

ACM-ICPC Template



GuessEver

Last Modify at February 13, 2015

Contents

1	Dynamic Programming	3
1.1	LCS - Longest Common Subsequence	3
1.2	LIS - Longest Increasing Subsequence	3
1.3	Maximum Continuous Subsequence Sum	4
1.4	数位 dp	4
2	Math	5
2.1	GCD && LCM	5
2.1.1	GCD - Greatest Common Divisor	5
2.1.2	LCM - Least Common Multiple	5
2.1.3	E_GCD - Extended Greatest Common Divisor	5
2.2	Prime	5
2.2.1	Make Prime List	5
2.2.2	Prime Factor	6
2.3	Fast Power	6
3	Datastructure	7
3.1	Leftist Tree	7
3.2	Partition Tree	7
3.3	Treap	8
3.3.1	@ Array	8
3.3.2	@ Pointer	10
3.4	Size Balanced Tree	12
4	Graph	14
4.1	Shortest path	14
4.1.1	Dijkstra	14
4.1.2	Spfa	14
4.1.3	Floyd	15
4.2	Minimum Spanning Tree	15
4.2.1	Prim	15
4.2.2	Kruskal	16
4.3	Tarjan - Strong Union	16
4.4	LCA	17
4.4.1	Tarjan	17
4.4.2	Doubling Algorithm	19
4.5	Bipartite Graph	19
4.5.1	Maximal Matching - The Hungarian algorithm	19
4.5.2	Optimal Matching - KM	19
4.6	Network Flow	20
4.6.1	Maximum Flow - isap	20
4.6.2	Minimum Cost Maximum Flow - spfa	20
5	Geometry	21
5.1	Convex Hull	21
5.2	All	22
6	String	28
6.1	Manacher	28
6.2	KMP	29
6.3	Suffix Array	30

7	Tools	32
7.1	BigInteger - C++	32
7.2	C char*	36
7.3	C++ std::string	36
7.4	Batch test	36
7.4.1	@Linux	36
7.4.2	@Windows	37
7.5	Vimrc Config For Linux	37

1 Dynamic Programming

1.1 LCS - Longest Common Subsequence

```

1 int LCS() // O(N*N)
2 { // 字符串纠正到以 1 为下标
3     int f[N][N];
4     int res = 0;
5     for(int i = 1; i < lena; i++)
6         for(int j = 1; j < lenb; j++)
7         {
8             if(a[i] == a[j]) f[i][j] = f[i-1][j-1] + 1;
9             else f[i][j] = max(f[i-1][j], f[i][j-1]);
10            res = max(res, f[i][j]);
11        }
12    return res;
13 }
14
15 int LCS() // O(NlogN)
16 { // 把 LCM 转化为 LIS 来做
17     // 1 2 5 9 3 --> 1 2 3 4 5
18     // 1 5 3 9 2 --> 1 3 5 4 2 --> 对这个序列跑 LIS()
19     //——change——
20     // 这里就要针对数据自己想尽办法转化了
21     for(int i = 1; i <= n; i++) h[a[i]] = i;
22     for(int i = 1; i <= n; i++) b[i] = h[b[i]];
23     //——end——
24     return LIS();
25 }

```

1.2 LIS - Longest Increasing Subsequence

```

1 int f[N];
2 int LIS() // O(N*N)
3 {
4     for(int i = 1; i <= n; i++)
5         for(int j = i-1; j >= 1; j--)
6             if(a[i] > a[j]) f[i] = max(f[i], f[j] + 1);
7     int res = 0;
8     for(int i = 1; i <= n; i++) res = max(res, f[i]);
9     return res;
10 }
11
12 int c[N], len = 0;
13 int LIS() // (NlogN)
14 {
15     for(int i = 1; i <= n; i++)
16     {
17         //——find——
18         int l = 1, r = len, mid;
19         while(l <= r)
20         {

```

```

21         mid = (l + r) / 2;
22         if(a[i] > c[mid]) l = mid + 1;
23         else r = mid - 1;
24     }
25     //-----end-----
26     c[l] = a[i];
27     len = max(len, l);
28 }
29 return len;
30 }

```

1.3 Maximum Continuous Subsequence Sum

```

1 int MaxSubSum()
2 {
3     int f[N], res;
4     for(int i = 1; i <= n; i++)
5     {
6         f[i] = max(a[i], f[i-1] + a[i]);
7         res = max(res, f[i]);
8     }
9     return res;
10 }
11
12 int MaxSubSum()
13 {
14     int res = 0, now = 0;
15     for(int i = 1; i <= n; i++)
16     {
17         now += a[i];
18         res = max(res, now);
19         if(now < 0) now = 0;
20     }
21     return res;
22 }

```

1.4 数位 dp

```

1 int predoing(LL a, int *num)
2 {
3     int le = 0;
4     while(a)
5     {
6         num[++le] = a % 10;
7         a /= 10;
8     }
9     return le;
10 }
11 int calc(int pos, int d, int u, int last)
12 {
13     if(pos == 0) return 1;

```

```

14     int &res = f[pos][d][u][last];
15     if(res != -1) return res;
16     res = 0;
17     int st = d ? L[pos] : 0;
18     int ed = u ? R[pos] : 9;
19     for(int i = st; i <= ed; i++)
20         if(合法) res += calc(pos - 1, d && i == L[pos], u && i == R[pos], i);
21     return res;
22 }

```

2 Math

2.1 GCD && LCM

2.1.1 GCD - Greatest Common Divisor

```

1 | int gcd(int a, int b) { return b ? gcd(b, a % b) : a; }

```

2.1.2 LCM - Least Common Multiple

```

1 | inline int lcm(int a, int b) { return a / gcd(a, b) * b; }

```

2.1.3 E_GCD - Extended Greatest Common Divisor

```

1 | ax + by = 1
2 | bx1 + (a%b)y1 = 1    ==>    bx + (a-a/b*b)y = 1
3 | ==> ay1 + b(x1-a/b*y1) = 1
4 | 对应  ax  + by          = 1
5 |
6 | int egcd(int a, int b, int &x, int &y)
7 | {
8 |     if(b == 0)
9 |     {
10 |         x = 1; y = 0;
11 |         return a;
12 |     }
13 |     int x1, y1;
14 |     int e = egcd(b, a%b, x1, y1);
15 |     x = y1;
16 |     y = x1 - a / b * y1;
17 |     return e;
18 | }

```

2.2 Prime

2.2.1 Make Prime List

```

1 void make_prime_list(int maxp) //  $O(2*N)$ 
2 {
3     for(int i = 2; i <= maxp; i++)
4     {
5         if(!h[i]) pri[l++] = i;
6         for(int j = 0; j < l && pri[j] <= maxp / i; j++)
7         {
8             h[i * pri[j]] = true;
9             if(i % pri[j] == 0) break;
10        }
11    }
12 }

```

2.2.2 Prime Factor

```

1 void factor()
2 {
3     make_prime_list();
4     for(int j = 0; j < Cnt && pri[j]*pri[j] <= n; j++)
5     {
6         if(n % pri[j] == 0)
7         {
8             printf("%d_", pri[j]);
9             while(n % pri[j] == 0) n /= pri[j];
10        }
11    }
12    if(n!=1) printf("%d",n);
13 }

