

## Contents

1	Setting	1
1.1	/.vimrc	1
1.2	/cp.sh	1
1.3	/new.sh	1
2	General	1
2.1	Template	1
2.2	/buglist	2
2.3	Builtin	2
2.4	BinarySearch	2
2.5	int128	2
2.6	StableMatching	2
2.7	Mergesort	3
2.8	Multi	3
2.9	ThreeSearch	3
2.10	Tree Policy	3
3	Data and Structure	4
3.1	Mo	4
3.2	Segment Tree	4
3.3	Treap	4
4	DP	5
4.1	Backpack Limit	5
4.2	CounterLine	5
4.3	LCS	6
4.4	LIS	6
4.5	ReRoot	6
4.6	TSP	6
5	Geometry	7
5.1	Basic	7
5.2	Convex Hull	7
6	Graph	7
6.1	Edge	7
6.2	Bloosom	8
6.3	CLE	8
6.4	Disjoint Set	9
6.5	Longest Common Ancestor	9
6.6	MST	10
6.7	TopologicalSort	10
6.8	TreeCentroid	10
7	Graph Bipartite	11
7.1	Bipartite	11
7.2	BipartiteMatch	11
7.3	KM	11
7.4	Relation	12
8	Graph Connectivity	12
8.1	decide	12
8.2	low	12
9	Graph Flow	12
9.1	Dinic	12
9.2	MCMF	13
10	Graph Shortest Path	14
10.1	BellmanFord	14
10.2	Dijkstra	14
10.3	FloydWarshall	15
10.4	SPFA	15
11	Math	15
11.1	Catalan	15
11.2	Combination	15
11.3	Extend Euclidean.cpp	16
11.4	FFT	16
11.5	GaussElimination	16
11.6	Matrix	17
11.7	Phi	17
11.8	PowerTower	17
11.9	Prime table	17
12	String	17
12.1	Aho Corasick	17
12.2	KMP	18
12.3	Manacher	19
12.4	Trie	19
12.5	Zvalue	19

## 1 Setting

### 1.1 /.vimrc

```

1 syntax on
2 color torte
3 set nu ts=4 sw=4 ai mouse=a bs=2 ci hls ru nocp
   showmatch ar fencs=utf-8
4 set guifont=Consolas:h10
5 filetype plugin indent on
6 so $VIMRUNTIME/mswin.vim
7 behave mswin
8
9 autocmd CursorMoved * exe printf('match VisualNOS
   /\V<%s\>/', escape(expand('<word>'), '\'))
10 autocmd CursorMovedi * exe printf('match VisualNOS
   /\V<%s\>/', escape(expand('<word>'), '\'))
11
12 map <F5> :r ~/sample.cpp<CR>
13 map <F9> :call Compile()<CR>
14 map! <F9> <ESC>:call Compile()<CR>
15 map <F10> :call Run()<CR>
16 map! <F10> <ESC>:call Run()<CR>
17
18 func! Compile()
19     exec "w"
20     exec "g++ -Wall -Wshadow -std=gnu++0x % -o %<
       2>log.txt"
21     exe "cg log.txt"
22     cw 5
23 endfunc
24
25 func! Run()
26     exec "!./%<" # "!%<" if windows
27 endfunc
28
29 cd ~/Desktop # C:\Users\???\Desktop

```

### 1.2 /cp.sh

```

1 #!/bin/bash
2 clear
3 g++ $1.cpp -DDBG -o $1
4 if [[ "$?" == "0" ]]; then
5     echo Running
6     ./$1 < $1.in > $1$2.out
7     echo END
8 fi

```

### 1.3 /new.sh

```

1 #!/bin/bash
2
3 clear
4 cat template.cpp > $1.cpp
5 touch $1.in $1.out
6 echo $1 Created

```

## 2 General

### 2.1 Template

```

1 #pragma GCC optimize("O2")
2 #include <bits/stdc++.h>
3 using namespace std;
4 using LL = long long;
5 using ULL = unsigned long long;
6 using PII = pair<int,int>;

```

```

7| using PLL = pair<LL, LL>;
8| using VI = vector<int>;
9| using VVI = vector<vector<int>>>;
10| using dvt = double;
11| const int INF = 1e9;
12| const int MXN = 0;
13| const int MXV = 0;
14| const double EPS = 1e-9;
15| const int MOD = 1e9+7;
16| #define MP make_pair
17| #define PB push_back
18| #define Fi first
19| #define Se second
20| #define FOR(i, L, R) for(int i = L; i < (int)R; ++i)
21| #define FORD(i, L, R) for(int i = L; i > (int)R; --i)
22| #define IOS cin.tie(nullptr); cout.tie(nullptr);
    ios_base::sync_with_stdio(false);
23|
24| int main()
25| {
26|     IOS;
27| }

```

## 2.2 /buglist

```

1| /*
2| cmp 不能 return true
3| 變數宣告在迴圈費時，要小心使用
4| <<運算小心溢位，good way: (1LL << x)
5| prime_table小心i,j溢位
6| */

```

## 2.3 Builtin

```

1| From 日月掛長
2| unsigned int / unsigned long +1 / unsigned long long
    +1l
3| int __builtin_ffs: 返回右起第一個1的位置
4| int __builtin_clz: 返回左起第一個1之前0的個數
5| int __builtin_ctz: 返回右起第一個1之後的0的個數
6| int __builtin_popcount: 返回1的個數
7| int __builtin_parity: 返回1的個數的奇偶性(1的個數 mod
    2的值)

```

## 2.4 BinarySearch

```

1| lower_bound(a, a + n, k); //最左邊 ≥ k 的位置
2| upper_bound(a, a + n, k); //最左邊 > k 的位置
3| upper_bound(a, a + n, k) - 1; //最右邊 ≤ k 的位置
4| lower_bound(a, a + n, k) - 1; //最右邊 < k 的位置
5| [lower_bound, upper_bound) //等於 k 的範圍
6| equal_range(a, a+n, k);

```

## 2.5 int128

```

1| istream &operator>>(istream &in, __int128 &x)
2| {
3|     char buf[30];
4|     in >> buf;
5|     bool minus = false;
6|     int len = strlen(buf);
7|     x = 0;
8|     for (int i = 0; i < len; i++)
9|     {
10|         if (i == 0 && buf[i] == '-')
11|         {
12|             minus = true;
13|         }

```

```

14|         else
15|         {
16|             x = x * 10 + buf[i] - 48;
17|         }
18|     }
19|     if (minus)
20|     {
21|         x *= -1;
22|     }
23|     return in;
24| }
25|
26| ostream &operator<<(ostream &out, __int128 &x)
27| {
28|     vector<int> v;
29|     __int128 tmp = x;
30|     bool minus = tmp < 0;
31|     if (minus)
32|         tmp *= -1;
33|
34|     while (tmp > 0)
35|     {
36|         v.push_back(tmp % 10);
37|         tmp /= 10;
38|     }
39|     if (minus)
40|     {
41|         out << "-";
42|     }
43|     for (int i = (int)v.size() - 1; i >= 0; i--)
44|     {
45|         out << v[i];
46|     }
47|     return out;
48| }

```

## 2.6 StableMatching

```

1| int t, n, b[N][N], bi[N], g[N][N], bg[N], gb[N];
2|
3| void sol()
4| {
5|     deque<int> dq;
6|     memset(gb, 0, sizeof(gb));
7|     memset(bi, 0, sizeof(bi));
8|     for (int i = 1; i <= n; i++)
9|         dq.push_back(i);
10|     while (!dq.empty())
11|     {
12|         int x = dq.front();
13|         dq.pop_front();
14|         int y = b[x][++bi[x]];
15|         if (!gb[y])
16|         {
17|             gb[y] = x;
18|             bg[x] = y;
19|         }
20|         else if (g[y][x] < g[y][gb[y]])
21|         {
22|             dq.push_back(gb[y]);
23|             gb[y] = x;
24|             bg[x] = y;
25|         }
26|         else
27|         {
28|             dq.push_back(x);
29|         }
30|     }
31|     for (int i = 1; i <= n; i++)
32|     {
33|         cout << bg[i] << '\n';
34|     }
35| }
36|
37| int main()
38| {

```

```

39     int x;
40     cin >> t;
41     for (int i = 0; i < t; i++)
42     {
43         cin >> n;
44         for (int i = 1; i <= n; i++)
45         {
46             for (int j = 1; j <= n; j++)
47             {
48                 cin >> b[i][j];
49             }
50         }
51         for (int i = 1; i <= n; i++)
52         {
53             for (int j = 1; j <= n; j++)
54             {
55                 cin >> x;
56                 g[i][x] = j;
57             }
58         }
59         if (i)
60             cout << '\n';
61         sol();
62     }
63 }

