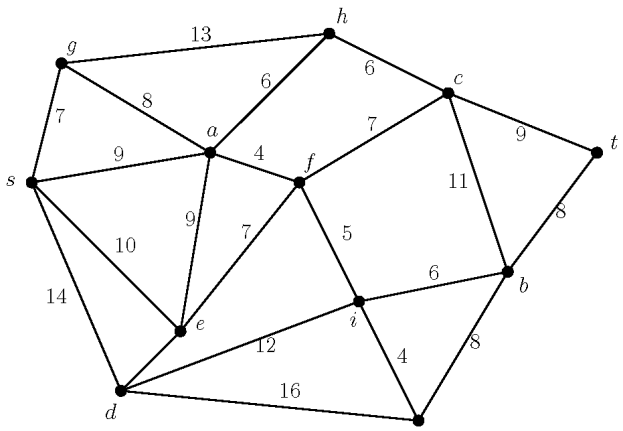
Lecture 14 : Greedy Algorithms (욕심쟁이법 알고리즘)

이번에 설명하는 Greedy 설계론(간단하게 그리디법)은 가장 직관적(Intuitive)이며 단순한[1](straightforward) 알고리즘 개발 방법론이다. 이 방법은 특별한 기술없이 누구나 쉽게 적용할 수 있는 기법이므로 난이도 높은 대회나 면접에서는 잘 나오는 형태는 아니다. 이 그리디 방법은 일단 문제에 접근하기 쉽기 때문에 현장에서는 초기해(initial solution)를 빠르게 구하는데 활용된다. 즉 주어진 문제가 얼마나 어려운 문제인지, 최종 구하고자 하는 답이 어느 정도인지를 빠르게 파악하는 것이 중요한 경우 이 방법을 사용하면 큰 어려움 없이 간단한 알고리즘을 쉽게 구현할 수 있다. 예를 들어 NP-complete[2] 류의 문제를 해결하기 위하여 다양한 방법, AI까지 동원될 경우가 있다. 이 상황에서 가장 단순한 방법으로 접근했을 때 어느 정도 수준의 답이 나오는지를 확인하는 것은 매우 중요한 단계의 작업인데 이때 Greedy Algorithm이 사용된다. 만일 이 단순한 알고리즘으로도 충분한 수준의 답이 나온다면 굳이 어려운 알고리즘이나 많은 resource를 사용하는 알고리즘을 개발할 필요는 없기 때문이다.

욕심쟁이 방법으로 최적 알고리즘을 구성할 수 있음을 증명하는 것은 이론적으로 큰 의미를 가지므로 여러분은 이 부분에 관심을 두고 이 장을 공부해야 한다. 욕심쟁이 방법은 하나의 거대한 방법론이므로 이 방법으로 접근하더라도 실제 개발되는 알고리즘의 수준은 다양할 수 있다. 그야말로 가장 "무식한" 수준의 욕심쟁이법도 있지만. 중간에 여러 지능적인 방법을 추가 활용하여 search space를 줄일 수 있다면 꽤 쓸만한 성능의 욕심쟁이 알고리즘을 얻을 수 있다. 이것은 일종의 heuristics라고 할 것이다. 무조건적으로 욕심쟁이법이 낮은 성능의 알고리즘이라고 인식하는 것은 조심해야 할 태도이다. 이후 현실적인 몇 문제를 사용하여 그리디 알고리즘의 개발과정과 그 효율을 알아보도록 하자.

어떤 도심의 도로망을 나타난 weighted geometric graph가 아래와 같이 있다. 우리는 지금 s의 위치에 있으며 목적지 t로 가야 한다.



만일 여러분이 s에 있다면 그 다음 s와 연결된 4개의 정점 {a, e, d, g }중에서 어

1) 단순하다(simple)기 좀 더 정확하게 말하자면 곧이곧대로 하는 표현이 더 어울리는 말이 될 것이다. 즉 문제를 설명하는 방식 그대로 적용하는 과정을 의미한다.