```

2.3 Fast Power

```

1 //x^y % mod
2 int mul(int x, LL y, int mod) // 递归
3 {
4     if(y == 1) return x;
5     if(y & 1) return (mul((x * (LL)x) % mod, y / 2, mod) * (LL)x)%mod;
6     else return mul((x * (LL)x) % mod, y / 2, mod) % mod;
7 }
8 int mul(int x, int y, int mod) // 非递归
9 {
10    int s = 1;
11    int ss = x;
12    while(y)
13    {
14        if(y & 1) s = s * ss;
15        y /= 2;
16        ss *= ss;
17    }
18    return s;
19 }

```

3 Datastructure

3.1 Leftist Tree

```

1 //很多时候需要配合并查集一起使用
2 int getroot(int x){return f[x]==x ? x : f[x]=getroot(f[x]);}
3
4 //把x和y合并在一起，其实就是把y插入x
5 int merge(int x,int y)//返回合并后子树的根
6 {
7     if(!x || !y) return x|y;
8     if(A[x] < A[y]) swap(x,y);//大根堆，如果y比x大，与其让y插入x，
        不如让x插入y
9     R[x]=merge(R[x],y);//始终往右子树合并
10    f[R[x]] = x;//更新并查集
11    if(D[R[x]] > D[L[x]]) swap(L[x],R[x]);//保持左偏树性质
12    D[x] = D[R[x]] + 1;
13    若还有其他维护信息也需要更新；
14    return x;//返回根
15 }
16
17 int del(int x)
18 {
19     int t = merge(L[x],R[x]);
20     f[L[x]] = L[x]; f[R[x]] = R[x];//更新并查集
21     L[x] = R[x] = D[x] = 0;
22     return t;
23 }

```

3.2 Partition Tree

```

1 struct Parti{int val, left;} val[30][N];
2 void build_tree(int d, int l, int r)
3 {
4     if(l == r) return;
5     int m = (l + r) >> 1, same = m - l + 1;
6     int lcnt = l, rcnt = m + 1;
7     for(int i = l; i <= r; i++)
8         if(val[d][i].val < sorted[m]) same--;
9     for(int i = l; i <= r; i++)
10    {
11        int flag = 0;
12        if((val[d][i].val < sorted[m]) || (val[d][i].val == sorted[m] &&
            same))
13        {
14            flag = 1;
15            val[d + 1][lcnt++] = val[d][i];
16            if(val[d][i].val == sorted[m]) same--;
17        }
18        else val[d][rcnt++] = val[d][i];
19        val[d][i].left = val[d][i - 1].left + flag;
20    }

```



```

21     build_tree(d + 1, l, m);
22     build_tree(d + 1, m + 1, r);
23 }
24 int query(int d, int l, int r, int x, int y, int k)
25 {
26     if(l == r) return val[d][l].val;
27     int m = (l + r) >> 1;
28     int lx = val[d][x - 1].left - val[d][l - 1].left; //[l,x-1] to left
29     int ly = val[d][y].left - val[d][x - 1].left; //[x,y] to left
30     int rx = (x - 1 - l + 1) - lx; //[l,x-1] to right
31     int ry = (y - x + 1) - ly; //[x,y] to right
32     if(ly >= k) return query(d+1, l, m, l-1+lx+1, l-1+lx+ly, k);
33     else return query(d+1, m+1, r, m+1-1+rx+1, m+1-1+rx+ry, k-ly);
34 }

```

3.3 Treap

3.3.1 @ Array

```

1  #include <cstdio>
2  #include <cstdlib>
3  #include <ctime>
4  // BZOJ 1503 郁闷的出纳员
5  int m, Limit;
6  struct Treap{
7      int fix, key, size;
8      Treap *left, *right;
9  }*root;
10 int leave;
11
12 void rotate_left(Treap *&p)
13 {
14     Treap *tmp = p -> right;
15     p -> right = tmp -> left;
16     int zsize = tmp -> left ? tmp -> left -> size : 0;
17     p -> size = p -> size - tmp -> size + zsize;
18     tmp -> left = p;
19     tmp -> size = tmp -> size - zsize + p -> size;
20     p = tmp;
21 }
22 void rotate_right(Treap *&p)
23 {
24     Treap *tmp = p -> left;
25     p -> left = tmp -> right;
26     int zsize = tmp -> right ? tmp -> right -> size : 0;
27     p -> size = p -> size - tmp -> size + zsize;
28     tmp -> right = p;
29     tmp -> size = tmp -> size - zsize + p -> size;
30     p = tmp;
31 }
32
33 void insert(Treap *&p, int x)

```

```

34 {
35     if(!p)
36     {
37         p = new Treap;
38         p -> fix = rand();
39         p -> key = x;
40         p -> size = 1;
41         p -> left = 0;
42         p -> right = 0;
43         return;
44     }
45     if(x < p -> key)
46     {
47         insert(p -> left, x);
48         p -> size++;
49         if(p -> left -> fix > p -> fix) rotate_right(p);
50     }
51     else {
52         insert(p -> right, x);
53         p -> size++;
54         if(p -> right -> fix > p -> fix) rotate_left(p);
55     }
56 }
57
58 void remove(Treap *&p, int L)
59 {
60     if(!p) return;
61     if(p -> key < L)
62     {
63         leave += (p -> left ? p -> left -> size : 0) + 1;
64         p = p -> right;
65         remove(p, L);
66     }
67     else {
68         remove(p -> left, L);
69         int lsize = p -> left ? p -> left -> size : 0;
70         int rsize = p -> right ? p -> right -> size : 0;
71         p -> size = lsize + rsize + 1;
72     }
73 }
74
75 int kth(Treap *&p, int k)
76 {
77     int Lsize = p -> left ? p -> left -> size : 0;
78     if(k <= Lsize) return kth(p -> left, k);
79     else if(k == Lsize + 1) return p -> key;
80     else return kth(p -> right, k - Lsize - 1);
81 }
82
83 int main()
84 {
85     srand(time(0));
86     scanf("%d%d", &m, &Limit);

```

```

87     int delta = 0;
88     while(m--)
89     {
90         char op; int x;
91         scanf("%c%d", &op, &x);
92         if(op == 'I')
93         {
94             if(x < Limit) continue;
95             insert(root, x - delta);
96         }
97         else if(op == 'A') delta += x;
98         else if(op == 'S')
99         {
100             delta -= x;
101             remove(root, Limit - delta);
102         }
103         else {
104             int tot = root ? root -> size : 0;
105             x = tot - x + 1;
106             if(x <= 0) puts("-1");
107             else printf("%d\n", kth(root, x) + delta);
108         }
109     }
110     printf("%d\n", leave);
111     return 0;
112 }

