Standard Code Library

by sqybi
Shanghai Jiao Tong University

Create: 2009-09-06
Last update: 2009-09-09

Content

1 Algorithms and Datastructures

1.1 High Precision (Integer)

```
//need iostream, string, algorithm

class HP
{
public:
    static const int MAXL = 10000;
    int len, s[MAXL];
    HP() { (*this) = 0; };
    HP(int inte) { (*this) = inte; };
    HP(const char* str) { (*this) = str; };
    HP(string str) { (*this) = str; };
    friend ostream& operator << (ostream &cout, const HP &x);
    HP operator = (int inte);
    HP operator = (const char* str); HP operator = (string str);
    HP operator + (const HP &b); HP operator += (const HP &b);
    HP operator - (const HP &b); HP operator -= (const HP &b);
    HP operator * (const HP &b); HP operator *= (const HP &b);
    HP operator / (const HP &b); HP operator /= (const HP &b);
    HP operator % (const HP &b); HP operator %= (const HP &b);
    int comp(const HP &b);
    bool operator > (const HP &b); bool operator < (const HP &b);
    bool operator >= (const HP &b); bool operator <= (const HP &b);
    bool operator == (const HP &b); bool operator != (const HP &b);
};

ostream& operator << (ostream &cout, const HP &x)
{
    for (int i = x.len - 1; i >= 0; --i) cout << x.s[i];
    return cout;
}

HP HP::operator = (int inte)
{
    if (inte == 0)
    {
        len = 1;
        s[0] = 0;
    }
    else
    {
        len = 0;
        while (inte)
        {
            s[len++] = inte % 10;
            inte /= 10;
        }
    }
    return (*this);
}

HP HP::operator = (const char* str)
```

```
{
    len = strlen(str);
    for (int i = 0; i != len; ++i) s[i] = int(str[len - 1 - i] - '0');
    return (*this);
}

HP HP::operator = (string str)
{
    len = str.size();
    for (int i = 0; i != len; ++i) s[i] = int(str[len - 1 - i] - '0');
    return (*this);
}

HP HP::operator + (const HP &b)
{
    HP c;
    c.s[0] = 0;
    for (int i = 0; i < len || i < b.len; ++i)
    {
        if (i < len) c.s[i] += s[i];
        if (i < b.len) c.s[i] += b.s[i];
        c.s[i + 1] = c.s[i] / 10;
        c.s[i] %= 10;
    }
    c.len = max(len, b.len) + 1;
    while (c.len > 1 && !c.s[c.len - 1]) --c.len;
    return c;
}

HP HP::operator - (const HP &b)
{
    HP c = (*this);
    for (int i = 0; i != b.len; ++i)
    {
        c.s[i] -= b.s[i];
        if (c.s[i] < 0)
        {
            c.s[i] += 10;
            --c.s[i + 1];
        }
    }
    while (c.len > 1 && !c.s[c.len - 1]) --c.len;
    return c;
}

HP HP::operator * (const HP &b)
{
    HP c;
    for (int i = 0; i != len + b.len; ++i) c.s[i] = 0;
    for (int i = 0; i != len; ++i)
        for (int j = 0; j != b.len; ++j)
        {
            c.s[i + j] += s[i] * b.s[j];
            c.s[i + j + 1] += c.s[i + j] / 10;
            c.s[i + j] %= 10;
        }
    c.len = len + b.len;
    while (c.len > 1 && !c.s[c.len - 1]) --c.len;
    return c;
}

HP HP::operator / (const HP &b)
{
    HP c, d;
    if (b.len == 1 && b.s[0] == 0) return c;
    for (int i = len - 1; i >= 0; --i)
    {
        if (d.len != 1 || d.s[0] != 0)
        {
            for (int j = d.len; j > 0; --j)
                d.s[j] = d.s[j - 1];
            ++d.len;
        }
        d.s[0] = s[i];
        c.s[i] = 0;
        while (d >= b)
        {
            d -= b;
            ++c.s[i];
        }
    }
    c.len = len;
    while (c.len > 1 && !c.s[c.len - 1]) --c.len;
    return c;
}

HP HP::operator % (const HP &b)
{
    HP d;
    if (b.len == 1 && b.s[0] == 0) return d;
    for (int i = len - 1; i >= 0; --i)
    {
        if (d.len != 1 || d.s[0] != 0)
        {
            for (int j = d.len; j > 0; --j)
                d.s[j] = d.s[j - 1];
            ++d.len;
        }
        d.s[0] = s[i];
        while (d >= b) d -= b;
    }
    return d;
}

int HP::comp(const HP &b)
{
    if (len > b.len) return 1;
    if (len < b.len) return -1;
    int i = len - 1;
    while ((i > 0) && (s[i] == b.s[i])) --i;
    return s[i] - b.s[i];
}

HP HP::operator += (const HP &b)
```

```
{
    (*this) = (*this) + b;
    return (*this);
}

HP HP::operator -= (const HP &b)
{
    (*this) = (*this) - b;
    return (*this);
}

HP HP::operator *= (const HP &b)
{
    (*this) = (*this) * b;
    return (*this);
}

HP HP::operator /= (const HP &b)
{
    (*this) = (*this) / b;
    return (*this);
}

HP HP::operator %= (const HP &b)
{
    (*this) = (*this) % b;
    return (*this);
}

bool HP::operator > (const HP &b) { return this->comp(b) > 0; }

bool HP::operator < (const HP &b) { return this->comp(b) < 0; }

bool HP::operator >= (const HP &b) { return this->comp(b) >= 0; }

bool HP::operator <= (const HP &b) { return this->comp(b) <= 0; }

bool HP::operator == (const HP &b) { return this->comp(b) == 0; }

bool HP::operator != (const HP &b) { return this->comp(b) != 0; }
```

