Contents

Huy

Add and query median

```
/**
 * Author: hieplpvip
 * Date: 2020-10-17
 * License: CC0
 * Source: own work
 * Description: Add integers and query median
 * Time: O(\log N)
 * Status: stress-tested
 */
#pragma once

struct MedianHeap {
  priority_queue<int> minPQ, maxPQ;
  bool empty() { return minPQ.empty(); }
  int top() { return -minPQ.top(); }
  int pop() {
    if (minPQ.empty()) return -1;
    int m = -minPQ.top(); minPQ.pop();
    if (minPQ.size() < maxPQ.size()) {
      minPQ.push(-maxPQ.top());
      maxPQ.pop();
    }
    return m;
  }
  void push(int c) {
    if (!minPQ.empty() && c > -minPQ.top()) {
      minPQ.push(-c);
      if (minPQ.size() > maxPQ.size() + 1) {
        maxPQ.push(-minPQ.top());
        minPQ.pop();
      }
    } else {
      maxPQ.push(c);
      if (maxPQ.size() > minPQ.size()) {
        minPQ.push(-maxPQ.top());
        maxPQ.pop();
      }
    }
  }
};
```

OrderStatisticTree.h: A set (not multiset!) with support for finding the n'th

```
/**
 * Author: Simon Lindholm
 * Date: 2016-03-22
 * License: CC0
 * Source: hacKIT, NWERC 2015
 * Description: A set (not multiset!) with support for finding the n'th
 * element, and finding the index of an element.
 * To get a map, change \texttt{null\_type}.
 * Time: O(\log N)
 */
#pragma once

#include <bits/extc++.h> /** keep-include */
using namespace __gnu_pbds;

template<class T>
using Tree = tree<T, null_type, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;

void example() {
  Tree<int> t, t2; t.insert(8);
  auto it = t.insert(10).first;
  assert(it == t.lower_bound(9));
  assert(t.order_of_key(10) == 1);
  assert(t.order_of_key(11) == 2);
  assert(*t.find_by_order(0) == 8);
  t.join(t2); // assuming T < T2 or T > T2, merge t2 into t
}
```

HullOptimization.h

```
/**
 * Author: hieplpvip
 * Date: 2020-10-17
 * License: CC0
 * Source: own work
 * Description: Add line in decreasing slope, query in increasing x
 * Time: O(\log N)
 * Status: untested
 */
#pragma once

template<typename T = long long>
struct MinHull {
  struct Line {
    T a, b;
    Line(T a, T b): a(a), b(b) {}
    T calc(T x) { return a * x + b; }
  };
  vector<Line> dq;
  size_t seen;
  bool overlap(Line &p1, Line &p2, Line &p3) {
    return 1.0 * (p3.b - p1.b) / (p1.a - p3.a) <= 1.0 * (p2.b - p1.b) / (p1.a
- p2.a);
  }
  void addLine(T a, T b) {
    Line newLine(a, b);
    while (dq.size() > seen + 1 && overlap(dq[(int)dq.size() - 2], dq.back(),
newLine))
      dq.pop_back();
    dq.pb(newLine);
  }
  T query(T x) {
    // change >= to <= this to get MaxHull
    while (seen + 1 < dq.size() && dq[seen].calc(x) >= dq[seen + 1].calc(x))
      ++seen;
    return dq[seen].calc(x);
  }
};
```

RMQ.h

```
/**
 * Author: Johan Sannemo, pajenegod
 * Date: 2015-02-06
 * License: CC0
 * Source: Folklore
 * Description: Range Minimum Queries on an array. Returns
 * min(V[a], V[a + 1], ... V[b - 1]) in constant time.
 * Usage:
 *  RMQ rmq(values);
 *  rmq.query(inclusive, exclusive);
 * Time: $O(|V| \log |V| + Q)$
 * Status: stress-tested
 */
#pragma once

template<class T>
struct RMQ {
  vector<vector<T>> jmp;
  RMQ(const vector<T>& V) : jmp(1, V) {
    for (int pw = 1, k = 1; pw * 2 <= sz(V); pw *= 2, ++k) {
      jmp.emb(sz(V) - pw * 2 + 1);
      rep(j,0,sz(jmp[k]))
        jmp[k][j] = min(jmp[k - 1][j], jmp[k - 1][j + pw]);
    }
  }
  T query(int a, int b) {
    assert(a < b); // or return inf if a == b
    int dep = 31 - __builtin_clz(b - a);
    return min(jmp[dep][a], jmp[dep][b - (1 << dep)]);
  }
};
```

