Aula 7 · Técnicas Diversas: Dois Ponteiros, Compressão de Coordenadas e mais — e Discussão de Problemas Anteriores

Desafios de Programação

Fernando Kiotheka Victor Alflen

UFPR

20/07/2022

Dois Ponteiros

O que são “dois ponteiros?”

- Um ponteiro é uma variável que aponta para uma posição, geralmente de um vetor.
- Geralmente ao invés de usarmos ponteiros literalmente, usamos índices.
- Em sua essência, a técnica dos dois ponteiros é a de utilizar dois ponteiros para o vetor para resolver um problema em um único laço $\mathcal{O}(n)$.
- Exemplo: A fusão de vetores ordenados usada no MergeSort.
- Recomendamos o curso de dois ponteiros da ITMO: `https://codeforces.com/edu/course/2/lesson/9`.

2SUM

Dado um vetor **ordenado** V e um valor X, encontre dois valores de V cuja soma seja X (ou informe que tal par não existe).

$$X = 24$$

$$V = \{1, 5, 6, 7, 14, 19, 21\}$$

$$\text{Resposta: } (5, 19)$$

Solução: Um ponteiro no início e outro no final; ambos se aproximam até achar a soma X (ou trocarem de lado, caso o par não exista)

Implementação do 2SUM

Código `2sum.cpp`

```
#include <bits/stdc++.h>
using namespace std;
int main() {
    int n, x; while (cin >> n >> x) {
        vector<int> v (n); for (int& x : v) { cin >> x; }
        int i = 0, j = n - 1, ok = 0;
        while (i < j) {
            int s = v[i]+v[j];
            if (ok = (s == x)) {
                cout << v[i] << " " << v[j] << "\n"; break; }
            if (s > x) { j--; } else { i++; }
        } if (!ok) { cout << "N\n"; }
    }
}
```

Entrada	Saída
7 24 1 5 6 7 14 19 21	5 19
6 24 1 5 6 7 14 18	6 18
6 24 1 5 6 7 14 15	N

Soma de subvetor

Dado um vetor V e um valor X, encontre um subvetor U (também chamado de substring, é uma sequência contígua de valores de V) cuja soma seja X (ou informe que tal subvetor não existe).

$$X = 27$$

$$V = \{1, 5, 6, 7, 14, 19, 21\}$$

$$\text{Resposta: } \{6, 7, 14\}$$

Solução: Ponteiros representam os limites do vetor; o do ínicio fica fixo enquanto o do final tenta chegar à soma X. Se a soma passar, aumentamos o início.

Implementação de soma de subvetor

Código `subsum.cpp`

```
#include <bits/stdc++.h>
using namespace std;
int main() {
    int n, s; while (cin >> n >> s) {
        vector<int> v (n); for (int& x : v) { cin >> x; }
        int i = 0, j = 1, w = v[0], ok = 0;
        while (j < n) {
            while (w + v[j] > s && i < j-1) { w -= v[i++]; }
            if (w < s) { w += v[j++]; }
            if (w == s) {
                for (int k = i; k < j; k++) cout << v[k] << " ";
                cout << "\n"; ok = 1; break; }
        } if (!ok) { cout << "N\n"; }}
}
```

Entrada	Saída
7 27 1 5 6 7 14 19 21	6 7 14
6 28 1 5 6 7 14 18	N
6 47 1 5 6 7 14 15	5 6 7 14 15

Compressão de Coordenadas

Objetivo

Dado um vetor de $1 \leq N \leq 10^5$ elementos $0 \leq X_i \leq 10^9$, queremos criar um vetor cuja posição $C[X_i]$ diz quantas vezes cada elemento X_i aparece.

Código `coordpre.cpp`

```
#include <bits/stdc++.h>
using namespace std;
int main() {
    int n; cin >> n;
    vector<int> c (1e9+1, 0);
    for (int i = 0; i < n; i++) {
        int xi; cin >> xi; c[xi]++;
    }
}
```

Porém, não temos memória para guardar 10^9 inteiros...

