Contents

1 Basic

1.1 Run

```
#use -> sh run.sh {name}
g++ -O2 -std=c++14 -Wall -Wextra -Wshadow -o $1 $1.cpp
./$1 < t.in > t.out
```

1.2 Default

```cpp
#include <bits/stdc++.h>
using namespace std;
using LL = long long;
#define IOS ios_base::sync_with_stdio(0); cin.tie(0);
#define pb push_back
#define eb emplace_back
const int INF = 1e9;
const int MOD = 1e9 + 7;
const double EPS = 1e-6;
const int MAXN = 0;

int main() {

}
```

1.3 Black Magic

```cpp
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
#include <ext/pb_ds/priority_queue.hpp>
using namespace std;
using namespace __gnu_pbds;
using set_t =
  tree<int, null_type, less<int>, rb_tree_tag,
    tree_order_statistics_node_update>;
using map_t =
  tree<int, int, less<int>, rb_tree_tag,
    tree_order_statistics_node_update>;
using heap_t =
  __gnu_pbds::priority_queue<int>;
using ht_t =
  gp_hash_table<int, int>;
int main() {
  //set---------------------------------
  set_t st;
  st.insert(5); st.insert(6);
  st.insert(3); st.insert(1);

  // the smallest is (0), biggest is (n-1), kth small
      is (k-1)
  int num = *st.find_by_order(0);
  cout << num << '\n'; // print 1

  num = *st.find_by_order(st.size() - 1);
  cout << num << '\n'; // print 6

  // find the index
  int index = st.order_of_key(6);
  cout << index << '\n'; // print 3

  // check if there exists x
  int x = 5;
  int check = st.erase(x);
  if (check == 0) printf("st not contain 5\n");
  else if (check == 1) printf("st contain 5\n");

  //tree policy like set
  st.insert(5); st.insert(5);
  cout << st.size() << '\n'; // print 4

  //map---------------------------------
  map_t mp;
  mp[1] = 2;
  cout << mp[1] << '\n';
  auto tmp = *mp.find_by_order(0); // pair
  cout << tmp.first << " " << tmp.second << '\n';

  //heap--------------------------------
  heap_t h1, h2;
  h1.push(1); h1.push(3);
  h2.push(2); h2.push(4);
  h1.join(h2);
```

```cpp
56    cout << h1.size() << h2.size() << h1.top() << '\n';
57    // 404
58
59    //hash-table-------------------------
60    ht_t ht;
61    ht[85] = 5;
62    ht[89975] = 234;
63    for (auto i : ht) {
64      cout << i.first << " " << i.second << '\n';
65    }
66 }
```

1.4 Python

```python
1  ### EOF
2  while True:
3    try:
4      pass
5    except EOFError:
6      break
7  ###math
8  import math
9
10 math.ceil(x)#上高斯
11 math.floor(x)#下高斯
12 math.factorial(x)#接乘
13 math.fabs(x)#絕對值
14 math.fsum(arr)#跟sum一樣但更精確(小數點問題)
15 math.gcd(x, y)#bj4
16 math.exp(x)#e^x
17 math.log(x, base)
18 math.log2(x)#2為底
19 math.log10(x)#10為底
20 math.sqrt(x)
21 math.pow(x, y, mod)#精確些(float型態) MOD!!!
22 math.sin(x)# cos tan asin acos atan atan2(弧度) sinh
       cosh tanh acosh asinh atanh
23 math.hypot(x, y)#歐幾里德範數
24 math.degrees(x)#x從弧度轉角度
25 math.radians(x)#x從角度轉弧度
26 math.gamma(x)#x的gamma函數
27 math.pi#常數
28 math.e#常數
29 math.inf
30
31 ### ascii
32 ord(x)#char to asc
33 chr(x)#asc to char
34
35 x.encode().hex()#string to hex
36 ### reverse string
37 string = "abc"
38 string_reverse = string[::-1]
```

1.5 Binary Search

```cpp
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);
```

1.6 Ternary Search

```cpp
1 const double EPS = 1e-6;
2 // target function
3 double f(double x) { return x * x; }
4 double ternarySearch() {
```

```cpp
5    double L = -1e5, R = 1e5;
6    while (R - L > EPS) {
7      double mr = (L + R) / 2.0;
8      double ml = (L + mr) / 2.0;
9      if (f(ml) < f(mr)) {
10       R = mr;
11     } else {
12       L = ml;
13     }
14   }
15   return L;
16 }
```

2 Data Structure

2.1 Disjoint Set

```cpp
1  // 0-base
2  const int MAXN = 1000;
3  int boss[MAXN];
4  void init(int n) {
5    for (int i = 0; i < n; i++) {
6      boss[i] = -1;
7    }
8  }
9  int find(int x) {
10   if (boss[x] < 0) {
11     return x;
12   }
13   return boss[x] = find(boss[x]);
14 }
15 bool uni(int a, int b) {
16   a = find(a);
17   b = find(b);
18   if (a == b) {
19     return false;
20   }
21   if (boss[a] > boss[b]) {
22     swap(a, b);
23   }
24   boss[a] += boss[b];
25   boss[b] = a;
26   return true;
27 }
```

2.2 BIT RARSQ

```cpp
1  // 1-base
2  #define lowbit(k) (k & -k)
3
4  int n;
5  vector<int> B1, B2;
6
7  void add(vector<int> &tr, int id, int val) {
8    for (; id <= n; id += lowbit(id)) {
9      tr[id] += val;
10   }
11 }
12 void range_add(int l, int r, int val) {
13   add(B1, l, val);
14   add(B1, r + 1, -val);
15   add(B2, l, val * (l - 1));
16   add(B2, r + 1, -val * r);
17 }
18 int sum(vector<int> &tr, int id) {
19   int ret = 0;
20   for (; id >= 1; id -= lowbit(id)) {
21     ret += tr[id];
22   }
23   return ret;
24 }
25 int prefix_sum(int id) {
```

```cpp
26    return sum(B1, id) * id - sum(B2, id);
27 }
28 int range_sum(int l, int r) {
29    return prefix_sum(r) - prefix_sum(l - 1);
30 }
```

2.3 zkw RMQ

```cpp
1  // 0-base
2  const int INF = 1e9;
3  const int MAXN = ;
4
5  int n;
6  int a[MAXN], tr[MAXN << 1];
7
8  // !!! remember to call this function
9  void build() {
10   for (int i = 0; i < n; i++) {
11     tr[i + n] = a[i];
12   }
13   for (int i = n - 1; i > 0; i--) {
14     tr[i] = max(tr[i << 1], tr[i << 1 | 1]);
15   }
16 }
17 void update(int id, int val) {
18   for (tr[id += n] = val; id > 1; id >>= 1) {
19     tr[id >> 1] = max(tr[id], tr[id ^ 1]);
20   }
21 }
22 int query(int l, int r) { // [l, r)
23   int ret = -INF;
24   for (l += n, r += n; l < r; l >>= 1, r >>= 1) {
25     if (l & 1) {
26       ret = max(ret, tr[l++]);
27     }
28     if (r & 1) {
29       ret = max(ret, tr[--r]);
30     }
31   }
32   return ret;
33 }
```

2.4 Segment Tree RARMQ

```cpp
1  struct Node {
2    int val, tag;
3    Node *lc, *rc;
4    Node() : lc(nullptr), rc(nullptr), tag(0) {}
5    void pull() {
6      if (!lc) {
7        val = rc->val;
8      } else if (!rc) {
9        val = lc->val;
10     } else {
11       val = max(lc->val, rc->val);
12     }
13   }
14   void push() {
15     if (lc) {
16       lc->tag += tag;
17       lc->val += tag;
18     }
19     if (rc) {
20       rc->tag += tag;
21       rc->val += tag;
22     }
23     tag = 0;
24   }
25 };
26 struct SegmentTree {
27   Node *root;
28   SegmentTree() : root(nullptr) {}
29   void build(Node* &T, int l, int r, const
         vector<int> &o) {
```

```cpp
30     T = new Node();
31     if (l == r) {
32       T->val = o[l];
33       return;
34     }
35     int mid = (l + r) / 2;
36     build(T->lc, l, mid, o);
37     build(T->rc, mid + 1, r, o);
38     T->pull();
39   }
40   void update(Node* &T, int l, int r, int ql, int qr,
         int v) {
41     if (ql <= l && r <= qr) {
42       T->val += v;
43       T->tag += v;
44       return;
45     }
46     T->push();
47     int mid = (l + r) / 2;
48     if (qr <= mid) {
49       update(T->lc, l, mid, ql, qr, v);
50     } else if (mid < ql) {
51       update(T->rc, mid + 1, r, ql, qr, v);
52     } else {
53       update(T->lc, l, mid, ql, mid, v);
54       update(T->rc, mid + 1, r, mid + 1, qr, v);
55     }
56     T->pull();
57   }
58   int query(Node* &T, int l, int r, int ql, int qr) {
59     if (ql <= l && r <= qr) {
60       return T->val;
61     }
62     T->push();
63     int mid = (l + r) / 2;
64     if (qr <= mid) {
65       return query(T->lc, l, mid, ql, qr);
66     } else if (mid < ql) {
67       return query(T->rc, mid + 1, r, ql, qr);
68     } else {
69       return max(query(T->lc, l, mid, ql, mid),
70           query(T->rc, mid + 1, r, mid + 1, qr));
71     }
72   }
73 };
```