```

## 2.7 Mergesort

```

1 long long sol(int L, int R)
2 {
3     if (R - L <= 1)
4         return 0;
5     int M = (R + L) / 2;
6     long long ans = sol(L, M) + sol(M, R);
7     int i = L, j = M, k = L;
8     while (i < M || j < R)
9     {
10         if (i >= M)
11             buf[k] = arr[j++];
12         else if (j >= R)
13             buf[k] = arr[i++];
14         else
15         {
16             if (arr[i] <= arr[j])
17                 buf[k] = arr[i++];
18             else
19             {
20                 buf[k] = arr[j++];
21                 ans += M - i;
22             }
23         }
24         k++;
25     }
26     for (int k = L; k < R; k++)
27         arr[k] = buf[k];
28     return ans;
29 }

```

## 2.8 Multi

```

1 multiset<int>
2 equal_range (T1 a): 回傳 iterator 的 pair<lower_bound
   (a), upper_bound (a)> , 為 a 所在範圍
3 erase (T1 a): 刪除所有元素 a , 如果只要刪除一個, 用
   s.erase (s.find (a))

```

## 2.9 ThreeSearch

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 #define N 20

```

```

4 int t, n, i, j;
5 struct happy
6 {
7     double a, b, c;
8 } h[N];
9 double f2(double x, double a, double b, double c)
10 {
11     return a * (x - b) * (x - b) + c;
12 }
13 double f(double x)
14 {
15     double ans = 0;
16     for (int i = 0; i < n; i++)
17     {
18         ans = max(ans, f2(x, h[i].a, h[i].b, h[i].c));
19         // cout<<ans<<'\n';
20     }
21     return ans;
22 }
23 int main()
24 {
25     cin.tie(NULL);
26     for (cin >> t; i < t; i++)
27     {
28         for (cin >> n, j = 0; j < n; j++)
29         {
30             cin >> h[j].a >> h[j].b >> h[j].c;
31         }
32         double L = 0, R = 300, M, MM;
33         while (R - L > 1e-9)
34         {
35             M = L + (R - L) / 3;
36             MM = (M + R) / 2;
37             if (f(M) > f(MM))
38             {
39                 L = M;
40             }
41             else
42             {
43                 R = MM;
44             }
45         }
46         cout << fixed << setprecision(5) << f(L) <<
47             '\n';
48     }
49 }

```

## 2.10 Tree Policy

```

1 #include <bits/stdc++.h>
2 #include <ext/pb_ds/assoc_container.hpp> // Common
   file
3 #include <ext/pb_ds/tree_policy.hpp>
4 #include <functional> // for less
5 using namespace std;
6 using namespace __gnu_pbds;
7 typedef tree<int, null_type, less<int>, rb_tree_tag,
   tree_order_statistics_node_update> set_t;
8 set_t t;
9 int main() {
10     t.insert(5);
11     t.insert(6);
12     t.insert(3);
13     t.insert(1);
14     // the smallest is (0), biggest is (n-1), kth
   small is (k-1)
15     int num = *t.find_by_order(0);
16     printf("%d\n", num); // print 1
17     num = *t.find_by_order(t.size()-1);
18     printf("%d\n", num); // print 6
19     // find the index
20     int index = t.order_of_key(6);
21     printf("%d\n", index); // print 3
22     // check if there exist x
23     int x = 5;
24     int check = t.erase(x);

```

```

25     if(check == 0) printf("t not contain 5\n");
26     else if(check == 1) printf("t contain 5\n");
27     //tree policy like set
28     t.insert(5); t.insert(5);
29     // get the size of t
30     printf("%d\n", t.size()); // print 4
31 }

```

## 3 Data and Structure

### 3.1 Mo

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  const int N = 100005;
4  int a[N];
5  int curmax;
6  int app[N], cnt[N];
7
8  struct Query
9  {
10     int L, R, qid, bid;
11     bool operator<(const Query &rhs) const
12     {
13         if (bid != rhs.bid)
14             return bid < rhs.bid;
15         return R < rhs.R;
16     }
17 } q[N];
18
19 bool cmp(Query a, Query b) { return a.L < b.L; }
20
21 void add(int x)
22 {
23     int now = ++app[x];
24     cnt[now - 1]--;
25     cnt[now]++;
26     curmax = max(curmax, now);
27 }
28
29 void sub(int x)
30 {
31     int now = --app[x];
32     cnt[now + 1]--;
33     cnt[now]++;
34     if (!cnt[curmax])
35         curmax--;
36 }
37
38 int main()
39 {
40     int n, Q;
41     int ans[N];
42     cin >> n >> Q;
43     for (int i = 1; i <= n; i++)
44     {
45         cin >> a[i];
46     }
47     int k = floor(sqrt(n / 1.0));
48     for (int i = 0; i < Q; i++)
49     {
50         cin >> q[i].L >> q[i].R;
51         q[i].qid = i;
52     }
53     sort(q, q + Q, cmp);
54     for (int i = 0; i < Q; i++)
55     {
56         q[i].bid = i / k;
57     }
58     sort(q, q + Q);
59     for (int i = 0, curL = 1, curR = 0; i < Q; i++)
60     {
61         while (curR < q[i].R)
62             curR++;

```

```

63             curR++;
64             add(a[curR]);
65         }
66         while (q[i].R < curR)
67         {
68             sub(a[curR]);
69             curR--;
70         }
71         while (curL < q[i].L)
72         {
73             sub(a[curL]);
74             curL++;
75         }
76         while (q[i].L < curL)
77         {
78             curL--;
79             add(a[curL]);
80         }
81         ans[q[i].qid] = curmax;
82     }
83     for (int i = 0; i < Q; i++)
84     {
85         cout << ans[i] << '\n';
86     }
87 }

```

### 3.2 Segment Tree

```

1  int built(int L, int R, int x)
2  {
3      if (L == R)
4          return heap[x - 1] = arr[L];
5      int M = (L + R) >> 1;
6      return heap[x - 1] = built(L, M, (x << 1)) +
7          built(M + 1, R, (x << 1) + 1);
8  }
9
10 void modify(int L, int R, int x, int a, int b, int mo)
11 {
12     if (b < L || R < a)
13         return;
14     if (L == R)
15     {
16         heap[x - 1] += mo;
17         return;
18     }
19     int M = (L + R) >> 1;
20     modify(L, M, (x << 1), a, b, mo);
21     modify(M + 1, R, (x << 1) + 1, a, b, mo);
22     heap[x - 1] += mo;
23     return;
24 }
25
26 int quest(int L, int R, int x, int a, int b)
27 {
28     if (b < L || R < a)
29         return 0;
30     if (a <= L && R <= b)
31         return heap[x - 1];
32     int M = (L + R) >> 1;
33     return quest(L, M, (x << 1), a, b) + quest(M + 1,
34         R, (x << 1) + 1, a, b);
35 }

```

### 3.3 Treap

```

1  struct Treap{
2      int val, pri, sz;
3      Treap *lc, *rc;
4      Treap(){
5          Treap(int _val)
6          {
7              val = _val;
8              pri = rand();
9              sz = 1;

```

```

10     lc = rc = NULL;
11 }
12 };
13
14 int getSize(Treap *a){
15     return (a == NULL ? 0 : a->sz);
16 }
17
18 void split(Treap *t, Treap *&a, Treap *&b, int k)
19 {
20     if(t == NULL)
21     {
22         a = b = NULL;
23         return;
24     }
25     if(getSize(t->lc) < k)
26     {
27         a = t;
28         split(t->rc, a->rc, b, k - getSize(t->lc) - 1);
29     }
30     else
31     {
32         b = t;
33         split(t->lc, a, b->lc, k);
34     }
35 }
36
37 Treap* merge(Treap *a, Treap *b)
38 {
39     if(!a || !b)
40     {
41         return (a ? a : b);
42     }
43     if(a->pri > b->pri)
44     {
45         a->rc = merge(a->rc, b);
46         return a;
47     }
48     else
49     {
50         b->lc = merge(a, b->lc);
51         return b;
52     }
53 }
54
55 void Insert(Treap *t, int x, int p)
56 {
57     Treap *a, *b;
58     split(t, a, b, x);
59     t = merge(a, merge(new Treap(p), b));
60 }
61
62 void Delete(Treap *t, int x)
63 {
64     Treap *a, *b, *c;
65     split(t, b, c, x);
66     split(b, a, b, x - 1);
67     t = merge(a, c);
68 }
69
70 /*
71 Usage
72 Treap *root = NULL; // declare
73 root = merge(root, new Treap(val)); // push back
74 Insert(root, x, y); // insert y after x-th element
75 Delete(root, x); // delete x-th element
76 */

