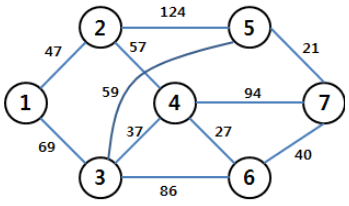
문제 7

연구활동 가는 길(S)

정올이는 GSHS에서 연구활동 교수님을 뵈러 A대학교를 가려고 한다. 출발점과 도착점을 포함하여 경유하는 지역 n개, 한 지역에서 다른 지역으로 가는 방법이 총 m개이며 GSHS는 지역 1이고 A대학교는 지역 n이라고 할 때 대학까지 최소 비용을 구하시오.

단, n은 10 이하, m은 30 이하, 그리고 한 지역에서 다른 지역으로 가는 데에 필요한 비용은 모두 200 이하 양의 정수이며 한 지역에서 다른 지역으로 가는 어떠한 방법이 존재하면 같은 방법과 비용을 통해 역방향으로 갈 수 있다.

다음 그래프는 예를 보여준다.(단, 정점a->정점b로의 간선이 여러 개 있을 수 있으며, 자기 자신으로 가는 정점을 가질 수도 있다.)



최소 비용이 드는 경로 : 1→3→5→7, 최소 비용 : 69+59+21=149

입력

첫 번째 줄에는 정점의 수 n과 간선의 수 m이 공백으로 구분되어 입력된다. 다음 줄부터 m개의 줄에 걸쳐서 두 정점의 번호와 가중치가 입력된다. (자기 간선, 멀티 간선이 있을 수 있다.)

출력

대학까지 가는 데 드는 최소 비용을 출력한다. 만약 갈 수 없다면 "-1"을 출력.

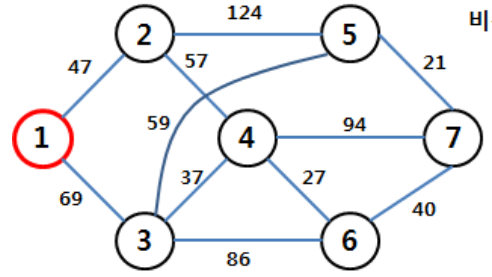
입력 예	출력 예
7 11 1 2 47 1 3 69 2 4 57 2 5 124 3 4 37 3 5 59 3 6 86 4 6 27 4 7 94 5 7 21 6 7 40	149

풀이

이 문제는 그래프 상의 최단경로를 구하는 매우 유명한 문제이다. 이 문제를 해결하는 알고리즘은 여러 가지가 알려져 있지만, 어려운 알고리즘을 모르더라도 전체탐색법을 통하여 해결할 수 있다.

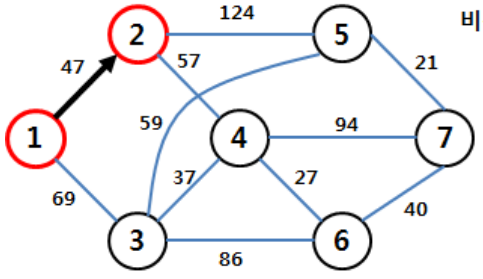
이 문제는 그래프 구조이므로 비선형탐색법으로 해를 구할 수 있다. 먼저 출발정점에서 깊이우선탐색을 이용하여 출발점으로부터 도착점까지 가능한 모든 경로에 대해서 구해본다. 하나의 경로를 구할 때마다 해를 갱신하면서 최종적으로 가장 적합한 해를 출력한다.

주어진 예를 통하여 전체탐색하는 과정을 간단하게 살펴보자.



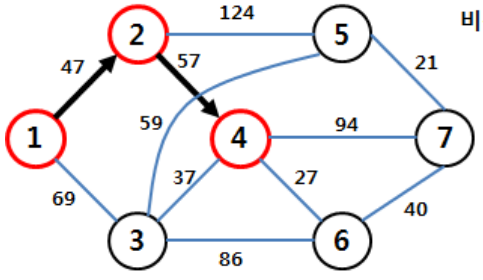
처음 출발점에서 2, 3의 정점 중 2번 정점으로 출발한다.

현재까지 구한 최소 이동거리 = ∞



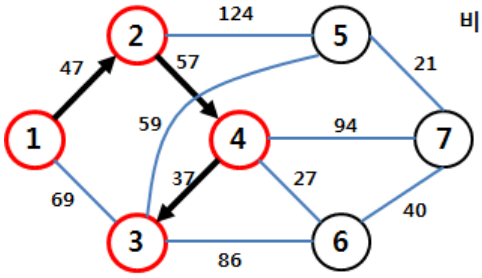
현재 2번 정점까지 이동거리 47, 다음으로 갈 수 있는 4, 5 중 4번을 먼저 탐색.

