

数据结构与算法 课程实验报告

学号：201700140056	姓名：李港	班级：跟 18.2 (17.4)
实验题目：实验八 散列表		
实验学时：2h	实验日期：2019.11.07	
<b>实验目的：</b> 1、掌握散列表结构的定义和实现。 2、掌握散列表结构的应用。		
<b>软件开发工具：</b> Virtual Studio 2019		
<b>1. 实验内容</b> 1. 分别使用线性开型寻址和链表散列解决溢出，创建散列表类； 2. 使用散列表设计实现一个字典，假设关键字为整数且 D 为 961，在字典中插入随机产生的 500 个不同的整数，实现字典的建立和搜索操作。*实现字典的删除。		
<b>2. 数据结构与算法描述（整体思路描述，所需要的数据结构与算法）</b> <b>总体思路：</b> 3. 线性开型寻址使用原生数组存储元素。 4. 链表散列使用链表存储元素 5. 两种结构均提供插入、删除、查询等操作, 两种结构相同功能函数的参数与返回值相同，确保兼容性。 6. 两种结构的某些函数因 0J 的要求不同而有细微区别。 7. 使用模板元编程的方法提供不同类型的哈希函数。 8. int 型哈希函数直接返回原数值。 9. 在基本数据结构方面，线性开型寻址的散列表比链表散列的散列表简单。 10. 在实现功能方面，链表散列的散列表比线性开型寻址的散列表简单。		
<b>数据结构：</b> 1. 哈希表采用原生数组作为底层数据结构 2. 链表散列使用链表类作为底层数据结构，一个链表散列的散列表类是多个链表的集合。链表提供插入，删除等操作，散列表根据哈希函数分发这些操作到不同链表中。 3. 使用 mypair 类来存储键值对，提供构造函数；线性开型寻址和链表散列的散列表类使用相同的 mypair 类。 4. 链表使用 mynode 作为节点。		
<b>算法：</b> 1. 线性开型寻址： 1. 搜索： 1. 先获得关键词的桶编号 2. 若该处为空，则搜索失败 3. 若该处元素关键词不需要的关键词，则向后寻找 4. 若将表搜索一遍仍未找到目标元素，则搜索失败 2. 插入：应当注意，线性开型寻址的散列表存在无法插入的情况 1. 首先获得关键词对应的桶编号，并不断向后对比		

2. 对比过程中，若桶中键值对关键词与要插入的关键词相同，则修改对应数据
  3. 若找不到关键词但有剩余空间则插入新的键值对
  4. 若找遍数组都没找到对应关键词，也没有剩余空间，则插入失败
3. 删除：需要注意删除元素后的数组整理过程
1. 首先按照搜索的方法找到对应位置，然后删除元素
  2. 接下来索引不断增加，将后续元素进行整理
    1. 若索引指向位置的键值对的关键词不在它应该在的位置上（通过 `_getPosByKey` 得到），则将此元素移动到它该在的位置上，并清空原来的位置。
    2. 若索引指向位置为空，则说明该表已整理好，返回移动的元素个数。

## 2. 链表散列：

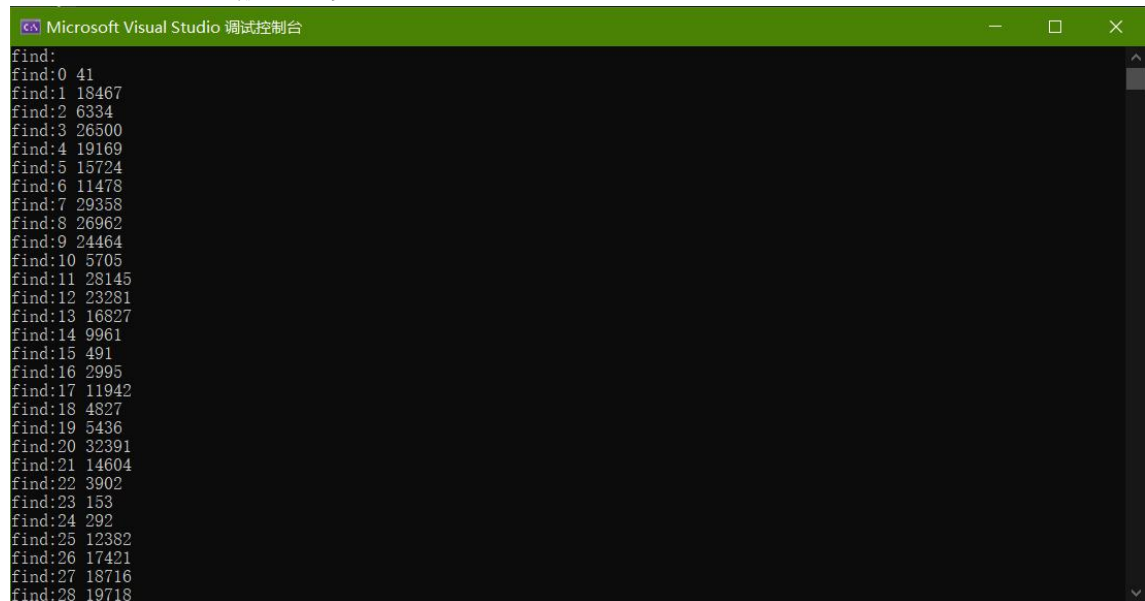
1. 搜索：
  1. 先获得关键词的链表编号
  2. 再从相应链表中寻找该元素
  3. 找到则返回数据指针，找不到返回空指针
2. 插入：
  1. 将键值对插入到相应的链表中
  2. 若关键词已存在则修改关键词对应的数据
3. 删除：
  1. 获得关键词对应的链表编号
  2. 在链表中删除该元素
    1. 若链表中存在此元素，则删除之
    2. 若链表中不存在此元素，则删除失败

