# Team notebook

July 1, 2023

## Contents

**1 algorithms** **1**
1.1 binarysearch . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1

**2 data-structures** **1**
2.1 fenwicktree . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1
2.2 segtree . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 2
2.3 simplebittree . . . . . . . . . . . . . . . . . . . . . . . . . . . 4
2.4 ufds . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5

**3 dynamic-programming** **6**
3.1 knapsack1 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6

**4 geometry** **6**
4.1 triangles . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6

**5 graphs** **6**
5.1 bfs . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6
5.2 dfs . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7
5.3 dijkstra . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7
5.4 kruskal . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8

**6 misc** **8**
6.1 functors . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8
6.2 template . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8

**7 number-theory** **9**
7.1 binpow . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 9
7.2 modular-arithmetic . . . . . . . . . . . . . . . . . . . . . . . . . 9
7.3 primefact . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 10
7.4 sieve . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 10

## 1 algorithms

### 1.1 binarysearch

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

int last_true(int lo, int hi, function<bool(int)> f) {
        // if none of the values in the range work, return lo - 1
        lo--;
        while (lo < hi) {
                // find the middle of the current range (rounding up)
                int mid = lo + (hi - lo + 1) / 2;
                if (f(mid)) {
                        // if mid works, then all numbers smaller than mid
                            also work
                        lo = mid;
                } else {
                        // if mid does not work, greater values would not
                            work either
                        hi = mid - 1;
                }
        }
        return lo;
}

//We will need to do the same thing, but when the condition is satisfied,
//we will cut the right part, and when it’s not, the left part will be
    cut.
int first_true(int lo, int hi, function<bool(int)> f) {
        hi++;
        while (lo < hi) {
                int mid = lo + (hi - lo) / 2;
        if (f(mid)) {
            hi = mid;
        } else {
            lo = mid + 1;
        }
    }
    return lo;
}

int main() {
    // all numbers satisfy the condition (outputs 10)
    cout << last_true(2, 10, [](int x) { return true; }) << endl;

    // outputs 5
    cout << last_true(2, 10, [](int x) { return x * x <= 30; }) <<
        endl;

    // no numbers satisfy the condition (outputs 1)
    cout << last_true(2, 10, [](int x) { return false; }) << endl;
}
```

## 2 data-structures

### 2.1 fenwicktree

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

template <typename T>
struct Fenwick {
    int n;
    std::vector<T> a;

    Fenwick(int n = 0) {
        init(n);
    }

    void init(int n) {
        this->n = n;
        a.assign(n, T());
    }

    void add(int x, T v) {
        for (int i = x + 1; i <= n; i += i & -i) {
            a[i - 1] += v;
        }
    }

    T sum(int x) {
        auto ans = T();
        for (int i = x; i > 0; i -= i & -i) {
            ans += a[i - 1];
        }
        return ans;
    }

    T rangeSum(int l, int r) {
        return sum(r) - sum(l);
    }

    int kth(T k) {
        int x = 0;
        for (int i = 1 << std::__lg(n); i; i /= 2) {
            if (x + i <= n && k >= a[x + i - 1]) {
                x += i;
                k -= a[x - 1];
            }
        }
        return x;
    }
};

int main() {
    return 0;
}
```

### 2.2 segtree

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

using namespace std;

struct segtree {
    using T = int;
    using F = int;
```

```
T e() {
    return (int) 1e9;
}

F id() {
    return 0;
}

T op(T a, T b) {
    return min(a, b);
}

T mapping(F f, T x) {
    return f + x;
}

F composition(F f, F g) {
    return f + g;
}

int n;
int size;
int log_size;
vector<T> node;
vector<F> lazy;

segtree() : segtree(0) {}
segtree(int _n) {
    build(vector<T>(_n, e()));
}
segtree(const vector<T>& v) {
    build(v);
}

void build(const vector<T>& v) {
    n = (int) v.size();
    if (n <= 1) {
        log_size = 0;
    } else {
        log_size = 32 - __builtin_clz(n - 1);
    }
    size = 1 << log_size;
    node.resize(2 * size, e());
    lazy.resize(size, id());
    for (int i = 0; i < n; i++) {
        node[i + size] = v[i];
    }
    for (int i = size - 1; i > 0; i--) {
        pull(i);
    }
}

void push(int x) {
    node[2 * x] = mapping(lazy[x], node[2 * x]);
    node[2 * x + 1] = mapping(lazy[x], node[2 * x + 1]);
    if (2 * x < size) {
        lazy[2 * x] = composition(lazy[x], lazy[2 * x]);
        lazy[2 * x + 1] = composition(lazy[x], lazy[2 * x + 1]);
    }
    lazy[x] = id();
}

void pull(int x) {
    node[x] = op(node[2 * x], node[2 * x + 1]);
}

void set(int p, T v) {
    assert(0 <= p && p < n);
    p += size;
    for (int i = log_size; i >= 1; i--) {
        push(p >> i);
    }
    node[p] = v;
    for (int i = 1; i <= log_size; i++) {
        pull(p >> i);
    }
}

