Cpp Templates

ZJUT 12

2021.3

Contents

1 Basic Algorithms

1.1 Sortings and Applications

1.1.1 Quick Sort

Time Complexity: $O(nlogn) \sim O(n^2)$

```
void qsort(int a[], int l, int r)
{
    if (l == r) return;
    int i = l - 1, j = r + 1, mid = a[l + r >> 1];
    while (i < j)
    {
        do ++i; while (a[i] < mid);
        do --j; while (a[j] > mid);
        if (i < j) swap(a[i], a[j]);
    }
    qsort(a, l, j); qsort(a, j + 1, r);
}
```

1.1.2 Quick Find

Time Complexity: $O(n)$

```
// Return the kth element in range [l,r].
int qfind(int a[], int l, int r, int k)
{
    if (l == r) return a[l];
    int i = l - 1, j = r + 1, mid = a[l + r >> 1];
    while (i < j)
    {
        do ++i; while (a[i] < mid);
        do --j; while (a[j] > mid);
        if (i < j) swap(a[i], a[j]);
    }
    int len = j - l + 1;
    if (len >= k) return qfind(a, l, j, k);
    else return qfind(a, j + 1, r, k - len);
}
```

1.1.3 Merge Sort

Time Complexity: $O(nlogn)$

```
int temp[];
void merge_sort(int a[], int l, int r)
```

```
{
    if (l == r) return;
    int mid = l + r >> 1;
    merge_sort(a, l, mid); merge_sort(a, mid + 1, r);
    int k = 0, i = l, j = mid + 1;
    while (i <= mid && j <= r)
    {
        if (a[i] <= a[j]) temp[++k] = a[i++];
        else temp[++k] = a[j++];
    }
    while (i <= mid) temp[++k] = a[i++];
    while (j <= r) temp[++k] = a[j++];
    for (int i = l, j = 1; i <= r; ++i, ++j) a[i] = temp[j];
    return;
}
```

1.1.4 Inversion Pairs

Time Complexity: $O(nlogn)$

```
// Return the number of inversion pairs in range [l, r].
int temp[];
ll invPair(int a[], int l, int r)
{
    if (l == r) return 0;
    int mid = l + r >> 1;
    ll ans = invPair(a, l, mid) + invPair(a, mid + 1, r);
    int k = 0, i = l, j = mid + 1;
    while (i <= mid && j <= r)
    {
        if (a[i] <= a[j]) temp[++k] = a[i++];
        else temp[++k] = a[j++], ans += mid - i + 1;
    }
    while (i <= mid) temp[++k] = a[i++];
    while (j <= r) temp[++k] = a[j++];
    for (int i = l, j = 1; i <= r; ++i, ++j) a[i] = temp[j];
    return ans;
}
```

1.2 Binary Search

Time Complexity: $O(logn)$

1.2.1 Integer Dichotomy

```
// Find one element in an ordered sequence.
function < int(int) > check = [&](int mid) {};
```

```cpp
int l, r, ans = -1;
while (l <= r)
{
    int mid = l + r >> 1, t = check(mid);
    if (!t)
    {
        ans = mid; break;
    }
    else
    {
        if (t > 0) r = mid - 1;
        else l = mid + 1;
    }
}
```

```cpp
// Find the maximum element satisfying some conditions.
function < bool(int) > check = [&](int mid) {};
int l, r, ans = l;
while (l <= r)
{
    int mid = l + r >> 1;
    if (check(mid)) ans = mid, l = mid + 1;
    else r = mid - 1;
}
```

```cpp
// Find the minimum element satisfying some conditions.
function < bool(int) > check = [&](int mid) {};
int l, r, ans = l;
while (l <= r)
{
    int mid = l + r >> 1;
    if (check(mid)) ans = mid, r = mid - 1;
    else l = mid + 1;
}
```

