Team reference document

sudo ku

Universiteit Leiden

Inhoudsopgave

Deze TCR is geschreven door Raymond van Bommel <`mailto:raymondvanbommel@gmail.com`>, Josse van Dobben de Bruyn <`mailto:josse.vandobbendebruyn@gmail.com`> en Erik Massop <`mailto:e.massop@hccnet.nl`>. Wij vinden het goed als anderen deze TCR gebruiken, *mits* eventuele verbeteringen met ons gedeeld zijn. Ook dient tekst van gelijke strekking als deze alinea in elke versie aanwezig zijn.

1 Useful tables

maximale complexiteit

Complexiteit	max. waarde n
$\mathcal{O}(n!), \mathcal{O}(n^6)$	10
$\mathcal{O}(2^n \cdot n^2)$	15
$\mathcal{O}(n^4)$	100
$\mathcal{O}(n^3)$	400
$\mathcal{O}(n^2 \lg n)$	2000
$\mathcal{O}(n^2)$	10.000
$\mathcal{O}(n \lg n)$	1000.000
$\mathcal{O}(n)$	100.000.000

maximale grootte

data type	max. waarde
int	$3.27 \cdot 10^4$
unsigned	$6.55 \cdot 10^4$
long	$2.14 \cdot 10^9$
unsigned long	$4.29 \cdot 10^9$
long long	$9.22 \cdot 10^{18}$
unsigned long	$1.84 \cdot 10^{19}$
float	7 digits
double	15 digits

Machten van 2

2-macht	dec. waarde
2^{10}	$1,02 \cdot 10^3$
2^{20}	$1,04 \cdot 10^6$
2^{30}	$1,07 \cdot 10^9$
2^{40}	$1,10 \cdot 10^{12}$
2^{50}	$1,13 \cdot 10^{15}$
2^{60}	$1,15 \cdot 10^{18}$

2 Bomen

2.1 Segment tree

Onderstaande code werkt voor het dynamische Range Minimum Query probleem. Range updates kunnen worden geimplementeerd met lazy propagation: houd bij of een segment volledig naar een bepaalde waarde geüpdated moet worden.

```
struct segment{
    int num;
    int beg, eind;
    int minIndex;
};

class SegmentTree{
public:
    SegmentTree(int n){
        ar.assign(n, inf);
        p.resize(3*n + 5); // to be sure
        p[1].num = 1; p[1].beg = 0; p[1].eind = n-1; p[1].minIndex = 0;
        for(unsigned int i = 2 ; i < p.size() ; i++){
            p[i].num = i;
            if( i%2 == 0 ){
                p[i].beg = p[i/2].beg; p[i].eind = (p[i/2].beg + p[i/2].eind)/2;}
            else{
                p[i].beg = (p[i/2].beg + p[i/2].eind)/2 + 1; p[i].eind = p[i/2].eind;}
            p[i].minIndex = p[i].beg;
        }
    }
    void update(int index, int newValue){
        ar[index] = newValue;
        update(1, index, newValue);
    }
    void update(int num, int index, int newValue){
        if( p[num].beg == p[num].eind )
            return;
        //if( ar[p[num].minIndex] > newValue)
        int mid = (p[num].beg + p[num].eind)/2;
        int num2;
        if( mid >= index )
            num *= 2;
        else
            num = 2*num+1;
        num2 = num^1;
```

```
        update(num, index, newValue);
        p[num/2].minIndex = index;
        if( ar[p[num2].minIndex] < ar[p[num/2].minIndex])
            p[num/2].minIndex = p[num2].minIndex;
        if( ar[p[num].minIndex] < ar[p[num/2].minIndex] )
            p[num/2].minIndex = p[num].minIndex;

    }
    int rmq(int num, int beg, int eind, int a, int b){
        if( beg >= a && eind <= b )
            return p[num].minIndex;
        if( beg > b || eind < a )
            return -1;

        int p1 = rmq(2*num   , beg,       (beg+eind)/2, a, b);
        int p2 = rmq(2*num +1, (beg + eind)/2+1, eind, a, b);
        if(p1 == -1)
            return p2;
        else{
            if( p2 == -1)
                return p1;
            return (ar[p1] < ar[p2] ? p1 : p2);
        }
    }
    int rmq(int a, int b){
        return rmq(1,0,ar.size()-1,a,b);
    }
    vector&lt;segment&gt; p;
    vi ar;
};
```

2.2 Fenwick tree

```
// Fenwick tree/binary indexed tree: dynamic range queries
// find sum of range:        lg(n)
// change value at index:    lg(n)
struct f_tree {
    vl data;
    ll n;       // size

    f_tree(ll n) : n(n) {
        data.assign(n, 0);
    }

    void add(ll idx, ll delta) {     // add to item at idx
        for (; idx < n; idx = idx | (idx + 1))
            data[idx] += delta;
    }

    ll sum(ll idx) {     // compute sum of elements 0,..,r
        ll ret = 0;
        for (; idx >= 0; idx = (idx & (idx + 1)) - 1)
            ret += data[idx];
        return ret;
    }
};
```

2.3 Minimal spanning tree

```
int N, M;
struct edge { int u, v; int weight; } edges[MAX_EDGES];

bool cmp (const edge & e, const edge & f) { return e.weight < f.weight; }

void kruskal (void) {
    sort (edges, edges + M, cmp); // sorteer kanten
    for (int i = 0; i < N; i++) { init (i); } // elke knoop een eigen component

    for (int i = 0; i < M; i++) { // itereer in oplopende volgorde
        int u = representant (edges[i].u), // verkrijg representanten van de
            v = representant (edges[i].v); // componenten van edges[j].{u,v}

        if (u != v) { // als verschillende componenten: merge
            add (edges[i]); // voeg toe aan minimum spanning tree in wording
            merge (u, v);
        }
    }
}
```

Voor een efficiënte implementatie van `init`, `merge` en `representant` is een of andere Disjoint-Set datastructuur handig.

2.4 UFDS

Een Disjoint-Set Forest (met amortized $\mathcal{O}(\alpha(n))$-tijd per operatie, met α de inverse van de Ackermann functie):

```
class UFDS {
private: vi p, rnk;
public:
    UFDS(int N) { rnk.assign(N, 0);
        p.assign(N, 0); for (int i =0 ; i< N; i++) p[i] = i; }
    int findSet(int i) {
        return (p[i] == i) ? i : (p[i] = findSet(p[i])); }
    bool isSameSet(int i, int j) { return findSet(i) == findSet(j); }
    void unionSet(int i, int j) {
        if (!isSameSet(i,j)) {
            int x = findSet(i), y = findSet(j);
            if( rnk[x] > rnk[y])  p[y] = x;
            else {  p[x] = y;
                    if (rnk[x] == rnk[y]} rnk[y]++; }
} } };
```

3 Grafen

3.1 Depth First Search

DFS heeft vele toepassingen. Bij sommige van deze dingen willen we knopen als actief danwel gedaan markeren. Dan kunnen we ook buren overslaan.