T get(int p) {
    assert(0 <= p && p < n);
    p += size;
    for (int i = log_size; i >= 1; i--) {
        push(p >> i);
    }
    return node[p];
}

T get(int l, int r) {
    assert(0 <= l && l <= r && r <= n);
    l += size;
```

```
        r += size;
        for (int i = log_size; i >= 1; i--) {
            if (((l >> i) << i) != l) {
                push(l >> i);
            }
            if (((r >> i) << i) != r) {
                push((r - 1) >> i);
            }
        }
        T vl = e();
        T vr = e();
        while (l < r) {
            if (l & 1) {
                vl = op(vl, node[l++]);
            }
            if (r & 1) {
                vr = op(node[--r], vr);
            }
            l >>= 1;
            r >>= 1;
        }
        return op(vl, vr);
    }

    void apply(int p, F f) {
        assert(0 <= p && p < n);
        p += size;
        for (int i = log_size; i >= 1; i--) {
            push(p >> i);
        }
        node[p] = mapping(f, node[p]);
        for (int i = 1; i <= log_size; i++) {
            pull(p >> i);
        }
    }

    void apply(int l, int r, F f) {
        assert(0 <= l && l <= r && r <= n);
        l += size;
        r += size;
        for (int i = log_size; i >= 1; i--) {
            if (((l >> i) << i) != l) {
                push(l >> i);
            }
            if (((r >> i) << i) != r) {
                push((r - 1) >> i);
            }
        }
        int ll = l;
        int rr = r;
        while (l < r) {
            if (l & 1) {
                node[l] = mapping(f, node[l]);
                if (l < size) {
                    lazy[l] = composition(f, lazy[l]);
                }
                l++;
            }
            if (r & 1) {
                r--;
                node[r] = mapping(f, node[r]);
                if (l < size) {
                    lazy[r] = composition(f, lazy[r]);
                }
            }
            l >>= 1;
            r >>= 1;
        }
        l = ll;
        r = rr;
        for (int i = 1; i <= log_size; i++) {
            if (((l >> i) << i) != l) {
                pull(l >> i);
            }
            if (((r >> i) << i) != r) {
                pull((r - 1) >> i);
            }
        }
    }
};

int main() {
    return 0;
}
```

## 2.3 simplebittree

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

```
using namespace std;

#define LSOne(S) ((S) & -(S))
typedef vector<int> vi;

// Simple version
class Bitree {

private:
    vi ft;
public:
    Bitree(int m) { ft.assign(m+1, 0); }

    int rsq(int j) {
        int sum = 0;
        for(; j; j -= LSOne(j))
            sum += ft[j];
        return sum;
    }

    int rsq(int i, int j) {
        return rsq(j) - rsq(i-1);
    }

    void update(int i, int v) {
        for(; i < (int)ft.size(); i += LSOne(i))
            ft[i] += v;
    }

};

// implementation
int main() {
    Bitree bt(10);
    bt.update(1, 0);
    bt.update(2, 1);
    bt.update(3, 0);
    bt.update(4, 1);
    bt.update(5, 2);
    bt.update(6, 3);
    bt.update(7, 2);
    bt.update(8, 1);
    bt.update(9, 1);
    cout<<"rsq(1): "<<bt.rsq(1)<<"\n";
    cout<<"rsq(2): "<<bt.rsq(2)<<"\n";
    cout<<"rsq(3): "<<bt.rsq(3)<<"\n";
    cout<<"rsq(4): "<<bt.rsq(4)<<"\n";
    cout<<"rsq(5): "<<bt.rsq(5)<<"\n";
    cout<<"rsq(6): "<<bt.rsq(6)<<"\n";
    cout<<"rsq(7): "<<bt.rsq(7)<<"\n";
    cout<<"rsq(8): "<<bt.rsq(8)<<"\n";
    cout<<"rsq(9): "<<bt.rsq(9)<<"\n";
    cout<<LSOne(8)<<"\n";
}
```

## 2.4 ufds

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

using namespace std;

// 1-indexed
class Ufds {

private:
vector<int> ps, size;
int numSets;

public:
    Ufds(int N) {
        ps.assign(N+1, 0); iota(ps.begin(), ps.end(), 0);
        size.assign(N+1, 1);
        numSets = N;
    }

    int findSet(int i) {
        return ps[i] == i ? i : (ps[i] = findSet(ps[i]));
    }

    bool sameSet(int i, int j) {
        return findSet(i) == findSet(j);
    }

    int getSetSize(int i) { return size[findSet(i)]; }
    int getNumSets() { return numSets; }

    // unify two sets
    void unionSet(int i, int j) {
        if(sameSet(i, j)) return;

        int pi = findSet(i);
        int pj = findSet(j);

        if(size[pi] > size[pj]) swap(pi, pj);

        ps[pi] = pj;
        size[pj] += size[pi];

        --numSets;
    }

};

// implementation
int main() {
    Ufds uf(5);
    cout<<"Num of sets: "<<uf.getNumSets()<<"\n";
    uf.unionSet(1, 2);
    cout<<"Num of sets: "<<uf.getNumSets()<<"\n";
    uf.unionSet(3, 4);
    cout<<"Num of sets: "<<uf.getNumSets()<<"\n";
    uf.unionSet(5, 4);
    cout<<"Num of sets: "<<uf.getNumSets()<<"\n";

    cout<<uf.sameSet(1, 4)<<"\n";
    cout<<uf.sameSet(5, 4)<<"\n";

    for(int i=1; i<=5; i++) {
        printf("findSet(%d) = %d, size(%d) = %d\n",
               i, uf.findSet(i), i, uf.getSetSize(i));
    }
}
```

