Aula 1 · Introdução a C++ e a Biblioteca Padrão, Juízes e Complexidade

Desafios de Programação

Fernando Kiotheka Victor Alflen

UFPR

08/06/2022

Objetivos

- Temos como objetivo principal motivar e preparar alunos para participar da Maratona de Programação da SBC, porém isso não é requisito.
- A ideia é fortalecer o grupo Capimara UFPR
- Outro lugar onde esse tipo de conhecimento pode ser aplicado é em entrevistas de emprego com desafios lógicos.
- Mas em sua essência, esta é uma matéria de algoritmos. Apresentaremos várias ideias que podem ser usadas para resolver vários problemas diferentes.

Pré-requisitos

- Algoritmos II (filas, pilhas), preferencialmente III (árvores);
- Análise de complexidade básica (de Algoritmos II);
- Noção mínima do que é um grafo (Grafos é recomendado).

Maratona de Programação da SBC

- A ideia é resolver problemas difíceis em pouco tempo e sob muita pressão
- Prova (normalmente presencial) com cerca de 13 questões
- Resolvendo problemas, seu time ganha balões
- Times de 3 estudantes universitários
- Material de consulta é permitido (construa um caderno!)
- Fases regional e nacional
- Nacional é a regional do International Collegiate Programming Contest (ICPC), que é mundial

Cronograma extracurricular

- I Maratona Interna de 2022 da UDESC: **25 de junho de 2022**
- 1º Treino Capimara UFPR de 2022: **A definir**
- Regional da Maratona SBC: **8 de outubro de 2022**
- Final brasileira da Maratona SBC: **16-18 de março de 2023**

Avaliação

- Competiçõezinhas no Juisto (um juiz que está no DINF).
- Iniciarão logo depois da aula acabar.
- Findarão um pouco antes da aula começar.
- Competição é só o nome. A nota é dada de forma individual conforme a quantidade de problemas que você resolver dentro da semana.
- Os problemas são ordenados do mais fácil para o mais difícil.
- Durante a semana, é permitido discutir os problemas, mas não é permitido compartilhar o código com seus colegas (plágio).
- A média final é dada pela média das 15 competições.
- Se você resolver os problemas depois do prazo, você obterá 1/3 da nota que obteria se tivesse feito na semana.
- Ocasionalmente teremos problemas bônus.

Formato das Aulas

- 2 horas de aula presencial na quarta (gravação a discutir).
- Não há aula na sexta, use o tempo para fazer a competição!
- Qualquer tempo de aula que sobrar pode ser usado para discutir problemas da última semana.
- Slides, links e conteúdo disponíveis no site da disciplina.
- Os slides podem ser usados para consulta pelos alunos.
- Linguagem de programação utilizada vai ser o C++, as submissões ao juiz da disciplina serão limitadas a C++.

Introdução a C++

Por que C++?

- C++ é C com baterias incluídas, com muitas estruturas de dados e algoritmos prontos
- A biblioteca padrão, a STL já contém implementações de:
  - Alocação dinâmica (vetores que crescem de tamanho sozinhos)
  - Árvores balanceadas (geralmente rubro-negras)
  - Filas de prioridade
  - Ordenação
  - Pares (ordenação já embutida)
  - Números complexos (pontos 2D)
  - Geração de permutações
  - Geração de números aleatórios
  - Expressões regulares
  - Manipulação de strings de forma dinâmica
  - Entrada/Saída mais concisa

Estrutura básica de um código em C++

Aviso que não são "boas práticas de programação". Usar isso em código *enterprise* é pedir pra ser demitido.

1. Incluímos todos os cabeçalhos da STL de uma vez só (equivalente a `#include <set>`, `#include <vector>`, etc). **Exclusivo ao GCC** e aumenta o tempo de compilação.
```
#include <bits/stdc++.h>
```
2. Usamos o espaço de nomes `std`. Assim, pra usar um `vector` não precisamos digitar `std::vector`.
```
using namespace std;
```
3. Encurtamos o nome do `long long` que é bastante usado.
```
using ll = long long;
```
4. Função `main` padrão. É possível usar `signed` ao invés de `int` pra permitir barbaridades como `#define int long long`.
```
int main() {
```

Estrutura básica de um código em C++ (continuado)

5. Desvinculamos a entrada da saída padrão. O comportamento padrão é fazer um `flush` da saída padrão ao ler da entrada. Fazer menos `flush` torna a saída mais rápida.

```
    cin.tie(0);
```

6. Dessincronizamos as funções do `<stdio.h>` (`printf`/`scanf`) com as do C++ (`cin`/`cout`/`cerr`). Isso significa que você **não pode misturar** funções da `<stdio.h>` com as do C++ no mesmo canal (`scanf`+`cout` é válido, `printf`+`cout` não é).

```
    ios_base::sync_with_stdio(0);
```

