RAUCODER 2024

INDICAȚII DE REZOLVARE

Data: **25 mai 2024**

Problema 1. Donjon

Autor: Crișan Daniela Alexandra

Cuvinte cheie: operatori pe biți, Jocul lui Nim

Indicații

In prima parte a problemei trebuie verificat, pentru fiecare scenariu, dacă există un număr natural *a* astfel încât *(a+1) + (a+2) + ... + (a+k) = n*, unde *n* este numărul de pietre, iar *k* este lățimea canalului de apărare.
Dar *(a+1) + (a+2) + ... + (a+k) = S(a+k) - S(a)*, unde *S(x)* este suma Gauss *x*(x+1)/2*.
Relația *S(a+k) - S(a) = n* se rescrie *a*k = n – k*(k + 1)/2*, de unde rezultă că *n – k*(k + 1)/2* trebuie să fie multiplu, număr natural, de *k*. Dacă este, se trece la cea de-a doua provocare.

Proba a doua constă în clasicul Joc al lui Nim. Practic, RAU-Gigel, fiind cel care începe jocul, va pierde numai dacă suma XOR a tuturor dimensiunilor treptelor este 0, adică: *(a+1) ^ (a+2) ^ ... ^ (a+k) = 0,* unde *a* a fost dedus conform relației de mai sus. Această suma XOR se poate calcula liniar (soluție de maxim *50* puncte) sau în timp constant, folosind observația că orice secvență de *4* numere naturale care începe cu un multiplu a lui *4* are suma XOR egală cu 0: *{ 0, 1, 2, 3}, {4, 5, 6, 7}, {8, 9, 10, 11}, {12, 13, 14, 15}*, …
Dacă *XOR(x) = 0 ^ 1 ^ 2 ^ ... ^ x*, atunci avem că: *(a+1) ^ (a+2) ^ ... ^ (a+k) = XOR(a) ^ XOR(a+k)*.

Pentru fiecare scenariu:
Complexitate de timp: O(1)
Complexitate de memorie: O(1)

* o soluție liniară în timp (pe scenariu) ar obține maxim 50p.

Soluție

```
#include <fstream>
using namespace std;

ifstream in("donjon.in");
ofstream out("donjon.out");

int XOR(long long  n)
{
    int mod = n % 4;
    if (mod == 0)
        return n;
    if (mod == 1)
        return 1;
    if (mod == 2)
        return n + 1;
    return 0;
}

int main()
{
    int Q;
    long long n, k, a, cat;
    in >> Q;
    for (int i = 1; i <= Q; ++i)
    {
        in >> n >> k;
        a = n - k * (k + 1) / 2;
        cat = a / k;
        if (a < 0 || cat * k != a)
            out << "C\n";
        else
        {
            a = cat;
            out << ((XOR(a) ^ XOR(a + k)) != 0 ? 'G' : 'C') << " " << a + 1 << '\n';
        }
    }
    return 0;
 }
```

Problema 2. Scrabble

Autor: Crișan Daniela Alexandra

Cuvinte cheie: măști de biți, programare dinamică

Indicații

Avem *n* cuvinte. Reținem într-o mască literele pe care le conține fiecare cuvânt, masca este chiar valoarea cuvântului. De exemplu, cuvântul *ABD* va avea masca: *1*2^0 (A) + 1*2^1 (B) + 0*2^3 (C) + 1*2^4 (D) = 11*.
(am notat cu ^ ridicarea la putere).

Pentru rezolvarea problemei, vom folosi programarea dinamică.
Dacă:

COST(i,j) = costul minim necesar pentru a uni secvența de cuvinte de la *i* la *j*, când *i < j*,
cuv(i,j) = masca obținută în urma uniunii secvenței de cuvinte de la *i* la *j*, când *i < j*,
cost(a,b) = costul, conform enunțului, necesar pentru a uni cuvintele a și b,

atunci avem relația de recurență:

COST(i,j)=minim i≤k<j (COST(i,k) + cost(cuv(i,k), cuv(k+1,j) + COST(k+1,j), când i<j.
COST(i,j) = 0, când i=j

Putem folosi o singură matrice în care vom reține deasupra diagonalei valorile *COST(i,j), i < j*, iar pe pozițiile simetrice măștile *cuv(i,j), i < j.*

Ne interesează valorile *A[1][n]* = costul minim necesar pentru a uni secventa de cuvinte de la *1* la *n*, respectiv cuvântul care are masca reținută în *A[n][1]*.

Complexitate de timp: O(n^3).
Complexitate de memorie: O(n^2).