현재까지 구한 최소 이동거리 = ∞



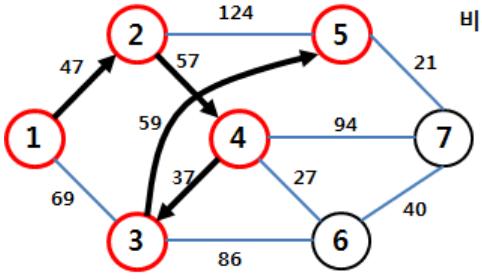
4번 정점까지 104를 이동한 후, 3, 6, 7중 먼저 3으로 먼저 탐색.

현재까지 구한 최소 이동거리 = ∞



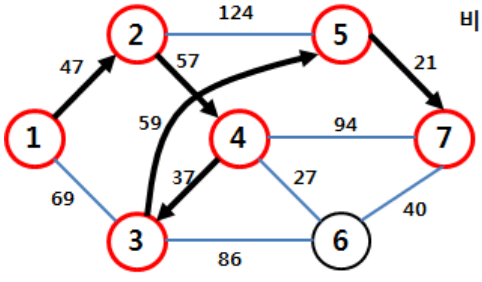
3으로 이동한 후, 후 다시 5번으로 이동.

현재까지 구한 최소 이동거리 = ∞



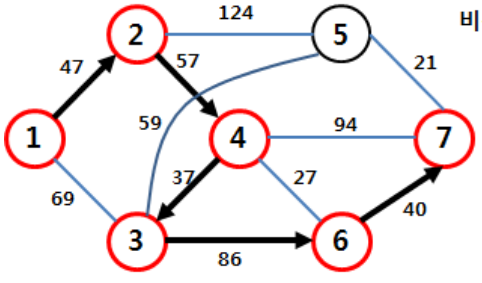
5번까지 이동 후, 7번으로 이동

현재까지 구한 최소 이동거리 = ∞



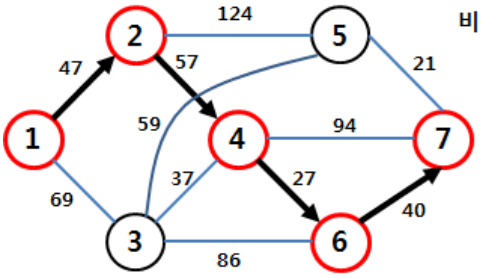
첫 번째 경로 찾음. 총 비용 221이므로 현재까지 구한 최소 이동거리는 221로 갱신

현재까지 구한 최소 이동거리 = 221



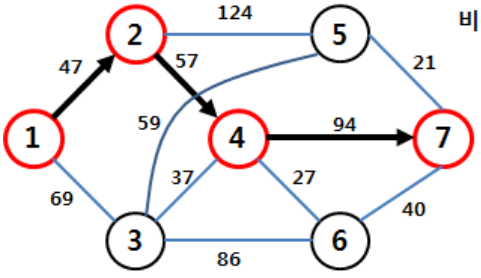
두 번째로 구한 경로는 267이 된다. 이 해는 지금까지의 해보다 더 크므로 갱신하지 않는다.

현재까지 구한 최소 이동거리 = 221



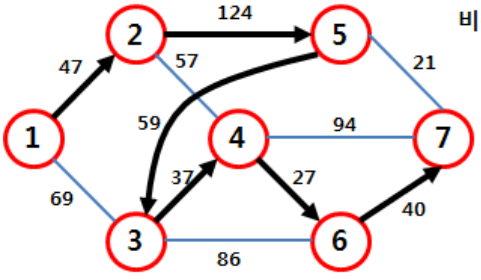
세 번째로 구한 해는 171이 된다.

현재까지 구한 최소 이동거리 = 171



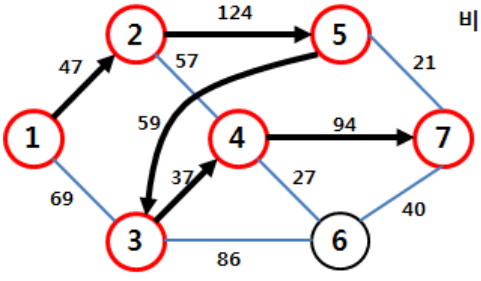
다음으로 구한 해는 198이 된다. 이 해는 지금까지 해보다 더 크므로 갱신하지 않는다.

현재까지 구한 최소 이동거리 = 171



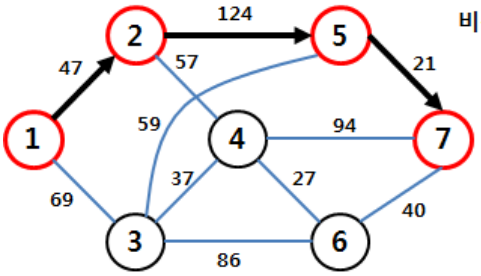
다음으로 구한 해는 302가 된다. 이 해는 지금까지 해보다 더 크므로 갱신하지 않는다.