Solução simples

- Podemos usar um `map<int, int>` pra contar.
- A cada acesso teremos um fator $\mathcal{O}(\lg n)$, mas como são apenas 10^5 inteiros, isso é tolerável.
- É importante ressaltar que um `multiset<int>` também funcionaria, mas lembre-se que `m.count(int v)` é $\mathcal{O}(n)$, então é melhor evitar.
- Vamos implementar um programa que lê os inteiros e os imprime em ordem de frequência (mais frequente primeiro).
- Se você já usou Python, vamos implementar a principal funcionalidade de uma coleção bem legal: `Counter`.

Implementação do multiset/bag ordenado por frequência

Código bag.cpp

```
#include <bits/stdc++.h>
using namespace std;
int main() {
    int n; cin >> n;
    map<int, int> c;
    for (int i = 0; i < n; i++) {
        int xi; cin >> xi; c[xi]++; }
    vector<pair<int, int>> bag;
    for (auto [x, f] : c)
        bag.push_back(make_pair(f, x));
    sort(bag.rbegin(), bag.rend());
    for (auto [f, x] : bag)
        cout << x << "x" << f << " ";
}
```

Entrada	Saída
10 2 3 1 2 3 4 5 1 1 9	1x3 3x2 2x2 9x1 5x1 4x1

Voltando ao problema original

- Como usar uma árvore de Fenwick indexada pelos valores?
- Como no pior caso são apenas N valores distintos, podemos atribuir uma representação mais compacta a cada um.
- Vamos normalizar os valores no vetor para que tenham um valor correspondente em $[1, N]$.
- Essa técnica é chamada de **compressão de coordenadas**.
- Em vez de indexar pelo valor, indexamos pela resposta à pergunta "em qual posição esse valor ficaria se ordenássemos os valores únicos?" (algo de $[1, N]$)

Compressão usando map

Guardamos os valores únicos em um set e, a partir disso, encontramos a posição correspondente a cada valor (e o valor correspondente a cada posição)

Código coordmap.h

```
void compress(vector<int>& v, vector<int>& cv) {
    set<int> vals (v.begin(), v.end());
    map<int, int> ida, volta;
    int ix = 0;
    for (int i : vals) {
        ida[i] = ix;
        volta[ix] = i;
        ix++;
    }
    for (int i = 0; i < v.size(); ++i)
        cv[i] = ida[v[i]];
}
```

Compressão usando busca binária

Podemos fazer isso mais facilmente com uma busca binária de cada elemento sobre um vetor dos valores únicos!

Código `coordbs.h`

```
#define all(x) (x).begin(), (x).end()
void compress(vector<int>& v, vector<int>& cv) {
    vector<int> vals = v;
    sort(all(vals));
    // opcional, para retirar repetidos
    vals.erase(unique(all(vals)), vals.end());
    for (int i = 0; i < n; ++i) {
        int ix = lower_bound(all(vals), v[i])
            - vals.begin();
        cv[i] = ix;
    }
}
```

Coordenadas usando ordenação de pares

Também podemos fazer compressão ordenando um vetor de pares.

Código `coordpair.h`

```
#define all(x) (x).begin(), (x).end()
void compress(vector<int>& v, vector<int>& cv) {
    // passos usando uniques opcionais, para tirar repetidos
    vector<int> uniques = v;
    sort(all(uniques));
    uniques.erase(unique(all(uniques)), uniques.end());
    vector<pair<int, int>> a (uniques.size());
    for (int i = 0; i < uniques.size(); i++)
        a[i] = make_pair(uniques[i], i);
    sort(a.begin(), a.end());
    for (int i = 0; i < uniques.size(); i++)
        cv[a[i].second] = i;
}
```

Problema dos Atrasados Anônimos

Queremos contar inversões em um vetor de $1 \leq N \leq 10^5$ elementos, onde cada elemento vale $1 \leq P \leq 10^9$. Os primeiros H elementos não contam para as inversões.