## 3. 测试结果（测试输入，测试输出）

### 1. 验收展示：

插入完毕后，屏幕上依次打印：查找过程，整个散列表，删除过程，整个散列表；可见代码结果正确。

### 1. 线性开型寻址的散列表类：



```
Microsoft Visual Studio 调试控制台
find:
find:0 41
find:1 18467
find:2 6334
find:3 26500
find:4 19169
find:5 15724
find:6 11478
find:7 29358
find:8 26962
find:9 24464
find:10 5705
find:11 28145
find:12 23281
find:13 16827
find:14 9961
find:15 491
find:16 2995
find:17 11942
find:18 4827
find:19 5436
find:20 32391
find:21 14604
find:22 3902
find:23 153
find:24 292
find:25 12382
find:26 17421
find:27 18716
find:28 19718
```

```
Microsoft Visual Studio 调试控制台
find:487 6902
find:488 14962
find:489 24596
find:490 3737
find:491 13261
find:492 10195
find:493 32525
find:494 1264
find:495 8260
find:496 6202
find:497 8116
find:498 5030
find:499 20326

show all:
0:41 1:18467 2:6334 3:26500 4:19169 5:15724 6:11478 7:29358 8:26962 9:24464 10:5705 11:28145 12:23281 13:16827 14:9961 1
5:491 16:2995 17:11942 18:4827 19:5436 20:32391 21:14604 22:3902 23:153 24:292 25:12382 26:17421 27:18716 28:19718 29:19
895 30:5447 31:21726 32:14771 33:11538 34:1869 35:19912 36:25667 37:26299 38:17035 39:9894 40:28703 41:23811 42:31322 43
:30333 44:17673 45:4664 46:15141 47:7711 48:28253 49:6868 50:25547 51:27644 52:32662 53:32757 54:20037 55:12859 56:8723
57:9741 58:27529 59:778 60:12316 61:3035 62:22190 63:1842 64:288 65:30106 66:9040 67:8942 68:19264 69:22648 70:27446 71:
23805 72:15890 73:6729 74:24370 75:15350 76:15006 77:31101 78:24393 79:3548 80:19629 81:12623 82:24084 83:19954 84:18756
85:11840 86:4966 87:7376 88:13931 89:26308 90:16944 91:32439 92:24626 93:11323 94:5537 95:21538 96:16118 97:2082 98:229
99:16541 100:4833 101:31115 102:4639 103:29658 104:22704 105:9930 106:13977 107:2306 108:31673 109:22386 110:5021 111
:28745 112:26924 113:19072 114:6270 115:5829 116:26777 117:15573 118:5097 119:16512 120:23986 121:13290 122:9161 123:186
36 124:22355 125:24767 126:23655 127:15574 128:4031 129:12052 130:27350 131:1150 132:16941 133:21724 134:13966 135:3430
136:31107 137:30191 138:18007 139:11337 140:15457 141:12287 142:27753 143:10383 144:14945 145:8909 146:32209 147:9758 14
8:24221 149:18588 150:6422 151:24946 152:27506 153:13030 154:16413 155:29168 156:900 157:32591 158:18762 159:1655 160:17
410 161:6359 162:27624 163:20537 164:21548 165:6483 166:27595 167:4041 168:3602 169:24350 170:10291 171:30836 172:9374 1
73:11020 174:4596 175:24021 176:27348 177:23199 178:19668 179:24484 180:8281 181:4734 182:1999 183:26418 184:27938 185:6
900 186:3788 187:18127 188:467 189:3728 190:14893 191:24648 192:22483 193:17807 194:2421 195:14310 196:6617 197:22813 19
8:26962 199:24464 200:5705 201:28145 202:23281 203:16827 204:9961 205:491 206:2995 207:11942 208:4827 209:5436 210:32391
211:14604 212:3902 213:153 214:292 215:12382 216:17421 217:18716 218:19718 219:895 220:5447 221:21726 222:14771 223:11538 224:1869
225:19912 226:25667 227:26299 228:17035 229:9894 230:28703 231:23811 232:31322 233:30333 234:17673 235:4664 236:15141
237:7711 238:28253 239:6868 240:25547 241:27644 242:32662 243:32757 244:20037 245:12859 246:8723 247:9741 248:27529
249:778 250:12316 251:3035 252:22190 253:1842 254:288 255:30106 256:9040 257:8942 258:19264 259:22648 260:27446 261:23805
262:15890 263:6729 264:24370 265:15350 266:15006 267:31101 268:24393 269:3548 270:19629 271:12623 272:24084 273:19954 274:18756
275:11840 276:4966 277:7376 278:13931 279:26308 280:16944 281:32439 282:24626 283:11323 284:5537 285:21538 286:16118 287:2082
288:229 289:16541 290:4833 291:31115 292:4639 293:29658 294:22704 295:9930 296:13977 297:2306 298:31673 299:22386 300:5021
301:28745 302:26924 303:19072 304:6270 305:5829 306:26777 307:15573 308:5097 309:16512 310:23986 311:13290 312:9161 313:186
36 314:22355 315:24767 316:23655 317:15574 318:4031 319:12052 320:27350 321:1150 322:16941 323:21724 324:13966 325:3430
326:31107 327:30191 328:18007 329:11337 330:15457 331:12287 332:27753 333:10383 334:14945 335:8909 336:32209 337:9758 338:24221
339:18588 340:6422 341:24946 342:27506 343:13030 344:16413 345:29168 346:900 347:32591 348:18762 349:1655 350:17410 351:6359
352:27624 353:20537 354:21548 355:6483 356:27595 357:4041 358:3602 359:24350 360:10291 361:30836 362:9374 363:11020 364:4596
365:24021 366:27348 367:23199 368:19668 369:24484 370:8281 371:4734 372:1999 373:26418 374:27938 375:6900 376:3788 377:18127