2.5 Segment Tree RURMQ

```cpp
1  // from aizu
2  const ll maxn = 1e6 + 5;
3  const ll INF = 2147483647;
4  ll tree[maxn << 2], a[maxn], laze[maxn << 2];
5  void push_down(int rt, int ln, int rn) {
6    if (laze[rt] != -1) {
7      laze[rt << 1] = laze[rt];
8      laze[rt << 1 | 1] = laze[rt];
9      tree[rt << 1] = laze[rt];
10     tree[rt << 1 | 1] = laze[rt];
11     laze[rt] = -1;
12   }
13 }
14 void push_up(int rt) { tree[rt] = min(tree[rt << 1 |
       1], tree[rt << 1]); }
15 void build(ll l, ll r, ll rt) {
16   if (l == r) {
17     tree[rt] = a[l];
18     return;
19   }
20   ll m = (l + r) >> 1;
21   build(l, m, rt << 1);
22   build(m + 1, r, rt << 1 | 1);
23   push_up(rt);
24 }
25 //区间更新
26 void update(ll L, ll R, ll c, ll l, ll r, ll rt) {
```

```
27    if (L <= l && r <= R) {
28      laze[rt] = c;
29      tree[rt] = c;
30      return;
31    }
32    ll m = (l + r) >> 1;
33    push_down(rt, m - l + 1, r - m);
34    if (m >= L) update(L, R, c, l, m, rt << 1);
35    if (m < R) update(L, R, c, m + 1, r, rt << 1 | 1);
36    push_up(rt);
37 }
38
39 //区间更新的区间查询
40 ll query(ll L, ll R, ll l, ll r, ll rt) {
41    if (L <= l && r <= R) {
42      return tree[rt];
43    }
44    ll m = (l + r) >> 1, sum = INF;
45    push_down(rt, m - l + 1, r - m);
46    if (m >= L) sum = min(sum, query(L, R, l, m, rt <<
         1));
47    if (m < R) sum = min(sum, query(L, R, m + 1, r, rt
         << 1 | 1));
48    return sum;
49 }
50 void init() {
51    for (int i = 0; i < maxn; i++) a[i] = INF;
52    for (int i = 0; i < maxn * 4; i++) laze[i] = -1;
53    build(0, n - 1, 1);
54 }
55 /*
56 update(x,y,z,0,n-1,1);
57 query(x,y,0,n-1,1)
58 */
```

2.6 Treap

```
1  struct Treap {
2    int val, pri, sz;
3    Treap *lc, *rc;
4    Treap() {}
5    Treap(int _val) {
6      val = _val;
7      pri = rand();
8      sz = 1;
9      lc = rc = NULL;
10   }
11 };
12 int getSize(Treap *a) { return (a == NULL ? 0 :
       a->sz); }
13 void split(Treap *t, Treap *&a, Treap *&b, int k) {
14   if (t == NULL) {
15     a = b = NULL;
16     return;
17   }
18   if (getSize(t->lc) < k) {
19     a = t;
20     split(t->rc, a->rc, b, k - getSize(t->lc) - 1);
21   } else {
22     b = t;
23     split(t->lc, a, b->lc, k);
24   }
25 }
26 Treap *merge(Treap *a, Treap *b) {
27   if (!a || !b) {
28     return (a ? a : b);
29   }
30   if (a->pri > b->pri) {
31     a->rc = merge(a->rc, b);
32     return a;
33   } else {
34     b->lc = merge(a, b->lc);
35     return b;
36   }
37 }
```

```
38 void Insert(Treap *&t, int x, int p) {
39   Treap *a, *b;
40   split(t, a, b, x);
41   t = merge(a, merge(new Treap(p), b));
42 }
43 void Delete(Treap *&t, int x) {
44   Treap *a, *b, *c;
45   split(t, b, c, x);
46   split(b, a, b, x - 1);
47   t = merge(a, c);
48 }
49
50 /*
51 Usage
52 Treap *root = NULL; // declare
53 root = merge(root, new Treap(val)); // push back
54 Insert(root, x, y); // insert y after x-th element
55 Delete(root, x); // delete x-th element
56 */
```

3 Graph

3.1 Dijkstra

```
1  // 0-base
2  const LL INF = 1e18;
3  const int MAXN = ;
4  struct Edge {
5    int to;
6    LL cost;
7    Edge(int v, LL c) : to(v), cost(c) {}
8    bool operator < (const Edge &other) const {
9      return cost > other.cost;
10   }
11 };
12
13 int n;
14 LL dis[MAXN];
15 vector<Edge> G[MAXN];
16
17 void init() {
18   for (int i = 0; i < n; i++) {
19     G[i].clear();
20     dis[i] = INF;
21   }
22 }
23 void Dijkstra(int st, int ed = -1) {
24   priority_queue<Edge> pq;
25   pq.emplace(st, 0);
26   dis[st] = 0;
27   while (!pq.empty()) {
28     auto now = pq.top();
29     pq.pop();
30     if (now.to == ed) {
31       return;
32     }
33     if (now.cost > dis[now.to]) {
34       continue;
35     }
36     for (auto &e : G[now.to]) {
37       if (dis[e.to] > now.cost + e.cost) {
38         dis[e.to] = now.cost + e.cost;
39         pq.emplace(e.to, dis[e.to]);
40       }
41     }
42   }
43 }
```

3.2 SPFA(negative cycle)

```
1  // 0-base
2  const LL INF = 1e18;
```

```cpp
const int MAXN = ;
struct Edge {
  int to;
  LL cost;
  Edge(int v, LL c) : to(v), cost(c) {}
};

int n;
LL dis[MAXN];
vector<Edge> G[MAXN];

void init() {
  for (int i = 0; i < n; i++) {
    G[i].clear();
    dis[i] = INF;
  }
}
bool SPFA(int st) {
  vector<int> cnt(n, 0);
  vector<bool> inq(n, false);
  queue<int> q;

  q.push(st);
  dis[st] = 0;
  inq[st] = true;
  while (!q.empty()) {
    int now = q.front();
    q.pop();
    inq[now] = false;
    for (auto &e : G[now]) {
      if (dis[e.to] > dis[now] + e.cost) {
        dis[e.to] = dis[now] + e.cost;
        if (!inq[e.to]) {
          cnt[e.to]++;
          if (cnt[e.to] > n) {
            // negative cycle
            return false;
          }
          inq[e.to] = true;
          q.push(e.to);
        }
      }
    }
  }
  return true;
}
```

3.3 Floyd Warshall

```cpp
// 0-base
// G[i][i] < 0 -> negative cycle
const LL INF = 1e18;
const int MAXN = ;

int n;
LL G[MAXN][MAXN];

void init() {
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
      G[i][j] = INF;
    }
    G[i][i] = 0;
  }
}
void floyd() {
  for (int k = 0; k < n; k++) {
    for (int i = 0; i < n; i++) {
      for (int j = 0; j < n; j++) {
        if (G[i][k] != INF && G[k][j] != INF) {
          G[i][j] = min(G[i][j], G[i][k] + G[k][j]);
        }
      }
    }
  }
}
```

3.4 Topological Sort

```cpp
// 0-base
// if ret.size < n -> cycle
int n;
vector<vector<int>> G;

vector<int> topoSort() {
  vector<int> indeg(n), ret;
  for (auto &li : G) {
    for (int x : li) {
      ++indeg[x];
    }
  }
  // use priority queue for lexic. largest ans
  queue<int> q;
  for (int i = 0; i < n; i++) {
    if (!indeg[i]) {
      q.push(i);
    }
  }
  while (!q.empty()) {
    int u = q.front();
    q.pop();
    ret.pb(u);
    for (int v : G[u]) {
      if (--indeg[v] == 0) {
        q.push(v);
      }
    }
  }
  return ret;
}
```

