ডাটা স্ট্রাকচার : লিংকড লিস্ট

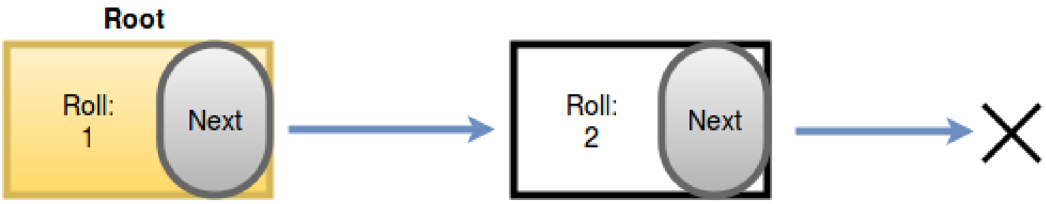
shafaetsplanet.com /planetcoding/

শাফায়েত

জানুয়ারি ৩০, ২০১৬

লিংকড লিস্ট বেসিক একটা ডাটা স্ট্রাকচার। আমরা সাধারণত তথ্য রাখার জন্য অ্যারে ব্যবহার করি, তবে অ্যারের কিছু সীমাবদ্ধতা আছে যে কারণে অনেক সময় লিংকড লিস্ট ব্যবহারের দরকার হয়। লিংকড লিস্ট নিয়ে জানতে হলে অবশ্যই পয়েন্টার সম্পর্কে ধারণা থাকতে হবে।

লিংক লিস্টের প্রতিটা এলিমেন্ট কে বলবো আমরা নোড। প্রতিটা নোডে সাধারণত দুইটা তথ্য থাকে: ১) যে তথ্যটা আমরা সংরক্ষণ করতে চাচ্ছি ২) পরবর্তি তথ্যটা কোথায় আছে তার ঠিকানা।



ছবিতে দেখা যাচ্ছে প্রথম নোড এ একজন ছাত্রের রোল নম্বর লেখা আছে, এবং পরবর্তি ছাত্রের তথ্য কোন নোড এ আছে সেটা দেখিয়ে দিচ্ছে next নামের একটা পয়েন্টার। দ্বিতীয় নোডটাই শেষ নোড, তাই এই নোডের নেক্সট পয়েন্টার একটা null নোডকে পয়েন্ট করছে। প্রথম নোডকে আমরা বলবো রুট নোড।

অ্যারের সাথে লিংক লিস্টের একটা বড় পার্থক্য হলো অ্যারের তথ্যগুলো মেমরিতে পরপর সংরক্ষণ করা হয়। যদি অ্যারেটা একটা 44 বাইটের ইন্টিজার অ্যারে হয় এবং অ্যারের প্রথম এলিমেন্টটা যদি থাকে xx তম মেমরি সেল এ, তাহলে পরের ৩টি এলিমেন্ট x+4,x+8,x+12x+4,x+8,x+12 মেমরি সেল এ থাকবে। নিচের কোডটা রান করলেই প্রমাণ পাবে।

```
int main(){
    int a[5];
    for(int i=0;i<5;i++)
    {
        printf("%u\n",&(a[i])); #print address of each element
    }
    return 0;
}
```

সেজন্য অ্যারের প্রথম এলিমেন্টের অ্যাড্রেস জানলেই এরপর যেকোনো এলিমেন্টের অ্যাড্রেস সহজেই বের করে ফেলা যায়, ইন্টিজার অ্যারের pp তম এলিমেন্ট থাকে x+p∗4x+p∗4 অ্যাড্রেসে যেখানে xx হলো শূন্যতম এলিমেন্টের অ্যাড্রেস।

লিংকড লিস্টে তথ্যগুলো থাকে ছড়িয়ে-ছিটিয়ে, তাই প্রতিটা এলিমেন্টকে পরের এলিমেন্টের ঠিকানা সংরক্ষণ করে রাখতে হয়। এই পদ্ধতির কিছু সুবিধাও আছে, অসুবিধাও আছে, সেগুলো আমরা দেখবো।

