

noiref

ngmh

February 2019

Contents

1	Data Structures	2
1.1	Fenwick Trees	2
1.1.1	Point Update, Range Query	2
1.1.2	Range Update, Point Query	2
1.1.3	Range Update, Range Query	2
1.2	Segment Trees	2
1.2.1	1D	2
1.2.2	Lazy Propagation	3
1.2.3	2D	3
1.3	Sparse Table	4
2	Graph Theory	4
2.1	Depth First Search	4
2.2	Breadth First Search	4
2.3	Floyd-Warshall	4
2.4	Dijkstra	5
2.5	Travelling Salesman Problem	5
2.6	Union Find Disjoint Subset	5
2.7	Minimum Spanning Tree	5
2.7.1	Kruskal	5
2.7.2	Prim's	6
2.8	Bipartite Matching	6
2.9	Articulation Points	6
2.10	Bridges	7
2.11	Strongly Connected Components	7
2.12	Trees	8
2.12.1	Diameter	8
2.12.2	2^K Decomposition	8
2.12.3	Lowest Common Ancestor	9
2.12.4	All Pairs Shortest Path	9
2.12.5	Preorder	9
2.12.6	Postorder	9
2.12.7	Subtree to Range	9
2.12.8	Leaf Pruning	10
2.12.9	Weighted Maximum Independent Set	10
2.12.10	Heavy-Light Decomposition	10
2.12.11	Centroid Decomposition	11
3	Dynamic Programming	11
3.1	Coin Change	11
3.2	Coin Combinations	11
3.3	Knapsack	11
3.3.1	0-1	11
3.3.2	0-K	11
3.4	Longest Increasing Subsequence	12

3.4.1	N^2	12
3.4.2	$N \log N$	12
3.4.3	Optimal	12
3.5	Longest Common Subsequence	12
3.5.1	N^2	12
3.5.2	Longest Increasing Subsequence	12
3.6	Digits	13
3.7	Convex Hull Speedup	13
3.8	Divide and Conquer	14
3.9	Prefix Sums	14
3.9.1	1D	14
3.9.2	2D	14
4	Math	15
4.1	Greatest Common Divisor	15
4.2	Lowest Common Multiple	15
4.3	Modular Functions	15
4.3.1	Multiplication	15
4.3.2	Exponentiation	15
4.4	Primes	15
4.4.1	Sieve of Eratosthenes	15
4.4.2	Prime Factorisation	16
4.5	Fibonacci	16
4.6	nC_k	16
5	Algorithms	16
5.1	Discretisation	16
5.2	Binary Search	17
5.3	Mo's Algorithm	17
6	Miscellaneous	17
6.1	Macros + Functions + Variables	17
6.2	Compile Commands	18
6.2.1	Compile	18
6.2.2	Build	18
6.2.3	Command Line	18
6.2.4	Simple Script	18
6.3	Input/Output	18
6.3.1	Fast	18
6.3.2	Faster	18
6.4	Pruning	18
6.5	Optimise	18

1 Data Structures

1.1 Fenwick Trees

1.1.1 Point Update, Range Query

```
void pu(int i, int v){
    for(; i <= n; i += ls(i)) fw[i] += v;
}
int pq(int i){
    int t = 0;
    for(; i; i -= ls(i)) t += fw[i];
    return t;
}
int rq(int s, int e){
    return pq(e)-pq(s-1);
}
```

1.1.2 Rang Update, Point Query

```
void ru(int s, int e, int v){
    pu(s, v);
    pu(e+1, -v);
}
```

1.1.3 Range Update, Range Query

```
void ru(int s, int e, int v){
    pu(fw1, s, v);
    pu(fw1, e+1, -v);
    pu(fw2, s, -v*(s-1));
    pu(fw2, e+1, v*e);
}
int ps(int i){
    return pq(fw1, i)*i+pq(fw2, i);
}
int rq(int s, int e){
    return ps(e)-ps(s-1);
}
```