7. Implemente sua lógica.
8. O `return 0;` é implícito.

```
}
```

Estrutura básica de um código em C++ (continuado)

Código `estrutura-basica.cpp`

```cpp
#include <bits/stdc++.h>
using namespace std;
using ll = long long;

int main() {
    cin.tie(0);
    ios_base::sync_with_stdio(0);
    // implemente sua lógica
}
```

Compilando C++

Podemos usar tanto o GCC (`g++`) ou o Clang (`clang++`).
`c++` aponta pra um deles (veja `update-alternatives(1)`):

```
$ c++ codigofonte.cpp -O2 -o programa && ./programa
$ c++ codigofonte.cpp -O2 && ./a.out
```

É recomendado compilar com `-O2` para ter resultados mais próximos ao do juiz.

Outros flags úteis

`-g` Permite depuração e melhora o uso do `valgrind`.

`-fsanitize=undefined,address,memory`
No momento da execução, aborta o programa em caso de comportamento indefinido, acesso fora dos limites dos vetores ou acesso a memória inválida (`memory` é exclusivo do Clang).

`-static` Geralmente usado nos juízes, permite que o tempo de execução do executável seja mais consistente.

Saída em C++

Para escrevermos coisas na saída padrão, usamos o Console Output (`cout`) com << (a informação é "direcionada" para o `cout`).
Sempre use `"\n"`, o `endl` faz *flush* que deixa tudo mais lento (mas é necessário nos raros problemas interativos).

Código `saida.cpp`

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

int main() {
    cin.tie(0); ios_base::sync_with_stdio(0);
    cout << "pi=" << fixed << setprecision(15) << acos(-1);
    cout << "\nn=" << setw(3) << setfill('0') << 7;
    cout << " h=" << uppercase << hex << 51966;
                cout << " ?=" << (3 < 2) << "\n";
}
```

Entrada	Saída
	pi=3.141592653589793 n=007 h=CAFE ?=0

Entrada em C++

Pra ler coisas da entrada padrão, usamos o Console Input (`cin`) com >> (a informação é "direcionada" para nossa variável). Observe que o **tipo da variável** muda o jeito como é lido.

Código `entrada.cpp`

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

int main() {
    cin.tie(0); ios_base::sync_with_stdio(0);
    int x; char c; string line;
    cin >> x >> c >> ws;
    getline(cin, line);
    cout << "x=" << x << " c=" << c << "\n" << line << "\n";
}
```

Entrada	Saída
1 a Hello World! You won't see this.	x=1 c=a Hello World!

Os espaços da entrada

Por padrão, o `cin` itera sobre a entrada até encontrar um caractere não espaço. Depois, a coisa que se quer ler começa a ser lida. Assim que o que for lido não for mais válido (por exemplo foi encontrado um espaço), a leitura termina e o próximo que ler terá que lidar com esse resíduo.
Esse comportamento pode ser alterado com `noskipws`.

```
int x, y; char a, b; string l, m;
cin >> x >> a >> b >> y;
getline(cin, l);
cin >> m;
```

Entrada
␣␣43a␣␣␣b56␣␣␣↵
␣␣␣␣␣␣123␣␣456↵

a	b	x	y	l	m
'a'	'b'	43	56	" "	"123"

Lendo até EOF

Muitas vezes é necessário ler até o "fim do arquivo". Para ajudar nisso, o objeto `cin` pode ser implicitamente convertido em um booleano que indica se a entrada ainda é válida.

