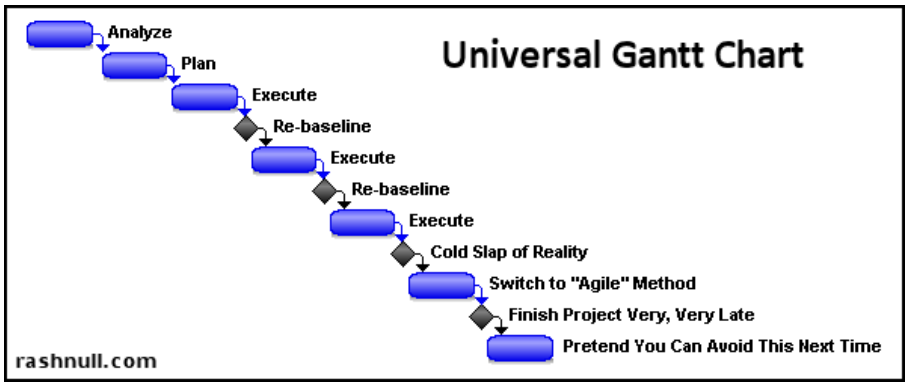
CS 302 – Assignment #11, Final Project: Project Management Support

Purpose: Learn concepts regarding graph algorithms and develop application specific data structures. Learn how graphs apply to real-world problems.
Due: Part A → Wednesday (11/21), Part B → Thursday (11/29)
Points: Part A → 75 pts, Part B → 225 pts

Assignment:

Project management[1] is the practice of initiating, planning, executing, controlling, and closing the work of a team to achieve specific goals and meet specific success criteria at the specified time.

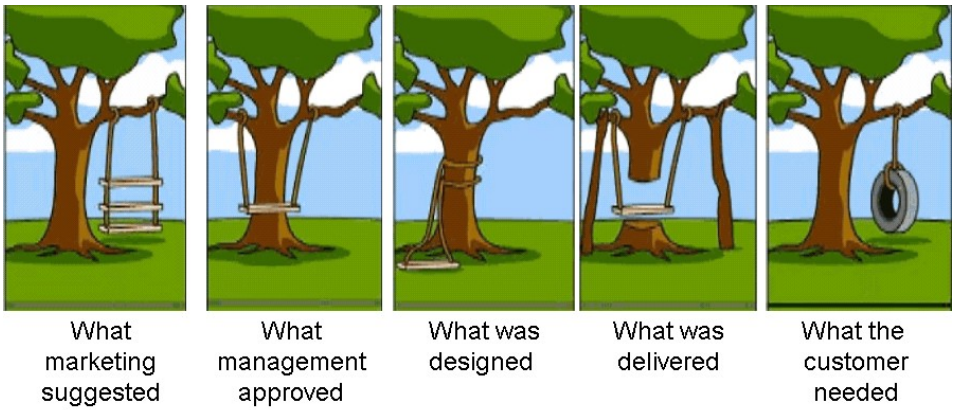


A Gantt chart[2], commonly used in project management, is one of the most popular and useful ways of showing activities displayed against time. On the left of the chart is a list of the activities and along the top is the time scale. Each activity is represented by a bar; the position and length of the bar reflects the start date, duration and end date of the activity.

Design a set of C++11 classes to read, store, and process a large directed acyclic graph[3] representing a Gantt Chart; tasks (edges) and milestones (vertices's). Your design, at a minimum, must provide functionality to:

- Read and store the task (graph) data.
- Project Gantt Information;
  - Process and verify command line arguments.
  - Graph information
    - Project title, tasks, milestones, density, milestones/tasks ratio, source milestone, dependency statistics, key milestone, node point, and verify validity (is DAG).
  - Gantt Information
    - Independent tasks, topological sort[4], determine critical path[5], find the articulation points, find the slack times.

The provided main will call the object functions. You may change the provided main to call the functions as threads to improve performance.



1 For more information, refer to: https://en.wikipedia.org/wiki/Project_management
2 For more information, refer to: https://en.wikipedia.org/wiki/Gantt_chart
3 For more information, refer to: https://en.wikipedia.org/wiki/Directed_acyclic_graph
4 For more information, refer to: https://en.wikipedia.org/wiki/Topological_sorting
5 For more information, refer to: https://en.wikipedia.org/wiki/Critical_path_method

Part A:
Perform the basic design. Your design should address the class hierarchy and applicable data structures. The data structures should be customized and optimized for this problem.

