山东大学 计算机科学与技术 学院

数据结构与算法课程设计 课程实验报告

|  |  |  |  |
| --- | --- | --- | --- |
| 学号：202300130207 | 姓名：李臣思 | | 班级：23.5 |
| 实验题目：应用优先队列实现作业的优先调度 | | | |
| 实验学时：10 | | 实验日期：25.5.7 | |
| 实验目的：应用优先队列实现一个作业优先调度的系统 | | | |
| 软件开发工具：Visual Studio 2022 | | | |
| 1. 需求描述：   1.1问题描述：  优先队列priority queue是一种可以用于很多场合的数据结构，设计并实现一个优先队列。应用该优先队列实现作业的优先调度：  一个作业ti =（si，ei），si为作业的开始时间（进入时间），ei为作业的结束时间（离开时间）。作业调度的基本任务是从当前在系统中的作业中选取一个来执行，如果没有作业则执行nop操作。本题目要求的作业调度是基于优先级的调度，每次选取优先级最高的作业来调度，优先级用优先数（每个作业一个优先数pi）表征，优先数越小，优先级越高。作业ti进入系统时，即si时刻，系统给该作业指定其初始优先数pi = ei - si，从而使越短的作业优先级越高。该优先数在作业等待调度执行的过程中会不断减小，调整公式为：pi = pi - wi，其中的wi为作业ti的等待时间：wi = 当前时间-si。一旦作业被调度，该作业就一直执行，不能被抢占，只有当前执行作业指向完成时，才产生下一轮调度。所以可以在每次调度前动态调整各作业的优先数。  编程实现这样一个作业调度系统。  1.2基本要求：   1. 使用小顶堆（最小堆）实现优先队列，动态管理作业的插入、删除   和优先级调整。   1. 随机生成或手动输入作业集合，每个作业包含开始时间 si 和结束   时间 ei。   1. 初始优先级计算为 pi = ei - si（作业时长越短，优先级越高）。 2. 调度过程中，优先数动态调整：pi = pi - wi（wi 为当前时间与 si   的差值）。   1. 调度时选择优先级最高的作业执行，且执行期间不可抢占。 2. 分别用堆和左高树来实现优先队列。   1.3输入说明：  首先进行一个界面字体颜色选择，而后选择手动输入作业或者电脑自动生成作业，然后分别输入需要输入或者生成的作业个数，如果是手动输入作业，那么就开始输入作业。  输入界面设计：      对于两种数据结构实现，分别设计了两个程序。  输入样例说明：      1.3输出要求：  何时作业进入，何时调度哪个作业，何时离开，每个作业等待多长时间，优先数的动态变化情况等。  输出界面设计：  直接按照输出要求进行输出，同时添加一些加载的进度条显示来使得程序更加用户使用友好。  输出样例：       1. 分析与设计（整体思路描述，所需要的数据结构与算法）   2.1问题分析：  设计一个作业调度系统，基于优先队列动态管理作业优先级，要求：   * 数据结构：分别使用堆和左高树实现优先队列。 * 动态调整：根据作业的等待时间实时更新优先级。 * 非抢占式调度：作业一旦开始执行，必须完成才能调度下一个。 * 输出信息：详细记录作业进入、调度、完成时间及优先级变化。   2.2主程序设计：   * 用户界面与颜色设置：   显示欢迎界面，提供颜色选择菜单，允许用户自定义输出颜色以增强体验。  通过color()函数调用Windows API修改控制台文字颜色。   * 主菜单与输入处理：   提供三个选项：手动输入作业、自动生成作业、退出程序。  输入验证确保用户输入合法，避免程序崩溃。   * 数据处理与生成：   手动模式：用户逐个输入作业的开始和结束时间，程序检查合法性（开始<结束）。  自动模式：随机生成符合要求的作业数据，确保开始时间小于结束时间。  使用loading()和loading2()模拟数据处理进度，提升交互感。  2.3设计思路：  实现调度算法核心（check函数）：  最小堆/左高树构建：将作业按到达时间插入最小堆/左高树，优先数动态计算为(end - start) - wait。  时间步进循环：每秒检查新到达作业，更新所有作业的等待时间和优先数，重建树结构。  作业调度：选择优先数最小（最紧急）的作业执行，输出调度详情，更新系统时间。  2.4数据及数据类型定义：  **数据结构：**  分别用**最小堆以及左高树**结构来实现，其中每个作业都有一个结构体定义node：其中包含start（开始时间）、end（结束时间）、num（作业编号）、w（优先数）、wait（等待时间，初始化为0）。  **最小堆：**  首先定义node h[100010]；node\* e；分别用来存放最小堆数组以及作业，heapsize记录最小堆长度。  **左高树：**  首先定义一个结构体LeftistNode，优先数key，零距离s，开始时间start，结束时间end、作业编号num、等待时间wait，以及分别指向右孩子和左孩子right、left。  2.5算法设计及分析：  最小堆：  维护当前未调度的作业，堆顶元素为优先级最高的作业。最小堆中主要包含 **insert(int x, int k):**插入函数。将新的作业放入最小堆数组的末端，然  后从这个末端开始向上比较，比较和父亲节点优先级大小，如果更小则向上继续比较，该父节点下调，然后继续向上比较，循环比较。  **renew(int u)：**更新最小堆。从u开始向下比较，找到该交换的两个界定啊进行交换后，继续递归从新的点开始向下交换。  **check(int k)：**核心调度函数。首先将存入的作业数组e，按照优先级进行排序，同时初始化系统此时的调度时间time=0，标记flag=0，循环时间i从1开始一直循环++，如果当前时间==time，标记flag=0，每次循环中，将开始时间==i的作业插入最小堆中，然后如果当前flag为0，那么说明此时前一个作业已经调度完，可以开始调度新的作业，那么此时就选出最小堆的堆顶作业进行调度，而后输出结果，更新最小堆，重新标记flag为1，更新time。并且在每次循环末尾要更新所有作业的等待时间。  左高树：  实现左高树的各个操作。  **merge(LeftistNode\* a, LeftistNode\* b)：**合并两个左高树。确保a是根较小的树，若不是则调换a、b，再递归调用merge合并右子树；维护左高树性质，左高树的s值>=右子树，更新当前节点的s值。  **insert(LeftistNode\* root, LeftistNode\* newNode)：**调用merge函数。  **deleteMin(LeftistNode\* root)：**删除最小值，将根的左右子树合并为一个新的树，删除根。  **update(LeftistNode\* root, int currentTime)：**递归更新优先数和等待时间。  **rebuild(LeftistNode\* root)：**重建左高树，遍历收集原左高树的节点，然后进行一个重新插入，建立新的左高树。  **check(int k)：**核心调度函数从 currentTime = 1 开始，逐步推进至 2000。  若当前无作业在执行 (currentTime > executingUntil)，从左高树堆顶取出优先级最高的作业。记录调度开始时间、结束时间、等待时间及优先数变化，输出详细信息。通过 deleteMin 删除堆顶节点，删除已调度作业，并合并左右子树。   1. 测试结果（测试输入，测试输出）   对于最小堆实现优先队列：      对于左高树实现优先队列：       1. 分析与探讨（结果分析，若存在问题，探讨解决问题的途径）   结果分析：   |  |  |  | | --- | --- | --- | | 编号 | 开始时间 | 结束时间 | | 1 | 55 | 86 | | 2 | 52 | 64 | | 3 | 71 | 90 | | 4 | 46 | 92 | | 5 | 16 | 23 | | 6 | 52 | 95 | | 7 | 20 | 28 | | 8 | 37 | 99 | | 9 | 2 | 7 | | 10 | 20 | 94 |   所以可以分析得到，首先当系统时间到达2时，作业9则会压入系统中，此时系统中只有这么一个作业，那么就直接执行该作业，调度时间为5，那么到达7时，该作业执行完毕；而此时系统中没有压入新的作业，那么系统时间继续向后进行，到达16时压入新的作业5，调度到23时离开，而在16-23器件，作业7和10均会压入，而作业7的优先级显然更高，那么就会执行作业7；执行到28时时，没有新的作业压入，那么就执行作业10，这样执行到94时时，此时都进入系统，  所以最终结果与预期一致。   1. 附录：实现源代码（本实验的全部源程序代码，程序风格清晰易理解，有充分的注释）   最小堆：  #include<iostream>  #include<string>  #include <windows.h>  #include<random>  using namespace std;  typedef long long LL;  mt19937\_64 gen(time(0));  struct node {//定义作业的属性，放在结构体里  int start;//开始时间  int end;//结束时间  int num;//作业编号  int w;//优先数  int wait = 0;//等待时间  }h[1000010];  node\* e;  bool cmp(node a, node b) { return a.start < b.start;}  int randInt(int l, int r) {  return gen() % (r - l + 1) + l;  }//生成[l,R]的int型数字  LL randLL(LL l, LL r) {  return gen() % (r - l + 1) + l;  }//生成[l,R]的 long long int型数字  void loading() {//数据处理进度条  cout<<"\n数据处理中,请稍后……\n";  for (int i = 1; i <= 5; i++) {  cout<<"[";  for (int j = 1; j <= i; j++) cout<<"■■";  for (int j = 0; j < 5 - i; j++) cout<<"□□";  printf("]\t%d%%", i \* 20);  Sleep(1000);  cout<<"\r";  }  cout<<"\n\n";  return;  }  void loading2() {//打印数据处理进度条  cout<<"\n数据生成中,请稍后……\n";  for (int i = 1; i <= 2; i++) {  cout<<"[";  for (int j = 1; j <= i; j++) cout<<"■■■■■";  for (int j = 0; j < 2 - i; j++) cout<<"□□□□□";  printf("]\t%d%%", i \* 50);  Sleep(1000);  cout<<"\r";  }  cout<<"\n\n";  return;  }  int heapsize = 0;//最小堆中作业个数  void insert(int x, int k) {//插入最小堆  h[++heapsize].w = x;  h[heapsize].num = k;  int t = heapsize;  while (t > 1 && h[t / 2].w > h[t].w) {  swap(h[t],h[t / 2]);  t/= 2;  }  h[t].w = x;  return;  }  void renew(int u) {//更新最小堆  int t = u;  if (2 \* u <= heapsize && h[2 \* u].w < h[t].w) t = 2 \* u;  if (2 \* u + 1 <= heapsize && h[2 \* u + 1].w < h[t].w) t = 2 \* u + 1;  if (t != u) {  swap(h[t], h[u]);  renew(t);  }  return;  }  bool judge(string