Código `eof.cpp`

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

int main() {
    cin.tie(0); ios_base::sync_with_stdio(0);
    int x;
    while (cin >> x) { cout << x << "."; }
    cout << "\n";
}
```

Entrada	Saída
1 2 3 4 5 6 7 8	1.2.3.4.5.6.7.8.

Problemas e Juízes

Hora de resolver problemas! Como são esses problemas?

Os problemas são em geral tem várias partes:

Limites de Execução A sua solução deve obdecer limites de memória e de tempo para cada teste;

Enunciado Uma historinha que serve só para gastar o seu tempo misturada com o problema computacional que você tem que resolver;

Entrada Formato da entrada que será dada ao seu programa;

Saída O que o seu programa deverá imprimir;

Exemplos Um ou mais exemplos de entrada/saída. Em inglês geralmente são chamados de *sample tests*.

Sabendo do problema, você deverá codificar uma solução e submeter o código-fonte para o juiz que irá utilizar várias entradas diferentes (respeitando os limites) para julgar o seu código e comparar as suas saídas com as saídas esperadas (note que você pode imprimir a qualquer momento, não precisa terminar acabar a entrada).

O veredito do juiz

AC – Accepted Seu programa passa em todos os casos de teste;

WA – Wrong Answer Seu programa entregou uma resposta incorreta em pelo menos um dos casos de teste;

TLE – Time Limit Exceeded Seu programa demorou para responder um caso de teste dentro do limite de tempo que o problema aceita;

MLE – Memory Limit Exceeded Seu programa excedeu o limite de memória em algum caso de teste;

CE – Compilation Error Seu programa deu erro na hora da compilação, lembrando que *warnings* não são erros;

RE – Runtime Error Seu programa falhou durante a execução. Isso significa um acesso de memória inválida, uma divisão por zero ou qualquer saída não zero da função principal.

Problema: Números Parceiros

Ariel aprendeu na aula de matemática que existem números pares e ímpares. Os números pares não eram desconhecidos para Ariel, já que era comum ter que dividir tudo em casa com seu irmão. Os dois juntavam seus carrinhos que eles recebiam de aniversário e os dividiam, e se a divisão fosse igual para os dois, Ariel declarava que eles tinham "números parceiros" de carrinhos. Escreva um programa que determine se dois números são parceiros como Ariel definiu.

Problema: Números Parceiros

Entrada

A primeira linha da entrada contém um número inteiro T ($1 \leq T \leq 10^3$), o número de casos de teste. Seguem T casos de teste, sendo que cada caso de teste é uma linha que contém dois números inteiros X e Y ($0 \leq X, Y \leq 2 \cdot 10^9$) separados por um espaço em branco.

Saída

Imprima `PARCEIROS` se X e Y forem parceiros e `NAO PARCEIROS` caso contrário. A saída deve ser encerrada por um caractere de quebra de linha (`'\n'`).

Exemplos

Entrada	Saída
2 4 5 42 42	NAO PARCEIROS PARCEIROS

Teste, teste, teste!

Toda tentativa que você submete que não resolve o problema gera uma penalidade na competição real que é aplicada quando você resolver o problema. É importante então testar antes de enviar. Os casos de teste de exemplo podem ser obtidos:

- Copiando o texto do documento dos problemas que geralmente está disponível no juiz
- Copiando do site (Codeforces e Juisto tem botões próprios)
- Copiando no olho, do papel ou de um documento

Testando sua solução usando diff

Você pode testar a sua solução usando a ferramenta `diff`:

```
$ c++ parceiros.cpp && ./a.out <exemplo-1.in \
  | diff - exemplo-1.out
1c1
< PARCEROS
--
> PARCEIROS
```

Assim, você pode descobrir erros de digitação, falta de espaços que a saída exige, ou simplesmente fazer uma avaliação simples sem pensar muito.

Os casos de borda

- Os testes de exemplo são quase sempre superficiais. Eles geralmente não incluem os chamados casos de borda (*edge cases*) que incluem as piores entradas possíveis. Se existe um limite para uma variável, tenha certeza que ele será testado.
- **Crie os seus próprios testes** durante a competição, explorando os limites das variáveis.
- Em problemas de otimização onde existe uma solução usando força bruta, você pode comparar a sua solução lenta com a sua solução rápida. Veremos isto em outra aula, isso se chama *stress testing*.

Solução do problema Números Parceiros?

Código parceiros.cpp

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

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

    while (t--) {
        int x, y;
        cin >> x >> y;
        if ((x+y) % 2 == 1)
            cout << "NAO PARCEIROS\n";
        else
            cout << "PARCEIROS\n";
    }
}
```

Limites dos inteiros

Sinal	Tipo	Bits	Mínimo	Máximo	Dígitos
$+/-$	`char`	8	-128	127	2
$+$	`char`	8	0	255	2
$+/-$	`short`	16	-32768	32 767	4
$+$	`short`	16	0	65 535	4
$+/-$	`int`/`long`	32	$\approx -2 \cdot 10^9$	$\approx 2 \cdot 10^9$	9
$+$	`int`/`long`	32	0	$\approx 4 \cdot 10^9$	9
$+/-$	`long long`	64	$\approx -9 \cdot 10^{18}$	$\approx 9 \cdot 10^{18}$	18
$+$	`long long`	64	0	$\approx 18 \cdot 10^{18}$	19

