과학계산 프로그래밍

Final Project

20040551 주영민

전체적 알고리즘 설명

이 프로그램은 패킷이 발생(IP주소 입력)하면 라우팅 테이블에서 검색을 한 후 출력 링크 번호를 받아내어 그 출력 링크 번호에 따라서 linked_list로 이루어진 같은 출력 링크 번호를 가진 큐를 찾아내어 그 큐에 저장 시키는 알고리즘이다.

전체적으로 Packet 클래스와 n_nary_tree 클래스, linked_list_Queue 클래스가 있는데 Packet 클래스는 IP주소를 저장하는 역할을 하고, n_nary_tree 클래스는 라우팅 테이블을 만드는 역할을 하고, linked_list_Queue 클래스는 발생된 패킷을 출력 링크번호별로 받아서 저장하고 출력하는 역할을 한다.

각 클래스와 클래스내의 함수의 알고리즘 설명

1. Packet

```
class Packet{
        friend class linked_list_Queue;
        private:
                char ip_address[20]; //IP주소
        public:
                Packet(void){strcpy(ip_address,"");} //기본 생성자
                void input_packet(char ip_address[]);
                //해당 IP주소를 가지는 패킷하나를 발생시키는 함수
};
void Packet::input_packet(char i_ip_address[])
{
        strcpy(ip_address,i_ip_address);
        cout<<"————————————————————————————————————"<<endl;
        cout<<ip_address<<" packet added"<<endl;
}
```

패킷은 위와 같이 구성.
패킷은 IP주소를 하나 받아 저장 시킨다.
저장 시킬 때 저장되었다고 출력한다.

2. n_nary_tree

```
class n_nary_tree{
 class node{
  public:
   char data[10]; //IP주소를 '.'으로 구분하여 저장시켜놓은 값
   int output_link; //출력 링크 번호(4번째 노드에만 출력링크번호를 저장)
  private:
   node *direc[11]; //tree의 10가지 방향 노드(direc[10]은 항상 NULL)
   node(void){
    for(int i=0; i<11; i++)
     direc[i]=NULL;
     strcpy(data,"");
     output_link=0;
   } //기본 생성자
  friend class n_nary_tree;
  };
 public:
  node *top[11]; //tree의 처음 시작 노드(top[10]은 항상 NULL)
  n_nary_tree(void){
   for(int i=0; i<11; i++)
   top[i]=NULL;
  } //기본 생성자
  void enter_one(char ip_address[], int output_link);
  //IP주소와 출력링크 번호를 입력받아 라우팅 태이블에 추가
  void enter(node *direc, char *adr, int output_link);
  //추가된 함수: enter_one을 사용할때 재귀함수를 이용 코드를 줄이는 함수
  //char *strchr(char *string_ptr, char find);
  //추가된 함수: 원하는 캐릭터의 위치를 찾아 포인터로 돌려줌
  int search_output_link(char ip_address[]);
  //특정 IP주소를 가진 패킷이 들어오면 라우팅 테이블을 찾아서 해당링크 번호를 리턴
  void delete_one(char ip_address[]);
  //입력받은 IP주소를 라우팅 테이블에서 지움
  void delete_all(void);
  //모든 라우팅 테이블의 entry를 지움
  void print(void);
  //라우팅 테이블에 입력된 IP주소와 해당 출력링크번호를 출력.
};
```

n_nary_tree의 함수의 알고리즘 설명

① void n_nary_tree::enter_one(char ip_address[], int output_link)

void n_nary_tree::enter(node *direc, char *adrs, int output_link)-추가함수

첫 번째가 주어진 멤버 함수이고 두 번째가 주어진 함수를 실행하기 위해 새로 추가시킨 재귀함수이다.

```
void n_nary_tree::enter_one(char ip_address[], int output_link)
{
        int tn; //top number
        int i;
        char tmp[10]; //임시로 IP주소를 받아둠

        for(i=0; ip_address[i]!='.'; i++)
        {       //처음 '.'이 나올때까지 IP주소를 임시변수에 저장
                tmp[i]=ip_address[i];
        }
        tmp[i]=NULL; //'.'을 NULL으로 변환

        for(tn=0; tn<10 && top[tn]!=NULL && strcmp(top[tn]->data,tmp)!=0; tn++);
                //같은 IP주소를 가진 top node를 찾음

        if(tn==10){ //10개가 넘는데도 같은것이 없으면 꽉 차 있으므로 더 추가시킬수 없다
                cout<<"————————————————————————————————————————"<<endl;
                cout<<"Tree's top node is full!!"<<endl;
                return;
        }

        if(top[tn]==NULL){ //같은 IP를 가진 top node가 없으면 새로 만든다
                top[tn] = new node;
                strcpy(top[tn]->data,tmp);
        }

        enter(top[tn], &ip_address[i+1], output_link);
}
```

위의 코드에서 처음부터 설명하면 일단 변수로 클래스의 시작노드의 방향을 가르키는 tn(top number)와 for문 사용에 필요한 i 그리고 함수가 동작할 동안 임시로 IP주소를 받아둘 tmp[10]을 선언한다. 그런 다음 IP주소의 처음‘.’이 나올때까지의 IP주소를 임시 변수 tmp에 저장시킨다. 그리고나서 for문이 종료되면 tmp의 마지막에 NULL을 추가시켜서 배열 tmp를 마무리 시킨다. 그런다음 그 배열값을 가지는 IP를 찾고 10개가 꽉 차 있을때 같은 것이 없으면 에러 메시지를 출력하고 10개가 차지 않았을때 업으면 새로운 top node를 생성한다. 그리고 마지막으로 재귀함수인 enter를 불러 top node에 이어지는 3개의 node를 생성시킨다.

```
void n_nary_tree::enter(node *direc, char *adrs, int output_link)
{
	int dn; //direction number
	int i;
	char tmp[10]; //임시로 IP주소를 받아둠

	if(adrs[0]==NULL)
	{	//주소를 다 입력 시키고 나서 마지막 node에 출력링크 번호를 입력 시킨다
		direc->output_link=output_link;
		return;
	}

	for(i=0; adrs[i]!='.' && adrs[i]!=NULL; i++)
	{	//'.'이 나오거나 NULL일 때까지 IP주소를 임시변수에 저장
		tmp[i]=adrs[i];
	}
	tmp[i]=NULL; //'.'을 NULL으로 변환

	for(dn=0; dn<10 && direc->direc[dn]!=NULL && strcmp(direc->direc[dn]->data,tmp)!=0; dn++);
		//같은 IP주소를 가진 node를 찾음

	if(dn==10){ //10개가 넘는데도 같은것이 없으면 꽉 차 있으므로 더 추가시킬수 없다
		cout<<"————————————————————————————————————"<<endl;
		cout<<"Tree's node is full!!"<<endl;
		return;
	}

	if(direc->direc[dn]==NULL){ //같은 IP를 가진 node가 없으면 새로 만든다
		direc->direc[dn] = new node;
		strcpy(direc->direc[dn]->data,tmp);
	}
	enter(direc->direc[dn], adrs+i+1, output_link);
	//마지막 IP주소까지 다 입력 시킬때 까지 자기자신을 호출
}
```