378:467 379:3728 380:14893 381:24648 382:22483 383:17807 384:2421 385:14310 386:6617 387:22813 388:26962 389:24464 390:5705
391:28145 392:23281 393:16827 394:9961 395:491 396:2995 397:11942 398:4827 399:5436 400:32391 401:14604 402:3902 403:153 404:292
405:12382 406:17421 407:18716 408:19718 409:895 410:5447 411:21726 412:14771 413:11538 414:1869 415:19912 416:25667 417:26299
418:17035 419:9894 420:28703 421:23811 422:31322 423:30333 424:17673 425:4664 426:15141 427:7711 428:28253 429:6868 430:25547
431:27644 432:32662 433:32757 434:20037 435:12859 436:8723 437:9741 438:27529 439:778 440:12316 441:3035 442:22190 443:1842 444:288
445:30106 446:9040 447:8942 448:19264 449:22648 450:27446 451:23805 452:15890 453:6729 454:24370 455:15350 456:15006 457:31101
458:24393 459:3548 460:19629 461:12623 462:24084 463:19954 464:18756 465:11840 466:4966 467:7376 468:13931 469:26308 470:16944
471:32439 472:24626 473:11323 474:5537 475:21538 476:16118 477:2082 478:229 479:16541 480:4833 481:31115 482:4639 483:29658
484:22704 485:9930 486:13977 487:2306 488:31673 489:22386 490:5021 491:28745 492:26924 493:19072 494:6270 495:5829 496:26777
497:15573 498:5097 499:16512 500:23986 501:13290 502:9161 503:18636 504:22355 505:24767 506:23655 507:15574 508:4031 509:12052
510:27350 511:1150 512:16941 513:21724 514:13966 515:3430 516:31107 517:30191 518:18007 519:11337 520:15457 521:12287 522:27753
523:10383 524:14945 525:8909 526:32209 527:9758 528:24221 529:18588 530:6422 531:24946 532:27506 533:13030 534:16413 535:29168
536:900 537:32591 538:18762 539:1655 540:17410 541:6359 542:27624 543:20537 544:21548 545:6483 546:27595 547:4041 548:3602 549:24350
550:10291 551:30836 552:9374 553:11020 554:4596 555:24021 556:27348 557:23199 558:19668 559:24484 560:8281 561:4734 562:1999
563:26418 564:27938 565:6900 566:3788 567:18127 568:467 569:3728 570:14893 571:24648 572:22483 573:17807 574:2421 575:14310
576:6617 577:22813 578:26962 579:24464 580:5705 581:28145 582:23281 583:16827 584:9961 585:491 586:2995 587:11942 588:4827
589:5436 590:32391 591:14604 592:3902 593:153 594:292 595:12382 596:17421 597:18716 598:19718 599:895 600:5447 601:21726
602:14771 603:11538 604:1869 605:19912 606:25667 607:26299 608:17035 609:9894 610:28703 611:23811 612:31322 613:30333 614:17673
615:4664 616:15141 617:7711 618:28253 619:6868 620:25547 621:27644 622:32662 623:32757 624:20037 625:12859 626:8723 627:9741
628:27529 629:778 630:12316 631:3035 632:22190 633:1842 634:288 635:30106 636:9040 637:8942 638:19264 639:22648 640:27446
641:23805 642:15890 643:6729 644:24370 645:15350 646:15006 647:31101 648:24393 649:3548 650:19629 651:12623 652:24084 653:19954
654:18756 655:11840 656:4966 657:7376 658:13931 659:26308 660:16944 661:32439 662:24626 663:11323 664:5537 665:21538 666:16118
667:2082 668:229 669:16541 670:4833 671:31115 672:4639 673:29658 674:22704 675:9930 676:13977 677:2306 678:31673 679:22386
680:5021 681:28745 682:26924 683:19072 684:6270 685:5829 686:26777 687:15573 688:5097 689:16512 690:23986 691:13290 692:9161
693:18636 694:22355 695:24767 696:23655 697:15574 698:4031 699:12052 700:27350 701:1150 702:16941 703:21724 704:13966 705:3430
706:31107 707:30191 708:18007 709:11337 710:15457 711:12287 712:27753 713:10383 714:14945 715:8909 716:32209 717:9758 718:24221
719:18588 720:6422 721:24946 722:27506 723:13030 724:16413 725:29168 726:900 727:32591 728:18762 729:1655 730:17410 731:6359
732:27624 733:20537 734:21548 735:6483 736:27595 737:4041 738:3602 739:24350 740:10291 741:30836 742:9374 743:11020 744:4596
745:24021 746:27348 747:23199 748:19668 749:24484 750:8281 751:4734 752:1999 753:26418 754:27938 755:6900 756:3788 757:18127
758:467 759:3728 760:14893 761:24648 762:22483 763:17807 764:2421 765:14310 766:6617 767:22813 768:26962 769:24464 770:5705
771:28145 772:23281 773:16827 774:9961 775:491 776:2995 777:11942 778:4827 779:5436 780:32391 781:14604 782:3902 783:153 784:292
785:12382 786:17421 787:18716 788:19718 789:895 790:5447 791:21726 792:14771 793:11538 794:1869 795:19912 796:25667 797:26299
798:17035 799:9894 800:28703 801:23811 802:31322 803:30333 804:17673 805:4664 806:15141 807:7711 808:28253 809:6868 810:25547
811:27644 812:32662 813:32757 814:20037 815:12859 816:8723 817:9741 818:27529 819:778 820:12316 821:3035 822:22190 823:1842 824:288