For part A, create and submit a brief write-up including the following:

- Name, Assignment, Section.
- UML diagram of the classes.
  - Please ensure the functions have appropriate names. Provide simple explanations as necessary. Include the applicable constructor's and destructor's.
  - Additional features/functions, as needed.
- Big-Oh for each of the functions called by the provided main.
- Detailed description of data structure for the graph (node type).
- Summary of other data structures proposed with an explanation.

The class information format should be similar to past assignments, showing a UML diagram table with the class variables/functions and the function descriptions (1-2 sentences). In addition, the write-up should include a graph showing the class hierarchy.

There are many options for the data structures and your choices should be as efficient as possible. A set of data files will be provided. All code must be your own. You may ***not*** use the standard template library. The goal is to provide an overall solution as efficient as possible and demonstrate effective coding techniques.

Submission:
Part A (11/14)

- Submit a copy of the write-up (PDF format).
  - You are welcome to come to my office to discuss your approach before submission.

Part B (11/28)

- Submit a compressed zip file of the program source files, header files, and makefile via the on-line submission by 23:50.
- All necessary files must be included in the ZIP file. The grader will download, uncompress, and type **make**. You must have a valid, working *makefile*.
- Do ***not*** submit the data files (we have them).

Part B:
Implement the objects designed in Part A. You do ***not*** need to wait until Part A is submitted or scored to start on part B. A simple main is provided. As needed, you may update/correct/alter the original design. Any major changes should be coordinated with the instructor. The graph statistics should be displayed. Refer to the output formatting section for additional information.

Required Functions
Based on the provided main, your implementation must include the following functions.

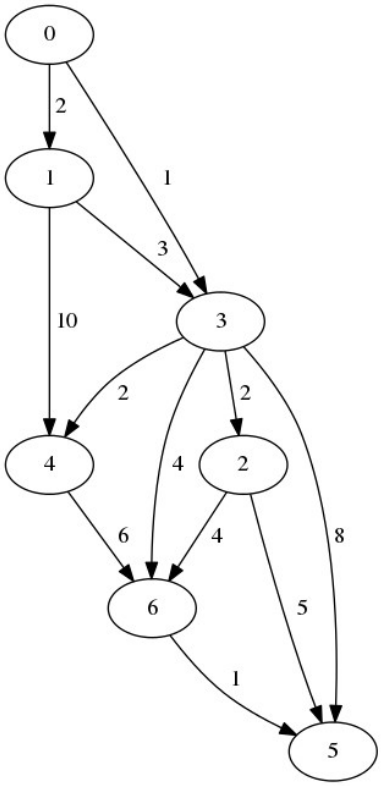
- getArguments(), readGraph(), isValidProject().
- findGraphInformation(), findKeyMilestone(), findNodePoint()
- findDependencyStats(), findIndependentMilestones()
- findAPs(), topsSort(), criticalPath(), findSlackTimes()

The functions should store the results and display only when applicable functions are called.

- printGraphInformation(), printGraph(), printDependencyStats(), printTopoSort()
- printAPs(), printCriticalPath(), printSlackTimes()

Input File Format

The input files will be task lists. The first three lines will contain a project title, milestone count, and source node (in that order). For example, below is *project1.txt*, a small test file.

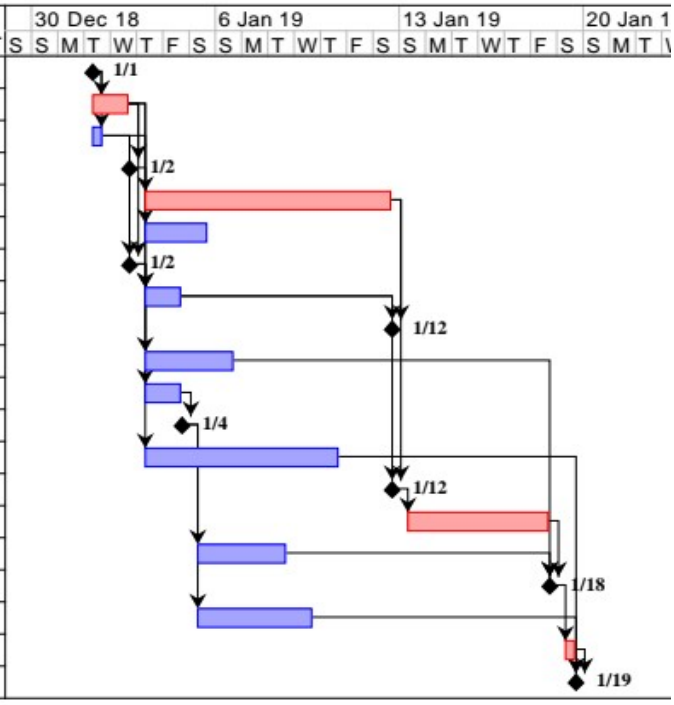



```
Project 0, Simple Example
milestones: 7
source: 0
0 1 2
0 3 1
1 4 10
1 3 3
3 4 2
3 6 4
3 5 8
3 2 2
2 5 5
2 6 4
4 6 6
6 5 1
```