3.5 Tree Diameter

```cpp
// 0-base;
const int MAXN = ;

struct Edge {
  int to;
  int cost;
  Edge(int v, int c) : to(v), cost(c) {}
};

int n, d = 0;
int d1[MAXN], d2[MAXN];
vector<Edge> G[MAXN];
// dfs(0, -1);
void dfs(int u, int from) {
  d1[u] = d2[u] = 0;
  for (auto e : G[u]) {
    if (e.to == from) {
      continue;
    }
    dfs(e.to, u);
    int t = d1[e.to] + e.cost;
    if (t > d1[u]) {
      d2[u] = d1[u];
      d1[u] = t;
    } else if (t > d2[u]) {
      d2[u] = t;
    }
  }
  d = max(d, d1[u] + d2[u]);
}
```

3.6 Directed MST

```cpp
// 0-base
const LL INF = 1e18;
const int MAXN = ;
```

```cpp
struct Edge {
  int from;
  int to;
  LL cost;
  Edge(int u, int v, LL c) : from(u), to(v), cost(c)
      {}
};

struct DMST {
  int n;
  int vis[MAXN], pre[MAXN], id[MAXN];
  LL in[MAXN];
  vector<Edge> edges;
  void init(int _n) {
    n = _n;
    edges.clear();
  }
  void add_edge(int from, int to, LL cost) {
    edges.eb(from, to, cost);
  }
  LL run(int root) {
    LL ret = 0;
    while (true) {
      for (int i = 0; i < n; i++) {
        in[i] = INF;
      }

      // find in edge
      for (auto &e : edges) {
        if (e.cost < in[e.to] && e.from != e.to) {
          pre[e.to] = e.from;
          in[e.to] = e.cost;
        }
      }

      // check in edge
      for (int i = 0; i < n; i++) {
        if (i == root) {
          continue;
        }
        if (in[i] == INF) {
          return -1;
        }
      }

      int nodenum = 0;
      memset(id, -1, sizeof(id));
      memset(vis, -1, sizeof(vis));
      in[root] = 0;

      // find cycles
      for (int i = 0; i < n; i++) {
        ret += in[i];
        int v = i;
        while (vis[v] != i && id[v] == -1 && v !=
            root) {
          vis[v] = i;
          v = pre[v];
        }
        if (id[v] == -1 && v != root) {
          for (int j = pre[v]; j != v; j = pre[j]) {
            id[j] = nodenum;
          }
          id[v] = nodenum++;
        }
      }

      // no cycle
      if (nodenum == 0) {
        break;
      }

      for (int i = 0; i < n; i++) {
        if (id[i] == -1) {
          id[i] = nodenum++;
        }
      }
```

```cpp
      // grouping the vertices
      for (auto &e : edges) {
        int to = e.to;
        e.from = id[e.from];
        e.to = id[e.to];
        if (e.from != e.to) {
          e.cost -= in[to]; //!!!
        }
      }

      n = nodenum;
      root = id[root];
    }
    return ret;
  }
};
```

3.7   LCA

```cpp
const int LOG = 20;
vector<int> tin(MAXN), tout(MAXN), depth(MAXN);
int par[MAXN][LOG];
int timer = 0;
vector<int> G[MAXN];

void dfs(int u, int f) {
  tin[u] = ++timer;
  par[u][0] = f;
  for (int v : G[u]) {
    if (v != f) {
      depth[v] = depth[u] + 1;
      dfs(v, u);
    }
  }
  tout[u] = ++timer;
}

void Doubling(int n) {
  for (int j = 1; j < LOG; ++j) {
    for (int i = 1; i <= n; ++i) {
      par[i][j] = par[par[i][j - 1]][j - 1];
    }
  }
}

bool anc(int u, int v) { return tin[u] <= tin[v] &&
    tout[v] <= tout[u]; }

int LCA(int u, int v) {
  if (depth[u] > depth[v]) {
    swap(u, v);
  }
  if (anc(u, v)) {
    return u;
  }
  for (int j = LOG - 1; j >= 0; --j) {
    if (!anc(par[u][j], v)) u = par[u][j];
  }
  return par[u][0];
}

int dis(int u, int v) {
  int lca = LCA(u, v);
  return depth[u] + depth[v] - 2 * depth[lca];
}

/*
dfs(root, root);
Doubling(n);
*/
```

3.8 Euler Circuit

七橋問題根據起點與終點是否相同，分成 Euler path（不同）及 Euler circuit（相同）。

- 判斷法

- 無向圖部分，將點分成奇點（度數為奇數）和偶點（度數為偶數）。

  - Euler path：奇點數為 0 或 2
  - Euler circuit：沒有奇點

- 有向圖部分，將點分成出點（出度 - 入度 = 1）和入點（入度 - 出度 = 1）還有平衡點（出度 = 入度）。

  - Euler path：出點和入點個數同時為 0 或 1。
  - Euler circuit：只有平衡點。

- 求出一組解

- 用 DFS 遍歷整張圖，設 S 為離開的順序，無向圖的答案為 S ，有向圖的答案為反向的 S 。

- DFS 起點選定：

  - Euler path：無向圖選擇任意一個奇點，有向圖選擇出點。
  - Euler circuit：任意一點。

```cpp
// Code from Eric
#define ll long long
#define PB push_back
#define EB emplace_back
#define PII pair<int, int>
#define MP make_pair
#define all(x) x.begin(), x.end()
#define maxn 50000+5

//structure
struct Eular {
  vector<PII> adj[maxn];
  vector<bool> edges;
  vector<PII> path;
  int chk[maxn];
  int n;

  void init(int _n) {
    n = _n;
    for (int i = 0; i <= n; i++) adj[i].clear();
    edges.clear();
    path.clear();
    memset(chk, 0, sizeof(chk));
  }

  void dfs(int v) {
    for (auto i : adj[v]) {
      if (edges[i.first] == true) {
        edges[i.first] = false;
        dfs(i.second);
        path.EB(MP(i.second, v));
      }
    }
  }

  void add_Edge(int from, int to) {
    edges.PB(true);

    // for bi-directed graph
    adj[from].PB(MP(edges.size() - 1, to));
    adj[to].PB(MP(edges.size() - 1, from));
    chk[from]++;
    chk[to]++;

    // for directed graph
    // adj[from].PB(MP(edges.size()-1, to));
    // check[from]++;
  }

  bool eular_path() {
    int st = -1;
    for (int i = 1; i <= n; i++) {
      if (chk[i] % 2 == 1) {
        st = i;
        break;
      }
    }
    if (st == -1) {
      return false;
    }
    dfs(st);
    return true;
  }

  void print_path(void) {
    for (auto i : path) {
      printf("%d %d\n", i.first, i.second);
    }
  }
};
```

```cpp
// Code from allen(lexicographic order)
#include <bits/stdc++.h>
using namespace std;
const int ALP = 30;
const int MXN = 1005;
int n;
int din[ALP], dout[ALP];
int par[ALP];
vector<string> vs[MXN], ans;
bitset<MXN> vis, used[ALP];

void djsInit() {
  for (int i = 0; i != ALP; ++i) {
    par[i] = i;
  }
}

int Find(int x) { return (x == par[x] ? (x) : (par[x]
    = Find(par[x]))); }

void init() {
  djsInit();
  memset(din, 0, sizeof(din));
  memset(dout, 0, sizeof(dout));
  vis.reset();
  for (int i = 0; i != ALP; ++i) {
    vs[i].clear();
    used[i].reset();
  }
  return;
}

void dfs(int u) {
  for (int i = 0; i != (int)vs[u].size(); ++i) {
    if (used[u][i]) {
      continue;
    }
    used[u][i] = 1;
    string s = vs[u][i];
    int v = s[s.size() - 1] - 'a';
    dfs(v);
    ans.push_back(s);
  }
}

bool solve() {
  int cnt = 1;
  for (int i = 0; i != n; ++i) {
    string s;
    cin >> s;
    int from = s[0] - 'a', to = s.back() - 'a';
    ++din[to];
    ++dout[from];
    vs[from].push_back(s);
    vis[from] = vis[to] = true;
    if ((from = Find(from)) != (to = Find(to))) {
      par[from] = to;
      ++cnt;
    }
```

```cpp
59      }
60      if ((int)vis.count() != cnt) {
61        return false;
62      }
63      int root, st, pin = 0, pout = 0;
64      for (int i = ALP - 1; i >= 0; --i) {
65        sort(vs[i].begin(), vs[i].end());
66        if (vs[i].size()) root = i;
67        int d = dout[i] - din[i];
68        if (d == 1) {
69          ++pout;
70          st = i;
71        } else if (d == -1) {
72          ++pin;
73        } else if (d != 0) {
74          return false;
75        }
76      }
77      if (pin != pout || pin > 1) {
78        return false;
79      }
80      ans.clear();
81      dfs((pin ? st : root));
82      return true;
83    }
84
85    int main() {
86      int t;
87      cin >> t;
88      while (t--) {
89        cin >> n;
90        init();
91        if (!solve()) {
92          cout << "***\n";
93          continue;
94        }
95        for (int i = ans.size() - 1; i >= 0; --i) {
96          cout << ans[i] << ".\n"[i == 0];
97        }
98      }
99    }
```