1.2 High Precision (Floating-point Number)

1.3 Fraction

1.4 Binary Heap

```
//need algorithm
//small on top

const int MAXN = 1048576;
int heapsize, heap[MAXN + 1];

void move_up(int x)
```

```
{
    while (x != 1)
    {
        if (heap[x / 2] > heap[x])
        {
            swap(heap[x], heap[x / 2]);
            x /= 2;
        }
        else
            break;
    }
}

void move_down(int x)
{
    int y = x * 2;
    while (y <= heapsize)
    {
        if (y < heapsize && heap[y + 1] < heap[y]) ++y;
        if (heap[y] < heap[x])
            swap(heap[x], heap[y]);
        else
            break;
        x = y; y = x * 2;
    }
}

int get_top() { return heap[1]; }

int remove_top()
{
    heap[1] = heap[heapsize--];
    move_down(1);
}

void add_heap(int x)
{
    heap[++heapsize] = x;
    move_up(heapsize);
}

void build_heap()
{
    for (int i = heapsize; i > 0; --i) move_down(i);
}
```

1.5 Winner Tree

1.6 Digital Tree

1.7 Segment Tree

1.8 Union-Find Set

1.9 Quick Sort

1.10 Merge Sort

1.11 Radix Sort

1.12 Select Kth Smallest Element

1.13 KMP

1.14 Suffix Sort

2 Graph Theory and Network Algorithms

2.1 Adjacency List

```
const int MAXNV = 10000, MAXNE = 100000;
int nv, ne, tot;
int head[MAXNV], next[MAXNE], adj[MAXNE];
int value[MAXNE];

void init()
{
    tot = 0;
    for (int i = 0; i != nv; ++i) head[i] = -1;
}

void add_edge(int x, int y, int z)
{
    next[tot] = head[x];
    adj[tot] = y;
    value[tot] = z;
    head[x] = tot;
    ++tot;
}
```

2.2 SSSP (Dijkstra + Binary Heap)

```
//need Adjacency List, alorithm

const int INF = 1000000000;
int heapsize;
int dist[MAXNV], heap[MAXNV], ref[MAXNV];
bool v[MAXNV];

void update(int x)
{
    while (x > 1 && dist[heap[x]] < dist[heap[x >> 1]])
    {
        swap(ref[heap[x]], ref[heap[x >> 1]]);
        swap(heap[x], heap[x >> 1]);
        x = x >> 1;
    }
}

int getmin()
{
    int x = 1, y = 2;
    swap(ref[heap[heapsize]], ref[heap[1]]);
    swap(heap[heapsize], heap[1]);
    --heapsize;
    while (y <= heapsize)
    {
        if (y < heapsize && dist[heap[y + 1]] < dist[heap[y]]) ++y;
        if (dist[heap[y]] < dist[heap[x]])
        {
            swap(ref[heap[x]], ref[heap[y]]);
            swap(heap[x], heap[y]);
        }
        else
            break;
        x = y; y = x << 1;
    }
    return heap[heapsize + 1];
}

int Dijkstra(int x, int y)
{
    int z, temp;
    memset(ref, 0xff, sizeof(ref));
    for (int i = 0; i != nv; ++i) dist[i] = INF;
    dist[x] = 0;
    heapsize = 1;
    heap[heapsize] = x; ref[x] = heapsize;
    memset(v, false, sizeof(v));
    while (!v[y])
    {
        for (int i = 0; i != nv; ++i) cout << dist[i] << " "; cout << endl;
        if (!heapsize) break;
        z = getmin(); v[z] = true;
        cout << z << endl;
        for (int i = 1; i <= heapsize; ++i) cout << heap[i] << " "; cout << endl;
        temp = head[z];
        while (temp != -1)
        {
            if (!v[adj[temp]] && dist[z] + value[temp] < dist[adj[temp]])
            {
                if (ref[adj[temp]] == -1)
                {
                    heap[++heapsize] = adj[temp];
                    ref[adj[temp]] = heapsize;
                }
                dist[adj[temp]] = dist[z] + value[temp];
                update(ref[adj[temp]]);
                for (int i = 1; i <= heapsize; ++i) cout << heap[i] << " "; cout <<
endl;
            }
            temp = next[temp];
        }
    }
    return dist[y];
}
```