s) {//判断字符串是否完全由数字构成,即字符串是否合法  for (int i = 0; i < s.size(); i++)  if (s[i] >= '0' && s[i] <= '9')  continue;  else  return false;  return true;  }  void check(int k)//调度核心函数  {  int ans = 1;//调度序号计数器  sort(e, e + k, cmp);//按开始时间排序所有作业  int flag = 0;//CPU占用标志（0为空闲）  int time=0;//当前调度结束时间  int a = 0;  for (int i = 1; i <= 2000; i++) {//模拟时间流逝（1~2000时间单位）  if (i == time) flag = 0;//检查当前时间是否完成前一个调度  for (int j = 0; j < k; j++)  {  if (e[j].start == i)  {  insert(e[j].end, j);//将当前时间到达的作业插入堆  a++;  }  /\*else {  break;  }\*/  }  if (heapsize > 0) {//处理当前可调度作业  if (flag == 0) {//空闲时进行调度  int tot = h[1].num;  int weight = h[1].w;  swap(h[1], h[heapsize]);  heapsize--;  renew(1);  time = i + e[tot].end - e[tot].start;  printf("%d-编号为%d的作业于%d时进入队列\t于%d时开始调度\t于%d时离开\t调度时长为%d\t等待时间%d\t优先数动态变化%d -> %d\n",  ans++, e[tot].num, e[tot].start, i,  i + e[tot].end - e[tot].start, e[tot].end - e[tot].start,  e[tot].wait,e[tot].end-e[tot].start, e[tot].end - i);  flag = 1;//标记有作业正在调度    }  for (int idx = 1; idx <= heapsize; idx++) {  e[h[idx].num].wait=i- e[h[idx].num].start;//更新等待时间  h[idx].w = (e[h[idx].num].end - e[h[idx].num].start) - e[h[idx].num].wait;//更新有心啊及  }  }  }  return;  }  void color(int x) {  if (x == 0) //白色  return;  else if (x == 1) //红色  SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE),  FOREGROUND\_INTENSITY | FOREGROUND\_RED);  else if (x == 2) //绿色  SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE),  FOREGROUND\_INTENSITY | FOREGROUND\_GREEN);  else if (x == 3) //蓝色  SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE),  FOREGROUND\_INTENSITY | FOREGROUND\_BLUE);  else if (x == 4) //粉色  SetConsoleTextAttribute(  GetStdHandle(STD\_OUTPUT\_HANDLE),  FOREGROUND\_INTENSITY | FOREGROUND\_RED | FOREGROUND\_BLUE);  else if (x == 5) //青色  SetConsoleTextAttribute(  GetStdHandle(STD\_OUTPUT\_HANDLE),  FOREGROUND\_INTENSITY | FOREGROUND\_GREEN | FOREGROUND\_BLUE);  else if (x == 6) //黄色  SetConsoleTextAttribute(  GetStdHandle(STD\_OUTPUT\_HANDLE),  FOREGROUND\_INTENSITY | FOREGROUND\_RED | FOREGROUND\_GREEN);  else if (x == 7) //随机生成  color(randInt(0, 6));  return;  }  int main() {  cout<<"\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n"; //输出文字信息  cout<<"\* -基于优先队列的作业调度算法实现- \*\n";  cout<<"\* -最小堆实现- \*\n";  cout<<"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\n";  cout<<"\n----------字体颜色----------\n";  cout<<"0.白色\n";  cout<<"1.红色\n";  cout<<"2.绿色\n";  cout<<"3.蓝色\n";  cout<<"4.粉色\n";  cout<<"5.青色\n";  cout<<"6.黄色\n";  cout<<"7.随机生成\n";  cout<<"-----------------------------\n";  cout<<"\n请输入您想用字体颜色的编号:";  string tmp;  while (getline(cin, tmp)) {  if (tmp == "0" || tmp == "1" || tmp == "2" || tmp == "3" ||  tmp == "4" || tmp == "5" || tmp == "6" || tmp == "7") {  color(stoi(tmp));  break;  }  else  cout<<"非法输入,请重新输入:";  }  cout<<"\n ----------菜单----------\n";  cout<<" 1.