ডাটা স্ট্রাকচার: ট্রাই (প্রিফিক্স ট্রি/রেডিক্স ট্রি)

ইন্টারনেট যারা ব্যবহার করে তারা সবাই মনে হয় কখনো না কখনো এটা ভেবে অবাক হয়েছে যে গুগলে সার্চ করলে কিভাবে এত তথ্য চোখের নিমিষে সামনে চলে আসে! চিন্তা করে দেখো হাজার কিলোমিটার দূরে দানবাকৃতির ডাটাবেস থেকে প্রয়োজনীয় ডাটা খুঁজে তোমার কাছে পাঠিয়ে দিচ্ছে কয়েক সেকেন্ডের মধ্যে। শুধুমাত্র শক্তিশালী হার্ডওয়্যার, দ্রুতগতির ইন্টারনেট দিয়ে এটা সম্ভব না, এর পিছে আছে দারুণ কিছু সার্চিং অ্যালগোরিদম এবং ডাটা স্ট্রাকচার।

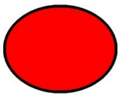
আমরা সেসব জটিল জিনিসে যাবোনা, আমরা আজকে শিখবো খুব সহজে ইমপ্লিমেন্ট করা যায় এমন একটা ডাটা স্ট্রাকচার যেটা ব্যবহার করে তুমি খুব দ্রুত অনেক অনেক নাম এর মধ্য থেকে একটা নাম খুজে বের করতে পারবে, ডাটাবেসে যত মিলিয়ন নামই থাকুক না কেন তুমি যে নামটা খুজে বের করতে চাও সেটাতে যতগুলো অক্ষর আছে সেটার উপর নির্ভর করবে কত সময় লাগবে সেটা খুজে বের করতে। তারমানে এখানে তোমার সার্চিং কমপ্লেক্সিটি ডাটাবেস সাইজের উপর নির্ভরশীল না, দারুণ একটা ব্যাপার তাইনা?

ধরো তোমাকে একটা ডিকশনারী দেয়া হলো যেখানে নিচের শব্দগুলো আছে:

> *algo*
> *algea*
> *also*
> *tom*
> *to*

এখন আমরা এই ডিকশনারিটাকে এমনভাবে মেমরিতে রাখতে চেষ্টা করবো যেন খুব সহজে কোনো একটা শব্দ খুজে পাওয়া যায়। একটা উপায় হলো শব্দগুলোকে সর্ট করে রাখা যেটা রিয়েল লাইফে কাগজের ডিকশনারি গুলোতে রাখা হয়, তাহলে বাইনারি সার্চ করেই আমরা কোনো একটা শব্দ খুজে বের করতে পারবো। আরেকটা উপায় হলো প্রিফিক্স ট্রি বা সংক্ষেপে ট্রাই(trie) ব্যবহার করা। trie শব্দটা এসেছে "re**trie**val" শব্দটা থেকে। সেই হিসাবে এটার উচ্চারণ "ট্রি" হওয়ার কথা কিন্তু গ্রাফ থিওরীতে ট্রি এর আরো ব্যপক ব্যবহার আছে তাই এটাকে বলা হয় "ট্রাই" । **প্রিফিক্স মানে হলো একটা স্ট্রিং এর শুরু থেকে কয়েকটা ক্যারেকটার নিয়ে নতুন স্ট্রিং তৈরি করা। যেমন blog এর প্রিফিক্স হলো b,bl,blo এবং blog।**

