Hash, topSort (stack, queue), deque, map

Hash

1. За дати низ од ***n*** елемената, где је ***n*** паран број и n <= 10^6, испитати да ли он може бити подељен на n/2 парова, тако да је збир елемената сваког пара тог низа дељив природним бројем k, k <= 10^9. Очекивана сложеност решења је O(n). У првом реду се учитава број ***n***, у другом реду елементи низа раздвојени размаком, а у последњем реду се учитава број ***k***. Сви елементи низа су природни бројеви чија је вредност је унутар сегмента [2 … 10^9].

Пример улаза:

4

9 7 5 3

6

Пример излаза:

`Moze biti podeljen.`

Решење:
Структура података која је основ алгоритма линеарне сложености је хеш табела.
Идеја је да се у хеш табели чувају фреквенције сваке вредности остатка при дељењу са ***k*** у низу. У првој итерацији се одређују све вредности у хеш табели.

У другој итерацији се, пролазећи кроз све елементе низа, извршавају следећи кораци:

*Наћи остатак при дељењу тренутног елемента низа са k.

* Ако је остатак једнак 0 или k/2, онда број појављивања мора бити паран број.

* Иначе, фревенција тог остатка мора бити једнака фреквенцији разлике k и тренутног остатка.

```
#include <iostream>
#include <map>
#include <vector>

bool canPairs(std::vector<int> arr, int n, int k)
{
   if (n & 1)
      return false;
   std::map<int, int> freq;
   for (int i = 0; i < n; i++)
```

```
        freq[arr[i] % k]++;

    for (int i = 0; i < n; i++)
    {
        int rem = arr[i] % k;
        if  (2*rem == k)
        {
            if (freq[rem] % 2 != 0)
                return false;
        }
        else if (rem == 0)
        {
            if (freq[rem] & 1)
                return false;
        }
        else if (freq[rem] != freq[k - rem])
            return false;
    }
    return true;
}

int main()
{
    int n, k;
    std::cin >> n;
    std::vector<int>arr(n);
    for (int i = 0; i < n; i++)
        std::cin >> arr[i];
    std::cin >> k;
    canPairs(arr, n, k)
    ? std::cout << "Niz moze biti podeljen."
            : std::cout << "Niz ne moze biti podeljen.";
    return 0;
}
```

2. Napisati program koji čita jednu reč teksta koju čine samo velika slova i ispisuje slovo koji se najčešće pojavljuje i koliko puta se pojavljuje. Ako se više slova najčešće pojavljuje, ispisuje se slovo koji se pre pojavilo u reči.

Ulaz

U jednoj liniji standardnog ulaza nalazi se jedna reč teksta sa ne više od 20 slova.

Izlaz

U prvoj liniji standarnog izlaza prikazati slovo koje se najčešće pojavljuje, a u drugoj liniji standardnog ulaza ispisati i koliko puta se pojavljuje.

Primer 1

Ulaz

```
POPOKATEPETL
```

Izlaz

```
P
3
```

Primer 2

Ulaz

```
BACACB
```

Izlaz

```
B
2
```

IDEJA

Ključni deo zadataka je da se za svako veliko slovo engleskog alfabeta izbroji koliko se puta pojavljuje u datoj reči. Potrebno je, dakle, uvesti 26 različitih brojača - po jedan za svako veliko slovo engleskog alfabeta. Jasno je da uvođenje 26 različitih promenljivih ne dolazi u obzir.
Potrebna je struktura podataka koja preslikava dati karakter u broj njegovih pojavljivanja. Ovakve strukture podataka nazivaju se asocijativni nizovi, konačna preslikavanja, mape ili rečnici i one preslikavaju date ključeve (u našem slučaju ključevi su karakteri) u njima pridružene vrednosti (u našem slučaju to su brojevi pojavljivanja).

Najbolja takva struktura u ovom zadatku je niz brojača u kojem se na poziciji nula čuva broj pojavljivanja slova `A`, na poziciji jedan slova `B` itd., sve do pozicije 25 na kojoj se nalazi broj pojavljivanja slova `Z`. Centralno pitanje je kako na osnovu karaktera odrediti poziciju njegovog brojača u nizu. Za to se koristi činjenica da su karakterima pridruženi numerički kodovi (ASCII u jeziku C++ tj. Unicode u jeziku Java, Python,...) i to na osnovu redosleda karaktera u engleskom alfabetu. Redni broj karaktera je zato moguće odrediti oduzimanjem koda karaktera `A` od koda tog karaktera (na primer, kod karaktera `D` je 68, dok je kod karaktera `A` 65, oduzimanjem se dobija 3, što znači da se brojač karaktera `D` nalazi na mestu broj 3 u nizu).
Sličnu tehniku baratanja sa kodovima karaktera sreli smo u početnim kursevima programiranja.