위에서 재귀함수를 호출 하였다. 이 함수는 노드의 포인터와 IP주소의 '.'이후의 값, 그리고 출력링크 번호를 받아서 실행된다. 여기서도 역시 임시로 주소를 받고 주소가 NULL값을 가르키면 출력링크번호를 그 노드에 저장시키고 함수를 종료 시킨다. 그렇지 않으면 임시변수에 '.'이나 NULL이 나오기 전까지의 IP주소를 저장 시키고 그 주소로 같은 IP주소를 갖는 것을 찾고 있으면 다시 enter로 다음 주소값과 다음 노드 값을 주고 enter를 부르고 아니면 node를 새로 만들고나서 부른다.

이런식으로 반복되고나면 IP주소값이 '.'을 구분해 4개로 잘라져서 저장되고 마지막 노드에 출력 링크번호가 저장되게 된다.

② char *strchr(char *string_ptr, char find) -추가함수

int n_nary_tree::search_output_link(char ip_address[])

첫 번째 함수는 수업시간에 배운 함수로 앞에 입력되는 배열에서 뒤에 입력되는 char를 찾아서 그 포인터 값을 돌려주는 함수이다(추가함수). 그리고 두 번째 함수가 주어진 함수로 앞의 함수를 이용하여 입력받은 IP주소를 4개로 잘라서 입력되어 있는 라우팅 테이블에서 같은 IP주소를 찾고 그 IP주소의 출력 링크번호를 리턴한다.

```
int n_nary_tree::search_output_link(char ip_address[])
{
        char tmp[20]; //임시로 IP주소를 받아둠
        char *tmp_ptr0; //첫번째 숫자열
        char *tmp_ptr1; //두번째 숫자열
        char *tmp_ptr2; //세번째 숫자열
        char *tmp_ptr3; //네번째 숫자열

        이부분이 IP주소를 4개로 잘라주는 부분이다.
        strcpy(tmp,ip_address);
        tmp_ptr0 = tmp;
        tmp_ptr1 = strchr(tmp,'.'); //ip_address에서 '.'을 찾음
        *tmp_ptr1 = NULL; //'.'을 null로 바꿈
        ++tmp_ptr1; //null 다음을 가르킴
        tmp_ptr2 = strchr(tmp_ptr1,'.'); //위와 같은 방법
        *tmp_ptr2 = NULL;
        ++tmp_ptr2;
        tmp_ptr3 = strchr(tmp_ptr2,'.'); //위와 같은 방법
        *tmp_ptr3 = NULL;
        ++tmp_ptr3;

        int k,ks; //top node 상수
        int j,js; //2번째 node 상수
        int l,ls; //3번째 node 상수
        int m,ms; //4번째 node 상수

        이부분은 차례대로 첫 번째 노드부터 같은 것이 있는지 확인한다.
        없으면 에러 메시지를 출력하고 0값을 리턴하고 종료시킨다.
        for(ks=0; ks<10 && top[ks]!=NULL; ks++); //NULL이 아닐때까지의 탑노드의 갯수를 찾음
        for(k=0; k<ks && strcmp(top[k]->data,tmp_ptr0)!=0; k++); //입력된 ip와 같을때까지 비교
        if(k==ks)
        {       //같은 ip가 존재하지 않음
                cout<<"-----------------------------------------"<<endl;
                cout<<"I can't search IP_address"<<endl;
                cout<<"IP : "<<ip_address<<" does not exist!"<<endl;
                return (0);
        }
        for(js=0; js<10 && top[k]->direc[js]!=NULL; js++); //NULL이 아닐때까지의 노드의 갯수를 찾음
        for(j=0; j<js && strcmp(top[k]->direc[j]->data,tmp_ptr1)!=0; j++); //입력된 ip와 같을때까지 비교
        if(j==js)
        {       //같은 ip가 존재하지 않음
                cout<<"-----------------------------------------"<<endl;
                cout<<"I can't search IP_address"<<endl;
                cout<<"IP : "<<ip_address<<" does not exist!"<<endl;
                return (0);
        }
        for(ls=0; ls<10 && top[k]->direc[j]->direc[ls]!=NULL; ls++); //NULL이 아닐때까지의 노드의 갯수를 찾음
        for(l=0; l<ls && strcmp(top[k]->direc[j]->direc[l]->data,tmp_ptr2)!=0; l++); //입력된 ip와 같을때까지 비교
        if(l==ls)
```

```
{           //같은 ip가 존재하지 않음
            cout<<"————————————————————————————————————————"<<endl;
            cout<<"I can't search IP_address"<<endl;
            cout<<"IP : "<<ip_address<<" does not exist!"<<endl;
            return (0);
}
for(ms=0; ms<10 && top[k]->direc[j]->direc[l]->direc[ms]!=NULL; ms++);
//NULL이 아닐때까지의 노드의 갯수를 찾음
for(m=0; m<ms && strcmp(top[k]->direc[j]->direc[l]->direc[m]->data,tmp_ptr3)!=0; m++);
//입력된 ip와 같을때까지 비교
if(m==ms)
{           //같은 ip가 존재하지 않음
            cout<<"————————————————————————————————————————"<<endl;
            cout<<"I can't search IP_address"<<endl;
            cout<<"IP : "<<ip_address<<" does not exist!"<<endl;
            return (0);
}
```

무사히 같은 IP주소를 찾으면 그 IP주소를 찾았다고 출력하고 그 IP의 출력링크값을 리턴해줌

```
cout<<"————————————————————————————————————————"<<endl;
cout<<"We found IP Address : "<<ip_address<<endl;
return(top[k]->direc[j]->direc[l]->direc[m]->output_link); //찾은 출력링크 번호를 리턴해줌
}
```

③ void n_nary_tree::delete_one(char ip_address[])

