Dynamic Programming 8.2 -8.4

Q Index

8-2. 문제: 와일드카드(ID: WILDCARD, 난이도 중)

8-3. 문제풀이

8-4. 전통적 최적화 문제

- 최적부분구조
- 최적화 문제 에 레시피

8 문제

와일드카드(ID: WILDCARD, 난이도 중)

문제

와일드카드는 다양한 운영체제에서 파일 이름의 일부만으로 파일 이름을 지정하는 방법이다. 와일드카드 문자열은 일 반적인 파일명과 같지만, * 나 ? 와 같은 특수 문자를 포함한다.

와일드카드 문자열을 앞에서 한 글자씩 파일명과 비교해서, 모든 글자가 일치했을 때 해당 와일드카드 문자열이 파일명과 매치된다고 하자. 단, 와일드카드 문자열에 포함된 ? 는 어떤 글자와 비교해도 일치한다고 가정하며, * 는 0 글자 이상의 어떤 문자열에도 일치한다고 본다.

예를 들어 와일드 카드 he?p 는 파일명 help 에도, heap 에도 매치되지만, helpp 에는 매치되지 않는다. 와일드 카드 *p* 는 파일명 help 에도, papa 에도 매치되지만, hello 에는 매치되지 않는다.

와일드카드 문자열과 함께 파일명의 집합이 주어질 때, 그 중 매치되는 파일명들을 찾아내는 프로그램을 작성하시오.

- "?": 어떤 글자도 대응

"*": 0 글자 이상의 어떤 문 자열에도 대응

- 예시)

p : help(o)

: papa(o)

: hello(x)

he?p:help(o)

: heap(o)

: helpp(x)

8 3 문제풀이

완전탐색

```
bool match(const string& w, const string& s)
       int pos = 0;
       while (pos < s.size() && pos < w.size() && (w[pos] == '?' || w[pos] == s[pos]))</pre>
       //더 이상 대응할 수 없으면 왜 while문이 끝났는지 확인한다.
       if (pos == w.size())
               return pos == s.size();
       if (w[pos] == '*')
               for (int skip = 0; pos + skip <= s.size(); skip++)</pre>
                       if (match(w.substr(pos + 1), s.substr(pos + skip)))
                               return true;
       return false;
```

모든 경우의 수를 시도한다. 주어진 패턴이 m개의 *이면 (m+1)로 나누어서 생각 $(a?b*c*de?f*g) \rightarrow \{ (a?),(b*),(c*),(de?f*),(g) \}$ 패턴 w가 문자열 s에 대응하는지 w와 s를 앞에서부터 한 글자씩 대응해보면서 1.*를 만나거나 2. 어떤 문자열이 끝날 때종료하는 알고리즘(완전탐색) (pos가 대응하는지 확인할 부분이다)

- s[pos] != w[pos]이면 대응하지 않는거니까 당연히 종료
- S의 끝에 도달: 패턴은 남았지만 문자열이 이미 끝난 경우 당연히 종료
- 만약 패턴 w[pos]가 *이라면?
 → *가 몇 글자에 대해 대응할지 알 수 없기 때문에 남은 문자열의 길이까지 순회하며 모든 가능성 검사 w[pos+1:]를 패턴 w', s[pos+skip:]를 s'로 match(w',s') 재귀호출.

8 3 문제풀이

동적 계획법 Q: 왜 완전탐색으로 풀면 안되나?

- → (1) 패턴의 각 *에 대응되는 <<mark>글자 수></mark> 모든 조합의 가능성을 검사하기 때문에 문자열이 길고 *가 많아질 수록 너무 많은 경우의 수 탐색 필요
 - (**예시**) 와일드카드 : *********a, 비교할카드 : aaaaaaaaaab 어차피 끝이 달라 서로 대응될 수 없지만 <mark>완전탐색</mark>이므로 앞에서부터 모든 경우의 수를 살펴볼 것이다
- → (2) W와 S가 제한되어 있다. (제한크기가 100이므로 재귀함수 match의 호출횟수는 101*101 = 10201 중복 계산이 많을 것이다)

→ 메모제이션(캐시): w는 항상 주어진 패턴 W의 접미사임을 활용.

8 3 문제풀이

동적 계획법

```
int cache[101][101];
string W, S;
bool matchMemoized(int w, int s)
       int& ret = cache[w][s];
       if (ret != -1) return ret;
       while (s < S.size() && w < W.size() && (W[w] == '?' || W[w] == S[s]))</pre>
               ++W;
               ++s;
       if (w == W.size())
               return ret = (s == S.size());
       //4. *를 만나서 끝난 경우 : *에 몇글자를 대응해야 할지 재귀 호촐하면서 확인
       if (W[w] == '*')
               for (int skip = 0; skip + s <= S.size(); ++skip)</pre>
                       if (matchMemoized(w + 1, s + skip))
                               return ret = 1;
```

아까 전의 완전탐색과 같은 코드이지만

cache[와일드카드의 시작점][비교할카드의 시작점]

-1: 아직계산안함

0 : 계산했는데 match안됨

1:계산했는데 match됨

저장해두고 사용하여 시간단축을 할 수 있다. (또한, 문자열 대신 대응위치를 입력으로 사용한다)

최대 입력크기 n에 대해서 최대 n번의 재귀호출로 문제해결이 가능해서 시간복잡도 : 0(n^3)이다.

