Lecture 25 : Branch and Bound의 실전 응용[1)]

속담에 “한 개의 우물만 파는 사람이 성공한다” 라는 말이 있는데 생각해보면 약간 무책임한 말이 아닐 수 없다. 한 우물만 파서 성공할 수 있으려면 작업 중인 땅 아래에 진짜 쓸만한 수맥이 존재하는 경우이다. 수 십미터, 100여 미터를 파가면서 내려가도 물 한 방울 없는 상황에서 더 깊이 무조건 깊이만 파보라고 조언하는 것은 무책임한 말이 아닐 수 없다. 속담에는 이런 식의 결과론적인 주장이 많다. 예를 들어 10번 찍어 안 넘어가는 나무가 없다는 둥. 이 말을 믿고 나무를 찍어 보지만 결과는 보장받을 수 없는 것이다. 결국 “한 우물만 파기” 방법론은 전혀 알고리즘적인 주장이 아니다. 공무원 시험을 10년째 도전하고 있는 사람에서 추가로 5년간 더 해보라고 하는 것은 현명한 조언이 아니다. 10년간 공부해서도 시험의 합격권 근처에도 가지 못하는 사람은 그러한 류의 시험에는 재주가 없다고 해야 할 것이며 그 정열을 다른 곳에 투자하는 것이 현실적인 방책이 된다. 강사만 고안한 “Zoh's 우물론"의 구조는 다음과 같다.

한 우물만 팔 것인가 여기저기 마구 조금씩 팔 것인가를 고민하기보다는 이미 상당히 깊이로 파인 우물 중에서 가장 가능성이 있는 우물을 골라서 그곳부터 조금씩 파보는 것이 가장 현실적인 방책이 된다. 알고리즘 문제를 풀 때로 마찬가지이다. 주어진 문제를 해결하기 위한 초기에는 매우 다양한 관점을 시도해 본다. 그 결과 중에는 좋은 것도 있겠지만 그렇지 못한 것도 있을 것이다. 그 경험을 바탕으로 가능성이 있는 몇 개를 골라 다시 그 방법을 확장해가면서 끊임없는 더 가능성이 있는 방법을 조금씩 확보해 나가는 것이다. 이것이 바로 Branch and Bound(BB) 방법의 근본적인 철학이다. 요약하자면 “조금씩, 계속 확인해가면서”이다. 이 기법 중에서 가장 중요한 단계는 바로 bounding, 즉 가능성이 없는 경우의 수를 표기하는 기술이다. 더 이상 가망성, 또는 지금의 현실보다 더 나은 것을 주지 못하는 도전을 바로 포기해야 한다. BB의 성능은 내가 확보한 지금의 이득과 비교해서 가망성이 없는 노드(state)를 얼마나 잽싸게 포기하는가, 즉 다시는 고려 대상에 넣지 않고 버리는 기술의 정교함에 달려있다. [2)]

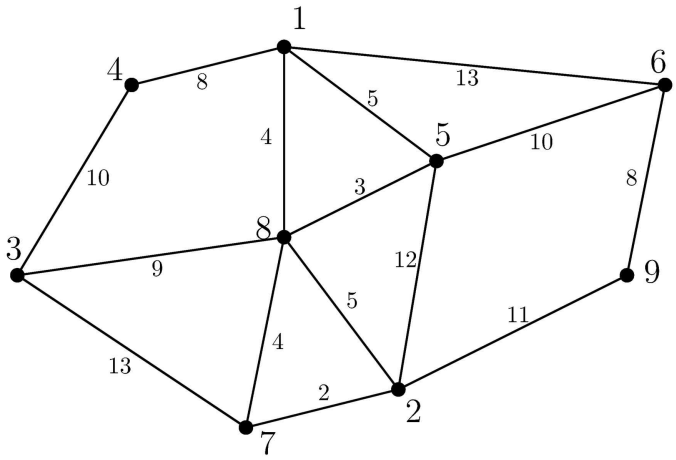
고려 중인 여러 대안을 빨리 버리는 기술, 즉 bounding하는 기법에는 2가지 접근법이 있다. 하나는 최대로 확보 가능한 이득을 결과를 빨리 확정하는 것이다. 이것이 높아지면 자연스럽게 포기되는 중간이 늘어나게 된다. 예를 들어 만일 여러분이 현재 연봉 6000인 기업에서 일을 한다면 그보다 낮은 어중간한 기업으로의 이적은 모두 고려대상에서 제외될 것이다. 그것이 고려될 경우는 현재는 연봉 3000이지만 앞으로 1억까지 올라갈 가능성이 있는 경우이다. 이런 기업은 bounding되지 않는다. 그러나 연봉 2500인 기업에 있다면 그보다 연봉이 높은 대부분의 기업은 고려대상이 되므로 그것 중에 더 좋은 것을 골라야 하는 작업에 많은 계산을 해야한다. 그 다음 중요한 BB의 기술은 지금의 방법으로 얻을 수 있는 이득의 최대값을 빠르고 정확하게 짐작(계산)하