Preslikavanje karaktera u njihov broj pojavljivanja moguće je ostvariti i bibliotečkim strukturama podataka koje predstavljaju tzv. mape tj. rečnike. U jeziku c++ bismo mogli upotrebiti `map` ili `unordered_map`.
Ipak, ove strukture podataka su primerenije za situacije u kojima nije tako jednostavno na osnovu ključa dobiti njihovu numeričku vrednost i kada je skup dopustivih ključeva širi.

Napomenimo i to da je za predstavljanje reči moguće upotrebiti ili tip `string` ili niz karaktera terminisan nulom (što je više u duhu jezika C).

```
#include <iostream>
#include <string>
#include <algorithm>
#include <map>
using namespace std;

int main() {
  // rec koja se analizira
  string rec;
  cin >> rec;

  // broj pojavljivanja svakog slova od A do Z
  map<char, int> brojPojavljivanja;
  // uvecavamo broj pojavljivanja svakog slova iz reci
  for (int i = 0; i < rec.size(); i++)
    brojPojavljivanja[rec[i]]++;

  // odredjujemo najveci broj pojavljivanja slova
  int maxPojavljivanja = 0;
  for (auto it = brojPojavljivanja.begin(); it != brojPojavljivanja.end();
it++)
    if (it->second > maxPojavljivanja)
      maxPojavljivanja = it->second;

  // ispisujemo slovo koje se prvo pojavljuje u reci sa tim brojem
  // pojavljivanja
  for (int i = 0; i < rec.size(); i++)
    if (brojPojavljivanja[rec[i]] == maxPojavljivanja) {
      cout << rec[i] << endl
           << maxPojavljivanja << endl;
      break;
    }

  return 0;
}
```

3. Nalazimo se na času engleskog u 9. razredu kod profesora D. Učenica Deni slabo zna engleski jezik i bavi se dosadnim aktivnostima. U želji da se zabavi, Deni analizira tekst napisan na tabli. Dok analizira tekst, ona ignoriše razmake između reči tako da ceo tekst je zapravo

jedna velika reč dužine ***N*** koju čine mala i velika slova engleske abecede. Dakle, ovu veliku reč čine samo slova i ne sadrži beline. Neka je broj različitih slova u velikoj reči jednak K. Deni počinje da razmatra različite podstringove od ove sekvence i ona zapisuje broj pojave svakog karaktera. Kada su ovi brojevi pojavljivanja svakog od tih K slova jednak, onda se takav podstring zove *magičan*. Tokom ovih časova engleskog, Deni može da proveri svaki podstring velike reči. U međuvremenu, ona računa koliko podstringova je magično i na kraju, ona je veoma radosna zbog rešenog zadatka analize teksta. Deni odluči da bi time mogla da se bavi tokom narednih časova engleskog. ALI, na predstojećim časovima, tekst na tabli koji piše prof. D će biti sve duži i duži. Tako da ona moli za Vašu pomoć – morate da napišete program koji će izračunati broj magičnih podstringova u datoj velikoj reči dužine ***N*** koju čine samo slova engleske abecede.

Zadatak

Napišite program koji izračunava broj magičnih podstringova u datoj sekvenci od ***N*** engleskih slova.

Ulaz

U prvoj liniji standardnog ulaza, Vaš program mora da čita jedan ceo broj ***N*** – broj slova u sekvenci koju je napisao prof. D. U sledećoj liniji, program učitava string, veliku reč od ***N*** slova engleske abecede. Slova mogu biti mala i velika!!! Vodite računa da, sun a primer, veliko slovo *A* i malo slovo *a* su različiti znaci.

Izlaz

Program mora da štampa na standardni izlaz broj magičnih podstringova u datoj velikoj reči slova. Pošto ovaj broj može da bude veoma velik, potrebno je da štampate ostatak pri deljenju tog broja sa 1 000 000 007.

Ograničenja

- $2 \leq N \leq 100\,000$

Podzadaci

Podzadatak	Poeni	N	Dodatna ograničenja
1	10	≤ 100	Ne postoje.
2	20	≤ 2000	Ne postoje.
3	30	≤ 100 000	Postoji samo dva tipa slova u datom stringu (K=2).
4	40	≤ 100 000	Ne postoje.

Primeri

Primer ulaza	*Primer izlaza*	*Objašnjenje*

8 abccbabc	4	Magični podstringovi su abc, cba, abc, abccba. To su podstringovi koji su istog sastava, ali sa početnim I završnim slovima izabranim na različitim pozicijama, te se računaju kao različiti podstringovi. Uočite da, na primer, podstring ab nije magičan jer ne sadrži slovo *c*. Ni string acb nije magičan zato što nije sačinjen od uzastopnih slova date velike reči, te ne može biti podstring.
7 abcABCC	1	Samo podstring abcABC je magičan (malo slovo a i veliko slovo A su različiti).
20 SwSSSwwwwSwSwwSwwwwS	22	Broj magičnih podstringova je 22 i jedan od njih je SwSwwS.

