ICPC TEMPLATE

BY

albertxwz

School of Computer Science & Engineering,
South China University of Technology

Published in

Nov 2020

This template is a supplementary version.

Contents

1 Standard Solution Template

1.1 support bits/stdc++.h

```
#include <bits/stdc++.h>
#define lc (o<<1)
#define rc ((o<<1)|1)
using namespace std;
#define DebugP(x) cout << "Line" << __LINE__
    << " " << #x << "=" << x << endl

const int maxn = 5e5 + 5;
const int modu = 998244353; // 1e9 + 7
const double eps = 1e-5;
const int dx[4] = {1, 0, -1, 0};
const int dy[4] = {0, 1, 0, -1};

typedef long long LL;

int main() {
    // freopen("my.txt", "w", stdout);
    return 0;
}
```

1.2 unsupport bits/stdc++.h

```
#include <cstdio>
#include <cstring>
#include <iostream>
#include <algorithm>
#include <cmath>
#include <iterator>
#include <set>
#include <map>
#include <queue>
#include <cstdlib>
#include <string>

using namespace std;

int main() {
    // freopen("my.txt", "w", stdout);
    return 0;
}
```

1.3 Python Template

```
for T in range(0, int( input())): #T组数据
    N= int( input())
    n,m= map( int, input().split())
    s= input()
    s=[ int(x) for x in
        input().split()] #一行输入的数组
    a[1:]=[ int(x) for x in
        input().split()] #从下标1开始读入一行
    for i in range(0, len(s)):
        a,b= map( int, input().split())

while True: #未知多组数据
    try:
        #n,m=map(int,input().split())
        #print(n+m,end="\n")
    except EOFError: #捕获到异常
        break
```

2 Graph

2.1 Network Flow

2.1.1 Dinic

```
struct Edge {
    int from, to, cap, flow;
    Edge(int from, int to=0, int cap=0, int
        flow=0):
        from(from), to(to), cap(cap), flow(
            flow) {}
};

vector<Edge> edges;
vector<int> g[maxn];
int d[maxn], cur[maxn];

void addedge(int u, int v, int cap) {
    edges.push_back(Edge(v, cap));
    g[u].push_back(edges.size()-1);
    edges.push_back(Edge(u, 0));
    g[v].push_back(edges.size()-1);
}

/*
 * 时间复杂度: O(n^2*m)
 * 对于特殊的图，所有容量为1的: O(min(n^0.67,
   m^0.5)*m)
 *             二分图最大匹配: O(n^0.5*m)
 *
 */

bool BFS(int s, int t) {
    memset(d, -1, sizeof(d));
    queue<int> Q;
    Q.push(s);
    d[s] = 0;
    while (!Q.empty()) {
        int x = Q.front(); Q.pop();
        for (int i = 0; i < g[x].size(); ++i)
            {
            Edge &e = edges[g[x][i]];
            if (d[e.to] == -1 && e.cap > e.
                flow) {
                d[e.to] = d[x] + 1;
                Q.push(e.to);
            }
        }
    }
    return d[t] > -1;
}

int DFS(int x, int a, int t) {
    if (x == t || a == 0) return a;
    int flow = 0, f;
    for (int &i = cur[x]; i < g[x].size(); ++
        i) {
        Edge &e = edges[g[x][i]];
        if (d[x] + 1 == d[e.to] && (f = DFS(e
            .to, min(a, e.cap-e.flow), t)) >
            0) {
            flow += f;
            e.flow += f;
            edges[g[x][i]^1].flow -= f;
            a -= f;
            if (a == 0) break;
        }
    }
    return flow;
}

int MaxFlow(int s, int t) {
    int res = 0;
    while (BFS(s, t)) {
        for (int i = 0; i <= n; ++i) cur[i] =
             0;
        res += DFS(s, inf, t);
    }
    return res;
}
```

