Frontiers in Computing and Intelligent Systems
ISSN: 2832-6024 | Vol. 6, No. 2, 2023

"Algorithm Analysis and Design" Python Teaching Example of Greedy and Dynamic Programming

Ying Zhang [1], Lele Xi [1], Yixia Wu [1], Canping Li [2], Zebin Ma [1, *]

[1] School of Mathematics and Computer, Guangdong Ocean University, 524088, China
[2] School of Electronics and Information Engineering, Guangdong Ocean University, 524088, China
[*] **Corresponding author**: Zebin Ma (Email: mazebin@stu.gdou.edu.cn)

Abstract: The greedy algorithm and dynamic programming algorithm have always been difficult for students to understand in the course of algorithm analysis and design. This article uses Python as a descriptive language and selects classic examples of greedy and dynamic programming algorithms to analyze these two algorithms in detail, providing effective references for learning the Python language.

Keywords: Algorithm Design and Analysis; Python; Experimental Teaching; Greed; Dynamic Planning.

1. Introduction

Both the greedy algorithm and the dynamic programming algorithm are two commonly used algorithms in algorithm design. The greedy algorithm only considers the optimal solution in the current state and cannot guarantee the global optimal solution, while the dynamic programming algorithm can guarantee the global optimal solution by decomposing the problem into many overlapping subproblems and using the optimal substructure algorithm. Selecting different algorithms for different problems can optimize the efficiency of the algorithm, improve the running speed of the program, and make the program better serve the practical applications. And Python, as a simple and easy-to-learn programming language, provides a very convenient way for algorithm design.

2. Greedy Algorithms

A greedy algorithm is an algorithm that focuses only on the local optimum and then approximates the global locally [1]. However, the solution obtained using the greedy algorithm is not always the global optimal solution, it depends on the original problem and the greedy strategy used. Therefore, when solving real-world problems, we need to make a judgment on whether the current problem can be solved using a greedy strategy. That is, although they are both greedy algorithms, their strategies may be different. When solving a problem, the greedy strategy chosen must be free of a posteriori effects, i.e., each process is independent of the other and has no influence on the previous and subsequent ones [2]. Although greedy algorithms sometimes do not lead to an optimal solution but to a near-optimal solution, they can greatly improve the efficiency of our solution within the error margin [3]. The general steps for applying the greedy algorithm to solve a problem are:

(1) Set up a model to describe the problem.

(2) Divide the original problem into subproblems.

(3) Solve each subproblem, resulting in a locally optimal solution to the subproblem.

(4) Combine the local optimal solutions of the subproblems into the solution of the original problem.

Example topic: shortest path problem. Given a directed acyclic graph DAG, and the weights of each edge, find the shortest path from the origin s to each point, as shown in Figure 1.

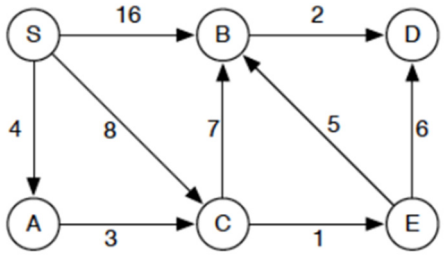

Figure 1. directed acyclic graph

Example problem analysis:

(1) First, we set the shortest distance to each point as D(X) starting at point S, where D(S)=0 and the D values of the remaining points are set to INF. and use an array to record whether the current node has been visited or not.

(2) Pick a node with the smallest D value from the unvisited nodes, traverse all the edges starting from it and update the D value at the end of the edge and set the current node as visited.

(3) Repeat step (2) until all nodes have been visited or the D values of the unvisited nodes are INF.

(4) Each time we pick one of the unvisited nodes with the smallest D-value to use it to update the shortest path to other nodes, even if the subsequent visited nodes have paths to reach the current point but the distance from that point to the current point must not be less than the D-value of the current point.

(5) Under the condition of satisfying (4) we can determine the minimum value of arriving at a point each time, and after n layers of loops we can get the shortest distance from point S to all the points, with a time complexity of $O(n^2)$.

