Srpska informatička olimpijada
23.08.2020.

2020sio01usavrsavanje

vreme	memorija	ulaz	izlaz
0,5 s	64 Mb	standardni ulaz	standardni izlaz

Софтверска компанија укупно има n запослених. Сваком запосленом се додељује јединствени број од 1 до n. Свака успешна софтверска компанија има хијерархијску структуру (тј. зна се ко је директор, ко је шеф тима, ко је програмер у тиму,...). Директору једне такве успешне софтверске компаније је додељен број 1. Сваки запослени, осим директора, има тачно једног непосредног шефа. Непосредни шеф запосленог i има број pi (pi < i).

Запослени x је подређени нивоа 1 за запосленог y ако је px = y, тј. y је шеф запосленом x. За k > 1,запослени x је подређени нивоа k за запосленог y, ако је запослени px подређени нивоа k−1 за запосленог y.

Директор компаније ће послати неке запослене на курсеве усавршавања тако што одабере два броја L и R и упути на курсеве све запослене са бројевима i, тако да важи L≤i≤R.

Пре него што је изабрао бројеве L и R, директор је примио m жеља од запослених у компанији. На пример, j-ту жељу описују два броја uj и kj што значи да запослени uj жели да се на усавршавање пошаље један од његових подређених нивоа kj. Да би уштедео новац, директор жели да одабере такве бројеве L и R тако да што мање запослених се пошаље на усавршавање, али и да се испуне све жеље.

Потребно је написати програм који поштује хијерархијску структуру компаније (јер се зна ко је шеф, а ко је подређени запослени) и жеље запослених и одређује такве бројеве L и R да уколико све запослене са бројевима од L до R пошаљете на курсеве за усавршавање, све жеље ће бити испуњене и на усавршавање ће бити послат минимални број запослених. Ако постоји неколико таквих оптималних парова бројева L и R, пронађите онај пар у којем је вредност L најмања.

Улаз

Са стандардног улаза се уноси у првом реду цео број n (2≤n≤200000). Други ред садржи n−1 бројева p2, p3, ..., pn (1≤pi < i) који представљају бројеве непосредних шефова. Трећи ред садржи број m (1≤m≤200000) - број жеља запослених.

Следећих m редова описују жеље запослених и сваки ред садржи два цела броја uj и kj (1≤uj<n,1≤kj<n). Можете претпоставити да запослени uj има најмање једног подређеног датог нивоа kj).

Излаз

Испишите на стандардни излаз два тражена броја: L и R. Ако постоји неколико оптималних парова (L, R), испишите онај пар у којем је вредност L минимална.

Пример 1

Улаз
```
7
1 1 2 2 3 3
3
1 1
3 1
1 2
```
Излаз
```
3 6
```

Пример 2

Улаз
```
10
1 2 2 2 1 5 6 4 7
10
1 4
4 1
4 1
5 2
7 1
4 1
1 2
2 1
7 1
5 2
```
Излаз
```
5 10
```
Појашњење примера 1: Запослени са бројевима 3, 4, 5 и 6 биће послати на усавршавање. Запослени 3 је подређени нивоа 1 за директора (тј. радника број 1), запослени 4 је подређени нивоа 2 за директора (тј. радника број 1), а запослени 6 је подређени нивоа 1 за запосленог са бројем 3.

Опис решења

Хијерархијска структура описана у задатку је коренско стабло чији чворови су нумерисани тако да родитељски чвор је обележен мањим бројем у односу на чвор детета. Постоји

неколико релација описаних уређеним паром (p[i],k[i]). Потребно је наћи најмањи интервал [L, R] тако да за сваки чвор p[i] постоји чвор чије обележје је у нађеном интервалу тако да чвор p[i] је предак нивоа k[i].

