Contents

1 Basic

1.1 Default Code

```
#include <bits/stdc++.h>
#define int long long
// #pragma GCC target("popcnt")
// #pragma GCC optimize("O3")
using namespace std;


void solve() {

}

signed main() {
  ios_base::sync_with_stdio(false);
  cin.tie(nullptr);
  int tt = 1;
  cin >> tt;
  while (t--) {
      solve();
  }
  return 0;
}
```

1.2 PBDS

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
using namespace __gnu_pbds;
using namespace std;

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

/*
如果有 define int long long 記得拿掉
Tree<int> t 就跟 set<int> t 一樣，有包好 template
rb_tree_tag 使用紅黑樹
第三個參數 less<T> 為由小到大，greater<T> 為由大到小
插入 t.insert(); 刪除 t.erase();
t.order_of_key
    (k); 從前往後數 k 是第幾個 (0-base 且回傳 int 型別)
t.find_by_order(k);
    從前往後數第 k 個元素 (0-base 且回傳 iterator 型別)
t.lower_bound
    (); t.upper_bound(); 用起來一樣 回傳 iterator
可以用 Tree<pair<int, int>> T 來模擬 mutiset
*/
```

1.3 int128 Input Output

```
// 抄 BBuf github 的

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

void scan(__int128 &x) // 輸入
{
  x = 0;
  int f = 1;
  char ch;
  if((ch = getchar()) == '-') f = -f;
  else x = x*10 + ch-'0';
  while((ch = getchar()) >= '0' && ch <= '9')
    x = x*10 + ch-'0';
  x *= f;
}

void print(__int128 x) // 輸出
{
  if(x < 0)
  {
    x = -x;
    putchar('-');
  }
  if(x > 9) print(x/10);
  putchar(x%10 + '0');
}

int main()
{
  __int128 a, b;
  scan(a);
  scan(b);
  print(a + b);
  puts("");
  print(a*b);
  return 0;
}
```

1.4 Python

```
## Input
# p q 都是整數，中間以空白分開輸入
p, q = map(int, input().split())

# 輸入很多個用空
    白隔開的數字，轉成 float 放進陣列，s 是 input 字串
arr = list(map(float, s.split()))


# 分數用法 Fraction(被除數, 除數)
from fractions import Fraction

arr = [Fraction
    (0), Fraction(1, 6), Fraction(1, 2), Fraction(5,
     12), Fraction(0), Fraction(-1, 12), Fraction(0)]

def fx(x):
    x = Fraction(x)
    ans = Fraction(0)
    for i in range(1, 7):
        ans += arr[i] * x ** (7 - i)
    return ans
```

2 Math

2.1 質數表

```
vector<int> prime_table(int n){
  vector<int> table(n + 1, 0);
  for(int i = 1; i <= n; i++){
    for(int j = i; j <= n; j += i){
      table[j]++;
    }
  }
  return table;
}
```

2.2 快速冪

```
#define int long long
```

```
// 根據費馬小定
    理，若 a p 互質，a^(p-2) 為 a 在 mod p 時的乘法逆元
// a ^ (b ^ c
    ) % mod = fast_pow(a, fast_pow(b, c, mod - 1), mod)
typedef unsigned long long ull;
inline int ksc(ull
     x, ull y, int p) {  // O(1)快速乘（防爆long long）
  return (x
      * y - (ull)((long double)x / p * y) * p + p) % p;
}

inline int fast_pow(int a, int b, int mod)
{
  // a^b % mod
  int res = 1;
  while(b)
  {
    if(b & 1) res =  ksc(res, a, mod);
    a = ksc(a, a, mod);
    b >>= 1;
  }
  return res;
}
```

2.3 擴展歐幾里得

```
int gcd(int a, int b)
{
  return b == 0 ? a : gcd(b, a % b);
}

int lcm(int a, int b)
{
  return a * b / gcd(a, b);
}

pair<int, int> ext_gcd
    (int a, int b) //擴展歐幾里德 ax+by = gcd(a,b)
{
  if (b == 0)
    return {1, 0};
  if (a == 0)
    return {0, 1};
  int x, y;
  tie(x, y) = ext_gcd(b % a, a);
  return make_pair(y - b * x / a, x);
}
```

2.4 矩陣

```
// 矩陣乘法 (A * B) % mod

template <typename T>
vector<vector<T>> matrix_mult(const vector<
    vector<T>>& A, const vector<vector<T>>& B, T mod) {
  int m = A.size();
  int n = A[0].size();
  int p = B[0].size();
  assert(A[0].size() == B.size());
  vector<vector<T>> result(m, vector<T>(p, 0));
  for (int i = 0; i < m; ++i) {
    for (int j = 0; j < p; ++j) {
      for (int k = 0; k < n; ++k) {
        result[i][j]
            = (result[i][j] + A[i][k] * B[k][j]) % mod;
      }
    }
  }
  return result;
}
```

2.5 Miller rabin Prime test

```
// fast_pow 去前面抄，需要處裡防暴乘法
// 記得 #define int long long 也要放
// long long 範圍內測試過答案正確
// time: O(logn)

inline bool mr(int x, int p) {
  if (fast_pow(x, p - 1, p) != 1) return 0;
  int y = p - 1, z;
  while (!(y & 1)) {
      y >>= 1;
      z = fast_pow(x, y, p);
      if (z != 1 && z != p - 1) return 0;
      if (z == p - 1) return 1;
  }
  return 1;
}

inline bool prime(int x) {
  if (x < 2) return 0;
  if (x == 2 ||
      x == 3 || x == 5 || x == 7 || x == 43) return 1;
  // 如果把 2
      到 37 前 12 個質數都檢查一遍 可以保證 2^78 皆可用
  return mr(2, x)
      && mr(3, x) && mr(5, x) && mr(7, x) && mr(43, x);
}
```

2.6 Pollard's Rho

```
// 主函數記得放 srand(time(nullptr))
// prime 檢測以及快速冪, gcd 等請從前面抄

// 輸入一個數字 p，隨
    機回傳一個 非 1 非 p 的因數，若 p 是質數會無窮迴圈
#define rg register int
inline int rho(int p) {
  int x, y, z, c, g;
  rg i, j;
  while (1) {
    y = x = rand() % p;
    z = 1;
    c = rand() % p;
    i = 0, j = 1;
    while (++i) {
      x = (ksc(x, x, p) + c) % p;
      z = ksc(z, abs(y - x), p);
      if (x == y || !z) break;
      if (!(i % 127) || i == j) {
        g = gcd(z, p);
        if (g > 1) return g;
        if (i == j) y = x, j <<= 1;
      }
    }
  }
}