1) a.k.a “가능성이 없는 일에 매달리지 말자”. 할 수 있는 것만 잘하기에도 바쁜 세상이다.
2) 가능성 없는 일에 더 이상의 미련을 두지 않은 것은 세속을 행복하게 살아가는 중요한 지혜이다.

는 것이다.

먼저 NP-complete 문제의 대표 격인 Traveling Sales Person(TSP) 문제를 BB로 풀어가는 과정을 통하여 이것이 어떻게 전개되고 활용되는지를 살펴보자.

25.1 TSP문제는 어떤 그래프에서 전체 노드를 방문하고 다시 처음으로 돌아오는 길 중에서 가장 짧은 것을 찾는 것이다. 단 이 과정에서 한번 방문한 노드를 다시 방문해도 상관이 없다. 이 문제에서 bounding을 위한 가장 중요한 point는 지금까지의 solution을 바탕으로 bound를 구하는 것이다. 즉 만일 내가 1 → 5 → 2 → 7 → 3까지 왔는데 이것으로부터 얻을 수 있는 가장 좋은 답(길이)이 현재 내가 가지고 있는 값, 예를 들어 115보다 더 긴 130 경로가 있다면 이쪽으로 확장되는 모든 경우는 bound를 시켜야 한다.



25.2 어떤 학생이 5개의 과목을 수강해야 한다. 각 과목은 서로 다른 5명의 강사가 담당하는데, 학교 규칙상 같은 강사에게 2개 이상의 과목은 수강할 수 없다. 여러분이 가장 높은 성적을 받으려면 어떤 강의를 들어야 할 것인지 정하시오. 박스 속에 있는 값은 예상되는 성적이다. 점수로 본다면 A-95, B-80 C-60, D-40이며, F=0, 그리고 '+'는 +5, '-' 마이너스는 -5이다. 즉 A+=100점이다.

a) 성적이 제일 좋다면 ?
b) 가장 최악의 성적은 ?
c) 예상되는 평균은 ?

	P1	P2	P3	P4	P5
C1	A+	F	B-	C-	D
C2	C-	C+	F	B	F
C3	B-	A+	B-	C-	C
C4	A+	D+	D	F	B+
C5	B-	C-	B+	B	C-

a) 성적이 제일 좋다면 ?
b) 가장 최악의 성적은 ?
c) 예상되는 평균은 ?

	P1	P2	P3	P4	P5
C1	C+	D-	A+	A0	B+
C2	B-	F	C+	F	C
C3	A-	C+	B-	A-	B+
C4	C+	B+	C	B+	C+
C5	B-	C-	B+	F	F-

25.3 다음과 같은 거리 행렬(distance) matrix가 있을 때 어떤 외판원이 A에서 시작하여 4개의 지점을 돌고 다시 자신의 자리로 돌아오는 가장 짧은 길의 길이를 구하고자 한다. 길은 대칭적이지 않다. 즉 *dist*(A,B)와 *dist*(B,A)는 서로 다른 시간이 소용된다. 두 경우에 대하여 각각 풀어보자.