현재까지 구한 최소 이동거리 = 171



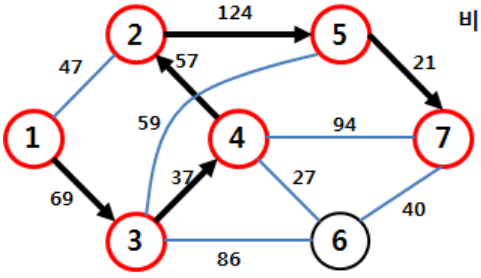
다음으로 구한 해는 329가 된다. 이 해는 지금까지 해보다 더 크므로 갱신하지 않는다.

현재까지 구한 최소 이동거리 = 171



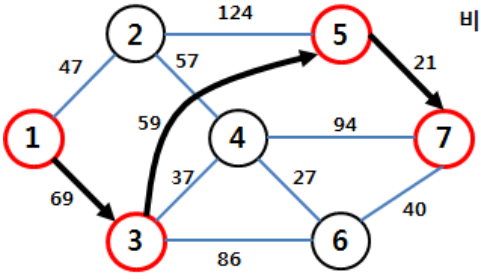
다음으로 구한 해는 192가 된다. 이 해는 지금까지 해보다 더 크므로 갱신하지 않는다.

현재까지 구한 최소 이동거리 = 171



다음으로 구한 해는 308이 된다. 이 해는 지금까지 해보다 더 크므로 갱신하지 않는다.

현재까지 구한 최소 이동거리 = 171



다음으로 구한 해는 149가 된다. 이 해는 지금까지 해보다 작으므로 갱신한다.

현재까지 구한 최소 이동거리 = 149

따라서 위의 경우 전체탐색법으로 탐색한 결과 최소 이동거리는 149가 됨을 알 수 있다. 위의 과정과 같은 방법으로 코딩한 결과는 다음과 같다.

줄	코드	참고
1	#include<stdio.h>	
2	int n, m, G[11][11], sol = 0x7fffffff, chk[11];	
3		
4	void solve(int V, int W)	
5	{	
6	if(V==n)	

줄	코드	참고

7	{
	  if(W<sol) sol=W;
	  return;
	}	

8		
9		
10		

11	for(int i=1; i<=n; i++)
	  if(!chk[i] && G[V][i])
	  {
	  chk[i]=1;
	  solve(i, W+G[V][i]);
	  chk[i]=0;
	  }
	}	

12		
13		
14		
15		
16		
17		
18		

19	int main(void)
	{
	  scanf("%d %d", &n, &m);
	  for(int i=0; i<m; i++)
	  {
	  int s, e, w;
	  scanf("%d %d %d", &s, &e, &w);
	  G[s][e]=G[e][s]=w;
	  }
	  solve(1, 0);
	  printf("%d\n", sol==0x7fffffff ? -1:sol);
	  return 0;
	}	

20		
21		
22		
23		
24		
25		
26		
27		
28		
29		
30		
31		

이 문제의 경우 정점과 간선의 수가 많지 않으므로 인접행렬로도 충분히 처리가 가능하기 때문에 인접행렬로 처리한다.

solve(a, b)는 현재 a정점까지 방문한 상태로 이동거리가 b라고 정의하고 있으며, chk배열이 현재까지 방문한 정점들의 정보를 가지고 있다. 다음 정점으로 진행할 때 14행과 같이 chk배열에 다음 방문할 정점을 체크하고 만약 백트랙해서 돌아온다면, 16행과 같이 체크를 해제하며 전체탐색을 진행한다.

6행에서 도착 여부를 확인하여 현재 정점이 도착점이라면, 지금까지의 이동 거리와 현재까지 구한 해를 비교하여 더 좋은 해가 있으면 해를 갱신한다. 이와 같이 작성할 경우 도시의 수가 n개라고 할 때 $O(n!)$의 계산이 필요하다.

문제 8

리모컨

컴퓨터실에서 수업 중인 정보 선생님은 냉난방기의 온도를 조절하려고 한다.

냉난방기가 멀리 있어서 리모컨으로 조작하려고 하는데, 리모컨의 온도 조절 버튼은 다음과 같다.

1) 온도를 1도 올리는 버튼
2) 온도를 1도 내리는 버튼
3) 온도를 5도 올리는 버튼
4) 온도를 5도 내리는 버튼
5) 온도를 10도 올리는 버튼
6) 온도를 10도 내리는 버튼

이와 같이 총 6개의 버튼으로 목표 온도를 조절해야 한다.

현재 설정 온도와 변경하고자 하는 목표 온도가 주어지면 이 버튼들을 이용하여 목표 온도로 변경하고자 한다.

이 때 버튼 누름의 최소 횟수를 구하시오. 예를 들어, 7도에서 34도로 변경하는 경우,

$$7 \rightarrow 17 \rightarrow 27 \rightarrow 32 \rightarrow 33 \rightarrow 34$$

이렇게 총 5번 누르면 된다.

입력

현재 온도 a와 목표 온도 b가 입력된다(0 <= a, b <= 40).

출력

최소한의 버튼 사용으로 목표 온도가 되는 버튼 누름의 횟수를 출력한다.