1.2 Segment Trees

1.2.1 1D

```
struct node {
    int s, e, m, v;
    node *l, *r;
    node(int _s, int _e){
        s = _s; e = _e; m = (s+e)/2; v = 0;
        if(s != e){
            l = new node(s, m);
            r = new node(m+1, e);
        }
    }
    void pu(int x, int y){
        if(s == e){ v = y; return; }
        if(x <= m) l->pu(x, y);
        if(x > m) r->pu(x, y);
        v = min(l->v, r->v);
    }
    int rq(int x, int y){
```

```

        if(s == x && e == y) return v;
        if(y <= m) return l->rq(x, y);
        if(x > m) return r->rq(x, y);
        return min(l->rq(x, m), r->rq(m+1, y));
    }
} *root;

```

1.2.2 Lazy Propagation

```

int pu(){
    if(s == e){ v += lazy; lazy = 0; return v; }
    v += lazy;
    l->lazy += lazy; r->lazy += lazy;
    lazy = 0;
    return v;
}

void ru(int x, int y, int z){
    if(s == x && e == y){ lazy += z; return; }
    if(y <= m) l->ru(x, y, z);
    else if(x > m) r->ru(x, y, z);
    else l->ru(x, m, z), r->ru(m+1, y, z);
    v = max(l->pu(), r->pu());
}

int rq(int x, int y){
    pu();
    if(s == x && e == y) return pu();
    if(y <= m) return l->rq(x, y);
    if(x > m) return r->rq(x, y);
    return max(l->rq(x, m), r->rq(m+1, y));
}

```

1.2.3 2D

```

struct node2D {
    int s, e, m;
    node1D *maxi;
    node2D *l, *r;
    node2D(int a, int b, int c, int d){
        s = a; e = b; m = (s+e)/2;
        maxi = new node1D(c, d);
        if(s != e){
            l = new node2D(s, m, c, d);
            r = new node2D(m+1, e, c, d);
        }
    }
    void pu(int a, int b, int v){
        if(s == e){ maxi->pu(b, v); return; }
        if(a <= m) l->pu(a, b, v);
        else r->pu(a, b, v);
        maxi->pu(b, max(l->maxi->rq(b, b), r->maxi->rq(b, b)));
    }
    int rq(int a, int b, int c, int d){
        if(s == a && e == b) return maxi->rq(c, d);
        if(b <= m) return l->rq(a, b, c, d);
        if(a > m) return r->rq(a, b, c, d);
        return max(l->rq(a, m, c, d), r->rq(m+1, b, c, d));
    }
} *root;

```

1.3 Sparse Table

```
int query(int l, int r){
    r++;
    int p = 31-__builtin_clz(r-l);
    return __gcd(sp[p][l], sp[p][r-(1<<p)]);
}

h = floor(log2(n));
for(int i = 0; i < n; i++) sp[0][i] = a[i];
for(int i = 1; i <= h; i++){
    for(int j = 0; j+(1<<i) <= n; j++){
        sp[i][j] = __gcd(sp[i-1][j], sp[i-1][j+(1<<(i-1))]);
    }
}
```

2 Graph Theory

2.1 Depth First Search

```
void dfs(int x, int p){
    for(auto it : adj[x]){
        if(it != p){
            par[it] = x;
            dep[it] = dep[x]+1;
            dist[it] = dist[x]+1;
            dfs(it, x);
        }
    }
}
```

2.2 Breadth First Search

```
d[nx][ny] = 0;
q.push(pi(sx, sy));
while(!q.empty()){
    pi f = q.front(); q.pop();
    for(int i = 0; i < 4; i++){
        nx = f.first+dx[i];
        ny = f.second+dy[i];
        if(nx < 0 || ny < 0 || nx >= h || ny >= w) continue;
        if(d[nx][ny] != -1) continue;
        d[nx][ny] = d[f.first][f.second]+1;
        q.push(pi(nx, ny));
    }
}
```

2.3 Floyd-Warshall

```
for(int i = 0; i < n; i++){
    for(int j = 0; j < n; j++){
        if(i == j) adj[i][j] = 0;
        else adj[i][j] = INT_MAX;
    }
}

for(int k = 0; k < n; k++){
    for(int i = 0; i < n; i++){
        for(int j = 0; j < n; j++){
            adj[i][j] = min(adj[i][j], adj[i][k]+adj[k][j]);
        }
    }
}
```

```

    }
}