	A	B	C	D	E
A	*0*	*5*	*2*	*1*	*6*
B	*1*	*0*	*4*	*5*	*8*
C	*3*	*6*	*0*	*7*	*9*
D	*17*	*13*	*9*	*0*	*11*
E	*4*	*5*	*6*	*7*	*0*

a) 최단 경로의 Lower bound는 얼마이며 어떻게 구할 것인가?
b) 하나의 upper bound는 어떻게 구할 것인가?

25.4 Branch and Bound는 과연 어떻게 Lower bound를 좀 더 실제와 가깝게 구하는 가에 달려있다. Lower Bound는 높을수록, 즉 실제의 현실치와 가까울수록 효용가치가 높다.

- $s-t$ Shortest Path
- TSP (Traveling Sales Person) 문제
- 연봉예상 (최저 연봉은 1000이다. 최고 연봉은 10억이다.)

25.5 취업을 코앞에 둔 졸업예정자들에게 가장 유용한(useful) 정보는 무엇인가요? (최악의 경우를 생각해서 각 경우를 나열해 봅시다. 또는 bound)

a) 당신의 연봉은 3억보다 작다.
b) 당신의 연봉은 5000만원 이하다.
c) 당신의 연봉은 최소 2400이며, 실적에 따라 추가 3000만원의
보너스가 있다. 작년 보너스는 최고 3000, 최저 100, 평균 800이다.
d) 당신의 연봉은 정확하기 3200이다. 해고되지 않는다면
e) 당신의 연봉은 아직 정해진 바가 없다. 그러나 섭섭지 않게 줄 것이다.

25.6 Branch and Bound를 설계할 때 주의할 점

- 빠르게 "예상 이득의 최대치(upper limit)"을 구할 수 있어야 한다.
- 예상하는 시간이 그냥 마구 계산하는 시간보다 더 걸리면 (비용) 그 알고리즘은 무의미하다. [3]
- 두 결혼대상자 중에서 좀 더 나은 사람을 선택하는 데 드는 비용이 다른 사람을 새로 구하는 데 드는 비용보다 비싸다면 어떻게 될까?

25.7 [도전문제] 졸업과제에 활용된 문제

한국 프로야구에서 10팀의 적절한 매치 업(match up) 일정 계산하기[4]
목적함수 1 : a) 전체 팀의 이동 거리의 합의 최소화
b) 가장 많이 움직여야 하는 팀의 이동 거리의 합의 최소화[5]
c) 동원되는 관중의 수, 어린이날 등 중요 공휴일에는 큰 경기장을 배정하여 이익을 최대화해야 한다.
d) 같은 팀끼리 연속해서 시합을 하지 않도록 하는 일. 특정 기간에 각 팀이 다른 팀과의 시합의 수는 비교적 균등해야 흥미를 돋울 수 있다.

25.8 Branch and Bound 알고리즘 설계법은 계산복잡도가 매우 높은(NP-Complete) 류의 문제를 푸는 가장 보편적인, 일반적인 방법론이다. 즉 어떤 문제든 이 방법을 이용하면 반드시 그 최적해를 구할 수 있다. 단 그 계산시간은 backtrack보다는 짧으나, 운이 나쁘면 거의 같을 경우도 있다. 즉 별 이득이 없다는 말이다.

이 방법은 일반적인 방법이므로 반드시 프로그램 과제를 이용해서 충분히 익혀둘 필요가 있다. 그러나 worst case에 exponential complexity를 피할 수는 없다. 즉 B&B는 polynomial time 알고리즘이 아니며 testing data와 bound 전략에 따라서 시간이 들쑥날쑥하다.

---

3) 전철을 타는 것이 빠른지 버스를 2번 갈아타는 것이 빠른지를 1시간 동안 계산하고 앉아있는 것이 좋은 예이다. 10분을 더 아끼기 위하여 1시간을 고민하는 것은 어리석은 일이다. "부먹 찍먹 고민말고 한 개라도 더 먹자" 이 생각을 떠올리면 된다.

4) 이러한 문제를 Sport tournament scheduling이라고 부르며 미국과 같이 프로 스프츠 팀의 종류도 많고, 팀의 수도 많을 경우 시합을 위하여 이동거리가 긴 경우에는 꼭 필요한 연구 주제이기도 한다.

