SAMSUNG SW ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

백트래킹

SAMSUNG SW ACADEMY FOR YOUTH

Contents

SAMSUNG SW ACADEMY FOR YOUTH

백트래킹 응용

Backtracking

백트래킹 응용

정의

- 모든 가능한 해를 찾는 데 있어 불필요한 부분을 탐색하지 않도록 하는 기법
- 트리 구조를 기반으로 한 상태 공간 탐색으로, 특정 조건을 만족하지 않으면 이전 단계로 돌아가 다른 경로를 탐색함.

알고리즘 흐름

1. 현재 상태에서 가능한 **모든 선택지를 시도.**
2. 각 선택지에 대해 재귀적으로 해당 선택지를 선택한 후의 상태를 탐색.
3. 만약 현재 선택지가 해가 될 수 없음을 알게 되면, **되돌아가서 다른 선택지를 시도**.
4. 가능한 모든 선택지를 탐색하거나, 해를 찾으면 탐색을 종료.

백트래킹 응용

핵심 원리

1. 재귀적 탐색
   - 재귀를 사용하여 문제를 단계적으로 해결. 큰 문제를 작은 하위 문제로 나누어 해결하는 방식.
2. 가지치기
   - 탐색 도중 유망하지 않은 경로를 미리 제거하여 탐색 공간을 줄임.
   - 불필요한 경로를 제거하여 시간 복잡도를 줄이는데 중요한 역할.
3. 상태 공간 트리
   - 가능한 모든 해의 공간을 표현하는 트리 구조.

백트래킹 알고리즘 설계 단계

백트래킹 응용

백트래킹의 일반적 패턴

1. 상태
   - 문제를 풀기 위한 특정 시점에서의 데이터.
2. 종료 조건
   - 문제의 해를 찾았거나, 더 이상 탐색할 필요가 없을 때를 지정.
   - 재귀 탐색이 멈춰야 하는 조건.
3. 유망성 검사
   - 현재 상태가 문제의 해가 될 가능성이 있는지 판단.
   - 다음 조사 대상으로 새로운 상태를 추가하여 조사하는 판단 근거.
4. 선택지를 되돌림 (backtrack)
   - 현재 상태를 되돌려 이전 상태로 복구하는 과정.
   - Ex) 방문 표시를 제거.

```
def backtrack(상태):
    if 종료 조건 만족:
        해를 기록
        return

    for 가능한 모든 선택지:
        선택지를 적용

        if 유망성 검사 통과:
            backtrack(새로운 상태)

        선택지를 되돌림 (백트래킹)
```

SAMSUNG SW ACADEMY FOR YOUTH

5208. 전기버스2

Learn Course Adv. Backtracking

5208. 전기버스2

목표

- 전기버스를 운행하는 동안 **최소한의 배터리 교환 횟수**로 목적지에 도달하는 것이 목표.

문제 접근 방법

1. 그리디 알고리즘 접근법
   - 현재 정류장에서 가장 멀리 갈 수 있는 선택지를 계속해서 선택하면서 문제 해결
2. 백트래킹 접근법
   - 모든 가능한 교환 지점을 탐색하면서, 최소 교환 횟수를 찾는 방법으로 문제 해결

5208. 전기버스2

백트래킹 알고리즘 설계

1. 출발지에서 시작하여, 현재 배터리 용량이 허용하는 모든 다음 정류장을 탐색.
2. 각 정류장에서 가능한 모든 배터리 교환을 시도하고, 재귀적으로 다음 정류장으로 이동하며 교환 횟수를 누적.
3. 종점에 도달할 때마다 최소 교환 횟수를 갱신.
4. 탐색 도중 현재 교환 횟수가 이미 기록된 최소 교환 횟수를 초과하면 해당 경로 탐색을 중단. (가지치기)

- 추가 조건

1. 최소 충전 횟수를 찾는 것이 목적이므로 충전을 안해도 되는 상황에는 최대한 충전을 하지 않는 방법으로 탐색.