একটা সি তে লিংকড লিস্ট তৈরির জন্য শুরুতেই একটা স্ট্রাকচার ডিফাইন করতে হবে, যেখানে থাকবে যে তথ্য সংরক্ষণ করতে চাই সেটা এবং পরবর্তী নোডের অ্যাড্রেস।

```
nstruct node
{
  int roll;
node *next;
};
node *root=NULL;
int main(){
    return 0;
}

```

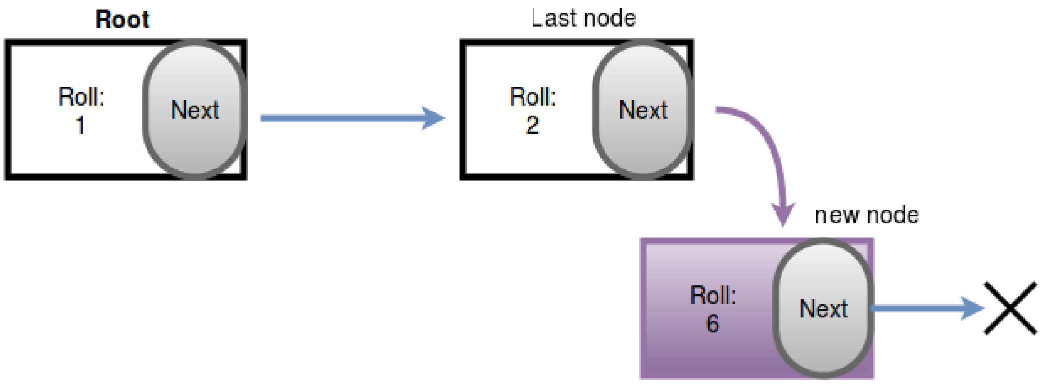
node *next হলো একটা পয়েন্টার যেটা একটা node এর অ্যাড্রেস সংরক্ষণ করে।

node *root হলো একটা পয়েন্টার যেটা সবসময় প্রথম নোডের অ্যাড্রেস সংরক্ষণ করবে। শুরুতে লিস্ট এ কোনো নোড নেই, তাই রুট পয়েন্টারের মান নাল(Null)। প্রথম নোডের অ্যাড্রেস ব্যাবহার করে আমরা পরবর্তীতে অন্য নোডের তথ্য পড়তে পারবো।

এখন আমাদের একটা ফাংশন দরকার যেটা ব্যবহার করে নতুন একটা নোড লিস্টে শেষে প্রবেশ করাতে পারবো। মনে করো ফাংশনটার নাম append। এই ফাংশনটা লেখার সময় ২টা কেস মাথায় রাখতে হবে। প্রথম কেস হলো যে নোডটা প্রবেশ করাচ্ছি সেটাই লিংক লিস্টের প্রথম নোড কি না। যদি তাই হয়, তাহলে রুট পয়েন্টার ব্যবহার করে প্রথম নোডটা তৈরি করতে হবে।

```
void append(int roll)
{
if(root==NULL) //If the list is empty
{
root=new node(); //create new node in root
root->roll=roll;
      root->next=NULL;
}
}
```

যদি লিংকড লিস্টে আগেই কিছু নোড থাকে তাহলে আমাদেরকে শেষ নোডটা খুজে বের করতে হবে। তারপর শেষ নোডের নেক্সট পয়েন্টার ব্যবহার করে পরবর্তী নোডটা তৈরি করতে হবে।



```
void append(int roll)
{
if(root==NULL) //If the list is empty
{
root=new node();
root->roll=roll;
root->next=NULL;
}
else
{
node *current_node=root; //make a copy of root node
while(current_node->next!=NULL) //Find the last node
{
current_node=current_node->next; //go to next address
}
node *newnode = new node(); //create a new node
newnode->roll=roll;
newnode->next=NULL;
current_node->next=newnode; //link the last node with new node
}
}
```

আমরা প্রথমে লুপ চালিয়ে শেষ নোডটা বের করছি। শেষ নোড কোনটা বোঝা খুব সহজ, যেই নোডের নেক্সট পয়েন্টার নাল সেটাই শেষ নোড। এরপর নতুন একটা নোড তৈরি করে শেষ নোডের সাথে সেটা লিংক করে দিচ্ছি। আমাদের এই অ্যাপেন্ড ফাংশনের কমপ্লেক্সিটি $O(n)O(n)$।

লক্ষ্য করো, রুট পয়েন্টারকে আমরা সামনে নিচ্ছি না, সেটার একটা কপি তৈরি সেটাকে সামনে নিচ্ছি। কারণ রুট পয়েন্টারকে আমরা সামনে নিলে প্রথম নোডের অ্যাড্রেস হারিয়ে ফেলবো!