5) 프로야구에서 롯데 자이언츠 팀의 경우 가장 이동거리가 길며 KBO에서는 이점을 별로 고려하고 있지않는 듯하다. 이 경우 전체 팀의 이동거리는 좀 증가하더라고 가장 많이 이동하는 롯데의 이동거리를 줄여주는 것이 합리적이다.

Lecture 26 : 분기(Branching)의 원칙

26.1 우리는 보물섬에서 보물을 찾아내었다. 이것을 들고 이 보물섬을 탈출하고자 한다. 문제는 이 보물섬이 아주 심한 정글로 되어있어 빠져나가기가 쉽지 않다는 것이다. 여러분은 가장 적은 힘을 들이고 빠져나가는 길을 찾아 나가는 것이다. (따라서 현재의 위치에서 상하좌우의 값만 알 수 있다.)

4	5	12	9	7	8	5
1	3	4	5	6	8	3
9	11	6	4	5	7	11
3	2	6	S	3	7	2
10	7	16	13	5	13	7
9	3	1	2	7	8	3
9	11	6	4	5	7	11

- 단 여러분이 확보하고 있는 <지도>는 없다.
- 여러분은 현재의 위치에서 인접한 4 방향의 cell값만 알 수 있다.

"길 값" = 현재까지의 고생 값 + 앞으로 예상되는 고생 값

26.2 이것은 결국 일반적인 탐색 방법(Searching method)도 귀결된다.
- 인공지능에서 말하는 A* 알고리즘 = Branch and Bound이다.
- 만일 optimal을 찾아내지 않아도 된다면 다양한 Heuristics가 존재한다.

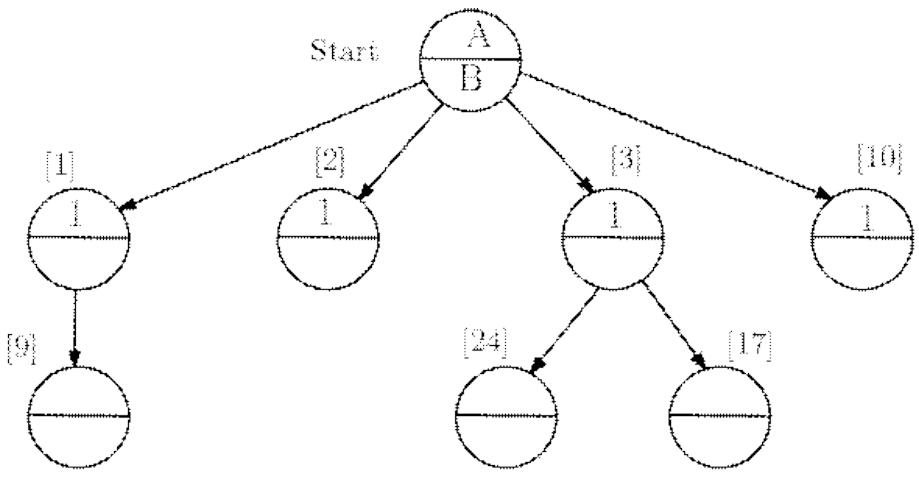
26.3 어떤 통에서 1에서 100까지의 값이 쓰인 카드를 꺼내고자 한다. 단 그 횟수는 3번으로 제한되어 있다. 마지막으로 선택한 값이 자신의 값이 된다. 단 그 안에 들어있는 숫자가 uniform 분포라고 하는 가정은 없다. 아무런 가정은 없다. 처음 꺼낸 수가 54였다. 여러분은 선택은?

Branch and Bound 실습자료: 미로에서 가장 빠른 길 찾기

동수는 S(start) 지역에서 탑이 있는 T(tower)까지 최단 경로를 통하여 가고 싶어 한다. 단 동수에게 전체 지도(map)는 주어져 있지 않으며 동수는 자신의 현 위치에서 이웃한 4개의 지역(상하좌우)의 상태만 알고 있을 뿐이다. 이 문제를 Branch and Bound 방법으로 해결하고 그때의 상태 트리(state tree)를 그려보시오. 각 cell의 번호는 해당 지역의 주소이다.

