Biblioteca STL aplicada à Maratona de Programação

MEDITEC 5

29 de Maio de 2014

Stardard Template **Library**

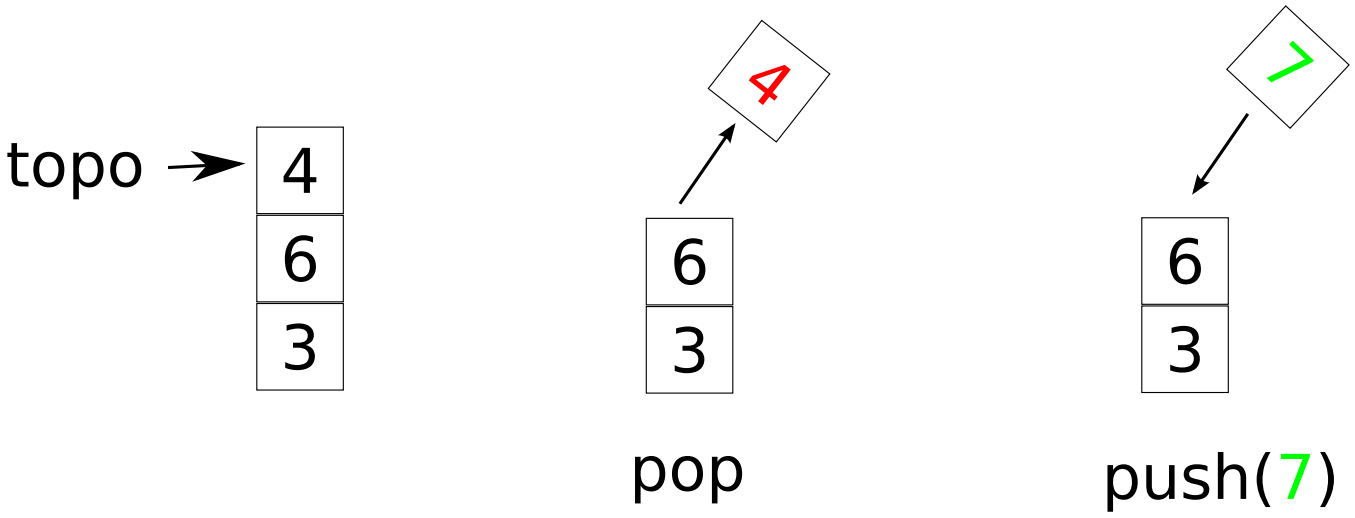
- *Biblioteca* pronta de $C++$
- Várias funções são usadas na maratona, mas nem todas são frequentes
- Facilita a **implementação**, não a criação de algoritmos!
- *www.cppreference.com*

Em C++

```
#include <cstdio>
#include <vector>
#include <stack>

using namespace std;

int main() {

    while(...) {
        ...
    }
    return 0;
}
```