처음부분은 search와 마찬가지 방식으로 같은 IP주소가 있는지 찾는다. 없으면 에러 메시지를 출력하고 있으면 일단 마지막 노드를 지우고 3번째 노드로 올라가서 3번째 노드가 비어 있으면 지우고 아니면 끝낸다. 2번째, 1번째 노드도 마찬가지 방식으로 한다.

```
void n_nary_tree::delete_one(char ip_address[])
{		//search_output_link방식으로 같은 주소를 찾은 다음 지운다

		char tmp[20]; //임시로 IP주소를 받아둠
		char *tmp_ptr0; //첫번째 숫자열
		char *tmp_ptr1; //두번째 숫자열
		char *tmp_ptr2; //세번째 숫자열
		char *tmp_ptr3; //네번째 숫자열

		strcpy(tmp,ip_address);
		tmp_ptr0 = tmp;
		tmp_ptr1 = strchr(tmp,'.'); //ip_address에서 '.'을 찾음
		*tmp_ptr1 = NULL; //'.'을 null로 바꿈
		++tmp_ptr1; //null 다음을 가르킴
		tmp_ptr2 = strchr(tmp_ptr1,'.'); //위와 같은 방법
		*tmp_ptr2 = NULL;
		++tmp_ptr2;
		tmp_ptr3 = strchr(tmp_ptr2,'.'); //위와 같은 방법
		*tmp_ptr3 = NULL;
		++tmp_ptr3;

		int k,ks; //top node 상수
		int j,js; //2번째 node 상수
		int l,ls; //3번째 node 상수
		int m,ms; //4번째 node 상수

		for(ks=0; ks<10 && top[ks]!=NULL; ks++); //NULL이 아닐때까지의 탑노드의 갯수를 찾음
		for(k=0; k<ks && strcmp(top[k]->data,tmp_ptr0)!=0; k++); //입력된 ip와 같을때까지 비교
		if(k==ks)
		{		//같은 ip가 존재하지 않음
				cout<<"————————————————————————————————————————"<<endl;
				cout<<"I can't delete IP_address"<<endl;
				cout<<"IP : "<<ip_address<<" does not exist!"<<endl;
				return;
		}
		for(js=0; js<10 && top[k]->direc[js]!=NULL; js++); //NULL이 아닐때까지의 노드의 갯수를 찾음
		for(j=0; j<js && strcmp(top[k]->direc[j]->data,tmp_ptr1)!=0; j++); //입력된 ip와 같을때까지 비교
		if(j==js)
		{		//같은 ip가 존재하지 않음
				cout<<"————————————————————————————————————————"<<endl;
				cout<<"I can't delete IP_address"<<endl;
				cout<<"IP : "<<ip_address<<" does not exist!"<<endl;
				return;
		}
		for(ls=0; ls<10 && top[k]->direc[j]->direc[ls]!=NULL; ls++); //NULL이 아닐때까지의 노드의 갯수를 찾음
		for(l=0; l<ls && strcmp(top[k]->direc[j]->direc[l]->data,tmp_ptr2)!=0; l++); //입력된 ip와 같을때까지 비교
		if(l==ls)
		{		//같은 ip가 존재하지 않음
				cout<<"————————————————————————————————————————"<<endl;
				cout<<"I can't delete IP_address"<<endl;
				cout<<"IP : "<<ip_address<<" does not exist!"<<endl;
				return;
		}
```

```
for(ms=0; ms<10 && top[k]->direc[j]->direc[l]->direc[ms]!=NULL; ms++);
//NULL이 아닐때까지의 노드의 갯수를 찾음
for(m=0; m<ms && strcmp(top[k]->direc[j]->direc[l]->direc[m]->data,tmp_ptr3)!=0; m++);
//입력된 ip와 같을때까지 비교
if(m==ms)
{		//같은 ip가 존재하지 않음
		cout<<"————————————————————————————————————"<<endl;
		cout<<"I can't delete IP_address"<<endl;
		cout<<"IP : "<<ip_address<<" does not exist!"<<endl;
		return;
}
```

같은 IP가 존재할 경우 마지막 노드부터 지워 나간다.

```
if(k!=ks && j!=js && l!=ls && m!=ms)
{
		int i;
		//top[k]->direc[j]->direc[l]->direc[m]가 입력된 IP의 라우팅 테이블 주소
		delete top[k]->direc[j]->direc[l]->direc[m];
		//4번째 노드를 지움
		for(; m<ms; m++)
		{		//지워진 노드뒤에 있는 노드들을 앞으로 당겨줌
				top[k]->direc[j]->direc[l]->direc[m] = top[k]->direc[j]->direc[l]->direc[m+1];
		}
		delete top[k]->direc[j]->direc[l]->direc[ms];
		top[k]->direc[j]->direc[l]->direc[ms]=NULL;
		//앞으로 당겨지고 마지막 남은 노드를 지움

		for(i=0; i<10 && top[k]->direc[j]->direc[l]->direc[i]==NULL; i++);
		//3번째 노드가 비었는지 확인
		if(i==10)
		{		//비었을때
				delete top[k]->direc[j]->direc[l];
				//3번째 노드를 지움
				for(; l<ls; l++)
				{		//지워진 노드뒤에 있는 노드들을 앞으로 당겨줌
						top[k]->direc[j]->direc[l] = top[k]->direc[j]->direc[l+1];
				}
				delete top[k]->direc[j]->direc[ls];
				top[k]->direc[j]->direc[ls]=NULL;
				//앞으로 당겨지고 마지막 남은 노드를 지움

				for(i=0; i<10 && top[k]->direc[j]->direc[i]==NULL; i++);
				//2번째 노드가 비었는지 확인
				if(i==10)
				{		//비었을때
						delete top[k]->direc[j];
						//2번째 노드를 지움
						for(; j<js; j++)
						{		//지워진 노드뒤에 있는 노드들을 앞으로 당겨줌
								top[k]->direc[j] = top[k]->direc[j+1];
						}
						delete top[k]->direc[js];
						top[k]->direc[js]=NULL;
						//앞으로 당겨지고 마지막 남은 노드를 지움

						for(i=0; i<10 && top[k]->direc[i]==NULL; i++);
						//첫번째 노드가 비었는지 확인
						if(i==10)
```

```
                                        {           //비었을때
                                                    delete top[k];
                                                    for(; k<ks; k++)
                                                    {           //지워진 노드뒤에 있는 노드들을 앞으로 당겨줌
                                                                top[k]= top[k+1];
                                                    }
                                                    delete top[ks];
                                                    top[ks]=NULL;
                                                    //앞으로 당겨지고 마지막 남은 노드를 지움
                                        }
                                }
                        }           //더이상 비는게 없으면 거기까지만 지우고 끝냄
```

지우고 나서 지워졌다고 출력함

```
cout<<"————————————————————————————————————"<<endl;
cout<<"Delete IP Address : "<<ip_address<<endl;
}

}
```

④ void n_nary_tree::delete_all(void)