825:30106 826:9040 827:8942 828:19264 829:22648 830:27446 831:23805 832:15890 833:6729 834:24370 835:15350 836:15006 837:31101
838:24393 839:3548 840:19629 841:12623 842:24084 843:19954 844:18756 845:11840 846:4966 847:7376 848:13931 849:26308 850:16944
851:32439 852:24626 853:11323 854:5537 855:21538 856:16118 857:2082 858:229 859:16541 860:4833 861:31115 862:4639 863:29658
864:22704 865:9930 866:13977 867:2306 868:31673 869:22386 870:5021 871:28745 872:26924 873:19072 874:6270 875:5829 876:26777
877:15573 878:5097 879:16512 880:23986 881:13290 882:9161 883:18636 884:22355 885:24767 886:23655 887:15574 888:4031 889:12052
890:27350 891:1150 892:16941 893:21724 894:13966 895:3430 896:31107 897:30191 898:18007 899:11337 900:15457 901:12287 902:27753
903:10383 904:14945 905:8909 906:32209 907:9758 908:24221 909:18588 910:6422 911:24946 912:27506 913:13030 914:16413 915:29168
916:900 917:32591 918:18762 919:1655 920:17410 921:6359 922:27624 923:20537 924:21548 925:6483 926:27595 927:4041 928:3602 929:24350
930:10291 931:30836 932:9374 933:11020 934:4596 935:24021 936:27348 937:23199 938:19668 939:24484 940:8281 941:4734 942:1999
943:26418 944:27938 945:6900 946:3788 947:18127 948:467 949:3728 950:14893 951:24648 952:22483 953:17807 954:2421 955:14310
956:6617 957:22813 958:26962 959:24464 960:5705 961:28145 962:23281 963:16827 964:9961 965:491 966:2995 967:11942 968:4827
969:5436 970:32391 971:14604 972:3902 973:153 974:292 975:12382 976:17421 977:18716 978:19718 979:895 980:5447 981:21726
982:14771 983:11538 984:1869 985:19912 986:25667 987:26299 988:17035 989:9894 990:28703 991:23811 992:31322 993:30333 994:17673
995:4664 996:15141 997:7711 998:28253 999:6868 1000:25547 1001:27644 1002:32662 1003:32757 1004:20037 1005:12859 1006:8723
1007:9741 1008:27529 1009:778 1010:12316 1011:3035 1012:22190 1013:1842 1014:288 1015:30106 1016:9040 1017:8942 1018:19264
1019:22648 1020:27446 1021:23805 1022:15890 1023:6729 1024:24370 1025:15350 1026:15006 1027:31101 1028:24393 1029:3548 1030:19629
1031:12623 1032:24084 1033:19954 1034:18756 1035:11840 1036:4966 1037:7376 1038:13931 1039:26308 1040:16944 1041:32439 1042:24626
1043:11323 1044:5537 1045:21538 1046:16118 1047:2082 1048:229 1049:16541 1050:4833 1051:31115 1052:4639 1053:29658 1054:22704
1055:9930 1056:13977 1057:2306 1058:31673 1059:22386 1060:5021 1061:28745 1062:26924 1063:19072 1064:6270 1065:5829 1066:26777
1067:15573 1068:5097 1069:16512 1070:23986 1071:13290 1072:9161 1073:18636 1074:22355 1075:24767 1076:23655 1077:15574 1078:4031
1079:12052 1080:27350 1081:1150 1082:16941 1083:21724 1084:13966 1085:3430 1086:31107 1087:30191 1088:18007 1089:11337 1090:15457
1091:12287 1092:27753 1093:10383 1094:14945 1095:8909 1096:32209 1097:9758 1098:24221 1099:18588 1100:6422 1101:24946 1102:27506
1103:13030 1104:16413 1105:29168 1106:900 1107:32591 1108:18762 1109:1655 1110:17410 1111:6359 1112:27624 1113:20537 1114:21548
1115:6483 1116:27595 1117:4041 1118:3602 1119:24350 1120:10291 1121:30836 1122:9374 1123:11020 1124:4596 1125:24021 1126:27348
1127:23199 1128:19668 1129:24484 1130:8281 1131:4734 1132:1999 1133:26418 1134:27938 1135:6900 1136:3788 1137:18127 1138:467
1139:3728 1140:14893 1141:24648 1142:22483 1143:17807 1144:2421 1145:14310 1146:6617 1147:22813 1148:26962 1149:24464 1150:5705
1151:28145 1152:23281 1153:16827 1154:9961 1155:491 1156:2995 1157:11942 1158:4827 1159:5436 1160:32391 1161:14604 1162:3902
1163:153 1164:292 1165:12382 1166:17421 1167:18716 1168:19718 1169:895 1170:5447 1171:21726 1172:14771 1173:11538 1174:1869
1175:19912 1176:25667 1177:26299 1178:17035 1179:9894 1180:28703 1181:23811 1182:31322 1183:30333 1184:17673 1185:4664 1186:15141
1187:7711 1188:28253 1189:6868 1190:25547 1191:27644 1192:32662 1193:32757 1194:20037 1195:12859 1196:8723 1197:9741 1198:27529
1199:778 1200:12316 1201:3035 1202:22190 1203:1842 1204:288 1205:30106 1206:9040 1207:8942 1208:19264 1209:22648 1210:27446
1211:23805 1212:15890 1213:6729 1214:24370 1215:15350 1216:15006 1217:31101 1218:24393 1219:3548 1220:19629 1221:12623 1222:24084
1223:19954 1224:18756 1225:11840 1226:4966 1227:7376 1228:13931 1229:26308 1230:16944 1231:32439 1232:24626 1233:11323 1234:5537