2.3 SSSP (SPFA)

```
//need Adjacency List

const int INF = 1000000000;
int qs, ql, temp;
int q[MAXNV], dist[MAXNV], vtime[MAXNV];
bool v[MAXNV];
```

```
int SPFA(int x, int y)
{
    qs = 0;
    ql = 1;
    q[qs] = x;
    memset(v, false, sizeof(v));
    v[q[qs]] = true;
    for (int i = 0; i != nv; ++i) dist[i] = INF;
    dist[q[qs]] = 0;
    memset(vtime, 0, sizeof(vtime));
    vtime[q[qs]] = 1;
    while (ql)
    {
        temp = head[q[qs]];
        while (temp != -1)
        {
            if (dist[q[qs]] + value[temp] < dist[adj[temp]])
            {
                dist[adj[temp]] = dist[q[qs]] + value[temp];
                if (!v[adj[temp]])
                {
                    v[adj[temp]] = true;
                    ++ql;
                    q[(qs + ql - 1) % nv] = adj[temp];
                    ++vtime[adj[temp]];
                    if (vtime[adj[temp]] > nv) return -1;
                }
            }
            temp = next[temp];
        }
        v[q[qs]] = false;
        qs = (qs + 1) % nv;
        --ql;
    }
    return dist[y];
}
```

2.4 MST (Kruskal)

2.5 Minimum Directed Spanning Tree

2.6 Maximum Matching on Bipartite Graph

2.7 Maximum Cost Perfect Matching on Bipartite Graph (n^4)

2.8 Maximum Cost Perfect Matching on Bipartite Graph (n^3)

2.9 Maximum Matching on General Graph

2.10 Maximum Flow (Dinic in Matrix)

2.11 Maximum Flow (Dinic in Link)

2.12 Minimum Cost Maximim Flow (in Matrix)

2.13 Minimum Cost Maximim Flow (in Link)

2.14 Bridge (DFS)

2.15 Cutvertex (DFS)

2.16 Block (DFS)

2.17 Topological Sort (DFS)

2.18 Strongly Connected Component (DFS)

2.19 Disjoint Set

3 Number Theory

3.1 Greatest Common Diviser

3.2 China Remainder Theorem

3.3 Prime Generator

3.4 Phi generator (Eular Function)

3.5 Discrete Logarithm

3.6 Square Roots in Z_p

4 Algebraic Algorithms

4.1 Linear Equations in Z_2

4.2 Linear Equations in Z

4.3 Linear Equations in Q

4.4 Linear Equations in R

4.5 Roots of Polynomial

4.6 Roots of Cubic and Quartic

4.7 Fast Fourier Transform

4.8 FFT - Polynomial Multiplication

4.9 FFT - Convolution

4.10 FFT - Reverse Bits

4.11 Linear Programming - Primal Simplex

5 Computational Geometry

5.1 Basic Operations

5.2 Extended Operations

5.3 Convex Hull

5.4 Point Set Diameter

5.5 Closest Pair

5.6 Circles

5.7 Largest Empty Convex Polygon

5.8 Triangle Centers

5.9 Polyhedron Volume

5.10 Planar Graph Contour

5.11 Rectangles Area

5.12 Rectangles Perimeter

5.13 Smallest Enclosing Circle

5.14 Smallest Enclosing Ball