- **Uma solução:** BIT com compressão de coordenadas!
- **Uma solução elegante:** Contagem usando Merge Sort

Implementação dos Atrasados Anônimos

Código atrasados.cpp

```
#include <bits/stdc++.h>
using namespace std; using ll = long long;
const int N = 1e5+15;
#include "bit.h"
#include "coordmap.h"
int main() {
    int n, h; cin >> n >> h;
    vector<int> v (n), cv (n);
    for (auto &i : v) cin >> i;
    compress(v, cv);
    ll ans = 0;
    for (int i = 0; i < n; ++i) {
        if (i >= h)
            ans += get(n) - get(cv[i]+1);
        add(cv[i]+1, +1);
    }
    cout << ans << endl;
}
```

Teto de Divisão de Inteiros

Problema

- Existe a função `ceil` que funciona com `double`, porém `double`s são limitados.
- Lembre-se sempre que o maior **inteiro** representado por `double` é 2^{53}. Depois disso fica *esparso*.
- Podemos aplicar matemática discreta para obter o teto a partir da operação que temos: chão da divisão.

Solução

Comecemos com $\frac{x}{y}$. $\left\lceil \frac{x}{y} \right\rceil$ = este valor, arredondado para cima.
Como obter isso se só temos à nossa disposição o piso?

- Perceba que $\lfloor a \rfloor \leq \lceil a \rceil < a + 2$, isto é, o piso é igual ao chão ou 1 a mais que ele
- Fazendo $\left\lfloor \frac{x+y-1}{y} \right\rfloor$, estamos somando o maior valor possível menor que 1 à fração original (a partir de $\frac{x}{y}$ temos valores maiores ou iguais a 1)
- Se o resto de $\frac{x}{y} < \frac{y}{2}$, teremos um valor igual ao piso. Caso contrário, teremos 1 a mais.

Para evitar *overflow* em $x + y$, ainda tiramos o y do denominador:

$$\left\lceil \frac{x}{y} \right\rceil = 1 + \left\lfloor \frac{x-1}{y} \right\rfloor$$

Implementação do teto de inteiros

Código `floor.cpp`

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

int main() {
    int x = 5;
    int y = 10;

    int floor1 = x/y;

    int ceil1 = (x+y-1)/y;
    int ceil2 = x/y + bool(x%y);

    int nextup = x+y-1 - (x+y-1)%y;
    int nextdw = x - (x%y);
}
```

Meet in the Middle

Problema de soma de subsequências

Dado um vetor V e um valor X, encontre quantas subsequências U têm soma X.

$$X = 5$$

$$V = \{1, 2, 3, 2\}$$

$$\text{Resposta: } 3$$

Solução: *Meet in the Middle*

Meet in the Middle

- Vamos guardar em um conjunto $\mathcal{S}$ quantas subsequências na primeira metade do vetor acumulam alguma soma $S \leq X$ (isto é, para cada soma guardamos a contagem - cheiro de `map`!)
- Depois, na segunda metade, vemos quantas subsequências acumulam alguma soma R tal que $R + S = X$ para algum $S \in \mathcal{S}$

Implementação do Meet in the Middle

Código `meet.cpp`

```
#include <bits/stdc++.h>
using namespace std; using ll = long long;
int main() {
    int n, x; cin >> n >> x;
    vector<int> v (n); for (int& el : v) { cin >> el; }
    map<ll, int> sums; ll c = 0;
    for (int ss = 0; ss < 1<<(n/2); ss++) {
        ll s = 0;
        for (int i = 0; i < n/2; i++) if (1<<i & ss)
            s += v[i];
        if (s <= x) { sums[s]++; }
    }
    for (int ss = 0; ss < 1<<(n/2+n%2); ss++) {
        ll s = 0;
        for (int i = n/2; i < n; i++) if (1<<(i-n/2) & ss)
            s += v[i];
        if (sums.count(x - s)) { c += sums[x - s]; }
    }
    cout << c << "\n";
}
```

Usos de funções prontas de logaritmos

Funções prontas de logaritmos

- Na biblioteca padrão, existem funções para logaritmos (`log`, `log10`, `log2`) e raiz quadrada (`sqrt`) prontos, além de outras funções matemáticas como `acos`.
- Elas produzem saída em ponto flutuante e são suficientemente rápidas (inclusive muitas vezes usando instruções de hardware!).
- É melhor evitar ponto flutuante, mas as vezes somos "obrigados" a usá-los.

Log para comparação

Compara potências de d^c para ver qual é a maior.

Código `log.cpp`

