STL的常见错误

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变量
- 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不分配内存，与push_back配伍
    - 用v[i]会导致数组访问越界
  - resize分配并初始化内存，与下标操作符配伍
    - 用push_back会添加到初始内存之后。用数组定义也有同样的问题：vector<int> v(n);
- 关于resize
  - 如果vector的定义可以推迟到知道数组大小之后，那么可以省掉resize
  - 如果resize之前数组的内容无需保留，那么可以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中分配的内存，不用delete
  - 除非vector中的元素含有用new显式分配的内存
- 性能考虑
  - 函数的形参是vector类型时，一般用引用类型来避免拷贝
    - 使用加const的常引用来避免误改
  - 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

访问vector中的元素

- 显示地遍历vector中的各元素，可以就用下标，不需要用iterator
  - for (int i=0; i<v.size(); i++) v[i] = …
  - for (vector<int>::const_iterator it=v.begin(); …) *it=…
- 如果用STL中的标准算法隐式地遍历vector，则需要使用iterator，但是不用前面的繁琐声明
  - 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需要显式的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

- 邻接矩阵与邻接表
  - 图论中绝大多数算法用邻接表的效率高
    - BFS、DFS、Dijkstra、SPFA
    - 例外：Floyd-Warshall算法一般基于邻接矩阵
    - 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，需要累积流量，而这些平行边在输入数据中的位置是随意的，则要用邻接矩阵
  - 图的规模比较小时，可以用邻接矩阵来模拟邻接表
    - 对稀疏图来说并不省内存，但是可以达到邻接表的效率

邻接表和邻接矩阵

- 邻接表：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个顶点有边指向第j个顶点：v[i].push_back(j);
  - 二维数组中第一维是预先分配好的，第二维是动态增加的
- 邻接矩阵
  - vector<vector<int> > v(n, vector<int>(n));
  - 第i个顶点有边指向第j个顶点：v[i][j] = 1;
- 注意：
  - 早期版本的编译器要求> >之间必须有空格
  - 如果顶点编号是从1到n，那么分配内存时把n改为n+1

为什么不用new来分配内存？

- 用new很难分配出同时满足以下条件的邻接矩阵空间
  - 矩阵的维度是运行时间决定的
    - int (*adj)[n] = new int[m][n];要求n在编译时是常量
  - 能够用下标操作符adj[i][j]访问矩阵中元素
    - int *adj = new int[m*n];需要把二维下标转换成一维
  - 用单个new语句而非循环完成（内存碎片）
    - 先分配一个指针数组，再给每个指针分配行空间
  - 有些高级技巧可以绕过以上限制，但是远不如使用vector方便
- 用new分配的内存需要用delete释放，而且无法初始化（除非只分配单个元素）

与C语言相比

- STL容器的主要好处是隐藏了动态内存分配的实现细节
- 链式前向星
  - 所有的边保存在一个大数组中
  - 每个顶点记录边表在数组中的起始位置
  - 用数组下标代替链表指针
  - 在一定程度上降低了编程复杂度
- 适用范围
  - 题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配大数组

容器适配器

- 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中的顶点队列、Prim算法中的优先队列
  - 不支持遍历或随机访问队列或栈中的元素
    - 除非用带有破坏性的方法，让各元素依次出队、出栈
  - 注意queue没有top()函数，而是用front()取队首元素
- 用vector来保存有长期价值的数据（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，需要显示地转为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：有些题目输入数据量大，用cin读入数据会TLE
  - 比如用判定二部图的方法来解蝴蝶分类那道题
  - 重定向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，返回的是非零值-1
    - 比如while (scanf(“%d %d”, &i, &j)) { … }
    - 相比之下while (cin >> i >> j)则没有问题

结束语

- 这门课的目的是学算法，不是学C++
- STL中的很多工具用起来比较顺手，可以提高效率，但是不要本末倒置
  - 刷题的评判标准是正确性和速度，不是代码的优雅