Ideja - hash

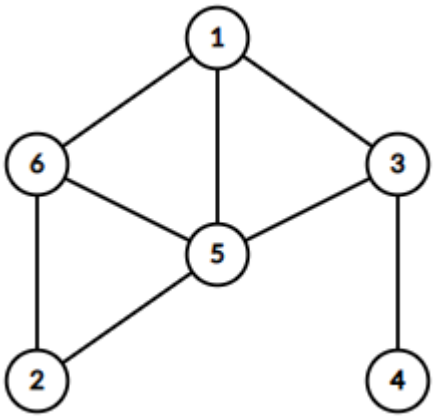
Решење 01

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

using namespace std;

const int NMAX = 100000 + 5;
const int MOD = 1000000000 + 7;

string str;
int v[NMAX];

map <vector <int>, int> Map;

vector <int> cr;

int main() {
   ios_base :: sync_with_stdio(false);
   int N;
   cin >> N;
   cin >> str;
   string _str = str;

   sort(_str.begin(), _str.end());
   _str.resize(unique(_str.begin(), _str.end()) - _str.begin());

   for (int i = 0; i < N; ++ i)
      v[i] = lower_bound(_str.begin(), _str.end(), str[i]) - _str.begin();
```

```
    cr.resize(_str.size(), 0);

    Map[cr] = 1;
    long long int ans = 0;
    for (int i = 0; i < N; ++ i) {
        if (!v[i]) {
            for (int j = 1; j < _str.size(); ++ j)
                -- cr[j];
        }
        else
            ++ cr[v[i]];

        ans += Map[cr];
        ++ Map[cr];
    }

    cout << ans % MOD << '\n';
    return 0;
}
```

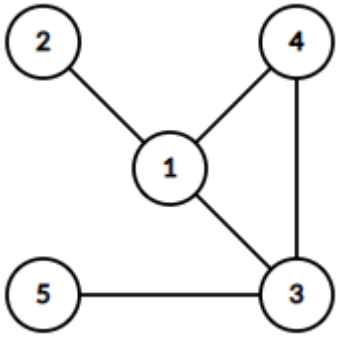
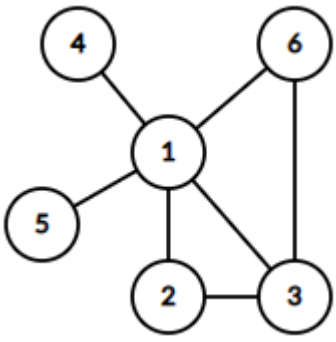
3. Dat je graf listom poveznaosti. Konstruisati algoritam linearne slozenosti koji ce ispisati grane grafa. Pogledajmo graf koji ima 6 cvorova i 8 grana:

```
6 8
1 3
1 5
1 6
2 5
2 6
3 4
3 5
5 6
```



Neka je graf dat opisom susesdstva za svaki cvor:

- Cvor 1: [3,5,6]
- Cvor 2: [5,6]
- Cvor 3: [1,4,5]
- Cvor 4: [3]
- Cvor 5: [1,2,3,6]
- Cvor 6: [1,2,5]

Ulaz	Izlaz	Slika
5 3 2 3 4 1 1 3 4 1 5 2 3 1 1 3	3 4 1 2 1 3 3 5 1 4	
3 2 2 3 2 3 1 2 1 2	-1	
6 5 3 4 5 6 2 2 3 1 3 1 2 6 1 1 1 1 2 3 1	1 3 2 3 3 6 1 4 1 5 1 6 1 2	

Resenje

```
#include <iostream>
#include <vector>
#include <unordered_map>
#include <algorithm>
#include <functional>
using namespace std;

constexpr int kMaxNodes = 1e5, kNotDag = -1;

pair<int, int> edge_of_node[kMaxNodes];

vector<vector<int>> ReadDiGraph() {
  cin.tie(0);
  ios_base::sync_with_stdio(false);

  int n; cin >> n;

  vector<vector<int>> graph;
  unordered_map<int64_t, int> edge_idx;
  int num_nodes = 0;
  for (int i = 0; i < n; i += 1) {
    int m; cin >> m;
    int prev_node = -1;
    for (int j = 0; j < m; j += 1) {
      int node; cin >> node; node -= 1;

      int mn = min(node, i);
      int64_t code = ((int64_t)mn << 32) + (mn ^ node ^ i);
      if (edge_idx.find(code) == edge_idx.end()) {
        edge_of_node[num_nodes] = make_pair(node, i);
        edge_idx[code] = num_nodes++;
        graph.push_back(vector<int>());
      }
      if (prev_node != -1) {
        graph[prev_node].push_back(edge_idx[code]);
      }
      prev_node = edge_idx[code];
    }
  }
  return graph;
}