Pilha

- Last In, First Out (LIFO)
- Operações *push* e *pop*

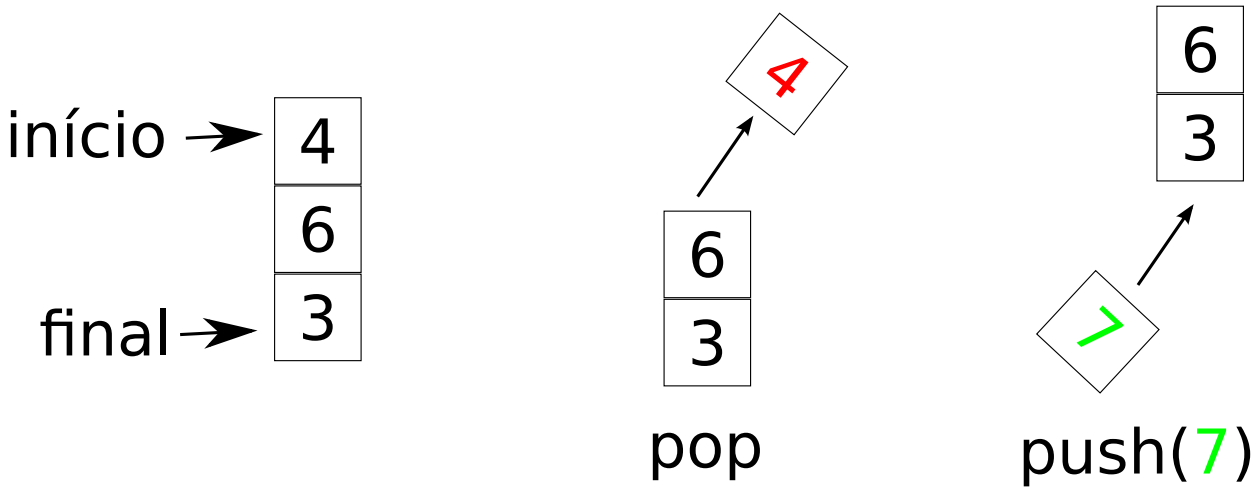


Uso

- `#include <stack>`
- `stack<TIPO> S;`
  - TIPO = int, char, double, pair<int,int>, vector<float>, etc.
- `S.push(ELEMENTO);` Empilha ELEMENTO na pilha S
- `S.top();` Retorna o elemento no topo da pilha.
  - Acessar o topo da pilha *não* o remove!
- `S.pop();` Desempilha o topo da pilha.
- `S.empty();` Verifica se a pilha está vazia.
  - `while (!S.empty()) S.pop();`

Fila

- First In, First Out (FIFO)
- Operações *push* e *pop*



Manipulação de filas

- `#include <queue>`
- `queue<TIPO> Q;`
  - TIPO = int, char, float, double, pair<int,int>, vector<float>, etc.
- `Q.push(ELEMENTO);` Enfileira ELEMENTO no final da fila Q.
- `Q.front();` Retorna o elemento no início da fila.
  - Acessar o início da fila *não* o remove!
- `Q.pop();` Desenfileira o início da fila;
- `Q.empty();` Verifica se a fila está vazia.
  - `while (!Q.empty()) Q.pop();`

Par

```
typedef struct {
    TIPO1 first;
    TIPO2 second;
} TPAR;
typedef pair<TIPO1,TIPO2> TPAR;

TPAR variavel;
variavel.first = A;
variavel.second = B;
```

ou

```
TPAR variavel = TPAR(A,B);
```

Exemplo

```
typedef pair<int, int> ii;
ii var = ii(2,3);
printf("%d\n",var.first); // imprime 2
var.second++;
printf("%d\n",var.second); // imprime 4
```

Encapsulamento

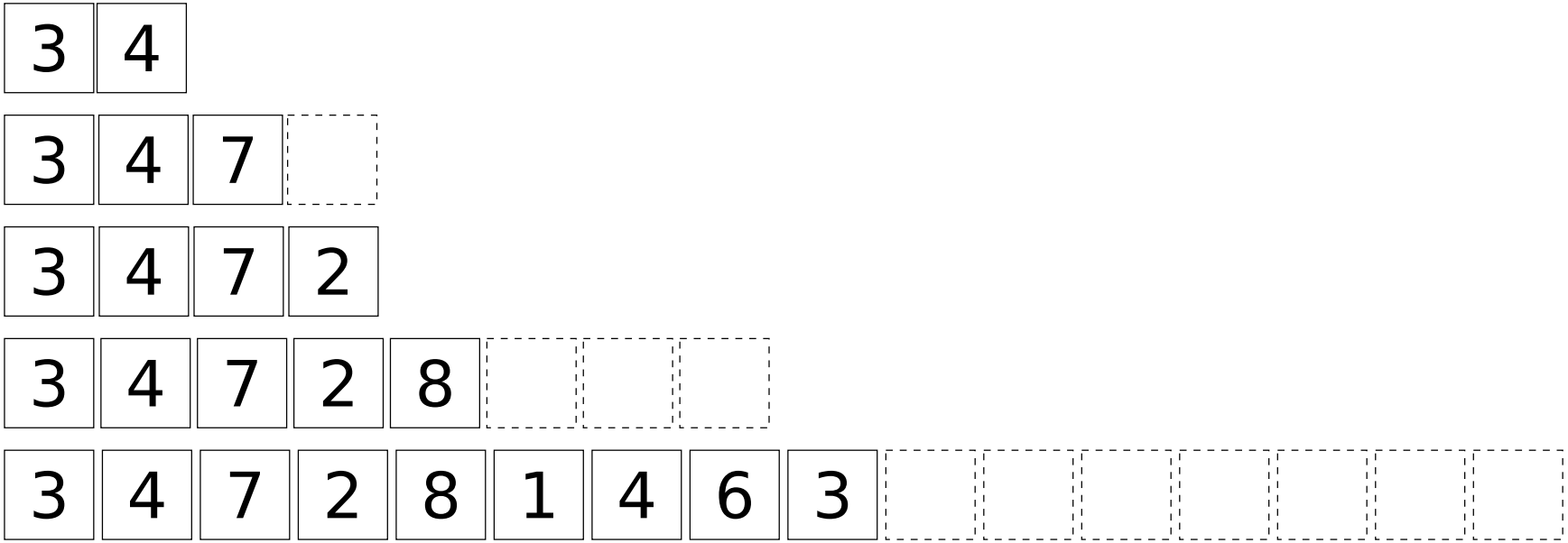
```
typedef pair<int,int> ii;
typedef pair<ii, char> i2c;

i2c var = i2c(ii(42,7), 'x');
printf("%d\n", var.first.first); // imprime 42
printf("%d\n", var.first.second); // imprime 7
printf("%c\n", var.second); // imprime 'x'
```