Soluție

```
#include <fstream>
using namespace std;

ifstream in("scrabble.in");
ofstream out("scrabble.out");

typedef long long ll

#define min(x,y) (x)<(y)?(x):(y)
#define LLONG_MAX 1e18

int masca(char* str)
{
    int m = 0;
    for (int i = 0; str[i]; ++i)
        m |= (1 << ((str[i] >= 'a' && str[i] <= 'z' ? str[i] - 32 : str[i]) - 'A'));
    return m;
}

void afis(int masca)
{
    for (ll i = 0; i < 26; ++i)
        if (masca >> i & 1)
            out << (char)(i + 'A');
}

ll uneste(ll a, ll b, ll& c)
{
    c = a | b;
    return a ^ b;
}

ll dp[100][101]; int n;

void DP()//iterativă, cu o singura matrice
{
    for (int lg = 1; lg < n; ++lg)
        for (int i = 0; i < n - lg; ++i){
            int j = i + lg;
            dp[i + lg][i] = dp[i + lg - 1][i] | dp[i + lg][i + 1];
            dp[i][j+1] = LLONG_MAX;
            for (int k = i; k < j; ++k)
      dp[i][j+1] = min(dp[i][j+1], dp[i][k+1] + dp[k+1][j+1] + (dp[k][i] ^ dp[j][k+1]));
        }
}

int main()
{
    in >> n;
    char s[101];
    for (int i = 0; i < n; ++i){
        in >> s;
        dp[i][i] = masca(s);
    }

    DP();
    afis(dp[n - 1][0]);
```

```
    out << ' ' << dp[0][n]<<endl;

    return 0;
}
```

Problema 3. Gcd Tree

Autor: Lorintz Alexandru

Cuvinte cheie: Cel mai mare divizor comun, Cel mai mic multiplu comun, Algoritmul lui Euclid, Arbori de intervale, Cel mai apropiat strămoș comun, Binary lifting, Heavy Light Decomposition

Indicații

Observația cheie a problemei este că dacă considerăm toate restricțiile care afectează un nod din arbore fixat, este optim să încercăm să-i asignăm acestuia valoarea **celui mai mic multiplu comun al acestor valori** și să verificăm la final dacă restricțiile se păstrează.

După ce se ajunge la observația anterioară, problema se reduce la cum se implementează eficient soluția. Pentru cazurile în care arborele este un simplu lanț, se poate folosi un **arbore de intervale**, de exemplu făcând o **baleiere** off-line a restricțiilor și la fiecare element să se obțină **cel mai mic multiplu comun al tuturor restricțiilor** care îl afectează sau folosind direct arborele pentru a „sparge" un interval al unei restricții în intervale din arbore pe care aplicăm restricția și apoi să obținem la final valoarea fiecărui element, combinând valorile din toate intervalele din structura de date care conțin elementul respectiv. Ambele abordări descrise au complexitatea **O(N * logN * logVAL_MAX)**.

Pentru a generaliza soluția pe arbore, se poate pleca de la soluțiile pe lanț și ori să se creeze restricții care „încep" în **capetele** unui lanț și „se termină" în **cel mai apropiat strămoș comun** al celor două capete (pentru prima soluție descrisă anterior), folosind apoi același tip de **arbore de intervale**, ori să se folosească **Heavy Light Decomposition** (pentru a doua soluție descrisă anterior). Se menționează că prima abordare ar avea complexitatea O**(N * logN * logVAL_MAX)**, pe când a doua are complexitatea O**(N * logN * logN * logVAL_MAX)**(factorul logaritmic adițional vine de la **Heavy Light Decompostion)**, dar ambele soluții ar trebui să obțină punctaj maxim, pentru că a doua soluție are constantă bună, deci se comportă bine în practică.

Va propunem mai jos două soluții, de 100p, respectiv 73p.