Solução do problema Números Parceiros

Código parceiros-2.cpp

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

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

    while (t--) {
        ll x, y;
        cin >> x >> y;
        if ((x+y) % 2 == 1)
            cout << "NAO PARCEIROS\n";
        else
            cout << "PARCEIROS\n";
    }
}
```

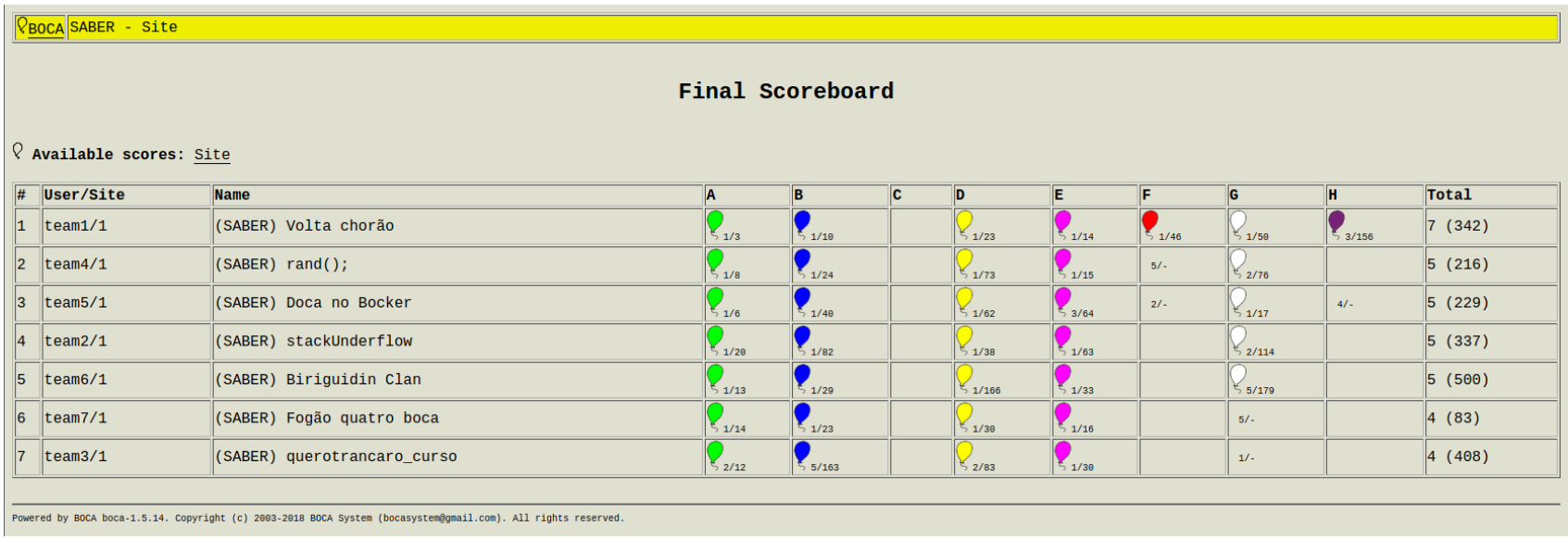
Esse é um jeito, tem outros?

- E se a soma não coubesse em um `unsigned long long`?
  (por exemplo $0 \leq X, Y \leq 2 \cdot 10^{19}$)
- E se os inteiros não coubessem em um
  `unsigned long long`?
  (por exemplo $0 \leq X, Y \leq 10^{100}$)

Porém, lembre-se sempre que o importante é resolver o problema. Se a solução que você propôs resolve o problema nos limites dados, ela é suficiente. A otimização prematura é a origem de todo o mal.

Resolvi o problema, e agora? Eu ganho alguma coisa?

Na Maratona da SBC, o juiz utilizado é o BOCA. Nele, todos os competidores podem ver o placar:



BOCA SABER - Site

Final Scoreboard

Available scores: Site

#	User/Site	Name	A	B	C	D	E	F	G	H	Total
1	team1/1	(SABER) Volta chorão	1/3	1/10		1/23	1/14	1/46	1/50	3/156	7 (342)
2	team4/1	(SABER) rand();	1/8	1/24		1/73	1/15	5/-	2/76		5 (216)
3	team5/1	(SABER) Doca no Bocker	1/6	1/40		1/62	3/64	2/-	1/17	4/-	5 (229)
4	team2/1	(SABER) stackUnderflow	1/20	1/82		1/38	1/63		2/114		5 (337)
5	team6/1	(SABER) Biriguidin Clan	1/13	1/29		1/166	1/33		5/179		5 (500)
6	team7/1	(SABER) Fogão quatro boca	1/14	1/23		1/30	1/16		5/-		4 (83)
7	team3/1	(SABER) querotrancaro_curso	2/12	5/163		2/83	1/30		1/-		4 (408)

Em cada coluna temos um problema da prova: A, B, C, etc. Quem resolveu aquele problema ganha balãozinho (inclusive fisicamente), e uma pontuação que pode ser vista na direita.

A pontuação

A	B	C	D	E	F	G	H	Total
1/3	1/10		1/23	1/14	1/46	1/50	3/156	7 (342)
1/8	1/24		1/73	1/15	5/-	2/76		5 (216)
1/6	1/40		1/62	3/64	2/-	1/17	4/-	5 (229)

Para cada problema, temos no BOCA as seguintes informações:

Submissões feitas/Tempo em minutos da 1ł submissão aceita

E a pontuação total de cada equipe é dada por:

Quantidade de problemas resolvidos (penalidade)

Sendo que a penalidade é contada apenas para os problemas resolvidos, e é dada pela soma dos tempos de envio da primeira submissão aceita + 20 minutos por cada submissão errada.
Como a ordenação é feita primeiro pela quantidade de problemas resolvidos, não há penalidade em tentar resolver um problema que você não tem certeza da solução.

Truques do placar

Nem sempre você vai ser o primeiro a resolver um problema. E geralmente as provas não tem problemas ordenados pela ordem de dificuldade. Então você pode usar o placar pra encontrar o próximo problema mais fácil ou pra entender a dificuldade da prova.

A	B	C	D	E	F	G	H	Total
1/3	1/10		1/23	1/14	1/46	1/50	3/156	7 (342)
1/8	1/24		1/73	1/15	5/-	2/76		5 (216)
1/6	1/40		1/62	3/64	2/-	1/17	4/-	5 (229)
1/20	1/82		1/38	1/63		2/114		5 (337)
1/13	1/29		1/166	1/33		5/179		5 (500)
1/14	1/23		1/30	1/16		5/-		4 (83)
2/12	5/163		2/83	1/30		1/-		4 (408)

- A é o problema mais fácil, seguido do E, do D e do B.