Alocação Dinâmica

- ~~TIPO *v = (TIPO *)malloc(N*sizeof(TIPO));~~
- ~~v = (TIPO *)realloc(v, (N+1)*sizeof(TIPO));~~
- ~~free(v);~~

```
#include <vector>
vector<TIPO> v;
v.push_back(ELEMENTO);
v.clear();
```

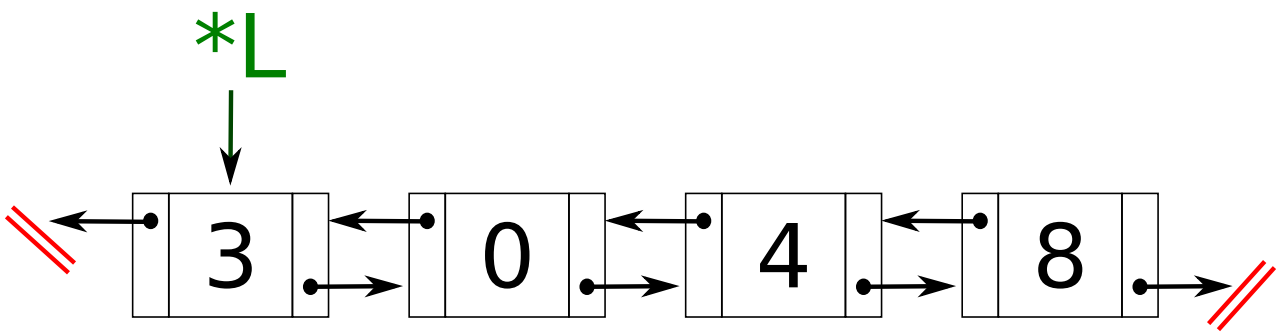
Alocação Logarítimica



Manipulação de vectors

- `#include <vector>`
- `vector<TIPO> v;`
  - TIPO = int, char, float, double, ii, vector<float>, etc.
- `v[i]` Acessa $i-$ésimo elemento de v (0-indexado)
- `v.clear();` Remove todos os elementos do vector v.
- `v.push_back(ELEMENTO);` Insere ELEMENTO no final do vetor v;
- `v.empty();` Verifica se o vector está vazio.
- `(int)v.size();` Retorna o número de elementos de v;
  - Cast para int evita warnings.