Наивно решење кубне сложености:

За сваку жељу (p[i],k[i]) креирамо листу Li од бројева чворова који супотомци чвора p[i] нивоа k[i] . Неопходно је одабрати неколико чворова тако да се из сваког скупа Li одабере најмање један чвор, а разлика између максималног и минималног броја обележја одабраних чворова је што мања. Фиксирамо изабрани чвор са минималним бројем и означавамо га са x. Ако скицирамо пар примера, јасно је да за сваку листу Li је оптимално одабрати чвор с минималним обележјем y[i], тако да је y[i] ≥ x. Након што прођемо кроз све могућности избора чвора x (на пример, из листе L1) пронађемо за сваку од опција максимум чворова y[i] и изаберемо минимум свих опција за избор x, добићемо одговор на проблем. Конструкција листа Li је квадратне сложености, а са обиласком чворова и тражење мин решења, добијамо кубну сложеност.

Поправка ефикасности:

Сортирајмо чворове сваке листе према нумеричком обележју чвора и уочимо да како вредност за x расте, вредност за y[i] се не смањује. Дакле, у свакој листи постоји показивач на y[i] и померањем показивача само у једном смеру у листи могу да се у квадратној сложености обраде све варијанте за избор x. Због сортирања, укупна сложеност је $O(n^2 \log n)$.

Још ефикасније: Обилазак стабла у дубину и формирање листи Lh на дубини h.

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

using namespace std;

typedef pair<int, int> segment;

int const MAX_N = 200000;

int n;
vector<int> tree[MAX_N];
vector<int> verticesByDepth[MAX_N];
vector<segment> levelSegments[MAX_N];

int dfsTime = 0;
int timeIn[MAX_N], timeOut[MAX_N], depth[MAX_N];
```

```
void dfs(int u, int d) {
	timeIn[u] = dfsTime++;
	depth[u] = d;
	verticesByDepth[d].push_back(u);
	for (int t = 0; t < (int) tree[u].size(); t++) {
		dfs(tree[u][t], d + 1);
	}
	timeOut[u] = dfsTime;
}

void initializeTree() {
	for (int i = 1; i < n; i++) {
		int parent;
		cin >>parent;
		tree[parent - 1].push_back(i);
	}
	dfs(0, 0);
}

void removeIncluded() {
	for (int level = 0; level < n; level++) {
		std::sort(levelSegments[level].begin(), levelSegments[level].end(),
			[](const segment &o1, const segment &o2) {
			if (o1.first == o2.first) {
				return o1.second > o2.second;
			} else {
				return o1.first < o2.first;
			}
			}
		);
		vector<segment> result;
		for (segment s : levelSegments[level]) {
			while (result.size() > 0 && result.back().second >= s.second) {
				result.pop_back();
			}
			result.push_back(s);
		}
		levelSegments[level] = result;
	}
}

segment findSegment(int depth, int tIn, int tOut) {
	int from, to;
```

```
	{
		int left = -1, right = (int) verticesByDepth[depth].size();
		while (left < right - 1) {
			int mid = (left + right) / 2;
			if (timeIn[verticesByDepth[depth][mid]] >= tIn) {
				right = mid;
			} else {
				left = mid;
			}
		}
		from = right;
	}
	{
		int left = -1, right = (int) verticesByDepth[depth].size();
		while (left < right - 1) {
			int mid = (left + right) / 2;
			if (timeOut[verticesByDepth[depth][mid]] <= tOut) {
				left = mid;
			} else {
				right = mid;
			}
		}
		to = left;
	}
	return segment(from, to);
}

void findBestSegment() {
	vector<int> mark(n, -1);
	int marks = 0;
	for (int i = 0; i < n; i++) {
		for (segment s : levelSegments[i]) {
			for (int j = s.first; j <= s.second; j++) {
				mark[verticesByDepth[i][j]] = marks;
			}
			marks++;
		}
	}
	vector<int> markCount(marks);
	int curmarks = 0;
	int r = 0;
	int bestL = 0, bestR = n;
	for (int l = 0; l < n; l++) {
		while (r < n && curmarks < marks) {
```