라우팅 테이블을 전부 지우는 함수로 모든 라우팅 테이블을 지운다고 출력하고 각각 NULL이 아닐때까지의 노드를 찾고 그 전까지의 노드를 4번째 노드부터 차례대로 다 지워 나간다. 그리고 다 지운다음에는 top node는 NULL로 초기화를 다시 시켜준다.

```
void n_nary_tree::delete_all(void)
{
            int tn_0; //top(1번째) node의 갯수
            int tn_1; //for문을 돌릴때 필요한 상수
            int nn1_0; //2번째 node의 갯수
            int nn1_1; //for문을 돌릴때 필요한 상수
            int nn2_0; //3번째 node의 갯수
            int nn2_1; //for문을 돌릴때 필요한 상수
            int nn3_0; //4번째 node의 갯수
            int nn3_1; //for문을 돌릴때 필요한 상수

            cout<<" ┌──────────────────────────────────────┐ "<<endl;
            cout<<" |          Delete All Routing table          | "<<endl;
            cout<<" └──────────────────────────────────────┘ "<<endl;

for(tn_0 =0; tn_0<10 && top[tn_0]!=NULL; tn_0++);
//NULL이 아닐때까지의 탑노드의 갯수를 찾음
for(tn_1 =0; tn_1<tn_0; tn_1++)
{
 for(nn1_0 =0; nn1_0<10 && top[tn_1]->direc[nn1_0]!=NULL; nn1_0++);
 //NULL이 아닐때까지의 노드의 갯수를 찾음
   for(nn1_1 =0; nn1_1<nn1_0; nn1_1++)
   {
     for(nn2_0 =0; nn2_0<10 && top[tn_1]->direc[nn1_1]->direc[nn2_0]!=NULL; nn2_0++);
     //NULL이 아닐때까지의 노드의 갯수를 찾음
           for(nn2_1 =0; nn2_1<nn2_0; nn2_1++)
           {
                for(nn3_0 =0; nn3_0<10 && top[tn_1]->direc[nn1_1]->direc[nn2_1]->direc[nn3_0]!=NULL; nn3_0++);
                //NULL이 아닐때까지의 노드의 갯수를 찾음
                      for(nn3_1 =0; nn3_1<nn3_0; nn3_1++)
                      {
                                delete top[tn_1]->direc[nn1_1]->direc[nn2_1]->direc[nn3_1]; //4번째 노드를 전부 지움
                      }
                      delete top[tn_1]->direc[nn1_1]->direc[nn2_1]; //3번째 노드를 전부 지움
           }
           delete top[tn_1]->direc[nn1_1]; //2번째 노드를 전부 지움
   }
   delete top[tn_1]; //1번째 노드를 전부 지움
}

 for(int i=0; i<11; i++)
  {         //다 지웠으니 초기화
            top[i]=NULL;
  }
}
```

⑤ void n_nary_tree::print(void)

라우팅 테이블에 들어 있는 모든 IP주소와 그 IP주소의 출력링크 값을 출력해주는 함수. top node의 첫 번째가 NULL인지 검사하고 NULL이면 모든 라우팅 테이블이 비어 있는것이므로 비어 있다고 출력하고 아니면 라우팅 테이블 전체를 주소와 출력링크 값을 정렬하여 출력하여 준다.

```
void n_nary_tree::print(void)
{
            int tn_0; //top(1번째) node의 갯수
            int tn_1; //for문을 돌릴때 필요한 상수
            int nn1_0; //2번째 node의 갯수
            int nn1_1; //for문을 돌릴때 필요한 상수
            int nn2_0; //3번째 node의 갯수
            int nn2_1; //for문을 돌릴때 필요한 상수
            int nn3_0; //4번째 node의 갯수
            int nn3_1; //for문을 돌릴때 필요한 상수

            cout<<" ┌──────────────────────────────────┐ "<<endl;
            cout<<" |              Routing   table              | "<<endl;
            cout<<" └──────────────────────────────────┘ "<<endl;

if(top[0]!=NULL)
{
 for(tn_0 =0; tn_0<10 && top[tn_0]!=NULL; tn_0++);
 //NULL이 아닐때까지의 탑노드의 갯수를 찾음
 for(tn_1 =0; tn_1<tn_0; tn_1++)
 {
   for(nn1_0 =0; nn1_0<10 && top[tn_1]->direc[nn1_0]!=NULL; nn1_0++);
   //NULL이 아닐때까지의 노드의 갯수를 찾음
   for(nn1_1 =0; nn1_1<nn1_0; nn1_1++)
   {
            for(nn2_0 =0; nn2_0<10 && top[tn_1]->direc[nn1_1]->direc[nn2_0]!=NULL; nn2_0++);
            //NULL이 아닐때까지의 노드의 갯수를 찾음
            for(nn2_1 =0; nn2_1<nn2_0; nn2_1++)
            {
              for(nn3_0 =0; nn3_0<10 && top[tn_1]->direc[nn1_1]->direc[nn2_1]->direc[nn3_0]!=NULL; nn3_0++);
             //NULL이 아닐때까지의 노드의 갯수를 찾음
             for(nn3_1 =0; nn3_1<nn3_0; nn3_1++)
             { //정렬되게 프린트 하기위해 간격을 일정하게 프린터 한다
               cout<<setw(3)<<top[tn_1]->data<<'.'
                   <<setw(3)<<top[tn_1]->direc[nn1_1]->data<<'.'
                   <<setw(3)<<top[tn_1]->direc[nn1_1]->direc[nn2_1]->data<<'.'
                   <<setw(3)<<top[tn_1]->direc[nn1_1]->direc[nn2_1]->direc[nn3_1]->data<<" output_link : "
                   <<setw(3)<<top[tn_1]->direc[nn1_1]->direc[nn2_1]->direc[nn3_1]->output_link<<endl;
             }
            }
   }
 }
}
            else  //table이 비었을때 비었다고 출력
            cout<<"          Routing table is empty"<<endl;
}
```

