



Assignment3

课程： 算法设计与分析

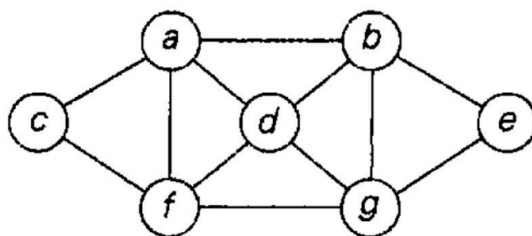
姓名： 雷翔

学号： 2053932

时间： 2023 年 5 月

Q1. 请用回溯法对下图求哈密顿回路问题 (从 a 点开始), 请详细给出解空间树, 搜索过程及结果。

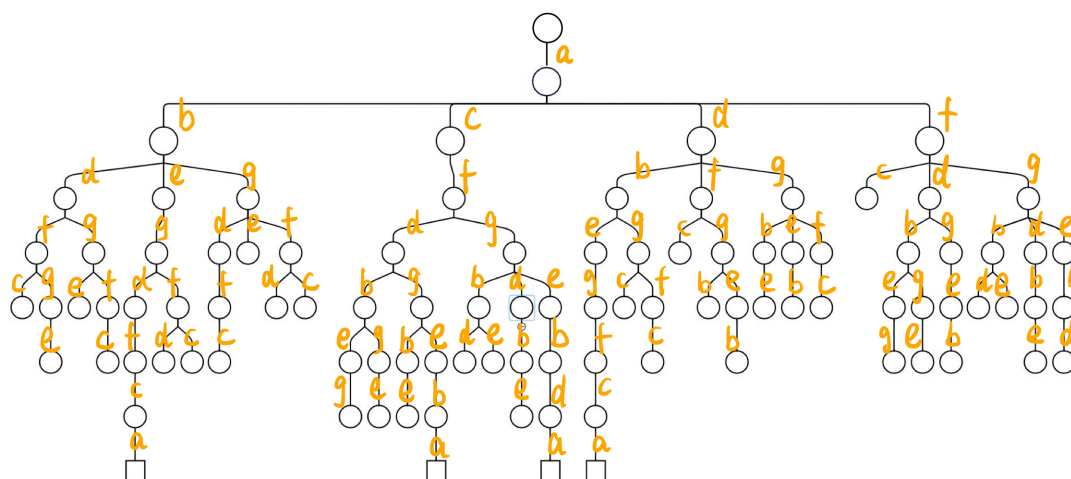
Ans.



1. 定义解空间

$X = \{abcdegfa, abcdegfa, \dots, agfedcba\}$

2. 解空间树



3. 从 a 出发按 DFS 搜索整棵树

a -> b -> d -> f -> c

回溯到 f

f -> g -> e

回溯到 d

d -> g -> e

回溯到 g

g -> f -> c

回溯到 b

b -> e -> g -> d -> f -> c -> a 得到第一个解

其余解同理可得

Q2. 找零问题: 给予币值为 1,3,5 的硬币若干 (每种硬币个数无限多), 如何用这些硬币组合, 使得面额为 9, 且硬币的个数最少? 请利用动态规划算法求其所有解。请给出算法伪代码并参考教材详细描述算法的运行过程。

Ans. 面值为 6 的组合的硬币最小个数可以由面值为 5、3、2 的组合的硬币最小个数中的最小值加 1 计算。

记面额为 j 的硬币组合个数为 $dp[j]$

初始条件: $dp[0] = 0$

状态方程:

$$f(x) = \begin{cases} \min\{dp[j-1]\} + 1 & 1 \leq j < 3 \\ \min\{dp[j-1], dp[j-3]\} + 1 & 3 \leq j < 5 \\ \min\{dp[j-1], dp[j-3], dp[j-5]\} + 1 & j \geq 5 \end{cases} \quad (1)$$

表 1: 面额与最小硬币个数的对应关系

j	0	1	2	3	4	5	6	7	8	9
dp[j]	0	1	2	1	2	1	2	3	2	3

$$dp[1] = dp[0] + 1 = 1$$

$$dp[2] = dp[1] + 1 = 2$$

$$dp[3] = \min\{dp[2], dp[0]\} + 1 = 1$$

$$dp[4] = \min\{dp[3], dp[1]\} + 1 = 2$$

$$\text{依次类推, } dp[9] = \min\{dp[8], dp[6], dp[4]\} + 1 = 3$$

组合: 3 个币值为 3 元的硬币或 1 个币值为 1 元 + 1 个币值为 3 元 + 1 个币值为 5 元

伪代码:

```
// 求面额为 n 的硬币最小个数
function dp(n) {
    for i <- 0 to n // 得到一个大小为 n + 1, 值为 0 的数组
        do dp[i] <- 0
    for j <- 1 to n do
        if j >= 1 && j < 3
            dp[j] = dp[j-1] + 1
        else if j >= 3 && j < 5
            dp[j] = min(dp[j-1], dp[j-3]) + 1
        else
```

```

        dp[j] = min(dp[j-1], dp[j-3], dp[j-5]) + 1

    return dp // 返回结果数组
}

```

Q3. 请用分支限界法对背包问题的以下示例求解，请详细给出解空间树，搜索过程及结果。

Ans.