1.2.2 Float Dichotomy

```cpp
// Find the maximum value satisfying some conditions.
const double eps = 1e-8;
function < bool(double) > check = [&](double mid) {};
double l, r, ans = l;
while (r - l > eps)
{
    double mid = l + (r - l) / 2;
    if (check(mid)) ans = mid, l = mid;
    else r = mid;
}
```

```cpp
// Find the minimum value satisfying some conditions.
const double eps = 1e-8;
function < bool(double) > check = [&](double mid) {};
double l, r, ans = l;
while (r - l > eps)
{
    double mid = l + (r - l) / 2;
    if (check(mid)) ans = mid, r = mid;
    else l = mid;
}
```

1.3 Ternary Search

Time Complexity: $O(logn)$

```cpp
// Find the maximum in an unimodal function.
const double eps = 1e-8;
function < double(double) > f = [&](double x) {};
double l, r;
while (r - l > eps)
{
    double m1 = l + (r - l) / 3.0;
    double m2 = r - (r - l) / 3.0;
    if (f(m1) > f(m2)) r = m2;
    else l = m1;
}
// l is the answer.
```

```cpp
// Find the minimum in an unimodal function.
const double eps = 1e-8;
function < double(double) > f = [&](double x) {};
double l, r;
while (r - l > eps)
{
    double m1 = l + (r - l) / 3.0;
    double m2 = r - (r - l) / 3.0;
    if (f(m1) > f(m2)) l = m1;
    else r = m2;
}
// r is the answer.
```

1.4 Non-negative Big Integer Calculations

Use strings to read in big integers and save each digit inversely in an vector. Answer is also saved inversely.

1.4.1 Big Integer Addition

Time Complexity: $O(m+n)$

```cpp
std::vector < int > add(std::vector < int > &A, std::vector < int > &B)
{
    std::vector < int > temp; int t = 0;
    for (int i = 0; i < A.size() || i < B.size(); ++i)
    {
        if (i < A.size()) t += A[i];
        if (i < B.size()) t += B[i];
        temp.push_back(t % 10);
        t /= 10;
    }
    if (t) temp.push_back(t);
    return temp;
}
```

1.4.2 Big Integer Subtraction

Time Complexity: $O(m+n)$

```cpp
// Use cmp(a, b) to detect whether string a is larger than string b.
// If so, swap a, b and print a '-'.
bool cmp(std::string &a, std::string &b)
{
    if (a.length() != b.length()) return a.length() < b.length();
    else
        for (int i = 0; i < a.length(); ++i)
            if (a[i] != b[i]) return a[i] < b[i];
    return 0;
}
std::vector < int > sub(std::vector < int > &A, std::vector < int > &B)
{
    std::vector < int > temp; int t = 0;
    for (int i = 0; i < A.size(); ++i)
    {
        t = A[i] - t;
        if (i < B.size()) t -= B[i];
        temp.push_back((t + 10) % 10);
        if (t < 0) t = 1; else t = 0;
    }
    // Remove leading zeros.
    while (temp.size() > 1 && temp.back() == 0) temp.pop_back();
```

```
    return temp;
}
```

1.4.3 Big Integer Multiplies Integer

Time Complexity: $O(n)$

```
std::vector < int > mul(std::vector < int > &A, int b)
{
    std::vector < int > temp; int t = 0;
    for (int i = 0; i < A.size() || t; ++i)
    {
        if (i < A.size()) t += A[i] * b;
        temp.push_back(t % 10);
        t /= 10;
    }
    // Remove leading zeros.
    while (temp.size() > 1 && temp.back() == 0) temp.pop_back();
    return temp;
}
```

1.4.4 Big Integer Divides Integer

Time Complexity: $O(n)$

```
// r means remainders.
std::vector < int > div(std::vector < int > &A, int b, int &r)
{
    std::vector < int > temp;
    for (int i = A.size() - 1; i >= 0; --i)
    {
        r = r * 10 + A[i];
        temp.push_back(r / b);
        r %= b;
    }
    reverse(temp.begin(), temp.end());
    while (temp.size() > 1 && temp.back() == 0) temp.pop_back();
    return temp;
}
```

2 Dynamic Programming

3 Data Structure

4 Graph Theory

5 Mathematics

6 Computational Geometry

7 Strings

8 Greedy

9 STL