```

2.4 Dijkstra

```

priority_queue<pi, vector<pi>, greater<pi> > pq;
dist[s] = 0;
pq.push(pi(0, s));
while(!pq.empty()){
    pi f = pq.top(); pq.pop();
    for(auto it:adj[f.second]){
        if(dist[it.first] == -1 || dist[it.first] > f.first+it.second){
            dist[it.first] = f.first+it.second;
            pq.push(pi(dist[it.first], it.first));
        }
    }
}

```

2.5 Travelling Salesman Problem

```

long long adj[14][14], dp[8200][14];

//start with 1, 0
//mask is 0 visited, at node 0
long long tsp(long long mask, long long pos){
    if(mask == visited) return adj[pos][0];
    if(dp[mask][pos] != -1) return dp[mask][pos];
    long long ans = LLONG_MAX;
    for(int i = 0; i < m; i++){
        if((mask&(1<<i)) == 0){
            long long newans = adj[pos][i]+tsp(mask|(1<<i), i);
            ans = min(ans, newans);
        }
    }
    return dp[mask][pos] = ans;
}

```

```

visited = (1 << m)-1;
res = tsp(1, 0);
if(res < 0) cout << "-1";
else cout << res;

```

2.6 Union Find Disjoint Subset

```

int root(int x){
    if(p[x] == -1) return x;
    return p[x] = root(p[x]);
}
void connect(int x, int y){
    p[root(x)] = root(y);
}

```

2.7 Minimum Spanning Tree

2.7.1 Kruskal

```

sort(edgs.begin(), edgs.end());
for(auto it:edgs){

```

```

        if(root(it.second.first) != root(it.second.second)){
            connect(it.second.first, it.second.second);
            cost += it.first;
        }
    }
}

```

2.7.2 Prim's

```

priority_queue<pi, vector<pi>, greater<pi> > pq;
dist[s] = 0;
pq.push(pi(0, s));
while(!pq.empty()){
    pi f = pq.top(); pq.pop();
    for(auto it:adj[f.second]){
        if(dist[it.first] == -1 || dist[it.first] > max(f.first, it.second)){
            dist[it.first] = max(f.first, it.second);
            pq.push(pi(dist[it.first], it.first));
        }
    }
}
}

```

2.8 Bipartite Matching

```

bool dfs(int u){
    if(v[u]) return 0;
    v[u] = 1;
    for(auto v:adj[u]){
        if(match[v] == -1 || dfs(match[v])){
            match[v] = u;
            match[u] = v;
            return 1;
        }
    }
    return 0;
}

memset(match, -1, sizeof(match));
for(auto lit:ho){
    for(auto rit:adj[lit]){
        if(match[rit] == -1){
            match[rit] = lit;
            match[lit] = rit;
            mcbm++;
            break;
        }
    }
}

for(auto lit:ho){
    if(match[lit] == -1){
        memset(v, 0, sizeof(v));
        mcbm += dfs(lit);
    }
}

mis = n-mcbm;

```

2.9 Articulation Points

```

void atp(int node, int depth){
    vi[node] = 1;
    dep[node] = depth;
    low[node] = depth;
}

```

```

    for(auto it:adj[node]){
        if(!vi[it]){
            par[it] = node;
            atp(it, depth+1);
            chi[node]++;
            if(low[it] >= dep[node]) atps[node]++;
            low[node] = min(low[node], low[it]);
        } else if(it != par[node]){
            low[node] = min(low[node], dep[it]);
        }
    }
}
}
atp(0, 0);
// root -> chi[i]
// other -> atps[i]+1

```

2.10 Bridges

```

void bridges(int x, int par){
    if(dep[x] != -1) return;
    dep[x] = low[x] = co++;
    int t = 0;
    for(auto it:adj[x]){
        if(it == par && t == 0){ t++; continue; }
        if(dep[it] != -1){
            if(low[it] > dep[x]) bgs.push_back(pi(x, it));
            low[x] = min(low[x], low[it]);
            continue;
        }
        bridges(it, x);
        if(low[it] > dep[x]) bgs.push_back(pi(x, it));
        low[x] = min(low[x], low[it]);
    }
}
for(int i = 1; i <= n; i++) bridges(i, 0);