3. linked_list_Queue

```
class linked_list_Queue{
  class queue{
    public:
      queue(void){
          output_link=0;
          next=NULL;
          data=NULL;
          size=0;
      } //기본 생성자
    private:
          queue *next; //다음 큐를 가르킴
          Packet *data; //queue가 받는 패킷
          int output_link; //출력 링크 번호(모든 큐마다 가지고 있음)
          int size; //큐에 연결된 패킷의 갯수 : 100개로 제한 걸기 위해 사용(first queue만 가지고 있음)
    friend class linked_list_Queue;
  };
  public:
    queue *first[100]; //시작 queue의 포인터, 100개의 출력 링크번호를 가진 queue저장가능
    n_nary_tree *r_table; //라우팅 테이블
    linked_list_Queue(void){
          for(int i=0; i<100; i++)
          first[i]=NULL;
    } //기본 생성자
    linked_list_Queue(n_nary_tree *tree){
          for(int i=0; i<100; i++)
          first[i]=NULL;
          r_table=tree;
    } //파라미터가 있는 생성자
    void push_queue(Packet *p);
    //패킷을 매개변수로 받아 해당패킷의 IP주소를 따라
    //출력 링크 번호 i번째 queue에 패킷을 삽입하는 역할을 하는 함수
    void find(queue *finder,int size,int fn,Packet *p);
    //재귀적 호출을 하여 같은 output num을 가지는 queue의 마지막에 도달하게 하는 함수(추가함수)
    void pop_up_queue(int output_link);
    //queue에 가장 먼저 들어온 패킷을 pop up하는 함수
    void output_queue(int output_link);
    //해당 출력 링크의 queue에 저장되어 있는 패킷의 수를 출력하는 함수
    void print_queue(int output_link);
    //해당 출력 링크의 queue에 들어있는 모든 패킷의 IP주소를 출력하는 역할을하는 함수
    void print(queue *printer,int size);
    //재귀적 호출을 하면서 같은 output num을 가지는 큐들을 전부 출력한다(추가함수)
    void print_all(void);
    //linked_list에 있는 모든 queue들을 출력함(추가함수)
    void delete_all(void);
    //linked_list에 있는 모든 queue들을 제거함(추가함수)
};
```

linked_list_Queue의 함수의 알고리즘 설명

① void linked_list_Queue::push_queue(Packet *p)

void linked_list_Queue::find(queue *finder,int size,int fn,Packet *p)-추가함수

첫 번째 함수는 주어진 함수이고 두 번째 함수는 이 함수를 실행시키기 위해 구현한 재귀적 함수이다.

```
void linked_list_Queue::push_queue(Packet *p)
{
        char *tmp;
        tmp=p->ip_address;
        int output=r_table->search_output_link(tmp); //들어온 패킷의 output번호를 찾는다
        if(output==0){ //들어온 패킷이 라우팅 테이블에 없는거면 리턴값이 0이므로 종료시킨다
          return;
        }
        int fn; //시작 queue의 포인터 number
        for(fn=0; fn<100 && first[fn]!=NULL && first[fn]->output_link!=output; fn++);
                //같은 출력링크번호를 가진 first queue를 찾음
        if(fn==100){ //100개가 넘는데도 같은것이 없으면 꽉 차 있으므로 더 추가시킬수 없다
                cout<<"————————————————————————————————————————"<<endl;
                cout<<"linked_list_Queue's first node is full!!"<<endl;
                return;
        }
        if(first[fn]==NULL){ //같은output num을 가진 first queue가 없으면 새로 만든다
                first[fn] = new queue;
                first[fn]->output_link=output;
                first[fn]->data=p;
                first[fn]->size++; //추가하고 queue의 크기를 1증가 시킴
                cout<<"queue is pushed"<<endl;
                return;
        }
        if(first[fn]->size==100){ //같은output num을 가지는 큐가 100개가 넘으면 중지
                cout<<"————————————————————————————————————————"<<endl;
                cout<<"linked_list_Queue's output num "<<output<<" is full!!"<<endl;
                return;
        }
        find(first[fn],first[fn]->size,fn,p);
        //재귀적 호출을 하여 같은 output num을 가지는 queue의 마지막에 도달하게 하는 함수(추가)
}
```

위에서 주석을 달아 놓은대로 내려오다가 마지막에 재귀함수인 find를 부른다.

```
void linked_list_Queue::find(queue *finder,int size,int fn,Packet *p)
{       //queue의 마지막을 찾는 함수(추가함수)
        if(size==1)
        {       //size가 1이면 마지막 queue이므로 새로운 queue를 만들어 들어온 packet을 저장시키고 종료한다.
                finder->next= new queue;
                finder->next->output_link=first[fn]->output_link;
                finder->next->data=p;
                first[fn]->size++; //추가하고 queue의 크기를 1증가 시킴
                cout<<"queue is pushed"<<endl;
                return;
        }
        else{
        int sz=size-1; //다음큐로 넘어가면서 count를 하나 줄인다(마지막 queue의 위치를 알기위해)
        find(finder->next,sz,fn,p);
        }
}
```

큐의 포인터, 현재 size, 첫 번째 queue가 몇 번째인가를 가르키는 fn, 마지막으로 패킷을 받아서 실행된다.

② void linked_list_Queue::pop_up_queue(int output_link)

주어진 함수로서 큐에 가장 먼저 입력된 것을 불러내어 출력하고 제거한다.
first queue를 임시로 tmp_q에 받아 두고 first queue의 next queue를 first queue로 지정하고 임시로 받아둔 tmp_q에서 first queue의 정보를 받아 출력한 뒤 first queue를 제거한다. 제거된 뒤 그 출력링크번호에 남아있는 queue가 없을 경우에는 뒤의 큐들을 앞으로 당기고 마지막에 당겨진 first queue를 제거하고 그 queue를 NULL로 초기화를 시켜준다.

```
void linked_list_Queue::pop_up_queue(int output_link)
{
        int fn; //시작 queue의 포인터 number

        for(fn=0; fn<100 && first[fn]!=NULL && first[fn]->output_link!=output_link; fn++);
                //같은 출력링크번호를 가진 first queue를 찾음

        if(fn==100||first[fn]==NULL){ //같은output num을 가지지 않으면 중지
                cout<<"————————————————————————————————————"<<endl;
                cout<<"linked_list_Queue has not output num : "<<output_link<<endl;
                return;
        }

        queue *tmp_q; //first queue를 임시로 받아둘 변수
        tmp_q=first[fn];
        if(first[fn]->next!=NULL)
        {       //pop_up하고 그 출력링크번호를 가진 queue가 남아있을때
                first[fn]=first[fn]->next; //first queue의 다음큐를 first queue로 지정
                first[fn]->size=tmp_q->size-1; //size를 하나 줄임
                cout<<"————————————————————————————————————"<<endl;
                cout<<"pop up the queue has output num : "<<output_link<<endl;
                cout<<"IP is "<<tmp_q->data->ip_address<<endl;
                delete tmp_q; //임시로 받아 뒀던 원래의 first queue를 지움
        }
        else
        {       //pop_up하고 그 출력링크번호를 가진 queue가 남아있지 않을때
                int fns;
                for(fns=0; fns<100 && first[fns]!=NULL; fns++);
                for(; fn<fns; fn++)
                {       //지워진 queue들 뒤에 있는 queue들을 앞으로 당김
                        first[fn]=first[fn+1];
                }
                cout<<"————————————————————————————————————"<<endl;
                cout<<"pop up the queue has output num : "<<output_link<<endl;
                cout<<"IP is "<<tmp_q->data->ip_address<<endl;
                delete tmp_q;
                delete first[fns];
                first[fns]=NULL; //지웠으니 NULL로 초기화를 시켜준다.
        }
}
```

