VI Simulado Ingressantes
Editorial

MaratonUSP

A - Kobus odeia sorteios

Antes de apresentar a solução, quero comentar algumas coisas.

A primeira delas, é que é possível, em C++ e em outras linguagens, comparar duas strings de maneira nativa, o que facilita bastante a implementação.

Outro detalhe, é que a nova permutação dada como resposta, pode ser usada para mapear palavras para um novo alfabeto. Por exemplo, o alfabeto *okbusacdefghijl..z* mapeia *kobus* para *bacde* e *kkbus* para *bbcde*. Graficamente, temos que:

o k	*b u*	*s a*	*c d*	*e f*	*g h*	*i j*	*l m*	*n p*	*q r*	*t v*	*w x*	*y z*
↓ ↓	↓ ↓	↓ ↓	↓ ↓	↓ ↓	↓ ↓	↓ ↓	↓ ↓	↓ ↓	↓ ↓	↓ ↓	↓ ↓	↓ ↓
a b	*c d*	*e f*	*g h*	*i j*	*k l*	*m n*	*o p*	*q r*	*s t*	*u v*	*w x*	*y z*

e então podemos comparar as palavras mapeadas como se elas tivessem no *alfabeto normal*.

Por ultimo, temos uma função chamada *next_permutation* em C++, que, quando utilizada em conjunto com um loop *do...while* pode gerar todas as permutações de uma sequencia de números, aqui esta a referencia deste metodo.

A primeira ideia que pode parecer solucionar esse problema é fazer com que o novo alfabeto seja *kobusacdefghijlmnpqrtvwxyz*. Com isso, você vai transformar *kobus* em *abcde*, e ela só não vai ficar em primeiro lugar na lista se existirem nomes como *kob*, *kobu* (prefixos de *kobus*) ou *kkoo*. Porém, essa ideia não funciona, basta olhar o caso em que um dos nomes da lista é *kkbus*. Neste exemplo, nosso programa diria que não existe uma solução, mesmo com uma das soluções possíveis sendo *okbusacdefghijlmnpqrtvwxyz*.

Agora, finalmente, vamos para a solução. A ideia é parecida com a inicial, porém, agora vamos usar todas as possíveis permutações da palavra *kobus* como inicio do novo alfabeto e simplesmente ver se alguma faz com que Kobus fique em primeiro lugar.

O código abaixo implementa esta ideia.

```cpp
#include "bits/stdc++.h"
using namespace std;

int main() {
    int n; cin >> n;

    vector<string> vs(n);
    for (int i = 0; i < n; i++) cin >> vs[i];

    string kobus = "kobus", alpha = "acdefghijlmnpqrtvwxyz";
    vector<int> permutation({0,1,2,3,4});

    // this do...while will generate every permutation of {0,1,2,3,4}
    do {

        int ok = 1;
        string perm_kob = ""; // will be a permutation of "kobus"
        for (auto x: permutation) perm_kob += kobus[x];

        string new_alphabet = perm_kob + alpha;
        map<char, char> old_to_new; // maps old alphabet to new one
```

```
22          for (int i = 0; i < 26; i++)
23              old_to_new[new_alphabet[i]] = 'a'+i; // 'a'+0=='a', 'a'+1=='b'
24
25          string new_kob = kobus;
26          for (char &c: new_kob) c = old_to_new[c];
27
28          for (auto x: vs) {
29              string s = x;
30              for (char &c: s) // remap each name to new alphabet
31                  c = old_to_new[c];
32              if (new_kob >= s) ok = 0; // kobus not in first place
33          }
34
35          if (ok) {
36              cout << new_alphabet << endl;
37              goto fim;
38          };
39
40      } while (next_permutation(permutation.begin(), permutation.end()));
41
42      cout << "sem chance" << endl;
43      fim:;
44  }
```

B - Guidi quer ficar monstro

Este problema é uma aplicação direta do problema de **Longest Common Subsequence**, que usa a técnica de programação dinâmica. Um excelente artigo sobre esse problema pode ser lido aqui.

A seguir, esta o código da solução.