입력 예	출력 예
7 34	5

풀이

이 문제는 시작 온도에서 목표 온도로 되는 과정에서 버튼의 최소 이용 횟수를 구하는 문제이다. 먼저 전체탐색법으로 문제를 해결해 보자.

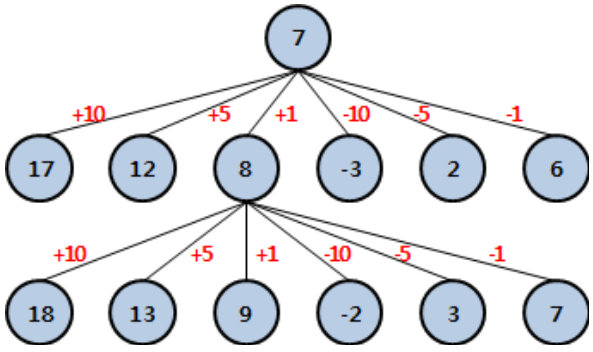
줄	코드	참고
1	`#include <stdio.h>`	
2		
3	`int a, b;`	
4	`int res=40;`	
5		
6	`void f(int temp, int cnt)`	
7	`{`	
8	`  if(cnt>res) return ;`	
9	`  if(temp==b)`	
10	`  {`	
11	`    if(cnt<res) res=cnt;`	
12	`    return;`	
13	`  }`	
14	`  f(temp+10,cnt+1);f(temp+5,cnt+1);f(temp+1,cnt+1);`	
15	`  f(temp-10,cnt+1);f(temp-5,cnt+1);f(temp-1,cnt+1);`	
16	`}`	
17		
18	`int main()`	
19	`{`	
20	`  scanf("%d%d", &a, &b);`	
21	`  f(a, 0);`	
22	`  printf("%d", res);`	
23	`  return 0;`	
24	`}`	

res의 초깃값을 40으로 설정한 이유는 입력의 정의역이 0~40이기 때문이며 최악의 경우 40번을 눌러야하기 때문에 최댓값으로 설정해 놓은 것이다. 이것은 다음에 백트래킹 함수 f의 호출 한계와도 관련이 있다.

백트래킹 함수의 의미는 다음과 같다.

f(temp, cnt) = 온도가 temp일 때, 버튼 누름 횟수는 cnt

각 온도에 대해서 다음으로 누르는 버튼은 +10도, +5도, +1도, −10도, −5도, −1도로 전체 탐색한다.



이렇게 탐색하여 목표 온도에 도달하면 탐색을 종료한다. 이 탐색 버튼의 수 6과 목표 온도에 도달하는 최악의 횟수 res에 비례하므로 계산량은 $O(6^{res})$이다. res를 40으로 설정하였기 때문에 계산량이 많아 제한 시간 안에 해결하기 어렵다. 따라서 계산량을 줄이기 위한 탐색영역을 배제할 필요가 있다.

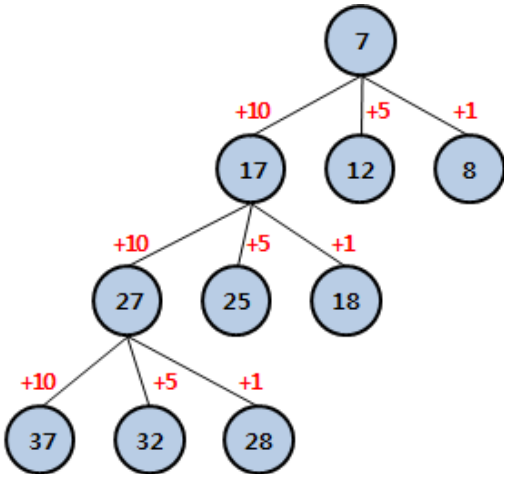
먼저 간단한 수학적인 상식을 적용시켜보면,

> 현재 온도가 목표 온도보다 작은 경우, 온도를 내릴 필요가 없다.
> 반대로, 현재 온도가 목표 온도보다 큰 경우, 온도를 올릴 필요가 없다.

이 원리를 이용하면 위 소스코드의 14~15행의 탐색범위를 줄여서 다음과 같이 표현할 수 있다.

줄	코드	참고
14	`if(temp<b){`	
15	`  f(temp+10,cnt+1);f(temp+5,cnt+1);f(temp+1,cnt+1);`	
16	`}`	
17	`else{`	
18	`  f(temp-10,cnt+1);f(temp-5,cnt+1);f(temp-1,cnt+1);`	
19	`}`	

위 소스를 토대로 실행하면 다음 호출 구조로 줄일 수 있다.



앞의 소스에 비해 계산량을 정확히 반으로 줄인 $O(3^{res})$ 결과를 볼 수 있다. 하지만 여전히 res의 최악의 횟수가 계산량에 영향을 많이 미치기 때문에, res를 줄이는 수학적 검증이 필요하다.