4 Connectivity

4.1 Kosaraju SCC

```cpp
1  // 0-base
2  int n;
3  vector<vector<int>> G, G2; // G2 = G rev
4  vector<bool> vis;
5  vector<int> s, color;
6  int sccCnt;
7  void dfs1(int u) {
8    vis[u] = true;
9    for (int v : G[u]) {
10     if (!vis[v]) {
11       dfs1(v);
12     }
13   }
14   s.pb(u);
15 }
16 void dfs2(int u) {
17   color[u] = sccCnt;
18   for (int v : G2[u]) {
19     if (!color[v]) {
20       dfs2(v);
21     }
22   }
23 }
24 void Kosaraju() {
25   sccCnt = 0;
26   for (int i = 0; i < n; i++) {
27     if (!vis[i]) {
28       dfs1(i);
```

```cpp
29       }
30     }
31     for (int i = n - 1; i >= 0; i--) {
32       if (!color[s[i]]) {
33         ++sccCnt;
34         dfs2(s[i]);
35       }
36     }
37 }
```

4.2 BCC

```cpp
1  typedef pair<int, int> PII;
2  int low[MXV], depth[MXV];
3  bool is_cut_vertex[MXV], visit[MXV];
4  vector<int> G[MXV];
5  vector<PII> BCC[MXV];
6  int bcc_cnt = 0;
7  stack<PII> st;
8
9  vector<pair<int, int>> my_cut_edge;
10
11 void dfs(int now, int cur_depth, int f) {
12   visit[now] = true;
13   depth[now] = low[now] = cur_depth;
14   int cut_son = 0;
15   for (auto i : G[now]) {
16     if (i == f) continue;
17     if (visit[i]) {   // ancestor
18       if (depth[i] < depth[now]) { // #
19         low[now] = min(low[now], depth[i]);
20         st.push({now, i});
21       }
22     } else {   // offspring
23       st.push({now, i});
24       dfs(i, cur_depth + 1, now);
25       cut_son += 1;
26       low[now] = min(low[now], low[i]);
27       if (low[i] >= depth[now]) {
28         is_cut_vertex[now] = true;
29         auto t = st.top();
30         st.pop();
31         while (t != make_pair(now, i)) {
32           BCC[bcc_cnt].push_back(t);
33           t = st.top();
34           st.pop();
35         }
36         BCC[bcc_cnt].push_back(t);
37         ++bcc_cnt;
38       }
39       // ###
40       if (low[i] > depth[now])
41         my_cut_edge.push_bach({now, i});
42     }
43   }
44   if (cur_depth == 0)
45     is_cut_vertex[now] = (cut_son != 1);
46   return;
47 }
48
49 bool is_2_edge_connected(int n) {
50   memset(visit, 0, sizeof(visit));
51   dfs(1, 0, -1);
52   return my_cut_edge.size() == 0;
53 }
```

4.3 Articulation Point

```cpp
1  // from aizu
2  typedef long long int ll;
3  typedef unsigned long long int ull;
4  #define BIG_SIZE 2000000000
5  #define MOD 1000000007
```

```
6  #define EPS 0.000000001
7  using namespace std;
8
9  #define SIZE 100000
10
11 vector<int> G[SIZE];
12 int N;
13 bool visited[SIZE];
14 int visited_order[SIZE], parent[SIZE], lowest[SIZE],
       number;
15
16 void dfs(int cur, int pre_node) {
17   visited_order[cur] = lowest[cur] = number;
18   number++;
19
20   visited[cur] = true;
21
22   int next;
23
24   for (int i = 0; i < G[cur].size(); i++) {
25     next = G[cur][i];
26     if (!visited[next]) {
27       parent[next] = cur;
28       dfs(next, cur);
29       lowest[cur] = min(lowest[cur], lowest[next]);
30     } else if (visited[next] == true && next !=
           pre_node) {
31       lowest[cur] = min(lowest[cur],
             visited_order[next]);
32     }
33   }
34 }
35
36 void art_points() {
37   for (int i = 0; i < N; i++) visited[i] = false;
38
39   number = 1;
40   dfs(0, -1);
41
42   int tmp_parent, root_num = 0;
43
44   vector<int> V;
45
46   for (int i = 1; i < N; i++) {
47     tmp_parent = parent[i];
48     if (tmp_parent == 0) {
49       root_num++;
50     } else if (visited_order[tmp_parent] <=
           lowest[i]) {
51       V.push_back(tmp_parent);
52     }
53   }
54   if (root_num >= 2) {
55     V.push_back(0);
56   }
57   sort(V.begin(), V.end());
58   V.erase(unique(V.begin(), V.end()), V.end());
59
60   for (int i = 0; i < V.size(); i++) {
61     printf("%d\n", V[i]);
62   }
63 }
64
65 int main() {
66   int E;
67   scanf("%d %d", &N, &E);
68   int from, to;
69   for (int i = 0; i < E; i++) {
70     scanf("%d %d", &from, &to);
71     G[from].push_back(to);
72     G[to].push_back(from);
73   }
74   art_points();
75 }
```

4.4 Bridges

```
1  // from aizu
2  typedef long long int ll;
3  typedef unsigned long long int ull;
4  #define BIG_NUM 2000000000
5  #define MOD 1000000007
6  #define EPS 0.000000001
7  using namespace std;
8
9  struct Edge {
10   bool operator<(const struct Edge &arg) const {
11     if (s != arg.s) {
12       return s < arg.s;
13     } else {
14       return t < arg.t;
15     }
16   }
17   int s, t;
18 };
19 struct Info {
20   Info(int arg_to, int arg_edge_id) {
21     to = arg_to;
22     edge_id = arg_edge_id;
23   }
24   int to, edge_id;
25 };
26
27 int V, E, number;
28 int order[100000], lowlink[100000];
29 bool visited[100000];
30 Edge edge[100000];
31 vector<Info> G[100000];
32
33 void recursive(int cur) {
34   order[cur] = number++;
35   lowlink[cur] = order[cur];
36
37   int next;
38
39   for (int i = 0; i < G[cur].size(); i++) {
40     next = G[cur][i].to;
41
42     if (order[next] == -1) {
43       visited[G[cur][i].edge_id] = true;
44       recursive(next);
45       lowlink[cur] = min(lowlink[cur], lowlink[next]);
46
47     } else if (visited[G[cur][i].edge_id] == false) {
48       lowlink[cur] = min(lowlink[cur], order[next]);
49     }
50   }
51 }
52
53 int main() {
54   scanf("%d %d", &V, &E);
55   for (int i = 0; i < E; i++) {
56     scanf("%d %d", &edge[i].s, &edge[i].t);
57     if (edge[i].s > edge[i].t) {
58       swap(edge[i].s, edge[i].t);
59     }
60     G[edge[i].s].push_back(Info(edge[i].t, i));
61     G[edge[i].t].push_back(Info(edge[i].s, i));
62   }
63
64   sort(edge, edge + E);
65
66   number = 0;
67   for (int i = 0; i < V; i++) {
68     order[i] = -1;
69     lowlink[i] = -1;
70   }
71   for (int i = 0; i < E; i++) {
72     visited[i] = false;
73   }
74
75   recursive(0);
```

```
76
77   int from, to;
78   for (int i = 0; i < E; i++) {
79     from = edge[i].s;
80     to = edge[i].t;
81     if (order[edge[i].s] > order[edge[i].t]) {
82       swap(from, to);
83     }
84     if (order[from] < lowlink[to]) {
85       printf("%d %d\n", edge[i].s, edge[i].t);
86     }
87   }
88   return 0;
89 }
```

5 Flow & Matching

5.1 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|
```

5.2 Bipartite Matching

```
1  // 0-base
2  const int MAXN = ;
3  int n;
4  vector<int> G[MAXN];
5  int vy[MAXN], my[MAXN];
6
7  bool match(int u) {
8    for (int v : G[u]) {
9      if (vy[v]) {
10       continue;
11     }
12     vy[v] = true;
13     if (my[v] == -1 || match(my[v])) {
14       my[v] = u;
15       return true;
16     }
17   }
18   return false;
19 }
20 int sol() {
21   int cnt = 0;
22   memset(my, -1, sizeof(my));
23   for (int i = 0; i < n; i++) {
24     memset(vy, 0, sizeof(vy));
25     if (match(i)) {
26       cnt++;
27     }
28   }
29   return cnt;
30 }
```

5.3 KM