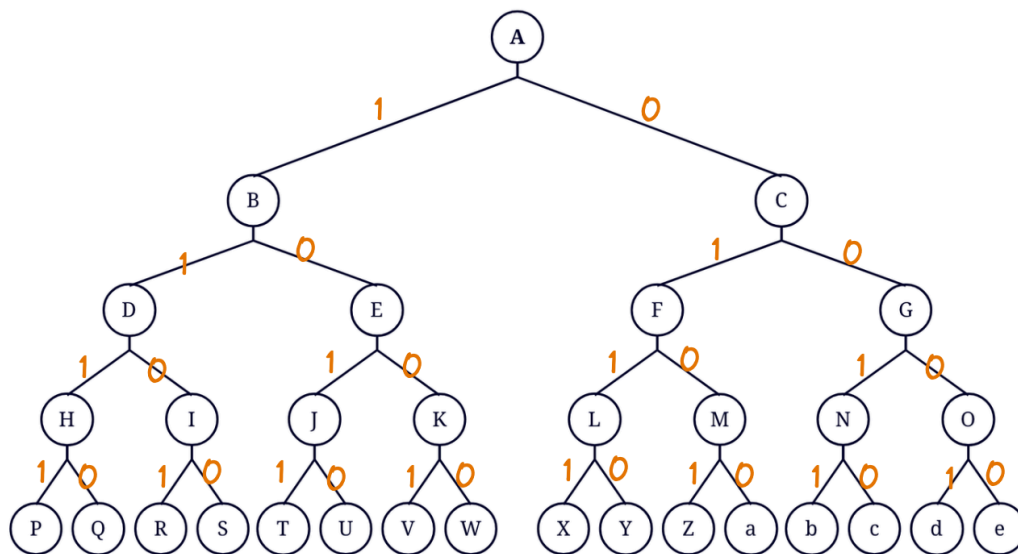
物 品	重 量	价值/美元
1	10	100
2	7	63
3	8	56
4	4	12

