1. vector是表示可变大小数组的序列容器。
2. 就像数组一样，vector也采用的连续存储空间来存储元素。也就是意味着可以采用下标对vector的元素进行访问，和数组一样高效。但是又不像数组，它的大小是可以动态改变的，而且它的大小会被容器自动处理。
3. 本质讲，vector使用动态分配数组来存储它的元素。当新元素插入时候，这个数组需要被重新分配大小为了增加存储空间。其做法是，分配一个新的数组，然后将全部元素移到这个数组。就时间而言，这是一个相对代价高的任务，因为每当一个新的元素加入到容器的时候，vector并不会每次都重新分配大小。
4. vector分配空间策略：vector会分配一些额外的空间以适应可能的增长，因为存储空间比实际需要的存储空间更大。不同的库采用不同的策略权衡空间的使用和重新分配。但是无论如何，重新分配都应该是对数增长的间隔大小，以至于在末尾插入一个元素的时候是在常数时间的复杂度完成的。
5. 因此，vector占用了更多的存储空间，为了获得管理存储空间的能力，并且以一种有效的方式动态增长。
6. 与其它动态序列容器相比（deques, lists and forward\_lists）， vector在访问元素的时候更加高效，在末尾添加和删除元素相对高效。对于其它不在末尾的删除和插入操作，效率更低。比起lists和forward\_lists统一的迭代器和引用更好。

**用法**

**1. 头文件**

#include<vector>

**2. vector声明及初始化**

vector<int> vec; //声明一个int型向量

vector<int> vec(5); //声明一个初始大小为5的int向量

vector<int> vec(10, 1); //声明一个初始大小为10且值都是1的向量

vector<int> vec(tmp); //声明并用tmp向量初始化vec向量

vector<int> tmp(vec.begin(), vec.begin() + 3); //用向量vec的第0个到第2个值初始化tmp

int arr[5] = {1, 2, 3, 4, 5};

vector<int> vec(arr, arr + 5); //将arr数组的元素用于初始化vec向量

//说明：当然不包括arr[4]元素，末尾指针都是指结束元素的下一个元素，

//这个主要是为了和vec.end()指针统一。

vector<int> vec(&arr[1], &arr[4]); //将arr[1]~arr[4]范围内的元素作为vec的初始值

**3. vector基本操作**

(1). 容量

* 向量大小： vec.size();
* 向量最大容量： vec.max\_size();
* 更改向量大小： vec.resize();
* 向量真实大小： vec.capacity();
* 向量判空： vec.empty();
* 减少向量大小到满足元素所占存储空间的大小： vec.shrink\_to\_fit(); //[shrink\_to\_fit](http://www.cplusplus.com/reference/vector/vector/shrink_to_fit/)

(2). 修改

* 多个元素赋值： vec.assign(); //类似于初始化时用数组进行赋值
* 末尾添加元素： vec.push\_back();
* 末尾删除元素： vec.pop\_back();
* 任意位置插入元素： vec.insert();
* 任意位置删除元素： vec.erase();
* 交换两个向量的元素： vec.swap();
* 清空向量元素： vec.clear();

(3)迭代器

* 开始指针：vec.begin();
* 末尾指针：vec.end(); //指向最后一个元素的下一个位置
* 指向常量的开始指针： vec.cbegin(); //意思就是不能通过这个指针来修改所指的内容，但还是可以通过其他方式修改的，而且指针也是可以移动的。
* 指向常量的末尾指针： vec.cend();

(4)元素的访问

* 下标访问： vec[1]; //并不会检查是否越界
* at方法访问： vec.at(1); //以上两者的区别就是at会检查是否越界，是则抛出out of range异常
* 访问第一个元素： vec.front();
* 访问最后一个元素： vec.back();
* 返回一个指针： int\* p = vec.data(); //可行的原因在于vector在内存中就是一个连续存储的数组，所以可以返回一个指针指向这个数组。这是是C++11的特性。

(4)算法

* 遍历元素

vector<int>::iterator it;

for (it = vec.begin(); it != vec.end(); it++)

cout << \*it << endl;

//或者

for (size\_t i = 0; i < vec.size(); i++) {

cout << vec.at(i) << endl;

}

* 元素翻转

#include <algorithm>

reverse(vec.begin(), vec.end());

* 元素排序

#include <algorithm>

sort(vec.begin(), vec.end()); //采用的是从小到大的排序

//如果想从大到小排序，可以采用上面反转函数，也可以采用下面方法:

bool Comp(const int& a, const int& b) {

return a > b;

}

sort(vec.begin(), vec.end(), Comp);

cin输入字符串的总结：

**遇空字符（包括空格、回车、TAB）表示一个输入的结束，连续的空字符会被忽略。**

1.使用System.in.read

此种方法能从控制台接收一个字符，并且将该字符打印出来

**public static void t1()**

**throws IOException {**

**System.out.println("Enter a Char: ");**

**char c = (char) System.in.read();**

**System.out.println("your char is: " + c);**

**}**

此方法的缺点显而易见：

1.每次只能获取一个字符。

2.read方法获取的是int类型，需要根据需求做各种类型转换。

2.使用BufferedReader

**public static void t2() {**

**BufferedReader br = new BufferedReader(new InputStreamReader(System.in));**

**String str;**

**try {**

**System.out.println("please enter your name: ");**

**String name = br.readLine();**

**System.out.println("Your name is: " + name);**

**} catch (IOException e) {**

**e.printStackTrace();**

**}**

**}**

这种方法可以从控制台接受一个字符串，。并且打印出来。但是这种方式对于多次输入也不是很方便。

3.使用Scanner

(3)常用的next()方法系列：       nextInt():输入整数    nextLine():输入字符串     nextDouble():输入双精度数     next():输入字符串（以空格作为分隔符）