- G tem alguma dificuldade extra.
- F e H são problemas difíceis.
- C é o problema mais difícil e não foi resolvido por ninguém.

Lendo o placar no Codeforces

O placar do Codeforces no formato ACM-ICPC é assim:

2020-2021 ACM-ICPC Latin American Regional Programming Contest

Final standings

Double click (or ctrl+click) each entry to view its submission history

Standings

#	Who	=	Penalty	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	Kobor 53: w0nsh, kobor, Anadi	12	1135	+2 01:46	+5 02:09	+ 00:42	+ 00:16	+ 00:10	+3 01:29		+ 01:04	-7	+ 02:30	+1 00:47	+1 00:32	+ 03:20	+ 00:10
1	Jebać pis: xman1024, Proszek_na_ludka	12	1135	+ 02:20	+1 00:27	+ 00:49	+ 00:09	+ 00:45	+4 03:54	+ 02:43	+ 00:58		+1 03:03	+ 00:59	+ 00:41		+ 00:07
3	noimi	12	1194	+4 03:54	+1 01:50	+1 00:39	+ 00:05	+ 00:20		+ 00:57	+3 01:23		+2 02:29	+ 01:02	+ 00:27	+ 02:58	+ 00:10
4	UCF Kamino: Xylenox, SecondThread, Harpae	12	1298	+1 03:04	+1 01:38	+ 00:35	+ 00:10	+2 01:00	-5	+ 03:35	+ 00:20	+4 03:41	+ 02:33	+ 01:26	+ 00:47		+ 00:09
5	dario2994	12	1304	+1 03:18	+ 02:41	+ 00:43	+ 00:13	+ 01:36	+ 01:16	+ 03:56	+ 01:50		+1 02:21	+ 02:03	+1 00:30		+ 00:17
6	SSRS_	11	717	+ 02:37	+ 01:45	+ 00:25	+ 00:07	+ 00:42	-4	+ 01:25	+1 00:58		+ 02:10	+ 00:47	+1 00:17		+ 00:04
7	Masarnian 1: jacynkaa, ggawryal, krzysiek27	11	895	+1 02:35	+ 01:03	+ 00:39	+ 00:16	+ 00:25	-12	+ 03:06	+ 01:31		+1 01:47	+ 00:58	+2 01:09		+ 00:06
8	Uns aposentados: tfg, Dranoel321, Nson	11	910	-2	-4	+ 00:33	+ 00:10	+ 01:14	+1 00:42	+ 01:50	+2 01:13	+ 03:50	+ 01:50	+ 01:35	+1 00:43		+ 00:10
9	Radiant: upobir, Anachor, solaimanope	11	916	+ 00:42	+1 01:50	+ 01:00	+ 00:16	+ 01:04	-2		+ 01:41	+2 03:32	+ 02:47	+ 00:38	+ 00:36		+ 00:10
10	UIUC - Must Pass: Suzukaze, yhchang3	11	1076		+3 02:42	+ 00:24	+ 00:12	+1 01:24	+ 02:21	+ 03:28	+ 00:51		+ 03:10	+ 01:36	+ 00:23		+ 00:05
11	Kal se Gym Jayenge: BhaskarTM, TheOneYouWant, Shivam_18	11	1086		+1 01:46	+2 00:53	+ 00:05	+ 00:47	+1 03:27	+ 03:38	+ 01:38	-4	+ 02:17	+ 01:16	+ 00:54		+ 00:05
12	ffao	11	1108		+ 01:49	+ 00:48	+ 00:13	+ 01:02	+ 03:42	+ 03:24	+1 02:52		+ 02:14	+ 01:20	+ 00:37		+ 00:07

+/- Submissões erradas

Lendo o placar no Juisto

#	Usuário	=	Penalidade	A	B	C	D	E	F	G	H	I	J
1	fulano	5	89	+ 00:02		+ 00:05	+ 00:15	+ 01:00		+ 00:07		–2	
1	beltrano	5	102	+ 00:03		+ 00:08	+1 00:12	+1 00:33		+ 00:06			
1	sicrano	4	41	+ 00:05		+ 00:09	+ 00:11	–2		+ 00:16			
2	zutano	3	122	+ 00:04		+1 00:11	–1			+3 00:27			
2	citano	2	98	+ 00:08				–1		+3 00:30			
2	caio	1	27	+1 00:07						–2			
2	ticio	1	55	+2 00:15									
3	mevio	0	0	–1									

+/- Submissões erradas

Complexidade e a STL

Revisão de análise (Alg II)

- **Objetivo:** analisar o comportamento de crescimento de uma função
- Qual cresce mais - $n!$ ou n^2?
- Podemos determinar a função de custo de um algoritmo (em relação ao tamanho da entrada) e saber seu desempenho a partir dela
- Associamos uma classe à função de custo

Revisão de $\mathcal{O}$

- Sejam $f, g : \mathbb{Z} \to \mathbb{Z}$
- $g(n)$ é dita $\mathcal{O}(f(n))$ ou diz-se que $g(n) = \mathcal{O}(f(n))$ se existem $n_0, k \in \mathbb{Z}$ tais que $g(n) \leq kf(n)\ \forall\ n > n_0$