또 최악의 횟수를 줄일 수 있지만 현재 온도와 목표 온도의 차이가 아주 많이 나는 경우에는 여전히 계산량이 많아진다. 따라서 res를 조절하는 것 보다 계산 과정 중 중복된 연산을 줄이고, 목표 온도에 도달한 경우 더 이상 함수를 호출하지 않는 방법을 선택하는 것이 좋다.

백트래킹은 모든 호출이 끝이 나야만 해답을 찾을 수 있기 때문에, 이러한 깊이우선 탐색은 이 문제에서 비효율적이다. 이 방법 대신 너비우선탐색 기법을 이용하여 버튼 누름 횟수를 기준으로 가장 먼저 목표 온도에 도달할 때까지 탐색하고 중단하는 방법으로 설계해 보자.

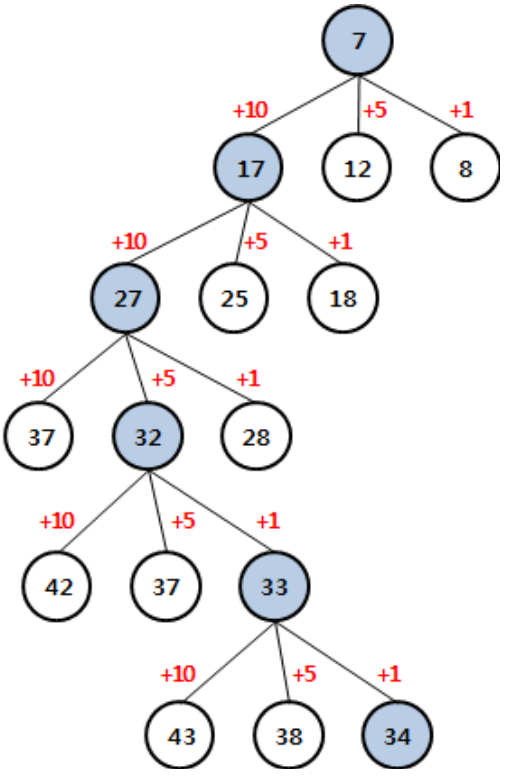
줄	코드	참고
1	`#include <stdio.h>`	
2	`#include <queue>`	
3	`using namespace std;`	
4		
5	`struct ELE{int v, cnt;};`	
6	`queue<ELE> Q;`	
7		

줄	코드	참고
8	int main()	
9	{	
10	int a, b, i;	
11	ELE temp;	
12	scanf("%d %d",&a,&b);	
13	Q.push({a, 0});	
14	while(!Q.empty())	
15	{	
16	temp = Q.front(), Q.pop();	
17	if(temp.v == b)	
18	break;	
19	if(temp.v<b)	
20	{	
21	Q.push({temp.v+10, temp.cnt+1});	
22	Q.push({temp.v+5, temp.cnt+1});	
23	Q.push({temp.v+1, temp.cnt+1});	
24	}	
25	else	
26	{	
27	Q.push({temp.v-10, temp.cnt+1});	
28	Q.push({temp.v-5, temp.cnt+1});	
29	Q.push({temp.v-1, temp.cnt+1});	
30	}	
31	}	
32	printf("%d", temp.cnt);	
33	}	

앞의 백트래킹 소스를 너비우선탐색으로 변형한 코드이다. 너비우선탐색 알고리즘은 목표 온도 b에 도달하면 곧 바로 다른 모든 탐색을 중지하고 결과를 얻을 수 있어, 불필요한 탐색을 막을 수 있고 계산량을 확실히 줄일 수 있다.

소스에 대해 잠깐 설명하면 2행에서 STL queue 라이브러리를 이용하고 있다. 5행에서 ELE라는 구조체 형식의 Q를 선언하고, 13~31행에서 너비우선탐색 알고리즘을 사용하고 있다. 13행, 21~23행, 27~29행에서 구조체를 큐에 넣을 때 중괄호{, }를 이용하여 한 번에 대입할 수 있음을 참고하기 바란다.

7에서 34도로 변하는 과정을 트리로 표현한 결과이다.



34도에 도달하게 되면 모든 연산을 중지하고 결과를 얻을 수 있다. 너비우선탐색으로 탐색하여 목표 온도에 도달하는 경우 계산량은 이 탐색 버튼의 수 3과 목표 온도에 도달하는 최적의 횟수 cnt에 비례하므로 계산량을 줄일 수 있다.

문제 9

오른편 절단 가능 소수

수학자들에게 소수란 매우 흥미 있는 연구 주제이다. 소수(prime number)란 약수가 1과 자기 자신밖에 없는 1보다 큰 자연수를 말한다. 수학자들은 소수를 연구하면서 특이한 소수들을 발견하여 이름을 명명하였다. 메르센 소수, 페르마 소수, 쌍둥이 소수 등이 그 예이다.

우리에게는 생소하지만 오른편 절단 가능 소수가 있다. 이 소수는 오른쪽부터 하나씩 제거해도 계속 소수가 되는 소수이다.