2.1.2 Edmonds-Karp

```
queue<int> Q;
int imp[maxn], p[maxn];

// 不断寻找增广路增广

int MaxFlow(int s, int t) {
    int res = 0;
    for (;;) {
        memset(imp, 0, sizeof(imp));
        while (!Q.empty()) Q.pop();
        Q.push(s);
```

```
        imp[s] = inf;
        while (!Q.empty()) {
            int x = Q.front(); Q.pop();
            for (int i = 0; i < g[x].size();
                ++i) {
                Edge &e = edges[g[x][i]];
                if (!imp[e.to] && e.cap > e.
                    flow) {
                    p[e.to] = g[x][i];
                    imp[e.to] = min(imp[x], e.
                        cap-e.flow);
                    Q.push(e.to);
                }
            }
            if (imp[t]) break;
        }
        if (!imp[t]) break;
        for (int u = t; u != s; u = edges[p[u
            ]].from) {
            edges[p[u]].flow += imp[t];
            edges[p[u]^1].flow -= imp[t];
        }
        res += imp[t];
    }
    return res;
}
```

2.2 DSU on Tree

3 Mathematics

3.1 Number Theory

3.1.1 Linear Inverse Modulo

```
const int maxn = 2e5 + 5;
const int modu = 1e9 + 7;
long long inv[maxn]; // k在模modu的意义下的逆元是inv[k]
inv[1] = 1;
for (int i = 2; i < maxn; ++i)
    inv[i] = (modu - (modu/i))*inv[modu%i]%modu;
```

3.1.2 Möbius Inversion

3.1.3 Dujiao Sieve

3.1.4 Min_25 Sieve

3.1.5 Lucas' Theorem

```
long long Lucas(long long n, long long m, long long p) {
    if (m == 0) return 1;
    return (C(n % p, m % p, p) * Lucas(n / p, m / p, p)) % p;
}
```

3.2 Karatsuba Multiply

```
int *karatsuba_polymul(int n, int *a, int *b) {
  if (n <= 32) {
    // 规模较小时直接计算，避免继续递归带来的效率损失
    int *r = new int[n * 2 + 1]();
    for (int i = 0; i <= n; ++i)
      for (int j = 0; j <= n; ++j) r[i + j] += a[i] * b[j];
    return r;
  }

  int m = n / 2 + 1;
  int *r = new int[m * 4 + 1]();
  int *z0, *z1, *z2;

  z0 = karatsuba_polymul(m - 1, a, b);
  z2 = karatsuba_polymul(n - m, a + m, b + m);

  // 计算 z1
  // 临时更改，计算完毕后恢复
  for (int i = 0; i + m <= n; ++i) a[i] += a[i + m];
  for (int i = 0; i + m <= n; ++i) b[i] += b[i + m];
  z1 = karatsuba_polymul(m - 1, a, b);
  for (int i = 0; i + m <= n; ++i) a[i] -= a[i + m];
  for (int i = 0; i + m <= n; ++i) b[i] -= b[i + m];
  for (int i = 0; i <= (m - 1) * 2; ++i) z1[i] -= z0[i];
  for (int i = 0; i <= (n - m) * 2; ++i) z1[i] -= z2[i];

  // 由 z0、z1、z2 组合获得结果
  for (int i = 0; i <= (m - 1) * 2; ++i) r[i] += z0[i];
  for (int i = 0; i <= (m - 1) * 2; ++i) r[i + m] += z1[i];
  for (int i = 0; i <= (n - m) * 2; ++i) r[i + m * 2] += z2[i];

  delete[] z0;
  delete[] z1;
  delete[] z2;
  return r;
}
// 计算a*b=c, 时间复杂度是O(n^1.585)
void karatsuba_mul(int a[], int b[], int c[]) {
  int *r = karatsuba_polymul(LEN - 1, a, b);
  memcpy(c, r, sizeof(int) * LEN);
  for (int i = 0; i < LEN - 1; ++i)
    if (c[i] >= 10) {
      c[i + 1] += c[i] / 10;
      c[i] %= 10;
    }
  delete[] r;
}
```

3.3 Fast Fourier Transform

```
// f是系数数组，处理完后，f表示:
// rev=1,是点表示法
// rev=−1,除N后是系数
// N=2^n
typedef complex<double> Comp; // 先导入头文件
    complex
void DFT(Comp *f, int N, int rev) {
    if (N == 1) return;
    for (int i = 0; i < N; ++i) tmp[i] = f[i
        ];
    for (int i = 0; i < N; ++i)
        if (i%2) f[i/2+N/2] = tmp[i];
        else f[i/2] = tmp[i];
    Comp *g = f, *h = f + N/2;
    DFT(g, N/2, rev); DFT(h, N/2, rev);
    // c[N]=cos(2*pi/N), s[N]=sin(2*pi/N)
    Comp w(c[N], s[N]*rev), cur(1, 0);
    for (int k = 0; k < N/2; ++k) {
        tmp[k] = g[k] + cur*h[k];
        tmp[k+N/2] = g[k] - cur*h[k];
        cur *= w;
    }
    for (int i = 0; i < N; ++i) f[i] = tmp[i
        ];
}
```