```

## 4 DP

### 4.1 Backpack Limit

```
1 struct State{
```

```

2     LL w, val;
3 };
4 struct Data{
5     LL v,w,m;
6 };
7
8 int main() {
9     LL n, W;
10    cin >> n >> W;
11    vector<Data> d(n);
12    vector<LL> dp(W + 5, -INF);
13    dp[0] = 0;
14    for(LL i = 0; i < n; ++i)
15    {
16        cin >> d[i].v >> d[i].w >> d[i].m;
17    }
18    deque<State> dq[MXW];
19    for(int i = 0; i < n; ++i)
20    {
21        LL v = d[i].v, w = d[i].w, m = d[i].m;
22        for(int j = 0; j <= W; ++j)
23        {
24            if(j < w)
25            {
26                dq[j].clear();
27                dq[j].pb({j, dp[j]});
28                continue;
29            }
30            int id = j % w;
31            while(dq[id].front().w + m * w < j) dq[id].pop_front();
32            LL tmp = dq[id].front().val + (j - dq[id].front().w) / w * v;
33            while(!dq[id].empty() && dq[id].back().val + (j - dq[id].back().w) / w * v <= dp[j]) dq[id].pop_back();
34            dq[id].push_back({j, dp[j]});
35            dp[j] = max(dp[j], tmp);
36        }
37    }
38
39    LL ans = -INF;
40    for(int i = 0; i <= W; i++){
41        ans = max(ans, dp[i]);
42    }
43    cout << ans << '\n';
44 }

```

### 4.2 CounterLine

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 const int N = 1 << 15;
4 int n, m, cur;
5 long long int dp[2][N];
6
7 void update(int a, int b)
8 {
9     if (b & (1 << m))
10    {
11        dp[cur][b ^ (1 << m)] += dp[1 - cur][a];
12    }
13 }
14
15 int main()
16 {
17     while (cin >> n >> m)
18     {
19         if ((n * m) & 1)
20         {
21             cout << "0\n";
22             continue;
23         }
24         if (n == 1 || m == 1)
25         {
26             cout << "1\n";

```

```

27     continue;
28 }
29 if (n < m)
30     swap(n, m);
31 memset(dp, 0, sizeof(dp));
32 cur = 0;
33 dp[0][(1 << m) - 1] = 1;
34 for (int i = 0; i < n; i++)
35 {
36     for (int j = 0; j < m; j++)
37     {
38         cur ^= 1;
39         memset(dp[cur], 0, sizeof(dp[cur]));
40         for (int k = 0; k < (1 << m); k++)
41         {
42             update(k, k << 1);
43             if (i && !(k & (1 << m - 1)))
44                 update(k, (k << 1) ^ (1 << m)
45                     ^ 1);
46             if (j && !(k & 1))
47                 update(k, (k << 1) ^ 3);
48         }
49     }
50     cout << dp[cur][(1 << m) - 1] << '\n';
51 }
52 }

```

### 4.3 LCS

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 int main()
5 {
6     int n, m;
7     vector<int> a, b, dp[2];
8     cin >> n >> m;
9     a.resize(n);
10    b.resize(m);
11    for (int i = 0; i < a.size(); i++)
12    {
13        cin >> a[i];
14    }
15    for (int i = 0; i < b.size(); i++)
16    {
17        cin >> b[i];
18    }
19    dp[0].resize(m + 1);
20    dp[1].resize(m + 1);
21    for (int i = 1; i <= n; i++)
22    {
23        for (int j = 1; j <= m; j++)
24        {
25            if (a[i - 1] == b[j - 1])
26                dp[i & 1][j] = dp[(i & 1) ^ 1][j - 1]
27                    + 1;
28            else
29                dp[i & 1][j] = max(dp[i & 1][j - 1],
30                    dp[(i & 1) ^ 1][j]);
31        }
32    }
33    cout << dp[n & 1][m] << '\n';
34 }

```

### 4.4 LIS

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 int main()
5 {
6     int n;

```

```

7     while (cin >> n)
8     {
9         vector<int> v;
10        for (int i = 0, x; i < n; i++)
11        {
12            cin >> x;
13            if (!v.size() || x > v.back())
14                v.push_back(x);
15            else
16                *lower_bound(v.begin(), v.end(), x) =
17                    x;
18        }
19        cout << v.size() << '\n';
20    }

```

### 4.5 ReRoot

```

1 LL dp[MXV], num[MXV], aa[MXV], sum = 0;
2 vector<LL> p[MXV];
3 bitset<MXV> vis;
4
5 void dfs(int s, LL depth)
6 {
7     vis[s] = 1;
8     num[s] = aa[s];
9     dp[1] += depth * aa[s];
10    for (int v : p[s])
11    {
12        if (!vis[v])
13        {
14            dfs(v, depth + 1);
15            num[s] += num[v];
16        }
17    }
18 }
19
20 void solve(int s, int n)
21 {
22     vis[s] = 1;
23     for (int v : p[s])
24     {
25         if (!vis[v])
26         {
27             dp[v] = dp[s] + sum - num[v] * 2;
28             solve(v, n);
29         }
30     }
31 }

```

### 4.6 TSP

```

1 void btb(int &x)
2 {
3     x = 0;
4     for (int i = 0, j = 1; i < n; i++, j *= 2)
5         x += b[i] * j;
6     return;
7 }
8 int main()
9 {
10    memset(dp, 0, sizeof(dp));
11    for (int i = 1, st; i <= n; i++)
12    { // st:state
13        for (int jj = 0; jj < n; jj++)
14            b[n - jj - 1] = (jj < i);
15        do
16        {
17            btb(st);
18            for (int x = 0; x < n; x++)
19            {
20                if (!b[x])
21                    continue;

```

```

22         if (i == 1)
23             dp[x][st] = dis[x][0];
24         for (int y = 0; y < n; y++)
25         {
26             if (x != y && b[y] &&
27                 (dp[x][st] == 0 ||
28                  dp[x][st] > dp[y][st - (1 <<
29                      x)] + dis[y][x]))
30             {
31                 dp[x][st] = dp[y][st - (1 <<
32                     x)] + dis[y][x];
33             }
34         } while (next_permutation(b, b + n));
35     }
36     cout << dp[0][(1 << n) - 1] << '\n';
37 }

```

## 5 Geometry

### 5.1 Basic

```

1 struct dot
2 {
3     dvt x, y;
4 };
5 struct Line
6 {
7     dot st, ed;
8 };
9
10 dot operator+(dot a, dot b) { return {a.x + b.x, a.y
11     + b.y}; }
12 dot operator-(dot a, dot b) { return {a.x - b.x, a.y
13     - b.y}; }
14 dot operator*(dot a, dvt c) { return {a.x * c, a.y *
15     c}; }
16 dot operator*(dvt c, dot a) { return a * c; }
17 dot operator/(dot a, dvt c) { return {a.x / c, a.y /
18     c}; }
19
20 bool operator<(dot a, dot b) { return std::tie(a.x,
21     a.y) < std::tie(b.x, b.y); }
22 bool operator==(dot a, dot b)
23 {
24     return std::tie(a.x, a.y) == std::tie(b.x, b.y);
25 }
26
27 dvt iproduct(dot a, dot b) { return a.x * b.x + a.y *
28     b.y; }
29 dvt cross(dot a, dot b) { return a.x * b.y - a.y *
30     b.x; }
31 int dis(dot a, dot b)
32 {
33     return (a.x - b.x) * (a.x - b.x) + (a.y - b.y) *
34         (a.y - b.y);
35 }
36
37 int side(Line L, dot a)
38 {
39     dvt cross_value = cross(a - L.st, L.ed - L.st);
40     if (cross_value > EPS)
41     {
42         return 1;
43     }
44     else if (cross_value < -EPS)
45     {
46         return -1;
47     }
48     return 0;
49 }
50
51 bool has_jiao(Line AB, Line CD)
52 {
53     int a = side(CD, AB.st);

```

```

44     int b = side(CD, AB.ed);
45     int c = side(AB, CD.st);
46     int d = side(AB, CD.ed);
47     if (a * b < 0 && c * d < 0)
48     {
49         return true;
50     }
51     if (a == 0 && iproduct(CD.st - AB.st, CD.ed -
52         AB.st) <= 0)
53     {
54         return true;
55     }
56     if (b == 0 && iproduct(CD.st - AB.ed, CD.ed -
57         AB.ed) <= 0)
58     {
59         return true;
60     }
61     if (c == 0 && iproduct(AB.st - CD.st, AB.ed -
62         CD.st) <= 0)
63     {
64         return true;
65     }
66     if (d == 0 && iproduct(AB.st - CD.ed, AB.ed -
67         CD.ed) <= 0)
68     {
69         return true;
70     }
71     return false;
72 }

```

### 5.2 Convex Hull