UnionFindRollback.h:

```
/**
 * Author: Lukas Polacek, Simon Lindholm
 * Date: 2019-12-26
 * License: CC0
 * Source: folklore
 * Description: Disjoint-set data structure with undo.
 * If undo is not needed, skip st, time() and rollback().
 * Usage: int t = uf.time(); ...; uf.rollback(t);
 * Time: $O(\log(N))$
 * Status: tested as part of DirectedMST.h
 */
#pragma once

struct RollbackUF {
  vi e; vector<pii> st;
  RollbackUF(int n) : e(n, -1) {}
  int size(int x) { return -e[find(x)]; }
  int find(int x) { return e[x] < 0 ? x : find(e[x]); }
  int time() { return sz(st); }
  void rollback(int t) {
    for (int i = time(); i --> t;)
      e[st[i].first] = st[i].second;
    st.resize(t);
  }
  bool join(int a, int b) {
    a = find(a), b = find(b);
    if (a == b) return false;
    if (e[a] > e[b]) swap(a, b);
    st.emb(a, e[a]); st.emb(b, e[b]);
    e[a] += e[b]; e[b] = a;
    return true;
  }
};
```

MatrixInverse.h

```
/**
 * Author: Max Bennedich
 * Date: 2004-02-08
 * Description: Invert matrix $A$. Returns rank; result is stored in $A$
unless singular (rank < n).
 * Can easily be extended to prime moduli; for prime powers, repeatedly
 * set $A^{-1} = A^{-1} (2I - AA^{-1})\  (\text{mod }p^k)$ where $A^{-1}$
starts as
 * the inverse of A mod p, and k is doubled in each step.
 * Time: O(n^3)
 * Status: Slightly tested
 */
#pragma once

int matInv(vector<vector<double>>& A) {
  int n = sz(A); vi col(n);
  vector<vector<double>> tmp(n, vector<double>(n));
  rep(i,0,n) tmp[i][i] = 1, col[i] = i;

  rep(i,0,n) {
    int r = i, c = i;
    rep(j,i,n) rep(k,i,n)
      if (fabs(A[j][k]) > fabs(A[r][c]))
        r = j, c = k;
    if (fabs(A[r][c]) < 1e-12) return i;
    A[i].swap(A[r]); tmp[i].swap(tmp[r]);
    rep(j,0,n)
      swap(A[j][i], A[j][c]), swap(tmp[j][i], tmp[j][c]);
    swap(col[i], col[c]);
    double v = A[i][i];
    rep(j,i+1,n) {
      double f = A[j][i] / v;
      A[j][i] = 0;
      rep(k,i+1,n) A[j][k] -= f*A[i][k];
      rep(k,0,n) tmp[j][k] -= f*tmp[i][k];
    }
    rep(j,i+1,n) A[i][j] /= v;
    rep(j,0,n) tmp[i][j] /= v;
    A[i][i] = 1;
  }

  /// forget A at this point, just eliminate tmp backward
  for (int i = n-1; i > 0; --i) rep(j,0,i) {
    double v = A[j][i];
    rep(k,0,n) tmp[j][k] -= v*tmp[i][k];
  }

  rep(i,0,n) rep(j,0,n) A[col[i]][col[j]] = tmp[i][j];
  return n;
}
```