```

3.3.2 @ Pointer

```

1  #include <stdio>
2  #include <stdlib>
3  #include <ctime>
4
5  int m, Limit;
6  struct Treap{
7      int fix, key, size;
8      Treap *left, *right;
9  }*root, *null;
10 int leave;
11
12 void rotate_left(Treap *&p)
13 {
14     Treap *tmp = p -> right;
15     p -> right = tmp -> left;
16     int zsize = tmp -> left -> size;
17     p -> size = p -> size - tmp -> size + zsize;
18     tmp -> left = p;
19     tmp -> size = tmp -> size - zsize + p -> size;
20     p = tmp;
21 }
22 void rotate_right(Treap *&p)
23 {

```

```

24     Treap *tmp = p -> left;
25     p -> left = tmp -> right;
26     int zsize = tmp -> right -> size;
27     p -> size = p -> size - tmp -> size + zsize;
28     tmp -> right = p;
29     tmp -> size = tmp -> size - zsize + p -> size;
30     p = tmp;
31 }
32
33 void insert(Treap *&p, int x)
34 {
35     if(p == null)
36     {
37         p = new Treap;
38         p -> fix = rand();
39         p -> key = x;
40         p -> size = 1;
41         p -> left = null;
42         p -> right = null;
43         return;
44     }
45     if(x < p -> key)
46     {
47         insert(p -> left, x);
48         p -> size++;
49         if(p -> left -> fix > p -> fix) rotate_right(p);
50     }
51     else {
52         insert(p -> right, x);
53         p -> size++;
54         if(p -> right -> fix > p -> fix) rotate_left(p);
55     }
56 }
57
58 void remove(Treap *&p, int L)
59 {
60     if(p == null) return;
61     if(p -> key < L)
62     {
63         leave += p -> left -> size + 1;
64         p = p -> right;
65         remove(p, L);
66     }
67     else {
68         remove(p -> left, L);
69         p -> size = p -> left -> size + p -> right -> size + 1;
70     }
71 }
72
73 int kth(Treap *&p, int k)
74 {
75     int Lsize = p -> left -> size;
76     if(k <= Lsize) return kth(p -> left, k);

```

```

77     else if(k == Lsize + 1) return p -> key;
78     else return kth(p -> right, k - Lsize - 1);
79 }
80
81 int main()
82 {
83     srand(time(0));
84     null = new Treap; root = null;
85     scanf("%d%d", &m, &Limit);
86     int delta = 0;
87     while(m--)
88     {
89         char op; int x;
90         scanf("%c%d", &op, &x);
91         if(op == 'I')
92         {
93             if(x < Limit) continue;
94             insert(root, x - delta);
95         }
96         else if(op == 'A') delta += x;
97         else if(op == 'S')
98         {
99             delta -= x;
100             remove(root, Limit - delta);
101         }
102         else {
103             x = root -> size - x + 1;
104             if(x <= 0) puts("-1");
105             else printf("%d\n", kth(root, x) + delta);
106         }
107     }
108     printf("%d\n", leave);
109     return 0;
110 }

```

3.4 Size Balanced Tree

```

1 int A[N], S[N], L[N], R[N], root, total;
2 void rotate_left(int &x)
3 {
4     int y = R[x];
5     R[x] = L[y];
6     L[y] = x;
7     S[y] = S[x];
8     S[x] = S[L[x]] + S[R[x]] + 1;
9     x = y;
10 }
11 void rotate_right(int &x)
12 {
13     int y = L[x];
14     L[x] = R[y];
15     R[y] = x;

```

```
16     S[y] = S[x];
17     S[x] = S[L[x]] + S[R[x]] + 1;
18     x = y;
19 }
20
21 void maintain(int &p, bool flag)
22 {
23     if(flag)//调整右边
24     {
25         if(S[R[R[p]]] > S[L[p]] rotate_left(p);
26         else if(S[R[L[p]]] > S[L[p]])
27         {
28             rotate_right(R[p]);
29             rotate_left(p);
30         }
31         else return;
32     }
33     else
34     {
35         if(S[L[L[p]]] > S[R[p]] rotate_right(p);
36         else if(S[L[R[p]]] > S[R[p]])
37         {
38             rotate_left(L[p]);
39             rotate_right(p);
40         }
41         else return;
42     }
43     maintain(L[p], 0);
44     maintain(R[p], 1);
45     maintain(p, 0);
46     maintain(p, 1);
47 }
48
49 void insert(int &p, int e)
50 {
51     if(!p)
52     {
53         p = ++total;
54         L[p] = R[p] = 0;
55         A[p] = e; S[p] = 1;
56         return;
57     }
58     S[p]++;
59     if(e < A[p]) insert(L[p], e);
60     else insert(R[p], e);
61     maintain(p, k >= A[p]);
62 }
63
64 int getmin()
65 {
66     for(int x = root; L[x]; x = L[x]);
67     return A[x];
68 }
```

```

69 int getmax()
70 {
71     for(int x = root; R[x]; x = R[x]);
72     return A[x];
73 }
74 int kth(int &p, int k)
75 {
76     int tmp = S[L[p]] + 1;
77     if(k == tmp) return A[p];
78     else if(k < tmp) return kth(L[p], k);
79     else return kth(R[p], k - tmp);
80 }

```

4 Graph

4.1 Shortest path

4.1.1 Dijkstra

```

1 void dijkstra()
2 {
3     memset(dist, 0, sizeof(dist));
4     while(!Q.empty())
5     {
6         int x = Q.top().second; Q.pop();
7         if(done[x]) continue;
8         done[x] = 1;
9         for(Link p = head[x]; p; p = p->next)
10             if(dist[p->y] > dist[x] + p->z)
11             {
12                 dist[p->y] = dist[x] + p->z;
13                 Q.push(make_pair(dist[p->y], p->y));
14             }
15     }
16 }

```

4.1.2 Spfa

```

1 void spfa()
2 {
3     memset(dist, 0x3f, sizeof(dist));
4     Q.push(S); // S为源点
5     while(!Q.empty())
6     {
7         int x = Q.front();
8         Q.pop(); inQ[x] = 0;
9         for(Link p = head[x]; p; p = p->next)
10             if(dist[p->y] > dist[x] + p->z)
11             {
12                 dist[p->y] = dist[x] + p->z;
13                 if(!inQ[p->y])

```

```

14         {
15             Q.push(p->y);
16             inQ[p->y] = 1;
17         }
18     }
19 }
20 }

```

4.1.3 Floyd

```

1 void floyd()
2 {
3     for(int k = 1; k <= n; k++) // 这里可以看作是一个加边的过程
4         for(int i = 1; i <= n; i++)
5             for(int j = 1; j <= n; j++)
6                 map[i][j] = min(map[i][j], map[i][k] + map[k][j]);
7 }
8
9 // 最小环
10 void MinCircle()
11 {
12     cap[] = map[];
13     int circle = 0x3f3f3f3f;
14     for(int k = 1; k <= n; k++)
15     {
16         for(int i = 1; i < k; i++)
17             for(int j = i+1; j < k; j++)
18                 circle = min(circle, map[i][j] + cap[j][k] + cap[k][i]);
19         for(int i = 1; i <= n; i++)
20             for(int j = 1; j <= n; j++)
21                 map[i][j] = min(map[i][j], map[i][k] + map[k][j]);
22     }
23     return circle == 0x3f3f3f3f ? -1 : circle;
24 }
25
26 // floyd判圈法 (大白书 p44)
27 void Circle()
28 {
29     int ans = k;
30     int k1 = k, k2 = k;
31     do{
32         k1 = next(k1);
33         k2 = next(k2); ans = max(ans, k2);
34         k2 = next(k2); ans = max(ans, k2);
35     }while(k1 != k2);
36     return ans;
37 }