2) 최적 알고리즘의 시간 복잡도가 지수(exponential)인 문제를 말한다. 이는 이후 계산복잡도 이론에서 다시 설명될 것이다.

디고 가려고 할까? 그리고 왜 그 그곳으로 가려고 하는지 설명을 해보자. s에서 만일 g로 가는 사람이 있다면 누가 보더라도 좀 이상한 사람이라고 할 것이다. 보통의 상식을 가진 사람이라면 아마도 a로 갈 것이다. 물론 이것이 항상 정답을 보장해주는 것은 아니지만 대부분의 도로망에서라면 a로 가는 것이 가장 합리적인 경로가 될 것이다.

이 a를 선택한 여러분의 욕심을 구체적인 수식으로 한번 제시해보자. 그리고 그 원칙을 끝까지 고수하는 것이 바로 욕심쟁이 방법의 핵심이다. 가장 간단한 방법은 현재의 위치와 목적지를 연결하는 직선과 가장 가까이 있는 정점을 선택하는 것이다.[3)]

14.1 가장 직관적인 알고리즘 설계법 – 욕심쟁이(Greedy) 접근법.

a. 가장 간단하며 직관적으로 접근할 수 있는 알고리즘 설계법
b. 그리디(Greedy) 알고리즘으로 최적해(Optimal Solution)를 구할 수 있는 문제들
c. 개발된 그리디(Greedy) 알고리즘이 최적임을 알 수 있는 증명법
d. 그리디(Greedy) 알고리즘으로 최적을 구하지 못하는 다양한 예

14.2 Greedy Algorithm의 일반적인 구조

a. 어떤 알고리즘은 단계적인 선택을 거쳐서 전체를 완성한다고 하자.
b. 한 단계에서 주어진 정보가 { I } 이고 선택이 $\{C_1, C_2, \ldots C_k\}$라고 할 때 욕심쟁이 알고리즘은 현재 정보만을 이용해서 최선의 결과를 얻을 수 있는 경우를 선택한다. 단 현재 구성 중인 Solution은 고칠 수 없다.

c. Constructive Algorithm vs. Iterative Algorithm
   - 결혼하기(한번 결정하고 나면 되돌리기 어려움)
   - 친구 사귀기 (계속 개선의 여지가 있음)

d. 욕심쟁이 방법으로 해결되는 전형적인 문제 예 - 동전 거슬러 주기
   - 동전 거슬러 주기 (가장 작은 수의 동전으로)
   - C= { 100, 50, 10, 1 } 일 때 K 원을 지불하고 거슬러 주기
   - C= { 9, 7, 1 } 일 때 K=14원 지불하고 거슬러 주기

Q) 지금 동전 시스템으로 잔돈을 줄 때 결과가 optimal임을 증명하시오.
즉 동전이 {1, 5, 10, 100, 500}으로 구성된 동전 시스템을 말한다.

e. 국민 게임이었든 테트리스(TETRIS) 게임을 할 때에도 높은 점수를 얻은 방법에 그리디 알고리즘이 관련되어 있다. 테트리스(TETRIS) 고수들의 주장에 떨어지는 속독 빠른 경우에는 뒤를 생각하여 준비하는 복잡한 알고리즘보다 그리디하게, 즉 새로 내려오는 막대를 사용해서 가장 많은 층을 파괴하는 방법에 가장 현실적이며 실전적이라고 한다.

---

3) 이러한 routing을 미리 준비된 지도가 없는 경우에 최단거리를 찾아가는 online routing이라고 부른다. 그 대표적인 방법이 mid-point 우선 routing이다.

14.3 Greedy 알고리즘으로 시도해볼 수 있는 몇 가지 문제

- 화투판에서 점수 내기
- 훔친 물건 배낭에 넣어가기: 용량이 정해진 배낭에 많은 물건 담기

S = { 1, 4, 9, 34, 54, 57, 67, 95 } K(배낭)의 한계 = 150

S = { 2, 9, 13, 21, 25, 30, 43 }

K(배낭)의 한계 = 51

K(배낭)의 한계 = 70

K(배낭)의 한계 = 90

14.4 숫자판 게임 (위에서 아래로 내려가면서 가장 높은 "숫자 길"을 찾아내기) 단 밑으로 내려갈 갈 경우에는 현재의 위치에서 바로 아래, 바로 왼쪽 아래, 바로 오른쪽 아래, 즉 ↓ , ↙ , ↘ 방향으로만 갈 수 있다고 제한한다. 즉 jump나 back은 불가능하다.