Note, some of the project files will have > 1,000,000 tasks. The program should provide an error for self links (which are not allowed). For multiple edges between the same vertices's, use the latest edge (thus, ignoring previous edges).

A Gantt Chart of this simple project might be as follows:

	Name	Duration	Predeces...
1	Project Start (0)	0 days	
2	Task A (0-1)	2 days	1
3	Task B (0-3)	1 day	1
4	Milestone 1	0 days	2
5	Task C (1-4)	10 days	2
6	Task D (1-3)	3 days	4
7	MIlestone 3	0 days	2;3
8	Task E (3-4)	2 days	2;3
9	Milestone 4	0 days	5;8
10	Task F (3-6)	4 days	7
11	Task G (3-2)	2 days	7
12	Milestone 2	0 days	11
13	Task H (3-5)	8 days	7
14	Milestone 4	0 days	5;8
15	Task I (4-6)	6 days	14
16	Task J (2-6)	4 days	12
17	Milestone 6	0 days	10;15;16
18	Task K (2-5)	5 days	12
19	Task L (6-5)	1 day	17
20	Milestone 5	0 days	13;18;19



Note, the Gantt Chart package shows the critical path in red. The lines after the other tasks (in blue) represent the slack time.

Determine if a Graph is a Directed Acyclic Graph (DAG)

A valid project graph must be a DAG. A Depth First Search traversal[6] can be used to detect a cycle in a graph. DFS for a connected graph produces a tree. There is a cycle in a graph only if there is a back-edge present in the graph. A back-edge is an edge that is from a node to itself (self-loop) or one of its ancestors in the tree as produced by DFS.

- bool isValidProject()
  - create *visited*[] and initialize all elements to false
  - create *marked*[] and initialize all elements to false
  - for every vertex ***v***
    - if ***v*** is not visited before
      - if isCycle(***v***, ***visited***, ***marked***)
        - return false
    - return true

- bool isCycle(***v***, ***visited***[], ***marked***[])
  - if !*visited*[***v***]
    - set ***v*** as *visited*[] and set ***v*** as *marked*[]
    - for every adjacent ***u*** of ***v***
      - if *visited*[***u***] and isCycle(***u***, ***visited***, ***marked***)
        - return true
      - else if *marked*[***u***]
        - return true
  - *marked*[***v***] = false
  - return false

Kahn's Topological Sort Algorithm[7]

The following algorithm is Kahn's Topological sort algorithm.

- Perform initializations
  - create in-degree array and initialize all elements to 0
  - create topo[] nodes array and initialize all elements to 0
  - initialize topo count to 0
- Compute in-degree for each vertex in graph
- En-queue each vertex with an in-degree of 0
- While queue is not empty
  - remove a vertex from queue
    - decrease in-degree by 1 for all of it neighbors
    - if in-degree of a neighbor is reduced to 0, add to queue.

The topo[] nodes array holds the topological sort.

Vertices's Degree Statistics

A vertices's degree is the count of connections to or from other nodes. The out-degree is the number of edges from the current vertex to any other vertex. The in-degree is the count of edges entering the vertex. The in-degree can be found as follows:

- for each vertex ***v*** from the graph
  - for each neighbor ***u*** of ***v***
    - in-degree[***u***] += 1

6 For more information, refer to: https://en.wikipedia.org/wiki/Depth-first_search
7 For more information, refer to: https://en.wikipedia.org/wiki/Topological_sorting

Critical Path

The following algorithm is used to find the critical path in a directed acyclic graph. For each vertex, must find the minimum possible starting time of each activity. The process is as follows:

- Create an integer *distances*[] array and initialize all elements to 0
- Compute the in-degree of each vertex (that is, count the number of edges entering them).
- Since the graph is acyclic, there is at least one vertex that has in-degree zero. The pseudo code for this is as follows:
    - en-queue all vertices with an in-degree of 0
    - create empty queue, Q
    - for each vertex ***v***
        - if in-degree[***v***] = 0:
            - insert ***v*** on Q
- De-queue the first vertex ***v*** from the queue. For each neighbor ***u*** of ***v***, update *distance*[***u***] as:

  distance[***u***] = max(*distance*[***u***], *distances*[***v***] + *time*(***v***, ***u***))

  where *time*(***v***, ***u***) is the time necessary to perform task (***u***, ***v***) as stored in the adjacency list. Logically, remove ***v*** from consideration. This can be done by decreasing the in-degree of each of its neighbors. En-queue any new vertex that now has a 0 in-degree. Repeat this process until all vertices are processed. The pseudo code for this is as follows:

    - while Q is not empty
        - ***v*** = de-queue front element from Q
        - for each neighbor ***u*** of ***v***
            - *distance*[***u***] = max(*distance*[***u***], *distance*[***v***] + *time*(***v***, ***u***))
            - in-degree[***u***] -= 1
            - if in-degree[***u***] = 0
                - insert ***u*** in Q

      The total project duration found from the vertex ***x*** with the largest distance and that distance is the project duration. Vertex ***x*** is also the final task in the project.
- Now, must re-construct the critical path. Start with vertex ***x*** and search for a path to the starting vertex, with the following constraint; if you're in a vertex ***a***, you can only go to vertex ***b***, with there's an edge (***b***, ***a***) such that:

  distance[***a***] = *distance*[***b***] + *time*(***b***, ***a***)

  So, start with vertex ***x*** (final task) and search for a path to the starting vertex.

    - create integer *crPath*[] array and initialize all elements to -1
    - *crPathCount* to 0
    - set *crPath*[*crPathCount*] = ***x*** and *crPthCount*++
    - while ***x*** is not source node
        - for each vertex ***v*** from the graph
            - for each neighbor ***u*** of ***v***
                - if ***x*** equal ***u***
                    - if *distance*[***x***] = *distance*[***v***] + *time*(***v***, ***u***)
                        - *crPath*[*crPathCount*] = ***v***
                        - *crPathCount*++
                        - ***x*** = ***v***

  The critical path is in the *crPath*[] array.

Slack Time

Slack time is the time that a task in a project network can be delayed without delaying subsequent tasks or the overall project. This would apply only to tasks that were not on the critical path. The following algorithm will compute the amount of time between the start of the next task and the end time of the current task.

- Create and initialize *slackTimes*[] to -1
- Set *slackTimes*[***i***] to 0 for each ***i*** on the critical path
- For each (***u***, ***v***) not on the critical path
  - *slackTimes*[***u***] = *distance*[***v***] - (*distance*[***u***] + *time*(***u***, ***v***))

Note, the distances (i.e., *distance*[]) are computed in the critical path function.
The slack times are in the *slackTimes*[] array.

Articulation Points

In a graph, a vertex is called an articulation point if removing it (and edges through it) disconnects the graph. It can be thought of as a potential vulnerability (e.g., key point of failure). The articulation points can be computer as follows:

- findAPs()
  - Create bool *aps*[] array and initialize to false
  - Create bool *visited*[] array and initialize to false.
  - Create integer *parent*[] array and initialize to -1
  - Create integer *low*[] array and initialize to 0
  - Create integer *discovered*[] array and initialize to 0
  - For each vertex ***v***
    - if ***v*** is not visited
      - findAPsHelper(***v***, *visited*, *discovered*, *low*, *parent*, *aps*)

- findAPsHelper()
  - initialize *children* to 0 and *time* to 0
  - mark current vertex, ***v***, as visited
  - initialize discovery order
    - *disc*[***v***] = *low*[***v***] = ++*time*;
  - for every adjacent ***u*** of ***v***
    - if ***u*** is not visited
      - *children*++
      - set *parent*[***u***] = ***v***
      - findAPsHelper(***u***, *visited*, *discovered*, *low*, *parent*, *aps*)
      - *low*[***v***] = min(*low*[***v***], *low*[***u***])
      - if *parent*[***v***] == -1 and *children* > 1
        - *aps*[***v***] = true
      - if *parent*[***v***] != -1 and *low*[***u***] ≥ *disc*[***v***]
        - *aps*[***v***] = true
    - else if (***v*** != *parent*[***v***])
      - *low*[***v***] = min(*low*[***v***], *disc*[***u***])

Graph Information
The basic graph information should include the graph title (from the file, 1st line), the milestones count, the tasks count, the tasks/milestone ratio, and the task density.