The program and running results are shown in Figure 2:

```
import heapq
def dijkstra(graph,source):
    priority_queue = []
    heapq.heappush(priority_queue, (0, source))
    visited = {}
    while priority_queue:
        (current_distance, current) = heapq.heappop(priority_queue)
        if current not in visited:
            visited[current] = current_distance
        if current not in graph: continue
        for neighbour, distance in graph[current].items():
            if neighbour in visited: continue
            new_distance = current_distance + distance
            heapq.heappush(priority_queue, (new_distance, neighbour))
    return visited

if __name__ == "__main__":
    graph = {'S': {'A': 4, 'C': 16, 'B':8},
             'A': {'B': 3},
             'C': {'D': 2},
             'B': {'C': 7, 'E': 1},
             'E': {'C': 5, 'D': 6},
             'D': {}}
    print("S到各点的最短路径：")
    print(dijkstra(graph,'S'))
```

```
S到各点的最短路径：
{'S': 0, 'A': 4, 'B': 7, 'E': 8, 'C': 13, 'D': 14}
```

Figure 2. Dijkstra's algorithm

Considerations for using the greedy algorithm:

(1) The greedy algorithm does not consider the overall optimality of the problem, but only chooses some sense of local optimality [4], so the results obtained by the greedy algorithm are not necessarily correct.

(2) Any topic that requires consideration of previously taken steps or paths is not suitable for the use of greedy algorithms.

(3) The use of greedy algorithms needs to be proven by reasoning at the mathematical level, otherwise the greedy algorithm adopted is not always correct.

3. Dynamic Planning Algorithm

Dynamic programming algorithm is an effective method suitable for solving overlapping subproblems and optimal substructure problems created by the American mathematician Bellman in the study of optimization problems of multi-stage decision-making process[5] , which is commonly used for solving complex problems in mathematics, finance and computer science, and whose basic idea is to decompose the problem to be solved into a number of simple and interconnected subproblems, solving the subproblems first, and then using the solution of the subproblems as a The basic idea is to decompose the problem to be solved into several simple and interrelated sub-problems, solve the sub-problems first, and then use the solution of the sub-problems as the condition of the upper problem until the solution of the original problem is found. It is worth noting that the problems solved by dynamic programming are often not independent of each other after the decomposition of the subproblems[6]. Because the dynamic programming algorithm for recurring subproblems, only in the first encounter to solve it, and the solution will be saved for subsequent states to be used again [7], so this algorithm to solve the problem is much less time-consuming than other methods. The steps to solve the problem by applying the dynamic programming algorithm are as follows [8]:

(1) Define sub-problem.

(2) Guess the partial solution.

(3) Develop recursive relationships between sub-problems.

(4) Solve the transfer of state equation since the bottom is up.

(5) Combine the solutions of all sub-problems to obtain the solution of the original problem.

Example topic: miner mining problem. It is known that there are 5 gold mines and 10 miners, and the corresponding gold reserves of the 5 mines and the required miners are as follows:

(1) 400 gold/5 people
(2) 500 gold/5 people
(3) 200 gold/3 people
(4) 300 gold/4 people
(5) 350 gold/3 people

A miner can only mine one gold mine, not one and then another, and each mine is either fully mined or not mined in the first place. Solve: which gold mines should be dug to get as much gold as possible?

Example Analysis:

(1) First, we start by assuming that the i^{th} gold mine requires w[i] people, and that it generates v[i].

(2) We first analyze the maximum gain that can be achieved at the current number of users when some of the elements are selected or not and let F[i][j] be the value of the maximum gain that can be achieved by using no more than j people for the first i gold mines.