// 回傳隨機一個質因數，若 input 為質數，則直接回傳
int prho(int p){
  if(prime(p)) return p;
  int m = rho(p);
  if(prime(m)) return m;
  return prho(p / m);
}

// 回傳將 n 質因數分解的結果，由小到大排序
// ex: input: 48, output: 2 2 2 2 3
vector<int> prime_factorization(int n){
  vector<int> ans;
  while(n != 1){
    int m = prho(n);
    ans.push_back(m);
    n /= m;
  }
  sort(ans.begin(), ans.end());
  return ans;
}
```

2.7 皮薩諾定理

```
// fib(x) % m = fib(x + kn) % m 當 k >= 1，求 n
// n 為費式數列 % m 會重複的週期
// pisano_period(m) <= 6m
// 通常這都要本地跑

#define int long long

int pisano_period(int m) {
  int pre = 0, cur = 1;
  int temp;
  for (int i = 0; i < m * m; i++) {
    temp = pre;
    pre = cur;
    cur = (temp + cur) % m;
    if (pre == 0 && cur == 1) return i + 1;
  }
  return 0;
}
```

2.8 卡特蘭數

```
"""
卡特蘭數 Catalan
公式：H(n) = C(2 * n, n) // (n + 1), n >= 2, n 為正整數
快速計算方式：
1. H(0) = H(1) = 1, H(n)
     = sum(H(i - 1) * H(n - i) for i in range(1, n + 1))
2. H(n) = H(n - 1) * (4 * n - 2) // (n + 1)
3. H(n) = C(2 * n, n) - C(2 * n, n - 1)
"""

"""
可解問題：
有效括號匹配問題：
    給定 n 個左括號與右括號，求有幾種不同的正確括號匹配
二元樹結構問題：給定 n 個節點，求有幾種不同的二元樹結構
將一個凸
    n + 2 邊形劃分成多個三角形，求有幾種不同的劃分方式
狄克路徑：給定 n * n 的網格，
    從左下到右上的路徑中，永不超過對角線的路徑有幾種
一個 stack 在 push 順
    序不變的情況下 (1, 2, 3, ..., n)，有幾種 pop 的方式
在圓上選擇 2 * n 個
    點，將這些點兩兩連接使得 n 條線段不相交的方法有幾種
"""

n = int(input())

catalan = [1 for _ in range(n + 1)]
for i in range(1, n + 1):
    catalan
        [i] = catalan[i - 1] * (4 * i - 2) // (i + 1)

ans = 0
for i in range(0, n + 1):  # 卡特蘭數的平方
    ans += catalan[i] * catalan[n - i]

print(ans)
# 185ms in codeforces, n <= 5000
```

3 Graph

3.1 DSU

```
class dsu{
  public:
    vector<int> parent;
    dsu(int num){
      parent.resize(num);
      for(int i = 0; i < num; i++) parent[i] = i;
    }
    int find(int x){
      if(parent[x] == x) return x;
      return parent[x] = find(parent[x]);
    }
    bool same(int a, int b){
      return find(a) == find(b);
    }
    void Union(int a, int b){
      parent[find(a)] = find(b);
    }
};
```

3.2 Dijkstra

```
// 傳入圖的 pair 為 {權重, 點}, 無限大預設 1e9 是情況改
#define pii pair<int, int>
vector<
    int> dijkstra(vector<vector<pii>> &graph, int src){
  int n = graph.size();
  vector<int> dis(n, 1e9);
  vector<bool> vis(n, false);
  priority_queue<pii, vector<pii>, greater<pii>> pq;
  pq.push({0, src});
  dis[src] = 0;
  while(!pq.empty()){
    auto [w, node] = pq.top();
    pq.pop();
    if(vis[node]) continue;
    vis[node] = true;
    for(auto [nw, nn]:graph[node]){
      if(w + nw < dis[nn]){
        dis[nn] = w + nw;
        pq.push({dis[nn], nn});
      }
    }
  }
  return dis;
}
```

3.3 SPFA

```
#define pii pair<int, int>
// {在 src 可到達
    的點中是否存在負環, 最短路徑}, arg 中 n 為點的數量
// arg 中 pair 裡的第一個值為權重, 第二個為點
pair<bool, vector<int>>
     SPFA(vector<vector<pii>> &graph, int n, int src){
  vector<int> dis(n + 1, 1e9);
  vector<int> cnt(n + 1, 0);
  vector<bool> vis(n + 1, false);
  queue<int> q;
  vis[src] = true; q.push(src); dis[src] = 0;
  while(!q.empty()){
    auto node = q.front(); vis[node] = false; q.pop();
    for(auto [w, nn]:graph[node]){
      if(w + dis[node] < dis[nn]){
        dis[nn] = w + dis[node];
        if(!vis[nn]){
          if(++cnt[nn] >= n) return {true, {}};
          q.push(nn);
          vis[nn] = true;
        }
      }
    }
  }
  return {false, dis};
}
```

3.4 Floyd Warshell

```
// 中繼點放外面

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

3.5 Tarjan SCC

```
class tarjan{
    //  1-base
    int time = 1;
    int id = 1;
    stack<int> s;
    vector<int> low;
    vector<int> dfn;
    vector<bool> in_stack;
    void dfs(int node, vector<vector<int>> &graph){
      in_stack[node] = true;
      s.push(node);
      dfn[node] = low[node] = time++;
      for(auto &j : graph[node]){
        if(dfn[j] == 0){
          dfs(j, graph);
          // 看看往下有沒有辦法回到更上面的點
          low[node] = min(low[node], low[j]);
        }
        else if(in_stack[j]){
          low[node] = min(low[node], low[j]);
        }
      }
      vector<int> t; // 儲存這個強連通分量
      if(dfn[node] == low[node]){
        while(s.top() != node){
          t.push_back(s.top());
          in_stack[s.top()] = false;
          scc_id[s.top()] = id;
          s.pop();
        }
        t.push_back(s.top());
        scc_id[s.top()] = id;
        in_stack[s.top()] = false;
        s.pop();
```

```
        id++;
      }
      if(!t.empty()) ans.push_back(t);
    }
  public:
    vector<int> scc_id;
    vector<vector<int>> ans;
    // ans ans[i] 代表第 i 個強連通分量裡面包涵的點
    // scc_id[i] 代表第 i 個點屬於第幾個強連通分量
    vector
        <vector<int>> scc(vector<vector<int>> &graph){
      int num = graph.size();
      scc_id.resize(num, -1);
      dfn.resize(num, 0);
      low.resize(num, 0);
      in_stack.resize(num, false);
      for(int i = 1; i < num; i++){
        if(dfn[i] == 0) dfs(i, graph);
      }
      return ans;
    }
};
```