Wavelet Tree

```
/**
 * Author: rachitiitr
 * Date: 2022-10-11
 * Source: https://codeforces.com/blog/entry/52854
 * Description: Similar to merge sort tree but faster
 * Time: O(N \log N)
 * Status: untested
 */
#pragma once
```

```
struct wavelet_tree {
#define vi vector<int>
#define pb push_back
  int lo, hi;
  wavelet_tree *l, *r;
  vi b;

  // nos are in range [x,y]
  // array indices are [from, to)
  wavelet_tree(int *from, int *to, int x, int y) {
    lo = x, hi = y;
    if (lo == hi or from >= to) return;
    int mid = (lo + hi) / 2;
    auto f = [mid](int x) {
      return x <= mid;
    };
    b.reserve(to - from + 1);
    b.pb(0);
    for (auto it = from; it != to; it++)
      b.pb(b.back() + f(*it));
    // see how lambda function is used here
    auto pivot = stable_partition(from, to, f);
    l = new wavelet_tree(from, pivot, lo, mid);
    r = new wavelet_tree(pivot, to, mid + 1, hi);
  }

  // kth smallest element in [l, r]
  int kth(int l, int r, int k) {
    if (l > r) return 0;
    if (lo == hi) return lo;
    int inLeft = b[r] - b[l - 1];
    int lb = b[l - 1];  // amt of nos in first (l-1) nos that go in left
    int rb = b[r];      // amt of nos in first (r) nos that go in left
    if (k <= inLeft) return this->l->kth(lb + 1, rb, k);
    return this->r->kth(l - lb, r - rb, k - inLeft);
  }

  // count of nos in [l, r] Less than or equal to k
  int LTE(int l, int r, int k) {
    if (l > r or k < lo) return 0;
    if (hi <= k) return r - l + 1;
    int lb = b[l - 1], rb = b[r];
    return this->l->LTE(lb + 1, rb, k) + this->r->LTE(l - lb, r - rb, k);
  }

  // count of nos in [l, r] equal to k
  int count(int l, int r, int k) {
    if (l > r or k < lo or k > hi) return 0;
    if (lo == hi) return r - l + 1;
    int lb = b[l - 1], rb = b[r], mid = (lo + hi) / 2;
    if (k <= mid) return this->l->count(lb + 1, rb, k);
    return this->r->count(l - lb, r - rb, k);
  }
  ~wavelet_tree() {
    delete l;
    delete r;
  }
};
```

KMP.h

```
/**
 * Author: Johan Sannemo
 * Date: 2016-12-15
 * License: CC0
 * Description: pi[x] computes the length of the longest prefix of s that
ends at x, other than s[0...x] itself (abacaba -> 0010123).
 * Can be used to find all occurrences of a string.
 * Time: O(n)
 * Status: Tested on kattis:stringmatching
 */
#pragma once

vi pi(const string& s) {
  vi p(sz(s));
  rep(i,1,sz(s)) {
    int g = p[i-1];
    while (g && s[i] != s[g]) g = p[g-1];
    p[i] = g + (s[i] == s[g]);
  }
  return p;
}
void compute_automaton(const string& s, vector<vi>& aut) {
  vi p = pi(s);
  aut.assign(sz(s), vi(26));
  rep(i,0,sz(s)) rep(c,0,26)
    if (i > 0 && s[i] != 'a' + c)
      aut[i][c] = aut[p[i - 1]][c];
    else
      aut[i][c] = i + (s[i] == 'a' + c);
}
vi match(const string& s, const string& pat) {
  vi p = pi(pat + '\0' + s), res;
  rep(i,sz(p)-sz(s),sz(p))
    if (p[i] == sz(pat)) res.emb(i - 2 * sz(pat));
  return res;
}
```

Suffix array.h

```
/**
 * Author: 罗穗骞, chilli
 * Date: 2019-04-11
 * License: Unknown
 * Source: Suffix array - a powerful tool for dealing with strings
 * (Chinese IOI National team training paper, 2009)
 * Description: Builds suffix array for a string.
 * \texttt{sa[i]} is the starting index of the suffix which
 * is $i$'th in the sorted suffix array.
 * The returned vector is of size $n+1$, and \texttt{sa[0] = n}.
 * The \texttt{lcp} array contains longest common prefixes for
 * neighbouring strings in the suffix array:
 * \texttt{lcp[i] = lcp(sa[i], sa[i-1])}, \texttt{lcp[0] = 0}.
 * The input string must not contain any zero bytes.
 * Time: O(n \log n)
 * Status: stress-tested
 */
#pragma once

struct SuffixArray {
  vi sa, lcp;
  SuffixArray(string& s, int lim=256) { // or basic_string<int>
    int n = sz(s) + 1, k = 0, a, b;
    vi x(all(s)+1), y(n), ws(max(n, lim)), rank(n);
    sa = lcp = y, iota(all(sa), 0);
    for (int j = 0, p = 0; p < n; j = max(1, j * 2), lim = p) {
      p = j, iota(all(y), n - j);
      rep(i,0,n) if (sa[i] >= j) y[p++] = sa[i] - j;
      fill(all(ws), 0);
      rep(i,0,n) ws[x[i]]++;
      rep(i,1,lim) ws[i] += ws[i - 1];
      for (int i = n; i--;) sa[--ws[x[y[i]]]] = y[i];
      swap(x, y), p = 1, x[sa[0]] = 0;
      rep(i,1,n) a = sa[i - 1], b = sa[i], x[b] =
        (y[a] == y[b] && y[a + j] == y[b + j]) ? p - 1 : p++;
    }
    rep(i,1,n) rank[sa[i]] = i;
    for (int i = 0, j; i < n - 1; lcp[rank[i++]] = k)
      for (k && k--, j = sa[rank[i] - 1];
          s[i + k] == s[j + k]; k++);
  }
};
```