```

4.2 Minimum Spanning Tree

4.2.1 Prim


```

1 void prime()
2 {
3     memset(dist, 0, sizeof(dist));
4     int res = 0;
5     while(!Q.empty())
6     {
7         int x = Q.top().second;
8         if(done[x]) {Q.pop(); continue;}
9         res += Q.top().first;
10        Q.pop();
11        for(Link p = head[x]; p; p = p->next)
12            if(dist[p->y] > p->z)
13            {
14                dist[p->y] = p->z;
15                Q.push(make_pair(dist[p->y], p->y));
16            }
17    }
18 }

```

4.2.2 Kruskal

```

1 void prime()
2 {
3     sort(edge, edge+Cnt, cmp);
4     int res = 0;
5     for(int i = 0; i < Cnt; i++)
6     {
7         if(getroot(edge[i].x) == getroot(edge[i].y)) continue;
8         merge(edge[i].x, edge[i].y);
9         res += edge[i].z;
10    }
11 }

```

4.3 Tarjan - Strong Union

```

1 void dfs(int x)
2 {
3     now[x] = low[x] = ++dfstime;
4     hash[x] = 1;
5     st.push(x); inst[x] = 1;
6     for(int i = 1; i <= n; i++)
7         if(map[x][i])
8         {
9             if(!hash[i])
10            {
11                dfs(i);
12                low[x] = min(low[x], low[i]);
13            }
14            else if(inst[i]) low[x] = min(low[x], now[i]);
15        }
16    if(low[x] == now[x])

```

```

17     {
18         while(!st.empty())
19         {
20             int u = st.top();
21             st.pop(); inst[u] = 0;
22             belong[u] = number;
23             if(u == x) break;
24         }
25         numer++;
26     }
27 }
28 void tarjan()
29 {
30     for(int i = 1; i <= n; i++)
31         if(!hash[i]) dfs(i);
32     if(!st.empty()) // 这是一个未知 bug    栈中还会剩下一个强连通分量
33     {
34         while!st.empty()
35         {
36             int u = st.top();
37             st.pop();
38             belong[u] = number;
39         }
40         number++;
41     }
42 }

```

4.4 LCA

4.4.1 Tarjan

```

1 // poj 1330 (changed something)
2 // LCA tarjan
3 #include <cstdio>
4 #include <cstring>
5
6 const int N = 10000 + 10;
7
8 int n;
9 struct Link{int y, idx; Link *next;}*head[N], *ask[N];
10 int tx, ty;
11 bool in[N], vis[N];
12 int f[N];
13 int ans[N]; // Query Answer
14
15 void inLink(int x, int y)
16 {
17     Link *p = new Link;
18     p->y = y;
19     p->next = head[x];
20     head[x] = p;
21 }

```

```

22 void inAsk(int x, int y, int idx)
23 {
24     Link *p = new Link;
25     p -> y = y;
26     p -> idx = idx;
27     p -> next = ask[x];
28     ask[x] = p;
29 }
30
31 int getroot(int x)
32 {
33     return f[x] == x ? x : f[x] = getroot(f[x]);
34 }
35
36 void LCA(int x)
37 {
38     vis[x] = 1;
39     f[x] = x;
40     for(Link *p = ask[x]; p; p = p -> next)
41         if(vis[p->y]) ans[p->idx] = getroot(p->y);
42     for(Link *p = head[x]; p; p = p -> next)
43         if(!vis[p->y])
44         {
45             LCA(p->y);
46             f[p->y] = x;
47         }
48 }
49
50 int main()
51 {
52     int T; scanf("%d", &T);
53     while(T--)
54     {
55         memset(head, 0, sizeof(head));
56         memset(ask, 0, sizeof(ask));
57         memset(in, 0, sizeof(in));
58         memset(vis, 0, sizeof(vis));
59         scanf("%d", &n);
60         for(int i = 1; i <= n; i++) f[i] = i;
61         for(int i = 1; i < n; i++)
62         {
63             int x, y;
64             scanf("%d%d", &x, &y);
65             inLink(x, y);
66             in[y] = 1;
67         }
68         int q = 1; // the number of query
69         for(int i = 1; i <= q; i++)
70         {
71             int x, y; scanf("%d%d", &x, &y);
72             inAsk(x, y, i); inAsk(y, x, i);
73         }
74         int root = -1;

```

```

75     for(int i = 1; i <= n; i++)
76         if(!in[i]) {root = i; break;}
77     LCA(root);
78     for(int i = 1; i <= q; i++)
79         printf("%d\n", ans[i]);
80 }
81 return 0;
82 }

```

4.4.2 Doubling Algorithm

还不会...

4.5 Bipartite Graph

4.5.1 Maximal Matching - The Hungarian algorithm

```

1  int ttt = 0; // 全局时间戳变量
2
3  bool search(int x)
4  {
5      for(int i = 1; i <= m; i++)
6          if(map[x][i] && vis[i] != ttt)
7              {
8                  vis[i] = ttt;
9                  if(pre[i] == -1 || search(pre[i]))
10                     {
11                         pre[i] = x;
12                         return 1;
13                     }
14             }
15     return 0;
16 }
17
18 int match()
19 {
20     int res = 0;
21     for(int i = 1; i <= n; i++)
22     {
23         ++ttt; // 这里不用 memset 节省时间
24         res += search(i);
25     }
26     return res;
27 }

```

4.5.2 Optimal Matching - KM

不会... 用费用流解决

4.6 Network Flow

4.6.1 Maximum Flow - isap

```

1 //  h[x] : 点 x 在第 h[x] 层
2 //  v[k] : 第 k 层有 v[k] 个点
3 int sap(int x, int flow)
4 {
5     if(x == n) return flow;
6     int res = 0;
7     for(int i = S; i <= T; i++)
8         if(g[x][i] && h[x] == h[i] + 1)
9             {
10                 int t = sap(i, min(g[x][i], flow - res));
11                 res += t; g[x][i] -= t; g[i][x] += t;
12                 if(res == flow) return res;
13                 if(h[S] >= T) return res;
14             }
15     //if(h[S] >= T) return res;
16     if((--v[h[x]]) == 0) h[S] = T;
17     ++v[++h[x]];
18     return res;
19 }
20 int main()
21 {
22     v[0] = T;
23     int maxflow = 0;
24     while(h[S] < T) maxflow += sap(1, inf);
25     return 0;
26 }

```

4.6.2 Minimum Cost Maximum Flow - spfa

```

1 struct EG{int from,to,flow,cost,next;}edge[M];
2
3 void add_edge(int a,int b,int c,int d)
4 {
5     edge[L]=(EG){a,b,c,+d,head[a]};
6     head[a]=L++;
7     edge[L]=(EG){b,a,0,-d,head[b]};
8     head[b]=L++;
9 }
10
11 bool spfa()
12 {
13     memset(inQ, 0, sizeof(inQ));
14     memset(dist, 0x3f, sizeof(dist));
15     dist[S] = 0;
16     q.push(S);
17     while(!q.empty())
18     {
19         int x = q.front();