$W = 16$

1. 定义解空间

$X = \{(0, 0, 0, 0), (0, 0, 0, 1), \dots, (1, 1, 1, 1)\}$

2. 定义解空间树



3. 按 BFS 方式进行搜索，采用优先队列

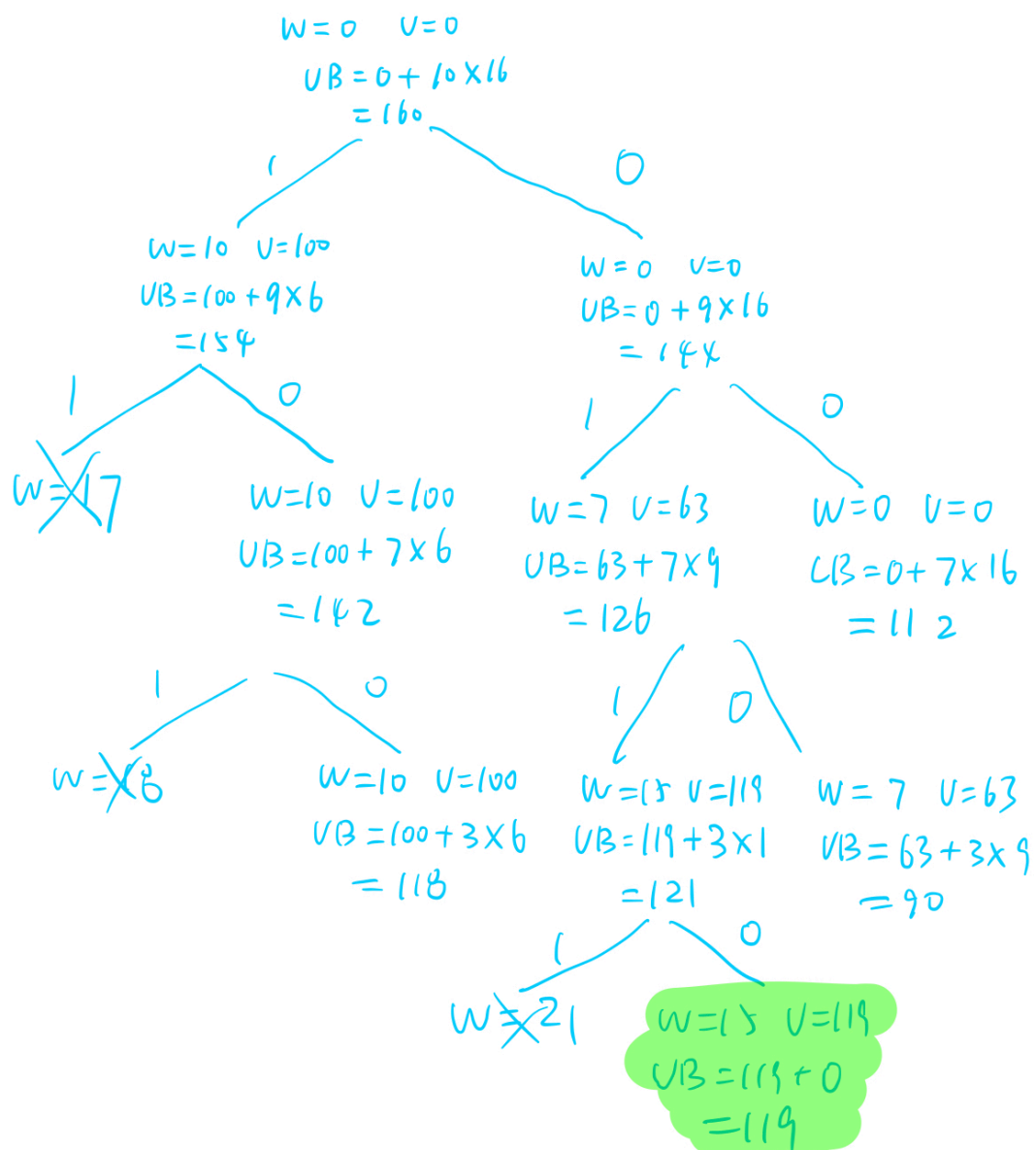
优先级取决于背包中已经装袋的物品价值加上剩下的最大价值物品装满袋的价值总和

w: 袋子里的重量

v: 袋子里物品的价值

UB: 上界

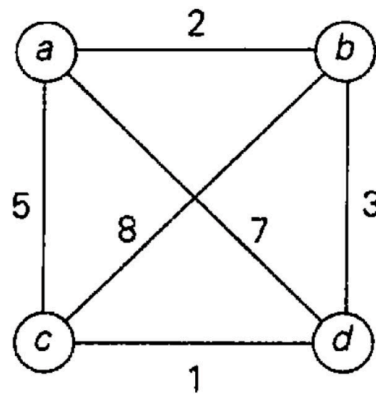
搜索过程:



结果: 选择物品 2 和物品 3, 最大价值为 119

Q4. 对于图 3，应用分支限界法求解从 a 点开始的 TSP 问题，请详细给出解空间树，搜索过程及最优解。

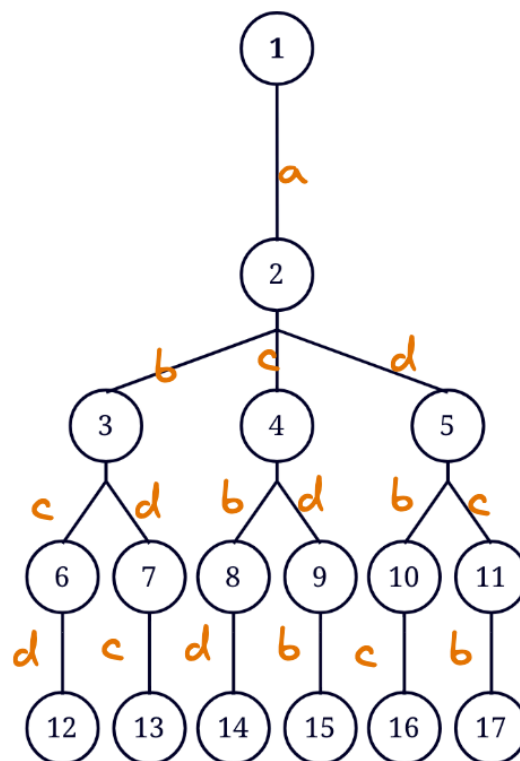
Ans.



1. 定义解空间

$X = \{abcda, abdca, acbda, acdba, adbca, adcba\}$

2. 构造解空间树



3. 按 BFS 方式进行搜索，采用优先队列

搜索过程：

Handwritten search process showing a sequence of numbers and their superscripts, likely representing a state space or a sequence of moves. The sequence starts with 2 and branches out, with some numbers circled at the end, indicating the final state or a specific path.