3.6 2 SAT

```
// 用
    下面的 tarjan scc 算法來解 2 sat 問題，若 事件 a 發
    生時，事件 b 必然發生，我們須在 a -> b 建立一條有向
// 用
     cses 的 Giant Pizza 來舉例子，給定 n 個人 m 個配料
    表，每個人可以提兩個要求，兩個要求至少要被滿足一個

// 3 5
// + 1 + 2
// - 1 + 3
// + 4 - 2

// 以這
    個例子來說，第一個人要求要加 配料1 或者 配料2 其中
    一項，第二個人要求不要 配料1 或者 要配料3 其中一項
// 試問能不能滿足所有人的要求，我們可以把 要加
    配料 i 當作點 i ，不加配料 i 當作點 i + m(配料數量)
// 關於第一個人的要求 我們可以看成若不加 配
    料1 則必定要 配料2 以及 若不加 配料2 則必定要 配料1
// 關於第二個人要求 可看做加了 配料
    1 就必定要加 配料3 以及 不加 配料3 就必定不加 配料1

// 以這些條件建立有像圖，並且
    找尋 scc ，若 i 以及 i + m 在同一個 scc 中代表無解
// 若要求解，則若 i 的 scc_id
     小於 i + m 的 scc_id 則 i 為 true ，反之為 false
// tarjan 的模板在上面
cin >> n >> m;

vector<vector<int>> graph(m * 2 + 1);
function<int(int)> tr = [&](int x){
  if(x > m) return x - m;
  return x + m;
};

for(int i = 0; i < n; i++){
  char c1, c2;
  int a, b;
  cin >> c1 >> a >> c2 >> b;
  // a 代表 a 為真，m + a 代表 a 為假
  if(c1 == '-') a += m;
  if(c2 == '-') b += m;
  graph[tr(a)].push_back(b);
  graph[tr(b)].push_back(a);
}

tarjan t;
auto scc = t.scc(graph);

for(int i = 1; i <= m; i++){
  if(t.scc_id[i] == t.scc_id[tr(i)]){
    cout << "IMPOSSIBLE\n";
    return 0;
  }
}

for(int i = 1; i <= m; i++){
  if(t.scc_id[i] < t.scc_id[tr(i)]){
    cout << '+';
  }
  else cout << '-';
  cout << ' ';
}
cout << '\n';
```

3.7 Euler Path

```
// 1. 無向圖是歐拉圖：
// 非零度頂點是連通的
// 頂點的度數都是偶數

// 2. 無向圖是半歐拉圖(有路沒有環)：
// 非零度頂點是連通的
// 恰有 2 個奇度頂點

// 3. 有向圖是歐拉圖：
// 非零度頂點是強連通的
// 每個頂點的入度和出度相等

// 4. 有向圖是半歐拉圖(有路沒有環)：
// 非零度頂點是弱連通的
// 至多一個頂點的出度與入度之差為 1
// 至多一個頂點的入度與出度之差為 1
// 其他頂點的入度和出度相等
vector<set<int>> adj;
vector<int> ans;

void dfs(int x) { // Hierholzer's Algorithm
  while (!adj[x].empty()) {
    auto next = *(adj[x].begin());
    adj[x].erase(next);
    adj[next].erase(x);
    dfs(next);
  }
  ans.emplace_back(x);
}

void solve() {
  // 建立雙向邊，set用來防重邊，點數n，邊數m
  for (int i = 1; i <= n; i++)
    if (adj[i].size() & 1) return; /* impossible */
  dfs(1);
  if (ans.size() != m + 1) return; /* impossible */
  reverse(ans.begin(), ans.end()); /* then print it */
}
```

3.8 Max flow min cut

```
#define int long long

// dicnic Algorithm Time: O(V^2E) 實際上會快一點
// 記得在 main 裡面 resize graph
// 最小割，找
    到最少條的邊切除，使得從 src 到 end 的 maxflow 為 0
// 枚舉所有邊 i -> j ， src 可
    以到達 i 但無法到達 j ，那這條邊為最小割裡的邊之一
// 若求無向圖最大流 ， 則反向邊建邊為 capacity

class edge{
  public:
    int next;
    int capacity;
    int rev;
    bool is_rev;
    edge(int _n, int _c, int _r, int _ir) :
        next(_n), capacity(_c), rev(_r), is_rev(_ir){};
};

vector<vector<edge>> graph;
vector<int> level, iter;

void add_edge(int a, int b, int capacity){
  graph[a].push_back
      (edge(b, capacity, graph[b].size(), false));
  graph[b].
      push_back(edge(a, 0, graph[a].size() - 1, true));
}

void bfs(int start) {
  fill(level.begin(), level.end(), -1);
  queue<int> q;
```

```cpp
  level[start] = 0;
  q.push(start);
  while (!q.empty()) {
    int v = q.front();
    q.pop();
    for (auto& e : graph[v]) {
      if (e.capacity > 0 && level[e.next] < 0) {
        level[e.next] = level[v] + 1;
        q.push(e.next);
      }
    }
  }
}

int dfs(int v, int end, int flow) {
  if (v == end) return flow;
  for (int &i = iter[v]; i < graph[v].size(); i++) {
    edge &e = graph[v][i];
    if (e.capacity > 0 && level[v] < level[e.next]) {
      int d = dfs(e.next, end, min(flow, e.capacity));
      if (d > 0) {
        e.capacity -= d;
        graph[e.next][e.rev].capacity += d;
        return d;
      }
    }
  }
  return 0;
}

int maxflow(int start, int end) {
  int flow = 0;
  level.resize(graph.size() + 1);
  while (true) {
    bfs(start);
    if (level[end] < 0) return flow;
    iter.assign(graph.size() + 1, 0);
    int f;
    while ((f = dfs(start, end, 1e9)) > 0) {
      flow += f;
    }
  }
}
```

