STL的 常 见　　误

主 讲 ： 肖 臻

STL中的容器（比如vector）

- 内存分配：vector中的元素是分配在heap上的，尽管vector本身是局部变量
  - int a[1024][1024];
  - vector<vector<int> > v(1024, vector<int>(1024));
  - 注意函数不要返回对局部变量的引用，包括对vector变
- vector的动态扩展
  - 下标操作符只能用于已经分配好的内存，不能通过赋值的方法动态扩展数组
    - vector<int> v; for (i=0; i<n; i++) v[i] = i; // 下标访问越界！
  - 用push_back来动态扩展数组

vector（续）

- 区分 reserve和 resize
  - reserve不分　内存，与 push_back　伍
    - 用 v[i]会导致数组访　越界
  - resize分　并初始化内存，与下标操作符　伍
    - 用 push_back会添加到初始内存之后。用数组定义也有同样的　：vector<int> v(n);
- 关于 resize
  - 如果 vector的定义可以推迟到知　数组大小之后，么可以省掉 resize
  - 如果 resize之前数组的内容无　保留，　么可以 clear之后再 resize，免得不必要的内存拷贝

内存管理与性能

- 内存管理
  - 系统会自动回收vector中分　　的内存，不用delete
  - 　　vector中的元素含有用new显式分　　的内存
- 性能考虑
  - 函数的形参是vector类型时，一般用引用类型来　　免拷贝
    - 使用加const的常引用来　　免误改
  - 返回vector的类型一般不会引起多余的拷贝：现代的编译器一般有优化

访　vector中的元素

- 显示地　历vector中的各元素，可以就用下标，不　要用iterator
  - for (int i=0; i<v.size(); i++) v[i] = …
  - for (vector<int>::const_iterator it=v.begin(); …) *it=…
- 如果用STL中的标准算法　式地　历vector，则要使用iterator，但是不用前　的繁琐声明
  - 比如reverse(v.begin(), v.end())
  - 注意：C语言中的数组名可以作为iterator，但是没有定义begin和end等成员函数
    - 比如int a[10]; sort(a, a+10); 而不是sort(a.begin(), a.end());

STL中的iterator

- 支持随机访问的容器一般不会要显式的iterator声明
  - 比如vector、string
- 用auto可以让编译器自动推断类型信息

图的两种表示方法

- 接矩 与 接表
  - 图论中绝大多数算法用 接表的效率
    - BFS、DFS、Dijkstra、SPFA
    - 例外：Floyd-Warshall算法一般基于 接矩
    - 朴素的Bellman-Ford算法只用一个包含所有边的大数组
  - 有时候输入数据的格式决定了哪种表示法更方便
    - 网络流图中有平行边， 要累积流 ，而这些平行边在输入数据中的位置是 意的，则要用 接矩
  - 图的规模比较小时，可以用 接矩 来模拟 接表
    - 对稀疏图来说并不省内存，但是可以达到 接表的效率

接表和　接矩

- 　接表：图中有n个节点，每个节点的出边用单独一行给出
  - vector<vector<int> > v(n);
  - 第i个　点有边指向第j个　点：v[i].push_back(j);
  - 二维数组中第一维是　先分　好的，第二维是动态增加的
- 　接矩
  - vector<vector<int> > v(n, vector<int>(n));
  - 第i个　点有边指向第j个　点：v[i][j] = 1;
- 注意：
  - 早期版本的编译器要求> >之　必　有空格
  - 如果　点编号是从1到n，　么分　内存时把n改为n+1

为什么不用new来分　内存？

- 用new很　分　出同时满足以下条件的　接矩　空
  - 矩　的维度是运行时　决定的
    - int (*adj)[n] = new int[m][n];要求n在编译时是常
  - 能够用下标操作符adj[i][j]访　矩　中元素
    - int *adj = new int[m*n];　要把二维下标转换成一维
  - 用单个new语句而　循环完成（内存碎片）
    - 先分　一个指　数组，再给每个指　分　行空
  - 有些　级技巧可以绕过以上　制，但是远不如使用vector方便
- 用new分　的内存　要用delete　放，而且无法初始化（　只分　单个元素）

与C语言相比

- STL容器的主要好处是　藏了动态内存分　的实现细节
- 　式前向星
  - 所有的边保存在一个大数组中
  - 每个　点记录边表在数组中的起始位置
  - 用数组下标代替　表指
  - 在一定程度上　低了编程复杂度
- 适用范围
  - 　中明确给出了图中边数的上界并且该上界远小于n^2，或者边数有天然上界（比如是棵树）
  - 否则只能按照图中最大可能的边数来分　大数组

容器适　器

- STL中的queue和stack一般只作为辅助数据结构来存储临时数据
  - 比如BFS中的　点　列、Prim算法中的优先　列
  - 不支持　历或　机访　　列或栈中的元素
    - 　用带有破坏性的方法，让各元素依次出　、出栈
  - 注意queue没有top()函数，而是用front()取　　元素
- 用vector来保存有　期价值的数据（比如函数的输入和返回值）
  - 如果产生的结果是逆序，比如用DFS实现的拓扑排序，用reverse来调整，而不要返回一个stack

C与C++的输入输出比较

- 一般来说，C++比C要安全、易用
  - 用scanf读入numeric值时，要用变量的地址，而不是变量本身
    - scanf(“%d”, &a[i]);
  - 读入double类型要用”%lf”，而不能用”%f”
    - 用printf写出时也应该用”%lf”
  - 注意printf没有输出bool类型的转换符，要显示地转为int类型后输出
    - 比如bool b[1024]; printf(“%d”, (int)b[100]);
  - 与cin不同，scanf对于输入中的空格是敏感的
    - 比如scanf(“%d %d ”, &i, &j); 相当于 cin >> i >> j;

C和 C++中 的 I/O

- 要 会 用 scanf：有 些　目 输 入 数 据　大，用 cin读 入 数 据 会 TLE
  - 比 如 用 判 定 二　图 的 方 法 来 解 蝴 蝶 分 类
  - 　定 向 stdin到 文 件 中 读 取 后，cin也 能 用
  - freopn(“file.txt”, “r”, stdin); cin >> i; …
- 关 于 EOF
  - Windows环 境 下，ctrl-Z要 单 独 占 一 行 并 回 车 才 能 产 生 EOF
  - 不 要 用 eof判 断 循 环 结 束
    - 比 如 while (!feof(fin)) { fscanf(fin, …); … }
  - 注 意 scanf读 入 失 败 时，返 回 的 是　值 -1
    - 比 如 while (scanf(“%d %d”, &i, &j)) { … }
    - 相 比 之 下 while (cin >> i >> j)则 没 有

结束语

- 这　课的目的是学算法，不是学C++
- STL中的很多工具用起来比较　手，可以提效率，但是不要本末倒置
  - 刷　的评判标准是正确性和速度，不是代码的优