结果：a -> b -> d -> c -> a 和 a -> c -> d -> b -> a

最短路径为 11

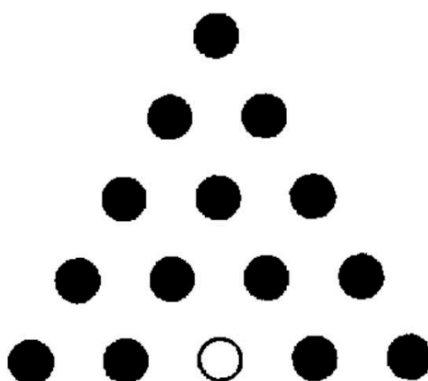
Q5. 在一个类似谜题的游戏中，等边三角形的板上布置了 15 个孔。在初始的时候，如下图所示，除了一个孔，所有孔都插上了插棒。一个插棒可以跳过它的直接邻居，移到一个空白的位上。这一跳会把被跳过的邻居从板上移走。

请使用回溯算法，描述求解该谜题的下列版本的主要思路并给出算法的伪代码：

a 已知空孔的位置，求出消去 13 个棒的最短步骤，对剩下的插棒的最终位置不限。

b 已知空孔的位置，求出消去 13 个棒的最短步骤，剩下的插棒最终要落在最初的空孔上。

Ans.



a.

首先初始化状态，主要包括定义棋盘数组，表示棋盘状态，用 1 表示有插棒，0 表示空孔，记录当前空孔位置的变量以及一个记录最短步数的变量。

从当前空孔位置开始，依次向四周能跳跃的方向跳跃，并将路径上的插棒移除。如果已经消去了 13 个插棒，就更新最短步数变量，如果当前步数超过最短步数变量，就回溯到上一步。

```
// 从 from 跳到 to, 中间经过 in
// empty_hole 表示当前空孔的位置
function backtrack(board, empty_hole, removed_count, step) {
    if removed_count == 13 { // 消除 13 个棒
        if step < min_step {
            min_step = step
        }
        return
    }
    for each in 临近的插棒 {
        如果将跳跃到的位置是空孔empty_hole { // 对应 to 位置
            将跳过的插棒移除 // 对应 in 位置
            backtrack(board, 跳过的位置in, removed_count + 1, step + 1)
            恢复跳过的插棒in // 对应 in 位置
        }
    }
}
```

b.

在 a 的基础上，保留下最初的空孔位置。在消除 13 个棒后，判断当前空孔位置是否为最初的空孔位置，如果是，就更新最短步数变量，否则回溯到上一步。

```
// 从 from 跳到 to, 中间经过 in
// empty_hole 表示当前空孔的位置
// target_hole 表示最终要落在的空孔的位置
function backtrack(board, empty_hole, removed_count, step) {
    if removed_count == 13 and empty_hole == target_hole { // 消除 13 个棒, 且
        剩下的插棒落在最初的空孔
        if step < min_step {
            min_step = step
        }
        return
    }
    for each in 临近的插棒 {
```



```
    如果将跳跃到的位置是空孔empty_hole { // 对应 to 位置
        将跳过的插棒移除 // 对应 in 位置
        backtrack(board, 跳过的位置in, removed_count + 1, step + 1)
        恢复跳过的插棒in // 对应 in 位置
    }
}
}
```

编程题

请使用 C++11 及以上编译 main.cpp

```
1 | g++ main.cpp --std=c++11
```

算法思路

遍历所有单元格，对于黄金数量为 0 的单元格，跳过，否则以该单元格为起点，进行探索。每次探索中，回更新最大黄金数量 max_gold。

在以(i, j)为起点的探索中，从(i, j)开始，依次向上右下左四个方向进行探索，如果搜索到的单元格未被访问过且有黄金，则将当前黄金数量加上该单元格的黄金数量，并继续向四个方向进行搜索。如果搜索到的单元格已经被访问过或者没有黄金，则回溯到上一个单元格。最终返回最大的黄金数量。

这里并不需要减去之前加上的单元格的黄金数量，因此当前黄金数量 current_gold 是函数形参，不会对上一层的 current_gold 产生影响。

算法时空复杂度

m = grid.size()

n = grid[0].size()

k 表示有黄金的单元格的数量

两层 for 循环遍历存在黄金数量的单元格，对应时间复杂度 $O(mn)$ ，其中有 k 个有黄金的单元格，显然 $k < mn$ 。对于每一个有黄金的单元格，作为起点开始遍历，除了第一步有 3 个方向可以试探外，其余均最多有 3 个方向可以试探。因此，每个单元格遍历对应时间复杂度 $O(3^k)$ 。