3.9 Minimum cost maximum flow

```cpp
#define int long long
#define pii pair<int, int>

// Edmonds-Karp Algorithm Time: O(VE^2) 實際上會快一點
// 一條邊的費用為 單位花費 * 流過流量
// 把原本的 BFS 換成 SPFA 而已
// 記得在 main 裡面 resize graph
// MCMF 回傳 {flow, cost}

class edge{
  public:
    int next;
    int capacity;
    int rev;
    int cost;
    bool is_rev;
    edge(int _n, int _c,
       int _r, int _co, int _ir) : next(_n), capacity
      (_c), rev(_r), cost(_co), is_rev(_ir){};
};

vector<vector<edge>> graph;

void add_edge(int a, int b, int capacity, int cost){
  graph[a].push_back(
      edge(b, capacity, graph[b].size(), cost, false));
  graph[b].push_back
      (edge(a, 0, graph[a].size() - 1, -cost, true));
}

pii dfs(int now
    , int end, pii data, vector<pii> &path, int idx){
  auto [flow, cost] = data;
  if(now == end) return {flow, 0};
  auto &e = graph[now][path[idx + 1].second];
  if(e.capacity > 0){
    auto [ret, nc] = dfs(e.next, end, {min(flow
        , e.capacity), cost + e.cost}, path, idx + 1);
    if(ret > 0){
      e.capacity -= ret;
      graph[e.next][e.rev].capacity += ret;
      return {ret, nc + ret * e.cost};
    }
  }
  return {0, 0};
}

vector<pii> search_path(int start, int end){
  int n = graph.size() + 1;
  vector<int> dis(n + 1, 1e9);
  vector<bool> vis(n + 1, false);
  vector<pii> ans; queue<int> q;
  vis[start] = true; q.push(start); dis[start] = 0;
  vector<pii> parent(graph.size(), {-1, -1});
  q.push(start);
  while(!q.empty()){
    auto node = q.front(); vis[node] = false; q.pop();
    for(int i = 0; i < graph[node].size(); i++){
      auto &e = graph[node][i];
      if(e.capacity
          > 0 and e.cost + dis[node] < dis[e.next]){
        dis[e.next] = e.cost + dis[node];
        parent[e.next] = {node, i};
        if(!vis[e.next]){
          q.push(e.next);
          vis[e.next] = true;
        }
      }
    }
  }
  if(parent[end].first == -1) return ans;
  int now = end;
  while(now != start){
    auto [node, idx] = parent[now];
    ans.emplace_back(node, idx);
    now = node;
  }
  ans.emplace_back(start, -1);
  reverse(ans.begin(), ans.end());
  return ans;
}

pii MCMF(int start, int end){
  int ans = 0, cost = 0;
  while(1){
    vector<bool> visited(graph.size() + 1, false);
    auto tmp = search_path(start, end);
    if(tmp.size() == 0) break;
    auto [flow, c] = dfs(start, end, {1e9, 0}, tmp, 0);
    ans += flow;
    cost += c;
  }
  return {ans, cost};
}
```

3.10 二分圖

```cpp
/*
判定二分圖: 著色法 dfs 下去，顏色相撞非二分圖

二分圖最大匹配: 用 maxflow 去做，一個 src
     點聯通所有左圖，左圖建邊向右圖，右圖再建邊向 end
     點，計算 src 跟 end 的最大流，若要還原，找出左圖
    通往右圖中 capacity 為 0 的邊，他的兩個端點就是答案

最小點覆蓋: 選最少的點，保證每條邊
    至少有一個端點被選到， 最小點覆蓋 = 二分圖最大匹配

最大獨立集: 選最多的點，滿足這些
    點兩兩間互不相連， 最大獨立集 = n - 二分圖最大匹配
*/
```

4 String

4.1 trie

```cpp
class trie{
  public:
    class node{
      public:
        int count;
        vector<trie::node*> child;
        node(){
          child.resize(26, nullptr);
```

```
          count = 0;
        }
        ~node() {
          for (auto c : child)
            if (c) delete c;
        }
    };
    node* root;
    trie(){
      root = new node;
    }
    ~trie() {
      delete root;
    }
    void insert(string s){
      auto temp = root;
      for(int i = 0; i < s.size(); i++){
        if(!temp -> child[s[i] -
            'a']) temp -> child[s[i] - 'a'] = new node;
        temp = temp -> child[s[i] - 'a'];
      }
      temp -> count++;
    }
    bool search(string &s){
      auto temp = root;
      for(int i = 0; i < s.size(); i++){
        temp = temp -> child[s[i] - 'a'];
        if(!temp) return false;
      }
      if(temp -> count > 0) return true;
      return false;
    }
};
```

4.2 KMP

```
vector<int> build(string &s){
  vector<int> next = {0, 0};
  // 匹配失敗跳去哪 (最長共同前後綴)
  int length = s.size(), j = 0;
  for(int i = 1; i < length; i++){
    while(j > 0 and s[j] != s[i]){
      j = next[j];
    }
    if(s[j] == s[i]) j++;
    next.push_back(j);
  }
  return next;
}

int match(string &a, string &b){
  auto next = build(b);
  int length
      = a.size(), length2 = b.size(), j = 0, count = 0;
  for(int i = 0; i < length; i++){
    while(j > 0 and a[i] != b[j]){
      j = next[j];
    }
    if(a[i] == b[j]) j++;
    if(j == length2){
      count++;
      j = next[j];
    }
  }
  return count;
}
```

4.3 Hash

```
vector<int> Pow(int num){
  int p = 1e9 + 7;
  vector<int> ans = {1};
  for(int i = 0; i < num; i++)
    ans.push_back(ans.back() * b % p);
  return ans;
}

vector<int> Hash(string s){
  int p = 1e9 + 7;
  vector<int> ans = {0};
  for(char c:s){
    ans.push_back((ans.back() * b + c) % p);
  }
  return ans;
}
```

```
// 閉區間[l, r]
int query
    (vector<int> &vec, vector<int> &pow, int l, int r){
  int p = 1e9 + 7;
  int length = r - l + 1;
  return
       (vec[r + 1] - vec[l] * pow[length] % p + p) % p;
}
```