크기가 네 자리인 7193을 예로 들어보자. 7193은 소수이고, 7193의 오른편 숫자 3을 제거하여 남은 719도 소수이다. 719의 오른편 숫자 9를 제거하여 남은 71도 소수이다. 71의 오른편 숫자 1을 제거하여 남은 7도 소수이다. 따라서 7193은 오른편 절단 가능 소수이다.

입력

자릿수 n이 정수로 입력된다.(1 <= n <= 10)

출력

1. n자리로 이루어진 오른편 절단 가능 소수들을 한 줄에 하나씩 오름차순으로 출력한다.
2. 마지막 줄에 출력된 오른편 절단 가능 소수들의 개수를 출력한다.

입력 예	출력 예
2	23 29 31 37 53 59 71 73 79 9

풀이

이 문제는 n자리의 숫자들 중 오른편 절단 가능 소수를 찾고 그 개수를 출력하는 문제이다. 먼저 길이가 n인 순열을 생성하고, 소수인지 판별해보는 전체탐색법으로 문제를 해결해 보자.

줄	코드	참고
1	#include<stdio.h>	
2		
3	int n, cnt;	
4	int isprime(int x)	
5	{	
6	if(x<2) return 0;	
7	for(int i=2; i*i<=x; i++)	
8	if(x%i==0)	
9	return 0;	
10	return 1;	
11	}	
12		
13	void f(int num, int len)	
14	{	
15	if(len==n)	
16	{	
17	if(num==0) return ;	
18		
19	int flag=1;	
20	int temp=num;	
21	while(temp)	
22	{	
23	if(!isprime(temp))	
24	return ;	
25	temp /= 10;	
26	}	
27	cnt++;	
28	printf("%d\n", num);	
29	return ;	
30	}	
31	else	
32	{	
33	for(int i=1; i<=9; i++)	

줄	코드	참고
34	f(num*10+i, len+1);	
35	}	
36	}	
37		
38	int main()	
39	{	
40	scanf("%d", &n);	
41	f(0, 0);	
42	printf("%d", cnt);	
43	}	

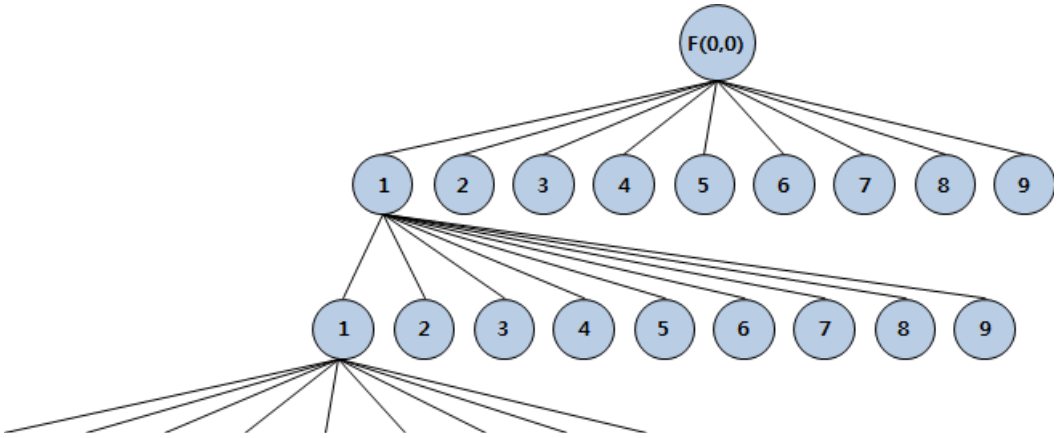
4~11행의 isprime(x)함수는 정수 x가 소수이면 1을 리턴, 소수가 아니면 0을 리턴하는 함수이다.

33~34행에서 1~9를 뒤에 오는 숫자로 추가하여 자릿수를 늘려간다. 여기서 0은 미리 제외 시켰다. 이 방법은 수학적 탐색영역의 배제의 한 방법으로 0으로 끝나는 수는 짝수이므로 소수가 될 수 없으므로 미리 제외시켰다.

백트래킹 함수의 의미는 다음과 같다.

f(num, len) = 현재 숫자 num과 그 길이 len을 의미

메인 함수에서 f(0, 0)으로 호출하면, 다음 구조로 호출이 이루어진다.



이런 구조로 길이가 n인 순열이 생성되고, 생성된 수를 4~11행의 소수 판별함수로 오른쪽부터 하나씩 절단하면서 소수인지 판단한다. 이 때 n/10으로 수를 분리하면 된다. 만약 오른쪽을 절단하면서 체크하는 과정에서 하나라도 소수가 아니면 바로 취소하고, 다음 숫자로 넘어간다. 만약 오른편 절단 가능 소수임이 판단되면 전체 개수를 저장하는 변수 cnt 값을 1증가시키고, 그 수를 화면에 바로 출력한다.