Listas (duplamente ligadas)



Declaração

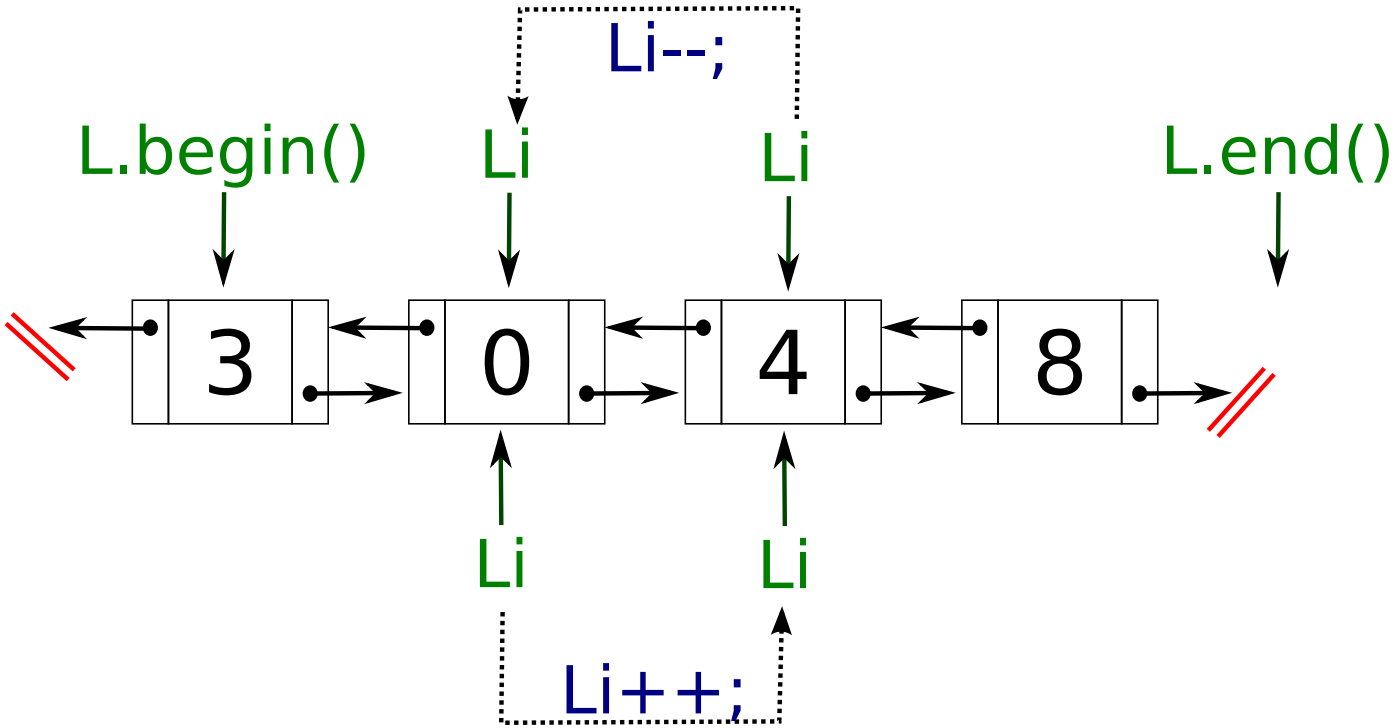
```
typedef struct no_ {
    TIPO data;
    struct no_ *ant, *prox;
} no;
no* L;
```

```
#include <list>
list<TIPO> L;
```

iterator

- "ponteiro" da STL que se comporta como um indexador
- Em C, vetor: `int i; i++; i--;`
- Em C, lista:
  `no *aux; aux = aux->prox; aux = aux->ant;`
- Com STL: `list<int>::iterator Li; Li++; Li--;`