4.4 Zvalue

```
vector<int> z_func(string s1){
  int l = 0, r = 0, n = s1.size();
  vector<int> z(n, 0);
  for(int i = 1; i < n; i++){
    if(i
        <= r and z[i - l] < r - i + 1) z[i] = z[i - l];
    else{
      z[i] = max(z[i], r - i + 1);
      while(i + z
          [i] < n and s1[i + z[i]] == s1[z[i]]) z[i]++;
    }
    if(i + z[i] - 1 > r){
      l = i;
      r = i + z[i] - 1;
    }
  }
  return z;
}
```

4.5 最長迴文子字串

```
// 找到對於每個位置的迴文半徑
vector<int> manacher(string s) {
  string t = "#";
  for (auto c : s) {
    t += c;
    t += '#';
  }
  int n = t.size();
  vector<int> r(n);
  for (int i = 0, j = 0; i
       < n; i++) {    // i 是中心, j 是最長回文字串中心
    if (2 * j - i >= 0 && j + r[j] > i) {
      r[i] = min(r[2 * j - i], j + r[j] - i);
    }
    while (i - r[i] >= 0 &&
         i + r[i] < n && t[i - r[i]] == t[i + r[i]]) {
      r[i] += 1;
    }
    if (i + r[i] > j + r[j]) {
      j = i;
    }
  }
  return r;
  // # a # b # a #
  // 1 2 1 4 1 2 1
  // # a # b # b # a #
  // 1 2 1 2 5 2 1 2 1
  // 值 -1 代表原回文字串長度
  // (id - val + 1) / 2 可得原字串回文開頭
}
```

4.6 Suffix Array

```
struct SuffixArray {
  int n; string s;
  vector<int> sa, rk, lc;
  // 想法 :
       排序過了，因此前綴長得像的會距離很近在差不多位置
  // n: 字串長度
  // sa: 後綴數組，sa[i] 表示第 i 小的後綴的起始位置
  // rk: 排名數組，rk[i] 表示從位置 i 開始的後綴的排名
  // lc: LCP 數組，
      lc[i] 表示 sa[i] 和 sa[i + 1] 的最長公共前綴長度
  // 求 sa[i] 跟 sa[j] 的
       LCP 長度 當 i < j : min(lc[i] ...... lc[j - 1])
  SuffixArray(const string &s_) {
    s = s_; n = s.length();
    sa.resize(n);
    lc.resize(n - 1);
    rk.resize(n);
    iota(sa.begin(), sa.end(), 0);
    sort(sa.begin(), sa.end
        (), [&](int a, int b) { return s[a] < s[b]; });
```

```
    rk[sa[0]] = 0;
    for (int i = 1; i < n; ++i)
      rk[sa[i]]
        = rk[sa[i - 1]] + (s[sa[i]] != s[sa[i - 1]]);
    int k = 1;
    vector<int> tmp, cnt(n);
    tmp.reserve(n);
    while (rk[sa[n - 1]] < n - 1) {
      tmp.clear();
      for (int i = 0; i < k; ++i)
        tmp.push_back(n - k + i);
      for (auto i : sa)
        if (i >= k)
          tmp.push_back(i - k);
      fill(cnt.begin(), cnt.end(), 0);
      for (int i = 0; i < n; ++i)
        ++cnt[rk[i]];
      for (int i = 1; i < n; ++i)
        cnt[i] += cnt[i - 1];
      for (int i = n - 1; i >= 0; --i)
        sa[--cnt[rk[tmp[i]]]] = tmp[i];
      swap(rk, tmp);
      rk[sa[0]] = 0;
      for (int i = 1; i < n; ++i)
        rk[sa[i]] = rk[sa[i - 1]] + (tmp[
            sa[i - 1]] < tmp[sa[i]] || sa[i - 1] + k ==
             n || tmp[sa[i - 1] + k] < tmp[sa[i] + k]);
      k *= 2;
    }
    for (int i = 0, j = 0; i < n; ++i) {
      if (rk[i] == 0) {
        j = 0;
      } else {
        for (j -= j > 0; i + j < n && sa[rk[i] - 1] + j
             < n && s[i + j] == s[sa[rk[i] - 1] + j]; )
          ++j;
        lc[rk[i] - 1] = j;
      }
    }
  }
};
```

5 Geometry

5.1 Point

```
template<typename T>
class point{
    public:
    T x;
    T y;
    point(){}
    point(T _x, T _y){
        x = _x;
        y = _y;
    }
    point<T> operator+(const point<T> &a);
    point<T> operator-(const point<T> &a);
    point<T> operator/(const point<T> &a);
    point<T> operator/(T a);
    point<T> operator*(const T &a);
    bool operator<(const point<T> &a);
};

template<typename T>
point<T> point<T>::operator+(const point<T> &a){
    return point<T>(x + a.x, y + a.y);
}

template<typename T>
point<T> point<T>::operator-(const point<T> &a){
    return point<T>(x - a.x, y - a.y);
}

template<typename T>
point<T> point<T>::operator/(const point<T> &a){
    return point<T>(x / a.x, y / a.y);
}

template<typename T>
point<T> point<T>::operator/(T a){
    return point<T>(x / a, y / a);
}

template<typename T>
point<T> point<T>::operator*(const T &a){
    return point<T>(x * a, y * a);
}

template<typename T>
bool point<T>::operator<(const point<T> &a){
    if(x != a.x) return x < a.x;
    return y < a.y;
}
```

5.2 內積, 外積, 距離

```
template<typename T>
T dot(const point<T> &a,const point<T> &b){
    return a.x * b.x + a.y * b.y;
}

template<typename T>
T cross(const point<T> &a,const point<T> &b){
    return a.x * b.y - a.y * b.x;
}

template<typename T>
T len(point<T> p){
    return sqrt(dot(p, p));
}
```