The tasks/milestone ratio is calculated as follows:

$$tasksMilestoneRatio = \frac{|tasks|}{|milestones|}$$

The task density is calculated as follows:

$$tasksDensity = \frac{2 \times |tasks|}{|milestones| \times (|milestones| - 1)}$$

All calculations must be performed as floating point. The | | notation refers to the count. For example, $|tasks|$ is the number of tasks in the project.

Node Point
The node point is the first milestone with the maximum out-degree.

Key Milestone
The key milestone is the latest milestone with the maximum in-degree.

Independent Milestones
Independent milestones are all the milestones with an out-degree of 0. With an out-degree of 0, the final completion date is not directly dependent on these tasks.

Dependency Statistics
The dependency statistics for the in-degrees include value of the highest in-degree, value of the lowest in-degree (excluding the source node) and the count of vertices with the highest in-degree and lowest in-degree.

The dependency statistics for the out-degrees include value of the highest out-degree (excluding the final node), value of the lowest out-degree and the count of vertices with the highest in-degree and lowest in-degree.

Output Formatting
To accommodate the testing, the program output must follow a specific format. The functions to find the information are separate from the functions to display the information. This supports the ability to thread the various functions to calculate the results.

Some additional notes about the output format are listed as follows:

- Each section of the output is separated by 60 dash's ('-').
- Each section has a title as shown on the example output.
- Lists of milestones are output as one space and the value for as many values as exists all on one long line (automatically wrapped).

Refer to the sample output for example output formatting.

Example Executions

The following are some example program executions;

```
ed-vm%
ed-vm% ./projectInfo -f test/project1.txt
****************************************************************
CS 302 - Assignment #11
Gantt Analysis Project

------------------------------------------------------------
Graph Information
   Project title: Project 0, Simple Example
   Milestone Count: 7
   Task Count: 12
   Source Milestone: 0
   Tasks/Milestones Ratio: 1.714286
   Project Tasks Density: 0.571429

   Key Milestone: 6, in-degree: 3 tasks
   Node Point: 3, out-degree: 4 tasks
   Independent Milestones
 5

------------------------------------------------------------
Dependency Statistics (in-degree):
   Highest In-Degree: 3
   Lowest In-Degree: 1
   Count of Highest Degree: 2
   Count of Lowest Degree: 2

Dependency Statistics (out-degree):
   Highest Out-Degree: 4
   Lowest Out-Degree: 1
   Count of Highest Degree: 1
   Count of Lowest Degree: 2

------------------------------------------------------------
Topological Sort:
 0 1 3 2 4 6 5


------------------------------------------------------------
Articulation Points:
 0 2 3 6

------------------------------------------------------------
Critical Path:
   Source Node: 0
   Final Task: 5
   Total Duration: 19

Critical Path:
 0 1 4 6 5

------------------------------------------------------------
Slack Times (task-slacktime):
 2-7 3-5

****************************************************************
Game Over, thank you for playing.
ed-vm%
ed-vm%
```

```
ed-vm% ./projectInfo -f test/project1.txt -p
************************************************************
CS 302 - Assignment #11
Gantt Analysis Project

------------------------------------------------------------
Graph Information
   Project title: Project 0, Simple Example
   Milestone Count: 7
   Task Count: 12
   Source Milestone: 0
   Tasks/Milestones Ratio: 1.714286
   Project Tasks Density: 0.571429

   Key Milestone: 6, in-degree: 3 tasks
   Node Point: 3, out-degree: 4 tasks
   Independent Milestones
 5

------------------------------------------------------------
Graph Adjacency List:
   Title: Project 0, Simple Example

Vertex    vrt /weight | vrt /weight | vrt /weight | ...
------    ------------------------------------------
     0 ->    3/     1 |    1/     2 |
     1 ->    3/     3 |    4/    10 |
     2 ->    6/     4 |    5/     5 |
     3 ->    2/     2 |    5/     8 |    6/     4 |    4/     2 |
     4 ->    6/     6 |
     5 ->    None
     6 ->    5/     1 |

------------------------------------------------------------
Dependency Statistics (in-degree):
   Highest In-Degree: 3
   Lowest In-Degree: 1
   Count of Highest Degree: 2
   Count of Lowest Degree: 2

Dependency Statistics (out-degree):
   Highest Out-Degree: 4
   Lowest Out-Degree: 1
   Count of Highest Degree: 1
   Count of Lowest Degree: 2

------------------------------------------------------------
Topological Sort:
 0 1 3 2 4 6 5


------------------------------------------------------------
Articulation Points:
 0 2 3 6

------------------------------------------------------------
Critical Path:
   Source Node: 0
   Final Task: 5
   Total Duration: 19

Critical Path:
 0 1 4 6 5
```

```
------------------------------------------------------------
Slack Times (task-slacktime):
 2-7 3-5

************************************************************
Game Over, thank you for playing.
ed-vm%
ed-vm%
ed-vm% ./projectInfo -f test/project3.txt
Invalid task list.
ed-vm%
ed-vm%
ed-vm% ./projectInfo -f test/project2.txt
************************************************************
CS 302 - Assignment #11
Gantt Analysis Project

------------------------------------------------------------
Graph Information
   Project title:  small/medium graph
   Milestone Count: 900
   Task Count: 355602
   Source Milestone: 0
   Tasks/Milestones Ratio: 395.113333
   Project Tasks Density: 0.879006

   Key Milestone: 886, in-degree: 808 tasks
   Node Point: 8, out-degree: 805 tasks
   Independent Milestones
 883 884 885 886 887 888 889 890 891 892 893 894 895 896 897 898 899

------------------------------------------------------------
Dependency Statistics (in-degree):
   Highest In-Degree: 808
   Lowest In-Degree: 0
   Count of Highest Degree: 1
   Count of Lowest Degree: 15

Dependency Statistics (out-degree):
   Highest Out-Degree: 805
   Lowest Out-Degree: 0
   Count of Highest Degree: 1
   Count of Lowest Degree: 16

------------------------------------------------------------
Topological Sort:
 0 1 2 3 4 5 6 7 8 893 894 895 896 897 898 899 18 17 16 15 14 13 12 11 10
9 19 24 23 22 21 20 34 33 32 31 30 29 28 27 26 25 42 41 40 39 38 37 36 35
48 47 46 45 44 43 56 55 54 53 52 51 50 49 64 65 63 62 61 60 59 58 57 70
69 68 67 66 79 78 77 76 75 74 73 72 71 87 86 85 84 83 82 81 80 95 92 89
97 96 94 93 91 90 88 101 105 104 103 102 100 99 98 110 113 112 111 109
108 107 106 119 118 117 116 115 114 122 127 126 125 124 123 121 120 131
134 133 132 130 129 128 142 143 141 140 139 138 137 136 135 151 150 149
148 147 146 145 144 152 158 157 156 155 154 153 163 162 161 160 159 168
167 166 165 164 178 177 176 175 174 173 172 171 170 169 180 187 186 185
184 183 182 181 179 197 196 195 194 193 192 191 190 189 188 203 202 201
200 199 198 207 208 206 205 204 213 216 215 214 212 211 210 209 226 225
224 223 222 221 220 219 218 217 234 233 230 235 232 231 229 228 227 242
241 240 239 238 237 236 251 250 249 248 247 246 245 244 243 256 253 255
254 252 261 260 259 258 257 271 270 269 268 267 266 265 264 263 262 280
279 278 277 276 275 274 273 272 281 285 284 283 282 293 292 291 290 289
288 287 286 302 301 300 299 298 297 296 295 294 303 308 307 306 305 304
```