1235:21538 1236:16118 1237:2082 1238:229 1239:16541 1240:4833 1241:31115 1242:4639 1243:29658 1244:22704 1245:9930 1246:13977
1247:2306 1248:31673 1249:22386 1250:5021 1251:2874
```

```
Microsoft Visual Studio 调试控制台
find:486 28520
find:487 6902
find:488 14962
find:489 24596
find:490 3737
find:491 13261
find:492 10195
find:493 32525
find:494 1264
find:495 8260
find:496 6202
find:497 8116
find:498 5030
find:499 20326

show all:
Chain0: 0:41 Chain1: 1:18467 Chain2: 2:6334 Chain3: 3:26500 Chain4: 4:19169 Chain5: 5:15724 Chain6: 6:11478
Chain7: 7:29358 Chain8: 8:26962 Chain9: 9:24464 Chain10: 10:5705 Chain11: 11:28145 Chain12: 12:23281 Chain
13: 13:16827 Chain14: 14:9961 Chain15: 15:491 Chain16: 16:2995 Chain17: 17:11942 Chain18: 18:4827 Chain19: 1
9:5436 Chain20: 20:32391 Chain21: 21:14604 Chain22: 22:3902 Chain23: 23:153 Chain24: 24:292 Chain25: 25:1238
2 Chain26: 26:17421 Chain27: 27:18716 Chain28: 28:19718 Chain29: 29:19895 Chain30: 30:5447 Chain31: 31:21726
Chain32: 32:14771 Chain33: 33:11538 Chain34: 34:1869 Chain35: 35:19912 Chain36: 36:25667 Chain37: 37:26299
Chain38: 38:17035 Chain39: 39:9894 Chain40: 40:28703 Chain41: 41:23811 Chain42: 42:31322 Chain43: 43:30333
Chain44: 44:17673 Chain45: 45:4664 Chain46: 46:15141 Chain47: 47:7711 Chain48: 48:28253 Chain49: 49:6868 Ch
ain50: 50:25547 Chain51: 51:27644 Chain52: 52:32662 Chain53: 53:32757 Chain54: 54:20037 Chain55: 55:12859 Ch
ain56: 56:8723 Chain57: 57:9741 Chain58: 58:27529 Chain59: 59:778 Chain60: 60:12316 Chain61: 61:3035 Chain62
: 62:22190 Chain63: 63:1842 Chain64: 64:288 Chain65: 65:30106 Chain66: 66:9040 Chain67: 67:8942 Chain68: 68:
19264 Chain69: 69:22648 Chain70: 70:27446 Chain71: 71:23805 Chain72: 72:15890 Chain73: 73:6729 Chain74: 74:2
4370 Chain75: 75:15350 Chain76: 76:15006 Chain77: 77:31101 Chain78: 78:24393 Chain79: 79:3548 Chain80: 80:19
629 Chain81: 81:12623 Chain82: 82:24084 Chain83: 83:19954 Chain84: 84:18756 Chain85: 85:11840 Chain86: 86:49

选择Microsoft Visual Studio 调试控制台
delete:490
delete:491
delete:492
delete:493
delete:494
delete:495
delete:496
delete:497
delete:498
delete:499
delete done
show all:
Chain0: nothing Chain1: nothing Chain2: nothing Chain3: nothing Chain4: nothing Chain5: nothing Chain6: nothing Chain7:
nothing Chain8: nothing Chain9: nothing Chain10: nothing Chain11: nothing Chain12: nothing Chain13: nothing Chain14: not
hing Chain15: nothing Chain16: nothing Chain17: nothing Chain18: nothing Chain19: nothing Chain20: nothing Chain21: noth
ing Chain22: nothing Chain23: nothing Chain24: nothing Chain25: nothing Chain26: nothing Chain27: nothing Chain28: nothi
ng Chain29: nothing Chain30: nothing Chain31: nothing Chain32: nothing Chain33: nothing Chain34: nothing Chain35: nothi
ng Chain36: nothing Chain37: nothing Chain38: nothing Chain39: nothing Chain40: nothing Chain41: nothing Chain42: nothing
Chain43: nothing Chain44: nothing Chain45: nothing Chain46: nothing Chain47: nothing Chain48: nothing Chain49: nothing
Chain50: nothing Chain51: nothing Chain52: nothing Chain53: nothing Chain54: nothing Chain55: nothing Chain56: nothing C
hain57: nothing Chain58: nothing Chain59: nothing Chain60: nothing Chain61: nothing Chain62: nothing Chain63: nothing Ch
ain64: nothing Chain65: nothing Chain66: nothing Chain67: nothing Chain68: nothing Chain69: nothing Chain70: nothing Cha
in71: nothing Chain72: nothing Chain73: nothing Chain74: nothing Chain75: nothing Chain76: nothing Chain77: nothing Cha
in78: nothing Chain79: nothing Chain80: nothing Chain81: nothing Chain82: nothing Chain83: nothing Chain84: nothing Chain
85: nothing Chain86: nothing Chain87: nothing Chain88: nothing Chain89: nothing Chain90: nothing Chain91: nothing Chain9
2: nothing Chain93: nothing Chain94: nothing Chain95: nothing Chain96: nothing Chain97: nothing Chain98: nothing Chain99
: nothing Chain100: nothing Chain101: nothing Chain102: nothing Chain103: nothing Chain104: nothing Chain105: nothing Ch
ain106: nothing Chain107: nothing Chain108: nothing Chain109: nothing Chain110: nothing Chain111: nothing Chain112: noth
ing Chain113: nothing Chain114: nothing Chain115: nothing Chain116: nothing Chain117: nothing Chain118: nothing Chain119
: nothing Chain120: nothing Chain121: nothing Chain122: nothing Chain123: nothing Chain124: nothing Chain125: nothing Ch
```