5.3 向量應用

```
template<typename T>
bool collinearity
    (point<T> p1, point<T> p2, point<T> p3){
    //檢查三點是否共線
    return cross(p2 - p1, p2 - p3) == 0;
}

template<typename T>
bool inLine(point<T> a, point<T> b, point<T> p){
    //檢查 p 點是否在ab線段
    return collinearity
        (a, b, p) && dot(a - p, b - p) <= 0;
}

template<typename T>
bool intersect
    (point<T> a, point<T> b, point<T> c, point<T> d){
    //ab線段跟cd線段是否相交
    return (cross(b - a, c - a) * \
        cross(b - a, d - a) < 0 && \
        cross(d - c, a - c) * \
        cross(d - c, b - c) < 0) \
        || inLine(a, b, c) || \
        inLine(a, b, d) || inLine(c, d, a) \
        || inLine(c, d, b);
}

template<typename T>
point<T> intersection
    (point<T> a, point<T> b, point<T> c, point<T> d){
    //ab線段跟cd線段相交的點
    assert(intersect(a, b, c, d));
    return a + (b -
        a) * cross(a - c, d - c) / cross(d - c, b - a);
}

template<typename T>
bool inPolygon(vector<point<T>> polygon, point<T> p){
    //判斷點
        p是否在多邊形polygon裡，vector裡的點要連續填對
    for(int i = 0; i < polygon.size(); i++)
        if(cross(p - polygon[i], \
            polygon[(i - 1 + polygon.size()) % \
            polygon.size()] - polygon[i]) * \
            cross(p - polygon[i], \
            polygon[(i +
              1) % polygon.size()] - polygon[i]) > 0)
            return false;
    return true;
}

template<typename T>
T triangleArea(point<T> a, point<T> b, point<T> c){
    //三角形頂點，求面積
    return abs(cross(b - a, c - a)) / 2;
}
```

```
template<typename T,typename F,typename S>
long double triangleArea_Herons_formula(T a, F b, S c){
    //三角形頂點，求面積(給邊長)
    auto p = (a + b + c)/2;
    return sqrt(p * (p - a) * (p - b) * (p - c));
}

template<typename T>
T area(vector<point<T>> &p){
    //多邊形頂點，求面積
    T ans = 0;
    for(int i = 0; i < p.size(); i++)
        ans += cross(p[i], p[(i + 1) % p.size()]);
    return ans / 2 > 0 ? ans / 2 : -ans / 2;
}
```

5.4 Static Convex Hull

```
// 需要使
    用前一個向量模板的 point ， 需要 operator - 以及 <
// 需要前面向量模板的 cross

template<typename T>
vector<point<T>> getConvexHull(vector<point<T>>& pnts){
    sort(pnts.begin(), pnts.end());
    auto cmp = [&](point<T> a, point<T> b)
    { return a.x == b.y && a.x == b.y; };
    pnts.erase(unique
        (pnts.begin(), pnts.end(), cmp), pnts.end());
    if(pnts.size()<=1) return pnts;
    vector<point<T>> hull;
    for(int i = 0; i < 2; i++){
        int t = hull.size();
        for(point<T> pnt : pnts){
            while(hull.size() - t >= 2 &&
                cross(hull.back() - hull[hull.size()
                - 2], pnt - hull[hull.size() - 2]) < 0)
                // <= 0 或者 < 0 要看點有沒有在邊上
                hull.pop_back();
            hull.push_back(pnt);
        }
        hull.pop_back();
        reverse(pnts.begin(), pnts.end());
    }
    return hull;
}
```

5.5 外心,最小覆蓋圓

```
int sign(double a)
{
  // 小於 eps
      回傳 0，否則正回傳 1 ，負回傳 應付浮點數誤差用
  const double eps = 1e-10;
  return fabs(a) < eps ? 0 : a > 0 ? 1 : -1;
}

// 輸入三個點求外心
template <typename T>
point<T> findCircumcenter(point<
    T> A, point<T> B, point<T> C, const T eps = 1e-10){
    point<T> AB = B - A;
    point<T> AC = C - A;
    T AB_len_sq = AB.x * AB.x + AB.y * AB.y;
    T AC_len_sq = AC.x * AC.x + AC.y * AC.y;
    T D = AB.x * AC.y - AB.y * AC.x;
    // 若三點接近共線
    assert(fabs(D) < eps);
    // 外心的座標
    T circumcenterX = A.x + (
        AC.y * AB_len_sq - AB.y * AC_len_sq) / (2 * D);
    T circumcenterY = A.y + (
        AB.x * AC_len_sq - AC.x * AB_len_sq) / (2 * D);
    return point<T>(circumcenterX, circumcenterY);
}

template<typename T>
pair<T, point<T>> MinCircleCover(vector<point<T>> &p) {
    // 引入前面的 len 跟 point
    // 回傳最小覆蓋圓{半徑,中心}
    random_shuffle(p.begin(), p.end());
    int n = p.size();
    point<T> c = p[0]; T r = 0;
    for(int i=1;i<n;i++) {
        if(sign(len(c-p[i])-r) > 0) { // 不在圓內
            c = p[i], r = 0;
            for(int j=0;j<i;j++) {
                if(sign(len(c-p[j])-r) > 0) {
                    c = (p[i]+p[j])/2.0;
                    r = len(c-p[i]);
                    for(int k=0;k<j;k++) {
                        if(sign(len(c-p[k])-r) > 0) {
                            c = findCircumcenter
                                (p[i],p[j],p[k]);
                            r = len(c-p[i]);
                        }
                    }
                }
            }
        }
    }
    return make_pair(r, c);
}
```

6 Data Structure

6.1 Sparse Table

```
class Sparse_Table{
  // 0-base
  // 要改成找最大把min換成max就好
  private:
  public:
    int spt[500005][22][2];
    Sparse_Table(vector<int> &ar){
      int n = ar.size();
      for (int i = 0; i < n; i++){
        spt[i][0][0] = ar[i];
        // spt[i][0][1] = ar[i];
      }
      for (int j = 1; (1 << j) <= n; j++) {
        for (int i = 0; (i + (1 << j) - 1) < n; i++) {
          spt[i][j][0] = min(spt[i + (1 <<
              (j - 1))][j - 1][0], spt[i][j - 1][0]);
          // spt[i][j][1] = max(spt[i + (1 <<
              (j - 1))][j - 1][1], spt[i][j - 1][1]);
        }
      }
    }
    int query_min(int l, int r)
    {
      if(l>r) return INT_MAX;
      int j = (int)__lg(r - l + 1);
      ///j = 31 - __builtin_clz(r - l+1);
      return min
        (spt[l][j][0], spt[r - (1 << j) + 1][j][0]);
    }
    int query_max(int l, int r)
    {
      if(l>r) return INT_MAX;
      int j = (int)__lg(r - l + 1);
      ///j = 31 - __builtin_clz(r - l+1);
      return max
        (spt[l][j][1], spt[r - (1 << j) + 1][j][1]);
    }
};
```