314 313 312 311 310 309 320 317 323 322 321 319 318 316 315 329 328 327
326 325 324 338 337 335 336 334 333 332 331 330 344 343 342 341 340 339
354 353 352 351 350 349 348 347 346 345 361 360 359 358 357 356 355 365
363 366 364 362 369 376 375 374 373 372 371 370 368 367 385 384 383 382
381 380 379 378 377 387 394 393 392 391 390 389 388 386 399 396 395 401
400 398 397 406 405 404 403 402 414 413 412 411 410 409 408 407 421 420
419 418 417 416 415 426 425 424 423 422 435 428 436 434 433 432 431 430
429 427 443 442 441 440 439 438 437 451 446 452 450 449 448 447 445 444
458 457 456 455 454 453 467 465 464 461 459 468 466 463 462 460 476 475
474 473 472 471 470 469 486 485 484 483 482 481 480 479 478 477 494 493
492 491 490 489 488 487 502 499 501 500 498 497 496 495 508 507 506 505
504 503 512 514 513 511 510 509 522 521 520 519 518 517 516 515 531 530
529 528 527 526 525 524 523 536 535 534 533 532 539 544 543 542 541 540
538 537 549 548 547 546 545 556 555 554 553 552 551 550 563 562 561 560
559 558 557 569 570 567 571 568 566 565 564 577 576 575 574 573 572 582
583 581 580 579 578 585 593 592 591 590 589 588 587 586 584 598 597 596
595 594 602 601 600 599 605 604 603 607 613 612 611 610 609 608 606 615
621 620 619 618 617 616 614 627 625 623 628 626 624 622 632 634 633 631
630 629 640 639 638 637 636 635 646 642 649 648 647 645 644 643 641 653
655 654 652 651 650 663 662 661 660 659 658 657 656 669 668 672 671 670
667 666 665 664 680 677 676 679 678 675 674 673 687 683 688 686 685 684
682 681 698 697 696 695 694 693 692 691 690 689 703 702 699 706 705 704
701 700 707 711 710 709 708 716 715 714 713 712 723 722 721 720 719 718
717 728 727 726 725 724 733 732 731 730 729 737 735 742 741 740 739 738
736 734 746 748 747 745 744 743 756 755 754 753 752 751 750 749 761 760
759 758 757 771 770 769 768 767 766 765 764 763 762 775 776 774 773 772
784 783 782 781 780 779 778 777 790 789 788 787 786 785 794 800 799 798
797 796 795 793 792 791 809 808 807 806 805 804 803 802 801 815 814 813
812 811 810 821 820 819 818 817 816 823 826 825 824 822 827 833 835 834
832 831 830 829 828 844 843 842 841 840 839 838 837 836 848 850 849 847
846 845 854 857 856 855 853 852 851 864 863 862 861 860 859 858 868 869
867 866 865 875 874 873 872 871 870 880 882 881 879 878 877 876 892 891
890 889 888 887 886 885 884 883