因此，最终时间复杂度为 $O(k * 3^k)$

时间复杂度： $O(k \times 3^k)$

因为我们维护了两个大小为 mxn 的二维数组，因此空间复杂度为 $O(mn)$

主要函数

```
1 | void backtrack(vector<vector<int>> &grid, int i, int j, int  
   | current_gold, vector<vector<int>> &visited);  
2 | int getMaximumGold(vector<vector<int>> &grid);
```

```

12 int getMaximumGold(vector<vector<int>> &grid)
13 {
14     for (int i = 0; i < grid.size(); i++)
15     {
16         for (int j = 0; j < grid[0].size(); j++)
17         {
18             if (grid[i][j] == 0) // 该单元格黄金数量为 0 跳过
19                 continue;
20
21             vector<vector<int>> visited(grid.size(), vector<int>(grid[0].size(), 0)); // 初始化一个二维数组，用于记录已访问过的单元格
22             backtrack(grid, i, j, 0, visited); // 回溯
23         }
24     }
25     return max_gold;
26 }

```

```

/**
 * @brief 回溯
 */
void backtrack(vector<vector<int>> &grid, int i, int j, int current_gold, vector<vector<int>> &visited)
{
    // 越界 或 已访问 或 该单元格黄金数量为 0
    if (i < 0 || i >= grid.size() || j < 0 || j >= grid[0].size() || visited[i][j] == 1 || grid[i][j] == 0)
    {
        if (current_gold > max_gold)
            max_gold = current_gold;
        return;
    }

    visited[i][j] = 1;
    current_gold += grid[i][j]; // 当前黄金数量加上该单元格黄金数量

    backtrack(grid, i - 1, j, current_gold, visited); // 上
    backtrack(grid, i, j + 1, current_gold, visited); // 右
    backtrack(grid, i + 1, j, current_gold, visited); // 下
    backtrack(grid, i, j - 1, current_gold, visited); // 左

    visited[i][j] = 0;
}

```

运行结果示例

```

> g++ main.cpp --std=c++11
> ./a.out
grid:
  0 6 0
  5 8 7
  0 9 0
max_gold: 24

```

```

> g++ main.cpp --std=c++11
> ./a.out
grid:
  1 0 7
  2 0 6
  3 4 5
  0 3 0
  9 0 20
max_gold: 28

```

```
> g++ main.cpp --std=c++11
> ./a.out
grid:
  1 1 1
  1 1 1
  1 1 1
max_gold: 9
```

力扣测试

题目链接: <https://leetcode.cn/problems/path-with-maximum-gold/>

题目描述

讨论 (353)

题解 (523)

提交记录

所有状态

所有语言

通过

1 分钟前

C++

通过

2023.05.13

C++

超出时间限制

2023.05.11

C++

超出时间限制

2023.05.11

C++

通过

2023.05.11

C++

解答错误

2023.05.11

C++

通过

2023.05.11

C++

通过

2023.05.11

C++

C++

智能模式

1 class Solution {

2 public:

3 int max_gold; // 最大黄金数量

4

5 int getMaximumGold(vector<vector<int>> &grid)

6 {

7 for (int i = 0; i < grid.size(); i++)

8 {

9 for (int j = 0; j < grid[0].size(); j++)

10 {

11 if (grid[i][j] == 0) // 该单元格黄金数量为 0 跳过

12 continue;

13

14 vector<vector<int>> visited(grid.size(), vector<int>(grid[0].size(), 0)); //

15 // 初始化一个二维数组, 用于记录已访问过的单元格

16 backtrack(grid, i, j, 0, visited);

17 // 回溯

18 }

19 return max_gold;

20 }

21 /**

22 * @brief 回溯

23 */

24 void backtrack(vector<vector<int>> &grid, int i, int j, int current_gold,

25 vector<vector<int>> &visited)

26 {

27 // 越界 或 已访问 或 该单元格黄金数量为 0

28 if (i < 0 || i >= grid.size() || j < 0 || j >= grid[0].size() || visited[i][j] == 1

29 || grid[i][j] == 0)

30 {

31 if (current_gold > max_gold)