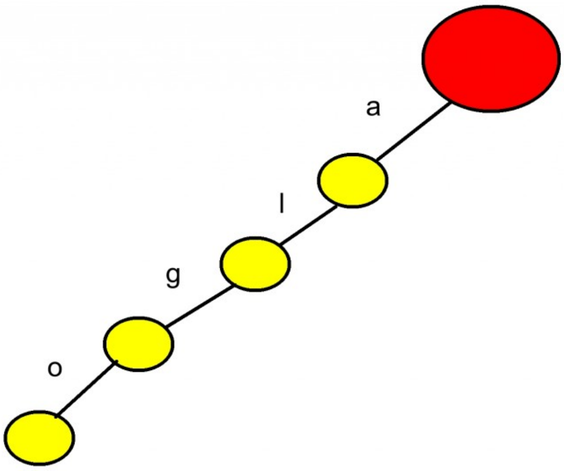
শুরুতে আমাদের একটা রুট নোড ছাড়া কিছুই নেই।

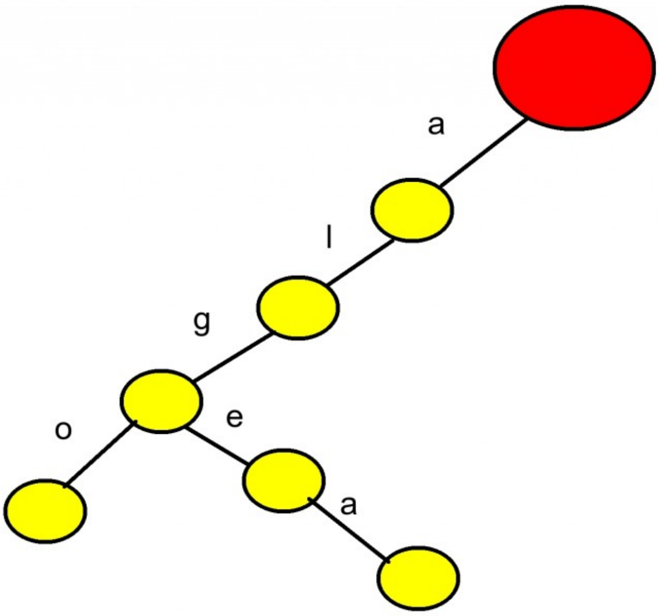


Empty tree with just the root

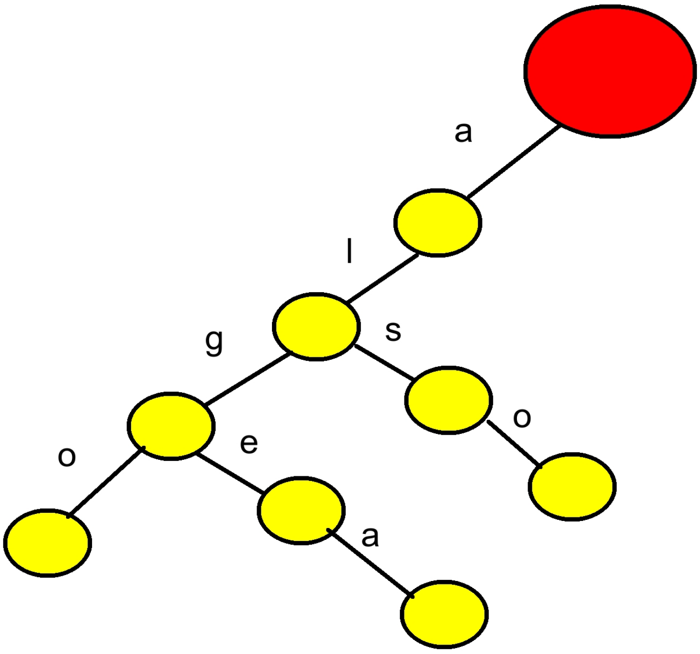
এখন আমরা algo শব্দটাকে যোগ করবো। নিচের ছবিতে দেখো কিভাবে শব্দটা যোগ করা হয়েছে। রুট নোড থেকে আমরা একটা এজ দিবো যেই এজ এর নাম হবে "a"। তারপর নতুন তৈরি হওয়া নোড থেকে "l" নামের একটা এজ তৈরি করবো। এভাবে "g" আর "o" এজ দুইটাও তৈরি করবো। লক্ষ্য করো নোডে আমরা কোনো তথ্য রাখছিনা, খালি নোড থেকে এজ বের করছি।