③ void linked_list_Queue::output_queue(int output_link)

출력링크 번호를 받아 해당링크의 queue에 몇 개의 패킷이 있는지 출력하는 함수.
같은 출력링크 번호를 가진 first queue를 찾는다. 없으면 중지 시키고 있으면 그 first queue에 저장되어 있는 queue의 size를 출력한다.

```
void linked_list_Queue::output_queue(int output_link)
{
        int fn; //시작 queue의 포인터 number

        for(fn=0; fn<100 && first[fn]!=NULL && first[fn]->output_link!=output_link; fn++);
                //같은 출력링크번호를 가진 first queue를 찾음

        if(fn==100||first[fn]==NULL){ //같은output num을 가지지 않으면 중지
                cout<<"————————————————————————————————————————"<<endl;
                cout<<"linked_list_Queue has not output num : "<<output_link<<endl;
                return;
        }
        first queue에 그 queue의 size를 저장시켜 놔서 찾기 쉽다.
        cout<<"————————————————————————————————————————"<<endl;
        cout<<"output num : "<<output_link<<" have "<<first[fn]->size<<" packet"<<endl;
        //first queue에 저장되어 있는 queue의 size를 출력
}
```

④ void linked_list_Queue::print_queue(int output_link)

입력된 출력링크번호를 갖는 모든 큐들을 차례대로 출력

void linked_list_Queue::print(queue *printer,int size)-추가함수

위의 함수를 실행하기 위해 구현한 재귀함수

void linked_list_Queue::print_all(void)-추가함수

linked_list에 있는 모든 큐들을 출력링크별로 출력하는 함수

```
void linked_list_Queue::print_queue(int output_link)
{
	int fn; //시작 queue의 포인터 number

	for(fn=0; fn<100 && first[fn]!=NULL && first[fn]->output_link!=output_link; fn++);
		//같은 출력링크번호를 가진 first queue를 찾음

	if(fn==100||first[fn]==NULL){ //같은output num을 가지지 않으면 중지
		cout<<"──────────────────────────────────────"<<endl;
		cout<<"linked_list_Queue has not output num : "<<output_link<<endl;
		return;
	}
	cout<<"──────────────────────────────────────"<<endl;
	cout<<"Output num : "<<output_link<<" queue has following IP"<<endl;

	print(first[fn],first[fn]->size);
	//재귀함수 print를 호출
}
void linked_list_Queue::print(queue *printer,int size)
{	//재귀적 호출을 하면서 같은 output num을 가지는 큐들을 전부 출력한다(추가함수)
	if(size==1){  //size가 1이면 마지막 queue이므로 출력하고 종료한다
		cout<<printer->data->ip_address<<endl;
		return;
	}
	cout<<printer->data->ip_address<<endl;

	int sz=size-1;
	print(printer->next,sz);
	//자기자신을 호출(다음 queue를 부르면서 size값을 1줄인다)
}
void linked_list_Queue::print_all(void)
{	//linked_list에 있는 모든 queue들을 출력함(추가함수)
	cout<<" ┌──────────────────────────────────────┐ "<<endl;
	cout<<" │              Print all queue              │ "<<endl;
	cout<<" └──────────────────────────────────────┘ "<<endl;

	int fn;
	if(first[0]!=NULL)
	{	//첫번째 queue가 NULL이 아닐때 출력
		for(fn=0; fn<100 && first[fn]!=NULL; fn++)
		{
			cout<<"output num : "<<first[fn]->output_link<<" queue has following IP"<<endl;
			print(first[fn],first[fn]->size);
		}
	}
	else	//모든 큐가 비었으므로 비었다고 출력
	cout<<"	linked_list is empty"<<endl;
}
```

⑤ void linked_list_Queue::delete_all(void)-추가함수

모든 큐들을 제거한다. 제거 후 first queue들은 NULL로 초기화 시킨다.

```
void linked_list_Queue::delete_all(void)
{
        //linked_list에 있는 모든 queue들을 제거함(추가함수)
        cout<<" ┌──────────────────────────────────────┐ "<<endl;
        cout<<" |              Delete all queue              | "<<endl;
        cout<<" └──────────────────────────────────────┘ "<<endl;

        queue *new_ptr;
        queue *after_ptr;
        int fn;

        for(fn=0; fn<100 && first[fn]!=NULL; fn++)
        {
                after_ptr=first[fn]->next;
                new_ptr=first[fn];
                while(new_ptr!=NULL)
                {
                        delete new_ptr;
                        new_ptr=after_ptr;
                        if(after_ptr!=NULL)
                        after_ptr=after_ptr->next;
                }
                delete new_ptr;
                delete after_ptr;
        }
        for(int i=0; i<100; i++)
        {       //다 지웠으니 초기화
                first[i]=NULL;
        }
}
```

