Editorial Pseudo-seletiva UFMG 2020

Bruno Monteiro

Universidade Federal de Minas Gerais

14 de Agosto de 2020

I - Word Correction

- Iteramos pelas letras; se a letra atual é vogal e a letra anterior também, ignorâ-mo-la. Caso contrário, printamos a letra atual.
- Devemos tomar cuidado, pois y é considerado vogal.

```
1  set<char> vog = {'a', 'e', 'i', 'o', 'u', 'y'};
2  for (int i = 0; i < n; i++) {
3      if (i > 0 and vog.count(s[i]) and vog.count(s[i-1])) continue;
4      cout << s[i];
5  }
```

- Complexidade: $\mathcal{O}(n)$

C - Death Note

- Mantemos quantos nomes estão escritos na última página. Para cada dia, incrementamos o número de nomes que devem ser escritos nesse dia.
- O número de vezes que viramos a página é essa soma dividido por m.

```
1  int resto = 0;
2  for (int i = 0; i < n; i++) {
3      resto += a[i];
4      cout << resto/m << " ";
5      resto %= m;
6  }
```

- Complexidade: $\mathcal{O}(n)$

A - Distances to Zero

- Primeiro computamos a menor distância entre todas as posições e o zero à esquerda mais próximo, mantendo a posição do último zero.
- Repetimos o processo de trás para frente para computar a menor distância até o zero à direita mais próximo. A resposta é o mínimo entre esses dois valores. Complexidade: $\mathcal{O}(n)$

```
1  vector<int> dist(n, 1000000);
2  for (int i = 0, last = -1; i < n; i++) {
3      if (v[i] == 0) last = i;
4      if (last != -1) dist[i] = abs(i-last);
5  }
6  for (int i = n-1, last = -1; i >=0; i--) {
7      if (v[i] == 0) last = i;
8      if (last != -1) dist[i] = min(dist[i], abs(i-last));
9  }
10 for (int i : dist) cout << i << " ";
```

K - Run For Your Prize

- Só importa o prêmio mais distante da pessoa mais próxima.
- Isso é verdade porque todos os outros podem ser pegos antes desse prêmio, se em cada momento a pessoa da esquerda andar para a direita, e a da direita andar para a esquerda.

```
1  int maxDist = 0;
2  for (int i = 0; i < n; i++) {
3      int a; cin >> a;
4      maxDist = max(maxDist, min(abs(1-a), abs(1000000-a)));
5  }
6  cout << maxDist << endl;
```

- Complexidade: $\mathcal{O}(n)$

G - Vasya and Book

- Devemos tratar alguns casos. Se a distância entre `x` e `y` é múltipla de `d`, podemos ir diretamente de `x` para `y`.
- Caso contrário, será necessário ir para uma das bordas e depois para `y`. Testamos essas duas possibilidades, e vemos qual, se válida, requer menos movimentos.
- Dica de implementação (para `a` e `b` inteiros):

$$\left\lceil \frac{a}{b} \right\rceil = \left\lfloor \frac{a+b-1}{b} \right\rfloor$$

- Complexidade: $\mathcal{O}(1)$

G - Vasya and Book

```
1  if ((x-y) % d == 0) cout << abs(x-y)/d << endl; // primeiro caso
2  else {
3      int ans = INF;
4
5      // esquerda
6      if (y%d == 0) ans = min(ans, (x+d-1)/d + y/d);
7
8      // direita
9      if ((n-y)%d == 0) ans = min(ans, (n-x+d-1)/d + (n-y)/d);
10
11     cout << (ans == INF ? -1 : ans) << endl;
12 }
```

E - Segment Occurrences

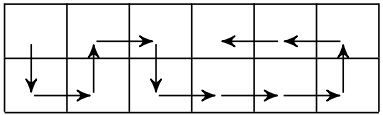
- Podemos computar cada posição que dá *match*, e computar um array de soma de prefixo das posições que dão *match*.
- Ao responder uma *query*, devemos ter cuidado para não considerar *matchings* que same do intervalo que queremos.

```
1  while (q--) {
2      int l, r; cin >> l >> r; l--, r--;
3      r = r - m + 1;
4      int ret;
5      if (r < l) ret = 0;
6      else if (l == 0) ret = pre[r];
7      else ret = pre[r] - pre[l-1];
8      cout << ret << endl;
9  }
```

- Complexidade: $\mathcal{O}(nm + q)$ ou $\mathcal{O}(n + m + q)$

D - Vasya And The Mushrooms

- É sempre ótimo fazer um zigue-zague até certa posição, depois ir até o final e voltar.
- Assim, computamos uma soma do sufixo de "ir e voltar", e fazemos brute-force no tamanho do zigue-zague.



- Complexidade: $\mathcal{O}(n)$

D - Vasya And The Mushrooms

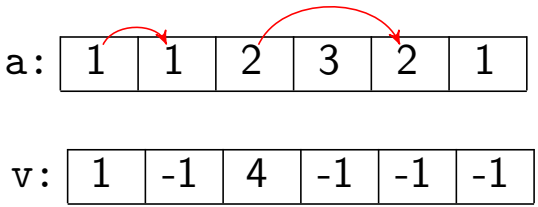
```
1  vector s(2, vector(n+1, ll()));
2  ll sum = 0, zig = 0, ans = 0;
3  for (int i = n-1; i >= 0; i--) {
4      sum += v[0][i] + v[1][i];
5      s[0][i] = s[0][i+1] + 2*n*v[1][i] + (2*i+1)*v[0][i] - sum;
6      s[1][i] = s[1][i+1] + 2*n*v[0][i] + (2*i+1)*v[1][i] - sum;
7  }
8  for (int i = 0; i < n; i++) {
9      ans = max(ans, zig + s[i&1][i]);
10     zig += 2*i*v[i&1][i] + (2*i+1)*v[!(i&1)][i];
11 }
12 cout << ans << endl;
```

J - Buy a Ticket

- Duplicamos os pesos das arestas e adicionarmos um vértice especial x, com arestas (x, i) de peso a_i.
- Notemos agora que a resposta para o vértice i é o caminho mínimo de x para i.
- Usando o algoritmo de Dijkstra com vértice x como fonte, podemos computar a resposta.

B - One Occurrence

- Vamos resolver o problema *offline*: vamos responder todas as *queries* que têm `l` como posição da esquerda.
- Vamos manter o seguinte array `v`: inicialmente $v[i] = -1 \ \forall i$.
- Considerando apenas o array inicial das posições `l` até `n`, vamos colocar, na primeira ocorrência de cada valor, o índice da segunda ocorrência dessa valor.



B - One Occurrence

- Para responder uma query, precisamos olhar no range equivalente do array `v` e verificar se algum valor no range é maior que `r` (isso nos diz que esse valor só ocorre uma vez).
- Podemos então usar uma *segment tree*, e fazer query de máximo do intervalo.
- Atualizamos a *segment tree* quando andamos com o `l`.
- É possível também resolver o problema usando o algoritmo de MO: mantemos a frequência de cada valor, e quando ela chega em 1, colocamos o valor numa fila ou pulha. Para fazer a query, enquanto o topo da pilha não tiver frequência 1, o tiramos da pilha.
- Complexidade: $\mathcal{O}(n \log n)$ ou $\mathcal{O}(n\sqrt{n})$

F - Petya and Graph