2. 平台提交

1. 线性开型寻址

#	状态	耗时	内存占用
#1	✓ Accepted ⓘ	3ms	328.0 KiB
#2	✓ Accepted ⓘ	3ms	332.0 KiB
#3	✓ Accepted ⓘ	2ms	328.0 KiB
#4	✓ Accepted ⓘ	7ms	384.0 KiB
#5	✓ Accepted ⓘ	5ms	328.0 KiB
#6	✓ Accepted ⓘ	6ms	488.0 KiB
#7	✓ Accepted ⓘ	8ms	456.0 KiB
#8	✓ Accepted ⓘ	6ms	512.0 KiB
#9	✓ Accepted ⓘ	6ms	456.0 KiB
#10	✓ Accepted ⓘ	8ms	468.0 KiB
#11	✓ Accepted ⓘ	7ms	512.0 KiB
#12	✓ Accepted ⓘ	7ms	496.0 KiB
#13	✓ Accepted ⓘ	2ms	328.0 KiB
#14	✓ Accepted ⓘ	3ms	328.0 KiB
#15	✓ Accepted ⓘ	3ms	336.0 KiB
#16	✓ Accepted ⓘ	7ms	416.0 KiB
#17	✓ Accepted ⓘ	6ms	456.0 KiB
#18	✓ Accepted ⓘ	6ms	464.0 KiB
#19	✓ Accepted ⓘ	7ms	396.0 KiB
#20	✓ Accepted ⓘ	12ms	464.0 KiB



## 2. 链表散列:

#	状态	耗时	内存占用
#1	✓ Accepted ☺	3ms	336.0 KiB
#2	✓ Accepted ☺	2ms	328.0 KiB
#3	✓ Accepted ☺	2ms	328.0 KiB
#4	✓ Accepted ☺	4ms	464.0 KiB
#5	✓ Accepted ☺	3ms	384.0 KiB
#6	✓ Accepted ☺	4ms	384.0 KiB
#7	✓ Accepted ☺	4ms	388.0 KiB
#8	✓ Accepted ☺	4ms	464.0 KiB
#9	✓ Accepted ☺	5ms	384.0 KiB
#10	✓ Accepted ☺	4ms	512.0 KiB
#11	✓ Accepted ☺	5ms	456.0 KiB
#12	✓ Accepted ☺	6ms	500.0 KiB
#13	✓ Accepted ☺	2ms	384.0 KiB
#14	✓ Accepted ☺	2ms	336.0 KiB
#15	✓ Accepted ☺	3ms	328.0 KiB
#16	✓ Accepted ☺	7ms	464.0 KiB
#17	✓ Accepted ☺	6ms	384.0 KiB
#18	✓ Accepted ☺	5ms	456.0 KiB
#19	✓ Accepted ☺	5ms	464.0 KiB
#20	✓ Accepted ☺	6ms	384.0 KiB

## 4. 分析与探讨（结果分析，若存在问题，探讨解决问题的途径）

本实验最终结果正确，在实验过程中有以下问题或心得：

1. 使用 OJ 给出的样例运行正常，但 OJ 就是通不过，后来添加了更多测试样例才开始正常 debug。
2. 在线性开型寻址中有个公有函数 `getPosByKey` 与私有函数 `_getPosByKey`，某些函数错误地将 `_getPosByKey` 调用成了 `getPosByKey`。导致 debug 很久。
3. 后来又发现线性开型寻址 `erase` 函数在删除元素后将后续元素前移的逻辑出现了错误。
4. 即使正确前移，在 oj 上提交还是错误，后来发现是 `delete` 之后没有将 `size` 减一，导致用到 `size` 的函数运行出错。
5. 进行总结，发现上述 bug 的出现是因为没有提前注意的缘故，考虑到我没办法听课，应当提前在网络上搜索了解各种易出错的坑。
6. 链表散列的散列表比线性开型寻址的散列表简单一些，由此可见，基本结构简单，实现功能的代码不一定简单。算法与数据结构是相辅相成的。

## 5. 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）

### 文件 1 hashchain.h

```
# pragma once
#include<iostream>

/*对应各种数据类型的哈希函数，此处只写 int 的哈希函数*/
template <class K> class myhash;
template<>
class myhash<int> {
public:
    int operator()(const int keyin) const { return int (keyin); }
};

/*用于存储键值的类*/
template <class K, class E>
class mypair {
public:
    K key;
    E data;
    mypair (K keyin, E datain) :key (keyin), data (datain) {}
};
```

```

};

/*用于构成链表的结点类*/
template <class K, class E>
struct mynode {
    mypair< K, E> element;
    mynode< K, E>* next;
    mynode (mypair< K, E> pairin) :element (pairin), next (nullptr) {}
    mynode (mypair< K, E> pairin, mynode< K, E>* nextin) :element (pairin), next (nextin) {}
};

/*链表类
template<class K, class E>
class myChain {
protected:
    mynode<K, E>* _head;                //保存链表头
    int _size;                          //保存元素个数
public:
    myChain ();                          //构造函数
    ~myChain ();                         //析构函数
    bool empty () const;                //返回是否为空
    int size () const;                  //返回元素数目
    mypair< K, E>* find (const K& keyin) const; //查找元素, 找不到返回空指针
    bool erase (const K& keyin);         //删除元素, 无此元素返回 false, 删除成功返回 true
    void insert (mypair< K, E>& pairin); //插入函数, 在插入时确保有序
    void output (ostream& out) const;   //输出链表元素
*/
template<class K, class E>
class myChain {
protected:
    mynode<K, E>* _head;
    int _size;
public:
    myChain () :_head (nullptr), _size (0) {}
    ~myChain () {
        while (_head != nullptr) {
            mynode<K, E>* nextNode = _head->next;
            delete _head;
            _head = nextNode;
        }
    }
    bool empty () const { return _size == 0; }
    int size () const { return _size; }
    mypair< K, E>* find (const K& keyin) const {
        mynode<K, E>* c_node = _head;

        /*一直寻找直到找到对应元素*/
        for (;;) {
            if (c_node == nullptr || c_node->element.key == keyin) {
                break;
            }
            c_node = c_node->next;
        }

        /*如果值不为空, 则*/
        if (c_node != nullptr && c_node->element.key == keyin) {
            return &c_node->element;
        } else {
            return nullptr;
        }
    }
    bool erase (const K& keyin) {
        mynode<K, E>* nodep = _head;
        mynode<K, E>* pre_nodep = nullptr;

        while (nodep != nullptr && nodep->element.key < keyin) {
            pre_nodep = nodep;
            nodep = nodep->next;
        }

        if (nodep != nullptr && nodep->element.key == keyin) {
            if (pre_nodep == nullptr) _head = nodep->next;
            else pre_nodep->next = nodep->next;
            delete nodep;
            _size--;
            return true;
        } else {
            return false;
        }
    }
}

/*插入函数, 在插入时确保有序*/
void insert (mypair< K, E>& thePair) {
    mynode<K, E>* nodep = _head;