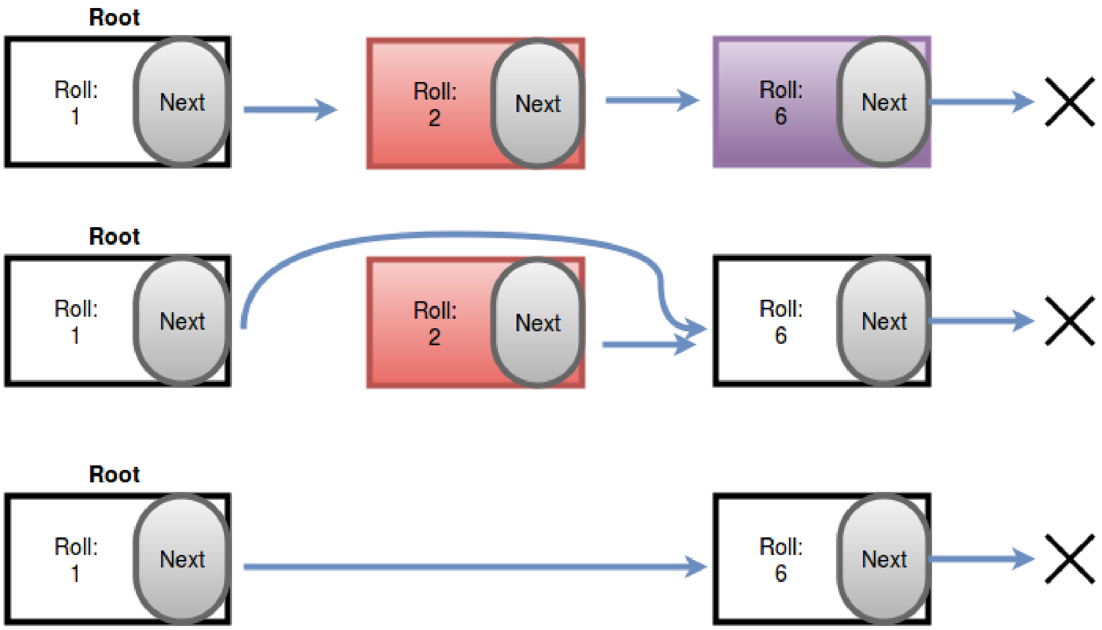
সবগুলো ছাত্রের রোল নম্বর প্রিন্ট করতে চাইলেও একইভাবে করতে পারবো। আগের মতই লুপ চালিয়ে শেষ নোড পর্যন্ত যাবো এবং সবগুলো মান প্রিন্ট করবো।

```
void print()
{
node *current_node=root;
while(current_node!=NULL) //loop until you reach null
{
printf("%d\n",current_node->roll);
current_node=current_node->next;
}
}
int main(){

append(1);
append(2);
append(6);
print();
    return 0;
}
```

এখন তুমি যদি চাও শুধুমাত্র ১০ তম ছাত্রের রোল প্রিন্ট করতে, তাহলে কি করবে? তোমাকে লুপ চালিয়ে ১০ নম্বর নোড খুজে বের করে প্রিন্ট করতে হবে। কিন্তু অ্যারেতে আমরা $roll[10]roll[10]$ লিখেই ১০তম ছাত্রের রোল প্রিন্ট করে ফেলতে পারতাম। লিংকড লিস্টে তথ্যগুলো মেমরিতে পরপর সাজান্য নেই তাই রেন্ডম এক্সেস করা যায় না। লিংকড লিস্টে কোনো ইনডেক্স খুজে বের করার কমপ্লেক্সিটি তাই $O(n)O(n)$, যেখানে অ্যারেতে $O(1)O(1)$। [পুরানো আমলের গানশোনার ফিতার ক্যাসেটগুলোর কথা মনে আছে? সেখানেও কোনো গানে লাফ দিয়ে চলে যাওয়া যেত না, ফিতা ঘুরিয়ে খুজে বের করতে হতো। এখানেও একই ব্যাপার ঘটছে!]