5208. 전기버스2

코드 작성 (1/2)

1. N, *stations
   - 정류장 수 N
   - 충전지 정보를 packing하여 각각 할당
2. result
   - 최종 결과값
   - 최악의 상황을 고려하여 초기화
   - 추후 탐색 하며 값을 갱신

```
T = int(input())

for tc in range(1, T+1):
    N, *station = map(int, input().split())
    # 충분히 많은 충전 횟수
    result = N
    search(0, station[0], 0)
    print(f'#{tc} {result}')
```

5208. 전기버스2

코드 작성 (2/2)

```
def search(now, battery, acc):
    global result
    '''
        now : 현재 정류장 위치
        battery : 남은 배터리 잔량
        acc : 누적 충전 횟수
    '''
    if acc > result:    # 아직 종점에 도달하지 않았으나, 최소 충전량 보다 많다면
        return          # 유망성 없음.

    if now == N-1:          # 최종 목적지에 도착했다면,
        if result > acc:    # 최솟값 비교
            result = acc    # 최솟값 갱신
        return              # 조사 종료
    else:
        if battery:                                     # 배터리가 남았으면
            search(now + 1, battery - 1, acc)           # 일단 충전 없이 가 보거나
        search(now + 1, station[now] - 1, acc + 1)      # 충전 하고 이동
```

SAMSUNG SW ACADEMY FOR YOUTH

5209. 최소 생산 비용

Learn Course Adv. Backtracking

5209. 최소 생산 비용

목표

- 최소 비용으로 N개의 제품을 각각 다른 N개의 공장에서 생성하는 방법을 찾는 것이 목표.

문제 접근 방법

1. 동적 계획법 접근법
   - 반복되는 부분 문제를 해결하여 전체 문제를 해결
2. 백트래킹 접근법
   - 각 제품을 생상 할 공장을 선택할 때, 가능한 모든 조합을 탐색하면서 최적의 조합을 찾아 문제 해결.

5209. 최소 생산 비용

Confidential

백트래킹 알고리즘 설계

1. 첫 번째 제품부터 시작하여 공장을 선택.
2. 현재 제품에 대해 남아 있는 모든 공장을 선택.
3. 선택한 공장의 비용을 현재 총 비용에 더하고, 다음 제품에 대해 재귀적으로 탐색.
4. 모든 제품에 대해 공장을 선택했을 때, 현재까지의 총 비용을 기록하고, 최소 비용을 갱신.
5. 선택을 되돌려, 다음 공장을 선택하는 과정을 반복
6. 탐색 도중 현재 비용이 이미 기록된 최소 비용을 초과하면 해당 경로 탐색을 중단 (가지치기)

- 추가 조건

1. 최소 충전 횟수를 찾는 것이 목적이므로 충전을 안해도 되는 상황에는 최대한 충전을 하지 않는 방법으로 탐색.

5209. 최소 생산 비용

Confidential

코드 작성 (1/2)

1. visited
   - N 개의 공장 사용 여부 체크
   - 조사 대상인지 유망성 체크
2. result
   - 최종 결과값
   - 최악의 상황을 고려하여 초기화
   - 최대 N * 최대 비용 Vij

```
T = int(input())

for tc in range(1, T+1):
    N = int(input())
    data = [list(map(int, input().split())) for _ in range(N)]
    result = 15*99
    visited = [0] * N
    search(0, 0)
    print(f'#{tc} {result}')
```

5209. 최소 생산 비용

코드 작성 (2/2)

```
def search(now, acc):
    global result
    '''
        now : 현재 조사 대상 제품
        acc : 누적 생산 비용
    '''

    if acc > result:    # 모든 제품을 조사하지는 않았으나, 누적값이 초과했다면,
        return          # 유망성 없음.

    if now == N:            # 모든 제품에 대해 조사를 마쳤다면
        if result > acc:    # 최솟값 비교
            result = acc    # 최솟값 갱신
        return
    else:
        for w in range(N):      # 모든 공장의 now 번째 제품 생산 비용을 누적 해 볼 수 있도록
            if not visited[w]:  # 아직 w 번째 공장에서 제품을 생산한 적이 없다면,
                visited[w] = 1  # 해당 공장에서 now 번째 제품을 생산할 것이라고 표기 후,
                search(now+1, acc + data[now][w])   # 다음 제품 조사 시작, 누적값 증가
                visited[w] = 0  # backtracking
```

SAMSUNG SW ACADEMY FOR YOUTH

1865. 동철이의 일 분배

Code Problem

1865. 동철이의 일 분배

목표

- N명의 직원과 N개의 일을 매칭하는 방법 중 모든 일이 성공할 확률의 최댓값을 찾는 것이 목표.

문제 접근 방법

1. 백트래킹 접근법
   - 각 직원에게 하나씩 일을 할당하는 모든 경우의 수를 탐색하여 문제 해결.