```

1 vector<dot> p, p1;
2
3 void convexhull()
4 {
5     sort(p.begin(), p.end());
6     p.erase(unique(p.begin(), p.end()), p.end());
7     p1.clear();
8     p1.resize(p.size());
9     int m = 0;
10    FOR(i, 0, p.size())
11    {
12        while (m > 1 && cross(p1[m - 1] - p1[m - 2],
13            p[i] - p1[m - 2]) <= 0)
14            m--;
15        p1[m++] = p[i];
16    }
17    int k = m;
18    FORD(i, p.size() - 2, 0 - 1)
19    {
20        while (m > k && cross(p1[m - 1] - p1[m - 2],
21            p[i] - p1[m - 2]) <= 0)
22            m--;
23        p1[m++] = p[i];
24    }
25    if (m > 1)
26        m--;
27    p1.resize(m);
28 }

```

## 6 Graph

### 6.1 Edge

```

1 struct Edge
2 {
3     int from, to, w;
4     bool operator<(const Edge& rhs) // optional
5     {
6         return w < rhs.w;
7     }
8 };

```

## 6.2 Blossom

```

1 int lca(int x, int y)
2 {
3     MSET(vis, false);
4     while (true)
5     {
6         x = base[x];
7         vis[x] = true;
8         if (match[x] == -1)
9             break;
10        x = fr[match[x]];
11    }
12    while (true)
13    {
14        y = base[y];
15        if (vis[y])
16            return y;
17        y = fr[match[y]];
18    }
19    return -1;
20 }
21 void set_path(int x, int fa)
22 {
23     int y;
24     while (x != fa)
25     {
26         y = match[x];
27         blossom[base[x]] = true;
28         blossom[base[y]] = true;
29         y = fr[y];
30         if (base[y] != fa)
31             fr[y] = match[x];
32         x = y;
33     }
34 }
35 void flower(int x, int y)
36 {
37     MSET(blossom, false);
38     int fa = lca(x, y);
39     set_path(x, fa);
40     set_path(y, fa);
41     if (base[x] != fa)
42         fr[x] = y;
43     if (base[y] != fa)
44         fr[y] = x;
45     REP(i, 1, n)
46     if (blossom[base[i]])
47     {
48         base[i] = fa;
49         if (!inq[i])
50         {
51             q.push(i);
52             inq[i] = true;
53         }
54     }
55 }
56 bool bfs(int root)
57 {
58     int cur, y, nxt;
59     q = queue<int>();
60     MSET(inq, false);
61     MSET(fr, -1);
62     REP(i, 1, n) base[i] = i;
63     q.push(root);
64     while (!q.empty())
65     {
66         cur = q.front();
67         q.pop();
68         inq[cur] = false;
69         for (int i = first[cur]; ~i; i = in[i].next)
70             if (base[cur] != base[in[i].t] &&
71                 match[cur] != in[i].t)
72                 if (in[i].t == root ||
73                     (~match[in[i].t] &&
74                     ~fr[match[in[i].t]]))

```

```

73         flower(cur, in[i].t);
74     else if (fr[in[i].t] == -1)
75     {
76         fr[in[i].t] = cur;
77         if (match[in[i].t] == -1)
78         {
79             cur = in[i].t;
80             while (cur != -1)
81             {
82                 y = fr[cur];
83                 nxt = match[y];
84                 match[cur] = y;
85                 match[y] = cur;
86                 cur = nxt;
87             }
88             return true;
89         }
90     else
91     {
92         q.push(match[in[i].t]);
93         inq[match[in[i].t]] = true;
94     }
95     }
96 }
97 }
98 return false;
99 }
100 int do_match()
101 {
102     int re = 0;
103     MSET(match, -1);
104     REP(i, 1, n) if (match[i] == -1 && bfs(i)) re++;
105     return re;
106 }

```

## 6.3 CLE

```

1 struct Edge {
2     int from;
3     int to;
4     int bdw;
5     int cost;
6 };
7
8 int n, m, budget;
9 int in[MAXN], pre[MAXN], id[MAXN], vis[MAXN];
10 Edge edges[MAXN], tedges[MAXN];
11
12 int CLE(int root, int tn, int lowb) {
13     copy(begin(edges), begin(edges) + m,
14         begin(tedges));
15
16     int res = 0;
17     while (true) {
18         for (int i = 0; i < tn; i++) {
19             in[i] = INF;
20         }
21
22         //find in edge
23         for (int i = 0; i < m; i++) {
24             Edge e = tedges[i];
25             if (e.from != e.to && e.bdw >= lowb &&
26                 e.cost < in[e.to]) {
27                 pre[e.to] = e.from;
28                 in[e.to] = e.cost;
29             }
30         }
31
32         //check in edge
33         for (int i = 0; i < tn; i++) {
34             if (i == root) {
35                 continue;
36             }
37             if (in[i] == INF) {
38                 return -1;
39             }

```



```

38     }
39
40     int nodenum = 0;
41     memset(id, -1, sizeof(id));
42     memset(vis, -1, sizeof(vis));
43     in[root] = 0;
44
45     //find cycles
46     for (int i = 0; i < tn; i++) {
47         res += in[i];
48         int v = i;
49         while (vis[v] != i && id[v] == -1 && v !=
50             root) {
51             vis[v] = i;
52             v = pre[v];
53         }
54
55         if (v != root && id[v] == -1) {
56             for (int j = pre[v]; j != v; j =
57                 pre[j]) {
58                 id[j] = nodenum;
59             }
60             id[v] = nodenum++;
61         }
62
63         //no cycle
64         if (nodenum == 0) {
65             break;
66         }
67
68         for (int i = 0; i < tn; i++) {
69             if (id[i] == -1) {
70                 id[i] = nodenum++;
71             }
72         }
73
74         //grouping the vertices
75         for (int i = 0; i < m; i++) {
76             int from = tedges[i].from;
77             int to = tedges[i].to;
78
79             tedges[i].from = id[from];
80             tedges[i].to = id[to];
81
82             if (tedges[i].from != tedges[i].to) {
83                 tedges[i].cost += in[to];
84             }
85         }
86
87         tn = nodenum;
88         root = id[root];
89     }
90     return res;

```

## 6.4 Disjoint Set

```

1 struct DisjointSet
2 {
3     int p[MXV], sz[MXV];
4     void init(int n)
5     {
6         for (int i = 0; i <= n; i++)
7         {
8             p[i] = i;
9             sz[i] = 1;
10        }
11    }
12    int find(int u) { return u == p[u] ? u : p[u] =
13        find(p[u]); }
14    void Union(int u, int v)
15    {
16        u = find(u);
17        v = find(v);
18        if (u == v)

```

```

18        {
19            return;
20        }
21        if (sz[u] < sz[v])
22        {
23            swap(u, v);
24        }
25        sz[u] += sz[v];
26        p[v] = u;
27    }
28 };
29
30 /*
31 Usage
32 DisjointSet djs; // declare
33 djs.init(int n); // initialize from vertex 0 to
34 vertex n
35 djs.find(int u) // find the parent of vertex u
36 djs.Union(int u, int v) // union vertex u and v
37 */

```

## 6.5 Longest Common Ancestor

```

1 const int LOG = 20;
2 vector<int> tin(MXV), tout(MXV), depth(MXV);
3 int par[MXV][LOG];
4 int timer = 0;
5 vector<int> G[MXV];
6
7 void dfs(int u, int f)
8 {
9     tin[u] = ++timer;
10    par[u][0] = f;
11    for (int v : G[u])
12    {
13        if (v != f)
14        {
15            depth[v] = depth[u] + 1;
16            dfs(v, u);
17        }
18    }
19    tout[u] = ++timer;
20 }
21
22 void Doubling(int n)
23 {
24     for (int j = 1; j < LOG; ++j)
25     {
26         for (int i = 1; i <= n; ++i)
27         {
28             par[i][j] = par[par[i][j - 1]][j - 1];
29         }
30     }
31 }
32
33 bool anc(int u, int v) { return tin[u] <= tin[v] &&
34     tout[v] <= tout[u]; }
35
36 int LCA(int u, int v)
37 {
38     if (depth[u] > depth[v])
39     {
40         swap(u, v);
41     }
42     if (anc(u, v))
43     {
44         return u;
45     }
46     for (int j = LOG - 1; j >= 0; --j)
47     {
48         if (!anc(par[u][j], v))
49             u = par[u][j];
50     }
51     return par[u][0];
52 }

```

```

53 int dis(int u, int v)
54 {
55     int lca = LCA(u, v);
56     return depth[u] + depth[v] - 2 * depth[lca];
57 }
58
59 /*
60 dfs(root, root);
61 Doubling(n);
62 */

```

## 6.6 MST

```

1 int MST()
2 {
3     DisjointSet djs;
4     vector<Edge> edges;
5     int n, m, ans = 0;
6     cin >> n >> m;
7     for (int i = 0, from, to, w; i < m; i++)
8     {
9         cin >> from >> to >> w;
10        edges.push_back({from, to, w});
11    }
12    sort(edges.begin(), edges.end());
13    djs.init(n);
14    for (auto edge : edges)
15    {
16        // Union also check if to vertex haven't
17        // connected
18        if (djs.Union(edge.from, edge.to))
19        {
20            ans += edge.w;
21        }
22    }
23    return ans;
24 }

```

## 6.7 TopologicalSort