# 3 dynamic-programming

## 3.1 knapsack1

```
#include <bits/stdc++.h>
#define pb(x) push_back(x)
#define all(x) x.begin(),x.end()

using namespace std;
using ll = int64_t;

void solve() {
    int N, W;
    cin>>N>>W;
    vector<int> w(N+1), v(N+1);
    for(int i=1; i<=N; i++) {
        cin>>w[i]>>v[i];
    }

    vector<vector<ll>> dp(N+1, vector<ll>(W+1, 0));

    for(int i=1; i<=N; i++) {
        for(int j=0; j<=W; j++) {
            if(w[i] <= j)
                dp[i][j] = max(dp[i-1][j], v[i] + dp[i-1][max(j-w[i], 0)]);
            else
                dp[i][j] = dp[i-1][j];
        }
    }
    cout<<dp[N][W]<<"\n";
}

int main ()
{
    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);

    int t = 1;
    while (t--) solve();
    return 0;
}
```

# 4 geometry

## 4.1 triangles

Let a, b, c be length of the three sides of a triangle.

$$p = (a + b + c) * 0.5$$

The inradius is defined by:

$$iR = \sqrt{\frac{(p-a)(p-b)(p-c)}{p}}$$

The radius of its circumcircle is given by the formula:

$$cR = \frac{abc}{\sqrt{(a+b+c)(a+b-c)(a+c-b)(b+c-a)}}$$

# 5 graphs

## 5.1 bfs

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

using namespace std;

int main() {
    vector<vector<int>> adj;
    int n; int s; // nodes and source

    queue<int> q;
    vector<bool> used(n);
    vector<int> d(n, 0), p(n); // distance and parents

    q.push(s);
    used[s] = true;
    p[s] = -1; // root
    d[s] = 0;
    while(!q.empty()) {
        int v = q.front();
        q.pop();
        for(auto u : adj[v]) {
            if(!used[u]) {
                q.push(u);
                used[u] = true;
                p[u] = v;
                d[u] = ++d[v];
            }
        }
    }

    // showing the shortest path
    int u;
    if(!used[u]) {
        cout<<"No path!\n";
    } else {
        vector<int> path;
        for(int v=u; v != -1; v=p[v])
            path.push_back(v);
        reverse(path.begin(), path.end());
        for(int v: path)
            cout<<v<<" ";
    }
    return 0;
}
```

## 5.2 dfs

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

int n = 6;
vector<vector<int>> adj(n);
vector<bool> visited(n);

int main() {
    vector<int> col(n);
    col[0] = 0;
    auto dfs = [&](int u, int p, auto&& dfs) -> void {
        for (int v : adj[u])
            if (v != p) {
                col[v] = col[u] ^ 1;
                dfs(v, u, dfs);
            }
    };
    dfs(0, -1, dfs);
}
```