1865. 동철이의 일 분배

백트래킹 알고리즘 설계

1. 첫 번째 직원부터 시작하여 각 직원에게 가능한 일을 할당.
2. 현재 직원에 대해 남은 모든 일을 선택하여 할당.
3. 다음 직원에 대해 재귀적으로 탐색을 진행.
4. 모든 일을 할당했을 때, 누적된 확률이 현재까지의 최대 확률 보다 높다면, 최대 확률을 갱신.
5. 선택을 되돌려, 다음 가능한 일을 선택하는 과정을 반복.
6. 탐색 도중 현재 확률이 이미 기록된 최대 확률보다 낮다면 해당 경로 탐색을 중단 (가지치기)

- 필수 조건

1. 확률을 퍼센트 단위로 소수점 아래 7번째 자리에서 반올림하여 6번째까지 출력.

1865. 동철이의 일 분배

Confidential

코드 작성 (1/2)

1. visited
   - N 개의 직원 업무 여부 체크
   - 조사 대상인지 유망성 체크
2. f'{result*100:6f}'
   - f-string의 float format 방법
   - 7번째 자리에서 반올림하여 총 6자리의 소수점을 나타냄.

```
T = int(input())

for tc in range(1, T+1):
    N = int(input())
    data = [list(map(int, input().split())) for _ in range(N)]
    result = 0
    visited = [0] * N
    search(0, 1)
    print(f'#{tc} {result*100:6f}')
```

1865. 동철이의 일 분배

코드 작성 (2/2)

```
def search(now, acc):
    global result
    if result >= acc: return

    if now == N:
        result = max(result, acc)   # 최댓값 갱신
        return
    else:
        for w in range(N):
            if not visited[w]:
                visited[w] = 1
                # 확률 계산을 위해 원본 값에서 100을 나눈 값을 곱셈
                search(now+1, acc * (data[now][w] / 100))
                visited[w] = 0
```

SAMSUNG SW ACADEMY FOR YOUTH

1952. 모의 sw 영략 테스트 수영장

Code Problem

1952. 모의 sw 영략 테스트 수영장

목표

- 주어진 다양한 요금제 중에서 **가장 적은 비용으로 수영장을 이용할 수 있는 방법**을 찾는 것이 목표.

문제 접근 방법

1. 동적 계획법 접근법
   - 매달 최소 비용을 계산하고, 그 결과를 누적하여 전체 최소 비용을 구하여 문제 해결
2. 백트래킹 접근법
   - 각 달마다 가능한 모든 요금제를 선택하여 재귀적으로 탐색하고, 최소 비용을 갱신하여 문제 해결.

1952. 모의 sw 영략 테스트 수영장

백트래킹 알고리즘 설계

1. 첫 번째 달부터 시작하여 모든 요금제에 대한 이용 요금을 탐색.
2. 각 달에 대해, 가능한 모든 요금제를 선택하여 재귀적으로 다음 달로 이동.
3. 모든 탐색이 끝난 후, 최소 비용을 갱신.
4. 탐색 도중 매번 현재까지의 비용이 최소 비용보다 크면 탐색 중단. (가지치기)

- 주의 사항

1. 3달 이용료의 경우, 조사 대상의 위치 역시 3달 뒤로 이동 할 수 있어야 함.

1952. 모의 sw 영략 테스트 수영장

코드 작성 (1/2)

1. data
   - 각 일자별 요금제 가격.
2. schedule
   - 1월부터 12월까지의 각 수영장 사용 일 수.
3. result
   - 최종 결과값.
   - 충분히 큰 수 = 1년 사용료
   - 최소 비용을 찾는 문제이므로
     반드시 result의 값이 최악의 경우로 초기화 할 필요 없음.
   - 어떠한 경우에도, 1년 이용료보다 비싼 경우는 유망성 없음.
   - 반대로, 어떠한 경우에도 1년 이용료보다 싼 경우가 있다면 충분한 비교 대상이 될 수 있음.

```
T = int(input())

for tc in range(1, T+1):
    # 하루, 한달, 세달, 1년
    data = list(map(int, input().split()))
    schedule = list(map(int, input().split()))
    result = data[3]  # 충분히 큰 값.
    search(0, 0)
    print(f'#{tc} {result}')
```

1952. 모의 sw 영략 테스트 수영장

코드 작성 (2/2)

```
def search(n, acc):
    global result
    if acc > result:    # 누적값이 결괏값보다 크다면 return
        return
    if n >= 12:             # 1년에 대해 모두 조회했다면
        if acc < result:    # 최솟값 초기화
            result = acc
    else:
        if schedule[n]:     # 스케쥴이 있다면,
            search(n + 1, acc + schedule[n] * data[0])    # 일로 계산
            search(n + 1, acc + data[1])                  # 월로 계산
            search(n + 3, acc + data[2])                  # 3달치 한 번에 계산
        else:               # 없다면,
            search(n + 1, acc)                            # 그냥 달만 증가
```