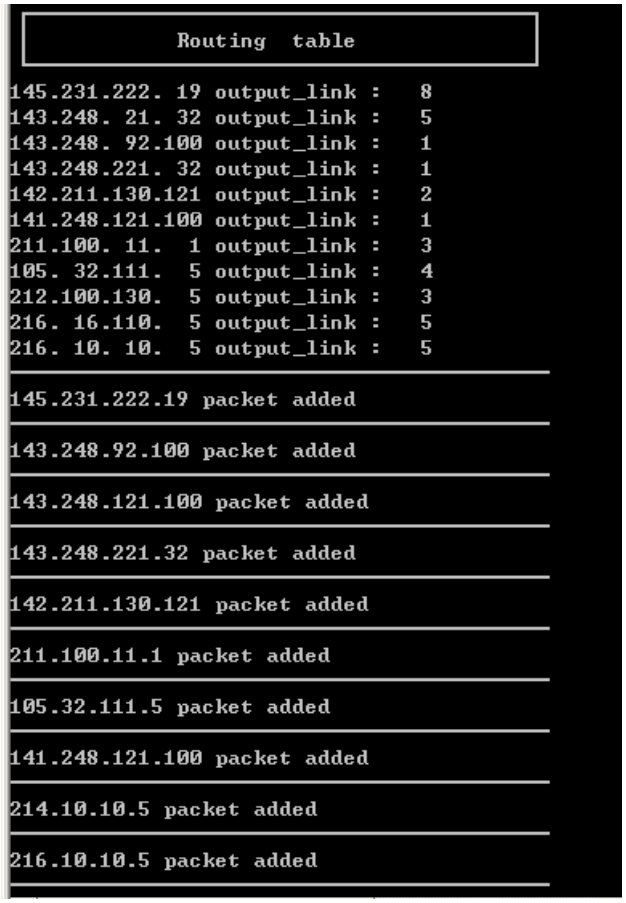
main 함수 시나리오 설명

```
void main() {
n_nary_tree t1;
Packet p0,p1,p2,p3;
Packet p4,p5,p6,p7,p8,p9;
linked_list_Queue queue(&t1);

//n_nary_tree를 이용 Routing table을 만듬
t1.enter_one("145.231.222.19",8);
t1.enter_one("143.248.21.32",5);
t1.enter_one("143.248.92.100",1);
t1.enter_one("142.211.130.121",2);
t1.enter_one("141.248.121.100",1);
t1.enter_one("211.100.11.1",3);
t1.enter_one("105.32.111.5",4);
t1.enter_one("212.100.130.5",3);
t1.enter_one("213.10.10.5",5);
t1.enter_one("143.248.221.32",1);
t1.enter_one("214.10.10.5",5);
t1.enter_one("216.16.110.5",5);
t1.enter_one("216.10.10.5",5);
t1.enter_one("218.10.10.5",5); //top node가 10개가 넘어
가서 error 메시지 출력(첫 출력-에러메시지)

//Routing table에서 입력된 IP를 찾음
cout<<t1.search_output_link("211.304.11.1")<<endl;
//없는 IP를 찾기때문에 에러메시지 출력과 0값 리턴
cout<<t1.search_output_link("211.100.11.1")<<endl;
cout<<t1.search_output_link("142.211.130.121")<<endl;
t1.print(); //Routing table출력

//delete_one을 사용해서 지움
t1.delete_one("214.10.10.5");
t1.delete_one("213.10.10.5");
t1.delete_one("214.10.10.5"); //없는 IP를 지울 수 없으므로 에러 메시지 출력
```

```
Tree's top node is full!!

I can't search IP_address
IP : 211.304.11.1 does not exist!
0

We found IP Address : 211.100.11.1
3

We found IP Address : 142.211.130.121
2

            Routing  table

145.231.222. 19 output_link :   8
143.248. 21. 32 output_link :   5
143.248. 92.100 output_link :   1
143.248.221. 32 output_link :   1
142.211.130.121 output_link :   2
141.248.121.100 output_link :   1
211.100. 11.  1 output_link :   3
105. 32.111.  5 output_link :   4
212.100.130.  5 output_link :   3
213. 10. 10.  5 output_link :   5
214. 10. 10.  5 output_link :   5
216. 16.110.  5 output_link :   5
216. 10. 10.  5 output_link :   5

Delete IP Address : 214.10.10.5

Delete IP Address : 213.10.10.5

I can't delete IP_address
IP : 214.10.10.5 does not exist!
```

여기까지가 옆에 출력된 부분이다. 일단 top node를 10개가 넘게 하여 에러 메시지를 띄어 봤다. 그리고 search함수에서 라우팅 테이블에 추가되지 않은 IP주소를 찾으면 찾을 수 없다는 에러 메시지를 띄우고 아니면 찾은 뒤 찾았다고 하고 그 IP주소의 출력링크번호를 리턴해 준다. 그런 뒤 라우팅 테이블 전체를 출력 시킨다. 그리고 delete_one함수에서 IP주소가 테이블에 있는 것이 입력되면 지우고 나서 지웠다고 출력하고 없는 IP주소가 입력되면 지울 수 없다고 출력한다.

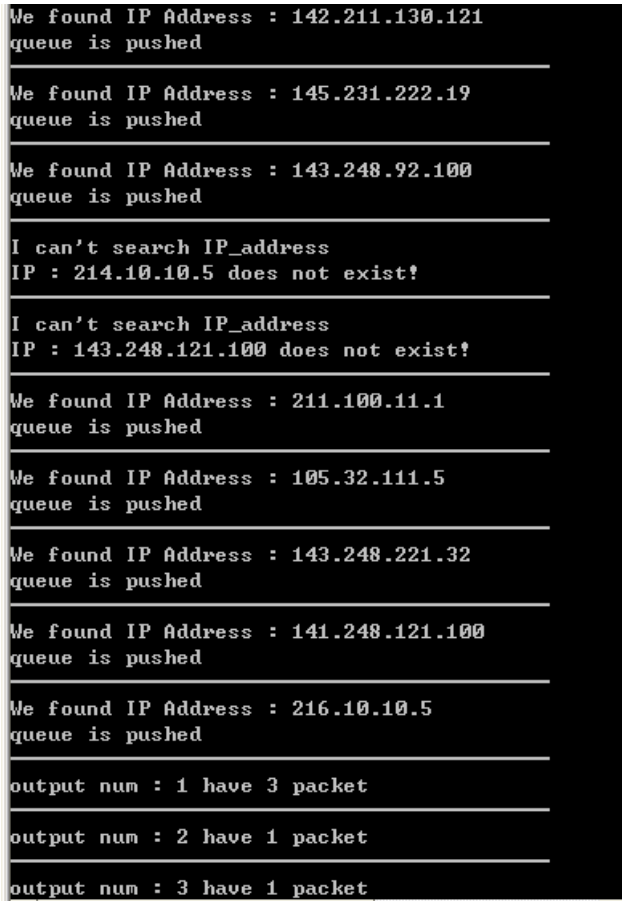
```
t1.print(); //지운후 Routing table출력

//packet 입력
p0.input_packet("145.231.222.19");
p1.input_packet("143.248.92.100");
p2.input_packet("143.248.121.100");
p3.input_packet("143.248.221.32");
p4.input_packet("142.211.130.121");
p5.input_packet("211.100.11.1");
p6.input_packet("105.32.111.5");
p7.input_packet("141.248.121.100");
p8.input_packet("214.10.10.5");
p9.input_packet("216.10.10.5");
```