- Gerichte cykels: als een reeds actieve knoop wordt ontwikkeld.
- Ongerichte cykels: als een reeds actieve, of voltooide knoop wordt ontwikkeld.

($\mathcal{O}(V+E)$ en $\mathcal{O}(V^2)$)

```
vector<vi> AdjList //Adjacency list

vi dfs_num; //set all values to UNVISITED
```

```
void dfs(int u){
ii v;
dfs_num[u] = VISITED;
for(int j = 0; j < AdjList[u].size(); j++){
v = AdjList[u][j];
if (dfs_num[v] == UNVISITED){
dfs(v);DFS
}   }   }
```

3.2 Breadth first search

($\mathcal{O}(V+E)$ en $\mathcal{O}(V^2)$)

```
vector<vi> AdjList //Adjacency list
vi d(V, INF); d[s] = 0;
queue<int> q; q.push(s);
int u,v;
while(!q.empty()){
u = q.front(); q.pop(); // get the first element from our queue
for(int j = 0; j < AdjList[u].size(); j++){
v = AdjList[u][j]
if(d[v] == INF){ //test if unvisited
d[v] = d[u] + 1; //mark our visit
q.push(v);
}   }   }
```

3.3 Topologisch sorteren

Laat de globale variabele `n` het aantal punten van een graaf zijn. De volgende code registreert de finishvolgorde van de knopen in een globale variable `vector<int> finished`. Hiermee is `finished[n-1]` $<\cdots<$ `finished[0]` een topologische sortering van de knooppunten.

3.4 2-SAT

De code gaat ervanuit dat er `n` knopen zijn, waarbij knopen $2i$ en $2i+1$ elkaars negatie zijn. De functie retourneert of er een oplossing is. Als er een oplossing is, dan is `val[comp[i]]` de valuatie van knoop i in een oplossing.

3.5 Code 2-SAT, SCC en DFS

```
// the ith variable corresponds to nodes 2i and 2i+1 in the graph
// 2i is true if the variable is true, 2i+1 if it is false
struct graph {
    vector<vector<int> > heen, terug; // the arguments of 2sat
    graph(n) : heen(2*n), terug(2*n) {} // graph with nodes for n vars
    int add_clause(int a, int b) { // a & b are nodes, adds clause (a or b)
        heen[a^1].push_back(b); terug[b].push_back(a^1);
        heen[b^1].push_back(a); terug[a].push_back(b^1);
    }
}; // 2sat gives output for both nodes of a variable
```

```
template <typename S, typename F>
void dfs_visit (int cur, vector<vector<int> > const& buren, S start, F finish,
    vector<bool> &mark) {
    if (mark[cur]) { return; }
    mark[cur] = true;
```

```

    start (cur);
    for (auto nb : buren[cur]) {
        dfs_visit (nb, buren, start, finish, mark);
    }
    finish (cur);
}
```

```
vector<int> finish_sort(vector<vector<int> > const& buren) {
    vector<int> finished; vector<bool> mark(buren.size());
    int n = buren.size();
    for (int i = 0; i < n; i++) {
        dfs_visit (i, buren, [](int){},
            [&](int cur){finished.push_back(cur);}, mark);
    }
    return finished;
}
```

```
pair<int, vector<int> >
 scc_and_top_sort(vector<vector<int> > const& heen, vector<vector<int> > const& terug) {
    vector<int> comp(heen.size());
    vector<bool> mark(heen.size(),false); int comp_n = 0;

    auto finished = finish_sort(heen);
    for (auto it = finished.rbegin(); it != finished.rend(); it++) {
        if (mark[*it]) { continue; }
        dfs_visit(*it, terug, [&](int cur){comp[cur] = comp_n;}, [](int){}, mark);
        comp_n++;
    }
    return make_pair(comp_n, comp);
}
```

```
vector<bool> two_sat(vector<vector<int> > const& heen, vector<vector<int> > const& terug)
    int n = heen.size();
    vector<bool> val;
    int comp_n; vector<int> comp;
    vector<int> opp(n);

    tie(comp_n, comp) = scc_and_top_sort(heen, terug);
    for (int i=0; i<n; i++) { opp[comp[i]] = comp[i^1]; }
    for (int i=0; i<comp_n; i++) { if (opp[i] == i) { return val; } }
    vector<bool> cval(comp_n, false);
    for (int i=0; i<comp_n; i++) { if (!cval[i]) { cval[opp[i]] = true; } }
    for (int i=0; i<n; i++) { val.push_back(cval[comp[i]]); }
    return val;
}
```

3.6 Biconnected components

De volgende code deelt de takken van de graaf op in componenten die verbonden blijven als er 1 knoop verwijderd wordt. In het commentaar staat hoe je de punten bepaald die de graaf splitsen als je ze verwijderd.

```
vector<int> buren[MAX_NODES];
bool visited[MAX_NODES];
int low[MAX_NODES];
int d[MAX_NODES];
vector<pair<int, int> > st;

```

```
void mark(int a, int b) {
  pair<int, int> e;
  do {
    e = st.back();
    st.pop_back();
    // doe iets met de tak
  } while((e.first != a || e.second != b) && (e.first != b || e.second != a));
}

void dfs(int n, int parent, int& count) {
  visited[n] = true;
  low[n] = d[n] = ++count;
  for(unsigned i = 0; i <buren[n].size(); ++i) {
    int v = buren[n][i];
    if(!visited[v]) {
      st.push_back(make_pair(n,v));
      dfs(v, n, count);
      if(low[v] >= d[n]) {
        mark(n,v); // als n niet de root is dan n is een cut vertex
      }
      low[n] = min(low[n], low[v]);
    } else if(parent != v && d[v] < d[n]) {
      st.push_back(make_pair(n,v));
      low[n] = min(low[n], d[v]);
    }
  }
  // root == cut vertex <=> als er 2+ kinderen direct vanuit de root visited zijn.
}


void bicon(int N) {
  int count = 0;
  st.clear();
  fill_n(visited, N, false);
  for(unsigned i = 0; i < N; ++i)
    if(!visited[i])
      dfs(i, -1, count);
}
```

3.7 Kortste pad

3.7.1 Dijkstra's algoritme

Dijkstra's algoritme werkt alleen voor grafen met niet-negatieve gewichten. Een korte implementatie die knopen meerdermaals aan de PQ toe kan voegen is:

```
vi dist(V, INF); dist[s] = 0;
priority_queue<ii, vector<ii>, greater<ii> > pq; pq.push(ii(0,s));
while(!pq.empty()) {
    ii front = pq.top(); pq.pop();
    int d = front.first, u = front.second;
    if(d > dist[u]) continue;
    for (int j = 0; j < (int)AdjList[u].size(); j++){
        ii v = AdjList[u][j];
        if(dist[u] + v.second < dist[v.first]) {
            dist[v.first] = dist[u] + v.second;
            pq.push(ii(dist[v.first], v.first));
}   }   }
```

Deze implementatie hoort ook te werken met negatieve gewichten, hoewel langzamer. Pas wel op voor negatieve cycels! met eigen ordening:

```
struct Order
{
    bool operator()(ii const& a, ii const& b) const
    {
        return a.first > b.first;//or any other ordering
    }
};
//...
priority_queue<ii, vector<ii>, Order > pq;
```

3.7.2 Negatieve cykels

Bellman-Ford: relax elke edge V keer (makkelijk te implementeren met edge list). Negatieve-cykeldetectie: kijk of er na het uitvoeren nog iets te relaxen valt.

