Universidade Federal de Itajubá

# ACM ICPC Reference

.++

*Dêner José Ribeiro*
*Eduardo Augusto de Oliveira*
*Leonardo Furtado de Oliveira*

Coaches
Felipe Kallás Silva

Junho / 2019

# Contents

*** The codes listed here use the following header.**

```cpp
#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>
#define pb push_back
#define ll long long
#define INF 1000000 // Large Number
#define MOD 1000000007
#define MAXN 1000000 // Problem Limit
```

# 1 Graphs and Trees

## 1.1 Depth-First Search (DFS)

Time: $O(V + E)$

```cpp
int n;
vi graph[MAXN];
bool vis[MAXN];

void dfs(int u) {
    vis[u] = true;
    for (int v : graph[x])
        if (!vis[v])
            dfs(v);
}
```

## 1.2 Breadth-First Search (BFS)

Time: $O(V + E)$

```cpp
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;
    while (!q.empty()) {
        int u = q.front();
        q.pop();
        vis[u] = true;
        for (int v : graph[u]) {
            if (!vis[v] && dist[u] + w < dist[v]) {
                dist[v] = dist[u] + w;
                q.push(v);
            }
        }
    }
    return dist;
}
```

## 1.3 Dijkstra

Time: $O(n^2 logn)$

```
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;
    while (!pq.empty()) {
        int u = pq.top().second;
        pq.pop();
        vis[u] = true;
        for (int i : graph[u] {
            int w = i.first;
            int v = i.second;
            if (!vis[v] && dist[v] > dist[u] + w) {
                dist[v] = dist[u] + w;
                pq.push(mk(dist[v], v));
            }
        }
    }
    return dist;
}
```

## 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;
    }
};

int n, m;
vector<edge> graph;

bool bellman(int src) {
    vi dist(n, INT_MAX);

    dist[src] = 0;
    for (int i = 0; i < n - 1; i++)
        for (edge e : graph)
            if (dist[e.u] != INT_MAX)
                dist[e.v] = min(dist[e.u] + e.w, dist[e.v]);
    for (edge e : graph)
        if (dist[e.u] != INT_MAX && dist[e.u] + e.w < dist[e.v])
            return true;
    return false;
}
```

## 1.5 Floyd-Warshall

Time: $O(n^3)$

```
int n, graph[MAXN][MAXN];

void floyd() {
    for (int k = 0; k < n; k++)
        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++)
                graph[i][j] = min(graph[i][j], graph[i][k] + graph[k][j]);
}
```

## 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 {
        return w < su.w;
    }
};

int n, m, root[MAXN];
vector<edge> graph;

int findset(int u) {
    return root[u] == u ? u : root[u] = findset(root[u]);
}

void initset() {
    for (int i = 0; i < n; i++) root[i] = i;
}

int kruskal() {
    initset();
    sort(graph.begin(), graph.end());
    vi tree[n];
    int total = 0;
    for (int i = 0; i < m; i++) {
        int u = graph[i].u;
        int v = graph[i].v;
        int w = graph[i].w;

        int fu = findset(u);
        int fv = findset(v);
        if (fu != fv) {
            root[fu] = fv;
            total += w;
            tree[u].pb(v);
        }
    }
    return total;
}
```

## 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;

    parent[src] = -2;
    q.push(src);
    while (!q.empty()) {
        int u = q.front();
        q.pop();
        for (int v : graph[u]) {
            if (parent[v] == -1 && flow[u][v]) {
                parent[v] = u;
                if (v == snk) return true;
                q.push(v);
            }
        }
    }
    return false;
}

int ford(int src, int snk) {
    int max_flow = 0;
    while (bfs(src, snk)) {
        int path_flow = INT_MAX;
        for (int v = snk; v != src; v = parent[v])
            path_flow = min(path_flow, flow[parent[v]][v]);

        for (int v = snk; v != src; v = parent[v]) {
            flow[parent[v]][v] -= path_flow;
            flow[v][parent[v]] += path_flow;
        }
        max_flow += path_flow;
    }
    return max_flow;
}
```

## 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;

    for (int v : graph[u]) {
        if (disc[v] == -1) {
            dfs(v);
            low[u] = min(low[u], low[v]);
        } else if (onstack[v])
            low[u] = min(low[u], disc[v]);
    }
    if (low[u] == disc[u]) {
        int v;
        do {
            v = st.top();
            st.pop();
            onstack[v] = false;
            scc[v] = id;
        } while (u != v);
        id++;
    }
}

void tarjan() {
    ct = id = 0;
    memset(disc, -1, sizeof disc);
    for (int i = 0; i < n; i++)
        if (disc[i] == -1)
            dfs(i);
}
```