```

```

20     q.pop();
21     inQ[x] = 0;
22     for(int i = head[x]; i != -1; i = edge[i].next)
23         if(edge[i].flow && dist[edge[i].to] > dist[x] + edge[i].
           cost)
24             {
25                 pre[edge[i].to] = i;
26                 dist[edge[i].to] = dist[x] + edge[i].cost;
27                 if(!inQ[edge[i].to])
28                     {
29                         inQ[edge[i].to] = 1;
30                         q.push(edge[i].to);
31                     }
32             }
33     }
34     return dist[T] != inf;
35 }
36 void MFMC()
37 {
38     memset(head, -1, sizeof(head));
39     建图调用 add_edge();
40
41     int mincost = 0, maxflow = 0;
42     while(spfa())
43     {
44         int res = inf;
45         for(int i = T; i != S; i = edge[pre[i]].from)
46             {
47                 res = min(res, edge[pre[i]].flow);
48             }
49         for(int i = T; i != S; i = edge[pre[i]].from)
50             {
51                 edge[pre[i]].flow -= res;
52                 edge[pre[i] ^ 1].flow += res;
53             }
54         maxflow += res;
55         mincost += res * dist[T];
56     }
57 }

```

5 Geometry

5.1 Convex Hull

```

1 //点集 p 的凸包 μlist[0~n-1]
2 //点集 s 的凸包 stack[0~top-1]
3 Point list[Maxn];
4 int Stack[Maxn],top;
5 bool _cmp (Point p1,Point p2)
6 {
7     double tmp=(p1-list[0])^(p2-list[0]);

```

```

8     if (fuhao(tmp)>0) return true;
9     else if (fuhao(tmp)==0&&fuhao(dist(p1,list[0])-dist(p2,list[0]))
        <=0)
10         return true;
11     else    return false;
12 }
13 void Graham(int n)
14 {
15     Point p0;
16     int k=0;
17     p0=list[0];
18     for (int i=1;i<n;++i)
19     {
20         if ((p0.y>list[i].y)||((p0.y==list[i].y&&p0.x>list[i].x))
21             {
22             p0=list[i];
23             k=i;
24         }
25     }
26     swap(list[k],list[0]);
27     sort(list+1,list+n,_cmp);
28     if (n==1)
29     {
30         top=1;
31         stack[0]=0;
32         return;
33     }
34     if (n==2)
35     {
36         top=2;
37         stack[0]=0;
38         stack[1]=1;
39         return;
40     }
41     stack[0]=0;
42     stack[1]=1;
43     top=2;
44     for (int i=2;i<n;++i)
45     {
46         while (top>1 && fuhao((list[stack[top-1]]-list[stack[top-2]])^(
            list[i]-list[stack[top-2]]))<=0)
47             top--;
48         stack[top++]=i;
49     }
50 }

```

5.2 All

```

1 #include <cstdio>
2 #include <cstdlib>
3 #include <cstring>
4 #include <cmath>

```

```

5 #include <algorithm>
6 #include <utility>
7 using std::max;
8 using std::min;
9 using std::sort;
10 using std::swap;
11 using std::pair;
12 using std::make_pair;
13 const double eps = 1e-8, inf = 1e20;
14 const double pi = 4.0 * atan(1.0);
15 #define Degree(_rad) (180.0 / pi * (_rad))
16
17 int fuhao(double x)
18 {
19     if (fabs(x)<eps) return 0;
20     if (x<0) return -1;
21     else return 1;
22 }
23
24 //////////////////////////////// Point && Vector
25 ////////////////////////////////
26 struct Point{
27     double x, y;
28     Point (){}
29     Point (double _x,double _y):x(_x),y(_y){}
30     void init(double a, double b) { x = a; y = b; }
31
32     // basic calc
33     bool operator == (const Point &b) const
34     {
35         return !fuhao(x - b.x) && !fuhao(y - b.y);
36     }
37     Point operator + (const Point &b) const
38     {
39         return Point(x + b.x, y + b.y);
40     }
41     Point operator - (const Point &b) const
42     {
43         return Point(x - b.x, y - b.y);
44     }
45     Point operator * (const double &b) const
46     {
47         return Point(x * b, y * b);
48     }
49
50     Point Rotate(Point p, double alpha) // alpha E [0, +oo) 逆时针
51     {
52         double x0 = p.x, y0 = p.y;
53         double tx = x - x0, ty = y - y0;
54         double nx = tx * cos(alpha) - ty * sin(alpha);
55         double ny = tx * sin(alpha) + ty * cos(alpha);
56         nx += x0; ny += y0;
57         return Point(nx, ny);

```



```

57     }
58
59     // Vector
60     double operator *(const Point &b)const
61     { // Dot
62         return x * b.x + y * b.y;
63     }
64     double operator ^ (const Point &b)const
65     { // Cross
66         return x * b.y - y * b.x;
67     }
68     double Abs() { return sqrt(x * x + y * y); }
69 };
70 double Dist(const Point &a, const Point &b) { return (a - b).Abs(); }
71 typedef Point Vector;
72
73 double Angle(Vector a, Vector b)
74 {
75     return acos(a * b / a.Abs() / b.Abs());
76 }
77 Vector Get_H(Vector A)
78 { // 求与向量垂直的单位向量    使用前确保不为0向量
79     // A != Vector(0.0, 0.0);
80     double L = A.Abs();
81     return Vector(-A.y / L, A.x / L);
82 }
83
84 ////////////////////////////////////////      E - N - D
85     ////////////////////////////////////////
86
87 ////////////////////////////////////////      Line
88     ////////////////////////////////////////
89 struct Line{
90     Point s,e;
91     Line() {}
92     Line(Point ss, Point ee)
93     {
94         s = ss; e = ee;
95     }
96
97     // 两直线的关系：重合0， 平行1， 相交2 并返回交点
98     pair<int,Point> operator &(const Line &b) const
99     {
100         Point ans = s;
101         if(fuhao((s-e)^(b.s-b.e))==0)
102         {
103             if (fuhao((s-b.e)^(b.s-b.e))==0)
104                 return make_pair(0,ans); //重合
105             else return make_pair(1,ans); //平行
106         }
107         double t = ((s-b.s)^(b.s-b.e)) / ((s-e)^(b.s-b.e));
108         ans.x += (e.x-s.x) * t;