লিংক লিস্ট এর সুবিধা হলো চাইলেও কোনো তথ্য মাঝখান থেকে মুছে ফেলা যায়। অ্যারেতে তুমি চাইলেই মাঝখান থেকে একটা ইনডেক্স মুছে ফেলতে পারবে না, মুছতে হলে ডানের সব এলিমেন্টকে একঘর বামে টেনে এনে ফাকা জায়গা পূরণ করতে হবে, এবং সবার শেষের এলিমেন্টটাকে মুছে ফেলতে হবে। কিন্তু লিংকড লিস্টে তুমি সহজেই মাঝখান থেকে একটা নোড মুছে ফেলতে পারবে।



ছবিতে রোল ২ কে কিভাবে মুছে ফেলা যায় দেখানো হয়েছে। রোল ২ এর আগের নোড রোল ১ এর পয়েন্টারকে দিয়ে রোল ২ এর পরের নোড এর অ্যাড্রেস কে পয়েন্ট করানো

হয়েছে, এবং মাঝের নোডটা মেমরি থেকে মুছে ফেলা হয়েছে।

লক্ষ্য করো, রুট নোডের আগে কোনো নোড নেই। তাই রুট নোড মুছে ফেলা আরো সহজ, শুধুমাত্র রুট পয়েন্টার এক ঘর এগিয়ে দিতে হবে এবং আগের নোডটা মুছে ফেলতে হবে।

```
void delete_node(int roll)
{
node *current_node=root;
node *previous_node=NULL;
while(current_node->roll!=roll) //Searching node
{
previous_node=current_node; //Save the previous node
current_node=current_node->next;
}
if(current_node==root) //Delete root
{
node *temp=root; //save root in temporary variable
root=root->next; //move root forward
delete(temp); //free memory
}
else //delete non-root node
{
previous_node->next=current_node->next; //previous node points the current node's next node
delete(current_node); //free current node
}
}

```

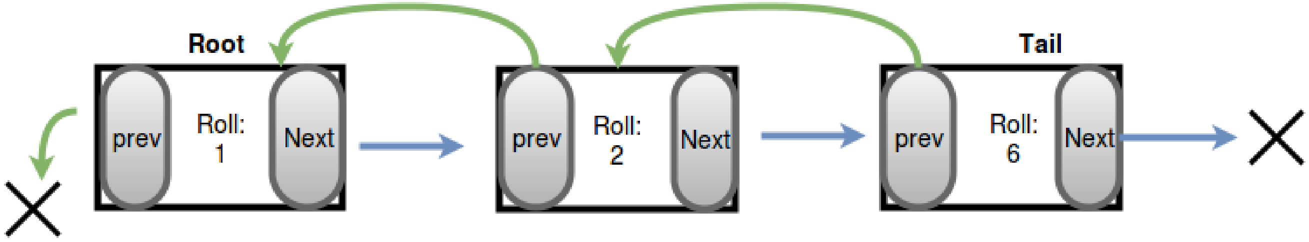
উপরের কোডে প্রথমে আমরা খুজে বের করেছি যে রোল নম্বরটা মুছতে হবে সেই নোডটাকে। যদি সেটাই রুট নোড হয় তাহলে রুটকে একঘর এগিয়ে দিয়েছি, নাহলে উপরের ছবির মত করে মুছেছি।

লক্ষ্য করো, আমি delete(node) নামের একটা লাইব্রেরি ফাংশন ব্যবহার করেছি। মোছার সময় পয়েন্টার ঠিকঠাক করার পর অবশ্যই delete ফাংশন ব্যবহার করে মেমরি ফ্রি করে দিতে হবে, নাহলে লিংকড লিস্ট থেকে নোড মুছে গেলেও নোডটা মেমরিতে থেকে যাবে, অন্য কোনো প্রোগ্রাম সেটাকে ব্যবহার করতে পারবে না। লিংকড লিস্টের কোড লেখার সময় delete() ফাংশন ব্যবহার করতে ভুলে যাওয়া খুবই কমন একটা ভুল।