Kan ook met Floyd-Warshall/

Nota Bene: deze implementatie ondersteunt slechts paden van hoogstens `INT_MAX/4` !

```
template<class DistType>
struct pijl {
    unsigned a,b;
    DistType l; // signed!
};

template<class DistType>
bool bellmanford (unsigned s, unsigned n, vector<pijl<DistType>> const& pijlen,
        vector<DistType> &dist) {
    unsigned m = pijlen.size();

    dist.clear();
    dist.resize(n, numeric_limits<DistType>::max()/2);
    dist[s] = 0;

    for (unsigned i = 0; i < n; i++) {
        for (unsigned j = 0; j < m; j++) {
            dist[pijlen[j].b] = min(dist[pijlen[j].b], dist[pijlen[j].a] + pijlen[j].l);
        }
    }

    // return alleen false bij negatieve kringen die bereikt kunnen worden
    for (unsigned j = 0; j < m; j++) {
        if (dist[pijlen[j].a] < numeric_limits<DistType>::max()/4 &&
             dist[pijlen[j].b] > dist[pijlen[j].a] + pijlen[j].l)
            return false;
    }

    return true;
}
```

3.7.3 Floyd-Warshall

Floyd-Warshall vindt de lengtes van de paden van elke knoop naar elke andere knoop.

```
for k = 1 to n {
    for i = 1 to n {
        for j = 1 to n {
```

```
                dist[i][j] = min (dist[i][j], dist[i][k] + dist[k][j]);
            }
        }
}
```

3.8 Flow netwerken

```
template<class Cap, class Cost>
struct FlowGraph {
    FlowGraph(size_t sz) : out(sz), potential(sz), dist(sz), pred(sz) {}

    void connect(unsigned s, unsigned t, Cap cap, Cap revcap, Cost cost = 0) {
        assert(!revcap || !cost);

        out[s].push_back(edges.size());
        edges.push_back({t, cap, cost});
        out[t].push_back(edges.size());
        edges.push_back({s, revcap, -cost});
    }

    pair<Cap, Cost> ford_fulkerson(unsigned s, unsigned t) { // s != t
        Cap total = 0;
        Cost cost = 0;
        while (find_dijkstra(s, t)) {
            auto flow = numeric_limits<Cap>::max();
            for (auto i = t; i != s; i = edges[pred[i]^1].dest)
                flow = min(flow, edges[pred[i]].cap);

            cost += potential[t] * flow;
            total += flow;

            for (auto i = t; i != s; i = edges[pred[i]^1].dest) {
                edges[pred[i]].cap -= flow;
                edges[pred[i]^1].cap += flow;
            }
        }
        return {total, cost};
    }

private:
    using halfpijl = pair<Cost, unsigned>;
    bool find_dijkstra(unsigned s, unsigned t) {
        dist = numeric_limits<Cost>::max()/2;
        dist[s] = 0;
        pred[t] = t;
        priority_queue<halfpijl, vector<halfpijl>, greater<halfpijl> > q;
        q.push(make_pair(0, s));

        while (!q.empty()) {
            halfpijl p = q.top(); q.pop();
            if (p.first > dist[p.second]) continue;
            for (auto e : out[p.second]) {

                int c = edges[e].cost + potential[p.second] - potential[edges[e].dest];
                if (edges[e].cap && dist[edges[e].dest] > p.first + c) {
                    pred[edges[e].dest] = e;
                    dist[edges[e].dest] = p.first + c;
```

```
                    q.push(make_pair(p.first + c,  edges[e].dest));
                }
            }
        }
        if(pred[t] == t)
            return false;

        potential += dist;
        return true;
    }

    vector<vector<unsigned>> out;
    struct Edge {
        unsigned dest;
        Cap cap;
        Cost cost;
    };
    vector<Edge> edges;

    valarray<Cost> potential, dist;
    vector<unsigned> pred;
};
```

Als je vanaf het begin al takken met negatieve kosten en strikt positieve capaciteit hebt moet je `potential` initialiseren met Bellman-Ford. Dit algoritme ondersteunt geen negatieve cykels.

3.9 Max-flow

De Edomond-Karp implementatie van Ford-Fulkerson (mbv BFS)

```
int res[MAX_V][MAX_V], mf, f, s, t;                          // global variables
vi p;

void augment(int v, int minEdge) {      // traverse BFS spanning tree from s to t
  if (v == s) { f = minEdge; return; }  // record minEdge in a global variable f
  else if (p[v] != -1) { augment(p[v], min(minEdge, res[p[v]][v])); // recursive
                         res[p[v]][v] -= f; res[v][p[v]] += f; }       // update
}


//inside main()
mf = 0;                                                   // mf stands for max_flow
while (1) {               // O(VE^2) (actually O(V^3E) Edmonds Karp's algorithm
    f = 0;
    // run BFS, compare with the original BFS shown in Section 4.2.2
    vi dist(MAX_V, INF); dist[s] = 0; queue<int> q; q.push(s);
    p.assign(MAX_V, -1);            // record the BFS spanning tree, from s to t!
    while (!q.empty()) {
      int u = q.front(); q.pop();
      if (u == t) break;       // immediately stop BFS if we already reach sink t
      for (int v = 0; v < MAX_V; v++)                       // note: this part is slow
        if (res[u][v] > 0 && dist[v] == INF)
          dist[v] = dist[u] + 1, q.push(v), p[v] = u;
    }
    augment(t, INF);      // find the min edge weight 'f' along this path, if any
    if (f == 0) break;       // we cannot send any more flow ('f' = 0), terminate
    mf += f;                  // we can still send a flow, increase the max flow!
}
```

3.10 Min-cut

Als je na max-flow kijk naar alle knopen die je kan berijken vanaf de scoure via positieve ($\neq 0$) takken kan bereiken, heb je een min-cut De min-cut wordt geïnduceerd door de verzameling punten die bereikbaar zijn vanuit de bron. Bedenk dat dit in principe al in de mark array staat na `ford_fulkerson`.