iterators para listas

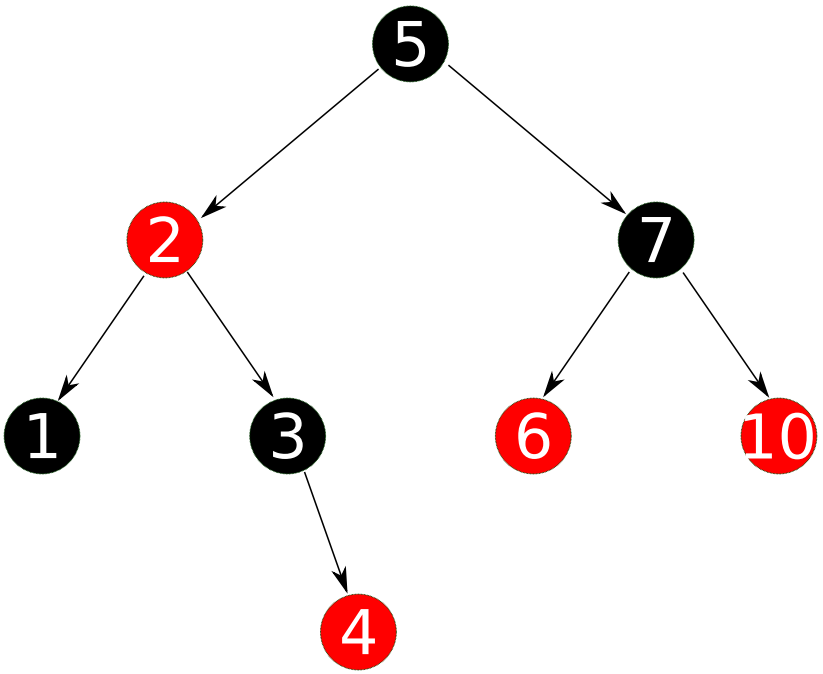


Manipulação de listas

- `list<TIPO>::iterator Li;` Declara um iterator
- `L.begin()`: Retorna um iterator para o primeiro elemento
- `L.end()`: Retorna um iterator para o elemento "depois do últmimo"
  - `Li=L.end(); Li--;` acessa o último elemento da lista
- `Li++;` e `Li--;` Avança e volta o iterator uma posição
- `*Li` Acessa o conteudo apontado pelo iterator
- `L.clear();` Remove todos os elementos da lista
- `L.insert(Li,ELEMENTO);` Insere ELEMENTO **antes** da posição de *Li* e retorna um iterator para elemento inserido.
- `L.erase(Li);` Remove a posição apontada por *Li* e retorna um interator para a posição seguinte da removida.