HLD.h

```
 /**
     * Author: Benjamin Qi, Oleksandr Kulkov, chilli
     * Date: 2020-01-12
     * License: CC0
     * Source: https://codeforces.com/blog/entry/53170,
    https://github.com/bqi343/USACO/blob/master/Implementations/content/graphs%20(12)/Tr
    ees%20(10)/HLD%20(10.3).h
     * Description: Decomposes a tree into vertex disjoint heavy paths and light
     * edges such that the path from any leaf to the root contains at most log(n)
     * light edges. Code does additive modifications and max queries, but can
     * support commutative segtree modifications/queries on paths and subtrees.
     * Takes as input the full adjacency list. VALS\_EDGES being true means that
     * values are stored in the edges, as opposed to the nodes. All values
     * initialized to the segtree default. Root must be 0.
     * Time: O((\log N)^2)
     * Status: stress-tested against old HLD
     */
    #pragma once
```

```
#include "../data-structures/LazySegmentTree.h"

template <bool VALS_EDGES> struct HLD {
  int N, tim = 0;
  vector<vi> adj;
  vi par, siz, depth, rt, pos;
  Node *tree;
  HLD(vector<vi> adj_)
    : N(sz(adj_)), adj(adj_), par(N, -1), siz(N, 1), depth(N),
      rt(N),pos(N),tree(new Node(0, N)){ dfsSz(0); dfsHld(0); }
  void dfsSz(int v) {
    if (par[v] != -1) adj[v].erase(find(all(adj[v]), par[v]));
    for (int& u : adj[v]) {
      par[u] = v, depth[u] = depth[v] + 1;
      dfsSz(u);
      siz[v] += siz[u];
      if (siz[u] > siz[adj[v][0]]) swap(u, adj[v][0]);
    }
  }
  void dfsHld(int v) {
    pos[v] = tim++;
    for (int u : adj[v]) {
      rt[u] = (u == adj[v][0] ? rt[v] : u);
      dfsHld(u);
    }
  }
  template <class B> void process(int u, int v, B op) {
    for (; rt[u] != rt[v]; v = par[rt[v]]) {
      if (depth[rt[u]] > depth[rt[v]]) swap(u, v);
      op(pos[rt[v]], pos[v] + 1);
    }
    if (depth[u] > depth[v]) swap(u, v);
    op(pos[u] + VALS_EDGES, pos[v] + 1);
  }
  void modifyPath(int u, int v, int val) {
    process(u, v, [&](int l, int r) { tree->add(l, r, val); });
  }
  int queryPath(int u, int v) { // Modify depending on problem
    int res = -1e9;
    process(u, v, [&](int l, int r) {
        res = max(res, tree->query(l, r));
    });
    return res;
  }
  int querySubtree(int v) { // modifySubtree is similar
    return tree->query(pos[v] + VALS_EDGES, pos[v] + siz[v]);
  }
};
```