```

1 #include <cstring>
2 #include <iostream>
3 #include <stack>
4 #include <vector>
5
6 #define S 50050
7
8 using namespace std;
9
10 vector<int> map[S];
11 stack<int> ans;
12 int state[S];
13 bool head[S];
14 bool valid;
15 int n, m;
16
17 void dfs(int cur)
18 {
19     state[cur] = 1;
20
21     for (auto next : map[cur])
22     {
23         if (!state[next])
24             dfs(next);
25         else if (state[next] == 1)
26         {
27             valid = false;
28             return;
29         }
30     }
31
32     state[cur] = 2;
33     ans.push(cur);
34 }

```

```

34
35 void topology_sort()
36 {
37     for (int i = 1; i <= n; i++)
38         if (valid && head[i])
39             dfs(i);
40
41     if (!valid)
42     {
43         cout << -1 << endl;
44         return;
45     }
46
47     while (!ans.empty())
48     {
49         cout << ans.top() << endl;
50         ans.pop();
51     }
52 }
53
54 int main()
55 {
56     cin >> n >> m;
57
58     memset(head, true, sizeof(head));
59
60     for (int i = 0; i < m; i++)
61     {
62         int a, b;
63         cin >> a >> b;
64
65         head[b] = false;
66
67         map[a].push_back(b);
68     }
69
70     memset(state, 0, sizeof(state));
71     valid = true;
72
73     topology_sort();
74
75     return 0;
76 }

```

## 6.8 TreeCentroid

```

1 PII treeCentroid(int u, int f, int sz)
2 {
3     // cout << u << ' ' << f << ' ' << sz << '\n';
4     treeSz[u] = 1;
5     PII res(__INT_MAX__, -1);
6     int mx = 0;
7     for (size_t i = 0; i != G[u].size(); ++i)
8     {
9         PII &e = G[u][i];
10        int v = e.first;
11        if (v == f || vis[v] == true)
12        {
13            continue;
14        }
15        res = min(res, treeCentroid(v, u, sz));
16        // cout << u << ' ' << res.first << ' ' <<
17        // res.second << '\n';
18        treeSz[u] += treeSz[v];
19        mx = max(mx, treeSz[v]);
20    }
21    mx = max(mx, sz - treeSz[u]);
22    // cout << u << ':' << mx << ' ' << u << '\n';
23    return min(res, {mx, u});
24 }

```

## 7 Graph Bipartite

### 7.1 Bipartite

```

1 #include <cstring>
2 #include <iostream>
3 #include <stack>
4 #include <vector>
5
6 #define S 50050
7
8 using namespace std;
9
10 vector<int> map[S];
11 int visit[S];
12 bool valid;
13
14 void check(int start)
15 {
16     stack<int> st;
17     st.push(start);
18     visit[start] = 1;
19
20     while (valid && !st.empty())
21     {
22         int cur = st.top();
23         st.pop();
24
25         for (int i = 0; i < map[cur].size(); i++)
26         {
27             int next = map[cur][i];
28
29             if (visit[next] == -1)
30             {
31                 st.push(next);
32
33                 if (visit[cur] == 1)
34                     visit[next] = 2;
35                 else
36                     visit[next] = 1;
37             }
38             else if (visit[cur] == visit[next])
39                 valid = false;
40         }
41     }
42 }
43
44 int main()
45 {
46     int n, m;
47     cin >> n >> m;
48
49     for (int i = 0; i < m; i++)
50     {
51         int a, b;
52         cin >> a >> b;
53
54         map[a].push_back(b);
55         map[b].push_back(a);
56     }
57
58     // -1 : not visit, 1 : tsudere, 2 : proud
59     memset(visit, -1, sizeof(visit));
60     valid = true;
61
62     for (int i = 1; i <= n; i++)
63     {
64         if (valid && visit[i] == -1)
65         {
66             check(i);
67         }
68     }
69
70     if (valid)
71         cout << "yes" << endl;
72     else

```

```

73         cout << "no" << endl;
74
75     return 0;
76 }

```

### 7.2 BipartiteMatch

```

1 int lhs, rhs, Left[MXV], G[MXV][MXV];
2 bitset<MXV> used;
3
4 bool dfs(int s)
5 {
6     for (int i = 1; i <= rhs; i++)
7     {
8         if (!G[s][i] || used[i])
9             continue;
10
11         used[i] = true;
12         if (Left[i] == -1 || dfs(Left[i]))
13         {
14             Left[i] = s;
15             return true;
16         }
17     }
18     return false;
19 }
20
21 int sol()
22 {
23     int ret = 0;
24     memset(Left, -1, sizeof(Left));
25     for (int i = 1; i <= lhs; i++)
26     {
27         used.reset();
28         if (dfs(i))
29             ret++;
30     }
31     return ret;
32 }
33
34 }
35

```

### 7.3 KM

```

1 template <typename T>
2 struct KM
3 {
4     int n;
5     int Left[N];
6     T w[N][N], Lx[N], Ly[N];
7     bitset<N> vx, vy;
8
9     void init(int _n)
10     {
11         n = _n;
12     }
13
14     bool match(int i)
15     {
16         vx[i] = true;
17         for (int j = 1; j <= n; j++)
18         {
19             if ((fabs(Lx[i] + Ly[j] - w[i][j]) <
20                 1e-9) && !vy[j])
21             {
22                 vy[j] = 1;
23                 if (!Left[j] || match(Left[j]))
24                 {
25                     Left[j] = i;
26                     return true;
27                 }
28             }
29         }
30     }
31 }

```

```

28     }
29     return false;
30 }
31
32 void update()
33 {
34     T a = 1e9;
35     for (int i = 1; i <= n; i++)
36     {
37         if (vx[i])
38         {
39             for (int j = 1; j <= n; j++)
40             {
41                 if (!vy[j])
42                 {
43                     a = min(a, Lx[i] + Ly[j] -
44                             w[i][j]);
45                 }
46             }
47         }
48         for (int i = 1; i <= n; i++)
49         {
50             if (vx[i])
51             {
52                 Lx[i] -= a;
53             }
54             if (vy[i])
55             {
56                 Ly[i] += a;
57             }
58         }
59     }
60
61 void hungarian()
62 {
63     for (int i = 1; i <= n; i++)
64     {
65         Left[i] = Lx[i] = Ly[i] = 0;
66         for (int j = 1; j <= n; j++)
67         {
68             Lx[i] = max(Lx[i], w[i][j]);
69         }
70     }
71     for (int i = 1; i <= n; i++)
72     {
73         while (1)
74         {
75             vx.reset();
76             vy.reset();
77             if (match(i))
78             {
79                 break;
80             }
81             update();
82         }
83     }
84 }
85 };
86
87 /*
88 usage
89 KM<int> km; // declare with weight type
90 km.init(n); // initialize with vertex
91 km.hungarian(); // calculate
92 km.w[][]; // weight array
93 km.Left[i] // y_i match x_Left[i]
94 */

```

## 7.4 Relation

```

1 | 1. 一般圖
2 | 最大匹配|+| 最小邊覆蓋|=|V|
3 | 最大獨立集|+| 最小點覆蓋|=|V|
4 | 最大圖|=| 補圖的最大獨立集|

```

```

5 | 2. 二分圖
6 | 最大匹配|=| 最小點覆蓋|
7 | 最大獨立集|=| 最小邊覆蓋|
8 | 最大獨立集|=|V|-| 最大匹配|
9 | 最大圖|=| 補圖的最大獨立集|

```

## 8 Graph Connectivity

### 8.1 decide

```

1 | 點雙連通
2 | 非根節點：low[i] >= depth[now]
3 | 根節點：如果子節點 >1，該點就是割點
4
5 | 邊雙連通：low[i] > depth[now]

```

### 8.2 low

```

1 | bitset<MXV> is_cut_vertex, visit;
2 | vector<int> G[MXV], low(MXV), depth(MXV);
3
4 | void dfs(int now, int cur_depth)
5 | {
6 |     visit[now] = true;
7 |     depth[now] = low[now] = cur_depth;
8 |     for (auto i : G[now])
9 |     {
10 |         if (visit[i])
11 |             { // ancestor
12 |                 low[now] = min(low[now], depth[i]);
13 |             }
14 |         else
15 |             { // offspring
16 |                 dfs(i, cur_depth + 1);
17 |                 low[now] = min(low[now], low[i]);
18 |             }
19 |     }
20 |     return;
21 | }

```

## 9 Graph Flow

### 9.1 Dinic