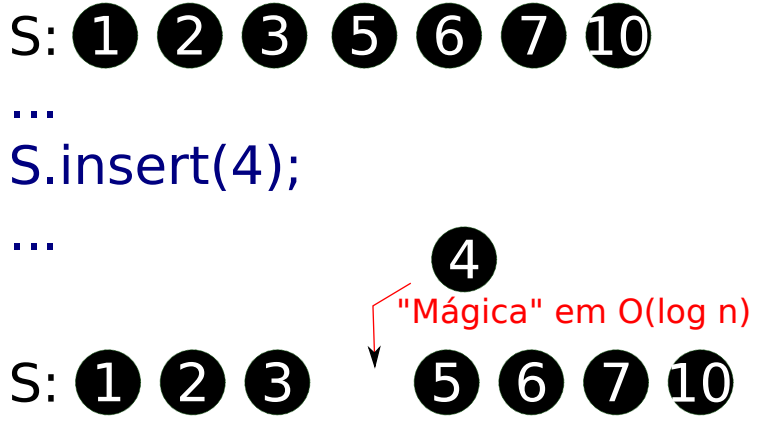
Árvore Rubro-Negra

- Inserção, Remoção e Busca em tempo logaritmico



Armazenamento de Conjuntos

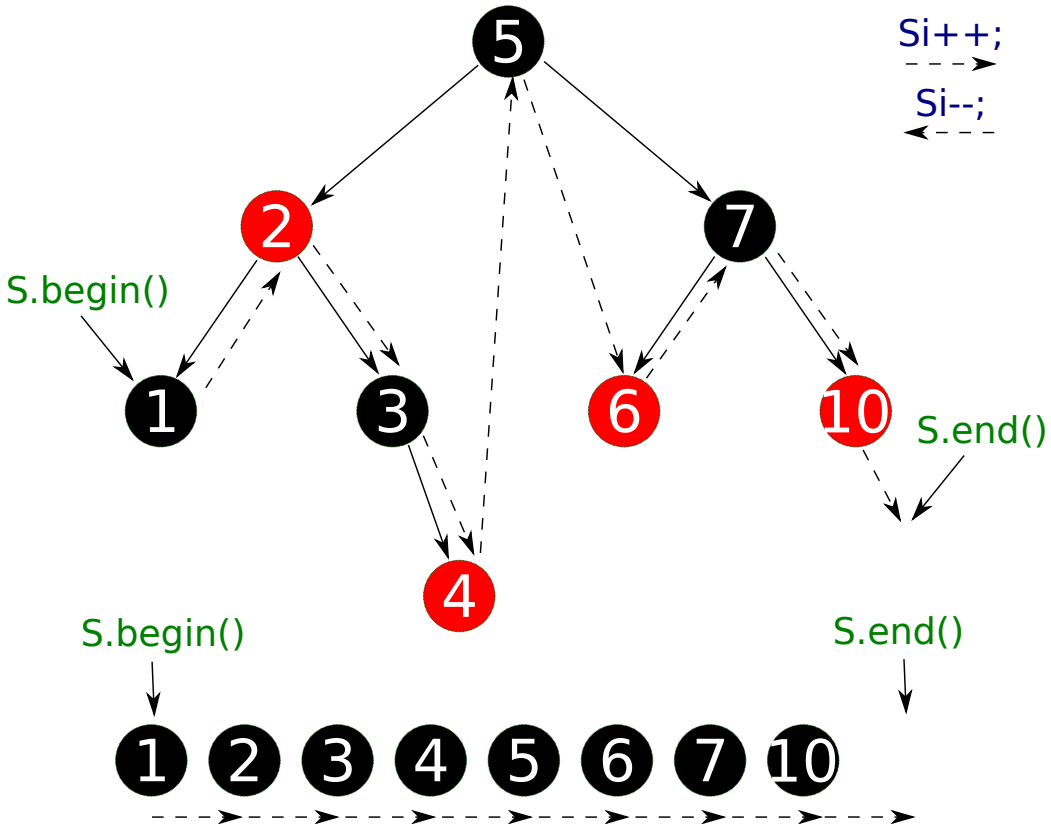
- A árvore pode ser melhor visualizada como a representação de um *conjunto* (*set*) *ordenado* de elementos.




```
#include <set>
set<TIPO> S;
```

Árvores Balanceadas

```
set<int>::iterator Si;
```



Manipulação de sets

- `S.begin()`: Retorna um iterator para o primeiro elemento
- `S.end()`: Retorna um iterator para o elemento "depois do últmimo"
- `Si++`, `Si--` e `*Si` Como na lista
- `S.clear();` Remove todos os elementos do set
- `S.insert(ELEMENTO);` Insere ELEMENTO no set, *se ele não existir no set*.
- `S.erase(ELEMENTO);` Remove ELEMENTO, se existir.
- `S.erase(Si);` Remove o elemento apontado por *Si*.
- `S.find(ELEMENTO);` Se ELEMENTO existe, retorna iterator para ele. Se não, retorna S.end()
  - `if (S.find(X) != S.end())` testa se X está no set.

Outros usos de set

- Um set não contém elementos repetidos; Um *multiset* contém!
  - `#include <set>`
  - `multiset<int> S;`
  - Uso da função *lower_bound()* para busca. Veremos mais adiante.
- Um set pode ser usado como uma *fila de prioridade mínima*
  - `typedef pair<int, int> ii;`
  - `set<ii> S;`
  - `S.insert(ii(PRIORIDADE,ELEMENTO));`
  - `ii prioritario = *S.begin(); S.erase(S.begin());`
  - Usado em uma implementação do algoritmo de Dijkstra.