이 함수의 계산량은 순열을 생성하는 부분에 $O(9^n)$이 되고, 생성된 길이가 n인 각 수 x에 대해 $O(n\sqrt{x})$의 시간이 걸리므로, 전체 계산량은 $O(9^n \cdot n\sqrt{x})$이다. n이 6이상만 되어도 속도가 점점 느려짐을 알 수 있다.

이보다 더 탐색영역을 배제할 수 있는 부분을 생각해보자. 앞에서 0을 배제시켰듯이 수학적 배제 방법을 생각해보자.

1. 한 자릿수 중 소수는 2, 3, 5, 7 밖에 없다. 따라서 제일 높은 자릿수의 값은 2, 3, 5, 7만 될 수 있다. (∵오른편 절단 가능 소수이므로)
2. 두 자릿수 이상 넘어가면서 마지막 자릿수 값은 짝수가 될 수 없고(∵2의 배수), 마찬가지로 5도 될 수 없다(∵5의 배수). 따라서 남는 숫자는 1, 3, 7, 9만 남게 된다. 10개의 자릿수를 모두 다 탐색하는 것에 비해 가짓수가 현저히 줄어들게 됨으로 탐색이 빨라진다. (가지치기)
3. 자릿값을 늘려갈 때 현재 숫자가 소수인지 판별하여 소수인 숫자에만 뒤에 1, 3, 7, 9를 붙여가며 늘려간다. 숫자가 커지면 소수 판별에도 시간이 많이 걸리므로 시간을 줄이기 위해 $O(\sqrt{n})$ 소수 판별 알고리즘을 사용한다.

이 배제 방법을 토대로 소스를 개선시켜보자.

줄	코드	참고
1	`#include<stdio.h>`	26: 두 번째 자릿수부터는 1, 3, 7, 9 39: 시작 수는 2, 3, 5, 7
2		
3	`int n, cnt;`	
4		
5	`int isprime(int x)`	
6	`{`	
7	`  for(int i=2; i*i<=x; i++)`	
8	`    if(x%i==0)`	
9	`      return 0;`	
10	`    return 1;`	
11	`}`	
12		

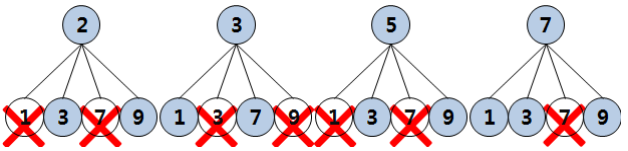
줄	코드	참고
13	void f(int num, int len)	
14	{	
15	if(len==n)	
16	{	
17	if(isprime(num))	
18	{	
19	cnt++;	
20	printf("%d\n", num);	
21	}	
22	return ;	
23	}	
24	else	
25	{	
26	if(isprime(num))	
27	{	
28	f(num*10+1, len+1);	
29	f(num*10+3, len+1);	
30	f(num*10+7, len+1);	
31	f(num*10+9, len+1);	
32	}	
33	}	
34	}	
35		
36	int main()	
37	{	
38	scanf("%d", &n);	
39	f(2,1); f(3,1); f(5,1); f(7,1);	
40	printf("%d", cnt);	
41	}	

이 배제 방법을 토대로 n이 1, 2, 3일 때 결과를 살펴보자.

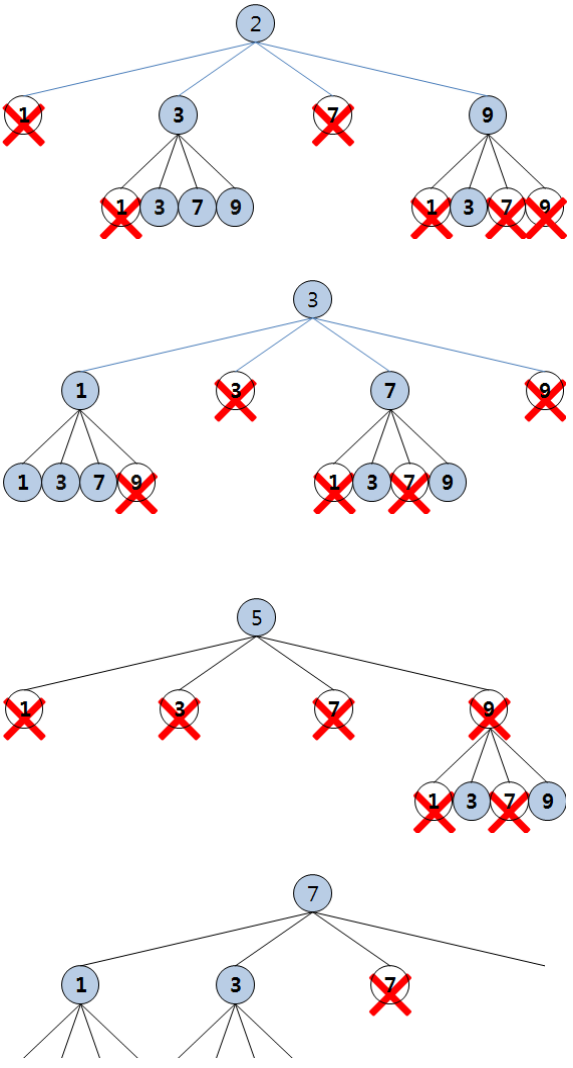
〈 N의 크기에 따른 상태 공간 〉

n = 1일 때,

n = 2일 때,



n = 3일 때,



이 소스의 계산량은 $O(4^n\sqrt{x})$ 이며 가지치기 전략에 의해 실질적으로는 수행시간을 더욱 단축할 수 있다.