```
                if (mark[r] != -1) {
                        if (markCount[mark[r]] == 0) {
                                curmarks++;
                        }
                        markCount[mark[r]]++;
                }
                r++;
        }
        if (curmarks == marks) {
                if (r - l < bestR - bestL) {
                        bestL = l;
                        bestR = r;
                }
        }
        if (mark[l] != -1) {
                markCount[mark[l]]--;
                if (markCount[mark[l]] == 0) {
                        curmarks--;
                }
        }
    }
    cout <<bestL + 1<<" "<<bestR;
}

int main() {

    ios::sync_with_stdio(false);
    cin.tie(nullptr);

    cin >> n;
    initializeTree();

    int m;
    cin >>m;

    for (int i = 0; i < m; i++) {
        int u, k;
        cin >>u>>k;
        u--;
        levelSegments[depth[u] + k].push_back(findSegment(depth[u] + k, timeIn[u],
timeOut[u]));
    }
    removeIncluded();
    findBestSegment();
```

```
    return 0;
}
```

Борба

Vremensko ograničenje	Memorijsko ograničenje	ulaz	izlaz
0,5 s	32 MB	standardni ulaz	standardni izlaz

Приближава се одлучујућа битка између робота и људи. Људски шпијуни су сазнали да су роботи остали са n бораца. Показало се да су и роботи и људи имали тачно по n бораца. Према искуству претходних битака, људи знају да i-ти човек може бити поражен само од робота који има број i.

Командант је одлучио да построји људе. Након сазнања о тајним плановима робота, командант је објаснио да ће се човек на i-том положају у борбеној линији борити против робота са бројем a_i. Тим људи ће победити само ако сваки од бораца победи у својој бици.

У почетку је командант поставио i-тог човека у i-ти положај у линији. Али тада је схватио да је остало мало времена до битке и људи могу изгубити ако их оставе у текућој формацији. У једној секунди, командант може померити особу са последњег места на почетку линије. Након ове операције, пребачени ратник се појављује на првом месту у линији, а број сваког другог борца се повећава за 1.

Напишите програм у ком ћете наћи минимално време за које командант може преуредити борбену линију тако да људи победе у одлучујућој бици.

Улаз

Са стандардног улаза се уноси у првом реду цео број n (1≤n≤200000), број бораца. Други ред садржи n целих различитих бројева a1, a2, ..., an где ai (1≤ai≤n) представља број робота који ће се борити против човека на i-тој позицији у борбеној линији.

Излаз

Испишите на стандардни излаз тражени број - минимални број секунди за које командант може преуредити линију тако да људи победе. Ако је немогуће победити роботе, испишите -1.

Пример 1

Улаз

```
5
1 4 2 3 5
```

Излаз

```
2
```

Пример 2

Улаз

```
5
1 3 5 2 4
```

Излаз

-1

Појашњење примера 1: У првом примеру роботи стоје наспрам људи на следећи начин:

Роботи 1 4 2 3 5

Људи 1 2 3 4 5

Људи ће изгубити битку, јер ће роботи, са бројевима 1 и 5, појединачно победити у својим биткама. Али, након првог преуређења од стране команданта, састав бораца изгледа овако:

Роботи 1 4 2 3 5

Људи 5 1 2 3 4

Али, сада роботи 2 и 3 побеђују у својим биткама, те је потребна још једна размена од стране комаданта. Након тога, борбена линија је таква да сви људи добијају своје борбе.

Роботи 1 4 2 3 5

Људи 4 5 1 2 3

Улазни податак је низ бројева a_1, a_2, ..., a_n који представља пермутацију бројева 1,2,3,...,n.

Потребно је написати програм који одређује минимални број позиција на које пермутација мора бити циклично померена улево, тако да не постоји позиција ***i*** у којој је број једнак вредности ***i***.

Након ***n*** цикличких померања пермутација се враћа у првобитни облик. Дакле, ако постоји одговор, он има вредности у интервалу од 0 до *n-1*.

Такође за сваку позицију ***i*** постоји тачно једно циклично померање пермутације a_1, a_2, ..., a_n на ***k*** позиција $(0 \le k \le n-1)$, при којој ће на овој позицији бити број ***i***. Назовимо ово померање губитничко (јер човек тада губи битку од робота). Вредност позиције је тада једнака остатку дељења броја $i-a_i$ са n.

За позиције проверавамо које цикличко померање је губитничко. За свако циклично померање на ***k*** позиција $(0 \le k \le n-1)$ гледамо да ли је губитничко за било коју позицију. Извршимо циклично померање на k позиција $(0 \le k \le n-1)$. Ако није губитничко, онда је нађено решење. Ако су сва померања губитничка, одговор је -1.

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

#define Nlen 200001
int n;
```

```
int p[Nlen],  k[Nlen];

int main()
 {
   ios::sync_with_stdio(false);
          cin.tie(nullptr);
   cin >> n;
          for (int i=0; i< n; i++)
   {
                    cin >> p[i];
                    p[i]--;
                    k[(i - p[i] + n) % n] = 1;
          }
          for (int i=0; i< n; i++)
    {
                    if (k[i] == 0)
       {
                              cout << i << "\n";
                return 0;
                    }
          }
          cout << -1 << "\n";
          return 0;
}
```