4 Computational Geometry

4.1 Geometry

```
#define INF 1e9
#define EPS 1e-9
#define PI acos(-1.0) // important constant; alternative #define PI (2.0 * acos(0.0))

double DEG_to_RAD(double d) { return d * PI / 180.0; }

double RAD_to_DEG(double r) { return r * 180.0 / PI; }

// struct point_i { int x, y; };     // basic raw form, minimalist mode
struct point_i { int x, y;      // whenever possible, work with point_i
  point_i() { x = y = 0; }                          // default constructor
  point_i(int _x, int _y) : x(_x), y(_y) {} };           // user-defined

struct point { double x, y;    // only used if more precision is needed
  point() { x = y = 0.0; }                          // default constructor
  point(double _x, double _y) : x(_x), y(_y) {}          // user-defined
  bool operator < (point other) const { // override less than operator
    if (fabs(x - other.x) > EPS)                      // useful for sorting
      return x < other.x;           // first criteria , by x-coordinate
    return y < other.y; }           // second criteria, by y-coordinate
  // use EPS (1e-9) when testing equality of two floating points
  bool operator == (point other) const {
   return (fabs(x - other.x) < EPS && (fabs(y - other.y) < EPS)); } };

double dist(point p1, point p2) {                   // Euclidean distance
                      // hypot(dx, dy) returns sqrt(dx * dx + dy * dy)
  return hypot(p1.x - p2.x, p1.y - p2.y); }             // return double

// rotate p by theta degrees CCW w.r.t origin (0, 0)
point rotate(point p, double theta) {
  double rad = DEG_to_RAD(theta);     // multiply theta with PI / 180.0
  return point(p.x * cos(rad) - p.y * sin(rad),
               p.x * sin(rad) + p.y * cos(rad)); }

struct line { double a, b, c; };           // a way to represent a line

// the answer is stored in the third parameter (pass by reference)
void pointsToLine(point p1, point p2, line &l) {
  if (fabs(p1.x - p2.x) < EPS) {                 // vertical line is fine
    l.a = 1.0;    l.b = 0.0;    l.c = -p1.x;            // default values
  } else {
    l.a = -(double)(p1.y - p2.y) / (p1.x - p2.x);
    l.b = 1.0;              // IMPORTANT: we fix the value of b to 1.0
    l.c = -(double)(l.a * p1.x) - p1.y;
} }
```

```
// not needed since we will use the more robust form: ax + by + c = 0 (see above)
struct line2 { double m, c; };      // another way to represent a line

int pointsToLine2(point p1, point p2, line2 &l) {
 if (abs(p1.x - p2.x) < EPS) {           // special case: vertical line
   l.m = INF;                     // l contains m = INF and c = x_value
   l.c = p1.x;                   // to denote vertical line x = x_value
   return 0;   // we need this return variable to differentiate result
 }
 else {
   l.m = (double)(p1.y - p2.y) / (p1.x - p2.x);
   l.c = p1.y - l.m * p1.x;
   return 1;     // l contains m and c of the line equation y = mx + c
} }

bool areParallel(line l1, line l2) {       // check coefficients a & b
  return (fabs(l1.a-l2.a) < EPS) && (fabs(l1.b-l2.b) < EPS); }

bool areSame(line l1, line l2) {           // also check coefficient c
  return areParallel(l1 ,l2) && (fabs(l1.c - l2.c) < EPS); }

// returns true (+ intersection point) if two lines are intersect
bool areIntersect(line l1, line l2, point &p) {
  if (areParallel(l1, l2)) return false;            // no intersection
  // solve system of 2 linear algebraic equations with 2 unknowns
  p.x = (l2.b * l1.c - l1.b * l2.c) / (l2.a * l1.b - l1.a * l2.b);
  // special case: test for vertical line to avoid division by zero
  if (fabs(l1.b) > EPS) p.y = -(l1.a * p.x + l1.c);
  else                  p.y = -(l2.a * p.x + l2.c);
  return true; }

struct vec { double x, y;  // name: 'vec' is different from STL vector
  vec(double _x, double _y) : x(_x), y(_y) {} };

vec toVec(point a, point b) {       // convert 2 points to vector a->b
  return vec(b.x - a.x, b.y - a.y); }

vec scale(vec v, double s) {        // nonnegative s = [<1 .. 1 .. >1]
  return vec(v.x * s, v.y * s); }               // shorter.same.longer

point translate(point p, vec v) {        // translate p according to v
  return point(p.x + v.x , p.y + v.y); }

// convert point and gradient/slope to line
void pointSlopeToLine(point p, double m, line &l) {
  l.a = -m;                                                // always -m
  l.b = 1;                                                  // always 1
  l.c = -((l.a * p.x) + (l.b * p.y)); }                 // compute this

void closestPoint(line l, point p, point &ans) {
  line perpendicular;         // perpendicular to l and pass through p
  if (fabs(l.b) < EPS) {              // special case 1: vertical line
    ans.x = -(l.c);   ans.y = p.y;      return; }

  if (fabs(l.a) < EPS) {            // special case 2: horizontal line
    ans.x = p.x;      ans.y = -(l.c);   return; }

  pointSlopeToLine(p, 1 / l.a, perpendicular);          // normal line
```

```
  // intersect line l with this perpendicular line
  // the intersection point is the closest point
  areIntersect(l, perpendicular, ans); }

// returns the reflection of point on a line
void reflectionPoint(line l, point p, point &ans) {
  point b;
  closestPoint(l, p, b);                          // similar to distToLine
  vec v = toVec(p, b);                                  // create a vector
  ans = translate(translate(p, v), v); }          // translate p twice

double dot(vec a, vec b) { return (a.x * b.x + a.y * b.y); }

double norm_sq(vec v) { return v.x * v.x + v.y * v.y; }

// returns the distance from p to the line defined by
// two points a and b (a and b must be different)
// the closest point is stored in the 4th parameter (byref)
double distToLine(point p, point a, point b, point &c) {
  // formula: c = a + u * ab
  vec ap = toVec(a, p), ab = toVec(a, b);
  double u = dot(ap, ab) / norm_sq(ab);
  c = translate(a, scale(ab, u));                  // translate a to c
  return dist(p, c); }           // Euclidean distance between p and c

// returns the distance from p to the line segment ab defined by
// two points a and b (still OK if a == b)
// the closest point is stored in the 4th parameter (byref)
double distToLineSegment(point p, point a, point b, point &c) {
  vec ap = toVec(a, p), ab = toVec(a, b);
  double u = dot(ap, ab) / norm_sq(ab);
  if (u < 0.0) { c = point(a.x, a.y);                      // closer to a
    return dist(p, a); }         // Euclidean distance between p and a
  if (u > 1.0) { c = point(b.x, b.y);                      // closer to b
    return dist(p, b); }         // Euclidean distance between p and b
  return distToLine(p, a, b, c); }           // run distToLine as above

double angle(point a, point o, point b) {  // returns angle aob in rad
  vec oa = toVec(o, a), ob = toVec(o, b);
  return acos(dot(oa, ob) / sqrt(norm_sq(oa) * norm_sq(ob))); }

double cross(vec a, vec b) { return a.x * b.y - a.y * b.x; }

//// another variant
//int area2(point p, point q, point r) { // returns 'twice' the area of this triangle A-B-
//  return p.x * q.y - p.y * q.x +
//         q.x * r.y - q.y * r.x +
//         r.x * p.y - r.y * p.x;
//}

// returns true if point r is on the same line as the line pq
bool collinear(point p, point q, point r) {
  return fabs(cross(toVec(p, q), toVec(p, r))) < EPS; }

// note: to accept collinear points, we have to change the '> 0'
// returns true if point r is on the left side of line pq
int ccw(point p, point q, point r) {
    if( collinear(p,q,r))
```

```
        return 0;
    return (cross(toVec(p, q), toVec(p, r)) > 0 ? 1 : -1);
}
```

4.2 Projecties

```
coord_t dot (vect u, vect v) { return u.x*v.x + u.y*v.y; }
```

De projectie $p(x, y)$ van $x \in \mathbb{R}^n$ op $y \in \mathbb{R}^n$ wordt gegeven door:

$$p(x,y) = \frac{x \cdot y}{|y|}\hat{y} = \frac{x \cdot y}{|y|^2}y = \frac{x \cdot y}{y \cdot y}y, \qquad \text{waarbij } \hat{y} = \frac{y}{|y|}.$$

Dit is een scalair veelvoud van $\hat{y}$. Let op dat $x \cdot y$ een inproduct is en geen scalair product!