```
1  const int INF = 1e9;
2  const int MAXN = ;
3  struct KM { //1-base
4    int n, G[MAXN][MAXN];
5    int lx[MAXN], ly[MAXN], my[MAXN];
```

```
6    bool vx[MAXN], vy[MAXN];
7    void init(int _n) {
8      n = _n;
9      for (int i = 1; i <= n; i++) {
10       for (int j = 1; j <= n; j++) {
11         G[i][j] = 0;
12       }
13     }
14   }
15   bool match(int i) {
16     vx[i] = true;
17     for (int j = 1; j <= n; j++) {
18       if (lx[i] + ly[j] == G[i][j] && !vy[j]) {
19         vy[j] = true;
20         if (!my[j] || match(my[j])) {
21           my[j] = i;
22           return true;
23         }
24       }
25     }
26     return false;
27   }
28   void update() {
29     int delta = INF;
30     for (int i = 1; i <= n; i++) {
31       if (vx[i]) {
32         for (int j = 1; j <= n; j++) {
33           if (!vy[j]) {
34             delta = min(delta, lx[i] + ly[j] -
                   G[i][j]);
35           }
36         }
37       }
38     }
39     for (int i = 1; i <= n; i++) {
40       if (vx[i]) {
41         lx[i] -= delta;
42       }
43       if (vy[i]) {
44         ly[i] += delta;
45       }
46     }
47   }
48   int run() {
49     for (int i = 1; i <= n; i++) {
50       lx[i] = ly[i] = my[i] = 0;
51       for (int j = 1; j <= n; j++) {
52         lx[i] = max(lx[i], G[i][j]);
53       }
54     }
55     for (int i = 1; i <= n; i++) {
56       while (true) {
57         for (int i = 1; i <= n; i++) {
58           vx[i] = vy[i] = 0;
59         }
60         if (match(i)) {
61           break;
62         } else {
63           update();
64         }
65       }
66     }
67     int ans = 0;
68     for (int i = 1; i <= n; i++) {
69       ans += lx[i] + ly[i];
70     }
71     return ans;
72   }
73 };
```

5.4 Dinic

```
1  #define eb emplace_back
2  const LL INF = 1e18;
3  const int MAXN = ;
4  struct Edge {
```

```
 5   int to;
 6   LL cap;
 7   int rev;
 8   Edge(int v, LL c, int r) : to(v), cap(c), rev(r) {}
 9 };
10 struct Dinic {
11   int n;
12   int level[MAXN], now[MAXN];
13   vector<Edge> G[MAXN];
14   void init(int _n) {
15     n = _n;
16     for (int i = 0; i <= n; i++) {
17       G[i].clear();
18     }
19   }
20   void add_edge(int u, int v, LL c) {
21     G[u].eb(v, c, G[v].size());
22     // directed graph
23     G[v].eb(u, 0, G[u].size() - 1);
24     // undirected graph
25     // G[v].eb(u, c, G[u].size() - 1);
26   }
27   bool bfs(int st, int ed) {
28     fill(level, level + n + 1, -1);
29     queue<int> q;
30     q.push(st);
31     level[st] = 0;
32     while (!q.empty()) {
33       int u = q.front();
34       q.pop();
35       for (const auto &e : G[u]) {
36         if (e.cap > 0 && level[e.to] == -1) {
37           level[e.to] = level[u] + 1;
38           q.push(e.to);
39         }
40       }
41     }
42     return level[ed] != -1;
43   }
44   LL dfs(int u, int ed, LL limit) {
45     if (u == ed) {
46       return limit;
47     }
48     LL ret = 0;
49     for (int &i = now[u]; i < G[u].size(); i++) {
50       auto &e = G[u][i];
51       if (e.cap > 0 && level[e.to] == level[u] + 1) {
52         LL f = dfs(e.to, ed, min(limit, e.cap));
53         ret += f;
54         limit -= f;
55         e.cap -= f;
56         G[e.to][e.rev].cap += f;
57         if (!limit) {
58           return ret;
59         }
60       }
61     }
62     if (!ret) {
63       level[u] = -1;
64     }
65     return ret;
66   }
67   LL flow(int st, int ed) {
68     LL ret = 0;
69     while (bfs(st, ed)) {
70       fill(now, now + n + 1, 0);
71       ret += dfs(st, ed, INF);
72     }
73     return ret;
74   }
75 };
```

5.5 MCMF

```
 1 // 0-base
 2 const LL INF = 1e18;
 3 const int MAXN = ;
 4 struct Edge {
 5   int u, v;
 6   LL cost;
 7   LL cap;
 8   Edge(int _u, int _v, LL _c, LL _cap) : u(_u),
       v(_v), cost(_c), cap(_cap) {}
 9 };
10 struct MCMF {        // inq times
11   int n, pre[MAXN], cnt[MAXN];
12   LL ans_flow, ans_cost, dis[MAXN];
13   bool inq[MAXN];
14   vector<int> G[MAXN];
15   vector<Edge> edges;
16   void init(int _n) {
17     n = _n;
18     edges.clear();
19     for (int i = 0; i < n; i++) {
20       G[i].clear();
21     }
22   }
23   void add_edge(int u, int v, LL c, LL cap) {
24     // directed
25     G[u].pb(edges.size());
26     edges.eb(u, v, c, cap);
27     G[v].pb(edges.size());
28     edges.eb(v, u, -c, 0);
29   }
30   bool SPFA(int st, int ed) {
31     for (int i = 0; i < n; i++) {
32       pre[i] = -1;
33       dis[i] = INF;
34       cnt[i] = 0;
35       inq[i] = false;
36     }
37     queue<int> q;
38     bool negcycle = false;
39
40     dis[st] = 0;
41     cnt[st] = 1;
42     inq[st] = true;
43     q.push(st);
44
45     while (!q.empty() && !negcycle) {
46       int u = q.front();
47       q.pop();
48       inq[u] = false;
49       for (int i : G[u]) {
50         int v = edges[i].v;
51         LL cost = edges[i].cost;
52         LL cap = edges[i].cap;
53
54         if (dis[v] > dis[u] + cost && cap > 0) {
55           dis[v] = dis[u] + cost;
56           pre[v] = i;
57           if (!inq[v]) {
58             q.push(v);
59             cnt[v]++;
60             inq[v] = true;
61
62             if (cnt[v] == n + 2) {
63               negcycle = true;
64               break;
65             }
66           }
67         }
68       }
69     }
70
71     return dis[ed] != INF;
72   }
73   LL sendFlow(int v, LL curFlow) {
74     if (pre[v] == -1) {
75       return curFlow;
76     }
77     int i = pre[v];
78     int u = edges[i].u;
```

```
79      LL cost = edges[i].cost;
80
81      LL f = sendFlow(u, min(curFlow, edges[i].cap));
82
83      ans_cost += f * cost;
84      edges[i].cap -= f;
85      edges[i ^ 1].cap += f;
86      return f;
87    }
88    pair<LL, LL> run(int st, int ed) {
89      ans_flow = ans_cost = 0;
90      while (SPFA(st, ed)) {
91        ans_flow += sendFlow(ed, INF);
92      }
93      return make_pair(ans_flow, ans_cost);
94    }
95 };
```

5.6 Stable Matching

```
1  int t, n, b[N][N], bi[N], g[N][N], bg[N], gb[N];
2
3  void sol() {
4    deque<int> dq;
5    memset(gb, 0, sizeof(gb));
6    memset(bi, 0, sizeof(bi));
7    for (int i = 1; i <= n; i++) dq.push_back(i);
8    while (!dq.empty()) {
9      int x = dq.front();
10     dq.pop_front();
11     int y = b[x][++bi[x]];
12     if (!gb[y]) {
13       gb[y] = x;
14       bg[x] = y;
15     } else if (g[y][x] < g[y][gb[y]]) {
16       dq.push_back(gb[y]);
17       gb[y] = x;
18       bg[x] = y;
19     } else {
20       dq.push_back(x);
21     }
22   }
23   for (int i = 1; i <= n; i++) {
24     cout << bg[i] << '\n';
25   }
26 }
27
28 int main() {
29   int x;
30   cin >> t;
31   for (int i = 0; i < t; i++) {
32     cin >> n;
33     for (int i = 1; i <= n; i++) {
34       for (int j = 1; j <= n; j++) {
35         cin >> b[i][j];
36       }
37     }
38     for (int i = 1; i <= n; i++) {
39       for (int j = 1; j <= n; j++) {
40         cin >> x;
41         g[i][x] = j;
42       }
43     }
44     if (i) cout << '\n';
45     sol();
46   }
47 }
```

5.7 Max General Graph Matching