(3) Establish the recursive relationship equation:

$$F[i][j] = \begin{cases} \max(F[i-1][j], F[i][j-1]) & j < w[i] \\ \max(F[i-1][j-w[i]] + v[i], F[i-1][j], F[i][j-1]) & j \geq w[i] \end{cases} \quad (1)$$

(4) Then iterates through all the gold mines accordingly, and finally F[5][10], which is the result needed for this question

Algorithmic Analysis:

(5) Time complexity analysis of the algorithm, let the number of miners be m and the number of mines be n, then the time complexity of the current algorithm is O(n*m)

The program and running results are shown in Figure 3:

```
import copy
def goleMining(n,w,g=[],p=[]):
    arr = [0]*w
    for i in range(w):
        if (i+1)>=p[0]:
            arr[i] = g[0]
    res = copy.deepcopy(arr)
    print(res)
    for i in range(1,n):
        for j in range(w):
            if (j+1)<p[i]:
                arr[j] = res[j]
            else:
                t = 0 if (j-p[i])<0 else j-p[i]
                arr[j] = max(res[j],res[t]+g[i])
        res = copy.deepcopy(arr)
        print(res)
    return res.pop()

if __name__ == '__main__':
    res = goleMining(5,10,[400,500,200,300,350],[5,5,3,4,3])
```

```
[0, 0, 0, 0, 400, 400, 400, 400, 400, 400]
[0, 0, 0, 0, 500, 500, 500, 500, 500, 900]
[0, 0, 200, 200, 500, 500, 500, 700, 700, 900]
[0, 0, 200, 300, 500, 500, 500, 700, 800, 900]
[0, 0, 350, 350, 500, 550, 650, 850, 850, 900]
```

Figure 3. Problem of miners digging

Considerations for using dynamic programming algorithms:

(1) Definition analyzes whether the problem has the properties of optimal substructure and overlapping subproblems [9] and can only be solved using dynamic programming algorithms when the problem satisfies these two properties.

(2) The results of each operation need to be saved so that resources are not wasted on calculating the same thing later.

(3) For the current problem if you can't represent its state using a low-dimensional array, you need to open a high-dimensional array to represent its state.

4. Conclusion

Through this article on the greedy algorithm and the dynamic programming algorithm, we can see the differences and applications between the two algorithms, with the greedy method being a special case of dynamic programming. In

Python, the implementation of the two algorithms is very similar, both need to explicitly perform state transfer and optimization solution. In practice, we need to choose the suitable algorithm to solve the problem according to the characteristics and requirements of the problem. Python, as a programming language with very high language readability and ease of learning, provides us with a good convenience for optimization algorithm design. We believe that Python and these two algorithms will play a more important role in the future learning and application.

Acknowledgments

This work was supported by Postgraduate Education Innovation Project of Guangdong Ocean University (202303) and Guangdong Ocean University Innovation and Entrepreneurship Training Program Project Grant (CYXL2023005) (CXXL2023124) (CXXL2023134).

References

[1] Liu Dan. Research on automatic class scheduling method based on greedy algorithm [J]. Information and Computer (Theoretical Edition),2020,32(04):36-38.

[2] Chang Youqu, Xiao Guiyuan, Zeng Min. Exploration and research on greedy algorithm[J]. Journal of Chongqing Electric Power College, 2008, 13(3):40-42.

[3] Bi Longge. Greedy algorithm and linear programming[J]. Computer Products and Distribution,2017, (11):239+251.

[4] Wang Ying. On the application of greedy algorithm in graph theory [J]. Computer CD-ROM Software and Applications, 2013, 16(16):309+311.

[5] CHENG Zhenbo, LI Qu,WANG Chunping. Algorithm design and analysis[M]. Tsinghua University Press:Tsinghua University Academic Research Building,Beijing,2018.

[6] Dong Junjun. Comparison and analysis of dynamic programming algorithm and greedy algorithm[J]. Software Journal, 2008, (02):129-130.

[7] ZHANG Aihua, GUO Xiyue,CHEN Qianjun. Analysis and research on dynamic programming algorithms[J]. Software Guide,2014,13(12):68-69.

[8] SHI Shaojian, ZHANG Hong,SHI Zheng. Research on dynamic programming algorithms[J]. Computer Knowledge and Technology,2020,16(18):48-49.

[9] Wang Junxiang. Research on the principle and application of dynamic programming algorithm[J]. Computer Knowledge and Technology: Academic Exchange, 2006.

[10] Zhang M, Zhao H, et al. Experimental tutorial on data structures and algorithms [M]. Beijing. Higher Education Press.2011.