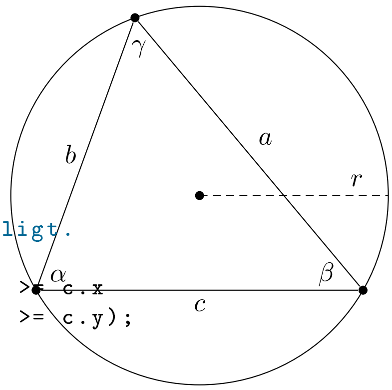
4.3 Rotaties

Om een vector $(x, y) \in \mathbb{R}^2$ precies θ graden in positieve richting te draaien:

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}.$$

4.3.1 Snijden twee lijnstukken?

Er zijn een paar vervelende randgevallen. Onderstaande code (functie `segmentsIntersect`) werkt. Deze functie onderzoekt of de lijnstukken $\overrightarrow{p_1p_2}$ en $\overrightarrow{p_3p_4}$ elkaar snijden. De code werkt ook als de coördinaten in `float` of `double` zijn.




```
// Check of c op het lijnstuk van a naar b ligt,
// gegeven het feit dat c op de lijn door a en b ligt.
bool onSegment(punt a, punt b, punt c) {
    return (min(a.x, b.x) <= c.x && max(a.x, b.x) >= c.x
         && min(a.y, b.y) <= c.y && max(a.y, b.y) >= c.y);
}

bool segmentsIntersect (punt p1, punt p2, punt p3, punt p4) {
    int d1 = ccw(p3, p4, p1);
    int d2 = ccw(p3, p4, p2);
    int d3 = ccw(p1, p2, p3);
    int d4 = ccw(p1, p2, p4);

    if (d1 * d2 < 0 && d3 * d4 < 0) return true;
    if (d1 == 0 && onSegment(p3, p4, p1)) return true;
    if (d2 == 0 && onSegment(p3, p4, p2)) return true;
    if (d3 == 0 && onSegment(p1, p2, p3)) return true;
    if (d4 == 0 && onSegment(p1, p2, p4)) return true;
    return false;
}
```

4.4 Oppervlakte van een veelhoek

$$\text{opp} = \frac{1}{2}\left| \sum_{(x,y)\to(x',y')\in A} xy' - x'y \right|$$

4.5 Zwaartepunt van een veelhoek

Het zwaartepunt kan bepaald worden met

$$C_x = \frac{1}{6A} \sum_{(x,y)\to(x',y')} (x + x')(xy' - x'y)$$

En

$$C_y = \frac{1}{6A} \sum_{(x,y)\to(x',y')} (y + y')(xy' - x'y)$$

Met A de oppervlakte van de veelhoek. Deze methode werkt niet als de veelhoek zichzelf doorsnijdt.

4.6 Polygon

```
// returns the perimeter, which is the sum of Euclidian distances
// of consecutive line segments (polygon edges)
double perimeter(const vector<point> &P) {
  double result = 0.0;
  for (int i = 0; i < (int)P.size()-1; i++)  // remember that P[0] = P[n-1]
    result += dist(P[i], P[i+1]);
  return result; }

// returns the area, which is half the determinant
double area(const vector<point> &P) {
  double result = 0.0, x1, y1, x2, y2;
  for (int i = 0; i < (int)P.size()-1; i++) {
    x1 = P[i].x; x2 = P[i+1].x;
    y1 = P[i].y; y2 = P[i+1].y;
    result += (x1 * y2 - x2 * y1);
  }
  return fabs(result) / 2.0; }

// returns true if we always make the same turn while examining
// all the edges of the polygon one by one
bool isConvex(const vector<point> &P) {
  int sz = (int)P.size();
  if (sz <= 3) return false;   // a point/sz=2 or a line/sz=3 is not convex
  bool isLeft = ccw(P[0], P[1], P[2]);               // remember one result
  for (int i = 1; i < sz-1; i++)            // then compare with the others
    if (ccw(P[i], P[i+1], P[(i+2) == sz ? 1 : i+2]) != isLeft)
      return false;            // different sign -> this polygon is concave
  return true; }                                    // this polygon is convex

// returns true if point p is in either convex/concave polygon P
bool inPolygon(point pt, const vector<point> &P) {
  if ((int)P.size() == 0) return false;
  double sum = 0;    // assume the first vertex is equal to the last vertex
  for (int i = 0; i < (int)P.size()-1; i++) {
    if (ccw(pt, P[i], P[i+1]))
         sum += angle(P[i], pt, P[i+1]);                       // left turn/ccw
    else sum -= angle(P[i], pt, P[i+1]); }                     // right turn/cw
  return fabs(fabs(sum) - 2*PI) < EPS; }

// line segment p-q intersect with line A-B.
point lineIntersectSeg(point p, point q, point A, point B) {
  double a = B.y - A.y;
  double b = A.x - B.x;
```

```
  double c = B.x * A.y - A.x * B.y;
  double u = fabs(a * p.x + b * p.y + c);
  double v = fabs(a * q.x + b * q.y + c);
  return point((p.x * v + q.x * u) / (u+v), (p.y * v + q.y * u) / (u+v)); }

// cuts polygon Q along the line formed by point a -> point b
// (note: the last point must be the same as the first point)
vector<point> cutPolygon(point a, point b, const vector<point> &Q) {
  vector<point> P;
  for (int i = 0; i < (int)Q.size(); i++) {
    double left1 = cross(toVec(a, b), toVec(a, Q[i])), left2 = 0;
    if (i != (int)Q.size()-1) left2 = cross(toVec(a, b), toVec(a, Q[i+1]));
    if (left1 > -EPS) P.push_back(Q[i]);        // Q[i] is on the left of ab
    if (left1 * left2 < -EPS)         // edge (Q[i], Q[i+1]) crosses line ab
      P.push_back(lineIntersectSeg(Q[i], Q[i+1], a, b));
  }
  if (!P.empty() && !(P.back() == P.front()))
    P.push_back(P.front());          // make P's first point = P's last point
  return P; }

point pivot;
bool angleCmp(point a, point b) {                  // angle-sorting function
  if (collinear(pivot, a, b))                                 // special case
    return dist(pivot, a) < dist(pivot, b);     // check which one is closer
  double d1x = a.x - pivot.x, d1y = a.y - pivot.y;
  double d2x = b.x - pivot.x, d2y = b.y - pivot.y;
  return (atan2(d1y, d1x) - atan2(d2y, d2x)) < 0; }   // compare two angles

vector<point> CH(vector<point> P) {   // the content of P may be reshuffled
  int i, j, n = (int)P.size();
  if (n <= 3) {
    if (!(P[0] == P[n-1])) P.push_back(P[0]); // safeguard from corner case
    return P;                           // special case, the CH is P itself
  }

  // first, find P0 = point with lowest Y and if tie: rightmost X
  int P0 = 0;
  for (i = 1; i < n; i++)
    if (P[i].y < P[P0].y || (P[i].y == P[P0].y && P[i].x > P[P0].x))
      P0 = i;

  point temp = P[0]; P[0] = P[P0]; P[P0] = temp;    // swap P[P0] with P[0]

  // second, sort points by angle w.r.t. pivot P0
  pivot = P[0];                    // use this global variable as reference
  sort(++P.begin(), P.end(), angleCmp);              // we do not sort P[0]

  // third, the ccw tests
  vector<point> S;
  S.push_back(P[n-1]); S.push_back(P[0]); S.push_back(P[1]);   // initial S
  i = 2;                                         // then, we check the rest
  while (i < n) {           // note: N must be >= 3 for this method to work
    j = (int)S.size()-1;
    if (ccw(S[j-1], S[j], P[i])) S.push_back(P[i++]);  // left turn, accept
    else S.pop_back(); }   // or pop the top of S until we have a left turn
  return S; }                                            // return the result
```