6.2 Segement Tree

```
// #define int long long

// 要改最大或
    者最小值線段樹需改 build 跟 queryRange, updateRange
// 0-base 注意
template<typename T>
class segment_tree {
private:
  vector<T> tree, lazy, arr;
  int size;
  void build
      (vector<T> &save, int node, int start, int end) {
    if (start == end) tree[node] = save[start];
    else {
      int mid = (start + end) / 2;
      build(save, 2 * node, start, mid);
      build(save, 2 * node + 1, mid + 1, end);
      tree[node] = tree[2 * node] + tree[2 * node + 1];
    }
  }
```

```cpp
  void updateRange(int node
      , int start, int end, int l, int r, T delta) {
    if (lazy[node] != 0) {
      tree[node] += (end - start + 1) * lazy[node];
      if (start != end) {
        lazy[2 * node] += lazy[node];
        lazy[2 * node + 1] += lazy[node];
      }
      lazy[node] = 0;
    }
    if (start > end or start > r or end < l) return;
    if (start >= l and end <= r) {
      tree[node] += (end - start + 1) * delta;
      if (start != end) {
        lazy[2 * node] += delta;
        lazy[2 * node + 1] += delta;
      }
      return;
    }
    int mid = (start + end) / 2;
    updateRange(2 * node, start, mid, l, r, delta);
    updateRange
        (2 * node + 1, mid + 1, end, l, r, delta);
    tree[node] = tree[2 * node] + tree[2 * node + 1];
  }
  T queryRange
      (int node, int start, int end, int l, int r) {
    if (lazy[node] != 0) {
      tree[node] += (end - start + 1) * lazy[node];
      if (start != end) {
        lazy[2 * node] += lazy[node];
        lazy[2 * node + 1] += lazy[node];
      }
      lazy[node] = 0;
    }
    if (start > end or start > r or end < l){
      // return numeric_limits
          <T>::max(); // 找區間最小值用這行
      // return numeric_limits
          <T>::min(); // 找區間最大值用這行
      return 0; // 區間和
    }
    if (start >= l and end <= r) return tree[node];
    int mid = (start + end) / 2;
    T p1 = queryRange(2 * node, start, mid, l, r);
    T p2
        = queryRange(2 * node + 1, mid + 1, end, l, r);
    return p1 + p2;
  }
  void updateNode(
      int node, int start, int end, int idx, T delta) {
    if (start == end) tree[node] += delta;
    else {
      int mid = (start + end) / 2;
      if (start <= idx and idx <= mid)
          updateNode(2 * node, start, mid, idx, delta);
      else updateNode
          (2 * node + 1, mid + 1, end, idx, delta);
      tree[node] = tree[2 * node] + tree[2 * node + 1];
    }
  }

public:
  void build(vector<T> &save, int l, int r) {
    int n = size = save.size();
    tree.resize(4 * n);
    lazy.resize(4 * n);
    arr = save;
    build(save, 1, l, r);
  }
  void modify_scope(int l, int r, T delta) {
    updateRange(1, 0, size - 1, l, r, delta);
  }
  void modify_node(int idx, T delta) {
    updateNode(1, 0, size - 1, idx, delta);
  }
  T query(int l, int r) {
    return queryRange(1, 0, size - 1, l, r);
  }
};

signed main()
{
  int n, q;
  cin >> n >> q;
  vector<int> save(n, 0);
  for(int i = 0; i < n; i++){
    cin >> save[i];
  }
  segment_tree<int> s;
  // init [0, n - 1]
  s.build(save, 0, n - 1);
  // modify [a, b] add c
  s.modify_scope(a, b, c);
  // query [a, b]
  s.query(a, b)
}
```

6.3 Link Cut Tree

```cpp
// 通常用於對樹上任兩點間的路徑做加值、修改、查詢等工作
// 與線段樹相同，要修改 LCT 的功能只需更改
// pull、push、fix、query 等函數，再加上需要的懶標即可
// 範例為樹上任兩點 x, y 路徑上的權值 xor
// 和，樹上任意點單點改值
const int N = 300005;
class LinkCutTree {
private:
#define lc(x) node[x].ch[0]
#define rc(x) node[x].ch[1]
#define fa(x) node[x].fa
#define rev(x) node[x].rev
#define val(x) node[x].val
#define sum(x) node[x].sum
  struct Tree {
    int val, sum, fa, rev, ch[2];
  } node[N];
  inline void pull(int x) {
    sum(x) = val(x) ^ sum(lc(x)) ^ sum(rc(x));
  }
  inline void reverse(int x) {
    swap(lc(x), rc(x));
    rev(x) ^= 1;
  }
  inline void push(int x) {
    if (rev(x)) {
      reverse(lc(x));
      reverse(rc(x));
      rev(x) ^= 1;
    }
  }
  inline bool get(int x) { return rc(fa(x)) == x; }
  inline bool isroot(int x) {
    return (lc(fa(x)) ^ x) && (rc(fa(x)) ^ x);
  }
  inline void update(int x) {
    if (!isroot(x)) update(fa(x));
    push(x);
  }
  void rotate(int x) {
    int y = fa(x), z = fa(y), d = get(x);
    if (!isroot(y))
      node[z].ch[get(y)] = x; // 重要，不能更換順序
    fa(x) = z;
    node[fa(node[x].ch[d ^ 1]) = y].ch[d] =
      node[x].ch[d ^ 1];
    node[fa(y) = x].ch[d ^ 1] = y;
    pull(y), pull(x); // 先 y 再 x
  }
  void splay(int x) {
    update(x);
    for (int y = fa(x); !isroot(x);
         rotate(x), y = fa(x)) {
      if (!isroot(y)) rotate(get(x) == get(y) ? y : x);
    }
    pull(x);
  }
  int access(int x) {
    int p = 0;
    for (; x; x = fa(p = x)) {
      splay(x), rc(x) = p, pull(x);
    }
    return p;
  }
  inline void makeroot(int x) {
    access(x), splay(x), reverse(x);
  }
  inline int findroot(int x) {
    access(x), splay(x);
    while (lc(x)) { push(x), x = lc(x); }
```

```
    return splay(x), x;
  }
  inline void split(int x, int y) {
    makeroot(x), access(y), splay(y);
  }

public:
  inline void init(int len, int *data) {
    for (int i = 1; i <= len; ++i) {
      node[i].val = data[i];
    }
  }
  inline void link(int x, int y) { // 連邊
    makeroot(x);
    if (findroot(y) == x) return;
    fa(x) = y;
  }
  inline void cut(int x, int y) { // 斷邊
    makeroot(x);
    if (findroot(y) != x || fa(y) != x || lc(y))
      return;
    fa(y) = rc(x) = 0;
    pull(x);
  }
  inline void fix(int x, int v) { // 單點改值
    splay(x);
    val(x) = v;
  }
  // 區間查詢
  inline int query(int x, int y) {
    return split(x, y), sum(y);
  }
};

LinkCutTree LCT;
int n, a[N];

signed main() {
  int n, q, op, x, y;
  cin >> n >> q;
  for (int i = 1; i <= n; ++i) { cin >> a[i]; }
  LCT.init(n, a);
  while (q--) {
    cin >> op >> x >> y;
    if (op == 0) {
      cout << LCT.query(x, y) << endl;
    } else if (op == 1) {
      LCT.link(x, y);
    } else if (op == 2) {
      LCT.cut(x, y);
    } else {
      LCT.fix(x, y);
    }
  }
  return 0;
}
```

