2023 학년도 1학기 출석과제물

|  |  |  |  |
| --- | --- | --- | --- |
|  | **교과목명** | : | 알고리즘 |
|  | **학번** | : | 202234-366307 |
|  | **성명** | : | 최문성 |
|  | **연락처** | : | 010-8010-6050 |

1. 다음은 입력 크기 n에 대한 Big-O 함수들이다. 알고리즘의 성능 관점에서 가장 나쁜 것부터 차례대로 나열하시오.

O(), O(2n), O(n), O(n3), O(1), O(n2), O(n)

알고리즘의 성능 관점에서 시간 복잡도가 높을수록 수행 시간이 길어지기 대문에 시간 복잡도 수치가 작을수록 성능이 좋은 알고리즘이라고 할 수 있다. n개의 데이터에 대한 시간 복잡도를 구할 때는 n이 포함된 최고차항이 무엇인지로 판별한다. 위의 항목을 나쁜 것 부터 나열한 결과는 아래와 같다.

O(2n), O(n3), O(n2) , O(n), O(n), O(), O(1)

1. 다음 4가지 경우에 해당하는 점화식과 폐쇄형을 쓰시오.
2. 이진탐색

이진탐색의 경우 크기가 n인 문제를 인 두 개의 문제로 분할하고, 거기에 중간값을 구하는 연산이 하나 포함된다. 그러므로 점화식과 폐쇄형은 다음과 같다.

1. 퀵 정렬의 최악의 경우

퀵 정렬은 만약 데이터가 이미 크기 순으로 정렬되어있는 경우 피벗의 왼쪽 오른쪽 중 한 쪽에는 원소가 하나도 없고 반대쪽에만 모든 원소가 있는 불균형한 분할이 이뤄지는데 이러한 경우에 최악의 수행시간을 갖는다. 점화식과 폐쇄형은 다음과 같다.

1. 합병 정렬

합병 정렬은 크기가 n인 배열을 두 개로 나눠서 순환적으로 정렬하고 합병하는 과정을 반복한다. 이에 대한 점화식과 폐쇄형은 다음과 같다.

1. 퀵 정렬의 최선의 경우

만약 피벗이 항상 정중앙에 위치하고 왼쪽과 오른쪽에 원소들이 균등하게 항상 의 크기로 분할된다면 퀵 정렬은 최선의 시간복잡도를 가지고 이에 대한 점화식과 폐쇄형은 다음과 같다.

1. 대표적인 알고리즘 설계 기법이 적용된 문제들을 모두 나열하고, 해당 문제의 정의/개념에 대해서 간단히 설명하시오.
2. 주어진 배열에 대해서 퀵 정렬의 분할 함수 Partition()을 한 번 적용한 후의 결과 배열을 구하시오. (단, A[0]이 피벗이다.)

A[] = {30, 35, 25, 55, 10, 50, 15, 45, 20, 40}

먼저 분할함수 Partition()을 호출하여 A[0] = 30을 피벗으로 지정하고 Left = 1, Right = 9로 초기화한다.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 피벗 | Left | -> |  |  |  |  |  | <- | Right |
| 30 | 35 | 25 | 55 | 10 | 50 | 15 | 45 | 20 | 40 |

초기상태에서 Left는 오른쪽으로 이동하면서 피벗보다 큰 값을 찾고, Right는 왼쪽으로 이동하면서 피벗보다 작은 값을 찾는다.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 피벗 | Left |  |  |  |  |  |  | Right |  |
| 30 | 35 | 25 | 55 | 10 | 50 | 15 | 45 | 20 | 40 |

35 > 30 이므로 Left는 시작했던 자리에 그대로 있고 Right는 왼쪽으로 이동하면서 처음으로 피벗보다 작은 값인 20에서 멈춘다. Left < Right 이므로 두 값의 위치를 교환한다.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 피벗 | Left |  |  |  |  |  |  | Right |  |
| 30 | 20 | 25 | 55 | 10 | 50 | 15 | 45 | 35 | 40 |

이 상태에서 다시 Left는 오른쪽으로 이동하면서 피벗보다 큰 값을 찾고, Right는 왼쪽으로 이동하면서 피벗보다 작은 값을 찾는다.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 피벗 |  |  | Left |  |  | Right |  |  |  |
| 30 | 20 | 25 | 55 | 10 | 50 | 15 | 45 | 35 | 40 |

피벗보다 큰 55에서 Left는 멈추고, 피벗보다 작은 15에서 Right는 멈춘다. Left < Right이므로 두 값을 교환한다.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 피벗 |  |  | Left |  |  | Right |  |  |  |
| 30 | 20 | 25 | 15 | 10 | 50 | 55 | 45 | 35 | 40 |

이 상태에서 다시 Left는 오른쪽으로 이동하면서 피벗보다 큰 값을 찾고, Right는 왼쪽으로 이동하면서 피벗보다 작은 값을 찾는다.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 피벗 |  |  |  | Right | Left |  |  |  |  |
| 30 | 20 | 25 | 15 | 10 | 50 | 55 | 45 | 35 | 40 |

Left > Right이므로 Right와 피벗의 위치를 바꾸고 Partition() 함수가 종료된다.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 왼쪽 부분 배열 | | | | 피벗 | 오른쪽 부분 배열 | | | | |
| 10 | 20 | 25 | 15 | 30 | 50 | 55 | 45 | 35 | 40 |

Partition() 함수가 한 번 수행된 결과로 왼쪽 부분 배열에는 피벗보다 작은 값만 남고, 오른쪽 부분 배열에는 피벗보다 큰 값만 남게 된다. 이런 방식으로 왼쪽 부분 배열과 오른쪽 부분 배열에 순환적으로 Partition() 함수 적용을 반복하면 전체 정렬 결과를 얻을 수 있다.