-1	3	12	7	9	0
-3	-3	1	5	21	23
34	13	-10	-2	-32	6
21	5	-11	-6	3	7
6	-11	21	-23	-7	-45
17	21	-15	1	-99	-34

욕심쟁이의 선택은 다음과 같다. 먼저 가장 높은 12를 선택한다. 12는 바로 "현찰"이기 때문이다. 그 다음 12에서 그 아래로 갈 수 있는 {-3, 1, 5 }중에서 가장 높은 5를 선택하고 다시 그 자리에서 내려갈 수 있는 {-10, -2, -32}중에서 제일 큰 -2를 선택하고 이를 아래쪽까지 반복한다.

14.5 욕심쟁이법 알고리즘의 계산적 의미

- 가장 단순한 알고리즘(solution)으로 확보할 수 있는 해답의 품질 측정
- 개선한 알고리즘의 상대적 우수성 비교
- 초기 해(initial solution)의 빠른 구성
  이 초기해를 반복 개선하여 최적에 가까운 답을 찾는다.
- 만일 욕심쟁이법으로 최적 알고리즘을 찾을 수 있다면 그것을 증명하라.

Lecture 15 : 욕심쟁이법으로 최적해를 찾을 수 있는 문제

이 장에서는 단순한 욕심쟁이 방법으로 접근했을 때 최적이 알고리즘이 되는 몇 가지 문제에 대하여 살펴보고, 왜 그것이 최적을 보장하는지를 증명하고자 한다. 그 증명은 모두 귀류법을 사용한다. 즉 만일 이 방법을 적용했을 때 최적이 아니라고 가정하면 나타날 수 있는 모순(contradiction)을 찾아서 제시함으로서 해당 알고리즘이 최적임을 증명하게 되는 것이다.

여러분은 Greedy algorithm이 최적을 보장하는 문제의 구체적인 제약 조건에 대하여 주의를 기울여야 한다. 예를 들어 동전 바꿔주기 문제는 전형적인 NP-complete류의 문제이지만, 그 바꿔줄 동전의 크기가 특정한 조건을 만족시키는 경우, 즉 작은 단위의 동전이 바로 그 위 동전의 단위의 약수가 되면 그리디로 최적의 해답을 찾을 수 있다.

15.1 Job Scheduling (일정 구성하기)

- 각 과업을 처리하는 몇 개의 processor가 있다.
- 어떤 Task T_i 가 있다. 과업(task)의 처리 시간은 각각 다르다.

우리의 목적은 전체의 일거리(Task)가 처리될 때까지 “각 task"가 대기한 시간의 합을 최소화시키는 것으로 정했다.

e.g) 미장원에서 머리하기, 기차표 사기, 햄버거 사기, Quad Core CPU

- 목적함수 = { a. 전체 시간의 최소화, b. 각 Job 대기시간의 최소화}
- processor(Queue)가 한 개인 경우와 여러 개가 있는 경우

e.g) $T_1 = 5,\quad T_2 = 10,\quad T_3 = 4$ 일 경우 아래와 같은 순서일 때 총 대기시간.