```

```

108     ans.y += (e.y-s.y) * t;
109     return make_pair(2,ans); //相交
110 }
111 };
112 /////////////////////////////////////////////////// E - N - D
113 ///////////////////////////////////////////////////
114 //判断线段相交
115 bool inter(Line l1,Line l2)
116 {
117     return
118     max(l1.s.x,l1.e.x) >= min(l2.s.x,l2.e.x) &&
119     max(l1.s.y,l1.e.y) >= min(l2.s.y,l2.e.y) &&
120     max(l2.s.x,l2.e.x) >= min(l1.s.x,l1.e.x) &&
121     max(l2.s.y,l2.e.y) >= min(l1.s.y,l1.e.y) &&
122     fuhao((l2.s-l1.e)^(l1.s-l1.e)) * fuhao((l2.e-l1.e)^(l1.s-l1.e))<=0
123     &&
124     fuhao((l1.s-l2.e)^(l2.s-l2.e)) * fuhao((l1.e-l2.e)^(l2.s-l2.e))<=0;
125 }
126 //判断直线与线段相交
127 bool Seg_inter_line(Line l1,Line l2)//l1为直线 l2为线段
128 {
129     return fuhao((l2.s-l1.e)^(l1.s-l1.e))*fuhao((l2.e-l1.e)^(l1.s-l1.e))
130     )<=0;
131 }
132 //点到直线距离
133 //返回点到直线最近的点
134 Point PointToLine(Point P,Line L)
135 {
136     Point ans;
137     double t=((P-L.s)*(L.e-L.s))/((L.e-L.s)*(L.e-L.s));
138     ans.x=L.s.x+(L.e.x-L.s.x)*t;
139     ans.y=L.s.y+(L.e.y-L.s.y)*t;
140     return ans;
141 }
142 //点到线段距离
143 //返回点到线段最近的点
144 Point NearestPointToLineSeg(Point P,Line L)
145 {
146     Point ans;
147     double t = ((P-L.s)*(L.e-L.s)) / ((L.e-L.s)*(L.e-L.s));
148     if (t>=0&&t<=1)
149     {
150         ans.x = L.s.x + (L.e.x-L.s.x)*t;
151         ans.y = L.s.y + (L.e.y-L.s.y)*t;
152     }
153     else {
154         if (Dist(P,L.s)<Dist(P,L.e))
155             ans = L.s;
156         else ans = L.e;
157     }
158     return ans;
159 }

```

```

158 // 多边形面积
159 double CalcArea(Point p[],int n)
160 {
161     double ans=0;
162     for (int i=0;i<n;++i)
163         ans+=(p[i]^p[(i+1)%n])/2;
164     return fabs(ans);
165 }
166 // 判断点在线段上
167 bool OnSeg(Point P,Line L)
168 {
169     return
170         fuhao((L.s-P)^(L.e-P))==0 &&
171         fuhao((P.x-L.s.x)*(P.x-L.e.x))<=0 &&
172         fuhao((P.y-L.s.y)*(P.y-L.e.y))<=0;
173 }
174 // 三点求圆心坐标
175 Point waixin(Point a,Point b,Point c)
176 {
177     double a1=b.x-a.x,b1=b.y-a.y,c1=(a1*a1+b1*b1)/2;
178     double a2=c.x-a.x,b2=c.y-a.y,c2=(a2*a2+b2*b2)/2;
179     double d=a1*b2-a2*b1;
180     return Point(a.x+(c1*b2-c2*b1)/d,a.y+(a1*c2-a2*c1)/d);
181 }
182
183
184 ///////////////////////////////////////////////////          Graham
185 ///////////////////////////////////////////////////
186 // 求凸包 点list[0~n-1]
187 // 凸包结果Stack[0~top-1]
188 const int Maxn = 100;////////////////////////////////////here!!
189 Point list[Maxn];          //////////////////////////////////??????? 补全 Maxn
190 //????????????????????
191 int Stack[Maxn],top;
192 bool _cmp (Point p1,Point p2)
193 {
194     double tmp=(p1-list[0])^(p2-list[0]);
195     if (fuhao(tmp)>0) return true;
196     else if (fuhao(tmp)==0&&fuhao(Dist(p1,list[0])-Dist(p2,list[0]))
197         <=0)
198         return true;
199     else return false;
200 }
201 void Graham(int n)
202 {
203     Point p0;
204     int k=0;
205     p0=list[0];
206     for (int i=1;i<n;++i)
207     {
208         if ((p0.y>list[i].y)||((p0.y==list[i].y&&p0.x>list[i].x))
209             {
210                 p0=list[i];

```

```

208         k=i;
209     }
210 }
211 swap(list[k],list[0]);
212 sort(list+1,list+n,_cmp);
213 if (n==1)
214 {
215     top=1;
216     Stack[0]=0;
217     return;
218 }
219 if (n==2)
220 {
221     top=2;
222     Stack[0]=0;
223     Stack[1]=1;
224     return;
225 }
226 Stack[0]=0;
227 Stack[1]=1;
228 top=2;
229 for (int i=2;i<n;++i)
230 {
231     while (top>1 && fuhao((list[Stack[top-1]]-list[Stack[top-2]])^(
        list[i]-list[Stack[top-2]]))<=0)
232         top--;
233     Stack[top++]=i;
234 }
235 }
236 ////////////////////////////////////////      E - N - D
237 ////////////////////////////////////////
238
239 ////////////////////////////////////////      Area
240 ////////////////////////////////////////
241 double PolygonArea(Point *pp, int nn) // pp[0, n-1]
242 {
243     double ans_area = 0.0;
244     for(int i = 1; i < nn-1; i++)
245     {
246         ans_area += (pp[i] - pp[0]) ^ (pp[i+1] - pp[0]);
247     }
248     return fabs(ans_area / 2);
249 }
250 ////////////////////////////////////////      E - N - D
251 ////////////////////////////////////////
252 ////////////////////////////////////////      点 在 多 边 形 内
253 ////////////////////////////////////////
254 int isPointInPolygon(Point p, Point *poly, int nn)
255 {
256     int w = 0;
257     for(int i = 0; i < n; i++)

```

```

256     {
257         if(OnSeg(p, Line(poly[i], poly[(i+1)%n]))) return -1; // 边界上
258         int k = fuhao((poly[(i+1)%n] - poly[i]) ^ (p - poly[i]));
259         int d1 = fuhao(poly[i].y - p.y);
260         int d2 = fuhao(poly[(i+1)%n].y - p.y);
261         if(k > 0 && d1 <= 0 && d2 > 0) wn++;
262         if(k < 0 && d1 > 0 && d2 <= 0) wn--;
263     }
264     if(wn != 0) return 1; //内部
265     return 0; // 外部
266 }
267 /////////////////////////////////////////////////// E - N - D
268 ///////////////////////////////////////////////////
269
270 int main()
271 {
272 }

```

6 String

6.1 Manacher

```

1  #include <cstdio>
2  #include <algorithm>
3  // HDU 3068
4  const int N = 110000 + 10;
5
6  char t[N], s[2*N];
7  int n, p[2*N];
8
9  void pre(char *origin, char *str, int &_len)
10 {
11     _len = 0;
12     str[_len++] = '$';
13     for(int i = 0; origin[i]; i++)
14     {
15         str[_len++] = '#';
16         str[_len++] = origin[i];
17     }
18     str[_len++] = '#';
19     str[_len] = 0;
20     //puts(str);
21 }
22
23 void getPi(char *str, int _len, int *_P)
24 {
25     int mx = 0, id;
26     for(int i = 1; i < _len; i++)
27     {
28         if(mx > i) _P[i] = std::min(_P[2*id-i], mx-i);