```

2.11 Strongly Connected Components

```

stack<int> s;
vector<int> cur, adj[100001];
set<int> adjsc[100001];
vector<vector<int>> > comps;
void scc(int v){
    idxs[v] = idx;
    lowlink[v] = idx;
    idx++;
    s.push(v);
    ins[v] = 1;
    for(auto w:adj[v]){
        if(idxs[w] == -1){
            scc(w);
            lowlink[v] = min(lowlink[v], lowlink[w]);
        } else if(ins[w]){
            lowlink[v] = min(lowlink[v], idxs[w]);
        }
    }
    if(lowlink[v] == idxs[v]){
        cur.clear();
        w = 0;
    }
}

```



```

        while(w != v){
            w = s.top(); s.pop();
            ins[w] = 0;
            cur.push_back(w);
        }
        comps.push_back(cur);
    }
}
idx = 1;
memset(idxs, -1, sizeof(idxs));
for(int i = 1; i <= n; i++){
    if(idxs[i] == -1) scc(i);
}
memset(com, -1, sizeof(com));
for(int i = 0; i < comps.size(); i++){
    for(auto it:comps[i]) com[it] = i;
}
for(int i = 1; i <= n; i++){
    for(auto it:adj[i]){
        if(com[i] != com[it]) adjsc[com[i]].insert(com[it]);
    }
}
}

```

2.12 Trees

2.12.1 Diameter

```

pi dfs(int node, int par, int dist){
    pi b = pi(node, dist);
    for(auto it:adj[node]){
        if(it.first != par){
            pi c = dfs(it.first, node, dist+it.second);
            if(c.second > b.second) b = c;
        }
    }
    return b;
}
f = dfs(0, -1, 0);
l = dfs(f.first, -1, 0);
// l.second

```

2.12.2 2^K Decomposition

```

int par(int x, int k){
    for(int i = 19; i >= 0; i--){
        if(k >= (1<<i)){
            if(x == -1) return x;
            x = p[x][i];
            k -= (1<<i);
        }
    }
    return x;
}

memset(p, -1, sizeof(p));
dfs(0);
for(int k = 1; k <= 19; k++){
    for(int i = 0; i < n; i++){
        if(p[i][k-1] != -1) p[i][k] = p[p[i][k-1]][k-1];
    }
}

```

```
}
```

2.12.3 Lowest Common Ancestor

```
int lca(int x, int y){
    if(dep[x] < dep[y]) swap(x, y);
    for(int k = 19; k >= 0; k--){
        if(p[x][k] != -1 && dep[p[x][k]] >= dep[y]) x = p[x][k];
    }
    if(x == y) return x;
    for(int k = 19; k >= 0; k--){
        if(p[x][k] != p[y][k]){
            x = p[x][k];
            y = p[y][k];
        }
    }
    return p[x][0];
}
```

2.12.4 All Pairs Shortest Path

```
int distance(int x, int y){
    return dist[x]+dist[y]-2*dist[lca(x, y)];
}
```

2.12.5 Preorder

```
// UNTESTED
void dfs(int x, int par){
    c++;
    pre[x] = c;
    for(auto it:adj[x]){
        if(it != par) dfs(adj[it], x);
    }
}
```

2.12.6 Postorder

```
// UNTESTED
void dfs(int x, int par){
    c++;
    for(auto it:adj[x]){
        if(it != par) dfs(adj[it], x);
    }
    post[x] = c;
}
```

2.12.7 Subtree to Range

```
int dfs(int x, int par){
    c++;
    pre[x] = c;
    for(auto it:adj[x]){
        if(it != par) rig[pre[x]] = max(rig[pre[x]], dfs(it, x));
    }
    if(rig[pre[x]] == 0) rig[pre[x]] = pre[x];
    return rig[pre[x]];
}
// node -> pre[x]
// children -> pre[x]+1, rig[pre[x]]
```