4 Data Structure

4.1 Treap

4.2 Splay Tree

4.3 Two-dimensional Segment Tree

5 String

5.1 KMP

```
int f[maxn];

void getfail(char *P, int *f) {
    f[0] = 0;
    f[1] = 0;
    int m = strlen(P);
    for (int i = 1; i < m; ++i) {
        int j = f[i];
        while (j && P[i] != P[j]) j = f[j];
        if (P[i] == P[j]) f[i+1] = j+1;
        else f[i+1] = 0;
    }
}

int find(char *T, char *P, int *f) {
    int res = 0;
    int n = strlen(T), m = strlen(P);
    getfail(P, f);
    for (int i = 0, j = 0; i < n; ++i) {
        while (j && P[j] != T[i]) j = f[j];
        if (P[j] == T[i]) j++;
        if (j == m) {  // 出现一次
            res++;
            j = f[m];
        }
    }
    return res;
}
```

A Theorem

A.1 Lucas' Theorem

对于质数 p, 有

$$\binom{n}{m}\ mod\ p = \binom{\lfloor n/p \rfloor}{\lfloor m/p \rfloor} \cdot \binom{n\ mod\ p}{m\ mod\ p}\ mod\ p$$

A.2 Betty's Theorem

如果两个无理数 a, b 满足：

$$\frac{1}{a} + \frac{1}{b} = 1$$

那么对于两个集合 A, B:

$$A = \{[na]\}, B = \{[nb]\}$$

有下面两个结论：

$$A \bigcap B = \emptyset, A \bigcup B = \mathbb{N}^{+}$$

B C++ STL: set

set 与 undered_set 的区别在于有无在内部存储时有无顺序。

B.1 Basic Method

begin() 返回 set 容器的第一个元素

end() 返回 set 容器的最后一个元素

clear() 删除 set 容器中的所有的元素

empty() 判断 set 容器是否为空

max_size() 返回 set 容器可能包含的元素最大个数

size() 返回当前 set 容器中的元素个数

rbegin() 返回的值和 end() 相同

rend() 返回的值和 rbegin() 相同

count() 用来查找 set 中某个某个键值出现的次数。(在 set 中只有 0 或 1 次)

equal_range() 返回一对定位器，分别表示第一个大于或等于给定关键值的元素和第一个大于给定关键值的元素，这个返回值是一个 pair 类型，如果这一对定位器中哪个返回失败，就会等于 end() 的值

erase(iterator) 删除定位器 iterator 指向的值

erase(first,second) 删除定位器 first 和 second 之间的值

erase(key_value) 删除键值 key_value 的值

insert(key_value) 将 key_value 插入到 set 中，返回值是 pair<set<int>::iterator,bool>，bool 标志着插入是否成功，而 iterator 代表插入的位置，若 key_value 已经在 set 中，则 iterator 表示的 key_value 在 set 中的位置

lower_bound(key_value) 返回第一个大于等于 key_value 的定位器

upper_bound(key_value) 返回最后一个大于等于 key_value 的定位器

B.2 Advanced Method

注： 必须导入 algorithm 头文件

set_intersection(first1,last1,first2,last2,d_first,comp)

first1, last1 - 要检验的第一元素范围

first2, last2 - 要检验的第二元素范围

d_first - 输出范围的起始

comp - 比较函数对象（即满足比较 (Compare) 概念的对象），若第一参数小于（即先序于）第二参数则返回 true

Example:

```
std::set_intersection(v1.begin(),v1.
    end(),v2.begin(),v2.end(),std::
    back_inserter(v_intersection));
    // v_intersection 就是交集,
        back_insecter ()用于 vector
```

set_union(first1,last1,first2,last2,d_first,comp) 同 intersection