Aula 04 - Bizus e Paradigmas

Kenji Yamane

Março de 2020

Tópicos da aula

Vector e Strings

Inicialmente vamos primeiro terminar o assunto de estruturas de dados falando de duas estruturas lineares bastante usadas: vector e strings! (seriam os vetores e strings de c++).

O primeiro é como se fosse um deque com mais algumas funcionalidades, enquanto o segundo é a estruturas de dados de c++ preparada para guardar palavras. (Com ela vocês não precisam mais tratar palavras como um vetor de char, agora existe um "tipo" string).

Vector

Vector, implementado na biblioteca vector, além das funções normais de deque, tem as funções de poder inserir e apagar elementos de qualquer posição dos dados em O(N), como exemplificado a seguir.

Código 1

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


int main( ){
    vector<int> v = {3, 4, 5, 6, 7, 8, 9};
    v.erase(v.begin() + 3); //{3, 4, 5, 7, 8, 9}
    v.insert(v.begin() + 2, 31); //{3, 4, 31, 5, 7, 8, 9}
    return 0;
}
```

Strings

Strings, inclusas na biblioteca string são as estruturas de dados que guardam palavras, e elas são bem parecidas com variáveis em c++!

Código 2

```
#include <iostream>
#include <string>


int main( ){
    string str = "corona" + "virus";
    if (str == "coronavirus") cout « str;
    str[2] = 'y'; //str == "coyonavirus"
    return 0;
}
```

Strings

As strings ainda podem ser tratadas de forma parecida com vector, por ser também uma estrutura de dados, tendo também funções como as seguintes.

- front()
- back()
- push_back(x)
- pop_back()
- size()

Biblioteca bizu

Uma boa quantidade de estrutura de dados foi apresentada, cada uma em uma biblioteca diferente e algumas nem fazem sentido o nome da biblioteca (utility?). Mas aí não se preocupem que existe uma biblioteca que inclui todas as bibliotecas, contando com stdio, stdlib, todas as bibliotecas de c e c++. O nome dela é **bits/stdc++.h**.

Agora vocês não precisam mais inserir um monte de bibliotecas no começo. Mas saibam que usar essa biblioteca é má prática, então considere seu uso permitido somente nas competições.

EOF

Existem alguns problemas que não dão quantos casos eles vão ler. Por exemplo, suponha um problema simples que dado um número lido N, ele quer que você imprima o número N, até EOF. EOF significa (End Of File). Os juízes online normalmente testam os códigos com um arquivo de entrada, e esse arquivo tem vários casos. Quando o problema diz aquilo, ele quer que você pare quando identificar o final do arquivo. Como fazer isso? scanf é uma função que retorna -1 quando não consegue ler, então basta ler até não der mais:

Código 3

```
while (scanf("%d", &N) != -1)
 printf("%d\n");
```

Acelerar cin cout

Vocês provavelmente não sabem, mas cin e cout são mais lentos que printf e scanf, tanto a ponto de em alguns raros casos, trocar de cin e cout para printf scanf pode realmente fazer um programa que estava dando TLE para accepted (bem raro, mas acontece). Só que, cin e cout são mais fáceis de usar e strings de c++ só podem ser lidas por eles. Se quiserem continuar usando cin cout saibam que é possível torná-las tão rápidas quanto printf scanf com as seguintes linhas mágicas no seu código:

Código 4

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

Como se calcula o número a mod b? Com essa equação?

$$modAB = a\%b;$$

Não! Essa equação não funciona para números negativos. O correto é:

$$modAB = (a\%b + b)\%b;$$

Números flutuantes

Use floats e doubles com precaução. Erros de aproximação são uma coisa muito chata e eles sempre acontecem quando um autor de um problema pensou numa resolução com números inteiros com erro 0 de aproximação. Mas se não tiver outro jeito, lembrem-se de sempre comparar igualdade de dois pontos flutuantes com epsilon, não com ==.

Algorithm

As seguintes funções da biblioteca Algorithm são simples mas bem úteis!

- max(a, b) $\Rightarrow$ (a $>$ b ? a : b)
- min(a, b) $\Rightarrow$ (a $<$ b ? a : b)
- sort (v, v + N) $\Rightarrow$ ordena o vetor v de 0 a N - 1

Last touches

Três últimos bizus:

- Não complique desnecessariamente um problema simples. Por exemplo, um problema que só precisa de ferramentas de queue e aí você trata com vector e inverte e brinca e dá erase um monte de vezes nos dados.
- Usem e abusem das variáveis globais. Saibam também que elas já vem inicializadas. Porém, mesma história da bits/stdc++. Elas são má prática e só considere permitido no nosso clubinho.
- Não conheço um juíz online que não requer \n no final das frases. Ou seja faça cout « x « '\n', não cout « x;

O que são paradigmas?

Existe uma variedade enorme de problemas por aí nos juízes online e nas competições. Com relação à solução de alguns desses problemas, pode ser algo meio carteado e obscuro. Mas existem raciocínios de resolução de problemas que constituem a resposta ou parte da resposta para uma boa parte dos problemas, podendo ajudar bastante na construção de um algoritmo-solução. Esses raciocínios se chamam paradigmas e são os seguintes:

- Força Bruta
- Guloso
- Dividir para Conquistar
- Programação Dinâmica

Como funciona força bruta

Esse paradigma é bem simples e direto: teste todas as possibilidades de resposta e veja qual dá certo. Ele exprime bem a ideia de um paradigma também: um raciocínio relativamente generalizado que é a resolução ou ajuda na resolução de vários problemas. Todavia, **cuidado**, esse paradigma normalmente dá a resposta certa quando você o aplica mas vocês devem conseguir ver isso também: frequentemente dá TLE.

Exemplos de problemas

Dados três inteiros A, B e C ($0 < A,B,C < 10001$), encontre outros três inteiros x, y, z tal que $x + y + z = A; xyz = B; x^2 + y^2 + z^2 = C$; Você pode dar uma de cursinho e resolver isso usando equações carteadas de polinômios em O(1). Porém, nosso paradigma em questão serve para isso! Só testar todos os x, y, e z. O problema especificou que são inteiros. Você poderia considerar a primeira equação e perceber que os números tem que estar entre -10000 e 10000. Assim você poderia testar todos as triplas de inteiros com essa limitação, como mostra o próximo slide.

Código 5


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

int main()
{
  int A, B, C;
  cin >> A >> B >> C;
  for (int x = -10000; x <= 10000; x++)
    for (int y = -10000; y <= 10000; y++)
      for (int z = -10000; z <= 10000; z++)
        if (x+y+z == A)
          if (x*y*z == B)
            if (x*x + y*y + z*z == C)
              printf("%d %d %d\n", x, y, z);
  return 0;
}
```