2.12.8 Leaf Pruning

```
for(int i = 1; i <= n; i++){
    if(adj[i].size() == 1) leafs.push(i);
}
while(!leafs.empty()){
    u = leafs.front(); leafs.pop();
    t = adj[u][0];
    adj[t].erase(find(adj[t].begin(), adj[t].end(), u));
    if(adj[t].size() == 1) leafs.push(t);
    adj[u].erase(find(adj[u].begin(), adj[u].end(), t));
}
```

2.12.9 Weighted Maximum Independent Set

```
int mis(int v, bool take, int p){
    if(dp[v][take] != -1) return dp[v][take];
    int ans = take*c[v];
    for(auto it:adj[v]){
        if(it == p) continue;
        int temp = mis(it, 0, v);
        if(!take) temp = max(temp, mis(it, 1, v));
        ans += temp;
    }
    return dp[v][take] = ans;
}

void ans(int v, bool take, int p){
    for(auto it:adj[v]){
        if(it == p) continue;
        int temp0 = dp[it][0], temp1 = (take ? -1 : dp[it][1]);
        if(temp0 > temp1) ans(it, 0, v);
        else { a.push_back(it); ans(it, 1, v); }
    }
}

mis(0, 0, -1);
mis(0, 1, -1);
if(dp[0][1] > dp[0][0]){ a.push_back(0); ans(0, 1, -1); }
else ans(0, 0, -1);
```

2.12.10 Heavy-Light Decomposition

```
int dfs(int node){
    int size = 1, max_c = 0;
    for(auto it:adj[node]){
        if(it.s != par[node]){
            par[it.s] = node; dep[it.s] = dep[node]+1;
            int c_size = dfs(it.s);
            size += c_size;
            if(c_size > max_c){ max_c = c_size; heav[node] = it.s; }
        }
    }
    return size;
}

void decomp(int node, int hea){
    head[node] = hea; pos[node] = c_pos++;
    if(heav[node] != -1) decomp(heav[node], hea);
    for(auto it:adj[node]){
        if(it.s == par[node]) continue;
        if(it.s != heav[node]) decomp(it.s, it.s);
    }
}
```

```

        root->update(pos[it.s], it.f);
    }
}
int query(int a, int b){
    int res = 0;
    for(; head[a] != head[b]; b = par[head[b]]){
        if(dep[head[a]] > dep[head[b]]) swap(a, b);
        res = max(res, root->query(pos[head[b]], pos[b]));
    }
    if(dep[a] > dep[b]) swap(a, b);
    res = max(res, root->query(pos[a]+1, pos[b]));
    return res;
}
dfs(0);
decomp(0, 0);

```

2.12.11 Centroid Decomposition

Decompose Tree into Centroid Tree
 where the node is central
 Use them to speed queries up

3 Dynamic Programming

3.1 Coin Change

```

dp[0] = 0;
for(int i = 1; i <= v; i++){
    dp[i] = INT_MAX;
    for(int j = 0; j < n; j++){
        if(i >= c[j]) dp[i] = min(dp[i], dp[i-c[j]]+1);
    }
}

```

3.2 Coin Combinations

```

ways[0] = 1;
for(int i = 0; i < c; i++){
    for(int j = 1; j <= v; j++){
        if(j >= coins[i]) ways[j] += ways[j-coins[i]];
    }
}

```

3.3 Knapsack

3.3.1 0-1

```

for(int i = 0; i < n; i++){
    for(int j = s; j >= w[i]; j--){
        dp[j] = max(dp[j], dp[j-w[i]]+v[i]);
    }
}

```

3.3.2 0-K

```

//further speedup:
//take top s/w most valued items
//for every possible item
//not just multiple copies of one item

```

```

for(int i = 0; i < n; i++){
    cin >> v >> w >> k;
    k = min(k, s/w);
    c = 1;
    while(true){
        if(k < c) break;
        items.push_back(pi(v*c, w*c));
        k -= c;
        c *= 2;
    }
    if(k != 0) items.push_back(pi(v*k, w*k));
}
for(auto it : items){
    for(int i = s; i >= it.second; i--){
        dp[i] = max(dp[i], dp[i-it.second]+it.first);
    }
}