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

int lcs[1009][1009];

int main() {
    string a, b; cin >> a >> b;

    for (int i = 0; i <= a.size(); i++) {
        for (int j = 0; j <= b.size(); j++) {
            if (i == 0 || j == 0)
                lcs[i][j] = 0;
            else if (a[i-1] == b[j-1])
                lcs[i][j] = 1 + lcs[i-1][j-1];
            else
                lcs[i][j] = max(lcs[i-1][j], lcs[i][j-1]);
        }
    }

    cout << lcs[a.size()][b.size()] << endl;
}
```

C - Harada e os números da sorte

Primeiramente, temos que notar que o valor que Q pode assumir é muito pequeno, no máximo 7. Além disso, podemos criar um vetor auxiliar que nos diz a frequência de cada valor das cartas que Harada ganhou.

Com isso, a ideia é verificar todos os possíveis subconjuntos de números da sorte, checar se é possível montar os números daquele subconjunto utilizando as cartas que Harada ganhou e manter guardado qual é o maior conjunto que satisfaz a condição visto até o momento.

A técnica mais utilizada para gerar todos os subconjuntos utiliza BitMask. Se o seu conjunto possui N elementos, a ideia é iterar sobre todos os valores no intervalo $[0, 2^N - 1]$ e utilizar a representação binaria do numero para decidir qual o conjunto (pego o i-ésimo elemento somente se o i-ésimo bit do numero esta ligado), leia mais sobre essa tecnica aqui.

Segue a implementação.

```cpp
#include "bits/stdc++.h"
using namespace std;

int main() {
    int q, n; cin >> q >> n;

    vector<int> v_q(q);
    for (int i = 0; i < q; i++) cin >> v_q[i];

    // frequency['0']==frequency if card with value 0
    map<char, int> frequency;
    for (int i = 0; i < n; i++) {
        int n_i; cin >> n_i;
        frequency[n_i+'0']++;
    }

    int ans = 0;
    // the mask makes it possible to generate all possible subsets
    for (int mask = 0; mask < (1<<q); mask++) {

        bool ok = 1;
        map<char, int> f = frequency;
        for (int i = 0; i < q; i++) {
            if ((mask&(1<<i)) == 0) continue; // i-th number not selected
            for (auto c: to_string(v_q[i])) {
                f[c]--;
                if (f[c] < 0) ok = 0;
            }
        }

        // __builtin_popcount tell us the number of 1 bits in the binary
        // representation of the number
        if (ok && __builtin_popcount(mask) > __builtin_popcount(ans))
            ans = mask;
    }

    cout << __builtin_popcount(ans) << endl;
```

```
38      for (int i = 0; i < q; i++)
39          if (ans&(1<<i)) cout << v_q[i] << ' ';
40
41      cout << endl;
42  }
```

D - Mashup do Corona

A informação de que toda pessoa que pode participar no dia x também pode participar nos dias depois de x nos dá uma intuição que devemos utilizar ordenação.

A maneira mais fácil de implementar este problema é utilizando um Map, que chamaremos de *day_count*. A ideia é que $day\_count[x]$ nos diga quantas pessoas podem participar a partir do dia x (valor não acumulado). Dado que o Map mantem as chaves ordenadas em sua estrutura interna, podemos iterar sobre essa estrutura e manter um variável que conta quantas pessoas podem participar no contest até o dia atual, sempre acrescentando o respectivo valor da chave da iteração. Toda vez que esse numero for divisível por 3, atualizamos a resposta.

Segue a implementação.

```cpp
#include "bits/stdc++.h"
using namespace std;

int main() {

    long long int n; cin >> n;

    // day_count[x] == 'participantes on day x'
    map<long long int, long long int> day_count;
    for (int i = 0; i < n; i++) {
        long long int x; cin >> x;
        day_count[x]++;
    }

    long long int cnt = 0, ans = -1;
    // here we iterate the elements in the map, this will give days
    // in increasing order. Also, each element in the map is a pair
    // containin a key (in the case, a day) and a value (in this
    // case) the number of participants that can participate in
    // on the given day
    for (auto x: day_count) {
        cnt += x.second;
        if (cnt%3 == 0) ans = x.first;
    }