Articulation Points:
 0 882

Critical Path:
   Source Node: 0
   Final Task: 887
   Total Duration: 5514

Critical Path:
 0 14 23 29 35 43 54 63 67 71 81 93 102 108 115 122 128 140 146 155 162
168 170 184 190 203 208 212 224 227 238 251 255 259 271 278 285 287 297
304 314 315 329 338 341 348 356 363 372 378 387 400 404 408 415 425 428
437 447 457 461 471 479 491 501 504 510 515 523 534 538 549 550 559 565
575 580 592 596 602 611 621 625 630 639 646 651 657 667 678 686 696 701
708 713 719 725 732 738 743 750 759 766 775 783 786 791 803 815 818 822
832 842 848 856 860 868 872 881 887

Slack Times (task-slacktime):
 1-33 2-40 3-13 4-23 5-42 6-25 8-26 9-26 10-23 11-20 12-25 13-43 15-11
17-17 18-9 19-29 20-7 21-35 24-18 25-22 26-17 27-10 28-3 30-44 31-20 32-
30 33-27 34-7 36-47 37-53 38-44 39-22 40-32 41-39 42-38 44-16 45-1 47-16
48-10 49-41 50-5 51-30 52-22 55-28 56-11 58-33 59-5 60-27 61-18 62-30 64-
63 65-32 66-22 68-36 69-31 70-23 72-2 73-10 74-49 75-30 76-23 77-20 78-25
80-27 82-28 83-4 84-17 86-23 87-11 88-17 89-2 90-8 91-10 92-18 94-30 95-9
96-50 98-9 99-29 100-4 103-3 104-40 105-14 106-34 107-24 109-12 110-23