Liao tree

```
/**
 * Author: hieplpvip
 * Date: 2020-10-17
 * License: CC0
 * Source: own work
 * Description: Container where you can add segment of the form kx+m, and
query maximum/minimum values at points x.
 *  Useful for dynamic programming (``convex hull trick'').
 * Time: O(\log N)
 * Status: stress-tested
 */
#pragma once

template<typename T, T minX, T maxX, T defVal, bool maximum>
struct DynamicLiChaoTree {
private:
  struct Line {
    T a, b;
    inline T calc(T x) const { return a * x + b; }
  };
  struct Node {
    Line line = {0, maximum ? defVal : -defVal};
    Node *lt = nullptr, *rt = nullptr;
  } *root;
  void update(Node* cur, T l, T r, T u, T v, Line nw) {
    #define newNode(x) if (!x) x = new Node()
    if (v < l || r < u) return;
    T mid = (l + r) >> 1;
    if (u <= l && r <= v) {
      if (cur->line.calc(l) >= nw.calc(l)) swap(cur->line, nw);
      if (cur->line.calc(r) <= nw.calc(r)) {
        cur->line = nw; return;
      }
      if (nw.calc(mid) >= cur->line.calc(mid)) {
        newNode(cur->rt);
        update(cur->rt, mid + 1, r, u, v, cur->line);
        cur->line = nw;
      } else {
        newNode(cur->lt);
        update(cur->lt, l, mid, u, v, nw);
      }
    } else {
      newNode(cur->lt); newNode(cur->rt);
      update(cur->lt, l, mid, u, v, nw);
      update(cur->rt, mid + 1, r, u, v, nw);
    }
    #undef newNode
  }
public:
  DynamicLiChaoTree() { root = new Node(); }
  void add(T a, T b, T l = minX, T r = maxX) {
    if (!maximum) a = -a, b = -b;
    update(root, minX, maxX, l, r, {a, b});
  }
  T query(T x) {
    Node* cur = root;
    T res = cur->line.calc(x), l = minX, r = maxX, mid;
    while (cur) {
      res = max(res, cur->line.calc(x));
      mid = (l + r) >> 1;
      if (x <= mid) cur = cur->lt, r = mid;
      else cur = cur->rt, l = mid + 1;
    }
    return maximum ? res : -res;
  }
};
```

Flow

```
#include<bits/stdc++.h>

using namespace std;



#define y1 as214

#define ii pair < ll , int >

#define iii pair < int , ii >

#define iv pair < ii , ii >



#define fi first

#define se second

#define fr front()

#define pb push_back

#define t top()



#define FOR(i , x , n) for(int i = x ; i <= n ; ++i)

#define REP(i , n) for(int i = 0 ; i < n ; ++i)

#define FORD(i , x , n) for(int i = x ; i >= n ; --i)
```

```
#define ll long long
#define oo 1e17
#define int long long
#define pow poww

const int N = 1e3 + 5;
int n , m , source , sink;
int dist[N] , work[N] , a[N] , b[N];
queue < int > q;

struct edge
{
    int to , rev;
    int flow , cap;
};

vector < edge > g[N];

void addedge(int u , int v , int cap)
{
    edge a = {v , g[v].size() , 0 , cap};
    edge b = {u , g[u].size() , 0 , 0};
    g[u].pb(a);
    g[v].pb(b);
}

bool BFS()
{
    FOR(i , source , sink)
```

```
        dist[i] = -1;
    dist[source] = 0;
    q.push(source);
    while(!q.empty())
    {
        int u = q.fr;
        q.pop();
        REP(s , g[u].size())
        {
            edge e = g[u][s];
            int v = e.to;
            if(e.flow < e.cap && dist[v] < 0)
            {
                dist[v] = dist[u] + 1;
                q.push(v);
            }
        }
    }
    return dist[sink] > 0;
}

int DFS(int u , int f)
{
    if(u == sink)
        return f;
    for(int &s = work[u] ; s < g[u].size() ; ++s)
    {
        edge &e = g[u][s];
        int v = e.to;
        if(e.flow < e.cap && dist[v] == dist[u] + 1)
        {
            int df = DFS(v , min(e.cap - e.flow , f));
            if(df > 0)
            {
                e.flow += df;
                g[v][e.rev].flow -= df;
                return df;
            }
        }
    }
    return 0;
}

int flow()
{
    FOR(i , source , sink)
        g[i].clear();
    addedge(1 , 4 , oo);
    addedge(1 , 6 , oo);
    addedge(2 , 5 , oo);
    addedge(2 , 4 , oo);
    addedge(3 , 6 , oo);
    addedge(3 , 5 , oo);
    FOR(i , 1 , 3)
    {
        addedge(source , i , a[i]);
        addedge(i + 3 , sink , b[i]);
        //cout << a[i] << " " << b[i] << endl;
    }
    int res = 0;
    while(BFS())
    {
        memset(work , 0 , sizeof(work));
        while(int del = DFS(source , oo))
            res += del;
    }
    return res;
}

main()
{
    //freopen("test1.inp","r",stdin);
    ios::sync_with_stdio(0);
    cin.tie(0);
    cin >> n;
    FOR(i , 1 , 3)
        cin >> a[i];
    FOR(i , 1 , 3)
        cin >> b[i];
    source = 0 , sink = 7;
    int maxi = 0;
    maxi += min(a[1] , b[2]);
    maxi += min(a[2] , b[3]);
    maxi += min(a[3] , b[1]);
    cout << n - flow() << " " << maxi << "\n";
}
```