এখন আমরা algea শব্দটা যোগ করতে চাই। রুট থেকে "a" নামের এজ দরকার, সেটা অলরেডি আছে, নতুন করে যোগ করা দরকার নাই। ঠিক সেরকম a থেকে l এবং l থেকে g তেও এজ আছে। তারমানে "alg" অলরেডি ট্রাই তে আছে, আমরা শুধু e আর a যোগ করবো।

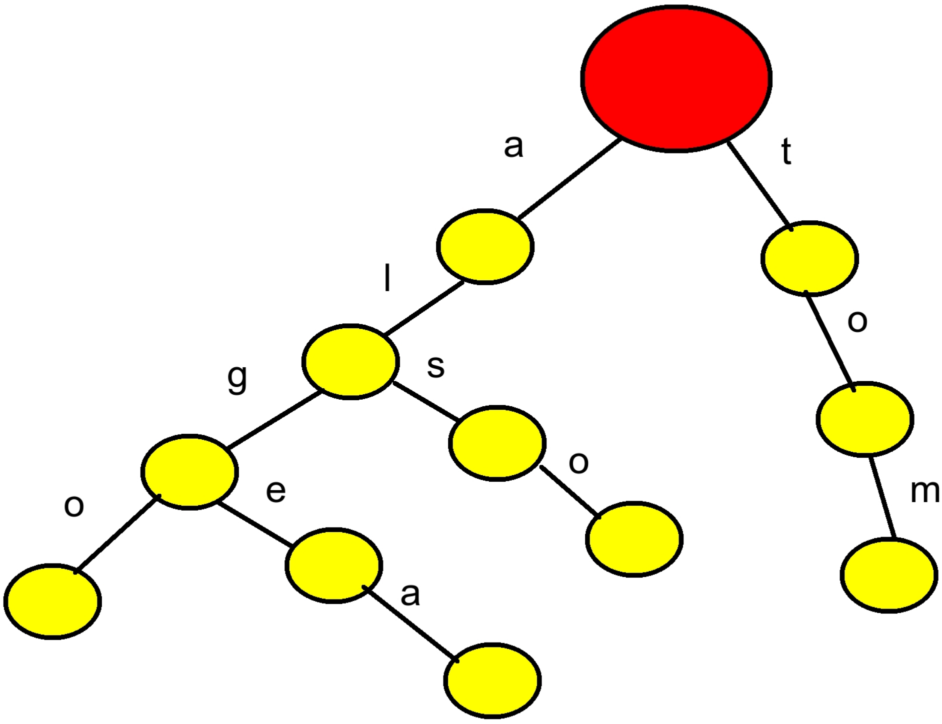




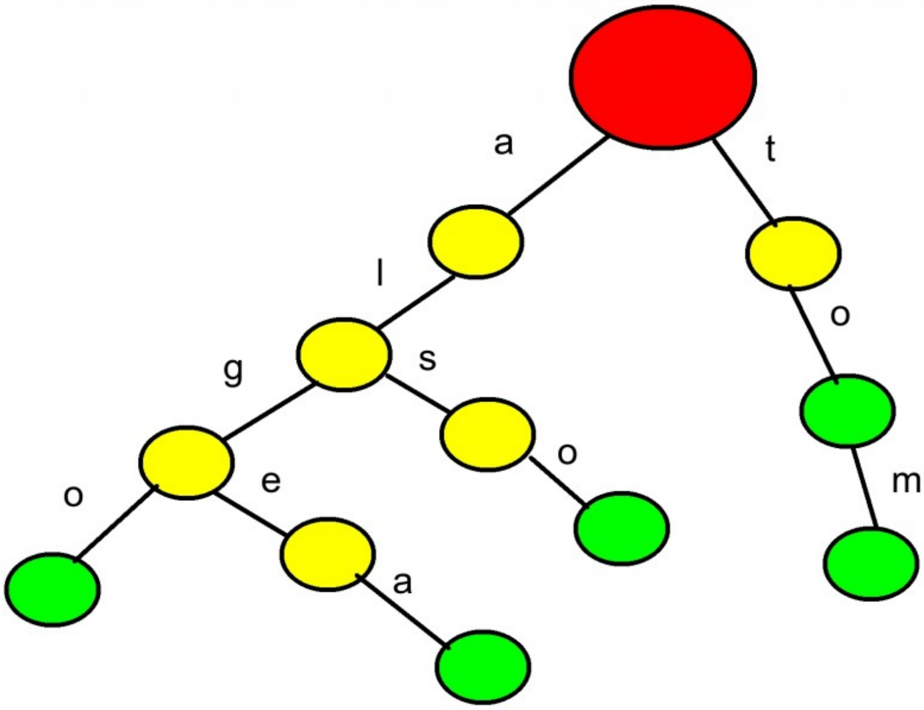
also শব্দটাকে যোগ করবো এবার। রুট থেকে “al” প্রিফিক্স এরইমধ্যে আছে, শুধু “so” যোগ করতে হবে।