5 Dynamic Programming

5.1 Knapsack

```
// max value under constraint upper bound on weight
int knapsack( vector<int> value, vector<int> weight, int max_weight ) {
    int n = value.size();
    vector<vector<int>> m( n+1, vector<int>( max_weight+1, 0 ) );
    for( int i = 0; i < n; ++i )
        for( int j = 0; j <= max_weight; ++j )
            if( weight[i] > j )
                m[i+1][j] = m[i][j];
            else
                m[i+1][j] = max( m[i][j], m[i][j-weight[i]] + value[i] );
    return m[n][max_weight];
}
```

6 Strings

6.1 String Matching

```
// s: haystack, w: needle, cb: match callback with word start
template<class F>
void match_string(string const& s, string const& w, F&& cb) {
    assert(!w.empty());
    vector<int> f(w.size() + 1, 0);
    for(unsigned i = 2, c = 0; i <= w.size();) {
        if(w[i-1] == w[c]) f[i++] = ++c;
        else if(c > 0) c = f[c];
        else ++i;
    }
    for(unsigned i = 0, q = 0; i < s.size(); ++i) {
        while(q > 0 && (q == w.size() || s[i] != w[q])) q = f[q];
        if(w[q] == s[i]) ++q;
        if(q == w.size()) cb(i + 1 - w.size());
    }
}
```

6.2 Levenšteinafstand

Het minimal aantal operaties nodig om een string in een andere te transformeren, met toegestane operaties invoeging, verwijdering en subsitutie van karakters. Hieronder is de expressie $d[i,j]$ de Levenšteinafstand tussen $x_1 \cdots x_i$ en $y_1 \cdots y_j$:

$$d[i,j] = \begin{cases} i+j & \text{als } i = 0 \text{ of } j = 0 \\ \min \begin{pmatrix} d[i-1,j]+1, \\ d[i,j-1]+1, \\ d[i-1,j-1]+1_{\{x_i \neq y_j\}} \end{pmatrix} & \text{als } i > 0 \text{ en } j > 0 \end{cases}$$

6.3 DJB2 hash

```
unsigned long hash = 5381;
int c;

while (c = *str++)
        hash = ((hash << 5) + hash) + c; /* hash * 33 + c */
```

7 Getaltheorie

7.1 Priemgetallen

```
void sieve(ll upperbound) {          // create list of primes in [0..upperbound]
  _sieve_size = upperbound + 1;                  // add 1 to include upperbound
  bs.set();                                              // set all bits to 1
  bs[0] = bs[1] = 0;                                  // except index 0 and 1
  for (ll i = 2; i <= _sieve_size; i++) if (bs[i]) {
    // cross out multiples of i starting from i * i!
    for (ll j = i * i; j <= _sieve_size; j += i) bs[j] = 0;
    primes.push_back((int)i);  // also add this vector containing list of primes
} }                                             // call this method in main method

bool isPrime(ll N) {                 // a good enough deterministic prime tester
  if (N <= _sieve_size) return bs[N];                     // O(1) for small primes
  for (int i = 0; i < (int)primes.size(); i++)
    if (N % primes[i] == 0) return false;
  return true;                     // it takes longer time if N is a large prime!
}                          // note: only work for N <= (last prime in vi "primes")^2
```

7.2 Uitgebreide Euclidische algoritme

Met dit algoritme kan zowel de ggd van twee getallen a en b bepaald worden als een paar (x, y) waarvoor geldt $ax + by = \text{ggd}(a, b)$.

```
template<class T>
pair <T, pair <T, T> > uggd(T a, T b) {
    T x, lastx, y, lasty;
    T q;

    x = lasty = 0;
    y = lastx = 1;

    while (b != 0) {
        q = a / b;

        a %= b;
        swap(a,b);

        lastx -= q*x;
        swap(x, lastx);

        lasty -= q*y;
        swap(y, lasty);
    }

    return make_pair (a, make_pair (lastx, lasty));
}
```

```
unsigned ggd(unsigned a, unsigned b) {
    while(b) {
        a %= b;
        swap(a,b);
    }
    return a;
}
```

7.3 CRT

```
template<class T>
T crt(T a, T b, T m, T n){ //assumes m, n are coprime
    pair<T,T> multinv = uggd(m,n).second;
    T x = b*((multinv.first*m)%(m*n)) + a*((multinv.second*n)%(m*n));
    x = (x%(m*n) + m*n)%(m*n);
    return x;
```

```
} // returns 0 <= x < m*n with x = a (mod m), x = b (mod n)
```

7.4 Priemtest

De volgende code implementeert een deterministische versie van Miller-Rabin:

```
__int128_t power(__int128_t a, unsigned long long e, unsigned long long n) {
    if(e == 1) return a;
    if(e & 1) return (a * power(a, e-1, n) % n);
    __int128_t v = power(a, e/2, n);
    return (v*v) % n;
}

bool isprime(unsigned long long n) {
    if(n <= 1) return false;
    unsigned long long d = n-1;
    unsigned s = 0;
    while(d % 2 == 0) { d /= 2; ++s; }
    // BELANGRIJK: typ deze getallen goed over!
    const unsigned a_arr[] = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37};
    for(unsigned i = 0; i < sizeof(a_arr)/sizeof(unsigned) && a_arr[i] < n; ++i) {
        unsigned a = a_arr[i];
        __int128_t v = power(a, d, n);
        if(v == 1) continue;
        bool composite = true;
        for(unsigned p = 0; p < s; ++p) {
            if(v == n-1) { composite = false; break; }
            v = (v * v) % n;
        }
        if(composite) return false;
    }
    return true;
}
```