Strings

- Tipo para string mais robusto que *char* *.
- É um tipo, logo pode ser usado em `vector<string> v`, `set<string> S`, etc.
- `#include <string>`
- `string s1, s2;`
- `if (s1 == s2)` se as strings são iguais
- `if (s1 < s2)` se uma string vem antes da outra alfabeticamente
- `(int)s.size()` retorna o numero de caracteres de s
- `s[i]` acessa o $i-$ésimo caracter de s

Leitura e Impressão

- É mais fácil ler em um *char* * e atribuir a uma string em seguida.
- `char aux[64]; scanf("%s",aux); string s = aux;`
- `s.c_str()` retorna um *char* * para impressão
- `printf("%s\n", s.c_str());`

Mapas

- "vetores" indexados por qualquer tipo ordenável
- *Dicionários* do Python
- Associa qualquer coisa à outra coisa

M:

Chave		**Valor**
"banana"	→	132
"caqui"	→	478
"maca"	→	325
"uva"	→	325

M:

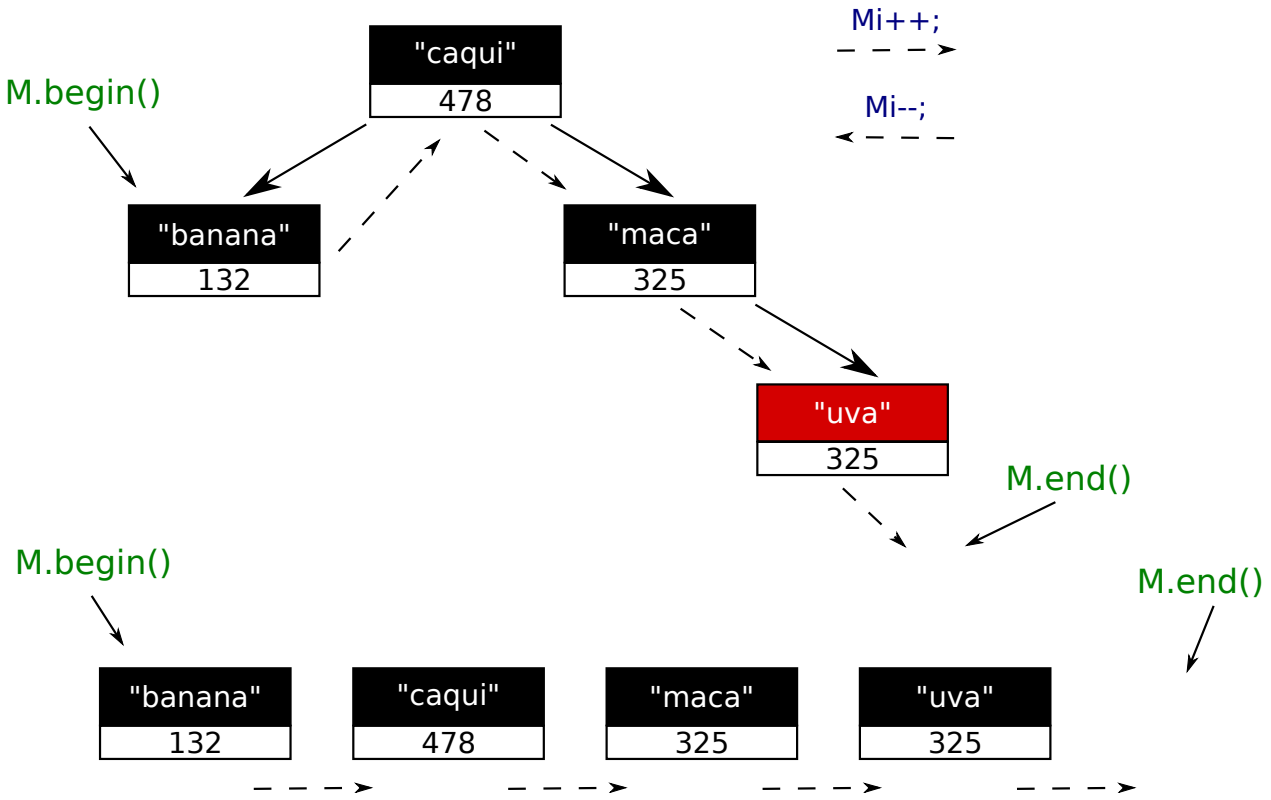
"banana"	"caqui"	"maca"	"uva"
132	478	325	325

Declaração e acesso

- `#include <map>`
- `map<TCHAVE,TVALOR> M;`
  - TCHAVE é o tipo da chave ("índice")
  - TVALOR é o tipo do valor ("conteúdo")
  - `map<string,int> M;` Associa strings a inteiros
- `M[CHAVE]` acessa o valor da CHAVE
  - `M["banana"] = 132;`
  - Acessar um item custa um numero *logarítimo* de comparações...