```
1  #define maxn
2
3  int fa[maxn], pre[maxn], match[maxn], s[maxn],
       v[maxn];
```

```
4  vector<int> g[maxn];
5  queue<int> q;
6  void Init(int n) {
7    for (int i = 0; i <= n; ++i) match[i] = pre[i] = n;
8    for (int i = 0; i < n; ++i) g[i].clear();
9  }
10 void AddEdge(int u, int v) {
11   g[u].push_back(v);
12   g[v].push_back(u);
13 }
14 int Find(int u) { return u == fa[u] ? u : fa[u] =
       Find(fa[u]); }
15 int LCA(int x, int y, int n) {
16   static int tk = 0;
17   tk++;
18   x = Find(x), y = Find(y);
19   for (;; swap(x, y)) {
20     if (x != n) {
21       if (v[x] == tk) return x;
22       v[x] = tk;
23       x = Find(pre[match[x]]);
24     }
25   }
26 }
27
28 void Blossom(int x, int y, int l) {
29   while (Find(x) != l) {
30     pre[x] = y, y = match[x];
31     if (s[y] == 1) q.push(y), s[y] = 0;
32     if (fa[x] == x) fa[x] = l;
33     if (fa[y] == y) fa[y] = l;
34     x = pre[y];
35   }
36 }
37
38 bool Bfs(int r, int n) {
39   for (int i = 0; i <= n; ++i) fa[i] = i, s[i] = -1;
40   while (!q.empty()) q.pop();
41   q.push(r);
42   s[r] = 0;
43   while (!q.empty()) {
44     int x = q.front();
45     q.pop();
46     for (int u : g[x]) {
47       if (s[u] == -1) {
48         pre[u] = x, s[u] = 1;
49         if (match[u] == n) {
50           for (int a = u, b = x, last; b != n; a =
                 last, b = pre[a])
51             last = match[b], match[b] = a, match[a] =
                   b;
52           return true;
53         }
54         q.push(match[u]);
55         s[match[u]] = 0;
56       } else if (!s[u] && Find(u) != Find(x)) {
57         int l = LCA(u, x, n);
58         Blossom(x, u, l);
59         Blossom(u, x, l);
60       }
61     }
62   }
63   return false;
64 }
65
66 int Solve(int n) {
67   int res = 0;
68   for (int x = 0; x < n; ++x) {
69     if (match[x] == n) res += Bfs(x, n);
70   }
71   return res;
72 }
```

6 String

6.1 Manacher

```cpp
int p[2 * MAXN];
int Manacher(const string &s) {
  string st = "@#";
  for (char c : s) {
    st += c;
    st += '#';
  }
  st += '$';
  int id = 0, mx = 0, ans = 0;
  for (int i = 1; i < st.length() - 1; i++) {
    p[i] = (mx > i ? min(p[2 * id - i], mx - i) : 1);
    for (; st[i - p[i]] == st[i + p[i]]; p[i]++);
    if (mx < i + p[i]) {
      mx = i + p[i];
      id = i;
    }
    ans = max(ans, p[i] - 1);
  }
  return ans;
}
```

6.2 Trie

```cpp
const int MAXL = ;
const int MAXC = ;
struct Trie {
  int nex[MAXL][MAXC];
  int len[MAXL];
  int sz;
  void init() {
    memset(nex, 0, sizeof(nex));
    memset(len, 0, sizeof(len));
    sz = 0;
  }
  void insert(const string &str) {
    int p = 0;
    for (char c : str) {
      int id = c - 'a';
      if (!nex[p][id]) {
        nex[p][id] = ++sz;
      }
      p = nex[p][id];
    }
    len[p] = str.length();
  }
  vector<int> find(const string &str, int i) {
    int p = 0;
    vector<int> ans;
    for (; i < str.length(); i++) {
      int id = str[i] - 'a';
      if (!nex[p][id]) {
        return ans;
      }
      p = nex[p][id];
      if (len[p]) {
        ans.pb(len[p]);
      }
    }
    return ans;
  }
};
```

6.3 Z-value

```cpp
// 0-base
// 對於個長度為 n 的字串 s
// 定義函數 z[i] 表示 s 和 s[i, n - 1]
// （即以 s[i] 開頭的後綴）的最長公共前綴（LCP）的長度
// z[0] = 0。
vector<int> z_function(string s) {
  int n = (int)s.length();
  vector<int> z(n);
  for (int i = 1, l = 0, r = 0; i < n; ++i) {
    if (i <= r && z[i - l] < r - i + 1) {
      z[i] = z[i - l];
    } else {
      z[i] = max(0, r - i + 1);
      while (i + z[i] < n && s[z[i]] == s[i + z[i]])
        ++z[i];
    }
    if (i + z[i] - 1 > r) l = i, r = i + z[i] - 1;
  }
  return z;
}
```

7 DP

7.1 LIS

```cpp
int LIS(vector<int> &a) {
  vector<int> s;
  for (int i = 0; i < a.size(); i++) {
    if (s.empty() || s.back() < a[i]) {
      s.push_back(a[i]);
    } else {
      *lower_bound(s.begin(), s.end(), a[i],
        [](int x, int y) {return x < y;}) = a[i];
    }
  }
  return s.size();
}
```

7.2 LCS

```cpp
int LCS(string s1, string s2) {
  int n1 = s1.size(), n2 = s2.size();
  vector<vector<int>> dp(n1 + 1, vector<int>(n2 + 1,
      0));
  for (int i = 1; i <= n1; i++) {
    for (int j = 1; j <= n2; j++) {
      if (s1[i - 1] == s2[j - 1]) {
        dp[i][j] = dp[i - 1][j - 1] + 1;
      } else {
        dp[i][j] = max(dp[i - 1][j], dp[i][j - 1]);
      }
    }
  }
  return dp[n1][n2];
}
```

7.3 Huge Knapsack

```cpp
// from aizu
#include <bits/stdc++.h>
typedef long long int ll;
typedef unsigned long long int ull;
#define BIG_NUM 2000000000
#define HUGE_NUM 99999999999999999
#define MOD 1000000007
#define EPS 0.000000001
using namespace std;

#define SIZE 25

struct Info {
  Info() { value = 0, weight = 0; }
  Info(ll arg_value, ll arg_weight) {
```

```
16      value = arg_value;
17      weight = arg_weight;
18   }
19   bool operator<(const struct Info &arg) const {
        return weight < arg.weight; }
20
21   ll value, weight;
22 };
23
24 ll N, W;
25 ll POW[SIZE];
26 ll table_B[1 << 21];
27 Info info[45];
28
29 int main() {
30   POW[0] = 1;
31   for (int i = 1; i < SIZE; i++) {
32     POW[i] = POW[i - 1] * 2;
33   }
34
35   scanf("%lld %lld", &N, &W);
36
37   for (int i = 0; i < N; i++) {
38     scanf("%lld %lld", &info[i].value,
           &info[i].weight);
39   }
40
41   if (N == 1) {
42     if (info[0].weight <= W) {
43       printf("%lld\n", info[0].value);
44     } else {
45       printf("0\n");
46     }
47
48     return 0;
49   }
50
51   vector<int> A, B;
52   for (int i = 0; i < N / 2; i++) {
53     A.push_back(i);
54   }
55   for (int i = N / 2; i < N; i++) {
56     B.push_back(i);
57   }
58
59   vector<Info> vec_A, vec_B;
60   for (int state = 0; state < POW[A.size()]; state++)
        {
61     ll sum_w = 0;
62     ll sum_value = 0;
63     for (int loop = 0; loop < A.size(); loop++) {
64       if (state & POW[loop]) {
65         sum_w += info[A[loop]].weight;
66         sum_value += info[A[loop]].value;
67       }
68     }
69     vec_A.push_back(Info(sum_value, sum_w));
70   }
71   sort(vec_A.begin(), vec_A.end());
72
73   for (int state = 0; state < POW[B.size()]; state++)
        {
74     ll sum_w = 0;
75     ll sum_value = 0;
76     for (int loop = 0; loop < B.size(); loop++) {
77       if (state & POW[loop]) {
78         sum_w += info[B[loop]].weight;
79         sum_value += info[B[loop]].value;
80       }
81     }
82     vec_B.push_back(Info(sum_value, sum_w));
83   }
84   sort(vec_B.begin(), vec_B.end());
85
86   table_B[0] = vec_B[0].value;
87   for (int i = 1; i < vec_B.size(); i++) {
88     //ある重さ以下の最大価値を求める
```