```

3.4 Longest Increasing Subsequence

3.4.1 N^2

```

for(int i = 0; i < n; i++){
    for(int j = 0; j < i; j++){
        if(a[j] < a[i]) lis[i] = max(lis[i], lis[j]);
    }
    lis[i]++;
    ans = max(ans, lis[i]);
}

```

3.4.2 $N \log N$

```

for(int i = 0; i < n; i++){
    t = query(a[i]-1)+1;
    update(a[i], t);
    ans = max(ans, t);
}

```

3.4.3 Optimal

```

for(int i = 0; i < n; i++){
    int p = lower_bound(dp, dp+l, a[i])-dp;
    dp[p] = a[i];
    l = max(p+1, l);
}

```

3.5 Longest Common Subsequence

3.5.1 N^2

```

int lcs(int a1, int b1){
    if(a1 == 0 || b1 == 0) return 0;
    if(dp[a1][b1] != -1) return dp[a1][b1];
    dp[a1][b1] = max(lcs(a1-1, b1), lcs(a1, b1-1));
    if(a[a1-1] == b[b1-1]) dp[a1][b1] = max(dp[a1][b1], lcs(a1-1, b1-1)+1);
    return dp[a1][b1];
}

```

3.5.2 Longest Increasing Subsequence

```

for(int i = 0; i < n; i++){ cin >> a[i].first; a[i].second = i+1; }
for(int i = 0; i < m; i++) cin >> b[i];
sort(a, a+n);
for(int i = 0; i < m; i++){
    p = lower_bound(a, a+n, pi(b[i], 0));
    if(p != a+n) c.push_back(p->second);
}
// Perform LIS on c

```

3.6 Digits

```

long long derp(int idx, int prev, int same, int allzero){
    if(idx == num.size()) return dp[idx][prev][same][allzero] = !allzero;
    if(dp[idx][prev][same][allzero] != -1) return dp[idx][prev][same][allzero];
    long long sum = 0, limit;
    if((!allzero && same) || (allzero && idx == 0)) limit = num[idx];
    else limit = 9;
    for(int i = 0; i <= limit; i++){
        if(i == 4) continue;
        if(allzero){
            if(i == 0) sum += derp(idx+1, 10, 1, 1);
            else sum += derp(idx+1, i, (idx == 0 && i == limit), 0);
        } else if(!(i == 3 && prev == 1)){
            if(same && i == num[idx]) sum += derp(idx+1, i, 1, 0);
            else sum += derp(idx+1, i, 0, 0);
        }
    }
    return dp[idx][prev][same][allzero] = sum;
}

void dcmp(long long x){
    num.clear();
    while(x > 0){ num.push_back(x%10); x /= 10; }
    reverse(num.begin(), num.end());
}

long long solve(long long x){
    if(memo.count(x) != 0) return memo[x];
    memset(dp, -1, sizeof(dp));
    dcmp(x);
    return memo[x] = derp(0, 10, 1, 1);
}

```

3.7 Convex Hull Speedup

```

// dp(x) = max(dp(i)+f(p(x)-p(i)))
//         = max(dp(i)+a(p(x)-p(i))^2+b(p(x)-p(i))+c)
//         = max(dp(i)+ap(x)^2-2ap(x)p(i)+ap(i)^2+bp(x)-bp(i))+c
//         = max(dp(i)+ap(i)^2-bp(i)-2ap(x)p(i))+ap(x)^2+bp(x)+c
//         = max(c(i)+m(i)p(x))+v(x)
// c(i) = dp(i)+ap(i)^2-bp(i)
// m(i) = -2ap(i)
// v(x) = ap(x)^2+bp(x)+c
long long func(pi line, long long x){
    return line.first*x+line.second;
}

long double intersection(long long m1, long long c1, long long m2, long long c2){
    return (long double)(c2-c1)/(m1-m2);
}

long double intersect(pi x, pi y){
    return intersection(x.first, x.second, y.first, y.second);
}