লিংকড লিস্ট এ তুমি চাইলে মাঝখানেও নোড যোগ করতে পারবে। এই পর্যন্ত বুঝে থাকলে তোমার কাজ হবে দুই নোড এর মাঝে নতুন নোড যোগ করার জন্য ফাংশন লেখা। এটা অনেকটা delete-node ফাংশনের মত করে লিখতে হবে। ফাংশনের প্যারামিটার হিসাবে নিবে roll1, roll2, তোমার ফাংশনের কাজ হবে roll1 যে নোডে আছে সেটা খুজে বের করে সেটার পরে roll2 নোডটা যোগ করা।

বাইডিরেকশনাল লিংকড লিস্ট

আমাদের আগের কোড এ অ্যাপেন্ড অপারেশন ছিলো O(n), লুপ চালিয়ে বারবার শেষ পর্যন্ত যেতে হচ্ছিলো। এছাড়া লিংকড লিস্টটা উল্টো দিক থেকে ট্রাভার্স করা সম্ভব হচ্ছিলো না।



উপরের ছবির লিংক লিস্টে প্রতি নোডে দুইটা পয়েন্টার ব্যবহার করা হয়েছে। prev পয়েন্টারটা প্রতিটা নোডের আগের নোডের অ্যাড্রেসকে পয়েন্ট করে আছে। এছাড়াও এআমরা কটা tail পয়েন্টার এর সাহায্যে শেষ নোডটার অ্যাড্রেস মনে রাখছি।

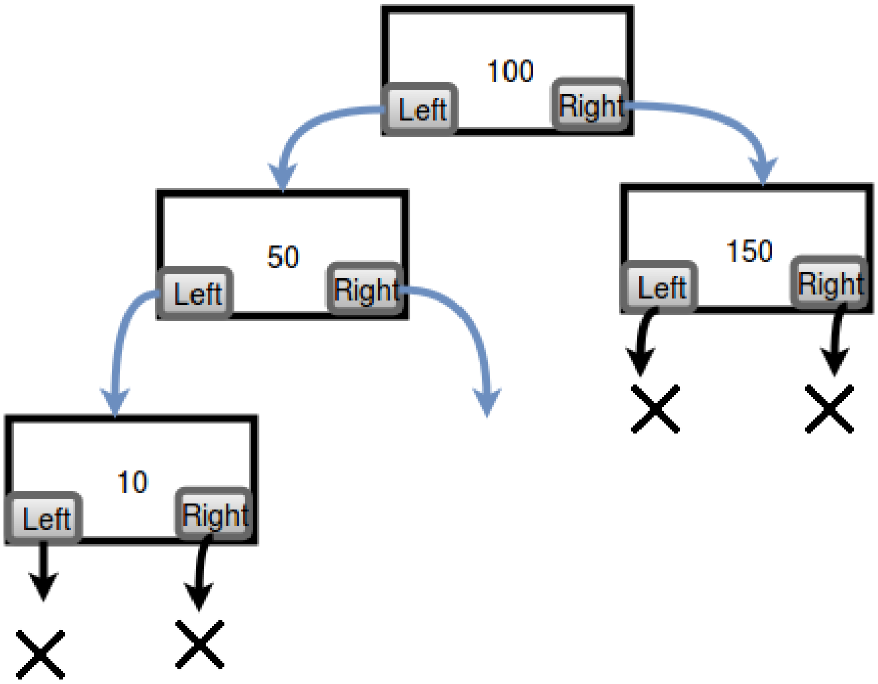
এখন লিংকড লিস্টের শেষে নতুন নোড যোগ করা সম্ভব O(1) কমপ্লেক্সিটিতে।

```
struct node
{
int roll;
node *next, *prev;
};
node *root, *tail;
void append(int roll)
{
if(root==NULL) //If the list is empty
{
root=new node();
root->roll=roll;
root->next=NULL;
tail=root;
}
else
{
node *newnode=new node();
newnode->roll=roll;
newnode->next=NULL;
tail->next=newnode; //add the new node after tail node
tail=tail->next; //move tail pointer forward
}
}
```