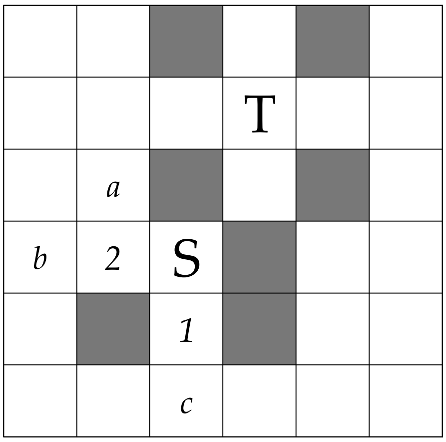
22	11	25	12		21
23			T	5	6
9	1		13	14	15
10	S	2			7
24	3			20	16
	17	18	4	19	8

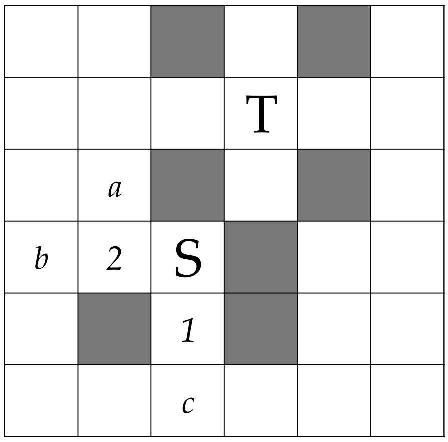
검은색으로 표시된 부분은 다른 건물이 있는 지역이므로 이 지역을 통과할 수는 없다. 두 지점 (p, q)와 (r, s) 간의 거리는 직교 거리 $|p-r|+|q-s|$로 계산된다. 각 branch node는 두 가지의 값 (A, B)를 가지는데 A는 S에서 그 지역까지의 거리, B는 현재 위치에서 T까지 가는 거리의 예상 하한치이다. 이 하한치를 계산할 때 우리는 $|p-r|+|q-s|$를 사용한다. 다음 트리를 완성하여 전체 공간을 탐색하시오.

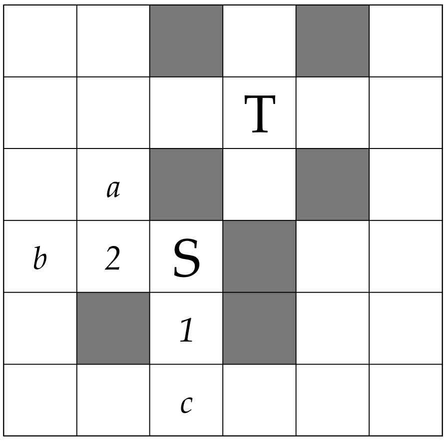


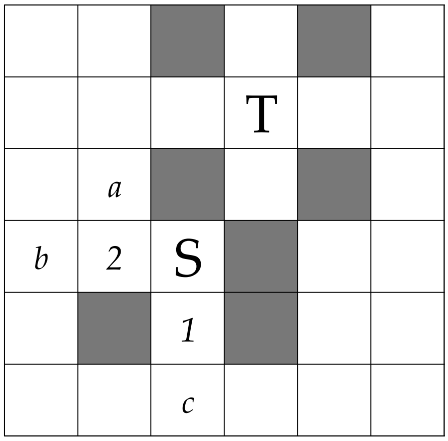
다음 공간에서 이 문제를 Branch and Bound로 풀어보고 Bound 되는 노드를 모두 표시해보자. 이 작업을 S로부터 방문 되는 cell을 번호순으로 표시하시오. 단 같은 예상 거리에 있으면 위, 오른쪽, 아래, 왼쪽 순서를 선호하여 표시한다. 아래 그림에서 1, 2

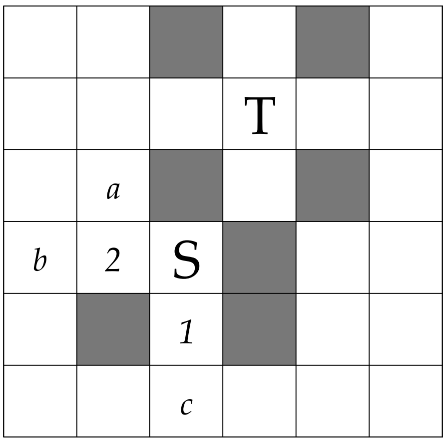
번 cell은 S에서 처음 방문 되는 후보 cell임을 보여 주고 있다. 이제 여러분이 branch 할 노드 집합은 { a, b, c }이다. 어떤 노드를 가장 먼저 고려해야 할지 계산해 보시오.