1. 물체를 쪼갤 수 있는 배낭 문제에 대해서 욕심쟁이 방법을 적용해서 최대 이익을 구하시오.

M=10, n=4

(p1,p2, p3, p4) = (18, 20, 9, 25), (w1, w2, w3, w4) = (5, 4, 3, 4)

위의 물체를 쪼갤 수 있는 배낭 문제의 단위 무게당 이익은 다음과 같다.

(, , , ) = () = (3.6, 5, 3, 6.25)

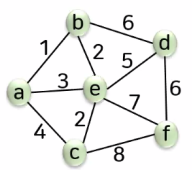
단위 무게당 이익은 물체 4, 물체 2, 물체 1, 물체 3 순으로 높다.

물체 4의 용량 4는 배낭의 용량 10보다 작으므로 물체 4를 통째로 배낭에 넣고 배낭의 남은 용량을 10-4=6으로 감소시킨다.

다음으로 물체 1의 용량 5는 배낭의 남은 용량 6보다 작으므로 역시 물체 1을 배낭에 통째로 넣고 배낭의 남은 용량을 6-5=1로 감소시킨다.

다음으로 물체 2의 용량 4는 배낭의 남은 용량 1보다 크므로 물체 2를 남은 용량만큼 로 쪼개서 배낭에 넣으면 배낭은 꽉 차게 되고, 배낭에 담은 물체의 총 이익은 아래와 같다.

물체 4의 이익+물체 1의 이익+(물체2의 이익)/4 = 25+18+20/4 = 48



1. 다음 그래프에 대한 최소 신장 트리와 해당 트리의 가중치의 합을 구하시오.

최소 신장 트리를 구하는 알고리즘은 크루스칼 알고리즘과 프림 알고리즘이 있는데 각각의 알고리즘을 사용한 풀이법은 다음과 같다

1. 크루스칼 알고리즘

우선 간선 집합을 공집합으로 초기화하고 각 정점이 모두 다른 연결성분에 속하도록 초기화 한다. 그리고 원래 있었던 간선을 가중치의 증가순으로 정렬하면 다음과 같은 결과가 나온다.

a b, 가중치 1

b e, 가중치 2

c e, 가중치 2

a e, 가중치 3

a c, 가중치 4

d e, 가중치 5

b d, 가중치 6

d f, 가중치 6

e f, 가중치 7

c f, 가중치 8

다음으로 가중치가 가장 작은 간선부터 차례대로 선택하면서 서로 다른 연결 성분을 연결하는 작업을 반복한다.

우선 가중치가 가장 낮은 간선 (a, b)를 선택하면 연결 성분은 다음과 같이 변한다.

{a, b}, {c}, {d}, {e}, {f}

다음으로 가중치가 낮은 간선 (b, e)를 선택한다.

{a, b, e}, {c}, {d}, {f}

정점 c 또한 정점 e가 포함된 연결 성분인 {a, b, e}과 연결되어 있지 않은 상태이므로 간선 (c, e)를 선택한다.

{a, b, c, e}, {d}, {f}

다음으로 가중치가 낮은 간선 (a, e)는 이미 연결 성분 {a, b, c, e}에 포함 되어있으므로 추가하지 않는다. 간선 (a, c)도 마찬가지로 추가하지 않는다. 다음으로 간선 (d, e)를 추가한다.

{a, b, c, e, d}, {f}

다음으로 가중치가 낮은 간선 (b, d)는 이미 연결 성분 {a, b, c, e}에 포함 되어있으므로 추가하지 않고, 간선 (d, f)를 선택하면 모든 연결 성분이 포함되고 사이클을 형성하지 않는 최소 신장 트리가 완성된다.

최종적으로 추가된 간선: (a, b), (b, e), (c, e), (d, e), (d, f)

b d

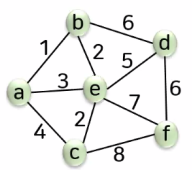
1 6

a 2 5 f 가중치의 합: 16

c e

2

1. 프림 알고리즘

프림 알고리즘은 임의의 한 정점에서 시작해 아직 연결되지 않은 정점 사이의 간선 중 가중치가 낮은 간선을 선택해서 추가한다.

우선 임의의 정점 e를 선택한다.

S = {e}, V-S = {a, b, c, d, f}

정점 e와 V-S를 연결하는 간선으로는 (e, a), (e, b), (e, c), (e, d), (e, f)가 있다. 그 중에서 가장 가중치가 작은 (e, b)를 선택해서 추가한다.

S = {e, b}, V-S = {a, c, d, f}

S와 V-S를 연결하는 간선으로는 (b, a), (b, d), (e, a), (e, c), (e, d), (e, f)가 있다. 그 중 가중치가 가장 낮은 (b, a)를 선택하여 추가한다.

S = {e, b, a}, V-S = {c, d, f}

이와 같은 작업을 반복하여 {e, b, a}와 {c, d, f}를 연결하는 간선 중 가장 가중치가 낮은 (e, c)를 선택하면

S = {e, b, a, c}, V-S = {d, f}

가 되고 같은 방식으로 (e, d)와 (d, f)를 추가하면 모든 정점이 포함되고 사이클이 없는 최고 신장 트리를 만들 수 있다.

최종적으로 추가된 간선: (e, b), (b, a), (e, c), (e, d), (d, f)

b d

1 6

a 2 5 f 가중치의 합: 16

c e

2