এবার append ফাংশন খুব সহজ হয়ে গিয়েছে। নতুন নোডটা tail এর সাথে যোগ করে টেইল এক ঘর এগিয়ে নেয়া হচ্ছে। নোড ডিলিট করার সময়েও এখন আর previous_node ভ্যারিয়েবলটা রাখা দরকার নেই।

বাইনারি সার্চ ট্রি

লিংকড লিস্টের আরেকটা ব্যবহার হলো বিভিন্ন ধরণের ট্রি তৈরি করা। যেমন নিচে একটা বাইনারি সার্চ ট্রি তৈরি করা হয়েছে:



বাইনারি সার্চ ট্রি তে বাম পাশের নোডে সবসময় ছোটো মান, ডানের নোডে সমান বা বড় মান থাকে। আমাদের left এবং right নামের দুইটা পয়েন্টার লাগবে।

```
struct node
{
int roll;
node *left, *right;
node() //initialize the node using null
{
left=NULL;
right=NULL;
}
};
node *root;
```

নতুন নোড ইনসার্ট করার সময় আগের মতই লুপ চালিয়ে শেষ নোডে আসতে হবে। লুপ চালানোর সময় বামে নাকি ডানে যাবে সেটা নোডের মান এবং নতুন মান তুলনা করে বের করতে হবে।

```
void insert(int roll)
{
if(root==NULL) //first node in tree
{
root=new node();
root->roll=roll;
}
else
{
node *current=root,*parent;
while(current!=NULL)
{
if(roll<current->roll)
{
parent=current; //keep track of parent node
current=current->left;
}
else
{
parent=current;
current=current->right;
}
}
node *newnode=new node();
newnode->roll=roll;
if(newnode->roll<parent->roll) parent->left=newnode;
else parent->right=newnode;
}
}
```

কোড ঠিক আছে নাকি বোঝার জন্য একটা প্রিন্ট ফাংশন লিখি:

```
void print_preorder(node *current)
{
if(current==NULL) return;
cout<<current->roll<<endl;
print_preorder(current->left);
print_preorder(current->right);
}

```

এই ফাংশনটা রিকার্সিভলি নোডগুলার মান প্রিন্ট করবে।

কোডটা যদি বুঝে থাকো তাহলে বাইনারি সার্চ ট্রি থেকে নোড মুছে ফেলার ফাংশনটা লিখে ফেলো। নোড মোছার সময় বেশ কয়েকটা কেস চিন্তা করতে হয়, সেটা নিয়ে এখানে আলোচনা করবো না। তুমি কোরম্যানের অ্যালগোরিদম বই থেকে বা গুগলে একটু সার্চ করে শিখে নিতে পারো।

লিংকড লিস্টে সাইকেল কিভাবে বের করতে হয় জানতে আমার এই লেখাটা পড়ো।

লিংকড লিস্টের কোড লেখার সময় কিছু কমন ভুল হয় শুরুর দিকে। যেমন পয়েন্টারের মান নাল হয়ে যাবার পরেও মান প্রিন্ট করার চেষ্টা করে বা আরো সামনে আগানোর চেষ্টা করা, সেক্ষেত্রে কোড রান টাইম ইরোর দিবে। এছাড়া মেমরি ফ্রি করতে ভুলে যাওয়াও খুব সাধারণ একটা ভুল। আরেকটা ভুল হলো রুট বা টেইল পয়েন্টারের মান বদলে ফেলা।

লিংকড লিস্ট শেখার পর স্ট্যাক, কিউ, বাইনারি ট্রি, হিপ ইত্যাদি ডাটা স্ট্রাকচারগুলো লিংকড লিস্ট দিয়ে ইমপ্লিমেন্ট করা শিখতে হবে। তুমি যদি কনটেস্ট করো আর আরেকটু অ্যাডভান্সড কিছু শিখতে চাও তাহলে এখান থেকে ট্রাই ডাটা-স্ট্রাকচার কিভাবে লিংকড লিস্ট ব্যাবহার করে ইমপ্লিমেন্ট করে শিখে নিতে পারো।

হ্যাপি কোডিং!