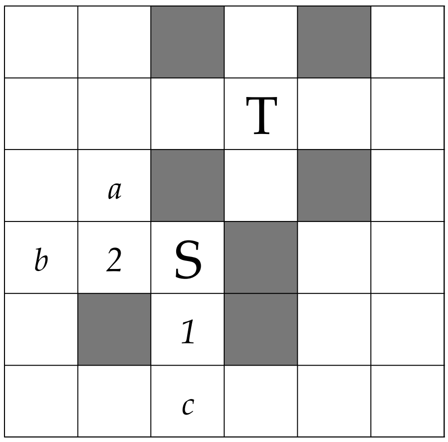












두 번째 문제. 과제와도 관련이 있는 6개의 가맹점 분점(음식점)으로 구성된 도시에서 아침에 모든 지점에 음식 재료를 배달하고 다시 본사로 돌아오는 제일 좋은 길을, 즉 가장 짧은 길(shortest cycle)을 찾으려고 합니다.

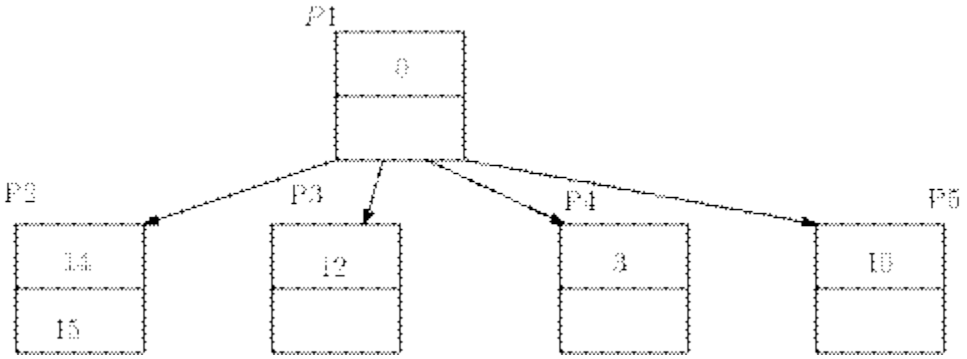
본사는 1번 위치에 있고 각 지점에서 다른 지점까지 가는 시간은 대칭적이지 않습니다. 예를 들면 일방통행로 또는 시간에 따라 달라지는 차량 흐름 때문에 그렇습니다. 이 문제를 Branch and Bound로 풀어보고자 합니다. 아래 행렬은 두 분점 (R_i, R_j) 사이의 거리를 나타내는 행렬입니다.

	R_1	R_2	R_3	R_4	R_5
R_1	*0*	*14*	*12*	*3*	*10*
R_2	*12*	*0*	*7*	*15*	*5*
R_3	*9*	*4*	*0*	*6*	*8*
R_4	*2*	*9*	*10*	*0*	*9*
R_5	*13*	*6*	*15*	*7*	*0*