## 1.9 Fenwick Tree

Time: $O(logn)$

```
int n, bit[MAXN], v[MAXN];

int query(int i) {
    int sum = 0;
    for (; i > 0; i -= i & (-i))
        sum += bit[i];
    return sum;
}

void update(int i, int value) {
    for (; i <= n; i += i & (-i))
        bit[i] += value;
}

void build() {
    memset(bit, 0, sizeof bit);
    for (int i = 1; i <= n; i++)
        update(i, v[i-1]);
}
```

## 1.10 Segment Tree

Build Time: $O(n)$ | Query Time: $O(logn)$ | Update Time: $O(logn)$

```cpp
int v[MAXN], st[4 * MAXN], /* lazy[4 * MAXN] */;

void build(int l, int r, int no = 0) {
    if (l > r) return;
    if (l == r) {
        st[no] = v[l];
        return;
    }
    int mid = l + (r - l) / 2;
    build(l, mid, no * 2 + 1);
    build(mid + 1, r, no * 2 + 2);
    st[no] = st[no * 2 + 1] + st[no * 2 + 2];
}

/* void prop(int l, int r, int no) {
    if (lazy[no] != 0) {
        st[no] += (r - l + 1) * lazy[no];
        if (l != r) {
            lazy[no * 2 + 1] += lazy[no];
            lazy[no * 2 + 2] += lazy[no];
        }
        lazy[no] = 0;
    }
} */

int query(int l, int r, int qs, int qe, int no = 0) {
    if (l > r || l > qe || r < qs) return 0;
    // prop(l, r, no);
    if (l >= qs && r <= qe) return st[no];
    int mid = l + (r - l) / 2;
    return query(l, mid, qs, qe, 2 * no + 1) + query(mid + 1, r, qs, qe, 2 * no + 2);
}

void update(int l, int r, int value, int pos, int no = 0) {
    if (l > pos || r < pos) return;
    st[no] += value;
    if (l == r) return;
    int mid = l + (r - l) / 2;
    update(l, mid, value, pos, no * 2 + 1);
    update(mid + 1, r, value, pos, no * 2 + 2);
}

/* void lazyUpdate(int l, int r, int qs, int qe, int value, int no = 0) {
    if (l > r || l > qe || r < qs) return;
    prop(l, r, no);
    if (l >= qs && r <= qe) {
        st[no] += (r - l + 1) * value;
        if (l != r) {
            lazy[no * 2 + 1] += value;
            lazy[no * 2 + 2] += value;
        }
        return;
    }
    int mid = l + (r - l) / 2;
    lazyUpdate(l, mid, qs, qe, value, no * 2 + 1);
    lazyUpdate(mid + 1, r, qs, qe, value, no * 2 + 2);
    st[no] = st[no * 2 + 1] + st[no * 2 + 2];
} */
```

* Lazy propagation commented.

# 2 Dynamic Programming

## 2.1 Merge-Sort

Time: $O(nlogn)$

```
int merge(int v[], int l, int r) {
    if (r == l)
        return 0;
    int invs = 0, mid = l + (r - l) / 2;

    invs += merge(v, l, mid);
    invs += merge(v, ++mid, r);

    int i = l, j = mid;
    queue<int> temp;

    while (i <= mid - 1 && j <= r) {
        if (v[i] <= v[j])
            temp.push(v[i++]);
        else {
            temp.push(v[j++]);
            invs += (mid - i);
        }
    }
    while (i <= mid - 1)
        temp.push(v[i++]);
    while (j <= r)
        temp.push(v[j++]);
    for (i = l; i <= r; i++) {
        v[i] = temp.front();
        temp.pop();
    }
    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++) {
        if (v[i] < v[tail[0]])
            tail[0] = i;
        else if (v[i] > v[tail[len - 1]]){
            prev[i] = tail[len - 1];
            tail[len++] = i;
        } else {
            int pos = distance(tail.begin(), upper_bound(tail.begin(), tail.begin() + len - 1, v[i]));
            prev[i] = tail[pos - 1];
            tail[pos] = i;
        }
    }
    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++) {
        for (int j = 0; j <= m; j++) {
            if (!i || !j)
                dp[i][j] = 0;
            else if (s[i - 1] == t[j - 1])
                dp[i][j] = dp[i - 1][j - 1] + 1;
            else
                dp[i][j] = max(dp[i - 1][j], dp[i][j - 1]);
        }
    }
    return dp[n][m];

    /* int i = n, j = m, index = dp[n][m];
    string seq(n + 1, ' ');
    while (i > 0 && j > 0) {
        if (s[i - 1] == t[j - 1]) {
            i--; j--; index--;
            seq[index] = s[i];
        } else if (dp[i - 1][j] > dp[i][j - 1])
            i--;
        else
            j--;
    }
    return seq; */
}
```

* Sequence building commented.