대기순서	총대기 시간의 합
[1, 2, 3]	5 + (5 + 10) + (5 + 10 + 4) = 39
[1, 3, 2]	5 + (5 + 4) + (5 + 4 +10) = 33
[2, 1, 3]	
[3, 1, 2]	4 + (4 + 5) + (4 + 5 + 10) = 32
[2, 3, 1]	
[3, 2, 1]	

만일 5개의 task들의 수행시간이 각각 { 6, 2, 7, 11, 1 8 } 이라고 한다면 여러분은 어떤 task부터 처리할 것인가. t=11 짜리 task 부터 처리하면 나머지 4개의task들 모두가 11분씩으로 더 기다려야 한다. 만일 t=1 짜리부터 처리한다면 나머지는 겨우 1분간의 대기시간만 늘어나게 된다. 직관적으로 대기실에 ”쳐박혀“

기다리는 작업시간을 줄이는 일, 즉 대기실에서 기다라는 task를 한개라도 더 줄이기 위해서는 빨리 빨리 대기줄에서 ”쫓아내는 것“이 가장 직관적인 전략이 된다. 그러면 어떤 순서, 어떤 기준으로 처리해야 할 것인가?

15.2 마감시간(due time)과 처리 이익(profit) 있는 Job Scheduling 문제
목적함수는 마감시간 안에 마친 작업의 이득의 총합을 최대화 하는 것이다.

작업	**T1**	**T2**	**T3**	**T4**	**T5**	**T6**	**T7**
마감시간	3	1	1	3	1	3	2
이익	*40*	*35*	*30*	*25*	*20*	*15*	*10*

Q) 이 문제를 그리디(greedy) 알고리즘으로 푼다면 어떻게 풀어야 할까 ?
목적은 제한된 시간에 가장 많은 이익을 얻는 순서를 정하는 것이다.

15.3 File Merging (비유하자면 무거운 돌탑의 각 부분을 최소의 힘으로 옮기기) : 서로 다른 크기의 sorted된 파일이 있다. 이 파일들을 두 개씩 묶어 전체를 하나의 sorted된 파일로 만들고자 한다. 목적함수는 전체의 비교횟수를 최소화 하는 것이다. 어떻게 하면 될지 생각해 봅시다. (서로 겹쳐질 수 있는 쓰레기 통 10개를 가장 빠른 시간에 포개기 문제와 유사함.)

15.4 커다란 돌판(Stone Plate)을 움직여 쌓아올려 석탑을 만들려고 한다. 어떤 돌을 어떤 순서로 움직이는 것이 가장 힘이 적게 드는 것인지 그 순서를 생각해보자.
w 무게의 돌을 L meter를 움직이는데 필요한 에너지는 $w \cdot L$ 이다.

무게	10	3	25	8	50	6
위치	1	2	3	4	5	6

15.5 Partial Knapsack (물품의 일부를 잘라서 넣을 수 있는 경우) : 어떤 물건을 용량이 정해진 배낭에 담아가려고 한다. 단 물건의 부피는 제각각이며 배낭의 부피는 C로 제한되어 있어 그 이상 담아가면 배낭이 터진다. 만일 배낭의 부피가 C=16 이라면 어떻게 해야 할지 생각해보자.

물건	A	B	C	D	E	F	G
부피	*9*	*15*	*5*	*7*	*12*	*3*	*2*
이득	*11*	*10*	*9*	*3*	*13*	*7*	*5*

여러분이 생각하는 Greedy Algorithm을 제시해보시오. 그것이 최적(Optimal)을 보장해줍니까 - Greedy Packing이 최적을 보장해주는 물건의 부피를 나타내는 값에는 특성이 있는지? 만일 부피당 이득이 다르다면 어떻게 넣어가야 할까요?

- 해외여행에서 남은 돈으로 마지막 면세점에서 선물 구입하기. 면세점 문제:
  귀국할 때 쓰고 남은 돈으로 75.8 유로가 있음을 확인했다. 우리는 이것을 모두 소진하려고 한다. 어떻게 하면 가장 알차게 소진할 것인지 아래 문제를 풀면서 생각해보자. 즉 만족도를 최고로 높이고자 한다.)

물품	**가격**	**만족도**
초코렛	14	9
양주	34	21
가방	50	24
껌	4	1

물품	**가격**	**만족도**
O1	24	6
O2	40	8
O3	5	5
O4	14	7
O5	20	4
O6	21	3

Lecture 16 : Minimum Spanning Tree (최소 연결 트리)

Minimum spanning tree 구하는 문제는 대표적인 그래프 알고리즘의 문제이기도 하지만 greedy algorithm으로 답을 구할 수 있는 몇 안되는 최적 알고리즘 중 하나이다.