```

1 | template <typename T>
2 | struct Dinic
3 | {
4 |     int n, s, t, level[M], now[M];
5 |     struct Edge
6 |     {
7 |         int v;
8 |         T rf; // rf: residual flow
9 |         int re;
10 |     };
11 |     vector<Edge> e[M];
12 |     void init(int _n, int _s, int _t)
13 |     {
14 |         n = _n;
15 |         s = _s;
16 |         t = _t;
17 |         for (int i = 0; i <= n; i++)
18 |         {
19 |             e[i].clear();
20 |         }
21 |     }
22 |     void add_edge(int u, int v, T f)
23 |     {
24 |         e[u].push_back({v, f, (int)e[v].size()});

```

```

25     e[v].push_back({u, f, (int)e[u].size() - 1});
26     // for directional graph
27     // e[v].push_back({u, 0, (int)e[u].size() - 1});
28 }
29 bool bfs()
30 {
31     fill(level, level + n + 1, -1);
32     queue<int> q;
33     q.push(s);
34     level[s] = 0;
35     while (!q.empty())
36     {
37         int u = q.front();
38         q.pop();
39         for (auto it : e[u])
40         {
41             if (it.rf > 0 && level[it.v] == -1)
42             {
43                 level[it.v] = level[u] + 1;
44                 q.push(it.v);
45             }
46         }
47     }
48     return level[t] != -1;
49 }
50 T dfs(int u, T limit)
51 {
52     if (u == t)
53         return limit;
54     T res = 0;
55     while (now[u] < (int)e[u].size())
56     {
57         Edge &it = e[u][now[u]];
58         if (it.rf > 0 && level[it.v] == level[u] + 1)
59         {
60             T f = dfs(it.v, min(limit, it.rf));
61             res += f;
62             limit -= f;
63             it.rf -= f;
64             e[it.v][it.re].rf += f;
65             if (limit == 0)
66             {
67                 return res;
68             }
69         }
70         else
71         {
72             ++now[u];
73         }
74     }
75     if (!res)
76     {
77         level[u] = -1;
78     }
79     return res;
80 }
81 T flow(T res = 0)
82 {
83     while (bfs())
84     {
85         T tmp;
86         memset(now, 0, sizeof(now));
87         do{
88             tmp = dfs(s, INF);
89             res += tmp;
90         }while(tmp);
91     }
92     return res;
93 }
94 };
95
96 /*
97 usage
98 Dinic<int> dinic; // declare, flow type is int

```

```

99 dinic.init(n, s, t); // initialize, n vertexs, start
    from s to t
100 dinic.add_edge(x, y, z); // add edge from x to y,
    weight is z
101 dinic.flow() // calculate max flow
102 */

```

## 9.2 MCMF

```

1 struct Edge
2 {
3     int v;
4     T cost;
5     int cap;
6     Edge(int _v, int _cost, int _cap) : v(_v),
    cost(_cost), cap(_cap) {}
7 };
8 vector<int> dis(MXV), pre(MXV);
9 vector<vector<int>>> G(MXV);
10 int n;
11 vector<Edge> edges;
12 bitset<MXV> inque;
13 queue<int> q;
14 void init(int _n)
15 {
16     n = _n;
17     edges.clear();
18     for (int i = 0; i <= MXV; ++i)
19     {
20         G[i].clear();
21     }
22 }
23 void addEdge(int u, int v, T cost, int cap)
24 {
25     G[u].push_back((int)edges.size());
26     edges.push_back(Edge(v, cost, cap));
27     G[v].push_back((int)edges.size());
28     edges.push_back(Edge(u, -cost, 0));
29 }
30 bool spfa(int s, int t)
31 {
32     FOR(i, 0, MXV) { dis[i] = INF; }
33     inque.reset();
34     while (!q.empty())
35     {
36         q.pop();
37     }
38     dis[s] = 0;
39     q.push(s);
40     while (!q.empty())
41     {
42         int u = q.front();
43         q.pop();
44         inque[u] = false;
45         FOR(i, 0, G[u].size())
46         {
47             Edge &e = edges[G[u][i]];
48             if (e.cap > 0 && dis[e.v] > dis[u] +
    e.cost)
49             {
50                 dis[e.v] = dis[u] + e.cost;
51                 pre[e.v] = G[u][i];
52                 if (!inque[e.v])
53                 {
54                     q.push(e.v);
55                     inque[e.v] = true;
56                 }
57             }
58         }
59     }
60     return dis[t] != INF;
61 }
62 void update(int s, int t, int bottleneck)
63 {
64     for (int u = t; u != s; )
65     {

```

```

66     int pos = pre[u];
67     edges[pos].cap -= bottleneck;
68     edges[pos ^ 1].cap += bottleneck;
69     u = edges[pos ^ 1].v;
70 }
71 }
72 void sol(int s, int t)
73 {
74     int mnCost = 0;
75     while (spfa(s, t))
76     {
77         update(s, t, 1);
78         mnCost += dis[t];
79     }
80     cout << mnCost << '\n';
81 }
82
83 int main()
84 {
85     IOS;
86     int n, m;
87     while (cin >> n >> m)
88     {
89         init(n);
90         for (int i = 0, f, t, w; i != m; ++i)
91         {
92             cin >> f >> t >> w;
93             addEdge(f, t, w, 1);
94             addEdge(t, f, w, 1);
95         }
96         int s = 0, t = n + 1;
97         addEdge(s, 1, 0, 2);
98         addEdge(n, t, 0, 2);
99         sol(s, t);
100     }
101 }

```

## 10 Graph Shortest Path

### 10.1 BellmanFord

```

1 struct Edge
2 {
3     int t, w;
4 };
5 int v, e;
6 int d[N], cnt[N];
7 bitset<N> inq;
8 queue<int> Q;
9 vector<Edge> G[N];
10
11 void addEdge(int from, int to, int w) {
12     G[from].push_back({to, w}); }
13 bool hasnegativeCycle()
14 {
15     while (!Q.empty())
16         Q.pop();
17     for (int i = 1; i <= v; i++)
18     {
19         inq[i] = true;
20         cnt[i] = d[i] = 0;
21         Q.push(i);
22     }
23     while (!Q.empty())
24     {
25         int s = Q.front();
26         Q.pop();
27         inq[s] = false;
28         for (Edge it : G[s])
29         {
30             if (d[it.t] > d[s] + it.w)
31             {
32                 d[it.t] = d[s] + it.w;

```

```

33             if (inq[it.t])
34                 continue;
35             Q.push(it.t);
36             inq[it.t] = true;
37             if (++cnt[it.t] > v)
38                 return true;
39         }
40     }
41 }
42 return false;
43 }

```

### 10.2 Dijkstra

```

1 struct Dijkstra
2 {
3     const int INF = 1000000000;
4     int d[MXV], p[MXV];
5     vector<Edge> E;
6     vector<int> v[MXV];
7     bitset<MXV> vis;
8
9     void init()
10     {
11         fill(d, d + MXV, INF);
12         memset(p, 0, sizeof(p));
13         E.clear();
14         for (int i = 0; i < MXV; i++)
15         {
16             v[i].clear();
17         }
18         vis.reset();
19     }
20
21     void addEdge(int from, int to, int w)
22     {
23         v[from].push_back(E.size());
24         E.push_back(Edge{from, to, w});
25     }
26
27     void dijkstra(int s)
28     {
29         d[s] = 0;
30         priority_queue<PII, vector<PII>,
31             greater<PII>> states;
32         vis.reset();
33         states.push(MP(d[s], s));
34         while (!states.empty())
35         {
36             PII state = states.top();
37             states.pop();
38             if (vis[state.second])
39                 continue;
40             vis[state.second] = true;
41             for (int u : v[state.second])
42             {
43                 Edge e = E[u];
44                 if (d[e.to] > d[e.from] + e.w)
45                 {
46                     d[e.to] = d[e.from] + e.w;
47                     p[e.to] = e.from;
48                     states.push(MP(d[e.to], e.to));
49                 }
50             }
51         }
52     }
53 };
54
55 /*
56 Usage
57 Dijkstra dijkstra; // declare
58 dijkstra.init();
59 dijkstra.addEdge(int from, int to, int w); // add a
60     directional Edge

```

```

61 | dijkstra.dijkstra(int s) // calculation shortest
62 | distance from s
   | */

```

### 10.3 FloydWarshall

```

1 | template <typename T> struct FloydWarshall
2 | {
3 |     T d[MXV][MXV];
4 |     void init() { memset(d, 0x3f, sizeof(d)); }
5 |     void floydWarshall(int n)
6 |     {
7 |         for (int k = 1; k <= n; ++k)
8 |         {
9 |             for (int i = 1; i <= n; ++i)
10 |            {
11 |                for (int j = 1; j <= n; ++j)
12 |                {
13 |                    d[i][j] = d[j][i] = min(d[i][j],
14 |                        d[i][k] + d[k][j]);
15 |                }
16 |            }
17 |        }
18 |    };
19 |
20 |    /*
21 |    usage
22 |    FloydWarshall<int> floydWarshall; // declare with
23 |    distance's type
24 |    floydWarshall.init(); // initialize
25 |    FloydWarshall.floydWarshall(); // calculate all-pair
26 |    shortest path
27 |    */

```

### 10.4 SPFA