এবার “tom” যোগ করি। এবার রুট থেকে নতুন এজ তৈরি করতে হবে কারণ tom এর কোনো প্রিফিক্স আগে যোগ করা হয়নি।



এখন “to” শব্দটা যোগ করবো কিভাবে? “to” পুরোপুরি tom এর প্রিফিক্স তাই নতুন কোনো এজ যোগ করা দরকার নাই। আমরা যে কাজটা করতে পারি সেটা বলে নোডগুলোতে কিছু এন্ড-মার্ক বসানো। যেসব নোডে এসে অন্তত একটা শব্দ কমপ্লিট হয়েছে সেসব নোডে আমরা এন্ডমার্ক বসিয়ে দেই, ছবিতে সবুজ রঙ হলো এন্ডমার্ক। আগের সব শব্দের জন্য এবং সেই সাথে নতুন শব্দ “to” এর জন্য এন্ডমার্ক বসালে ট্রাইটা এরকম দেখাবে:



নিশ্চয়ই বুঝতে পারছো সবুজ মার্কগুলো কেন বসিয়েছি। মার্ক দেখে সহজেই বোঝা যাচ্ছে কোন কোন শব্দ ট্রাইতে আছে। কোন ক্যারেকটার নিচ্ছি সেই তথ্য থাকবে এজ এ, আর এন্ডমার্কগুলো থাকবে নোড এ।

এভাবে শব্দগুলো রাখার সুবিধা কি? ধরো তোমাকে বলা হলো “alice” শব্দটা ডিকশনারিতে আছে কিনা বলতে। তুমি শুরু থেকে ট্রাই ধরে আগাতে থাকো। প্রথমে দেখো রুট থেকে a নামের এজ আছে নাকি, তারপর চেক করো a থেকে l নামের এজ আছে নাকি। এরপরে l থেকে i নামের এজ খুজে পাওয়া যাচ্ছেনা, তাই বলতে পারো alice শব্দটা নেই।

“alg” শব্দটা খুজতে দিলে তুমি root->a, a->l এবং l->g এজগুলো সবই খুজে পাবে, কিন্তু শেষ পর্যন্ত কোনো সবুজ নোডে যেতে পারবেনা, তারমানে alg ও ডিকশনারিতে নেই। “tom” খুজতে গেলে তুমি একটা সবুজ নোডে গিয়ে শেষ করবে তাই শব্দটা ডিকশনারিতে আছে।

ট্রাই ইমপ্লিমেন্ট করার সহজ একটা উপায় হলো লিংকড লিস্ট ব্যবহার করা। লিংকড লিস্ট, পয়েন্টার এসব শুনে ভয়ের কিছুই নেই, তুমি যদি লিংকলিস্ট ব্যবহার করতে অভ্যস্ত নাও হও আশা করি এই ইমপ্লিমেন্টেশনটা দেখে শিখে ফেলতে পারবে।আমাদের প্রতিটা নোডে ২টি জিনিস থাকবে:

১. এন্ড-মার্ক রাখার জন্য একটা ভ্যারিয়েবল।

২. প্রতিটা নোড থেকে a,b,c,……..,x,y,z ইত্যাদি নামের এজ তৈরি হতে পারে। আমরা প্রতিটা ক্যারেক্টারের জন্য একটা পয়েন্টার রাখবো। পয়েন্টারের সাহায্যে একটা নোড আরেকটার সাথে যোগ হবে। a নামের পয়েন্টার দিয়ে যোগ হলে বুঝতে হবে কারেন্ট নোড থেকে a নামের একটা এজ আছে। শুরুতে সবগুলো পয়েন্টার “নাল” থাকবে।

আমরা প্রথমেই নোডটা তৈরি করে ফেলি:

C++

```
struct node
{
   bool endmark;
   node *next[26+1];
   node()
   {
      endmark=false;
      for(int i=0;i<26;i++) next[i]=NULL;
   }
}*root;
int main(){
   root=new node();
   return 0;
}
```

“next[]” অ্যারের প্রতিটা এলিমেন্ট আরেকটা নোডকে পয়েন্ট করে। next[0] দিয়ে নতুন নোডকে পয়েন্ট করা হলে সেই এজ এর নাম “a”, next[1] এর জন্য এজ এর নাম “b”, next[25] এর জন্য “z”। শুরুতে সবগুলো পয়েন্টার নাল।

লক্ষ্য করো নোডের ভিতর একটা কনস্ট্রাক্টর “node()” তৈরি করেছি। যখনই নতুন নোড তৈরির জন্য new node() কল করবো তখনই ভ্যারিয়েবলগুলোকে শূন্য বা নাল বানিয়ে দিবে। এটা না দিলে গার্বেজ ভ্যালু থাকতো। root ভ্যারিয়েবলটা হলো আমাদের রুট নোড, উপরের ছবিগুলোতে লাল রঙ এর নোড। আসলে রুট একটা পয়েন্টার, যখন root=new node(); লাইনটা এক্সিকিউট হবে তখনই একটা নতুন নোড তৈরি করে root যে মেমরি অ্যাড্রেসকে পয়েন্ট করে সেখানে অ্যাসাইন করে দেয়া হবে। এটাকে একটু গালভরা ভাষায় বলে “instance” তৈরি করা।

এবার আমাদের একটা ফাংশন লাগবে নতুন শব্দ ট্রাইতে যোগ করার জন্য:

C++

```
void insert(char *str,int len)
{
   node *curr=root;
   for(int i=0;i<len;i++)
   {
      int id=str[i]-'a';
      if(curr->next[id]==NULL)
         curr->next[id]=new node();
      curr=curr->next[id];
   }
   curr->endmark=1;

}
```

রুট ভ্যারিয়েবলটা আমাদের সবসময় দরকার হবে তাই “curr” এর মধ্যে সেটার কপি তৈরি করে কাজ করি। যেহেতু পয়েন্টার নিয়ে কাজ করছি তাই রুট থেকে নতুন এজ তৈরি করা আর “curr” থেকে নতুন এজ তৈরি করা একই কথা। আমরা এখন শুধু a-z নিয়ে কাজ করছি, তাই অ্যাসকি ভ্যালুগুলোকে 0-25 এ কনভার্ট করে নিবো ‘a’ এর অ্যাসকি ভ্যালু বিয়োগ করে। insert করা খুব সহজ, আমরা শুধু চেক করবো কারেন্ট নোড (curr) থেকে বর্তমানে যে ক্যারেকটারে আছি সেই নামের কোনো এজ আছে নাকি, না থাকলে নতুন নোড তৈরি করতে হবে। এরপরে সেই এজ ধরে আমরা পরের নোডে যাবো। সবার শেষ নোডটায় এন্ড-মার্ক true করে দিবো।