```
89     table_B[i] = max(table_B[i - 1], vec_B[i].value);
90   }
91
92   int tail = vec_B.size() - 1;
93   ll ans = 0;
94   for (int i = 0; i < vec_A.size(); i++) {
95     while (tail >= 0 && vec_A[i].weight +
           vec_B[tail].weight > W) tail--;
96     if (tail < 0) break;
97
98     ans = max(ans, vec_A[i].value + table_B[tail]);
99   }
100
101  printf("%lld\n", ans);
102  return 0;
103 }
```

7.4   Coin Change

```
1  // from aizu
2  int main() {
3    int n, m, min, tmp;
4    scanf("%d", &n);
5    int minimum[n + 1];
6    scanf("%d", &m);
7    int coin[m];
8    for (int i = 0; i < m; i++) scanf("%d", &coin[i]);
9
10   minimum[0] = 0;
11   minimum[1] = 1;
12   for (int i = 2; i <= n; i++) {
13     min = n + 1;
14     for (int k = 0; k < m; k++) {
15       if (coin[k] <= i) {
16         tmp = 1 + minimum[i - coin[k]];
17         min = (min <= tmp) ? min : tmp;
18       }
19     }
20     minimum[i] = min;
21   }
22
23   printf("%d\n", minimum[n]);
24
25   return 0;
26 }
```

7.5   Edit Distance

```
1  // from aizu
2  typedef long long int ll;
3  typedef unsigned long long int ull;
4  #define BIG_NUM 2000000000
5  #define MOD 1000000007
6  #define EPS 0.000000001
7
8  int main() {
9    char A[1001], B[1001];
10   int len_A, len_B;
11   scanf("%s %s", A, B);
12
13   for (len_A = 0; A[len_A] != '\0'; len_A++);
14   for (len_B = 0; B[len_B] != '\0'; len_B++);
15
16   int** dp = new int*[len_B + 1];
17
18   for (int row = 0; row <= len_B; row++) {
19     dp[row] = new int[len_A + 1];
20   }
21
22   for (int col = 0; col <= len_A; col++) {
23     dp[0][col] = col;
24   }
25
```

```
26    for (int row = 1; row <= len_B; row++) {
27      dp[row][0] = row;
28    }
29
30    int cost;
31
32    for (int row = 1; row <= len_B; row++) {
33      for (int col = 1; col <= len_A; col++) {
34        if (A[col - 1] == B[row - 1]) {
35          cost = 0;
36        } else {
37          cost = 1;
38        }
39        dp[row][col] = min(dp[row - 1][col - 1] + cost,
40          min(dp[row - 1][col] + 1, dp[row][col - 1] +
                  1));
41      }
42    }
43
44    printf("%d\n", dp[len_B][len_A]);
45 }
```

7.6 Maximum Subrectangle

```
1  int kadane2D(int array[N][M]) {
2    // Modify the array's elements to now hold the sum
3    // of all the numbers that are above that element
         in its column
4    for (int i = 1; i < N; i++) {
5      for (int j = 0; j < M; j++) {
6        array[i][j] += array[i - 1][j];
7      }
8    }
9
10   int ans = 0;   // Holds the maximum sum matrix found
         till now
11
12   for (int bottom = 0; bottom < N; bottom++) {
13     for (int top = bottom; top < N; top++) {
14       // loop over all the N^2 sub problems
15       int[] sums = new int[N];
16
17       // store the sum of numbers between the two rows
18       // in the sums array
19       for (int i = 0; i < M; i++) {
20         if (bottom > 0) {
21           sums[i] = array[top][i] - array[bottom -
                  1][i];
22         } else {
23           sums[i] = array[top][i];
24         }
25       }
26
27       // O(n) time to run 1D kadane's on this sums
              array
28       ans = Math.max(ans, kadane1d(sums));
29     }
30   }
31   return ans;
32 }
```

8 Math

8.1 Number Theory

- Inversion:
  $aa^{-1} \equiv 1 \pmod{m}$. a^{-1} exists iff $\gcd(a, m) = 1$.

- Linear inversion:
  $a^{-1} \equiv (m - \lfloor \frac{m}{a} \rfloor) \times (m \bmod a)^{-1} \pmod{m}$

- Fermat's little theorem:
  $a^p \equiv a \pmod{p}$ if p is prime.

- Euler function:
  $\phi(n) = n \prod_{p|n} \frac{p-1}{p}$

- Euler theorem:
  $a^{\phi(n)} \equiv 1 \pmod{n}$ if $\gcd(a, n) = 1$.

- Extended Euclidean algorithm:
  $ax + by = \gcd(a, b) = \gcd(b, a \bmod b) = \gcd(b, a - \lfloor \frac{a}{b} \rfloor b) = bx_1 + (a - \lfloor \frac{a}{b} \rfloor b)y_1 = ay_1 + b(x_1 - \lfloor \frac{a}{b} \rfloor y_1)$

- Divisor function:
  $\sigma_x(n) = \sum_{d|n} d^x$. $n = \prod_{i=1}^{r} p_i^{a_i}$.
  $\sigma_x(n) = \prod_{i=1}^{r} \frac{p_i^{(a_i+1)x} - 1}{p_i^x - 1}$ if $x \neq 0$. $\sigma_0(n) = \prod_{i=1}^{r} (a_i + 1)$.

- Chinese remainder theorem:
  $x \equiv a_i \pmod{m_i}$.
  $M = \prod m_i$. $M_i = M/m_i$. $t_i = M_i^{-1}$.
  $x = kM + \sum a_i t_i M_i,\ k \in \mathbb{Z}$.

8.2 Extended GCD

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

8.3 Gaussian Elimination + det

```
1  const double EPS = 1e-6;
2  double Gauss(vector<vector<double>> &d) {
3    int n = d.size(), m = d[0].size();
4    double det = 1;
5    for (int i = 0; i < m; i++) {
6      int p = -1;
7      for (int j = i; j < n; j++) {
8        if (fabs(d[j][i]) < EPS) {
9          continue;
10       }
11       if (p == -1 || fabs(d[j][i]) > fabs(d[p][i])) {
12         p = j;
13       }
14     }
15     if (p == -1) {
16       continue;
17     }
18     if (p != i) {
19       det *= -1;
20     }
21     for (int j = 0; j < m; j++) {
22       swap(d[p][j], d[i][j]);
23     }
24     for (int j = 0; j < n; j++) {
25       if (i == j) {
26         continue;
27       }
28       double z = d[j][i] / d[i][i];
29       for (int k = 0; k < m; k++) {
30         d[j][k] -= z * d[i][k];
31       }
32     }
33   }
34   for (int i = 0; i < n; i++) {
35     det *= d[i][i];
36   }
37   return det;
38 }
39 // new
40 const int MAXN = 300;
```

```
41  const double EPS = 1e-8;
42  int n;
43  double A[MAXN][MAXN];
44  void Gauss() {
45    for (int i = 0; i < n; i++) {
46      bool ok = 0;
47      for (int j = i; j < n; j++) {
48        if (fabs(A[j][i]) > EPS) {
49          swap(A[j], A[i]);
50          ok = 1;
51          break;
52        }
53      }
54      if (!ok) continue;
55      double fs = A[i][i];
56      for (int j = i + 1; j < n; j++) {
57        double r = A[j][i] / fs;
58        for (int k = i; k < n; k++) {
59          A[j][k] -= A[i][k] * r;
60        }
61      }
62    }
63  }
```

8.4 Prime Table

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

8.5 Phi

- 歐拉函數計算對於一個整數 N，小於等於 N 的正整數中，有幾個和 N 互質

- 如果 $gcd(p, q) = 1, \Phi(p) \cdot \Phi(q) = \Phi(p \cdot q)$

- $\Phi(p^k) = p^{k-1} \times (p - 1)$

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

8.6 Chinese Remainder Thm