- f é um teto para g a partir de certo ponto; isto é: pior do que f, g com certeza não fica (a menos de uma constante)

Análise para Maratona

Geralmente não é complicada, ou é complicada mas conseguimos "roubar" a resposta por intuição (assumindo n como tamanho da entrada):

- Três `for` aninhados: $\mathcal{O}(n^3)$
- Função recursiva que se chama em x pedaços da entrada: $\mathcal{O}(\log_x n)$
- Operação $\mathcal{O}(1)$ quase sempre, e $\mathcal{O}(n)$ em pontos ocasionais: $\mathcal{O}(n)$
- Testar todas as permutações: $\mathcal{O}(n!)$

Só precisamos saber se o algoritmo a ser utilizado respeita o **limite de tempo** do problema.

Limite de tempo, é?

- Um dos principais gargalos em resolver problemas é justamente resolvê-los de uma forma que seja eficiente no uso do tempo.
- Devemos então prestar atenção nas complexidades exigidas pelas estruturas de dados que utilizamos. Você tem que sempre ter em mente por exemplo que uma operação de deleção de uma posição arbitrária em um vetor é $\mathcal{O}(n)$.
- Podemos estimar se um algoritmo vai passar ou não usando a regra de que um computador realiza por volta de 10^8 operações por segundo.

Estimativa de complexidade

n	Algoritmo que passa	Comentário
$\leq [10..11]$	$\mathcal{O}(n!)$, $\mathcal{O}(n^6)$	ex. Enumerar permutações
$\leq [15..18]$	$\mathcal{O}(2^n n^2)$	ex. PD do Caixeiro-Viajante
$\leq [18..22]$	$\mathcal{O}(2^n n)$	ex. PD com técnica de máscara de bits
≤ 100	$\mathcal{O}(n^4)$	ex. PD com 3 dimensões + laço $O(n)$, ${}_nC_{k=4}$
≤ 500	$\mathcal{O}(n^3)$	ex. Floyd-Warshall
$\leq 2 \cdot 10^3$	$\mathcal{O}(n^2 \lg n)$	ex. 2 laços aninhados + consulta em árvore
$\leq 5 \cdot 10^4$	$\mathcal{O}(n^2)$	ex. Bubble/Selection/Insertion Sort
$\leq 10^5$	$\mathcal{O}(n \lg^2 n)$	ex. Construção de vetor de sufixos padrão
$\leq 10^6$	$\mathcal{O}(n \lg n)$	ex. Merge Sort, construir árvore de segmento
$\leq 10^7$	$\mathcal{O}(n \lg \lg n)$	ex. Crivo de Eratóstenes, função totiente
$\leq 10^8$	$\mathcal{O}(n)$, $\mathcal{O}(\lg n)$, $\mathcal{O}(1)$	ex. Solução matemática. Maioria dos problemas tem $n \leq 10^9$ (gargalo de E/S)

Adaptado de 2013, Competitive Programming 3. Steven Halim, Felix Halim

Como a STL pode nos ajudar?

- STL significa *Standard Template Library*. O *template* é uma funcionalidade avançada do C++ que permite que os tipos de dados sejam genéricos.
- Isto significa por exemplo que você pode criar ordenar qualquer estrutura de dados, inclusive as que você inventar, desde que você defina o critério de comparação.
- Veremos a seguir algumas estruturas e algoritmos da STL que nos economizam o tempo de escrever uma heap ou uma árvore balanceada na hora da competição. Elas gerem a alocação dinâmica de memória automaticamente, nada de `new`, `malloc`, `free` ou `realloc`!
- Não é necessário você entender absolutamente tudo agora, veremos elas extensivamente.

Tipos genéricos?

Digamos que você queira implementar de forma genérica, uma função que retorna se uma coisa é menor do que a outra. Em C++, você pode se expressar assim:

Código `generics.cpp`