Хачапури

Vremensko ograničenje	Memorijsko ograničenje	ulaz	izlaz
0,2 s	32 MB	standardni ulaz	standardni izlaz

Гога Битадзе је познати грузијски кошаркаш који осим кошарке воли хачапури бројеве, тј. бројеве који немају две једнаке суседне цифре. За дати цели број n , Гога жели да нађе најмањи цели хачапури број већи од n . Напишите програм који ће помоћи Гоги.

Улаз

Са стандардног улаза се уноси у првом реду цео број n ($1 \le n \le 10^{18}$).

Излаз

Испишите на стандардни излаз тражени хачапури број (најмањи цео број већи од n у ком нису једнаке две суседне цифре).

Пример 1

Улаз

1

Излаз

2

Пример 2

Улаз

98
Излаз
101

Опис решења:

У односу на дато ***n***, морамо пронаћи најмањи цео број који је већи од ***n***, а да му две узастопне цифре нису једнаке.

Проналазимо највећу цифру у којој се разликују решење и полазни број:

a1 a2 … ai … an (број дат на улазу)

= = = ≠

b1 b2 … bi … bn (решење)

o Са леве стране не постоје две суседне једнаке цифре;

o Можемо повећати a_i за 1 (али ако је $a_i + 1 = a_{i-1}$, тада ћемо повећати за 2);

o Из таквих разреда цифара бирамо најдешњи (позиције мање тежине);

o Повећавамо број у њему за 1 или 2, а резултирајући број не сме бити већи од 9:

a1 a2… ai… (задат број)

a1 a2… $a_i + {1,2} $ 0 1 0 ... (коначан број)

o Остале цифре попуните са 0 и 1 редом, почевши од 0.

Нина Грузијска

Vremensko ograničenje	Memorijsko ograničenje	ulaz	izlaz
1 s	256 MB	standardni ulaz	standardni izlaz

Нина је вредна девојчица из Грузије која је прикупила већу своту новаца да оствари свој сан, односно да оде на пут око света. Трошкове ће плаћати платном картицом. На жалост, банка је наметнула строго ограничење колико новца Нина може користити за један дан. Зато мора добро планирати путовање.

Нина ће на путу око света обићи N одредишта (градова) који су нумерисни бројевима од 1 до N. Нина је открила да постоји M авио линија које повезују неке од одредишта. Да би пут учинила што интересантнијим, планира сваког дана да посети нови град. Знајући цену

летова на свакој од авио линија, сада се пита колико најмање мора да износи дневно ограничење наметнуто од стране банке да би могла да путују између одређеног почетног и крајњег одредишта. Другим речима, Нина је заинтересована за најмањи могућ износ максималне цене лета којим ће путовати током пута око света. Напишите програм који одговара на Q оваквих питања. Свака два града су доступна кроз један или више авио лета.