    cout << ans << endl;
}
```

E - Aprendendo novas linguagens

A ideia desse problema é manter alguma estrutura que nos diga quais palavras nos conhecemos, podemos usar tanto um Set quando o Map para isso. Porém, quando aprendemos uma nova palavra, como fazemos para verificar se não é possível aprender outra palavra, dado que o nosso *vocabulário* aumentou?

Como m só vai até 100 você pode fazer o seguinte: para cada palavra do dicionario, mantenha uma lista de quais palavras você precisa saber para aprende-la. Percorra a lista de palavras do dicionario e veja se fosse conseguiu aprender alguma. Para isso, percorra a lista de palavras correspondes e cheque, usando seu Set/Map, se você conhece todas as palavras dessa lista. Caso você tenha conseguido aprender uma palavra nova, percorra a lista de palavras do dicionario novamente, repetindo este processo até que você não consiga mais aprender uma palavra. Após isso, basta imprimir o numero total de palavras que você sabe.

Segue a implementação.

```cpp
#include "bits/stdc++.h"
using namespace std;

int main() {

    int n; cin >> n;

    map<string, bool> known_words;
    for (int i = 0; i < n; i++) {
        string s; cin >> s;
        known_words[s] = 1;
    }

    int m; cin >> m;
    // learn_list[s] == "words you need to know in order to learn s"
    map<string, vector<string>> learn_list;

    while (m--) {
        string s; cin >> s;
        int k; cin >> k;
        while (k--) {
            string aux; cin >> aux;
            learn_list[s].push_back(aux);
        }
    }

    bool ok = 1;
    while (ok) {
        ok = 0;
        for (auto x: learn_list) {
            if (known_words[x.first]) // already know this word
                continue;

            int cnt = 0;
            for (auto y: x.second) // add 1 if I know the necessary word
                cnt += known_words[y];

```

```
38              if (cnt == x.second.size()) { // know all the necessary words
39                  known_words[x.first] = 1;
40                  ok = 1;
41              }
42          }
43      }
44
45      int ans = 0;
46      for (auto x: known_words)
47          ans += x.second; // add 1 for each known word
48
49      cout << ans << endl;
50  }
```

F - Confundindo o Morete

Para resolver este problema vamos utilizar um vetor de soma acumulada, acumulando os valores nos cartões. Se em nenhum momento o valor acumulado ficou negativo, o Morete errou a conta. Caso contrario, vamos criar uma lista de indices de cartões suspeitos e iterar do fim até os começos dos cartões, isto é, da direita para a esquerda.

Se em algum momento o nosso vetor de soma acumulada ficar menor que 0, limpamos os elementos da nossa lista de cartões suspeitas, pois, nenhuma carta a direita daquela posição vai ser suspeita, dado que retira-la ainda vai fazer com que tenhamos um ponto com valor acumulado negativo.

Caso contrario, checamos se o valor do cartão na posição atual é menor ou igual ao valor minimo visto até agora (valor minimo entre os cartões da posição atual até o fim). Caso positivo, retirar essa cartão significa fazer com que a soma acumulada nunca fique negativa, portanto, este seria um cartão suspeito.

Ao final, basta checar se a lista de indices de cartões suspeitos está vazia. Caso positivo, Morete ficou com saldo negativo. Caso contrario, podemos imprimir os indices das cartas suspeitas

Segue a implementação.

```cpp
#include "bits/stdc++.h"
using namespace std;

long long int r[112345], accumulated_sum[112345];

int main() {

    long long int n; cin >> n;
    for (int i = 0; i < n; i++)
        cin >> r[i];

    bool never_negative = 1;
    for (int i = 0; i < n; i++) {
        if (i > 0) accumulated_sum[i] = accumulated_sum[i-1];
        accumulated_sum[i] += r[i];

        if (accumulated_sum[i] < 0) never_negative = 0;
    }

    if (never_negative) {
        cout << "morete chapou: errou conta!" << endl;
        return 0;
    }

    vector<int> suspects; // kinda sus
    long long int mn = accumulated_sum[n-1];
    for (int i = n-1; i >= 0; i--) {

        // there cannot be any suspect card after this point
        if (accumulated_sum[i] < 0)
            suspects.clear();

        mn = min(mn, accumulated_sum[i]);
```

```
34          if (mn >= r[i]) // remove this card makes always non-negative
35              suspects.push_back(i);
36      }
37
38      if (suspects.empty()) {
39          cout<<"morete chapou: ficou com saldo negativo!"<<endl;
40          return 0;
41      }
42
43      cout << suspects.size() << endl;
44      for (auto x: suspects) cout << x+1 << ' ';
45      cout << endl;
46  }
```

G - Jogo dos pontinhos azuis

Este problema é baseado no Enigma dos Olhos Verdes. A reposta é o valor minimo entre n e a quantidade de alunos com pontos azuis + 1.

Segue a implementação.

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

int main() {

    int amt = 0, n; cin >> n;

    for (int i = 0; i < n; i++) {
        int x; cin >> x;
        amt += x; // add 1 if blue
    }

    int ans = min(n, amt+1);
    cout << ans << endl;
}
```