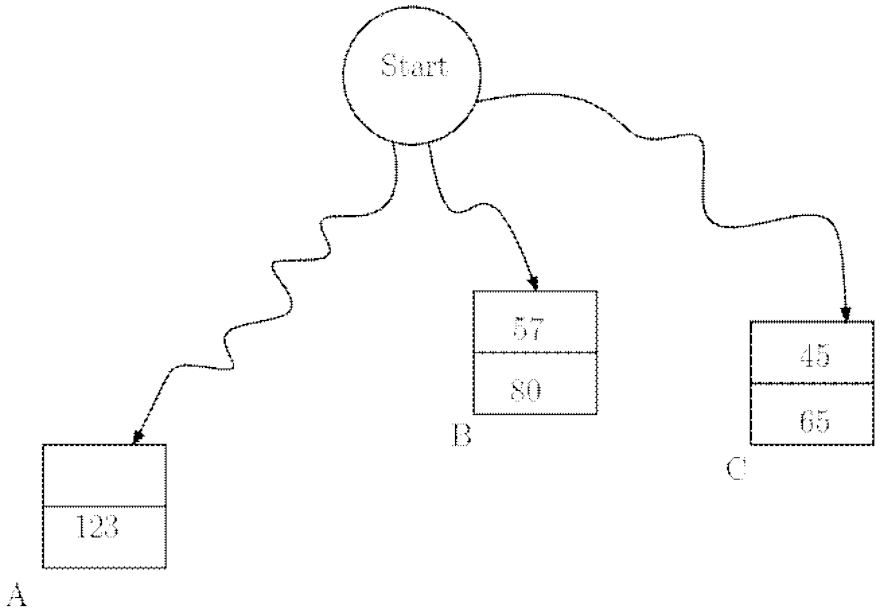
1에서 2로 왔을 때 즉 1⇒2일 때 나머지 { 3, 4, 5 }를 모두 거치고 다시 1번으로 돌아오는 길 중에서 가장 짧은 길의 하한치를 찾아보는 것입니다. 즉 { 2, 3, 4, 5 } 중 하나를 시작점으로 두고 { 3, 4, 5, 1 }에서 구성된 submatrix 에서 하한치를 구하는 것입니다.

2번에 연결된 { 3, 4, 5 } 중에서 가장 짧은 길은 (2,5) edge.
3번에 연결된 { 4, 5, 1 } 중에서 가장 짧은 길은 (3,4) edge.
4번에 연결된 { 3, 5, 1 } 중에서 가장 짧은 길은 (4,1) edge.
5번에 연결된 { 3, 4, 1 } 중에서 가장 짧은 길은 (5,4) edge.

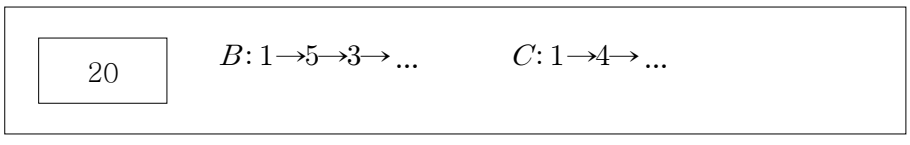
따라서 2에서 { 3, 4, 5}를 거쳐서 1번으로 돌아오는 모든 길의 길이는 아무리 짧아도 5+6+2+7=20보다 더 짧을 수는 없습니다. 따라서 1⇒2 를 통하여 돌아오는 길의 경로 하한치는 14 + 20 = 34가 되는 것입니다. 자 이제 같은 식으로 1⇒3 인 경우, 1⇒4, 1⇒5 의 경우에 대해서도 예상되는 최단 거리를 구해보자.



왜 이렇게 복잡하게 계산하는지 그 이유를 설명해주는 example을 하나 보여 주겠습니다. 이 복잡한 작업의 목적은 가능한 빨리 bound를 시키기 위해서입니다.



S는 시작점이며 A는 이미 S에서 시작하여 S도 돌아온 실제로 존재하는 어떤 경로입니다. 그 길이는 [123]으로 이미 최종 결판이 난 상태입니다. 확보되었다는 말이지요. 그런데 이제 B와 C와 같이 중간에 반쯤 계산된 노드를 살펴봅시다. 예를 들어 6개 지점이 있는 경우에 A는 1→6→5→2→3→4→1 이라고 합시다. 그리고 B와 C는 각각 아래와 같다고 합시다.