ইনসার্ট করার পর এখন সার্চ করবো। এটা আসলে ঠিক ইনসার্ট এর মতোই। পার্থক্য হলো যে এজটা দরকার সেটা না থাকলে তৈরি করে নিচ্ছিলাম, এখন এজ না থাকলে false রিটার্ন করে দিবো।

C++

```
bool search(char *str,int len)
{
    node *curr=root;
    for(int i=0;i<len;i++)
    {
        int id=str[i]-'a';
        if(curr->next[id]==NULL) return false;
        curr=curr->next[id];
    }
    return curr->endmark;
}
```

লক্ষ্য করো সবকাজ ইনসার্ট এর মতোই করেছি। সবার শেষে লাস্ট নোডটার এন্ডমার্ক রিটার্ণ করে দিয়েছি। এন্ডমার্ক true হলে শব্দটা আছে, false হলে নাই।

আমাদের মূল কোড শেষ। আমরা এখন যেকোনো শব্দ ট্রাইতে যোগ করতে পারবো, আবার ট্রাই থেকে কোনো শব্দ খুজতে পারবো। অনেক সময় প্রতিটা টেস্টকেস এর জন্য ট্রাই তৈরি করতে গেলে মেমরি লিমিট নিয়ে সমস্যা হয়। তাই নিরাপদ উপায় হলো প্রতি কেস এর পর ব্যবহৃত মেমরি-সেল গুলোকে ডিলিট করে দেয়া। শুধু root ডিলিট করলে হবেনা, প্রতিটা নোড করতে হবে। আমরা সে জন্য একটা রিকার্সিভ ফাংশন লিখতে পারি:

C++

```
void del(node *cur)
{
     for(int i=0;i<26;i++)
          if(cur->next[i])
               del(cur->next[i]) ;
          delete(cur) ;
}
```

এই ফাংশনটা প্রতিটা নোডে গিয়ে আগে চাইল্ডগুলোকে ডিলিট করে এসে তারপর নোডটাকে ডিলিট করে দিবে।

সম্পূর্ণ কোডটা এরকম:

C++

```
struct node
{
    bool endmark;
    node *next[26+1];
    node()
    {
        endmark=false;
        for(int i=0;i<26;i++) next[i]=NULL;
    }
}*root;

void insert(char *str,int len)
{
    node *curr=root;
    for(int i=0;i<len;i++)
    {
        int id=str[i]-'a';
        if(curr->next[id]==NULL)
            curr->next[id]=new node();
        curr=curr->next[id];
    }
    curr->endmark=true;

}
```

```
bool search(char *str,int len)
{
    node *curr=root;
    for(int i=0;i<len;i++)
    {
        int id=str[i]-'a';
        if(curr->next[id]==NULL) return false;
        curr=curr->next[id];
    }
    return curr->endmark;
}
void del(node *cur)
{
     for(int i=0;i<26;i++)
            if(cur->next[i])
                 del(cur->next[i]) ;

          delete(cur) ;
}
int main(){

    puts("ENTER NUMBER OF WORDS");
    root=new node();
    int num_word;
    cin>>num_word;
    for(int i=1;i<=num_word;i++)
    {
        char str[50];
        scanf("%s",str);
        insert(str,strlen(str));
    }
    puts("ENTER NUMBER OF QUERY";);
    int query;
    cin>>query;
    for(int i=1;i<=query;i++)
    {
        char str[50];
        scanf("%s",str);
        if(search(str,strlen(str)))    puts("FOUND");
        else puts("NOT FOUND");
    }
    del(root); //ট্রাইটা ধ্বংস করে দিলাম
    return 0;
}