手动输入作业信息\n";  cout<<" 2.电脑自动生成作业信息\n";  cout<<" 3.退出\n";  cout<<" ------------------------\n";  cout<<"\n请输入您所选的操作编号:";  string s, str;  while (getline(cin, s)) {  if (s == "1" || s == "2" || s == "3")  break;  else  cout<<"非法输入,请重新输入:";  }  if (s == "1") {  cout<<"请输入作业个数:";  while (getline(cin, str)) {  int num = stoi(str);  e = new node[num + 1];  if (num == 0) {  cout<<"当前没有作业需要被调度\n";  break;  }  cout<<"请按顺序给出%d个作业的开始时间和结束时间\n", num;  for (int i = 0; i < num; i++) {  printf(  "请输入第%d个作业的开始时间和结束时间，中间用空格间隔("  "1<=时间<=100):",  i + 1);  string str1, str2;  while (cin >> str1) {  cin >> str2;  if (judge(str1) && judge(str2) && stoi(str1) >= 1 &&  stoi(str1) <= 100 && stoi(str2) >= 1 &&  stoi(str2) <= 100 && stoi(str2) > stoi(str1)) {  e[i].start = stoi(str1);  e[i].end = stoi(str2);  e[i].num = i + 1;  break;  }  else  cout<<"非法输入,请重新输入:";  }  }  loading();  check(num);  break;  }  }  else if (s == "2") {  cout<<"请输入自动生成作业个数:";  while (getline(cin, str)) {  loading2();  int num = stoi(str);  e = new node[num + 1];  int k = 0;  while (k != num) {  int a = randInt(1, 100); //生成随机数  int b = randInt(1, 100);  if (a < b) {//去掉生成的非法随机数  e[k].start = a;  e[k].end = b;  e[k].num = k + 1;  k++;  }  }  if (k == 0) {  cout<<"当前没有作业需要被调度\n";  break;  }  cout<<"\n自动生成的数据如下:\n";  for (int i = 0; i < k; i++)  printf("第%d个作业的开始时间是:%d,结束时间是:%d\n", i + 1,  e[i].start, e[i].end);  loading();  check(k);  break;  }  }  cout<<"\n谢谢使用\n"; //程序结束  SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE),  FOREGROUND\_INTENSITY | FOREGROUND\_RED |  FOREGROUND\_GREEN |  FOREGROUND\_BLUE); //将字体重新设置为白色  return 0;  }  左高树：  struct LeftistNode {  int key; // 优先数 pi = (end - start) - wait  int s; // 零距离（s-value）  int start; // 开始时间  int end; // 结束时间  int num; // 作业编号  int wait; // 等待时间  LeftistNode\* left;  LeftistNode\* right;  LeftistNode(int k, int st, int e, int n)  : key(k), start(st), end(e), num(n), wait(0), s(1), left(nullptr), right(nullptr) {}  };  // 合并两棵左高树  LeftistNode\* merge(LeftistNode\* a, LeftistNode\* b) {  if (a == nullptr) return b;  if (b == nullptr) return a;  // 确保 a 是根较小的树  if (a->key > b->key) swap(a, b);  // 递归合并右子树  a->right = merge(a->right, b);  // 维护左高性质：左子树的 s 值 >= 右子树  if (a->left == nullptr || (a->left->s < a->right->s)) {  swap(a->left, a->right);  }  // 更新当前节点的 s 值  a->s = (a->right == nullptr) ? 