|  |
| --- |
| Algorithm anatjariWaterRefill**(**S**,** m**)**  Input **:** Sequence S of water refilling place**,** m max distance **for** one bottle  Output **:** Sequence S2 as path **for** anatjari to cross the desert with less stops **for** refill possible  Q **<-** **new** heap**-**based priority queue  **for** each x of S **do**  Q**.**insert**(**x**,**x**)**  t**<-**0  **while** **!**Q**.**isEmpty**()** **do**  t **<-** t **+** Q**.**removeMin**()**  **if** t **+** Q**.**peekMin**()** **<** m then  S2**.**insertElement**(**t**)**  **else**  S2**.**insertElement**(**t**)**  **<-**0  **return** S2  Algorithm anatjariWaterRefill**(**G**,** k**,** s**)**  Input **:** Graph G of map **for** water refilling place**,** k max distance **for** one bottle**,** s as anitjari's starting vertices  Output **:** Sequence S2 as path **for** anatjari to cross the desert with less stops **for** refill possible  Q **<-** **new** heap**-**based priority queue  **for** each v on G**.**vertices**()** **do**  **if** v **=** s then  Q**.**insert**(**s**,**v**)**  **else**  Q**.**insert**(**v**,** v**)**  **while** **!**Q**.**isEmpty**()** then  t **<-** Q**.**removeMin**()**  currWeight **<-** 0  **for** each e on G**.**incidentEdges**(**vTemp**)** **do**  **if** weight**(**e**)** **<** k and weight**(**e**)** **>** currWeight then  currWeight **<-** weight**(**e**)**  QQ**.**removeItem**(**vTemp**)**  S2**.**insertElement**(**vTemp**)**  **return** S2 |

**C-5.1 Answer:**