7 Dynamic Programing

7.1 LCS

```
int LCS(string t1, string t2) {
  if(t1.size() < t2.size()) swap(t1, t2);
  int len = t1.size();
  vector<vector<int>> dp(2, vector<int>(len + 1, 0));
  for(int j = 1; j <= t2.size(); j++){
    for(int i = 1; i <= len; i++){
      if(t2[j - 1] == t1[i - 1])
           dp[j % 2][i] = dp[(j + 1) % 2][i - 1] + 1;
      else dp[j % 2][i]
           = max(dp[(j + 1) % 2][i], dp[j % 2][i - 1]);
    }
  }
  return dp[t2.size() % 2][t1.size()];
}
```

7.2 LIS

```
int LIS(vector<int>& save) {
  vector<int> dp;
  int n = save.size();
  for (int i = 0; i < n; i++) {
    auto it = lower_bound(dp.begin(),dp.end(),save[i]);
    if(it == dp.end()) dp.push_back(save[i]);
    else *it = save[i];
  }
  return dp.size();
}
```

7.3 位元 dp

```
// 檢查第 n 位是否為1
if(a & (1 << n))

// 強制將第 n 位變成1
a |= (1 << n)

// 強制將第 n 位變成0
a &= ~(1 << n)

// 將第 n 位反轉(1變0, 0變1)
a ^= (1 << n)

// 第 0 ~ n - 1位 全部都是1
(1 << n) - 1

// 兩個集合的聯集
S = a | b

// 兩個集合的交集
S = a & b
```

7.4 經典 dp 轉移式

```
/*
最大區間和:

dp[i] 代表 由第 i 項結尾時的最大區間和
dp[0] = arr[0]
dp[i] = max(dp[i - 1], arr[i])
ans = max_element(dp)

*/
```

8 Divide and conquer

8.1 逆序數對

```
int merge(
      vector<pair<int, int>>& v, int l, int mid, int r) {
  vector<pair<int, int>> temp(r - l + 1);
  int i = l, j = mid + 1, k = 0, inv_count = 0;
  while (i <= mid && j <= r) {
      if (v[i].second <= v[j].second) {
          temp[k++] = v[i++];
      } else {
          temp[k++] = v[j++];
          inv_count += (mid - i + 1);
      }
  }
  while (i <= mid) temp[k++] = v[i++];
  while (j <= r) temp[k++] = v[j++];
  for (int i = l; i <= r; i++) {
    v[i] = temp[i - l];
  }
  return inv_count;
}

int mergeSort
      (vector<pair<int, int>>& v, int l, int r) {
  int count = 0;
  if (l < r) {
    int mid = l + (r - l) / 2;
    count += mergeSort(v, l, mid);
    count += mergeSort(v, mid + 1, r);
    count += merge(v, l, mid, r);
  }
  return count;
}

signed main()
{
  int n;
  cin >> n;
  vector<pair<int, int>> arr(n);
  for(int i = 0; i < n; i++){
    arr[i].first = i;
    cin >> arr[i].second;
  }
  cout << mergeSort(arr, 0, n - 1) << '\n';
}
```

9 Tree

9.1 樹直徑

```
int dis[200005];
int c;

void dfs(int now, int fa, vector<vector<int>> &graph){
  for(auto i: graph[now]){
    if(i != fa){
      dis[i] = dis[now] + 1;
      if(dis[i] > dis[c]) c = i;
      dfs(i, now, graph);
    }
  }
}

signed main()
{
  int n;
  cin >> n;
  vector<vector<int>> graph(n + 1);
  for(int i = 0; i < n - 1; i++){
    int a, b;
    cin >> a >> b;
    graph[a].push_back(b);
    graph[b].push_back(a);
  }
  dfs(1, 0, graph);
  dis[c] = 0;
  dfs(c, 0, graph);
  cout << dis[c];
}
```

9.2 LCA

```
// n 為點數，graph 由子節點往父節點建有向邊
// graph 要 resize

int n, q;
int fa[20][200001];
int dep[200001];

vector<vector<int>> graph;

void dfs(int now, int lst){
  fa[0][now] = lst;
  for(int &i:graph[now]){
    dep[i] = dep[now] + 1;
    dfs(i, now);
  }
}

void build_lca(int root){
  dep[root] = 1;
  dfs(root, root);
  for(int i = 1; i < 18; i++){
    for(int j = 1; j < n + 1; j++){
      fa[i][j] = fa[i - 1][fa[i - 1][j]];
    }
  }
}

int lca(int a, int b){
  // 預設a比b淺
  if(dep[a] > dep[b]) return lca(b, a);
  // 讓a和b跳到同一個地方
  int step = dep[b] - dep[a];
  for (int i = 0; i < 18; i++)
  {
    if(step >> i & 1){
      b = fa[i][b];
    }
  }
  if(a == b) return a;
  for(int i = 17; i >= 0; i--){
    if(fa[i][a] != fa[i][b]){
      a = fa[i][a];
      b = fa[i][b];
    }
  }
  return fa[0][a];
}
```