Soluție 100p

```
#include <fstream>
#include <vector>
#include <tuple>

using namespace std;

#define int long long


const int kN = 1e5;
const int kLog = 17;
vector<int> g[1 + kN];
int par[1 + kN];
int anc[1 + kLog][1 + kN];
int timer, tin[1 + kN], tout[1 + kN];
int dep[1 + kN];
int sub[1 + kN], chainTop[1 + kN], heavySon[1 + kN];
tuple<int, int, int> treeRestriction[1 + kN];
int nodeValue[1 + kN], gcdTable[1 + kLog][1 + kN];

int gcd(int x, int y) {
  if (x == 0 || y == 0) {
    return x + y;
  }

  while (y) {
    int r = x % y;
    x = y;
    y = r;
  }

  return x;
}

class SegTree {
  private:
    int n;
    vector<int> t;

    int queryRes;

    int combine(int x, int y) {
      if (x == -1) {
        return y;
      }

      if (y == -1) {
        return x;
      }

      return (x / gcd(x, y)) * y;
    }

    void update(int x, int lx, int rx, int st, int dr, int val) {
```

```
    if (st <= lx && rx <= dr) {
      t[x] = combine(t[x], val);
      return;
    }

    int mid = (lx + rx) >> 1;

    if (st <= mid) {
      update(x << 1, lx, mid, st, dr, val);
    }

    if (mid < dr) {
      update(x << 1 | 1, mid + 1, rx, st, dr, val);
    }
  }

  void query(int x, int lx, int rx, int pos) {
    queryRes = combine(queryRes, t[x]);

    if (lx == rx) {
      return;
    }

    int mid = (lx + rx) >> 1;

    if (pos <= mid) {
      query(x << 1, lx, mid, pos);
    } else {
      query(x << 1 | 1, mid + 1, rx, pos);
    }
  }

public:
  SegTree(int N) : n(N) {
    int dim = 1;

    while (dim < n) {
      dim <<= 1;
    }

    t.resize(dim << 1, -1);
  }

  void update(int st, int dr, int val) {
    if (st <= dr) {
      update(1, 1, n, st, dr, val);
    }
  }

  int query(int pos) {
    if (pos == 0) {
      return 0;
    }

    queryRes = -1;

    query(1, 1, n, pos);

    return queryRes;
  }
```

```
};

void computeAncestors(int n) {
  for (int v = 1; v <= n; ++v) {
    anc[0][v] = par[v];
  }

  for (int level = 1; level <= kLog; ++level) {
    for (int v = 1; v <= n; ++v) {
      anc[level][v] = anc[level - 1][anc[level - 1][v]];
    }
  }
}

void dfs1(int u, int level) {
  dep[u] = level;
  sub[u] = 1;
  chainTop[u] = u;
  heavySon[u] = 0;

  for (int v : g[u]) {
    dfs1(v, level + 1);

    sub[u] += sub[v];

    if (sub[heavySon[u]] < sub[v]) {
      heavySon[u] = v;
    }
  }
}

bool isAncestor(int x, int y) {
  if (x == 0) {
    return true;
  }

  return tin[x] <= tin[y] && tout[y] <= tout[x];
}

void dfs2(int u) {
  tin[u] = ++timer;

  if (heavySon[u]) {
    chainTop[heavySon[u]] = chainTop[u];
    dfs2(heavySon[u]);

    for (int v : g[u]) {
      if (v != heavySon[u]) {
        dfs2(v);
      }
    }
  }

  tout[u] = timer;
}

void computeAncestorsGcd(int n) {
  for (int v = 1; v <= n; ++v) {
    gcdTable[0][v] = gcd(nodeValue[v], nodeValue[par[v]]);
  }
```

```
  for (int level = 1; level <= kLog; ++level) {
    for (int v = 1; v <= n; ++v) {
      gcdTable[level][v] = gcd(gcdTable[level - 1][v], gcdTable[level - 1][anc[level -
1][v]]);
    }
  }
}

void updateChain(int u, int v, int val, SegTree& t) {
  while (chainTop[u] != chainTop[v]) {
    if (dep[chainTop[u]] < dep[chainTop[v]]) {
      swap(u, v);
    }

    t.update(tin[chainTop[u]], tin[u], val);
    u = par[chainTop[u]];
  }

  int l = tin[u], r = tin[v];

  if (r < l) {
    swap(l, r);
  }

  t.update(l, r, val);
}

int queryGcd(int x, int y) {
  if (x == y) {
    return nodeValue[x];
  }

  if (dep[y] < dep[x]) {
    swap(x, y);
  }

  int result = 0;

  for (int i = kLog; i >= 0; --i) {
    if (!isAncestor(anc[i][x], y)) {
      result = gcd(result, gcdTable[i][x]);
      x = anc[i][x];
    }
  }

  result = gcd(result, nodeValue[x]);
  x = par[x];

  for (int i = kLog; i >= 0; --i) {
    if (!isAncestor(anc[i][y], x)) {
      result = gcd(result, gcdTable[i][y]);
      y = anc[i][y];
    }
  }

  result = gcd(result, nodeValue[y]);

  return result;
}
```

```
ifstream fin("gcd_tree.in");
ofstream fout("gcd_tree.out");

void testCase(string& solution) {
  int n, m;
  fin >> n >> m;

  for (int v = 1; v <= n; ++v) {
    g[v].clear();
  }

  for (int v = 2; v <= n; ++v) {
    fin >> par[v];
    g[par[v]].emplace_back(v);
  }

  computeAncestors(n);

  timer = 0;
  dfs1(1, 1);
  dfs2(1);

  SegTree t(n);

  for (int i = 1; i <= m; ++i) {
    int x, y, val;
    fin >> x >> y >> val;

    treeRestriction[i] = make_tuple(x, y, val);

    updateChain(x, y, val, t);
  }

  for (int v = 1; v <= n; ++v) {
    nodeValue[v] = t.query(tin[v]);
  }

  computeAncestorsGcd(n);

  char result = '1';

  for (int i = 1; i <= m && result == '1'; ++i) {
    int x, y, val;
    tie(x, y, val) = treeRestriction[i];

    if (queryGcd(x, y) != val) {
      result = '0';
    }
  }

  solution += result;
}

int32_t main() {
  int tests;
  fin >> tests;

  string solution = "";
```

```
  for (int tc = 0; tc < tests; ++tc) {
    testCase(solution);
  }

  fout << solution << '\n';

  return 0;
}
```