```

1 | struct Edge
2 | {
3 |     int at;
4 |     long long cost;
5 | };
6 | int n;
7 | long long dis[MXN];
8 | vector<Edge> G[MXN];
9 | void init()
10 | {
11 |     for (int i = 0; i < n; i++)
12 |     {
13 |         G[i].clear();
14 |         dis[i] = INF;
15 |     }
16 | }
17 | bool SPFA(int st)
18 | {
19 |     vector<int> cnt(n, 0);
20 |     vector<bool> inq(n, false);
21 |     queue<int> q;
22 |
23 |     q.push(st);
24 |     dis[st] = 0;
25 |     inq[st] = true;
26 |     while (!q.empty())
27 |     {
28 |         int now = q.front();
29 |         q.pop();
30 |         inq[now] = false;
31 |         for (auto &e : G[now])
32 |         {
33 |             if (dis[e.at] > dis[now] + e.cost)
34 |             {
35 |                 dis[e.at] = dis[now] + e.cost;
36 |                 if (!inq[e.at])

```

```

37 |             {
38 |                 cnt[e.at]++;
39 |                 if (cnt[e.at] > n)
40 |                 {
41 |                     // negative cycle
42 |                     return false;
43 |                 }
44 |                 inq[e.at] = true;
45 |                 q.push(e.at);
46 |             }
47 |         }
48 |     }
49 |     return true;
50 | }
51 |

```

## 11 Math

### 11.1 Catalan

$$C_0 = 1 \quad \text{and} \quad C_{n+1} = \frac{2(2n+1)}{n+2} C_n,$$

### 11.2 Combination

```

1 | #include <bits/stdc++.h>
2 | using namespace std;
3 | typedef long long LL;
4 | const int M = 1000005;
5 | int n, k;
6 | LL m, phi;
7 | vector<int> facs;
8 | LL dp[M], dp2[M][32];
9 | LL pw(LL x, LL y)
10 | {
11 |     LL ret = 1, tmp = x % m;
12 |     while (y)
13 |     {
14 |         if (y & 1)
15 |         {
16 |             ret = ret * tmp % m;
17 |         }
18 |         tmp = tmp * tmp % m;
19 |         y >>= 1;
20 |     }
21 |     return ret;
22 | }
23 | void init()
24 | {
25 |     facs.clear();
26 |     LL x = m, sq = (LL)sqrt(m);
27 |     phi = 1;
28 |     for (LL i = 2; i <= sq; i++)
29 |     {
30 |         if (x % i)
31 |             continue;
32 |         phi *= i - 1;
33 |         x /= i;
34 |         facs.push_back(i);
35 |         while (x % i == 0)
36 |         {
37 |             phi *= i;
38 |             x /= i;
39 |         }
40 |     }
41 |     if (x > 1)
42 |     {
43 |         phi *= x - 1;
44 |         facs.push_back((int)x);
45 |     }
46 |     k = facs.size();
47 |     dp[0] = 1;
48 |     memset(dp2, 0, sizeof(dp2));
49 |     for (int i = 1; i < M; i++)

```

```

50 {
51     LL tmp = i;
52     for (int j = 0; j < k; j++)
53     {
54         dp2[i][j] = dp2[i - 1][j];
55         while (tmp % facs[j] == 0)
56         {
57             tmp /= facs[j];
58             dp2[i][j]++;
59         }
60     }
61     dp[i] = dp[i - 1] * tmp % m;
62 }
63 return;
64 }
65 int main()
66 {
67     while (cin >> n >> m)
68     {
69         init();
70         while (n--)
71         {
72             LL ans = 1;
73             int x, y;
74             cin >> x >> y;
75             for (int i = 0; i < k; i++)
76             {
77                 ans = ans * pw(facs[i], dp2[x][i] -
78                               dp2[x - y][i] - dp2[y][i]) %
79                     m;
80             }
81             ans = ans * dp[x] % m;
82             ans = ans * pw(dp[y], phi - 1) % m;
83             ans = ans * pw(dp[x - y], phi - 1) % m;
84             cout << ans << '\n';
85         }
86     }

```

### 11.3 Extend Euclidean.cpp

```

1 int extgcd(int a, int b, int &x, int &y)
2 {
3     int d = a;
4     if (b)
5     {
6         d = extgcd(b, a % b, y, x), y -= (a / b) * x;
7     }
8     else
9         x = 1, y = 0;
10    return d;
11 } // ax+by=1 ax同餘 1 mod b

```

### 11.4 FFT

```

1 struct Complex
2 {
3     LD r, i;
4     Complex(LD _r = 0.0, LD _i = 0.0)
5     {
6         r = _r;
7         i = _i;
8     }
9     Complex operator+(Complex rhs) { return Complex(r
10 + rhs.r, i + rhs.i); }
11     Complex operator-(Complex rhs) { return Complex(r
12 - rhs.r, i - rhs.i); }
13     Complex operator*(Complex rhs)
14     {
15         return Complex(r * rhs.r - i * rhs.i, r *
16             rhs.i + i * rhs.r);
17     }
18 };

```

```

16
17 template <typename T> struct FFT
18 {
19     void fft(vector<Complex> &a, int n, int inv)
20     {
21         for (int i = 1, j = 0; i < n; ++i)
22         {
23             for (int k = (n >> 1); (j ^= k) < k; k
24                 >>= 1)
25                 ;
26             if (i > j)
27                 swap(a[i], a[j]);
28         }
29     }
30     for (int m = 2; m <= n; m <= 1)
31     {
32         Complex wm(cos(2 * PI * inv / m), sin(2 *
33             PI * inv / m));
34         for (int k = 0; k < n; k += m)
35         {
36             Complex w(1.0, 0.0);
37             for (int j = 0; j < (m >> 1); ++j, w
38                 = w * wm)
39             {
40                 Complex u = a[k + j], t = w * a[k
41                     + j + (m >> 1)];
42                 a[k + j] = u + t;
43                 a[k + j + (m >> 1)] = u - t;
44             }
45         }
46         if (inv == -1)
47         {
48             FOR(i, 0, n)
49             {
50                 a[i].r /= n;
51                 a[i].i /= n;
52             }
53         }
54     }
55     void convolution(vector<Complex> &A,
56         vector<Complex> &B, vector<Complex> &C,
57         vector<Complex> &D, int n,
58         vector<Complex> &ans)
59     {
60         fft(A, n, 1);
61         fft(B, n, 1);
62         fft(C, n, 1);
63         fft(D, n, 1);
64         FOR(i, 0, n) { ans[i] = A[i] * B[i] * C[i] *
65             D[i]; }
66         fft(ans, n, -1);
67         return;
68     }
69 };

```

### 11.5 GaussElimination

```

1 const int MAXN = 300;
2 const double EPS = 1e-8;
3 int n;
4 double A[MAXN][MAXN];
5 void Gauss()
6 {
7     for (int i = 0; i < n; i++)
8     {
9         bool ok = 0;
10        for (int j = i; j < n; j++)
11        {
12            if (fabs(A[j][i]) > EPS)
13            {
14                swap(A[j], A[i]);
15                ok = 1;
16                break;
17            }
18        }
19    }

```



```

18     }
19     if (!ok)
20         continue;
21     double fs = A[i][i];
22     for (int j = i + 1; j < n; j++)
23     {
24         double r = A[j][i] / fs;
25         for (int k = i; k < n; k++)
26         {
27             A[j][k] -= A[i][k] * r;
28         }
29     }
30 }
31 }

```

## 11.6 Matrix

```

1 struct Mat
2 {
3     int sz;
4     LL x[MXN][MXN];
5     Mat() { memset(x, 0, sizeof(x)); }
6     Mat(int _sz)
7     {
8         sz = _sz;
9         memset(x, 0, sizeof(x));
10    }
11    Mat operator*(Mat a)
12    {
13        Mat res(sz);
14        FOR(i, 1, sz + 1) FOR(j, 1, sz + 1) FOR(k, 1,
15            sz + 1)
16        {
17            res.x[i][j] += x[i][k] * a.x[k][j];
18            res.x[i][j] %= MOD;
19        }
20        return res;
21    }
22    void output()
23    {
24        FOR(i, 1, sz + 1) FOR(j, 1, sz + 1) cout <<
25            x[i][j] << " \n"[j == sz];
26    }
27 };

```

## 11.7 Phi