16.1 Minimum Spanning Tree Problem

a. 입력: edge weighted graph $G(V,E)$
   출력: a subgraph with tree structure

b. Spanning의 의미는 무엇인가 ?

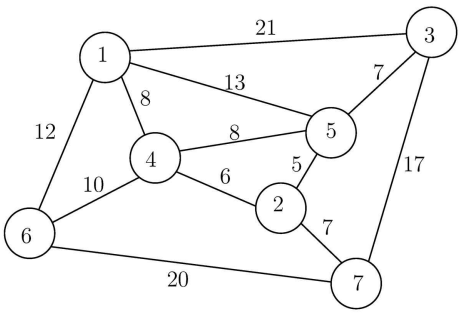
c. Spanning Tree를 위한 대표적인 두개의 알고리즘은 ?

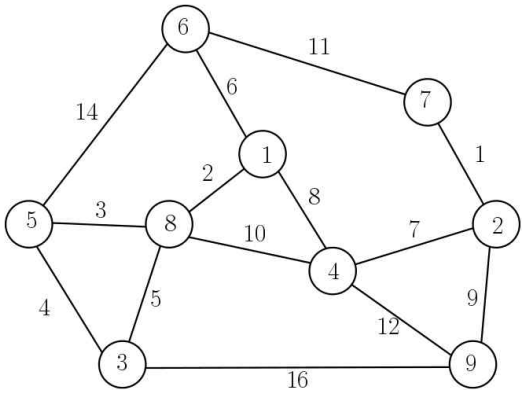
d. Spanning Tree가 응용되는 예: {죄수탈출, 간단한 Clustering }

e. Spanning Tree에서 각 edge weight의 의미
   각 쌍의 거리는 최적이 아니다. 전체의 합이 최소

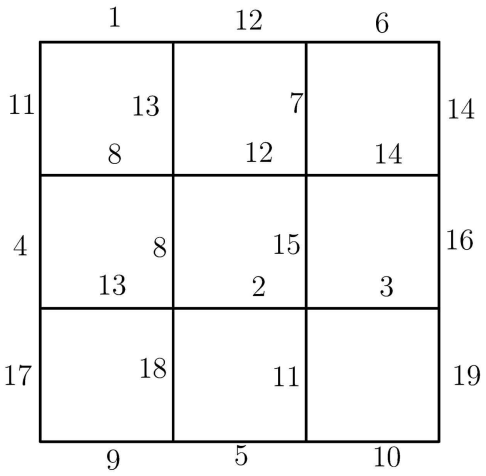
f. Steiner Tree와 Minimum Spanning Tree의 차이:
   특정한 몇 개의 노드를 연결하는 트리 중에서 가장 weight가 작은 트리를 구하는 문제

16.2 아래는 어떤 도시의 연결망이다. 이 도시에서 저수지는 (4)번 정점에 있다. 모든 지역에 물 공급하기 위한 최소의 비용, 사용된 연결 파이프의 총 길이는?





16.3 아래는 그리드로 표현된 9개의 독방을 가진 감옥을 나타내고 있다. 죄수는 모두 9개의 개별 cell에 갇혀있다. 여기에서 9명의 모든 죄수가 탈출하기 위해서 최소의 노력은 얼마인지 계산해보자. 각 벽에 붙어있는 숫자는 그 벽을 뚫기 위한 최소의 노력을 나타낸다. 예를 들면 그 벽에 구멍을 뚫는데 걸리는 시간(days)이라고 생각하면 된다.



16.4 Greedy Algorithm으로 Minimum Spanning Tree 구성하기

a. Kruskal의 MST 알고리즘

i) 모든 edge를 weight로 sorting을 한다. O(m log m)

ii) 가장 작은 weight의 edge를 하나씩 T에 추가한다.

iii) 만일 T + (u,v)에서 cycle이 생기면 버린다. ⇐ 이것을 어떻게?

iv) 이 짓을 전체가 $n-1$개의 edge가 될 때까지 계속 한다. (keep going)

b. 문제는 iii)번 확인 작업을 얼마나 빨리 구현하는가에 있다.