Soluție 73p

```
#include <iostream>
#include <fstream>
#include <vector>
#include <utility>
#include <string>

using namespace std;

#define int long long


int gcd(int x, int y) {
  if (x == 0 || y == 0) {
    return x + y;
  }

  while (y) {
    int r = x % y;
    x = y;
    y = r;
  }

  return x;
}

class SegTree {
  private:
    int n;
    vector<int> t;

    int combine(int x, int y) {
      if (x == -1) {
        return y;
      }

      if (y == -1) {
        return x;
      }

      return (x / gcd(x, y)) * y;
    }

    void update(int x, int lx, int rx, int pos, int val) {
      if (lx == rx) {
        t[x] = val;
        return;
```

```
      }

      int mid = (lx + rx) >> 1;

      if (pos <= mid) {
        update(x << 1, lx, mid, pos, val);
      } else {
        update(x << 1 | 1, mid + 1, rx, pos, val);
      }

      t[x] = combine(t[x << 1], t[x << 1 | 1]);
    }

  public:
    SegTree(int N) : n(N) {
      int dim = 1;

      while (dim < n) {
        dim <<= 1;
      }

      t.resize(dim << 1, -1);
    }

    void update(int pos, int val) {
      update(1, 1, n, pos, val);
    }

    int queryAll() {
      return t[1];
    }
};

ifstream fin("gcd_tree.in");
ofstream fout("gcd_tree.out");

void testCase(string& solution) {
  int n, m;
  fin >> n >> m;

  for (int i = 2; i <= n; ++i) {
    int aux;
    fin >> aux;
  }

  vector<pair<int, int>> restriction(m + 1);
  vector<pair<int, int>> intervalRestriction(m + 1);
  vector<vector<int>> start(n + 1);
  vector<vector<int>> en(n + 1);

  for (int i = 1; i <= m; ++i) {
    int l, r, val;
    fin >> l >> r >> val;

    if (r < l) {
      swap(l, r);
    }

    restriction[i] = make_pair(val, -1);
    intervalRestriction[i] = make_pair(l, r);
```

```
    start[l].emplace_back(i);
    en[r].emplace_back(i);
  }

  vector<int> lg2(n + 1);
  for (int i = 2; i <= n; ++i) {
    lg2[i] = lg2[i >> 1] + 1;
  }
  vector<vector<int>> rmq(1 + lg2[n], vector<int>(n + 1));

  int restrictions = 0;
  SegTree st(n);

  for (int i = 1; i <= n; ++i) {
    for (int index : start[i]) {
      restriction[index].second = ++restrictions;
      st.update(restrictions, restriction[index].first);
    }

    rmq[0][i] = st.queryAll();

    for (int index : en[i]) {
      st.update(restriction[index].second, -1);
    }
  }

  for (int l = 1; l <= lg2[n]; ++l) {
    for (int i = 1; i <= n - (1 << l) + 1; ++i) {
      rmq[l][i] = gcd(rmq[l - 1][i], rmq[l - 1][i + (1 << (l - 1))]);
    }
  }

  auto queryGcd = [&](int l, int r) -> int {
    int k = lg2[r - l];
    return gcd(rmq[k][l], rmq[k][r - (1 << k) + 1]);
  };

  char result = '1';

  for (int i = 1; i <= m && result == '1'; ++i) {
    if (queryGcd(intervalRestriction[i].first, intervalRestriction[i].second) !=
restriction[i].first) {
      result = '0';
    }
  }

  solution += result;
}

int32_t main() {
  int tests;
  fin >> tests;

  string solution = "";
  for (int tc = 0; tc < tests; ++tc) {
    testCase(solution);
  }

  fout << solution << '\n';
```

```
  return 0;
}
```