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.

알고리즘 설계 기법으로는 크게 분할정복 알고리즘, 동적 프로그래밍 알고리즘, 욕심쟁이 알고리즘이 있다. 분할정복 방법이란 주어진 문제의 입력을 두 개 이상의 작은 문제로 순환적으로 분할하고, 분할된 작은 문제들의 해를 구한 후 결합하여 전체의 해를 구하는 방식으로 이진 탐색, 합병 정렬, 퀵 정렬, 선택 문제 등에 사용 된다. 이진 탐색은 크기 순으로 정렬된 데이터가 있을 때 정렬된 배열의 중간에 있는 원소와 탐색키 x를 비교하여 탐색키 x가 중간 원소보다 작으면 왼쪽에서 탐색, 클 경우 오른쪽에서 탐색을 반복적으로 수행하는 방식이다. 합병 정렬은 주어진 배열을 1/2크기의 부분배열로 분할하는 작업을 반복한 뒤 각 부분배열에 대해서 합병 정렬을 사용해 순환적으로 정렬하고, 마지막으로 정렬된 부분배열들을 합병하여 하나로 만드는 방법이다. 퀵 정렬은 주어진 배열을 피벗을 기준으로 하여 두 부분배열로 분할하고, 부분배열에 대해서 퀵 정렬을 순환적으로 적용하여 정렬하는 작업을 반복하는 방식이다. 마지막으로 선택문제란 n개의 원소가 저장된 배열에서 i번째로 작은 원소를 찾는 문제인데 퀵 정렬의 Partition() 함수를 사용하면 피벗값의 왼쪽에는 피벗보다 작은 값만 남게되고 오른쪽에는 큰 값만 남게되는 것을 이용하여서 Partition() 함수를 순환적으로 호출하여서 피벗의 인덱스를 통해 i번째로 작은 원소를 찾는 방식이다. 이 때 피벗값이 최솟값이나 최댓값쪽으로 한 쪽에 치우칠 때는 시간 복잡도가 O(n2)이지만 임의의 중간값이 된다면 최악의 경우에도 평균 성능 O(n)이 보장되므로 배열의 원소를 5개씩 나눠서 중간값을 선정하고 그 중간값들 중에 중간값을 피벗으로 선정하는 과정을 추가하기도 한다.

동적 프로그래밍 방법은 크기가 작은 소문제에 대한 해를 저장하고 이를 이용해 크기가 큰 문제의 해를 구해가는 방법으로 주어진 문제에 대해 최적해를 제공하는 점화식을 도출하고 가장 작은 소문제부터 점화식의 해를 구해서 저장하고 저장된 해를 이용해 상위문제의 해를 구해가는 방식으로 적용된다. 피보나치 수열의 경우 정의 자체가 이전 숫자의 해와 전전 숫자의 해를 이용해서 답을 구하는 방식이므로 각 숫자의 해를 테이블에 저장해뒀다가 필요할 때 꺼내쓰면 분할정복 방법을 이용할 때보다 계산 횟수를 엄청나게 줄일 수 있다. 연쇄 행렬 곱셈 문제는 n개의 행렬을 곱하는 최적의 곱셈순서를 구하는 문제인데 해당 문제의 최종적인 해는 전체 n개 행렬의 어떤 부분집합을 곱하는 최적의 순서를 포함하고 있고 특정 부분집합끼리 곱할 때 결합시키는 비용이 가장 작은 분리위치 k를 찾아서 최적해를 구하는 방식이다. 스트링 편집 거리 문제는 어떤 문자열 X를 다른 문자열 Y로 변환하고자 할 때, 삽입, 삭제, 변경 연산이 일어나는데 각각의 연산에 대한 비용이 주어졌을 때 어떤 방식으로 변환해야 최소의 비용으로 변경할 수 있는지에 대한 문제이다. 두 문자열을 변환하는 최적의 편집 방법이 있다면 그 방법은 두 문자열에서 각각 마지막 글자를 뺀 나머지 글자의 최적의 편집 방법도 포함하고 있음을 의미하므로 이를 이용해서 점화식을 도출하고 테이블을 이용하여 연산한다. 모든 정점 간의 최단 경로 문제는 가중 방향 그래프에서 두 정점을 연결하는 경로 중 간선의 가중치 합이 가장 작은 최단 경로를 모든 조합의 두 정점 간에 대해서 구하는 문제이다. 저울 문제는 저울과 무게가 다른 n개의 추가 있을 때 무게가 M인 물체를 추를 이용해서 달 수 있는지 확인하는 문제이다. 만약 무게가 M인 물체를 달 수 있는 해가 있다면, 여러 추의 무게를 더해서 합을 M으로 만들 수 있다는 뜻이다. 해당 해에 만약 n번의 추가 포함되지 않는다면 1번부터 n-1번까지의 추를 이용하여 물체를 달 수 있는지 확인하는 문제가 된다. 만약 n번 추가 포함되는 경우를 이항하면 1번부터 n-1번 추의 무게를 더 하면 총 무게 M에서 n번 추의 무게를 뺀 값과 같다는 말이 된다. 이 경우에 대한 점화식을 도출하여서 풀이하는 것이 저울 문제이다.