111-33 113-14 114-47 116-20 117-18 118-25 120-54 121-40 123-12 124-30
125-10 126-13 127-4 129-9 130-27 131-13 132-14 134-37 135-25 136-16 137-
51 139-64 141-17 142-19 143-47 144-31 145-39 147-36 149-45 150-19 151-24
152-16 154-29 156-12 157-28 158-6 159-39 160-17 161-24 163-7 164-40 166-
14 167-23 169-25 171-3 172-38 174-20 175-41 176-33 177-26 178-11 180-9
181-10 182-27 183-29 185-52 186-4 187-44 188-9 189-12 191-53 192-8 193-18
194-35 195-47 196-43 197-50 198-13 199-4 200-7 201-4 202-6 204-26 205-33
206-5 207-22 209-27 210-13 211-2 213-25 215-39 216-63 217-29 218-36 219-
53 220-9 221-60 222-54 223-41 225-19 226-17 228-5 229-21 231-31 232-19
233-9 234-12 235-34 236-37 237-44 239-51 240-43 241-46 243-35 244-7 246-
18 247-35 248-11 249-48 250-21 252-13 254-4 258-8 260-22 261-26 262-41
263-37 264-36 265-12 266-53 267-11 268-19 269-55 272-15 273-10 274-12
275-23 276-31 277-16 279-26 280-31 281-50 282-19 283-50 284-5 288-23 289-
24 290-8 291-29 292-56 293-30 294-33 295-24 296-45 298-13 299-19 300-9
302-17 303-51 305-28 306-24 307-47 308-23 309-35 310-33 311-43 312-31
313-37 316-2 317-29 318-34 319-40 320-33 322-28 323-14 324-4 325-45 326-
56 328-24 330-9 332-15 333-31 334-32 335-7 336-5 337-29 339-45 340-11
342-29 343-10 344-47 345-39 346-41 349-40 350-11 351-21 352-12 353-12
354-24 355-43 357-36 358-11 359-1 360-15 361-25 362-37 364-53 365-33 366-
17 367-9 368-80 369-39 370-45 371-10 373-45 375-9 376-15 377-34 379-1
381-27 382-3 383-1 384-1 385-39 386-31 388-51 389-55 390-23 391-30 392-31
393-27 394-14 395-8 396-32 397-22 401-22 402-7 403-33 405-15 407-4 409-23
411-15 412-42 413-40 414-22 416-24 417-20 418-34 419-20 422-15 423-52
424-47 426-8 427-28 429-13 430-42 431-12 432-17 433-40 434-55 435-33 436-
8 439-30 440-7 441-32 442-24 443-3 444-35 445-12 446-31 448-28 449-38
451-20 452-15 453-4 454-20 456-17 458-14 459-39 460-13 462-5 463-7 464-51
465-4 466-20 467-5 468-33 469-33 470-14 472-25 474-19 475-40 476-16 477-
44 478-7 480-6 481-23 482-39 483-42 484-34 485-9 487-25 488-30 489-23
490-28 492-45 493-13 494-5 495-11 496-30 497-38 498-18 499-20 500-6 502-6
503-52 505-7 506-30 508-30 509-39 511-40 512-39 513-19 514-55 516-59 517-
46 518-8 519-49 520-23 521-41 522-53 524-13 525-29 526-33 527-36 528-2
530-32 531-32 532-53 533-1 535-37 536-48 539-8 540-58 541-18 542-44 543-
47 544-25 545-18 546-22 547-57 548-29 551-38 552-46 553-28 554-59 556-31
557-15 558-20 560-20 561-22 563-17 564-41 566-37 567-16 569-21 570-24
571-26 572-58 573-21 574-52 577-33 578-4 579-33 581-56 582-40 583-44 584-
29 585-50 586-19 587-43 588-59 589-56 590-36 591-29 593-19 594-16 595-2
597-12 598-31 599-30 600-11 603-7 604-12 605-18 606-19 607-36 608-41 609-
42 612-12 613-29 614-2 615-27 616-39 617-29 618-14 619-5 620-7 622-12
623-18 624-32 627-37 628-40 629-38 631-44 632-12 633-44 635-23 636-54
637-9 640-6 641-36 642-40 643-50 644-20 645-7 647-25 648-21 650-11 653-4
654-26 655-16 656-13 658-30 659-19 661-37 662-4 663-7 664-46 665-19 666-
11 668-28 669-22 670-8 671-20 673-28 674-44 675-27 676-34 677-22 680-37
681-25 682-11 684-33 685-26 687-18 688-43 689-1 690-55 691-4 692-44 693-
15 694-3 695-9 697-46 698-30 699-16 700-19 702-29 703-30 704-14 706-20
707-57 710-51 711-62 712-31 714-52 715-46 717-53 718-12 720-45 721-26
722-39 723-27 724-8 726-12 728-9 729-27 731-44 733-17 734-67 735-34 736-
62 737-6 739-36 740-22 741-20 742-33 744-1 745-24 746-17 747-44 748-7
749-10 751-20 752-16 753-36 754-21 755-26 757-50 758-25 760-58 761-24
762-19 763-25 764-43 765-53 767-24 768-45 769-27 770-39 771-29 772-6 773-
19 774-3 777-2 778-41 779-21 781-42 782-16 784-2 785-51 787-40 788-47
789-49 790-9 792-33 793-34 794-40 796-28 797-24 798-15 799-28 800-12 801-
8 802-50 804-19 805-2 806-24 807-2 808-14 810-26 811-7 812-7 814-16 816-
35 817-48 819-60 820-45 821-52 824-49 825-14 826-16 827-10 828-26 829-36
830-30 833-30 834-40 835-27 836-12 838-7 839-32 840-44 841-45 843-35 845-
41 846-28 847-35 849-28 851-3 852-20 853-8 855-22 857-22 858-31 861-12
862-15 863-24 864-4 865-34 866-24 869-15 870-31 871-44 873-36 874-33 875-
9 876-35 877-46 878-42 879-34 880-51

****
Game Over, thank you for playing.
ed-vm%