Dit gebruiken we om te voorkomen dat we grote priemgetallen in Pollards-Rho gaan gooien

```
#define abs_val(a) (((a)>=0)?(a):-(a))
ll mulmod(ll a, ll b, ll c) { // returns (a * b) % c, and minimize overflow
  ll x = 0, y = a % c;
  while (b > 0) {
    if (b % 2 == 1) x = (x + y) % c;
    y = (y * 2) % c;
    b /= 2;
  }
  return x % c;
}

ll gcd(ll a, ll b) { return !b ? a : gcd(b, a % b); }       // standard gcd

ll pollard_rho(ll n) {
  int i = 0, k = 2;
  ll x = 3, y = 3;                // random seed = 3, other values possible
  while (1) {
    i++;
    x = (mulmod(x, x, n) + n - 1) % n;                 // generating function
    ll d = gcd(abs_val(y - x), n);                         // the key insight
    if (d != 1 && d != n) return d;          // found one non-trivial factor
    if (i == k) y = x, k *= 2;
} }

int main() {
```

```
    ll n = 2063512844981574047LL;    // we assume that n is not a large prime
    ll ans = pollard_rho(n);          // break n into two non trivial factors
    if (ans > n / ans) ans = n / ans;          // make ans the smaller factor
    printf("%lld %lld\n", ans, n / ans);  // should be: 1112041493 1855607779
} // return 0;
```

7.5 Partitiefunctie

De partitiefunctie geeft het aantal manieren dat een getal n opgedeeld kan worden in positieve getallen als volgorde niet uitmaakt, zodanig dat de som weer n is.

```
long long parts[P][P]; // [m][n] = # opdelingen van n die overal <= m zijn

long long partition(int upper) {
    parts[0][0] = 1;
    fill_n (parts[0] + 1, upper, 0);
    for (int n = 1; n <= upper; n++) {
        parts[n][0] = 1;
        for (int sum = 1; sum <= upper; sum++) {
            parts[n][sum] = parts[n - 1][sum];
            if (sum >= n)
                parts[n][sum] += parts[n][sum - n];
        }
    }
    return parts[upper][upper];
}
```

Overflow

type	iteraties veilig
s32	121
u32	127
s64	405
u64	416
s128	1437
u128	1458

8 Lineaire stelsels oplossen

We kunnen een stelsel lineaire vergelijkingen oplossen met het vegen van een matrix. De volgende code doet dat als de matrix inverteerbaar is en geeft `false` als de matrix niet inverteerbaar is.

```


bool linsolve(vector<vector<double> > A, vector<double> b, vector<double>& x,
            unsigned rows, unsigned cols) {
    unsigned done = 0;
    for(unsigned i = 0; i < cols; ++i) {
        int r = -1;
        for(unsigned j = done; j < rows; ++j)
            if(fabs(A[j][i]) > 1e-12) { r = j; break; }
        if(r == -1) continue;
        swap(A[done], A[r]);
        swap(b[done], b[r]);
        double dd = 1.0 / A[done][i];
        for(unsigned j = 0; j < cols; ++j)
            A[done][j] *= dd;
        b[done] *= dd;
        for(unsigned j = 0; j < rows; ++j) {
            if(done == j) continue;
            double d = A[j][i] / A[done][i];
            for(unsigned k = 0; k < cols; ++k)
                A[j][k] -= d * A[done][k];
            b[j] -= d* b[done];
        }
        done++;
    }
```

```
    if(done == cols && rows == cols) {
        for(unsigned i = 0; i < cols; ++i)
            x[i] = b[i] / A[i][i];
        return true;
    } else
        return false;
}
```

9 Tips

9.1 Mogelijke algoritmes, inspiratie

- Probeer te denken vanuit de grenzen.
- Is het te doorzoeken met min/max binair/ternair zoeken? (monotoon?)
- Brute Force (met pruning)
- Brute Force small instances to discover patterns or test algorithm
- Probeer Greedy
- Dynamic Programming
- (Mincost)Maxflow
- longest increasing subsequence
- Kan je de graaf bipartiet krijgen?
- Precalculeren
- Inclusion/Exclusion
- Neigt het naar NIM?
- Line sweep / radial sweep
- Coordinaten comprimeren
- Vervang strings met iets snellers.

9.2 Bugs

- Maak je aannamens die niet in de probleembeschrijving staan?
- Kijk uit voor variabelen met dezelfde naam.
- Geef je altijd de juiste uitvoer? (e.g.: geen pad, print ¨impossible¨)
- Ergens een overflow? Array out of bounds?
- *Gebruik de juiste epsilons om doubles te vergelijken.*
- Zijn er belangrijke regels in commentaar gezet?
- Gebruik je overal groot genoege variablen (long long)? Ook bij bit operations?
- Lees het probleem nog eens (is de input gesoorteerd of niet?).
- Laat een teamgenoot het probleem herlezen.
- Maak je eigen testcases (wat gebeurt er als de input ergens 0 is?).
- Controleer je of je nodes al hebt bezocht?

9.3 Minijudge

```
#!/bin/bash

TESTF=~/final_samples

c++ -g -o $1 $1.cc || {
    echo "COMPILE ERROR"
    exit
}

for filen in $TESTF/$2/*.in; do
    ./$1 < $filen > out.tst || {
        echo "RUN ERROR"
        exit
    }
    diff out.tst "${filen%.in}.ans" > /dev/null 2>/dev/null || {
        echo "Output":
        cat $filen
```

```
        echo "Diff:"
        diff out.tst "${filen%.in}.ans" -y
    }
    rm ./out.tst
done
```

10 Toevoegingen sudo ku

10.1 Binairy search

binairy search (voor een funcie *can* die *false* is voor $x < ans$ en *true* voor $x \geq ans$)

```
bool can(double x){
//test is x is a solution (or larger that the solution)
}
...
//instide main()
double lo = 0.0, hi = 100000.0, mid = 0.0, ans = 0.0;
double eps = 0.01;

while(fabs(hi-lo) > eps){
mid = (lo + hi) / 2.0;
if(can(mid)) {ans = mid; high = mid;}
//save the answer, look for smaller solution
else {lo = mid;} //look for larger solution
}
```

sort using a custom function object

```
struct {
bool operator()(int a, int b) const{
return a < b;
}
} customLess;

array<int, 10> s = {5, 7, 4, 2, 8, 6, 1, 9, 0, 3};
sort(s.begin(), s.end(), customLess);
```