16.5 Kruskal 알고리즘이 올바르게 동작함(correctness)을 증명하기

- Cut property: 어떤 Tree에 있는 tree edge가 아닌 edge *(x,y)*를 이으면 반드시 cycle이 생긴다. 그런데 그 edge에 있는 모든 edge의 Weight은 (x,y)보다 크다.

16.6 Kruskal Algorithm Time Complexity의 주요 time bottleneck.

- 어떤 edge (x,y)를 추가 했을 때 어떤 현상이 발생하는가 ?
  그것이 새로운 cycle을 만드는가 ?
- Lemma : Tree edge가 아닌 edge를 추가하면 반드시 cycle이 생긴다.

16.7 Minimum number of increasing subsequences. 어떤 수열 S가 있다. 이 서열 S를 increasing subsequence로 분해(decompose)하려고 한다. 가장 적은 수의 increasing subsequence로 하는 방법을 찾으시오.

예) S= [3 34 12 45 6 9 21 40 32 60 12 8 40]

예) S= [2 60 12 8 40 3 34 12 45 6 9 21 40]

예) S= [11 3 22 5 6 19 21 20 12 16 2 8 22]

예) S= [13 14 5 6 9 21 24 22 16 18 28 24]

16.8 Minimum Swaps for Bracket Balancing. 여러분은 N개의 '['기호와 N개의 ']'의 기호로 구성된 문자열이 있다. 이 문자열을 고쳐서 balanced bracket을 만들고자 한다. 단 허용되는 작업은 'swap'이다. 즉 이웃한 두 문자의 위치를 바꿀 수 있다. 최소의 작업으로 주어진 Bracket 문자열을 balanced로 만드는 알고리즘을 제시하시오.

```
Input  : [ ] ] [ ] [ : Output : 2
First swap: Position 3 and 4
[ ] [ ] ] [
Second swap: Position 5 and 6
[ ] [ ] [ ]
```

```
Input  :  [ [ ] [ ] ]
Output : 0
String is already balanced.
```

16.10 **[케이블 문제, Cable Connection Problem]**

몇 가지 기본 단위 케이블을 이어서 길이 L인 케이블을 만들어야 한다. 먼저 두 종류의 케이블 A, B이 준비되어 있고 각각의 길이는 각각 L_A, L_B라고 하자. 그리고 단위(unit) 케이블은 더 짧은 케이블을 위하여 자를 수 없다. 여러분은 몇 개의 단위 케이블을 골라 이어서 전체 길이가 L인 케이블을 만들어야 한다. 단 가능하면 그 이음매의 수가 최소가 되도록 해야 한다. 따라서 직관적으로 생각해볼 때 가능하면 긴 단위 케이블을 사용하는 것이 유리할 것으로 예상된다.

예를 들어 L=24, L_A=3, L_B=4인 경우를 생각해보자. 즉 길이 24인 케이블을 3개, 4개 단위로의 케이블로 적절하게 이어서 만들어야 한다. 이 경우 2가지 경우가 가능하다. 24 = c_1*3개 + c_2*3개 되므로 A 3개, B 3개를 사용하면 된다. 또는 3*8도 가능하다. 전자는 모두 5번의 용접 작업이 필요하지만 후자는 7번의 작업을 해야 한다. 만일 어떻게 조합을 해도 L을 맞추지 못한 경우도 있는데 이 경우에는 “불가능”하다고 답을 해야 한다. 이 상황에서 다음 문제를 풀어보시오.

L	L_i	조합 방법
20	3 11 15	
50	7 19 31	
70	3 11 13 35	
87	12 15 17 30	
110	5 33 35 40 60	

L	L_i	조합 방법
151	3 11 15	
200	7 19 31	
300	3 11 13 35	
400	11 70 111	
900	22 33 55 101	