Mapas

- ... porque `map<TC,TV>` = `set<pair<TC,TV> >`!



Manipulação de mapas

- `M[CHAVE]` acessa o valor da CHAVE
  - `M[CHAVE] = VALOR;` seta o VALOR para a CHAVE
  - Se CHAVE ainda nao existe, cria automaticamente
- `M.begin()`, `M.end()`, `Mi++`, `Mi--` e `*Mi` como no set
  - `Mi->first` e `Mi->second` acessa chave e valor dado pelo iterator
- `M.clear();` Remove todos os elementos do mapa.
- `M.erase(CHAVE);` Remove o elemento com CHAVE do mapa.
- `M.find(CHAVE);` Retorna iterator para o elemento com a CHAVE ou M.end() se nao existe
  - `if (M.find(X) != M.end())` testa se a chave X está no mapa.

Ordenação

- `#include <algorithm>`
- `sort(v.begin(),v.end());` ordena o vector v em ordem crescente em $O(n \lg n)$
- Iterator de `vector<TIPO>` é compatível com `TIPO *` !
- `sort(&v[0], &v[n]);` ordena o vetor $v[]$
  - Identico a `sort(v,v+n);`

Função de comparação

```
bool cmp(int a, int b) {
    return a > b;
}

int main() {
    ...
    sort(v,v+n, cmp);
    ...
}
```

Unicidade

- `unique(v.begin(),v.end()); unique(v,v+n);` remove elementos repetidos do vetor *ordenado* v.
- `[2 4 4 7 9 9 9 10]` → `[2 4 7 9 10]`
- Retorna iterator para o "novo" v.end()
- `n = unique(v,v+n) - v;`

Geração de permutações

- Se *v* contém uma permutação, `next_permutation(v,v+n)` o transforma na próxima permutação e retorna *true*, ou retorna *false* se *v* contém a última permutação.
- `[2 9 1 5]` → `[2 9 5 1]` → `[5 1 2 9]`

```
for (int i=0;i<n;i++) v[i] = i;
do {
    // processa uma permutacao
} while( next_permutation(v,v+n) );
```

Busca binária - binary_search()

- `binary_search(v,v+n,x)` retorna um `bool` igual a *true* se x está no vetor v, *false* caso contrário.
- Com $O(\lg n)$ comparações
- Assume v ordenado em ordem crescente (mas aceita função cmp como sort()). Comportamento indefinido caso contrário.

Busca binária - lower_bound()

- `lower_bound(v,v+n,x);` retorna iterator para o menor elemento **maior ou igual** a x no vetor v
- Também logarítmico
- `int pos = lower_bound(v,v+n,x) - v;`
- Presente em sets e multisets
  - `Si = S.lower_bound(x);`

Busca binária - upper_bound()

- `lower_bound(v,v+n,x);` retorna iterator para o menor elemento **maior ou igual** a x no vetor v
- `upper_bound(v,v+n,x);` retorna iterator para o menor elemento **maior** a x no vetor v
- `int pos = upper_bound(v,v+n,x) - v;`
- Presente em sets e multisets
  - `Si = S.upper_bound(x);`

Outras funções interessantes

- `swap(a,b)` troca as variáveis *a* e *b*;
- `min(a,b)` e `max(a,b)` retornam o menor e maior elemento entre *a* e *b*;
- `reverse(v,v+n)` inverte o vetor *v*;
- Muitas outras!
  - *http://www.cppreference.com*

Pratique!

http://br.spoj.com/problems/JANELA13/
http://br.spoj.com/problems/CAPITA13/
http://br.spoj.com/problems/JASPION/
http://br.spoj.com/problems/EXPRES11/
http://br.spoj.com/problems/TRAPEZ08/
http://br.spoj.com/problems/PIZZA07/
http://www.urionlinejudge.com.br/judge/pt/problems/view/1454
http://www.urionlinejudge.com.br/judge/pt/problems/view/1215