Улаз

Први ред стандардног уноса садржи два природна броја - број дестинација N и број авио линија M. У наредних M линија дата су три броја u, v и w (u≠v) који описују да између градова u и v постоји двосмерна авионска веза, а трошак превоза је w. У следећем реду је дат природни број Q - број питања. Потом следи Q линија са два различита броја s и f, која описују текући упит за полазни град s и коначно одредиште f.

Излаз

У првом реду стандардног излаза, програм треба да испише збир свих одговора на свих Q питања.

Ограничења:

$2 \leq N, M \leq 10^5$

$1 \leq Q \leq 10^6$

$1 \leq u, v, s, f \leq N$

$1 \leq w \leq 10^9$

Пример

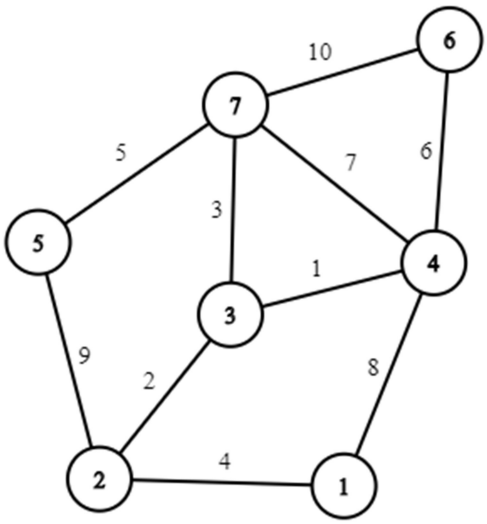
Улаз

```
7 10
1 2 4
1 4 8
2 3 2
2 5 9
3 4 1
3 7 3
4 6 6
4 7 7
5 7 5
6 7 10
3
1 5
2 4
```

3 6

Излаз

13



Појашњење примера:

У овом примеру имамо 7 дестинација везаних за 10 авио линија. Број питања је 3.

У првом питању s = 1 и f = 5. Приметили смо да за достизање одредишта 5, морамо проћи једну од линија 7-> 5 или 2-> 5. Изабрали смо 7-> 5 јер је јефтиније.

За следећа два питања, лако је видети да су оптимални правци 2-> 3-> 4 и 3-> 4-> 6.

Да би успешно прошли, гранична вредност банке мора бити 2 и 6.

Коначни одговор 13 је збир одговора на сва три питања: 13=5+2+6

Опис решења

Решење задатка је комбинација неколико стандардних техника и алгоритама за рад са графовима. Иако је основна идеја једноставно уочљива, ефикасно решење није тако очигледно.

Задатак се може формулисати и на следећи начин: Дат је повезан граф G=(V, E) који има N чворова и M грана. За сваки пут од чвора s до чвора f проналазимо грану са највећом вредности и тражимо минимум вредности ових грана.

Једно могуће решење за свако је испитивање свих путева између задатих чворова. То се може постићи обиласком графа у дубину (ДФС), али у експоненцијалној временској сложености. Ово решење може да буде прихватљиво за графове са мање чворова или за веће графове који имају структуру стабла (када постоји тачно један пут између свака два чвора).
Уочимо и да упит за чворове графа s,t можемо да обрађујемо као упит над чворовима s,t покривајућег стабла графа минималне цене. Дакле, једно могуће решење је да се примени алгоритам за налажење покривајућег стабла графа минималне цене и да за сваки упит размотримо путању између датих чворова. Ако применимо алгоритам Крускала и пронађемо одговоре на сваки упит имаћемо решење чија временска сложеност је . Укупна сложеност овог решења је $O(M * \log_2 M + N * (M + Q))$.