10.2 Longest increasing subsequence

```
void find_lis(vector<int> &a, vector<int> &b) {
    vector<int> p(a.size());
    int u, v;
    if (a.empty()) return;
    b.push_back(0);
    for (size_t i = 1; i < a.size(); i++) {
        if (a[b.back()] < a[i]) {    //If next element a[i] is greater than last
            p[i] = b.back();         //   element of current longest subsequence
            b.push_back(i);          // a[b.back()], just push it at back of "b"
            continue;
        }
        // search for smallest element referenced by b  just bigger than a[i]
        for (u = 0, v = b.size()-1; u < v;) {
            int c = (u + v) / 2;
            if (a[b[c]] < a[i]) u=c+1; else v=c;
        }
        if (a[i] < a[b[u]]) {                // Update b if new value is smaller
            if (u > 0) p[i] = b[u-1];        // then previously referenced value
            b[u] = i;
```

```
        }
    }
    for (u = b.size(), v = b.back(); u--; v = p[v]) b[u] = v;
}
```

10.3 Code snippets

vimrc

```
execute pathogen#infect()
filetype plugin indent on

set number
set tabstop=4
set shiftwidth=4
```

Template

```
#include <bits/stdc++.h>
using namespace std;

#define pb push_back
#define eb emplace_back
#define mp make_pair
#define pq priority_queue

using ll = long long;        // integral value type
using ff = long double;      // floating point type
using vl = vector<ll>;
using pl = pair<ll, ll>;
using vpl = vector<pl>;

int main() {
    ios::sync_with_stdio(false);
    cin.tie(NULL);
    cout << fixed << setprecision(10);

    ll N; cin >> N;
}
```

zet alle elementen ven een (multi-dimentionale) array naar *value*:

```
memset(array, value, sizeof array);
```

of voor longs

```
void memset64( void * dest, uint64_t value, uintptr_t size ){
    uintptr_t i;
    for( i = 0; i < (size & (~7)); i+=8 ){
        memcpy( ((char*)dest) + i, &value, 8 );
    }
    for( ; i < size; i++ ){
        ((char*)dest)[i] = ((char*)&value)[i&7];
}   }
```

geef eendecimale precisie voor cout ($n = 4$ geeft $\pi = 3.1415$)

```
cout << setprecision(n);
```

Bekijk alle permutaties van een array, in compleciteit $\mathcal{O}(n!)$:

```
int n = 4;
int p[n] = {0,1,2,3};

```

```
do{
    //your code here
}while(next_permutation(p,p+n));
```

Bekijk alle subset van een array, in compleciteit $\mathcal{O}(2^n)$:

```
//given an array "array" with size "arraySize"
unsigned int i, j, bits, i_max = 1U << arraySize;

for (i = 0; i < i_max ; ++i) {
    for (bits = i, j = 0; bits; bits >>= 1, ++j) {
        if (bits & 1){
            //do something for each element in subset
}   }   }
```

10.4 Bit operations

Een bitset is een meer geheugenvriendelijke boolean array.

```
bitset<8> b1;/* [0,0,0,0,0,0,0,0] */ bitset<8> b2(42); // [0,0,1,0,1,0,1,0]
```

```
int S;
S << 1; //S*2
S << 2; //S*4
S >> 1; //S/2
S >> 2; //S/4
1 << 3; //001000
A | B; //bitwise or
A & B; //bitwise and
A ^ B; //bitwise xor
S |= (1 << j); //ensure jth bit is on
S &= ~(1 << j); //ensure jth bit is off
S ^= (1 << j); //toggle jth bit
S & (1 << j) != 0; //test jth bit
```

10.5 gdb debugger

Compileer je programma met
`g++ -g -Wall -o test test.cc`
Open GDB met
`gdb test`

Dit zijn de belangrijkste gdb commando's:

- **b** break REGELNUMMER / FUNCTIENAAM (stelt een breakpoint in; de executie zal worden onderbroken vóórdat de regel is uitgevoerd)
- **wa** watch VARIABELE / CONDITIE (het programma zal worden onderbroken als de waarde van de variabele wordt veranderd)
- **i b** info breakpoints (geeft een genummerde lijst van alle breakpoints)
- **dis/en** disable/enable BREAKPOINTNUMMER (het breakpoint blijft bestaan onder hetzelfde nummer, maar zal in de executie worden genegeerd)
- **r** run < infile.in > outfile.out (mag ook zonder pipes, dan van stdin/stdout)
- **c** continue (tot volgende breakpoint)
- **s** step (voer de volgende stap uit, gaat zonodig de functie in)
- **n** next (voer de volgende regel code uit)
- **u** until REGELNUMMER (ga door tot de genoemde regel)
- **k** kill (maar blijf in gdb)
- **q** quit (verlaat gdb)
- **ba/bt** backtrace (print een stack trace, zodat je ziet in welke functie je zit)
- **f** frame NUMMER-IN-DE-STACK (verplaatst de scope van gdb naar de gegeven plek in de stack trace)
- **p** print NAAM-VAN-VARIABELE (eerste 10 elementen van array a bekijken: print *a@10)
- **l** list (print de huidige regel broncode met een paar regels context)

10.6 Priemgetallen

6.15. Stelling. *Chinese Reststelling Laat m en n onderling ondeelbare gehele getallen zijn. Dan is de natuurlijke afbeelding*

$$\begin{aligned}\psi : \quad \mathbf{Z}/mn\mathbf{Z} &\xrightarrow{\sim} \mathbf{Z}/m\mathbf{Z} \times \mathbf{Z}/n\mathbf{Z} \\ (x \bmod mn) &\longmapsto (x \bmod m, x \bmod n)\end{aligned}$$

een ringisomorfisme. De afbeelding ψ induceert een isomorfisme van eenhedengroepen

$$\psi_* : (\mathbf{Z}/mn\mathbf{Z})^* \xrightarrow{\sim} (\mathbf{Z}/m\mathbf{Z})^* \times (\mathbf{Z}/n\mathbf{Z})^*,$$

en de Euler-φ-functie voldoet aan $\varphi(mn) = \varphi(m)\varphi(n)$.

6.16. Gevolg. *Voor ieder positief geheel getal n is er een natuurlijk ringisomorfisme*

$$\mathbf{Z}/n\mathbf{Z} \xrightarrow{\sim} \textstyle\prod_{p|n}(\mathbf{Z}/p^{\mathrm{ord}_p(n)}\mathbf{Z}).$$

Voor de Euler-φ-functie geldt dienovereenkomstig

$$\varphi(n) = \prod_{p|n} \varphi(p^{\mathrm{ord}_p(n)}) = \prod_{p|n} (p-1)p^{\mathrm{ord}_p(n)-1} = n \cdot \prod_{p|n}(1 - \frac{1}{p}).$$

6.17. Stelling. *Voor a en $n \geq 1$ onderling ondeelbaar geldt $a^{\varphi(n)} \equiv 1 \bmod n$.* □

Het geval dat n een priemgetal is was al bestudeerd door Fermat (1601–1665), die rond 1640 het volgende speciale geval van 6.17 formuleerde.

6.18. Kleine stelling van Fermat. *Voor p een priemgetal en a een geheel getal geldt*

$$a^p \equiv a \bmod p.$$