## 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;
}
```

## 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++) {
        for (int j = 0; j <= w; j++) {
            if (!i || !j)
                dp[i][j] = 0;
            else if (weight[i - 1] <= j)
                dp[i][j] = max(value[i - 1] + dp[i - 1][j - weight[i - 1]], dp[i - 1][j]);
            else
                dp[i][j] = dp[i - 1][j];
        }
    }
    return dp[n][w];
}
// With repetitions
int unbKnapsack(int n, int w) {
    vi dp(w + 1);
    for (int i = 1; i <= w; i++)
        for (int j = 0; j < n; j++)
            if (weight[j] <= i)
                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;

    for (int i = 1; i <= w; i++) {
        for (int j = 0; j < n; j++) {
            if (coins[j] <= i) {
                int sub_res = dp[i - coins[j]];
                if (sub_res != INT_MAX && sub_res + 1 < dp[i])
                    dp[i] = sub_res + 1;
            }
        }
    }
    return dp[w];
}
// Total possibilities
int count(int n, int w) {
    if (!w)
        return 1;
    if (w < 0)
        return 0;
    if (n <= 0 && w >= 1)
        return 0;
    return count(n - 1, w) + count(n, w - coins[n - 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)
        gcd = -gcd, x = -x, y = -y;
    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++)
        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) {
    if (!r)
        return 1;
    return (ncr(n / p, r / p, p) * ncrDp(n % p, r % p, p)) % p;
}
```

## 3.5 Modular Multiplication

Time: $O(logn)$

```
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: $O(logn)$

```
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);
}
```

## 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++)
        for (int j = 0; j < n; j++)
            for (c.mat[i][j] = k = 0; k < n; k++)
                c.mat[i][j] += a.mat[i][k] * b.mat[k][j];
    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++)
        if (prime[p])
            for (int i = p * 2; i <= n; i += p)
                prime[i] = false;
    vi v;
    for (int p = 2; p <= n; p++)
        if (prime[p])
            v.pb(p);
    return v;
}
```

## 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;
}
```

## 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;
}
```

## 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;
    }
};

int n;
Point points[MAXN], p0;

Point nextToTop(stack<Point> &S) {
    Point p = S.top();
    S.pop();
    Point res = S.top();
    S.push(p);
    return res;
}

int distSq(Point p1, Point p2) {
    return pow(p1.x - p2.x, 2) + pow(p1.y - p2.y, 2);
}

int orientation(Point p, Point q, Point r) {
    int val = (q.y - p.y) * (r.x - q.x) - (q.x - p.x) * (r.y - q.y);
    if (val == 0) return 0;
    return (val > 0)? 1 : 2;
}

int compare(Point a, Point b) {
    int o = orientation(p0, a, b);
    if (!o)
        return (distSq(p0, b) >= distSq(p0, a));
    return (o == 2);
}

double convexHull() {
    int ymin = points[0].y, min = 0;
    for (int i = 1; i < n; i++) {
        int y = points[i].y;
        if ((y < ymin) || (ymin == y && points[i].x < points[min].x))
            ymin = points[i].y, min = i;
    }
```

```
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

```
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);
    }
    Point operator -(const Point & other) const {
        return Point (x - other.x, y - other.y);
    }
    double operator ^(const Point & other) const {
        return x * other.y - y * other.x;
    }
    double operator *(const Point & other) const {
        return x * other.x + y * other.y;
    }
    double operator ~() const {
        return x * x + y * y;
    }

    double distanceToSegment2 (const Point s1, const Point s2) const {
        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);
            return (area * area) / (~(s2 - s1));
        }
    }
};
```

## 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) {
        return (a * other.b == other.a * b);
    }
    Point intersect (const Line & other) {
        if (this -> parallel(other)) return Point(-INF, -INF);
        else {
            double det = a * other.b - other.a * b;
            double x = (b * other.c - other.b * c) / det;
            double y = (c * other.a - other.c * a) / det;
            return Point(x, y);
        }
    }
    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) {
        Line ab = Line(a, b);
        Line bc = Line(b, c);
        Point mAB = Point((a.x + b.x) * 0.5, (a.y + b.y) * 0.5);
        Point mBC = Point((b.x + c.x) * 0.5, (b.y + c.y) * 0.5);
        ab = ab.perpendicular(mAB);
        bc = bc.perpendicular(mBC);

        if (ab.parallel(bc)) {
            o = Point(-INF, -INF);
            r = -1.0;
        } else {
            o = ab.intersect(bc);
            r = ~(o - a);
        }
    }
};
```

# 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] |