```
1   //参数可为负数的扩展欧几里德定理
2   void exOJLD(int a, int b, int& x, int& y) {
3     //根据欧几里德定理
4     if (b == 0) {  //任意数与0的最大公约数为其本身。
5       x = 1;
6       y = 0;
7     } else {
8       int x1, y1;
9       exOJLD(b, a % b, x1, y1);
10      if (a * b < 0) {  //异号取反
11        x = -y1;
12        y = a / b * y1 - x1;
13      } else {  //同号
14        x = y1;
15        y = x1 - a / b * y1;
16      }
17    }
18  }
19  //剩余定理
20  int calSYDL(int a[], int m[], int k) {
21    int N[k];      //这个可以删除
22    int mm = 1;    //最小公倍数
23    int result = 0;
24    for (int i = 0; i < k; i++) {
25      mm *= m[i];
26    }
27    for (int j = 0; j < k; j++) {
28      int L, J;
29      exOJLD(mm / m[j], -m[j], L, J);
30      N[j] = m[j] * J + 1;    // 1
31      N[j] = mm / m[j] * L;   // 2
          1和2这两个值应该是相等的。
32      result += N[j] * a[j];
33    }
34    return (result % mm + mm) % mm;
35    //落在(0, mm)之间，这么写是为了防止result初始为负数
36    //本例中不可能为负可以直接
37    //写成：return result%mm;即可。
38  }
39
40  int main() {
41    int a[3] = {2, 3, 6};  // a[i]=n%m[i]
42    int m[3] = {3, 5, 7};
43    cout << calSYDL(a, m, 3) << endl;
44    //輸出為滿足兩條陣列的最小n,第3參數為陣列長度
45    //所有滿足答案的數字集合為n+gcd(m0,m1,m2...)*k,
          k為正數
46    return 0;
47  }
```

8.7 Josephus

```
1   int josephus(int n, int k) {  //
        有n個人圍成一圈,每k個一次
2     return n > 1 ? (josephus(n - 1, k) + k) % n : 0;
3   }  // 回傳最後一人的編號, 0 index
```

8.8 Catalan

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

```
1   long long f[N] = {1}, i, t, p;
2   int main() {
3     for (int i = 1; i <= 100; i++) {
4       f[i] = f[i - 1] * (4 * i - 2) % mod;
5       for (t = i + 1, p = mod - 2; p; t = (t * t) %
            mod, p >>= 1LL) {
6         if (p & 1) {
7           f[i] *= t;
```

```
8          f[i] %= mod;
9        }
10      }
11    }
12 }
```

8.9 Matrix Multiplication

```
1  struct Matrix {
2    int row, col;
3    vector<vector<int>> v;
4    Matrix() : row(0), col(0) {}
5    Matrix(int r, int c) : row(r), col(c) {
6      v = vector<vector<int>>(r, vector<int>(c, 0));
7    }
8  };
9  Matrix operator * (Matrix &a, Matrix &b) {
10   assert(a.col == b.row);
11   Matrix ret(a.row, b.col);
12   for (int i = 0; i < a.row; i++) {
13     for (int j = 0; j < b.col; j++) {
14       for (int k = 0; k < a.col; k++) {
15         ret.v[i][j] += a.v[i][k] * b.v[k][j];
16       }
17     }
18   }
19   return ret;
20 }
21 Matrix mPow(Matrix a, int n) {
22   assert(a.row == a.col);
23   Matrix ret(a.row, a.col);
24   ret.v[0][0] = ret.v[1][1] = 1;
25   while (n > 0) {
26     if (n & 1) {
27       ret = ret * a;
28     }
29     a = a * a;
30     n >>= 1;
31   }
32   return ret;
33 }
```

8.10 Fibonacci

$$f(n) = f(n-1) + f(n-2)$$

$$\begin{bmatrix} f(n) \\ f(n-1) \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^{(n-1)} \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

$$O(logn)$$

```
1  LL fib(int n) {
2    if (n <= 1) {
3      return n;
4    }
5    Matrix a(2, 2), b(2, 1);
6    a.v[0][0] = a.v[0][1] = a.v[1][0] = 1;
7    b.v[0][0] = 1;
8    auto t = mPow(a, n - 1);
9    t = t * b;
10   return t.v[0][0];
11 }
```

9 Geometry

9.1 Point

```
1  // notice point type!!!
2  using dvt = int;
3  const double EPS = 1e-6;
4  const double PI = acos(-1);
5
6  struct Pt {
7    dvt x;
8    dvt y;
9  };
10 bool operator < (const Pt &a, const Pt &b) {
11   return a.x == b.x ? a.y < b.y : a.x < b.x;
12 }
13 bool operator == (const Pt &a, const Pt &b) {
14   return a.x == b.x && a.y == b.y;
15 }
16 Pt operator + (const Pt &a, const Pt &b) {
17   return {a.x + b.x, a.y + b.y};
18 }
19 Pt operator - (const Pt &a, const Pt &b) {
20   return {a.x - b.x, a.y - b.y};
21 }
22 // multiply constant
23 Pt operator * (const Pt &a, const dvt c) {
24   return {a.x * c, a.y * c};
25 }
26 Pt operator / (const Pt &a, const dvt c) {
27   return {a.x / c, a.y / c};
28 }
29 // |a| x |b| x cos(x)
30 dvt iproduct(const Pt &a, const Pt &b) {
31   return a.x * b.x + a.y * b.y;
32 }
33 // |a| x |b| x sin(x)
34 dvt cross(const Pt &a, const Pt &b) {
35   return a.x * b.y - a.y * b.x;
36 }
37 dvt dis_pp(const Pt &a, const Pt, &b) {
38   dvt dx = a.x - b.x;
39   dvt dy = a.y - b.y;
40   return sqrt(dx * dx, dy * dy);
41 }
```

9.2 Line

$$d(P, L) = \frac{|ax_0 + by_0 + c|}{\sqrt{a^2 + b^2}}$$

```
1  struct Line {
2    Pt st;
3    Pt ed;
4  };
5  // return point side
6  // left, on line, right -> 1, 0, -1
7  int side(Line l, Pt a) {
8    dvt cross_val = cross(a - l.st, l.ed - l.st);
9    if (cross_val > EPS) {
10     return 1;
11   } else if (cross_val < -EPS) {
12     return -1;
13   } else {
14     return 0;
15   }
16 }
17 // AB infinity, CD segment
18 bool has_intersection(Line AB, Line CD) {
19   int c = side(AB, CD.st);
20   int d = side(AB, CD.ed);
21   if (c == 0 || d == 0) {
22     return true;
23   } else {
24     // different side
25     return c == -d;
26   }
27 }
28 // find intersection point, two line, not seg
29 pair<int, Pt> intersection(Line a, Line b) {
```

```
30    Pt A = a.ed - a.st;
31    Pt B = b.ed - b.st;
32    Pt C = b.st - a.st;
33    dvt mom = cross(A, B);
34    dvt son = cross(C, B);
35    if (std::abs(mom) <= EPS) {
36      if (std::abs(son) <= EPS) {
37        return {1, {}}; // same line
38      } else {
39        return {2, {}}; // parallel
40      }
41    } else {                // ok
42      return {0, a.st + A * (son / mom)};
43    }
44 }
45 // line to point distance
46 dvt dis_lp(Line l, Pt a) {
47    return area3x2(l.st, l.ed, a) / dis_pp(l.st, l.ed);
48 }
```

```
17      m--;
18    }
19    ret[m++] = a[i];
20  }
21  if (sz > 1) {
22    m--;
23  }
24  ret.resize(m);
25  return ret;
26 }
```

9.3 Area

```
1 // triangle
2 dvt area3(Pt a, Pt b, Pt c) {
3    return std::abs(cross(b - a, c - a) / 2);
4 }
5 dvt area3x2(Pt a, Pt b, Pt c) { // for integer
6    return std::abs(cross(b - a, c - a));
7 }
8 // simple convex area(can in)
9 dvt area(vector<Pt> &a) {
10   dvt ret = 0;
11   for (int i = 0, sz = a.size(); i < sz; i++) {
12     ret += cross(a[i], a[(i + 1) % sz]);
13   }
14   return std::abs(ret) / 2;
15 }
16 // check point in/out a convex
17 int io_convex(vector<Pt> convex, Pt q) {
18   // convex is Counterclockwise
19   for (int i = 0, sz = convex.size(); i < sz; i++) {
20     Pt cur = convex[i] - q;
21     Pt nex = convex[(i + 1) % sz] - q;
22     dvt cross_val = cross(cur, nex);
23     if (std::abs(cross_val) <= EPS) {
24       return 0;  // on edge
25     }
26     if (cross_val < 0) {
27       return -1; // outside
28     }
29   }
30   return 1;      // inside
31 }
```

9.4 Convex Hull

```
1 vector<Pt> convex_hull(vector<Pt> &a) {
2    sort(a.begin(), a.end());
3    a.erase(unique(a.begin(), a.end()), a.end());
4    int sz = a.size(), m = 0;
5    vector<Pt> ret(sz + 5); // safe 1 up
6    for (int i = 0; i < sz; i++) {
7      while (m > 1 &&
8        cross(ret[m - 1] - ret[m - 2], a[i] - ret[m -
              2]) <= EPS) {
9        m--;
10     }
11     ret[m++] = a[i];
12   }
13   int k = m;
14   for (int i = sz - 2; i >= 0; i--) {
15     while (m > k &&
16       cross(ret[m - 1] - ret[m - 2], a[i] - ret[m -
              2]) <= EPS) {
```