**public static void t3() {**

**Scanner sc = new Scanner(System.in);**

**System.out.println("please input your name: ");**

**String name = sc.nextLine();**

**System.out.println("please input your age: ");**

**int age = sc.nextInt();**

**System.out.println("please input your salary: ");**

**float salary = sc.nextFloat();**

**System.out.println("your msg is: ");**

**System.out.println("name: " + name + ", age: " + age + ", salary: " + salary);**

**}**

从 JDK 5.0 开始，基本类库中增加了java.util.Scanner类，根据它的 API 文档说明，这个类是采用正则表达式进行基本类型和字符串分析的文本扫描器。使用它的Scanner(InputStream source)构造方法，可以传入系统的输入流System.in而从控制台中读取数据。

## 一 动态规划

动态规划问题是面试题中的热门话题，如果要求一个问题的最优解（通常是最大值或者最小值），而且该问题能够分解成若干个子问题，并且小问题之间也存在重叠的子问题，则考虑采用动态规划。

使用动态规划特征：   
1. 求一个问题的最优解   
2. 大问题可以分解为子问题，子问题还有重叠的更小的子问题   
3. 整体问题最优解取决于子问题的最优解**（状态转移方程）**   
4. 从上往下分析问题，从下往上解决问题   
5. 讨论底层的**边界问题**

实例1   
剪绳子问题   
给你一根长度为N的绳子，请把绳子剪成M段（m,n都是整数），每段绳子的   
长度记为k[0],k[1],k[2]…. 请问如何剪绳子使得k[0],k[1],k[2]   
的乘积最大   
例如 绳子长度8 最大乘积18 = 2\*3\*3

def jianshengzi(n):

# 先对边界问题进行求解，因为明显剪的值小于不剪的值

# 则提出先讨论这三种情况

if n < 2:

return 0

if n == 2:

return 1 #长度为2，只能剪成1\*1

if n == 3:

return 2 #长度为3，剪成2\*1 > 1\*1\*1

#若绳子长于４呢,申请一个长度为50的数组

#罗列出切割的边界问题

h = [0]\*50

h[0] = 0

h[1] = 1

h[2] = 2

h[3] = 3

# 递归问题是 f(n) = max{f(i)\*f(n-i)}

for i in range(4,n+1):

maxs = 0

for j in range(1,i/2+1):

mult = h[j] \* h[i-j]

if maxs < mult:

maxs = mult

h[i] = maxs # 每次J的迭代轮询出该长度的最大值

print h

return h[n]

print jianshengzi(8)

实例2   
硬币问题   
我们有面值为1元3元5元的硬币若干枚，如何用最少的硬币凑够11元？   
分析：   
1 求问题的最优解：最小的硬币数   
2 是否有子问题：f(n)表示的最少硬币数是是上一次拿时候的硬币数最少。   
注意：f(n)是n元的最小硬币数，最后一次可拿的硬币数为1,3,5 则下一步   
的最小硬币数为 f(n-vi) 它的状态变更不是按元数的，是按照上次拿的硬币钱目   
3 状态转移方程为 f(n)= min(f(n-vi)+1)   
4 边界问题(找到最后一个重复的问题) 这里   
f(1)=1 ,f(2)=f(1)+f(1)=2 f(3)=min(1,f(2)+1)   
f(4)=f(3)+1 f(5)=1   
5 从上往下分析问题，从下往上解决问题。

def f(n):

if n == 1: #把所有的边界问题找到

return 1

if n == 2:

return 2

if n == 3:

return 1

if n == 4:

return 2

if n == 5:

return 1

h = [1,3,5]

minx = n

for i in range(3):

coun = f(n-h[i])+1 # 采用了递归的思想 这里是从上到下，

if minx > coun: # 复杂度比较高

minx = coun

return minx

print f(11)

def f(n):

if n == 1:

return 1

if n == 2:

return 2

if n == 3:

return 1

if n == 4:

return 2

if n == 5:

return 1

h = [1,3,5]

for x in range(6,n+1): #从下往上的思维解决

minx = n

for i in range(3):

coun = f(x-h[i])+1 #从下往上的思维解决

if minx > coun:

minx = coun

return minx

print f(11)

1．分析下面代码有什么问题？

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | void test1()  {   char string[10];   char\* str1 = "0123456789";   strcpy( string, str1 );  } |

# 参考答案

字符串str1需要11个字节才能存放下（包括末尾的’\0’），而string只有10个字节的空间，strcpy会导致数组越界；

2分析下面代码有什么问题？

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | void test2()  {   char string[10], str1[10];   int i;   for(i=0; i<10; i++)   {   str1  = 'a';   }  strcpy( string, str1 );  } |

# 参考答案

首先，代码根本不能通过编译。因为数组名str1为 char \*const类型的右值类型，根本不能赋值。

再者，即使想对数组的第一个元素赋值，也要使用 \*str1 = 'a';

其次，对字符数组赋值后，使用库函数strcpy进行拷贝操作，strcpy会从源地址一直往后拷贝，直到遇到'\0'为止。所以拷贝的长度是不定的。如果一直没有遇到'\0'导致越界访问非法内存，程序就崩了。

3浏览器页面有哪三层构成，分别是什么，作用是什么?

# 参考答案

构成：结构层、表示层、行为层

分别是：HTML、CSS、JavaScript

作用：HTML实现页面结构，CSS完成页面的表现与风格，JavaScript实现一些客户端的功能与业务