1 : a->right->s + 1;  return a;  }  // 插入新节点  LeftistNode\* insert(LeftistNode\* root, LeftistNode\* newNode) {  return merge(root, newNode);  }  // 删除最小值（堆顶）  LeftistNode\* deleteMin(LeftistNode\* root) {  if (root == nullptr) return nullptr;  LeftistNode\* left = root->left;  LeftistNode\* right = root->right;  delete root;  return merge(left, right);  }  // 递归更新优先数和等待时间  void update(LeftistNode\* root, int currentTime) {  if (root == nullptr) return;  root->wait = currentTime - root->start;  root->key = (root->end - root->start) - root->wait;  update(root->left, currentTime);  update(root->right, currentTime);  }  // 重建左高树（先遍历收集节点，再重新插入）  LeftistNode\* rebuild(LeftistNode\* root) {  if (root == nullptr) return nullptr;  vector<LeftistNode\*> nodes;  queue<LeftistNode\*> q;  q.push(root);  while (!q.empty()) {  LeftistNode\* node = q.front();  q.pop();  nodes.push\_back(node);  if (node->left) q.push(node->left);  if (node->right) q.push(node->right);  }  LeftistNode\* newRoot = nullptr;  for (auto node : nodes) {  node->left = node->right = nullptr;  newRoot = insert(newRoot, node);  }  return newRoot;  }  void check(int k) {  sort(e, e + k, [](node a, node b) { return a.start < b.start; });  int ans = 1;  int currentTime = 1;  LeftistNode\* root = nullptr;  int executingUntil = -1; // 当前作业执行结束时间  while (currentTime <= 2000) {  //添加新到达的作业到左高树  for (int j = 0; j < k; j++) {  if (e[j].start == currentTime) {  int pi = e[j].end - e[j].start;  LeftistNode\* newNode = new LeftistNode(pi, e[j].start, e[j].end, e[j].num);  root = insert(root, newNode);  }  }  //动态更新优先数和等待时间，并重建左高树  if (root != nullptr) {  update(root, currentTime);  root = rebuild(root); // 重建以维护堆性质  }  //调度作业（如果当前无作业在执行）  if (currentTime > executingUntil) {  if (root != nullptr) {  LeftistNode\* minNode = root;  int duration = minNode->end - minNode->start;  executingUntil = currentTime + duration - 1;  // 输出调度信息  printf("%d-编号为%d的作业于%d时进入队列\t于%d时开始调度\t于%d时离开\t调度时长为%d\t等待时间%d\t优先数动态变化%d -> %d\n",  ans++, minNode->num, minNode->start, currentTime,  currentTime + duration, duration, minNode->wait,  (minNode->end - minNode->start), minNode->key);  // 删除已调度的作业  root = deleteMin(root);  }  }  currentTime++;  }  }  void color(int x) {  if (x == 0) //白色  return;  else if (x == 1) //红色  SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE),  FOREGROUND\_INTENSITY | FOREGROUND\_RED);  else if (x == 2) //绿色  SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE),  FOREGROUND\_INTENSITY | FOREGROUND\_GREEN);  else if (x == 3) //蓝色  SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE),  FOREGROUND\_INTENSITY | FOREGROUND\_BLUE);  else if (x == 4) //粉色  SetConsoleTextAttribute(  GetStdHandle(STD\_OUTPUT\_HANDLE),  FOREGROUND\_INTENSITY | FOREGROUND\_RED | FOREGROUND\_BLUE);  else if (x == 5) //青色  SetConsoleTextAttribute(  GetStdHandle(STD\_OUTPUT\_HANDLE),  