```

```

mynode<K, E>* pre_nodep = nullptr;
while (nodep != nullptr && nodep->element.key < thePair.key) {
    pre_nodep = nodep;
    nodep = nodep->next;
}
/*查看是否已有此元素*/
if (nodep != nullptr && nodep->element.key == thePair.key) {
    //已有则插入则修改该位置元素
    nodep->element.data = thePair.data;
    return;
} else {
    //无则插入新元素
    mynode<K, E>* newNode = new mynode<K, E> (thePair, nodep);

    if (pre_nodep == nullptr) _head = newNode;
    else pre_nodep->next = newNode;

    _size++;
    return;
}
}
/*输出链表元素*/
void output (std::ostream& out) const {
    mynode<K, E>* c_node = _head;
    while (c_node != nullptr) {
        out << c_node->element.key << ":" << c_node->element.data << " ";
        c_node = c_node->next;
    }
}

};

/*输出链表元素的重载函数*/
template <class K, class E>
std::ostream& operator<<(std::ostream& out, const myChain<K, E>& in) {
    in.output (out);
    return out;
}

/*哈希链表类
template<class K, class E>
class myhashChains {
protected:
    myChain<K, E>* _chains;           //我的链表
    myhash<K> _myhash;               //自定义的哈希函数
    int _size;                       //元素个数
    int _divisor;                    //除数，链表的个数
    int inline _getPosByKey (const K& keyin)const; //获取该关键词的理论位置，注意该返回值需要配个各
函数进行分别分析
public:
    myhashChains (int divisorin = 20); //构造函数
    ~myhashChains ();                 //析构函数
    bool empty () const;              //返回是否为空
    int size () const;                //返回元素个数
    E* find (const K& keyin) const;    //查找关键词，返回值的指针
    bool insert (mypair<K, E> pairin); //插入键值对，返回插入成功失败与否
    bool erase (const K& keyin);       //删除元素的包装
    void output (ostream& out) const;  //输出元素
    int getLengthByKey (const K& keyin) const; //通过关键词获取长度，为了 OJ 而增加
*/

template<class K, class E>
class myhashChains {
protected:
    myChain<K, E>* _chains;
    myhash<K> _myhash;
    int _size;
    int _divisor;
    int inline _getPosByKey (const K& keyin)const {
        return _myhash (keyin) % _divisor;
    }
public:
    myhashChains (int divisorin = 20) {
        _size = 0;
        _divisor = divisorin;
        _chains = new myChain<K, E>[_divisor];
    }
    ~myhashChains () { delete[] _chains; }

    /*查找关键词，返回值的指针*/
    E* find (const K& keyin) const {
        mypair< K, E>* temp = (_chains[_getPosByKey (keyin)].find (keyin));
        if (temp == nullptr) {
            return nullptr;
        }
    }
}

```

```

    } else {
        return &(temp->data);
    }
}

/*插入键值对，返回插入成功失败与否*/
bool insert (mypair<K, E> pairin) {
    int pos = _getPosByKey (pairin.key);
    int size = _chains[pos].size ();
    _chains[pos].insert (pairin);

    /*链表长度扩大说明插入了，没有扩大说明覆盖了*/
    if (_chains[pos].size () > size) {
        _size++;
        return true;
    } else {
        return false;
    }
}

/*删除元素的包装*/
bool erase (const K& keyin) { return _chains[_getPosByKey (keyin)].erase (keyin); }

/*输出元素*/
void output (std::ostream& out) const {
    for (int i = 0; i < _divisor; i++) {
        if (_chains[i].size () == 0) {
            std::cout << "Chain" << i << ": nothing ";
        } else {
            std::cout << "Chain" << i << ":" << _chains[i] << " ";
        }
    }
}

/*通过关键词获取长度，为了0J而增加*/
int getLengthByKey (const K& keyin) const {
    return _chains[_getPosByKey (keyin)].size ();
}

bool empty () const { return _size == 0; }
int size () const { return _size; }
};

/*输出重载*/
template <class K, class E>
std::ostream& operator<<(std::ostream& out, const myhashChains<K, E>& in) {
    in.output (out);
    return out;
}

```

---

## 文件 2 hashchaintest.cpp

---

```

#include<iostream>
#include"hashchain.h"
using namespace std;
#define LGRAND(min,max) ((rand()%(max-min+(int)1))+(int)min )

int main () {
    myhashChains<int,int> chain(961);

    /*随机获取 50 个不重复的数字*/
    for (int i = 0; i < 500; i++) {
        mypair<int, int> temp (i, LGRAND (0, 50000));
        if (chain.find (temp.data) == nullptr) {
            chain.insert (temp);
            i++;
        }
    }
    cout << "find:\n";
    for (int i = 0; i < 500; i++) {
        cout << "find:" << i << " " << *chain.find(i) << endl;
    }
    cout << "delete:\n";
    for (int i = 0; i < 500; i++) {
        chain.erase (i);
        cout << "delete:" << i << "\n";
    }
    cout << "delete done\n";

    cout << endl << endl << chain << endl << endl;
    return 0;
}