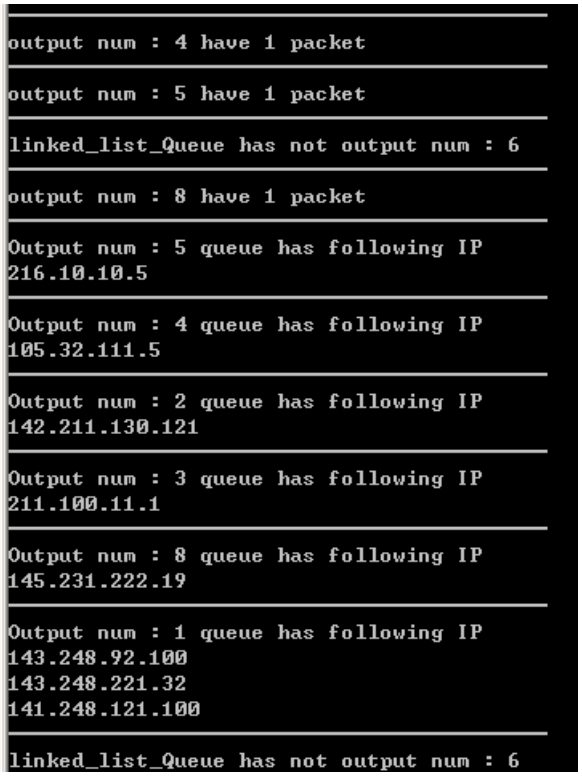
여기까지가 옆에 출력된 부분이다 위에서 라우팅 테이블의 IP를 몇 개 지운다음에 다시 출력해 보고 그런 다음 IP주소를 패킷에다가 발생시킨다. 그러면 각 IP주소가 패킷에 추가되었다고 출력한다.

```
//queue에 packet입력
queue.push_queue(&p4);
queue.push_queue(&p0);
queue.push_queue(&p1);
queue.push_queue(&p8);//없는 IP이므로 에러 메시지 출력
queue.push_queue(&p2);//없는 IP이므로 에러 메시지 출력
queue.push_queue(&p5);
queue.push_queue(&p6);
queue.push_queue(&p3);
queue.push_queue(&p7);
queue.push_queue(&p9);

//주어진 출력링크번호를 가지는 queue들의 갯수를 구함
queue.output_queue(1);
queue.output_queue(2);
queue.output_queue(3);
```

여기까지가 옆에 출력된 부분이다. 큐에 다가 패킷을 추가 시키면 추가되었다는 메시지를 출력한다. 그리고 output_queue 함수를 실행 시키면 출력링크별로 몇 개의 패킷을 가지고 있는지 출력하여 준다.

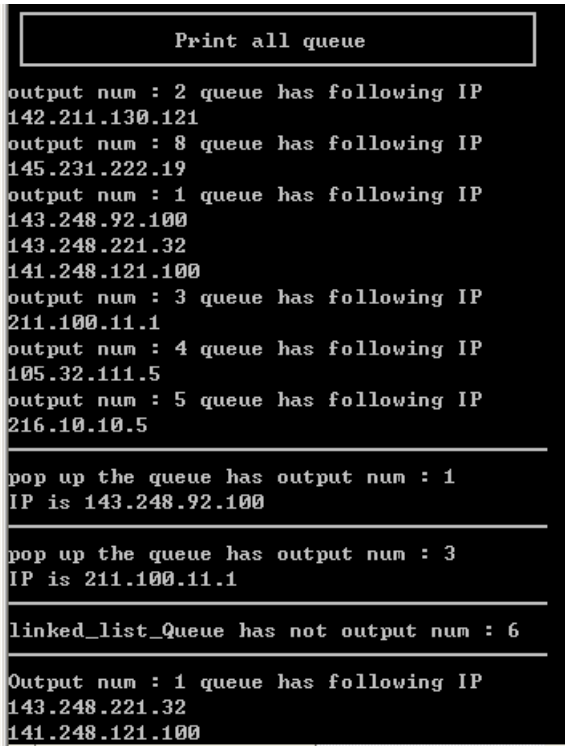


```
queue.output_queue(4);
queue.output_queue(5);
queue.output_queue(6); //output num : 6은 가지지 않으므로 에러 메시지 출력
queue.output_queue(8);

//주어진 출력링크번호를 가지는 queue들의 IP주소를 모두 출력
queue.print_queue(5);
queue.print_queue(4);
queue.print_queue(2);
queue.print_queue(3);
queue.print_queue(8);
queue.print_queue(1);
queue.print_queue(6); //output num : 6은 가지지 않으므로 에러 메시지 출력
```



여기까지가 옆에 출력된 부분이다. 출력링크번호별로 몇 개를 가진지 계속 출력해보고 출력링크번호가 없는 6을 넣었을때는 없는 출력링크번호라고 에러메시지를 띄운다. 그리고 print_queue 함수는 그 출력링크번호를 가지는 모든 패킷의 IP주소를 출력해준다. 모든 출력링크번호별로 출력해본 뒤 없는 번호 6을 넣었을때는 없는 출력링크번호라고 에러메시지를 띄운다.

```
queue.print_all(); //(추가함수) 모든 queue들을 출력함

queue.pop_up_queue(1); //출력링크가 1인 queue의 첫번째 패킷을 pop_up함
queue.pop_up_queue(3); //출력링크가 3인 queue의 첫번째 패킷을 pop_up함
queue.pop_up_queue(6); //output num : 6은 가지지 않으므로 에러 메시지 출력
queue.print_queue(1); //pop_up을 하고 나서 출력링크번호 1을 가지는 큐를 출력
```



여기까지가 옆의 출력부분이다. 새로 추가 시킨 함수 print_all의 결과를 출력 시켜 보았다. 출력링크번호별로 각각의 큐가 정렬되어서 잘 출력되었다. 그리고 pop_up_queue 함수를 실행시켜서 패킷이 3개가 있는 출력링크번호 1도 pop_up시켜 보고 패킷이 1개 밖에 없는 출력링크번호 3도 pop_up시켜 보았다.
그런 다음 출력링크번호가 없는 6을 실행시키면 없다고 에러메시지를 띄운다.
pop_up시키고 난후에 출력링크번호 1의 IP주소들을 출력시켜 보았다.

```
queue.print_all(); //(추가함수) 모든 queue들을 출력함

queue.delete_all();  //(추가함수) 모든 queue들을 제거함
queue.print_all(); //queue들이 다 지워졌으므로 비었다고 출력

t1.delete_all(); //Routing table의 모든 정보를 삭제
t1.print(); //Routing table이 비었으므로 비었다고 출력
}
```

여기까지가 옆의 출력된 부분이다.
앞에서 두개를 pop_up시켰으므로 모든 queue들을 새로 출력하여 보았다. 그리고 새로 추가시킨 함수 delete_all을 실행시켜보았다. 그런 후 제대로 잘 지워졌는지 print_all을 실행시켜보면 비었다고 에러메시지를 출력한다. 그런다음 라우팅 테이블 전체를 지우고 나서 잘 지워졌는지 또 라우팅 테이블 전체를 출력해 보면 라우팅 테이블이 비었다고 에러메시지를 출력한다.