## 5.3 dijkstra

```
/**
 *    author: mralves
 *    created: 11-05-2023 21:24:59
**/
#include <bits/stdc++.h>

using namespace std;
using ll = int64_t;

const int INF = 1000000000;

vector<vector<pair<int, int>>> adj;

void dijkstra(int s, vector<int> &d, vector<int> &p) {
    int n = adj.size();
    d.assign(n, INF);
    p.assign(n, -1);
    vector<bool> u(n, false); // used
    d[s] = 0;

    for(int i=0; i<n; i++) {
        int v = -1;
        for(int j =0; j<n; j++) {
            if(!u[j] && (v == -1 || d[j] < d[v]))
                v = j;
        }

        if(d[v] == INF)
            break;

        u[v] = true;

        for(auto edge: adj[v]) {
            int to = edge.first;
            int len = edge.second;

            if(d[v] + len < d[to]) {
                d[to] = d[v] + len;
                p[to] = v;
            }
        }
    }
}

int main ()
{
    // simulation
    adj = {
        {},
        {{2, 6}, {4, 1}},
        {{1, 6}, {3, 5}, {4, 2}},
        {{2, 5}, {5, 5}},
        {{1, 1}, {2, 2}, {5, 1}},
        {{3, 5}, {4, 1}}
    };

    // 1 to 4
    int start = 1, end = 5;
    vector<int> d, p;
    dijkstra(1, d, p);

    cout<<d[end]<<"\n";

    vector<int> path;
    for(int i = end; i != -1; i = p[i]) {
        path.push_back(i);
    }

    reverse(path.begin(), path.end());

    for(auto x : path) {
        cout<<x<<" ";
    }

    cout<<"\n";

    return 0;
}
```

## 5.4 kruskal

# 6 misc

## 6.1 functors

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

struct Edge {
        int a, b, w;
};

struct cmp {
        bool operator()(const Edge &x, const Edge &y) const { return x.w <
            y.w; }
};

int main() {
        int M = 4;
        set<Edge, cmp> v;
        for (int i = 0; i < M; ++i) {
                int a, b, w;
                cin >> a >> b >> w;
                v.insert({a, b, w});
        }
        for (Edge e : v) cout << e.a << " " << e.b << " " << e.w << "\n";
}
```

## 6.2 template

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

#define debug (x) cout << #x << " = " << x << endl
#define pb(x) push_back(x)
#define all(x) x.begin(),x.end()

using namespace std;
using ll = int64_t;
using ii = pair<int, int>;

ll ceil(ll a, ll b) {return a % b == 0 ? a / b : a / b + 1;}
vector<ii> dir4 = {{1, 0}, {0, 1}, {-1, 0}, {0, -1}};
vector<ii> dir8 = {{1, 0}, {1, 1}, {0, 1}, {-1, 1}, {-1, 0}, {-1, -1},
    {0, -1}, {1, -1}};

void solve() {
}

int main ()
{
    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);

    int t = 1;
    cin>>t;
    while (t--) solve();
    return 0;
}
```

# 7 number-theory

## 7.1 binpow

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

using namespace std;

const long long MOD = 998244353;

long long binpow(long long a, long long b) {
    a %= MOD;
    long long res = 1;
    while(b > 0) {
        if(b & 1)
            res = res * a % MOD;
        a = a * a % MOD;
        b >>= 1;
    }
    return res;
}

int main() {
    cout<<binpow(2,5);
    return 0;
}
```

## 7.2 modular-arithmetic

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

// Operation modulo something

// adition
long long add(long long a, long long b, long long m) {
    long long x = (a + b) % m;
    return x;
}
// subtraction
long long sub(long long a, long long b, long long m) {
    long long x = (a - b) % m;
    // sometimes x can be negative
    if (x < 0) x += m;
    return x;
}
// multiplication
long long multi(long long a, long long b, long long m) {
    long long x = (a * b) % m;
    return x;
}

// division
long long div(long long a, long long b, long long m) {
    // just works for prime m
    long long b_inverse = binpow(b, m-2);
    long long x = (a * b_inverse) % m;
    return x;
}
```

## 7.3 primefact

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

using namespace std;

using ll = long long;

map<ll, ll> primeFact(ll N) {
    map<ll, ll> fact;
    for(ll i=2; i*i<=N; i++) {
        while(N % i == 0) {
            N /= i;
            fact[i]++;
        }
    }
    if(N > 1)
        fact[N]++;
    return fact;
}

int main() {

    map<ll, ll> factorials = primeFact(100);
    for(auto f : factorials) {
        cout<<f.first<<" "<<f.second<<"\n";
    }
    return 0;
}
```

## 7.4 sieve

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

using namespace std;

using ll = long long;
const ll MAX = 1e6;

// Sieve of Eratosthenes
// O(n log log(n))
vector<ll> sieve(ll MAX) {
    vector<bool> prime(MAX + 1, true);
    vector<ll> plist;
    for(ll i=2; i<=MAX; i++) {
        if(prime[i]) {
            plist.push_back(i);
            for(ll j=i*i; j<=MAX; j+=i) {
                prime[j] = false;
            }
        }
    }
    return plist;
}

int main() {

    vector<ll> plist = sieve(MAX);

    for(auto x: plist) {
        cout<<x<<" ";
    }
    cout<<"\n";
    return 0;
}
```