```
#include <bits/stdc++.h>
using namespace std;
template <class T> void lt (T u, T v) {
    cout << u << ((u < v) ? " < " : " >= ") << v;
    cout << " r=" << (u < v) << " ";
}
int main() {
    lt('a', 'b'); lt('a', 'a'); cout << "\n";
    lt(1, 2); lt(2, 1); cout << "\n";
}
```

Entrada	Saída
	`a < b r=1 a >= a r=0` `1 < 2 r=1 2 >= 1 r=0`

Funções em variáveis?

Para nossa sorte, isso já existe na biblioteca padrão: Podemos instanciar uma função `less<T>()` ou `greater<T>()` em um tipo específico e armazenar essa função em uma variável.

Código `preds.cpp`

```
#include <bits/stdc++.h>
using namespace std;
int main() {
    function<bool(int, int)> lt = less<int>();
    function<bool(int, int)> gt = greater<int>();
    cout << lt(1, 2) << " " << gt(1, 2) << "\n";
    cout << lt(1, 1) << " " << gt(1, 1) << "\n";
    cout << lt(2, 1) << " " << gt(2, 1) << "\n";
}
```

Entrada	Saída
	1 0 0 0 0 1

Eu quero minha própria função!

É muito útil passar como argumento uma função customizada por exemplo para fazer uma ordenação, então foi criado o lambda, uma sintaxe mais simples pra uma função em variável. E se estiver com preguiça, use `auto` pra adivinhar o tipo!

Código `lambda.cpp`

```
#include <bits/stdc++.h>
using namespace std;
void externo(int x) { cout << x << " z=?\n"; }
int main() {
    int z = 0;
    auto cop = [=](int x) { cout << x << " z=" << z; };
    auto ref = [&](int x) { cout << x << " z=" << z; };
    auto pon = externo;
    z++;
    cop(1); cout << " "; ref(2); cout << " "; pon(3);
}
```

Entrada	Saída
	1 z=0 2 z=1 3 z=?

Qual que é essa do &?

O & simboliza no C++, além do significado usual de obter um endereço de memória, o próprio conceito de referência. No C precisávamos brigar com ponteiros pra fazer a mesma coisa.

Código `ref.cpp`

```
#include <bits/stdc++.h>
using namespace std;
void coloca_5(int& x) { x = 5; }
int main() {
    int x = 1;
    int& y = x;
    int& z = y;
    y = 2;
    cout << x << " " << y << " " << z << ", ";
    coloca_5(z);
    cout << x << " " << y << " " << z << "\n";
}
```

Entrada	Saída
	2 2 2, 5 5 5

Algumas utilidades

- `swap(T& u, T& v)`: Troca o valor de `u` e `v` (lembra Alg I?).
- `abs(T u)`: Valor absoluto dado por $|u|$.
- `max(T u, T v)`: Máximo entre dois elementos.
- `max({ T u, T v, ... })`: Máximo entre vários elementos.
- `min(T u, T v)`: Mínimo entre dois elementos.
- `min({ T u, T v, ... })`: Mínimo entre vários elementos.

O clássico vetor (`vector`)

O `vector<T>` é um vetor: Uma sequência contígua de memória alocada na *heap* do tipo `T` especificado que aumenta de tamanho sozinha, podendo ser endereçada por índices aleatórios em $\mathcal{O}(1)$.

- `vector<T> v (int n = 0, T val = {})` [$\mathcal{O}(n)$]: Cria um `vector` do tipo `T` com `n` elementos inicializados em `val`.
- `v.resize(int n, T val = {})` [$\mathcal{O}(n)$]: Muda o tamanho do vetor preenchendo com `val` se `n` for maior que o atual.
- `v.size()` [$\mathcal{O}(1)$]: Retorna o tamanho (cuidado: `unsigned`!).
- `v[size_t i]` [$\mathcal{O}(1)$]: Acessa o índice i.
- `v.front()`, `v.back()` [$\mathcal{O}(1)$]: Acessa o primeiro, último.
- `v.push_back(T x)` [$\mathcal{O}(1)$]: Adiciona um valor ao final.
- `v.pop_back()` [$\mathcal{O}(1)$]: Remove o elemento do final.
- `v.clear()` [$\mathcal{O}(1)$]: Remove todos os elementos.
- `v.empty()` [$\mathcal{O}(1)$]: Está vazio?

Exemplo do vector

Código vector.cpp

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

vector<vector<int>> matriz (15, vector<int>(15, 5));
vector<bool> b (15); // ocupa 15 bits já inicializados

int main() {
    vector<int> xs (3, 5);
    for (auto& x : xs) { cout << x << ' '; x = 3; }
    xs.resize(5, 1); xs.push_back(10); cout << "\n";
    for (int i = 0; i < xs.size(); i++)
        cout << xs[i] << ' ';
    xs.clear(); cout << " z=" << xs.size() << "\n";
}
```