```

---



### 文件 3 hashtable.h

```
# pragma once
#include<iostream>

/*对应各种数据类型的哈希函数，此处只写 int 的哈希函数*/
template <class K> class myhash;
template<>
class myhash<int> {
public:
    int operator()(const int theKey) const {
        return int (theKey);
    }
};

/*用于存储键值的类*/
template <class K, class E>
class mypair {
public:
    K key;
    E data;
    mypair (K keyin, E datain) :key (keyin), data (datain) {}
};

/*template<class E, class K>
class myhashTable {
public:
    struct pair;
protected:
    myhash<K> myhashf;
    int _divisor; //散列函数的除数
    pair** _table_head; //散列数组
    int _getPosByKey (const K& keyin) const; //返回下标，不含有返回-1
public:
    myhashTable (int divisorin = 20); //
    int insertGetPos (const K& keyin, const E& datain); //元素存在则返回-1，不出在则返回插入后的下
标
    int deleteGetNum (const K& keyin); //通过关键词删除键值对，返回移动次数或
-1*
    friend std::ostream& operator << (std::ostream& out, myhashTable& in)
    int getPosByKey (const K& keyin) const; //通过关键词获取该元素的理论位置
*/
template<class E, class K>
class myhashTable {
//public:
//    typedef struct mypair<K,E> {
//        K key;
//        E data;
//        mypair<K,E> (K keyin, E datain) :key (keyin), data (datain) {}
//    }mypair<K,E>;
protected:
    myhash<K> myhashf;
    int _divisor; //散列函数的除数
    mypair<K,E>** _table_head; //散列数组
    /*返回下标，不含有返回-1*/
    int _getPosByKey (const K& keyin) const {
        int i = keyin % _divisor; //找 k 对应散列表的位置
        int j = i;
        do {
            if (!(_table_head[j] != nullptr && _table_head[j]->key != keyin)) return j; //找
到或者为空，则返回
            j = (j + 1) % _divisor;
        } while (j != i); //若又回到 i，则退出循环
        return j;
    }
public:
    int getPosByKey (const K& keyin) const {
        int pos = _getPosByKey (keyin);
        if (_table_head[pos] == nullptr) {
            return -1;
        } else {
            return pos;
        }
    }
    E* find (K& keyin) {
        int temp = getPosByKey (keyin);
        if (temp== -1) {
            return nullptr;
        } else {
            return &(_table_head[temp]->data);
        }
    }
};
```

```

}
myhashTable (int divisorin = 20) :_divisor (divisorin) {
    table_head = new mypair<K,E> * [_divisor];
    for (int i = 0; i < _divisor; i++) {
        _table_head[i] = nullptr;
    }
}
~myhashTable () {
    for (int i = 0; i < _divisor; i++) {
        delete[] _table_head;
    }
}
int erase (const K& keyin) {
    return deleteGetNum (keyin);
}
/*通过关键词删除键值对，返回移动次数或-1*/
int deleteGetNum (const K& keyin) {
    int delete_pos = _getPosByKey (keyin);
    if (_table_head[delete_pos] == nullptr) {
        return -1;
    } else if (_table_head[delete_pos]->key == keyin) {
        _table_head[delete_pos] = nullptr; //内存溢出

        int current_pos = delete_pos; //当前元素当前的实际位置
        int last_pos = delete_pos; //
        int original_pos; //当前元素一般情况下本应该在的位置

        int move_count = 0;
        for (;;) {
            current_pos++; //将下一个元素设置为当前元素
            current_pos = current_pos % _divisor; //当前元素的实际位置
            if (_table_head[current_pos] == nullptr) { //当前桶为空或，为空说明前面的元素都
                //在正确位置，直接删除并返回
                return move_count;
            } else {
                original_pos = myhashf (_table_head[current_pos]->key) % _divisor; //当前元
                //素本应该在哪里
                if (current_pos != _getPosByKey (_table_head[current_pos]->key)) {
                    _table_head[last_pos] = _table_head[current_pos]; //移动指针，无需 delete
                    _table_head[current_pos] = nullptr; //内存溢出
                    move_count++;
                    last_pos = current_pos;
                } else {
                    continue;
                }
            }
        }
    }
}

/*元素存在则返回-1，不出在则返回插入后的下标*/
int insertGetPos (const K& keyin, const E& datain) {
    int b = _getPosByKey (keyin);
    if (_table_head[b] == nullptr) {
        _table_head[b] = new mypair<K,E> (keyin, datain);
        return b;
    } else {
        return -1;
    }
}
int insert (mypair<K,E>& pairin) {
    return insertGetPos (pairin.key, pairin.data);
}
friend ostream& operator << (std::ostream& out, myhashTable& in) {
    for (int i = 0; i < in._divisor; i++) {
        if (in._table_head[i] == nullptr) {
            out << "null" << " ";
        } else {
            out << in._table_head[i]->key << " ";
        }
    }
    return out;
}
};

```

文件 4 hashtable.cpp

```

#include<iostream>
#include"hashtable.h"

```

```
using namespace std;
#define LGRAND(min,max) ((rand()%(max-min+(int)1))+(int)min )

int main () {
    myhashTable<int, int> table (961);

    /*随机获取 50 个不重复的数字*/
    for (int i = 0; i < 500;) {
        mypair<int, int> temp (i, LGRAND (0, 50000));
        if (table.find (temp.data) == nullptr) {
            table.insert (temp);
            i++;
        }
    }
    cout << "find:\n";
    for (int i = 0; i < 500; i++) {
        cout << "find:" << i << " " << *table.find (i) << endl;
    }
    cout << "delete:\n";
    for (int i = 0; i < 500; i++) {
        table.erase (i);
        cout << "delete:" << i << "\n";
    }
    cout << "delete done\n";

    cout << endl << endl << table << endl << endl;
    return 0;
}
```

---