32 }

33 }

34 }

35 }

36 }

37 }

38 }

39 }

40 }

41 }

42 }

43 }

44 }

45 }

46 }

47 }

48 }

49 }

50 }

51 }

52 }

53 }

54 }

55 }

56 }

57 }

58 }

59 }

60 }

61 }

62 }

63 }

64 }

65 }

66 }

67 }

68 }

69 }

70 }

71 }

72 }

73 }

74 }

75 }

76 }

77 }

78 }

79 }

80 }

81 }

82 }

83 }

84 }

85 }

86 }

87 }

88 }

89 }

90 }

91 }

92 }

93 }

94 }

95 }

96 }

97 }

98 }

99 }

100 }

101 }

102 }

103 }

104 }

105 }

106 }

107 }

108 }

109 }

110 }

111 }

112 }

113 }

114 }

115 }

116 }

117 }

118 }

119 }

120 }

121 }

122 }

123 }

124 }

125 }

126 }

127 }

128 }

129 }

130 }

131 }

132 }

133 }

134 }

135 }

136 }

137 }

138 }

139 }

140 }

141 }

142 }

143 }

144 }

145 }

146 }

147 }

148 }

149 }

150 }

151 }

152 }

153 }

154 }

155 }

156 }

157 }

158 }

159 }

160 }

161 }

162 }

163 }

164 }

165 }

166 }

167 }

168 }

169 }

170 }

171 }

172 }

173 }

174 }

175 }

176 }

177 }

178 }

179 }

180 }

181 }

182 }

183 }

184 }

185 }

186 }

187 }

188 }

189 }

190 }

191 }

192 }

193 }

194 }

195 }

196 }

197 }

198 }

199 }

200 }

201 }

202 }

203 }

204 }

205 }

206 }

207 }

208 }

209 }

210 }

211 }

212 }

213 }

214 }

215 }

216 }

217 }

218 }

219 }

220 }

221 }

222 }

223 }

224 }

225 }

226 }

227 }

228 }

229 }

230 }

231 }

232 }

233 }

234 }

235 }

236 }

237 }

238 }

239 }

240 }

241 }

242 }

243 }

244 }

245 }

246 }

247 }

248 }

249 }

250 }

251 }

252 }

253 }

254 }

255 }

256 }

257 }

258 }

259 }

260 }

261 }

262 }

263 }

264 }

265 }

266 }

267 }

268 }

269 }

270 }

271 }

272 }

273 }

274 }

275 }

276 }

277 }

278 }

279 }

280 }

281 }

282 }

283 }

284 }

285 }

286 }

287 }

288 }

289 }

290 }

291 }

292 }

293 }

294 }

295 }

296 }

297 }

298 }

299 }

300 }

301 }

302 }

303 }

304 }

305 }

306 }

307 }

308 }

309 }

310 }

311 }

312 }

313 }

314 }

315 }

316 }

317 }

318 }

319 }

320 }

321 }

322 }

323 }

324 }

325 }

326 }

327 }

328 }

329 }

330 }

331 }

332 }

333 }

334 }

335 }

336 }

337 }

338 }

339 }

340 }

341 }

342 }

343 }

344 }

345 }

346 }

347 }

348 }

349 }

350 }

351 }

352 }

353 }

354 }

355 }

356 }

357 }

358 }

359 }

360 }

361 }

362 }

363 }

364 }

365 }

366 }

367 }

368 }

369 }

370 }

371 }

372 }

373 }

374 }

375 }

376 }

377 }

378 }

379 }

380 }

381 }

382 }

383 }

384 }

385 }

386 }

387 }

388 }

389 }

390 }

391 }

392 }

393 }

394 }

395 }

396 }

397 }

398 }

399 }

400 }

401 }

402 }

403 }

404 }

405 }

406 }

407 }

408 }

409 }

410 }

411 }

412 }

413 }

414 }

415 }

416 }

417 }

418 }

419 }

420 }

421 }

422 }

423 }

424 }

425 }

426 }

427 }

428 }

429 }

430 }

431 }

432 }

433 }

434 }

435 }

436 }

437 }

438 }

439 }

440 }

441 }

442 }

443 }

444 }

445 }

446 }

447 }

448 }

449 }

450 }

451 }

452 }

453 }

454 }

455 }

456 }

457 }

458 }

459 }

460 }

461 }

462 }

463 }

464 }

465 }

466 }

467 }

468 }

469 }

470 }

471 }

472 }

473 }

474 }

475 }

476 }

477 }

478 }

479 }

480 }

481 }

482 }

483 }

484 }

485 }

486 }

487 }

488 }

489 }

490 }

491 }

492 }

493 }

494 }

495 }

496 }

497 }

498 }

499 }

500 }

501 }

502 }

503 }

504 }

505 }

506 }

507 }

508 }

509 }

510 }

511 }

512 }

513 }

514 }

515 }