Exemplos de problemas

A complexidade do código anterior é $O(N^3)$. Como N nesse caso é 10000, seu computador nunca aguentaria esse algoritmo. Mas aí que entra a otimização. Você pode testar todas as possibilidades, mas vai descartando o espaço amostral. Nesse caso se você considerar a terceira equação, x e y e z só podem ir até 100, o que torna o algoritmo viável.

Um outro ponto nesses algoritmos de força bruta é algo simples mas desafiador: como testar todas as possibilidades? Por exemplo, outro problema: dada uma sequência de números determine se existe uma subsequência contígua tal que sua soma é S. A resposta se encontra no próximo slide.

Código 6


```
...
int N, arr[1000], S;
cin >> N;
for (int i = 0; i < N; i++) cin >> arr[i];
cin >> S;
for (int i = 0; i < N; i++)
  for (int j = i; j < N; j++){
    int sum = 0;
    for (int k = i; k <= j; k++)
      sum += arr[k];
    if (sum == S) printf("encontrado\n");
  }
return 0;
}
```

Exemplos de problemas

Complexidade também de $O(N^3)$, com uma certa chance de dar TLE dependendo da entrada (os paradigmas das próximas aulas são menos propensos a dar TLE). Em outros problemas, avaliar todos os resultados pode ser mais fácil de se realizar utilizando recursão.

Em outro exemplo de problema, suponha que, dada uma sequência e um valor S, eu queira saber se existe subsequência (dessa vez não necessariamente contíguas) tal que a soma seja igual a S.

Código 7


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

int N, arr[100], S;

void func(int position, int sum){
  if (position == N && sum == S)
    printf("encontrado\n");

  //ou o numero n entra na soma
  func(position + 1, sum);
  //ou entra
  func(position + 1, sum + arr[position]);
}
```

Código 7


```
int main(){
  cin >> N;
  for (int i = 0; i < N; i++) cin >> arr[i];
  cin >> S;
  func(0, 0);
  //comeca na posicao 0
  //e com soma acumulada 0
  return 0;
}
```

Exemplos de problemas

O algoritmo é simples, ele só é difícil de entender por ser recursivo. É o mesmo princípio de como contar quantos subconjuntos um dado conjunto tem. Para gerar todos os subconjuntos, cada elemento pode ter duas possibilidades: 1 ou 0, ou ele entra nesse subconjunto ou não. Claramente, complexidade de $O(2^N)$ o que é absurdamente grande.

Para ordenar um vetor, é possível fazer por força bruta. Basta testar todas as permutações e verificar qual a que está ordenada. Mas seria em O(N!) e existem vários outros algoritmos para resolver esse tipo de problema. Um dos mais simples é o bubble sort ($O(N^2)$), a seguir.

Código 9


```
...
  int N, arr[1000];
  cin >> N;
  for (int i = 0; i < N; i++) cin >> arr[i];
  for (int i = N; i > 0; i--)
    for (int j = 0; j < i - 1; j++)
      if (arr[j] > arr[j + 1]){
        int aux = arr[j];
        arr[j] = arr[j + 1];
        arr[j + 1] = aux;
      }
  //o laco do j transfere o maior ate i - 1
  return 0;
}
```

Problemas de força bruta

Obs 1: Os três últimos problemas são do uva, um site em decadência mas que tem problemas muito bons. Se o link não funcionar, tente em um navegador diferente, ou pesquisa no google direto.

Obs 2: Entenda a diferença entre uma linha entre os casos de teste e uma linha depois de cada caso de teste para não sofrer com presentation error no problema division.

- Baile de Reconciliação
- Média
- Division
- Rat Attack
- 8 Queens Chess Problem