```

1 void phi_table(int n) // [1,n]
2 {
3     phi[1] = 1;
4     for (int i = 2; i <= n; i++)
5     {
6         if (phi[i])
7             continue;
8         for (int j = i; j <= n; j += i)
9         {
10            if (!phi[j])
11                phi[j] = j;
12            phi[j] = phi[j] / i * (i - 1);
13        }
14    }
15 }

```

## 11.8 PowerTower

```

1 int POW(int a, int b, int mod)
2 {
3     int ret = 1;
4     int tmp = 1;
5     for (int i = 0; i < b; i++)
6     {

```

```

7         tmp *= a;
8         if (tmp > mod)
9             break;
10    }
11    tmp = (tmp >= mod) ? mod : 0;
12    for (; b >= 1)
13    {
14        if (b & 1)
15            ret = ret * a % mod;
16        a = a * a % mod;
17    }
18    return ret + tmp;
19 }
20
21 int dfs(int d, int MOD)
22 {
23     if (d == n - 1)
24     {
25         if (a[d] >= MOD)
26             return (a[d] % MOD) + MOD;
27         return a[d];
28     }
29     int k = dfs(d + 1, phi[MOD]);
30     return POW(a[d], k, MOD);
31 }

```

## 11.9 Prime table

```

1 void primeTable()
2 {
3     is_notp.reset();
4     is_notp[0] = is_notp[1] = 1;
5     for (int i = 2; i < N; i++)
6     {
7         if (!is_notp[i])
8         {
9             p.push_back(i);
10        }
11        for (int j = 0; j < (int)p.size() && i * p[j]
12            < N; j++)
13        {
14            is_notp[i * p[j]] = 1;
15            if (i % p[j] == 0)
16            {
17                break;
18            }
19        }
20    }

```

## 12 String

### 12.1 Aho Corasick

```

1 struct Node
2 {
3     char ch;
4     int v;
5     Node *next[MXW];
6     Node *fail;
7     Node(): v(0), fail(0) {
8         memset(next, 0, sizeof(next)); }
9 };
10 void insert(Node *root, char *s)
11 {
12     int sz = strlen(s);
13     FOR(i, 0, sz)
14     {
15         int v = s[i] - 'a';
16         if (root->next[v] == NULL)
17         {

```

```

18     root->next[v] = new Node();
19 }
20 root = root->next[v];
21 root->ch = s[i];
22 }
23 ++root->v;
24 }
25
26 queue<Node *> q;
27 void bulidAC(Node *root)
28 {
29     Node *k, *tmp;
30     FOR(i, 0, MXW)
31     {
32         if (root->next[i] != NULL)
33         {
34             root->next[i]->fail = root;
35             q.push(root->next[i]);
36         }
37     }
38     while (!q.empty())
39     {
40         k = q.front();
41         q.pop();
42         FOR(i, 0, MXW) if (k->next[i] != NULL)
43         {
44             tmp = k->fail;
45             while (tmp != NULL)
46             {
47                 if (tmp->next[i] != NULL)
48                 {
49                     k->next[i]->fail = tmp->next[i];
50                     break;
51                 }
52                 tmp = tmp->fail;
53             }
54             if (tmp == NULL)
55             {
56                 k->next[i]->fail = root;
57             }
58             q.push(k->next[i]);
59         }
60     }
61 }
62
63 int ans;
64 void acAutomation(Node *root, char *s)
65 {
66     Node *p = root;
67     int sz = strlen(s);
68     FOR(i, 0, sz)
69     {
70         int v = s[i] - 'a';
71         while (p->next[v] == NULL && p != root)
72         {
73             p = p->fail;
74         }
75         p = p->next[v];
76         if (p == NULL)
77         {
78             p = root;
79         }
80         Node *k = p;
81         while (k != root)
82         {
83             if (k->v >= 0)
84             {
85                 ans += k->v;
86                 k->v = -1;
87             }
88             else
89             {
90                 break;
91             }
92             k = k->fail;
93         }
94     }

```

```

95 }
96
97 char s[MXS];
98 int main()
99 {
100     int t;
101     scanf("%d", &t);
102     while (t-->0)
103     {
104         int n;
105         Node *root = new Node();
106         scanf("%d", &n);
107         while (n-->0)
108         {
109             scanf("%s", s);
110             insert(root, s);
111         }
112         bulidAC(root);
113         scanf("%s", s);
114         ans = 0;
115         acAutomation(root, s);
116         printf("%d\n", ans);
117     }
118 }

```

## 12.2 KMP

```

1 void bulid_fail_funtion(string B, int *fail)
2 {
3     int len = B.length(), current_pos;
4     current_pos = fail[0] = -1;
5     for (int i = 1; i < len; i++)
6     {
7         while (current_pos != -1 && B[current_pos + 1] != B[i])
8         {
9             current_pos = fail[current_pos];
10        }
11        if (B[current_pos + 1] == B[i])
12        {
13            current_pos++;
14        }
15        fail[i] = current_pos;
16    }
17 }
18 void match(string A, string B, int *fail)
19 {
20     int lenA = A.length(), lenB = B.length();
21     int current_pos = -1;
22     for (int i = 0; i < lenA; i++)
23     {
24         while (current_pos != -1 && B[current_pos + 1] != A[i])
25         {
26             current_pos = fail[current_pos];
27         }
28         if (B[current_pos + 1] == A[i])
29             current_pos++;
30         if (current_pos == lenB - 1)
31             { // match! A[i-lenB+1,i]=B
32                 current_pos = fail[current_pos];
33             }
34     }
35 }
36 int main()
37 {
38     int t, i;
39     string s;
40     for (i = 0; cin >> t; i < t; i++)
41     {
42         cin >> s;
43         int fail[N];
44         bulid_fail_funtion(s, fail);
45         int p = s.length() - 1;
46         if (fail[p] != -1 && (p + 1) % (p - fail[p]) == 0)

```

```

47         printf("%d\n", p - fail[p]);
48     else
49         printf("%d\n", p + 1);
50 }
51 }

```

## 12.3 Manacher

```

1 void sol(char *s)
2 {
3     int sz = strlen(s);
4     si = 0;
5     ss[si++] = '$';
6     ss[si++] = '#';
7     FOR(i, 0, sz)
8     {
9         ss[si++] = s[i];
10        ss[si++] = '#';
11    }
12    ss[si++] = '_';
13    int mx = 0, id = 0;
14    FOR(i, 0, si)
15    {
16        if (mx > i)
17        {
18            ma[i] = min(ma[2 * id - i], mx - i);
19        }
20        else
21        {
22            ma[i] = 1;
23        }
24        while (ss[i + ma[i]] == ss[i - ma[i]])
25        {
26            ++ma[i];
27        }
28        if (i + ma[i] > mx)
29        {
30            id = i;
31            mx = i + ma[i];
32        }
33    }
34 }

```

## 12.4 Trie

```

1 struct Node
2 {
3     char ch;
4     int v;
5     Node *next[26];
6     Node()
7     {
8         v = 0;
9         FOR(i, 0, 26) next[i] = NULL;
10    }
11 };
12
13 void insert(Node *root, string s)
14 {
15     FOR(i, 0, s.size())
16     {
17         int v = s[i] - 'a';
18         if (root->next[v] == NULL)
19         {
20             root->next[v] = new Node();
21         }
22         root = root->next[v];
23         ++root->v;
24         root->ch = s[i];
25     }
26     return;
27 }
28 void search(Node *root, string s)

```

```

29 {
30     FOR(i, 0, s.size())
31     {
32         int v = s[i] - 'a';
33         root = root->next[v];
34         if (root->v == 1)
35         {
36             cout << s << ' ' << s.substr(0, i + 1) <<
37                 '\n';
38             return;
39         }
40     }
41     cout << s << ' ' << s << '\n';
42 }
43 int main()
44 {
45     vector<string> v;
46     string s;
47     Node *root = new Node();
48     while (cin >> s)
49     {
50         insert(root, s);
51         v.push_back(s);
52     }
53     FOR(i, 0, v.size()) { search(root, v[i]); }
54 }

```

## 12.5 Zvalue

```

1 void z_value(string s)
2 {
3     int L = 0, R = 0;
4     z[0] = 0;
5     for (int i = 1; i < (int)s.size(); i++)
6     {
7         if (i > R)
8         {
9             z[i] = 0;
10        }
11        else
12        {
13            int ip = i - L;
14            if (ip + z[ip] < z[L])
15            {
16                z[i] = z[ip];
17            }
18            else
19            {
20                z[i] = R - i + 1;
21            }
22        }
23        while (i + z[i] < (int)s.size() && s[i +
24            z[i]] == s[z[i]])
25        {
26            z[i]++;
27        }
28        if (i + z[i] - 1 > R)
29        {
30            L = i;
31            R = L + z[i] - 1;
32        }
33    }
34 }

```