욕심쟁이 방법은 해를 구하는 단계마다 이전 이후 단계의 선택을 고려하지 않고 당장의 상황만 놓고 봤을 때 해당 단계에서 가장 최선인 해를 선택해서 더해나가면서 전체적인 최적해를 구하는 방식이다. 다른 요소에 대한 고려 없이 해당 단계에서 최선인 것만 구하다보니 계산이 간단하고 빠르다는 장점이 있지만 최종적인 결과가 반드시 가장 효율적인 최적해라고 보장할 수가 없다. 동전 거스름돈 문제의 경우 거스름돈을 여러 종류의 동전을 이용해서 거슬러줄 때 거스름돈의 액수를 초과하지 않는 선에서 가장 큰 화폐단위를 우선 선택해서 거스름돈을 남겨주면서 동전의 개수를 최소화하는 문제이다. 동전의 액면가가 임의로 주어지는 경우는 욕심쟁이 방법을 이용할 수 없게 된다. (물건을 쪼갤 수 있는) 배낭문제는 각각 용량과 이익이 정해진 물건을 배낭에 어떻게 담아야 최대 이익을 얻을 수 있는지 구하는 문제이다. 각각 물건별로 무게 대비 이익을 구한 뒤, 배낭의 용량을 초과하지 않는 선에서 물건을 넣고 배낭의 용량을 초과하는 마지막 단계에서는 해당 물건을 남은 용량만큼 쪼개서 집어넣는다. 최소 신장 트리 문제는 가중 무방향 그래프에서 모든 정점이 포함되고 사이클이 존재하지 않는 신장 트리를 만들되, 가중치가 최소인 경우를 구하는 것이다. 간선을 가중치가 낮은 순으로 정렬하여 사이클을 만들지 않는 간선을 하나씩 추가하는 크루스칼 알고리즘과 임의의 한 정점부터 시작하여 해당 정점을 포함하지 않는 정점의 집합을 가중치가 낮은 순서로 연결해나가는 프림 알고리즘이 있다. 최단 경로 문제는 가중 그래프에서 두 정점을 연결하는 경로 중 간선의 가중치 합이 가장 작은 경로를 찾는 문제로 그래프에 음의 가중치가 없는 경우 다익스트라 알고리즘을 사용하여 풀 수 있다. 출발점에서 거리가 최소인 정점을 u를 선택하고, u의 인접 정점에 대해서 u를 경우하는 거리와 기존 거리를 비교해서 작은 값을 거리값으로 조정하는 과정을 반복적으로 수행하여 모든 정점까지의 최소 거리를 구한다. 작업 스케줄링 문제는 가장 적은 개수의 기계를 이용해서 작업 간의 충돌이 발생하지 않도록 모든 작업을 기계에 할당하는 문제로각 단계에서 시작 시간이 빠른 작업을 우선적으로 선택해서, 충돌 여부에 따라 기존 기계 또는 새 기계에 할당하는 과정을 반복한다. 작업 선택 문제는 하나의 기계만 사용해서 충돌 없이 최대 개수의 작업을 기계에 할당하는 문제로 각 단계에서 완료시간이 빠른 작업을 우선적으로 선택하고, 충돌되는 작업을 버리는 과정을 반복해서 구할 수 있다. 허프만 코딩은 문자가 텍스트에서 출현하는 빈도수를 이용해 빈도수가 높은 문자는 짧은 코드, 빈도수가 낮은 문자는 상대적으로 긴 코드를 부여하여 압축하는 방법으로 각 문자의 출현 빈도수를 계산하여 허프만 트리를 생성 후 빈도수에 따라 이진코드를 부여하고 주어진 텍스트를 코드로 변환하여 압축하는 방법이다.

1. 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)

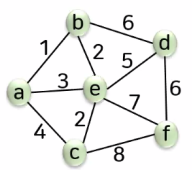
단위 무게당 이익은 물체 4, 물체 2, 물체 1, 물체 3 순으로 높다.

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

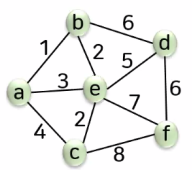
1 6

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