516 }

517 }

518 }

519 }

520 }

521 }

522 }

523 }

524 }

525 }

526 }

527 }

528 }

529 }

530 }

531 }

532 }

533 }

534 }

535 }

536 }

537 }

538 }

539 }

540 }

541 }

542 }

543 }

544 }

545 }

546 }

547 }

548 }

549 }

550 }

551 }

552 }

553 }

554 }

555 }

556 }

557 }

558 }

559 }

560 }

561 }

562 }

563 }

564 }

565 }

566 }

567 }

568 }

569 }

570 }

571 }

572 }

573 }

574 }

575 }

576 }

577 }

578 }

579 }

580 }

581 }

582 }

583 }

584 }

585 }

586 }

587 }

588 }

589 }

590 }

591 }

592 }

593 }

594 }

595 }

596 }

597 }

598 }

599 }

600 }

601 }

602 }

603 }

604 }

605 }

606 }

607 }

608 }

609 }

610 }

611 }

612 }

613 }

614 }

615 }

616 }

617 }

618 }

619 }

620 }

621 }

622 }

623 }

624 }

625 }

626 }

627 }

628 }

629 }

630 }

631 }

632 }

633 }

634 }

635 }

636 }

637 }

638 }

639 }

640 }

641 }

642 }

643 }

644 }

645 }

646 }

647 }

648 }

649 }

650 }

651 }

652 }

653 }

654 }

655 }

656 }

657 }

658 }

659 }

660 }

661 }

662 }

663 }

664 }

665 }

666 }

667 }

668 }

669 }

670 }

671 }

672 }

673 }

674 }

675 }

676 }

677 }

678 }

679 }

680 }

681 }

682 }

683 }

684 }

685 }

686 }

687 }

688 }

689 }

690 }

691 }

692 }

693 }

694 }

695 }

696 }

697 }

698 }

699 }

700 }

701 }

702 }

703 }

704 }

705 }

706 }

707 }

708 }

709 }

710 }

711 }

712 }

713 }

714 }

715 }

716 }

717 }

718 }

719 }

720 }

721 }

722 }

723 }

724 }

725 }

726 }

727 }

728 }

729 }

730 }

731 }

732 }

733 }

734 }

735 }

736 }

737 }

738 }

739 }

740 }

741 }

742 }

743 }

744 }

745 }

746 }

747 }

748 }

749 }

750 }

751 }

752 }

753 }

754 }

755 }

756 }

757 }

758 }

759 }

760 }

761 }

762 }

763 }

764 }

765 }

766 }

767 }

768 }

769 }

770 }

771 }

772 }

773 }

774 }

775 }

776 }

777 }

778 }

779 }

780 }

781 }

782 }

783 }

784 }

785 }

786 }

787 }

788 }

789 }

790 }

791 }

792 }

793 }

794 }

795 }

796 }

797 }

798 }

799 }

800 }

801 }

802 }

803 }

804 }

805 }

806 }

807 }

808 }

809 }

810 }

811 }

812 }

813 }

814 }

815 }

816 }

817 }

818 }

819 }

820 }

821 }

822 }

823 }

824 }

825 }

826 }

827 }

828 }

829 }

830 }

831 }

832 }

833 }

834 }

835 }

836 }

837 }

838 }

839 }

840 }

841 }

842 }

843 }

844 }

845 }

846 }

847 }

848 }

849 }

850 }

851 }

852 }

853 }

854 }

855 }

856 }

857 }

858 }

859 }

860 }

861 }

862 }

863 }

864 }

865 }

866 }

867 }

868 }

869 }

870 }

871 }

872 }

873 }

874 }

875 }

876 }

877 }

878 }

879 }

880 }

881 }

882 }

883 }

884 }

885 }

886 }

887 }

888 }

889 }

890 }

891 }

892 }

893 }

894 }

895 }

896 }

897 }

898 }

899 }

900 }

901 }

902 }

903 }

904 }

905 }

906 }

907 }

908 }

909 }

910 }

911 }

912 }

913 }

914 }

915 }

916 }

917 }

918 }

919 }

920 }

921 }

922 }

923 }

924 }

925 }

926 }

927 }

928 }

929 }

930 }

931 }

932 }

933 }

934 }

935 }

936 }

937 }

938 }

939 }

940 }

941 }

942 }

943 }

944 }

945 }

946 }

947 }

948 }

949 }

950 }

951 }

952 }

953 }

954 }

955 }

956 }

957 }

958 }

959 }

960 }

961 }

962 }

963 }

964 }

965 }

966 }

967 }

968 }

969 }

970 }

971 }

972 }

973 }

974 }

975 }

976 }

977 }

978 }

979 }

980 }

981 }

982 }

983 }

984 }

985 }

986 }

987 }

988 }

989 }

990 }