```
#include <bits/stdc++.h>
using namespace std;
int main() {
    int n; cin >> n;
    double best = 0;
    int best_i = 0;
    for (int i = 0; i < n; i++) {
        int d, c;
        cin >> d >> c;
        double f = c*log(d);
        if (f > best) { best = f; best_i = i; }
    }
    cout << best_i << "\n";
}
```

Log para pegar primeiros dígitos

Pega os três primeiros dígitos de n^k.

Código `log10.cpp`

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
int main() {
    int k; ll n;
    while (cin >> n >> k) {
        double logged = k*log10(n);
        logged -= int(logged);
        logged += 2; // queremos mais 2 dígitos (3 no total)
        int first = int(pow(10, logged));
        cout << first << "\n";
    }
}
```

Problemas Anteriores

ing'

Dada a quantidade de números disponíveis para cada coluna da cartela e a posição de algumas casas coringa da cartela, quantas cartelas são possíveis?

- Todas as colunas inicialmente podem ter N números.
- Acumula-se quantidade de coringas naquela coluna, W_i.
- Quantidade de casas da coluna é $C_i = N - W_i$.
- Quantas combinações possíveis para uma coluna?

$$\binom{B_i}{C_i} C_i! = \frac{B_i!}{C_i!(B_i - C_i)!} C_i! = \frac{B_i!}{(B_i - C_i)!}$$

- Como computar rápido: Pré-calcular todos os fatoriais.
- Não cabe? O módulo cabe.
- Divisão em módulo: $a^{-1} = a^{p-2} \pmod{P}$, onde P é primo.
- Exponenciação binária.

Fita Remendável de Acesso Aleatório

- A ideia é "simular a deleção" em complexidade boa.
- O vetor original nunca muda, então ele permanece no lugar, o que mudam são os índices que podem ser acessados.
- **Ideia:** Manter uma árvore de Fenwick com os índices que podem ser acessados. Quando deletamos um elemento, subtraímos 1 de um intervalo de índices da árvore.
- Podemos usar a Árvore de Segmentos também, só precisamos saber como fazer atualização em intervalo e consulta em posição.
- Quando queremos saber qual o índice "real" dado um índice "falso", fazemos uma busca binária pelo índice na Árvore de Fenwick em $O(\lg^2 n)$.

Árvore de Índices

Inicialmente todos os índices equivalem a suas posições reais:

$$A = \left[{}_0 1 \, {}_1 2 \, {}_2 3 \, {}_3 4 \, {}_4 5 \, {}_5 6 \, {}_6 7 \, {}_7 \right]$$

Quando deletamos o elemento 3 por exemplo, subtraímos 1 de todo o intervalo de 3 até o final:

$$A = \left[{}_0 1 \, {}_1 2 \, {}_2 2 \, {}_3 3 \, {}_4 4 \, {}_5 5 \, {}_6 6 \, {}_7 \right]$$

Isso significa que se fizermos uma busca binária pelo índice 3, "pulamos" o elemento 3, e pegamos o índice que aponta para o elemento 3 real no vetor.
É importante fazer essa busca binária também para deletar, para achar onde deletar, assumindo que apenas mantemos no nosso código os índices "falsos" (movemos pra esquerda e pra direita sem se preocupar com esses pulos).

Busca binária na Árvore de Fenwick em $\mathcal{O}(\lg^2 n)$

Código `fenwick-bs.h`

```
int lower_bound(int t) {
    int lo = 1, hi = N+1;
    while (lo < hi) {
        int mi = lo + (hi - lo) / 2;
        if (query(mi) < t) {
            lo = mi + 1;
        } else {
            hi = mi;
        }
    }
    return lo;
}
```

Busca binária na Árvore de Fenwick em $\mathcal{O}(\lg n)$

Código `fenwick-bslog.h`

```
int lower_bound(int t) {
    int sum = 0, pos = 0;
    for (int p = log2(N); p >= 0; i--) {
        if (
            pos + (1 << p) < N
            && sum + ft[pos + (1 << p)] < t
        ) {
            sum += ft[pos + (1 << p)];
            pos += (1 << p);
        }
    }
    return pos;
}
```