문제 10

minimum sum(S)

n*n개의 수가 주어진다. (1<=n<=10)

이때 겹치지 않는 각 열과 각 행에서 수를 하나씩 뽑는다.
(즉, 총 n개의 수를 뽑을 것이다, 그리고 각 수는 100 이하의 값이다.)

이 n개의 수의 합을 구할 때 최솟값을 구하시오.

입력

첫 줄에 n이 입력된다. 다음 줄부터 n+1줄까지 n개씩의 정수가 입력된다.

출력

구한 최소 합을 출력한다.

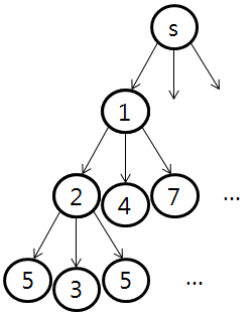
입력 예	출력 예
3 1 5 3 2 4 7 5 3 5	7

풀이

먼저 주어진 입력예제에 대해서 전체탐색법으로 접근하는 방법에 대해서 알아보자.

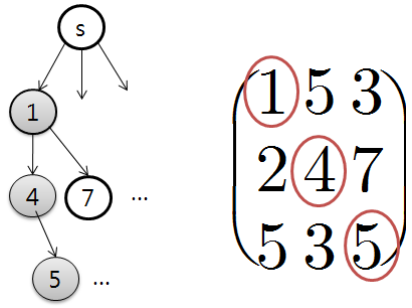
$$\begin{pmatrix} 1 & 5 & 3 \\ 2 & 4 & 7 \\ 5 & 3 & 5 \end{pmatrix}$$

행과 열이 중복되지 않아야 하므로 일단 각 행에서 1개 이상의 값은 얻을 수 없다. 그리고 서로 같은 열도 중복되지 않아야 하므로 위 문제는 다음과 같이 구조화할 수 있다.



탐색 구조

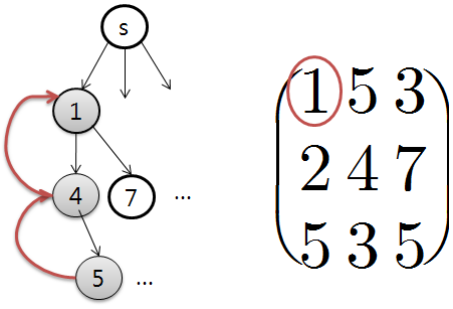
위 그림은 1행 1열의 1을 선택했을 때의 탐색 구조의 일부를 나타낸다. 1행에서 1열을 택했으므로 2행의 {2, 4, 7}중 1열의 {2}는 선택할 수 없다. 따라서 깊이우선으로 구할 수 있는 첫 번째 해는 아래 그림과 같다.



처음으로 구한 해

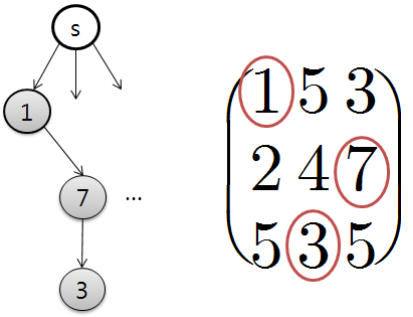
처음으로 구한 해는 위와 같이 1-4-5를 탐색하여 구한 10이다. 문제의 조건에 따라 1행 1열, 2행 2열을 선택했기 때문에 3행에서는 1열과 2열을 선택할 수 없다. 이 해 10은 최종적인 해인지 아닌지 현재 상태로는 확정할 수 없다. 따라서 백트래킹하여 가능한 모든 해를 구해야만 최종적으로 해를 구할 수 있다.

위의 상태에서 백트래킹 하면 다음과 같은 상태가 된다.



백트랙 후의 상태

위의 그림과 같은 상태에서는 7을 선택하여 계속해서 깊이우선탐색을 진행할 수 있다.



2번째로 구한 해 11

위의 그림에서 다시 11이라는 해를 구할 수 있다. 여기서 다시 백트랙하여 구할 수 있는 모든 해를 나열하면 다음과 같다.

$$(5, 2, 5) = 12,\ (5, 7, 5) = 17,\ (3, 2, 3) = 8,\ (3, 4, 5) = 12$$

따라서 이 문제에서 구할 수 있는 최소 점수는 3, 2, 3을 선택하여 얻을 수 있는 8점이 된다. 이 방법으로 작성한 소스코드는 다음과 같다.