H - O comediante Nathan

A ideia principal deste problema é utilizar a tecnica de Line Sweep. Basicamente, ao colocar todos os N pares em um vetor e ordena-lo, podemos extrair a resposta muito facilmente.

Podemos imaginar que cada pessoa vai formar um seguimento na linha do tempo, que começa quando ela entra e termina quando ela sai. Além disso, devido ao caráter fofoqueiro do estudantes, podemos pensar que quando dois segmentos se intersectam, eles viram um só. Isso é uma generalização que ajuda a gente encontrar a reposta.

Com isso, podemos calcular a união de todos os segmentos. Por exemplo, no caso de teste da folha, ficaremos com o vetor ordenado $[[2, 7], [6, 9], [8, 13], [21, 25]]$ e sua união é definida pelos pares $[2, 13]$ e $[21, 25]$.

Se um par da união é definido por $[X, Y]$, podemos calcular o numero de piadas necessárias iterando sobre o intervalo $[X, Y)$ e somando 1 á um contador toda vez que o valor da iteração for divisível por 5. Fazendo isso para todos os pares resultantes na união e pegando o maior valor de piadas necessárias podemos obter a resposta.

Exercício de casa: por que podemos simplesmente iterar nos segmentos sem estourarmos o limite de tempo? Segue a implementação.

```cpp
#include "bits/stdc++.h"
using namespace std;

int main() {

    int n; cin >> n;

    vector<pair<int,int>> v;
    for (int i = 0; i < n; i++) {
        int e, s; cin >> e >> s;
        v.push_back({e, 1}); // entering the bus will come after
        v.push_back({s, -1}); // because of the sorting
    }

    sort(v.begin(), v.end());
    int earlier_time = -1, passager_count = 0, ans = 0;

    for (auto x: v) {

        int cur_time = x.first, cur_operation = x.second;

        if (cur_operation == -1) {
            passager_count--;
            if (passager_count == 0) { // last passager to departure
                int jokes = 0;
                for (int t = earlier_time; t < cur_time; t++)
                    jokes += (t%5 == 0); // add 1 if time is divisible by 5
                ans = max(ans, jokes);
            }
        }

        if (cur_operation == 1) {
            if (passager_count == 0)
```

```
34                  earlier_time = cur_time;
35              passager_count++;
36          }
37      }
38
39      cout << ans << endl;
40  }
```

I - Mario Kart Competitivo

Para solucionar este problema você pode criar um vetor auxiliar em que a primeira posição é o numero de pontos ganhos ao ficar em primeiro lugar, a segunda posição é o numero de pontos ganhos por ficar em segundo lugar, e assim por diante. A resposta ótima é sempre dada quando tentamos alcançar as melhores colocações o maior numero de vezes.

Segue a implementação.

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

int main() {

    int n; cin >> n;

    vector<int> ans, points({15, 12, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1});

    while (n > 0) {
        for (int i = 0; i < points.size(); i++) {
            if (n >= points[i]) {
                ans.push_back(i+1);
                n -= points[i];
                break;
            }
        }
    }

    cout << ans.size() << endl;
    for (auto x: ans) cout << x << " ";
    cout << endl;
}
```

J - Raphael Cantor

Com o primeiro par *ano* e *titulo* podemos estimar seu ano de nascimento. Agora basta checar se o ano de nascimento vai ser o mesmo para todos os próximos álbuns.

Segue a implementação.

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

int main() {

    int birth = -1, ok = 1, n;
    cin >> n;

    for (int i = 0; i < n; i++) {

        int a, t;
        cin >> a >> t;

        if (birth == -1) {
            birth = a-t;
            continue;
        }

        if (a-t != birth) ok = 0;
    }

    if (!ok) cout << "mentiu a idade" << endl;
    else     cout << "idades corretas" << endl;
}
```