Још ефикасније решење се добија разматрањем односа између најближег заједничког претка на стаблу (LCA) и минимума у упиту распона (RMQ).

```
/// MINIMUM COST SPANNING TREE, LCA -> RMQ, DSU, SPARSE TABLE
#include <iostream>
#include <algorithm>
#include <vector>
#include <utility>
#define fi first
#define se second
using namespace std;

const int MAXN = 100005;
const int SIZE = 20;

struct edge
{
   int u, v, w;
};
```

```
int n, m, q;
edge e[MAXN];
int bin[MAXN << 2];

vector<int> g[MAXN << 1];
int p[MAXN << 1], r[MAXN << 1];

int t[MAXN << 1][2];
vector<pair<int, int>> v[SIZE];

bool cmp(edge x, edge y)
{
   return x.w < y.w;
}

void toZero()
{
   for (int i = 1; i <= n; ++ i)
   {
      p[i] = i;
      r[i] = 1;
   }
}

void Init()
{
   ios :: sync_with_stdio(false);
   cin.tie(NULL); cout.tie(NULL);

   cin >> n >> m;
   for (int i = 1; i <= m; ++ i)
   {
      cin >> e[i].u >> e[i].v >> e[i].w;
   }
```

```
    sort(e + 1, e + m + 1, cmp);

    bin[1] = 0;
    for (int i = 2; i <= (n << 2); ++ i)
    {
        bin[i] = bin[i - 1];
        if ((i & (-i)) == i) bin[i]++;
    }

    toZero();
}

int root(int x)
{
    if (p[x] == x) return x;
    else return p[x] = root(p[x]);
}

void join(int x, int y, int z = 0)
{
    x = root(x);
    y = root(y);

    if (z != 0)
    {
        g[z].push_back(x);
        g[z].push_back(y);
        p[x] = p[y] = p[z] = z;
        return;
    }

    if (r[x] <= r[y])
    {
        p[y] = x;
        r[x] += r[y];
    }
```

```
  else
  {
    p[x] = y;
    r[y] += r[x];
  }
}

void lca2rmq(int u = n, int l = 1)
{
  t[u][0] = v[0].size();
  v[0].emplace_back(u, l);

  for (int adj : g[u])
  {
    lca2rmq(adj, l + 1);
    v[0].emplace_back(u, l);
  }
}

void precomp()
{
  for (int i = 1; i < SIZE; ++ i)
  {
    int l = (int)(v[0].size()) - (1 << i);
    if (l >= 0) v[i].reserve(l + 1);
    for (int j = 0; j <= l; ++ j)
    {
      if (v[i - 1][j].se <= v[i - 1][j + (1 << (i - 1))].se)
      {
        v[i].emplace_back(v[i - 1][j].fi, v[i - 1][j].se);
      }
      else
      {
        v[i].emplace_back(v[i - 1][j + (1 << (i - 1))].fi, v[i - 1][j + (1 << (i - 1))].se);
      }
    }
```

```
    }
}

int query(int s, int f)
{
    s = t[s][0];
    f = t[f][0];

    if (s > f) swap(s, f);

    int i = bin[f - s + 1], j;
    if (v[i][s].se <= v[i][f - (1 << i) + 1].se)
    {
        j = v[i][s].fi;
    }
    else
    {
        j = v[i][f - (1 << i) + 1].fi;
    }

    return t[j][1];
}

int main()
{
    Init();

    for (int i = 1; i <= m; ++ i)
    {
        if (root(e[i].u) != root(e[i].v))
        {
            join(e[i].u, e[i].v);
        }
        else
        {
            e[i].w = -1;
```

```
    }
  }

  toZero();

  for (int i = 1; i <= m; ++ i)
  {
    if (e[i].w != -1)
    {
      join(e[i].u, e[i].v, ++n);
      t[n][1] = e[i].w;
    }
  }

  v[0].reserve(n << 2);
  lca2rmq();
  precomp();

  long long ans = 0;

  cin >> q;
  for (int i = 1; i <= q; ++ i)
  {
    int s, f;
    cin >> s >> f;
    ans = ans + query(s, f);
  }

  cout << ans << endl;
  return 0;
}
```