enum Colors {
  WHITE = 0,
  GREY = 1,
  BLACK = 2
};

int main() {
```

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

  auto&& G = ReadDiGraph();
  auto TopologicalSort = [&]() {
    const int n = (int)G.size();
    vector<int> topo_stk;
    vector<Colors> color(n, WHITE);

    function<int(const int)> Df = [&](const int node) {
      color[node] = GREY;
      for (auto&& neigh : G[node]) {
        if (color[neigh] == WHITE) {
          if (Df(neigh) == kNotDag) {
            return kNotDag;
          }
        } else if (color[neigh] == GREY) {
          return kNotDag;
        }
      }

      color[node] = BLACK;
      topo_stk.push_back(node);
      return 0;
    };

    for (int node = 0; node < n; node += 1) {
      if (color[node] == WHITE) {
        if (Df(node) == kNotDag) {
          return vector<int>(1, -1);
        }
      }
    }
    return topo_stk;
  };

  vector<int> top_sort = TopologicalSort();
  if (top_sort.front() == kNotDag) {
    cout << "-1\n";
    return 0;
  }

  reverse(top_sort.begin(), top_sort.end());
  for (auto&& node : top_sort) {
    cout << 1 + edge_of_node[node].first << ' ' << 1 + edge_of_node[node].second << '\n';
  }
}
```

RESENJE 2 bez izgradnje grafa

```
#include <algorithm>
```

```
#include <iostream>
#include <vector>
#include <map>
using namespace std;

const int kMaxN = 1e5+5;

vector<pair<int, int>> edges;
map<pair<int, int>, int> edge_id;

int GetEdgeId(int a, int b) {
  if (edge_id.count({a, b})) {
    return edge_id[{a, b}];
  }

  int id = edges.size();
  edges.push_back({a, b});
  edge_id[{a, b}] = id;
  edge_id[{b, a}] = id;
  return id;
}

vector<int> vertex[kMaxN];
int num_front[kMaxN];

int main() {
  int n; cin >> n;
  for (int i = 1; i <= n; i += 1) {
    int num; cin >> num;
    while (num--) {
      int x; cin >> x;
      int id = GetEdgeId(i, x);
      vertex[i].push_back(id);
    }
  }

  vector<int> wait_list;
  auto Add = [&](int id) {
    num_front[id] += 1;
    if (num_front[id] == 2) {
      wait_list.push_back(id);
    }
  };

  auto Pop = [&](int id) {
    int a = edges[id].first;
    int b = edges[id].second;
    for (auto c : {a, b}) {
      vertex[c].pop_back();
      if (vertex[c].size()) {
        Add(vertex[c].back());
```

```
        }
      }
    };

    for (int i = 1; i <= n; i += 1) {
      reverse(vertex[i].begin(), vertex[i].end());
      if (vertex[i].size()) {
        Add(vertex[i].back());
      }
    }

    vector<pair<int, int>> answer;
    while (wait_list.size()) {
      auto id = wait_list.back();
      wait_list.pop_back();
      answer.push_back(edges[id]);
      Pop(id);
    }

    bool ok = true;
    for (int i = 1; i <= n; i += 1) {
      ok &= (vertex[i].size() == 0);
    }

    if (ok) {
      for (auto itr : answer) {
        cout << itr.first << ' ' << itr.second << '\n';
      }
    } else {
      cout << "-1\n";
    }

    return 0;
}
```

RESENJE 3 topSort

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

using namespace std;



int n;

vector<int> e[100005];

vector<pair<int, int>> sol;




int main() {
```

```
ios_base::sync_with_stdio(false);
cin.tie(nullptr);
cout.tie(nullptr);
cerr.tie(nullptr);

cin >> n;
for (int i=1; i<=n; i++) {
        int k;
        cin >> k;
        e[i].resize(k);
        for (int j=k-1; j>=0; j--) {
           cin >> e[i][j];
        }
}

queue<int> q;

for (int i=1; i<=n; i++) {
        q.push(i);
}

while (!q.empty()) {
        int x = q.front(); q.pop();
        if (e[x].size() == 0) {
                continue;
        }
        int y = e[x].back();
        if (e[y].size() == 0) {
                continue;
        }
        if (e[y].back() != x) {
```

```
                continue;
            }

            sol.push_back({x, y});
            e[x].pop_back();
            e[y].pop_back();

            q.push(x);
            q.push(y);
        }

        for (int i=1; i<=n; i++) {
            if (e[i].size()) {
                cout << -1;
                return 0;
            }
        }

        for (auto p : sol) {
            cout << p.first << ' ' << p.second << '\n';
        }


}
```