FOREGROUND\_INTENSITY | FOREGROUND\_GREEN | FOREGROUND\_BLUE);  else if (x == 6) //黄色  SetConsoleTextAttribute(  GetStdHandle(STD\_OUTPUT\_HANDLE),  FOREGROUND\_INTENSITY | FOREGROUND\_RED | FOREGROUND\_GREEN);  else if (x == 7) //随机生成  color(randInt(0, 6));  return;  }  int main() {  cout << "\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n"; //输出文字信息  cout << "\* -基于优先队列的作业调度算法实现- \*\n";  cout << "\* -左高树实现- \*\n";  cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\n";  cout << "\n----------字体颜色----------\n";  cout << "0.白色\n";  cout << "1.红色\n";  cout << "2.绿色\n";  cout << "3.蓝色\n";  cout << "4.粉色\n";  cout << "5.青色\n";  cout << "6.黄色\n";  cout << "7.随机生成\n";  cout << "-----------------------------\n";  cout << "\n请输入您想用字体颜色的编号:";  string tmp;  while (getline(cin, tmp)) {  if (tmp == "0" || tmp == "1" || tmp == "2" || tmp == "3" ||  tmp == "4" || tmp == "5" || tmp == "6" || tmp == "7") {  color(stoi(tmp));  break;  }  else  cout << "非法输入,请重新输入:";  }  cout << "\n ----------菜单----------\n";  cout << " 1.手动输入作业信息\n";  cout << " 2.电脑自动生成作业信息\n";  cout << " 3.退出\n";  cout << " ------------------------\n";  cout << "\n请输入您所选的操作编号:";  string s, str;  while (getline(cin, s)) {  if (s == "1" || s == "2" || s == "3")  break;  else  cout << "非法输入,请重新输入:";  }  if (s == "1") {  cout << "请输入作业个数:";  while (getline(cin, str)) {  int num = stoi(str);  e = new node[num + 1];  if (num == 0) {  cout << "当前没有作业需要被调度\n";  break;  }  cout << "请按顺序给出%d个作业的开始时间和结束时间\n", num;  for (int i = 0; i < num; i++) {  printf(  "请输入第%d个作业的开始时间和结束时间，中间用空格间隔("  "1<=时间<=100):",  i + 1);  string str1, str2;  while (cin >> str1) {  cin >> str2;  if (judge(str1) && judge(str2) && stoi(str1) >= 1 &&  stoi(str1) <= 100 && stoi(str2) >= 1 &&  stoi(str2) <= 100 && stoi(str2) > stoi(str1)) {  e[i].start = stoi(str1);  e[i].end = stoi(str2);  e[i].num = i + 1;  break;  }  else  cout << "非法输入,请重新输入:";  }  }  loading();  check(num);  break;  }  }  else if (s == "2") {  cout << "请输入自动生成作业个数:";  while (getline(cin, str)) {  loading2();  int num = stoi(str);  e = new node[num + 1];  int k = 0;  while (k != num) {  int a = randInt(1, 100); //生成随机数  int b = randInt(1, 100);  if (a < b) {//去掉生成的非法随机数  e[k].start = a;  e[k].end = b;  e[k].num = k + 1;  k++;  }  }  if (k == 0) {  cout << "当前没有作业需要被调度\n";  break;  }  cout << "\n自动生成的数据如下:\n";  for (int i = 0; i < k; i++)  printf("第%d个作业的开始时间是:%d,结束时间是:%d\n", i + 1,  e[i].start, e[i].end);  loading();  check(k);  break;  }  }  cout << "\n谢谢使用\n"; //程序结束  SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE),  FOREGROUND\_INTENSITY | FOREGROUND\_RED |  FOREGROUND\_GREEN |  FOREGROUND\_BLUE); //将字体重新设置为白色  return 0;  } | | | |