최적화 문제: 여러 개의 답 중 가장 좋은 답을 찾는 문제 해당 문제가 특정 성질을 성립할 경우 동적 계획법(메모제이션)보다 더 효울적인 DP 적용 가능하다

<최적 부분 구조(Optical Substructure)>

→ 부분 문제의 최적해만 있으면 전체 문제의 최적해를 구할 수 있는 경우.

(예시1) 서울에서 부산까지의 최단 경로를 찾는 문제. 이 때, 대전이 중간경로라면 전체문제를 (서울,대전) / (대전,부산) 두 개의 부분 문제로 나눌 수 있으며 각 부분 문제의 최적해를 알면 전체 문제의 최적해를 알 수 있다.

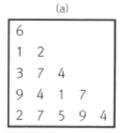
($\frac{0}{1}$ $\frac{1}{2}$) 문제1과 동일하지만, 통행료 합이 3만원 미만으로 최단경로를 찾고자 하며 (대전,부산)의 경로는 a(2시간 소요 , 만원),b(1시간 소요 , 2만원) 2가지가 존재한다고 할 경우

문제: 삼각형 위의 최대 경로 (ID: <u>TRIANGLEPATH</u>, 난이도 하)

문제

```
6
1 2
3 7 4
9 4 1 7
2 7 5 9 4
```

위 형태와 같이 삼각형 모양으로 배치된 자연수들이 있습니다. 맨 위의 숫자에서 시작해, 한 번에 한 칸씩 아래로 내려가 맨 아래 줄로 내려가는 경로를 만들려고 합니다. 경로는 아래 줄로 내려갈 때마다 바로 아래 숫자, 혹은 오른쪽 아래 숫자로 내려갈 수 있습니다. 이 때 모든 경로 중 포함된 숫자의 최대 합을 찾는 프로그램을 작성하세요.



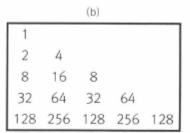


그림 8.6 삼각형 위의 최대 경로 문제의 예제 입력

바로 아래 숫자로 내려가거나 오른쪽 아래 숫자로 내려간다.

재귀호출로 완전 탐색:

$$path1(y, x, sum) = \max \begin{cases} path(y+1, x, sum + triangle[y][x]) \\ path(y+1, x+1, sum + triangle[y][x]) \end{cases}$$

pathSum (y, x , sum) 현재 위치 (Y,X) 지금까지의 경로합 (SUM)

문제: 삼각형 위의 최대 경로 (ID: TRIANGLEPATH, 난이도 하)

바로 아래 숫자로 내려가거나 오른쪽 아래 숫자로 내려간다.

재귀호출로 완전 탐색:

$$path1(y, x, sum) = \max \begin{cases} path(y+1, x, sum + triangle[y][x]) \\ path(y+1, x+1, sum + triangle[y][x]) \end{cases}$$

pathSum (y, x , sum) 현재 위치 (Y,X)

지금까지의 경로합 (SUM)

2^n -1 지금까지처럼 메모제이션을 적용해서 Cache에 경로합을 저장?

(b)

1
2 4
8 16 8
32 64 32 64
128 256 128 256 128

문제를 다시한번 생각해보면

(y,x)는 현재 위치 그리고 앞으로 거쳐가야할 조각들을 정의하고 (풀지 않은 문제) SUM은 이미 결정한 (해결한) 조각에 대한 정보.

- 그렇다면, SUM이 앞으로 어떤 경로를 갈지에 대해 영향을 미치는가?
- → 재귀함수에 SUM을 입력으로 주지 않아도 풀 수 있다.

```
path2(y, x) = triangle[y][x] + max
                                     path2(y+1, x+1)
import sys
input = sys.stdin.readline
C = int(input())
for _ in range(C):
   n = int(input())
    triangle = [list(map(int, input().split())) for _ in range(n)]
    for i in range(1, n):
        for | in range(i + 1):
            if i == 0:
                triangle[i][j] += triangle[i - 1][j]
            elif i == i:
                triangle[i][j] += triangle[i-1][j-1]
            else:
                triangle[i][j] += \max(\text{triangle}[i-1][j], \text{triangle}[i-1][j-1])
    print(max(triangle[-1]))
```

```
path1(y, x, sum) = \max \begin{cases} path(y+1, x, sum + triangle[y][x]) \\ path(y+1, x+1, sum + triangle[y][x]) \end{cases}
```

Sum이라는 정보가 맨아래줄까지 내려가는 문제를해결하는데 아무상관이 없다는사실을 파악!

-

지금까지의 선택과 상관 없이 각 부분 문제를 최적으로 풀기만 하면 전체 문제의 최적해도 알 수 있다

→ 최적부분구조

Q&A