```

```

29     else _P[i] = 1;
30     for(; str[i+_P[i]] == str[i-_P[i]]; _P[i]++) ;
31     if(_P[i] + i > mx)
32     {
33         mx = _P[i] + i;
34         id = i;
35     }
36 }
37 }
38
39 int main()
40 {
41     while(scanf("%s", t) == 1)
42     {
43         pre(t, s, n);
44         getPi(s, n, p);
45         int res = 1;
46         for(int i = 1; i < n; i++)
47             res = std::max(res, p[i]-1);
48         printf("%d\n", res);
49     }
50     return 0;
51 }

```

6.2 KMP

```

1  #include <cstdio>
2  #include <cstring>
3  // POJ 3461 : Count the number of t occurrences in s
4  char s[1000000+10], t[1000000+10];
5  int next[1000000+10];
6
7  void getNext(char *t, int len, int *Next)
8  {
9      memset(Next, 0, sizeof(Next)); Next[0] = -1;
10     for(int j = 0, k = -1; j < len; )
11     {
12         if(k == -1 || t[j] == t[k]) Next[++j] = ++k;
13         else k = Next[k];
14     }
15 }
16 int kmp(char *s, int lens, char *t, int lent)
17 {
18     int res = 0;
19     getNext(t, lent, next);
20     for(int i = 0, j = 0; i < lens; )
21     {
22         if(j == -1 || s[i] == t[j]) { i++; j++; }
23         else j = next[j];
24         if(j == lent) res++; // Bingo! [pos = j - lent]
25     }
26     return res;

```

```

27 }
28
29 int main()
30 {
31     int T; scanf("%d", &T);
32     while(T--)
33     {
34         scanf("%s%s", t, s);
35         printf("%d\n", kmp(s, strlen(s), t, strlen(t)));
36     }
37     return 0;
38 }

```

6.3 Suffix Array

```

1  #include <cstdio>
2  #include <algorithm>
3  #include <map>
4  using std::map;
5  // POJ 3261 找重复了K次的最长子串
6  const int N = 20000 + 10;
7  /*
8      sa[rank[i]] = i
9      sa[i] = j      : rank i is s[j, n)
10     rank[j] = i      : s[j, n) is rank i
11     height[i] = j    : the longest common prefix of string rank _i and
                        _i-1
12 */
13
14 int sa[N], rank[N];
15 int c[N], tmp[N];
16 int height[N];
17
18 bool cmp(int *r, int a, int b, int l)
19 {
20     return r[a] == r[b] && r[a+l] == r[b+l];
21 }
22
23 void DA(int *s, int n, int m) // s[0...n-1] E [1, m)
24 {
25     int i, j, p, *x = rank, *y = tmp;
26     for(i = 0; i < m; i++) c[i] = 0;
27     for(i = 0; i < n; i++) c[x[i] = s[i]]++;
28     for(i = 1; i < m; i++) c[i] += c[i-1];
29     for(i = n-1; i >= 0; i--) sa[--c[x[i]]] = i;
30     for(j = 1, p = 0; p < n; j *= 2, m = p)
31     {
32         for(p = 0, i = n-j; i < n; i++) y[p++] = i;
33         for(i = 0; i < n; i++) if(sa[i] >= j) y[p++] = sa[i] - j;
34         for(i = 0; i < m; i++) c[i] = 0;
35         for(i = 0; i < n; i++) c[x[y[i]]]++;
36         for(i = 1; i < m; i++) c[i] += c[i-1];

```

```

37     for(i = n-1; i >= 0; i--) sa[--c[x[y[i]]]] = y[i];
38     for(std::swap(x, y), p = 1, x[sa[0]] = 0, i = 1; i < n; i++)
39         x[sa[i]] = cmp(y, sa[i], sa[i-1], j) ? p - 1 : p++;
40 }
41 for(i = 0; i < n; i++) rank[sa[i]] = i;
42
43 int k = 0; height[0] = 0;
44 for(i = 0; i < n; height[rank[i+1]] = k) if(rank[i])
45     for(k ? k-- : 0, j = sa[rank[i]-1]; s[j+k] == s[i+k]; k++);
46 }
47
48 int n, K, a[N];
49 map<int, int> hash;
50
51 bool check(int len)
52 {
53     int cnt = 0;
54     for(int i = 1; i < n; i++)
55     {
56         if(height[i] >= len) cnt++;
57         else cnt = 0;
58         if(cnt >= K - 1) return 1;
59     }
60     return 0;
61 }
62
63 int Solve()
64 {
65     int low = 0, high = n, ans = 0;
66     while(low <= high)
67     {
68         int mid = low + (high - low) / 2;
69         if(check(mid)) { low = mid + 1; ans = mid; }
70         else high = mid - 1;
71     }
72     return ans;
73 }
74
75 int main()
76 {
77     //-----Read-----
78     scanf("%d%d", &n, &K);
79     for(int i = 0; i < n; i++)
80     {
81         scanf("%d", &a[i]);
82         tmp[i] = a[i];
83     }
84     std::sort(tmp, tmp+n);
85     int cnt = 0;
86     for(int i = 0; i < n; i++)
87         if(i == 0 || tmp[i] != tmp[i-1]) hash[tmp[i]] = ++cnt;
88     for(int i = 0; i < n; i++) a[i] = hash[a[i]];
89     a[n++] = 0; ///////////////

```



```

90     DA(a, n, cnt+1);
91 /*   for(int i = 0; i < n; i++)
92     {
93         printf("rank = %d -> [%d, %d) [%d] :", i, sa[i], n, height[i]);
94         for(int j = sa[i]; j < n; j++) printf(" %d", a[j]);
95         puts("");
96     } */
97     printf("%d\n", Solve());
98     return 0;
99 }

```

7 Tools

7.1 BigInteger - C++

```

1 //程序中全部为正整数之间的操作
2 #include <cstdio>
3 #include <cstring>
4 #include <algorithm>
5 using std::max;
6
7 const int base = 10000; // 压4位
8
9 struct BigInt{
10     int c[1000], len, sign;
11     BigInt() { memset(c, 0, sizeof(c)); len = 1; sign = 0; }
12     void Zero()
13     {
14         while(len > 1 && c[len] == 0) len--;
15         if(len == 1 && c[len] == 0) sign = 0;
16     }
17     void writein(char *s)
18     {
19         int k = 1, L = strlen(s);
20         for(int i = L-1; i >= 0; i--)
21         {
22             c[len] += (s[i]-'0') * k;
23             k *= 10;
24             if(k == base)
25             {
26                 k = 1;
27                 len++;
28             }
29         }
30     }
31     void Read()
32     {
33         char s[5000] = {0};
34         scanf("%s", s);
35         writein(s);
36     }