Entrada	Saída
	5 5 5 3 3 3 1 1 10 z=0

Iteradores do `vector`

Faltaram algumas operações interessantes para o `vector`:

- `v.insert(iterator it, int val)` [$\mathcal{O}(n)$]: Insere após o iterador `it` o valor `val`.
- `v.erase(iterator it)` [$\mathcal{O}(n)$]: Remove o elemento apontado pelo iterador `it`.
- `v.erase(iterator first, iterator last)` [$\mathcal{O}(n)$]: Remove os elementos dados pelo intervalo [*first*, *last*).

Um iterador é uma abstração do famoso ponteiro. Fazer `*it` por exemplo te dá o valor apontado pelo iterador `it`. Você obtém iteradores do `vector` da seguinte maneira:

- `v.begin()` [$\mathcal{O}(1)$]: Obtem um iterador que aponta para `v[0]`.
- `v.end()` [$\mathcal{O}(1)$]: Obtem um iterador que aponta para `v[n]` (sim, fora do `vector`, não acesse!).

Os iteradores do `vector` são somáveis como os ponteiros.

Exemplo de iteradores do vector

Código vector-it.cpp

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

int main() {
    vector<int> xs;
    for (int i = 0; i < 10; i++) xs.push_back(i);
    xs.erase(xs.begin() + 2);
    for (auto x : xs) { cout << x << ' '; } cout << "\n";
    xs.insert(xs.begin() + 4, -1);
    xs.erase(xs.begin(), xs.begin()+3);
    for (auto it = xs.begin(); it != xs.end(); it++)
        cout << *it << ' ';
    cout << "\n";
}
```

Entrada	Saída
	0 1 3 4 5 6 7 8 9 4 -1 5 6 7 8 9

Cadeias de caracteres! (`string`)

Chega de `char*`. Precisamos de strings, sem precisar ficar adivinhando o tamanho delas em $\mathcal{O}(n)$, e que podem crescer automaticamente e serem concatenadas com +. Elas tem todas as funções de um `vector<char>` (inclusive iteradores), e mais:

- `string s = "Hel"` $[\mathcal{O}(n)]$: Constrói string de uma em C.
- `s += "lo"` $[\mathcal{O}(n)]$: Concatena outra string.
- `s.find(string str, size_t pos = 0)` $[\mathcal{O}(nm)]$: Busca uma string começando do `pos`, retornando a posição ou `string::npos` se não achar.
- `s.substr(size_t pos, size_t len = string::npos)` $[\mathcal{O}(n)]$: Constrói uma substring começando em `pos` com tamanho `len`.
- `s.c_str()` $[\mathcal{O}(1)]$: String em C (termina em `'\0'`).
- `s.copy(char* s, size_t len, size_t pos = 0)` $[\mathcal{O}(n)]$: Copia uma substring para um buffer `char*`.

Algumas utilidades de sequência

- `fill(iterator begin, end, T val)` [$\mathcal{O}(n)$]: Preenche [*begin*, *end*) com `val`
- `find(iterator begin, end, T val)` [$\mathcal{O}(n)$]: Retorna o iterador do primeiro elemento igual a `val` (ou `end` se não achar).
- `count(iterator begin, end, T val)` [$\mathcal{O}(n)$]: Retorna a contagem de elementos iguais a `val`.
- `search(iterator begin1, end1, begin2, end2)` [$\mathcal{O}(nm)$]: Retorna a primeira ocorrência de [*begin*2, *end*2) em [*begin*1, *end*1).
- `copy(iterator begin, end, tobegin)` [$\mathcal{O}(n)$]: Copia [*begin*, *end*) em [*tobegin*,.

Entre outros que são menos utilizados.

E isso é tudo!

- O conteúdo que vocês precisam para fazer a competição que logo começará está todo aqui.
- Não é necessário você entender absolutamente tudo agora, com o tempo você ficará mais habituado com a STL e ainda veremos mais conteúdo sobre a STL.
- Recomendamos o cppreference como referência (`https://en.cppreference.com/`), pois está bem atualizada. A referência do CPlusPlus (`https://www.cplusplus.com/reference/`) é eventualmente útil mas não tem as últimas funcionalidades.
- Bons estudos!