```

```

}
long long query(long long x){
    while(hull.size() > 1){
        if(func(hull[0], x) < func(hull[1], x)){
            hull.pop_front();
        } else break;
    }
    return func(hull[0], x);
}
void insert(long long m, long long c){
    pi line = pi(m, c);
    while(hull.size() > 1){
        long long s = hull.size();
        if(intersect(hull[s-1], line) <= intersect(hull[s-2], line)){
            hull.pop_back();
        } else break;
    }
    hull.push_back(line);
}
insert(0, 0);
for(int i = 1; i <= n; i++){
    dp[i] = query(ps[i])+a*ps[i]*ps[i]+b*ps[i]+c;
    insert(-2*a*ps[i], dp[i]+a*ps[i]*ps[i]-b*ps[i]);
}

```

3.8 Divide and Conquer

```

long long cost(int s, int e){
    return (ps[e]-ps[s-1])*(e-s+1);
}
void dnc(int s, int e, long long x, int y, int k){
    if(s > e) return;
    int m = (s+e)/2, best = 0;
    dp[m][k] = LLONG_MAX/2;
    for(int i = x; (i <= y && i <= m); i++){
        if(dp[m][k] > dp[i][!k]+cost(i+1, m)){
            dp[m][k] = dp[i][!k]+cost(i+1, m);
            best = i;
        }
    }
    if(s < m) dnc(s, m-1, x, best, k);
    if(m < e) dnc(m+1, e, best, y, k);
}
for(int i = 1; i <= n; i++) dp[i][0] = LLONG_MAX/2;
for(int i = 1; i <= g; i++){
    for(int j = 1; j <= n; j++) dp[j][i%2] = LLONG_MAX/2;
    dnc(0, n, 0, n, i%2);
}

```

3.9 Prefix Sums

3.9.1 1D

```

int query(int s, int e){
    return ps[e]-ps[s-1];
}

```

```

for(int i = 1; i <= n; i++) ps[i] = ps[i-1]+a[i];

```

3.9.2 2D

```

int query(int x1, int y1, int x2, int y2){
    return ps[x2][y2]-ps[x1-1][y2]-ps[x2][y1-1]+ps[x1-1][y1-1];
}

for(int i = 1; i <= r; i++){
    for(int j = 1; j <= c; j++){
        ps[i][j] = ps[i-1][j]+ps[i][j-1]-ps[i-1][j-1]+a[i][j];
    }
}

```

4 Math

4.1 Greatest Common Divisor

```

// Alternatively, find highest common powers for each factor
int gcd(int a, int b){
    if(b == 0) return a;
    return gcd(b, a%b);
}

```

4.2 Lowest Common Multiple

```

// Alternatively, find highest powers for each factor
int lcm(int a, int b){
    return (a*b)/gcd(a, b);
}

```

4.3 Modular Functions

4.3.1 Multiplication

```

int mulmod(int a, int b, int m) {
    int res = 0;
    while(b > 0) {
        if(b % 2 == 1) res = (res+a)%m;
        a = (a*2)%m;
        b /= 2;
    }
    return res % m;
}

```

4.3.2 Exponentiation

```

int powmod(int a, int b, int m) {
    int res = 1;
    while(b > 0) {
        if(b % 2 == 1) res = mulmod(res, a, m);
        a = mulmod(a, a, m);
        b /= 2;
    }
    return res % m;
}

```

4.4 Primes

4.4.1 Sieve of Eratosthenes

```

memset(prime, 1, sizeof(prime));
prime[0] = prime[1] = 0;
for(int i = 2; i <= 1000000; i++){
    if(prime[i]){

```



```

        for(int j = 2; i*j <= 1000000; j++){
            prime[i*j] = 0;
        }
    }
}

```

4.4.2 Prime Factorisation

```

while(x % 2 == 0){ cnt[2]++; x /= 2; }
for(int i = 3; i*i <= x+1; i += 2){
    while(x % i == 0){ cnt[i]++; x /= i; }
    if(x == 1) break;
}
if(x > 1) cnt[x]++;