```

```

37 void Print()
38 {
39     if(sign) printf("-");
40     printf("%d", c[len]);
41     for(int i = len-1; i >= 1; i--) printf("%04d", c[i]);
42     printf("\n");
43 }
44 BigInt operator = (int a)
45 {
46     char s[100] = {0};
47     sprintf(s, "%d", a);
48     writein(s);
49     return *this;
50 }
51 bool operator > (const BigInt &b)
52 {
53     if(len != b.len) return len > b.len;
54     for(int i = len; i >= 1; i--)
55     {
56         if(c[i] != b.c[i]) return c[i] > b.c[i];
57     }
58     return 0;
59 }
60 bool operator < (const BigInt &b)
61 {
62     if(len != b.len) return len < b.len;
63     for(int i = len; i >= 1; i--)
64     {
65         if(c[i] != b.c[i]) return c[i] < b.c[i];
66     }
67     return 0;
68 }
69 bool operator == (const BigInt &b)
70 {
71     if(len != b.len) return 0;
72     for(int i = 1; i <= len; i++)
73         if(c[i] != b.c[i]) return 0;
74     return 1;
75 }
76 bool operator == (const int &a)
77 {
78     BigInt b; b = a;
79     return *this == b;
80 }
81 BigInt operator + (const BigInt &b)
82 {
83     BigInt r; r.len = max(len, b.len) + 1;
84     for(int i = 1; i <= r.len; i++)
85     {
86         r.c[i] += c[i] + b.c[i];
87         r.c[i+1] += r.c[i] / base;
88         r.c[i] %= base;
89     }

```

```

90         r.Zero();
91         return r;
92     }
93     BigInt operator + (const int &a)
94     {
95         BigInt b; b = a;
96         return *this + b;
97     }
98     BigInt operator - (const BigInt &b)
99     {
100         BigInt a, c; // a - c
101         a = *this; c = b;
102         if(a < c)
103         {
104             std::swap(a, c);
105             a.sign = 1;
106         }
107         for(int i = 1; i <= len; i++)
108         {
109             a.c[i] -= c.c[i];
110             if(a.c[i] < 0)
111             {
112                 a.c[i] += base;
113                 a.c[i+1]--;
114             }
115         }
116         a.Zero();
117         return a;
118     }
119     BigInt operator - (const int &a)
120     {
121         BigInt b; b = a;
122         return *this - b;
123     }
124     BigInt operator * (const BigInt &b)
125     {
126         BigInt r; r.len = len + b.len + 2;
127         for(int i = 1; i <= len; i++)
128         {
129             for(int j = 1; j <= b.len; j++)
130             {
131                 r.c[j+i-1] += c[i] * b.c[j];
132             }
133         }
134         for(int i = 1; i <= r.len; i++)
135         {
136             r.c[i+1] += r.c[i] / base;
137             r.c[i] %= base;
138         }
139         r.Zero();
140         return r;
141     }
142     BigInt operator * (const int &a)

```

```

143     {
144         BigInt b; b = a;
145         return *this * b;
146     }
147     BigInt operator / (BigInt b)//整除
148     {
149         BigInt t, r;
150         if(b == 0) return r;
151         r.len = len;
152         for(int i = len; i >= 1; i--)
153         {
154             t = t * base + c[i];
155             int div;
156             //-----try-----
157             int up = 10000, down = 0;
158             while(up >= down)
159             {
160                 int mid = (up + down) / 2;
161                 BigInt ccc ; ccc = b * mid;
162                 if(ccc > t) up = mid - 1;
163                 else {
164                     down = mid + 1;
165                     div = mid;
166                 }
167             }
168             //-----end-----
169             r.c[i] = div;
170             t = t - b * div;
171         }
172         //最后的t为余数，要用的自己想办法传出去
173         r.Zero();
174         return r;
175     }
176     BigInt operator / (const int &a)
177     {
178         BigInt b; b = a;
179         return *this / b;
180     }
181     BigInt operator % (const BigInt &b)
182     { //其实可以复制上面除法的，这里换一种写法
183         return *this - *this / b * b;
184     }
185     BigInt operator % (const int &a)
186     {
187         BigInt b; b = a;
188         return *this % b;
189     }
190 };
191
192 int main()
193 {
194     return 0;
195 }

```

7.2 C char*

```

1 头文件cstring
2 strlen(s); //获取长度 O(N)
3 strcpy(a+2,b+1) //从b+1开始全部赋值给a+2开始的字符串
4 strncpy(a+2,b+1,2) //从b+1开始赋值2个给a+2开始的字符串
5 strcmp(a,b) //比较a和b的大小, 相等返回0, a>b返回正整数
6 strcat(a,b) //相当于string类的 a += b;
7 strstr(a,b)-a; //返回b在a中第一次出现的位置, 不存在返回NULL(即0), 由于-a
    , 所以最后应该是-a

```

7.3 C++ std::string

```

1 //====初始化====
2 头文件string并加上std::
3 string s(str); //相当于string s=str;
4 string s(cstr); //把char数组类型的字符串cstr作为s的初值
5 s.clear(); //清空, 相当于 s="";
6
7 //====长度====
8 s.length(); //获取s的长度, O(1)
9 s.size(); //一样
10
11 //====插入删除====
12 s.insert(2, "a"); //在s的位置2插入string类字符串"a"
13 s.erase(2, 3); //从s的位置2开始删除3个字符
14
15 //====查找====
16 s.find("abc"); //查找字符串"abc"在s中第一次出现的位置 (据说是KMP实现的)
17 //s="aabcc"; printf("%d %d\n", (int)s.find("abc"), (int)s.find("aabb"));
18 //上一行程序应输出 1 -1 (若没找到必须强行转换为int才为 -1 )

```

7.4 Batch test

7.4.1 @Linux

```

1 mkdata=mk
2 filea=a
3 fileb=b
4
5 g++ $mkdata.cpp -o $mkdata
6 g++ $filea.cpp -o $filea
7 g++ $fileb.cpp -o $fileb
8 cas=0
9 while true; do
10     ./ $mkdata > $filea.in
11     ./ $filea < $filea.in > $filea.out
12     ./ $fileb < $filea.in > $fileb.out
13     if ! diff $filea.out $fileb.out
14     then
15         echo "Wrong Answer"

```

```

16         break
17     fi
18     echo $((cas=cas+1)) "␣Accepted"
19 done

```

7.4.2 @Windows

```

1 :loop
2     mk > A.in
3     A < A.in > A.out
4     p < A.in > p.out
5     fc A.out p.out
6     if errorlevel 1 goto end
7     goto loop
8 :end
9     pause

```

7.5 Vimrc Config For Linux

```

1 set nobackup
2 set cin
3 set nu
4 set st=4
5 set ts=4
6 set sw=4
7
8 map <F7> <Esc>:w<CR>:!javac %:r.java<CR>:!java %:r<CR>
9 imap <F7> <Esc>:w<CR>:!javac %:r.java<CR>:!java %:r<CR>
10 map <F8> <Esc>:w<CR>:!g++ -g %:r.cpp -o %:r<CR>:!gdb %:r<CR>
11 imap <F8> <Esc>:w<CR>:!g++ -g %:r.cpp -o %:r<CR>:!gdb %:r<CR>
12 map <F9> <Esc>:w<CR>:!g++ -g %:r.cpp -o %:r<CR>:!./%:r<CR>
13 imap <F9> <Esc>:w<CR>:!g++ -g %:r.cpp -o %:r<CR>:!./%:r<CR>

```