     Circle () {}
6     Circle (Point _o, double _r) {
7         o = _o; r = _r;
8     }
9     Circle (Point a, Point b, Point c) {
10         Line ab = Line(a, b);
11         Line bc = Line(b, c);
12         Point mAB = Point((a.x + b.x) * 0.5, (a.y + b.y) * 0.5);
13         Point mBC = Point((b.x + c.x) * 0.5, (b.y + c.y) * 0.5);
14         ab = ab.perpendicular(mAB);
15         bc = bc.perpendicular(mBC);
16
17         if (ab.parallel(bc)) {
18             o = Point(-INF, -INF);
19             r = -1.0;
20         } else {
21             o = ab.intersect(bc);
22             r = ~(o - a);
23         }
24     }
25 };

```

5 Miscellaneous

5.1 ASCII Table

Dec	Bin	Hex	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	@	96	0110 0000	60	`
1	0000 0001	01	[SOH]	33	0010 0001	21	!	65	0100 0001	41	A	97	0110 0001	61	a
2	0000 0010	02	[STX]	34	0010 0010	22	"	66	0100 0010	42	B	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	%	69	0100 0101	45	E	101	0110 0101	65	e
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	'	71	0100 0111	47	G	103	0110 0111	67	g
8	0000 1000	08	[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	0A	[LF]	42	0010 1010	2A	*	74	0100 1010	4A	J	106	0110 1010	6A	j
11	0000 1011	0B	[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	l
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	O	111	0110 1111	6F	o
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	y
26	0001 1010	1A	[SUB]	58	0011 1010	3A	:	90	0101 1010	5A	Z	122	0111 1010	7A	z
27	0001 1011	1B	[ESC]	59	0011 1011	3B	;	91	0101 1011	5B	[123	0111 1011	7B	{
28	0001 1100	1C	[FS]	60	0011 1100	3C	<	92	0101 1100	5C	\	124	0111 1100	7C	
29	0001 1101	1D	[GS]	61	0011 1101	3D	=	93	0101 1101	5D]	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	3F	?	95	0101 1111	5F	_	127	0111 1111	7F	[DEL]