```

4.5 Fibonacci

```

typedef pair<pi, pi> matrix;
matrix multiply(matrix x, matrix y, long long z){
    return matrix(
        pi((x.f.f*y.f.f+x.f.s*y.s.f)%z,
            (x.f.f*y.f.s+x.f.s*y.s.s)%z),
        pi((x.s.f*y.f.f+x.s.s*y.s.f)%z,
            (x.s.f*y.f.s+x.s.s*y.s.s)%z));
}
matrix square(matrix x, long long y){
    return multiply(x, x, y);
}
matrix power(matrix x, long long y, long long z){
    if(y == 1) return x;
    if(y % 2 == 0) return square(power(x, y/2, z), z);
    return multiply(x, square(power(x, y/2, z), z), z);
}
long long fibo(long long n, long long m){
    matrix x = matrix(pi(1, 1), pi(1, 0));
    return power(x, n, m).f.s;
}

```

4.6 nC_k

```

if(k > n-k) k = n-k;
for(int i = 0; i < k; i++){
    ans *= (n-i);
    ans /= (i+1);
}

```

5 Algorithms

5.1 Discretisation

```

for(int i = 0; i < n; i++){
    cin >> a[i];
    b[i] = a[i];
}
sort(b, b+n);
for(int i = 0; i < n; i++){
    d = lower_bound(b, b+n, a[i])-b;
    a[i] = d+1;
}

```

5.2 Binary Search

```
while(mini < maxi){
    medi = mini+(maxi-mini)/2;
    if(can(medi)) maxi = medi;
    else mini = medi+1;
}
```

5.3 Mo's Algorithm

```
//queries : index, left, right
//answers : index, answer

bool cmp(pii x, pii y){
    if(x.second.first/blk != y.second.first/blk){
        return x.second.first/blk < y.second.first/blk;
    }
    if(x.second.first/blk & 1){
        return x.second.second < y.second.second;
    }
    return x.second.second > y.second.second;
}

blk = sqrt(n);
sort(qs, qs+q, cmp);
for(int i = 0; i < q; i++){
    l = qs[i].second.first; r = qs[i].second.second;
    while(lft > l){
        //remove a[lft-1]
        lft--;
    }
    while(rgt <= r){
        //add a[rgt]
        rgt++;
    }
    while(lft < l){
        //add a[lft]
        lft++;
    }
    while(rgt > r+1){
        //remove a[rgt-1]
        rgt--;
    }
    ans[i] = pi(qs[i].first, cur);
}
sort(ans, ans+q);
```

6 Miscellaneous

6.1 Macros + Functions + Variables

```
#define f first
#define s second
#define pb push_back
#define ins insert
int ls(int x){ return (x)&(-x); }
mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());
// 4 Directions
int dx[]={0, 0, -1, 1};
```

```

int dy[]={-1, 1, 0, 0};
// 8 Directions
int dx[]={0, 0, -1, 1, -1, 1, -1, 1};
int dy[]={-1, 1, -1, 1, 0, 0, 1, -1};
// Knight Moves
int dx[]={-1, -2, 1, 2, 2, 1, -2, 1};
int dy[]={-2, -1, -2, -1, 1, 2, 1, 2};

```

6.2 Compile Commands

6.2.1 Compile

```
g++ -c "%f" -std=c++11
```

6.2.2 Build

```
g++ -o "%e" "%f" -std=c++11
```

6.2.3 Command Line

```
g++ "file.cpp" -o "file" -std=c++11
```

6.2.4 Simple Script

```

int main(int argc, char **argv){
    system(g++ -o (argv[1]) (argv[1]).cpp -std=c++11);
}

```

6.3 Input/Output

6.3.1 Fast

```

ios_base::sync_with_stdio(false);
cin.tie(0);

```

6.3.2 Faster

```

inline int read_int() {
    int x = 0;
    char ch = getchar_unlocked();
    while (ch < '0' || ch > '9') ch = getchar_unlocked();
    while (ch >= '0' && ch <= '9'){
        x = (x << 3) + (x << 1) + ch - '0';
        ch = getchar_unlocked();
    }
    return x;
}

```

6.4 Pruning

```

auto start = chrono::high_resolution_clock::now();
auto end = chrono::high_resolution_clock::now();
auto elapse = chrono::duration<double>(end-start);
if(elapse.count() > 2.9) break;

```

6.5 Optimise

```
int __attribute__((optimize("Ofast"), target("arch=sandybridge"))) f(){}

```