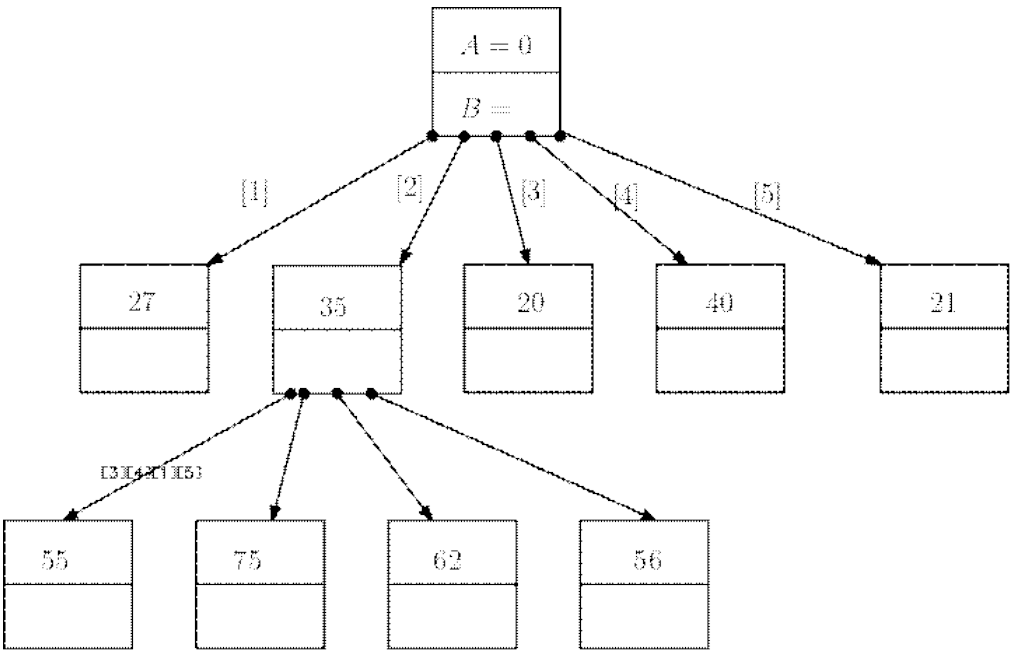
여러분은 B의 경우 그 다음 어떤 상태를 살펴볼 필요가 있다고 생각합니까? C에 대해서는 어떻게 생각합니까? 이것이 바로 Branch and Bound 기법의 핵심이며 예상되는 모든 경우의 하한치를 구하는 것이 빠르게 알고리즘을 종료시키는 가장 중요한 핵심기

술이 됩니다.

마지막으로 0/1 Knapsack의 예를 들어 Branch and Bound 문제를 살펴보도록 합시다. 이전 수업 시간에 사용한 예제를 좀 고쳐 사용하기로 합니다. 우리가 가진 공간의 크기 W=45 이라고 합시다. 즉 우리는 부피 45를 넘지 않으면서 가장 많은 이득을 가지도록 물건을 잘 선택해서 채워 넣어야 합니다.

물건	부피	이득	이득/부피
C1	27	27	1.0
C2	20	35	1.5
C3	10	20	2.0
C4	16	40	2.5
C5	7	21	3.0

아래 표에서 A는 현재까지 확보된 이득, B는 앞으로 확보할 수 있는 최대 이득을 표시합니다. 0/1 Knapsack으로 확보할 수 있는 최대 이득은 Partial Knapsack보다 항상 작으므로 이 값을 우리는 잘라 넣을 수 있는 Knapsack, 즉 partial Knapsack 문제로 해결해야 합니다. 아래 그림에서 B에 해당되는 부분을 모두 계산해서 채워 넣어 봅시다.



일단 물건 C_1을 넣었을 때의 경우를 생각해 봅시다. 이제 남아있는 물건은 { 2, 3, 4, 5 }이며 남아있는 공간은 45－27 = 18입니다. 이제 남은 18 용량에 그 다음 물건인 C_4를 최적으로 잘라 넣으면 됩니다. 이 잘라서 넣어 BIN을 채우는 방법은 greedy 방법이 최적(optimal)을 보장합니다. C_5을 넣으면 부피는 +7이고, 이득은

+21이 됩니다. 남은 공간은 18-7=11인데 그 다음 C_4를 다 넣지 못하고 가 그것의 11/16만 넣어야 하므로 추가되는 이득은 +40(11/16)입니다. 따라서 최대로 확보할 수 있는 아래쪽 경로로 진행할 때 우리가 확보할 수 있는 이득, 현찰(?)은 21+40(11/16) 이 됩니다.