991 }

992 }

993 }

994 }

995 }

996 }

997 }

998 }

999 }

1000 }

1001 }

1002 }

1003 }

1004 }

1005 }

1006 }

1007 }

1008 }

1009 }

1010 }

1011 }

1012 }

1013 }

1014 }

1015 }

1016 }

1017 }

1018 }

1019 }

1020 }

1021 }

1022 }

1023 }

1024 }

1025 }

1026 }

1027 }

1028 }

1029 }

1030 }

1031 }

1032 }

1033 }

1034 }

1035 }

1036 }

1037 }

1038 }

1039 }

1040 }

1041 }

1042 }

1043 }

1044 }

1045 }

1046 }

1047 }

1048 }

1049 }

1050 }

1051 }

1052 }

1053 }

1054 }

1055 }

1056 }

1057 }

1058 }

1059 }

1060 }

1061 }

1062 }

1063 }

1064 }

1065 }

1066 }

1067 }

1068 }

1069 }

1070 }

1071 }

1072 }

1073 }

1074 }

1075 }

1076 }

1077 }

1078 }

1079 }

1080 }

1081 }

1082 }

1083 }

1084 }

1085 }

1086 }

1087 }

1088 }

1089 }

1090 }

1091 }

1092 }

1093 }

1094 }

1095 }

1096 }

1097 }

1098 }

1099 }

1100 }

1101 }

1102 }

1103 }

1104 }

1105 }

1106 }

1107 }

1108 }

1109 }

1110 }

1111 }

1112 }

1113 }

1114 }

1115 }

1116 }

1117 }

1118 }

1119 }

1120 }

1121 }

1122 }

1123 }

1124 }

1125 }

1126 }

1127 }

1128 }

1129 }

1130 }

1131 }

1132 }

1133 }

1134 }

1135 }

1136 }

1137 }

1138 }

1139 }

1140 }

1141 }

1142 }

1143 }

1144 }

1145 }

1146 }

1147 }

1148 }

1149 }

1150 }

1151 }

1152 }

1153 }

1154 }

1155 }

1156 }

1157 }

1158 }

1159 }

1160 }

1161 }

1162 }

1163 }

1164 }

1165 }

1166 }

1167 }

1168 }

1169 }

1170 }

1171 }

1172 }

1173 }

1174 }

1175 }

1176 }

1177 }

1178 }

1179 }

1180 }

1181 }

1182 }

1183 }

1184 }

1185 }

1186 }

1187 }

1188 }

1189 }

1190 }

1191 }

1192 }

1193 }

1194 }

1195 }

1196 }

1197 }

1198 }

1199 }

1200 }

1201 }

1202 }

1203 }

1204 }

1205 }

1206 }

1207 }

1208 }

1209 }

1210 }

1211 }

1212 }

1213 }

1214 }

1215 }

1216 }

1217 }

1218 }

1219 }

1220 }

1221 }

1222 }

1223 }

1224 }

1225 }

1226 }

1227 }

1228 }

1229 }

1230 }

1231 }

1232 }

1233 }

1234 }

1235 }

1236 }

1237 }

1238 }

1239 }

1240 }

1241 }

1242 }

1243 }

1244 }

1245 }

1246 }

1247 }

1248 }

1249 }

1250 }

1251 }

1252 }

1253 }

1254 }

1255 }

1256 }

1257 }

1258 }

1259 }

1260 }

1261 }

1262 }

1263 }

1264 }

1265 }

1266 }

1267 }

1268 }

1269 }

1270 }

1271 }

1272 }

1273 }

1274 }

1275 }

1276 }

1277 }

1278 }

1279 }

1280 }

1281 }

1282 }

1283 }

1284 }

1285 }

1286 }

1287 }

1288 }

1289 }

1290 }

1291 }

1292 }

1293 }

1294 }

1295 }

1296 }

1297 }

1298 }

1299 }

1300 }

1301 }

1302 }

1303 }

1304 }

1305 }

1306 }

1307 }

1308 }

1309 }

1310 }

1311 }

1312 }

1313 }

1314 }

1315 }

1316 }

1317 }

1318 }

1319 }

1320 }

1321 }

1322 }

1323 }

1324 }

1325 }

1326 }

1327 }

1328 }

1329 }

1330 }

1331 }

1332 }

1333 }

1334 }

1335 }

1336 }

1337 }

1338 }

1339 }

1340 }

1341 }

1342 }

1343 }

1344 }

1345 }

1346 }

1347 }

1348 }

1349 }

1350 }

1351 }

1352 }

1353 }

1354 }

1355 }

1356 }

1357 }

1358 }

1359 }

1360 }

1361 }

1362 }

1363 }

1364 }

1365 }

1366 }

1367 }

1368 }

1369 }

1370 }

1371 }

1372 }

1373 }

1374 }

1375 }

1376 }

1377 }

1378 }

1379 }

1380 }

1381 }

1382 }

1383 }

1384 }

1385 }

1386 }

1387 }

1388 }

1389 }

1390 }

1391 }

1392 }

1393 }

1394 }

1395 }

1396 }

1397 }

1398 }

1399 }

1400 }

1401 }

1402 }

1403 }

1404 }

1405 }

1406 }

1407 }