```

কমপ্লেক্সিটি: প্রতিটা শব্দ খুজতে লুপ চালাতে হচ্ছে শব্দটার লেংথ পর্যন্ত, সার্চিং এর কমপ্লেক্সিটি O(length)। প্রতিটা শব্দ ইনসার্ট করার কমপ্লেক্সিটিও একই। মেমরি কতখানি লাগবে সেটা ডিপেন্ড করে ইমপ্লিমেন্টেশন এবং শব্দগুলোর প্রিফিক্স কতখানি ম্যাচ করে তার উপর। উপরের ইমপ্লিমেন্টেশন দিয়ে প্রায় 10^6 টা ক্যারেকটার ট্রাইতে ইনসার্ট করা যাবে (10^6 টা ওয়ার্ড নয়, ক্যারেকটার বা লেটার)।

তুমি চাইলে ট্রাই লিংকলিস্ট ছাড়া সাধারণ অ্যারে ব্যবহার করে ইমপ্লিমেন্ট করতে পারো, নিজে চেষ্টা করো!

ট্রাই এর কিছু ব্যবহার:

১. একটা ডিকশনারিতে অনেকগুলো শব্দ আছে, কোনো একটা শব্দ আছে নাকি নাই খুজে বের করতে হবে। এই প্রবলেমটা আমরা উপরের কোডে সলভ করেছি।

২. ধরো তোমার ৩ বন্ধুর টেলিফোন নম্বর হলো “৫৬৭৮”, “৪৩২২”, “৫৬৭”। তুমি যখন প্রথম বন্ধুকে ডায়াল করবে তখন ৫৬৭ চাপার সাথে সাথে ৩ বন্ধুর কাছে ফোন চলে যাবে কারণ ৩য় বন্ধুর নাম্বার প্রথম জনের প্রিফিক্স! অনেকগুলো ফোন নম্বর দেয়া আছে, বলতে হবে এরকম কোনো নম্বর আছে নাকি যেটা অন্য নম্বরের প্রিফিক্স। (UVA 11362)।

৩. একটা ডিকশনারিতে অনেকগুলো শব্দ আছে। এখন কোনো একটা শব্দ কয়বার “prefix” হিসাবে এসেছে সেটা বের করতে হবে। যেমন “al” শব্দটা উপরের ডিকশনারিতে ৩বার প্রিফিক্স হিসাবে এসেছে (algo, algea, also এই সবগুলো শব্দের প্রিফিক্স “al”)। এটা বের করার জন্য প্রতিট নোডে একটা কাউন্টার ভ্যারিয়েবল রাখতে হবে, কোনো নোডে যতবার যাবে ততবার কাউন্টারের মান বাড়িয়ে দিবে। সার্চ করার সময় প্রিফিক্সটা খুজে বের করে কাউন্টারের মান দেখবে।

৪. মোবাইলের ফোনবুকে সার্চ করার সময় তুমি যখন কয়েকটা লেটার লিখো তখন সেই প্রিফিক্স দিয়ে কি কি নাম শুরু হয়েছে সেগুলো সাজেশন বক্সে দেখায়। এটা তুমি ট্রাই দিয়ে ইমপ্লিমেন্ট করতে পারবে?

৪. দুঃ্টি স্ট্রিং এর “longest common substring” বের করতে হবে। (subsequence হলে ডিপি দিয়ে সহজে করা যায়, এখানে substring চেয়েছি)।
(হিন্টস: একটা স্ট্রিং এর শেষ থেকে এক বা একাধিক ক্যারেকটার নেয়া হলে সেটাকে স্ট্রিংটার সাফিক্স বলে, যেমন blog এর সাফিক্স g,og,log,blog। আর প্রতিটা substring ই কিন্তু কোনো না কোনো সাফিক্স এর প্রিফিক্স!! তাই সবগুলো সাফিক্সকে ট্রাইতে ইনসার্ট করলে কাজটা সহজ হয়ে যায়!)

৫. (অ্যাডভান্সড) সম্ভবত ২০১১তে ডেফোডিল ইউনিভার্সিটির ন্যাশনাল কনটেস্টে এসেছিলো প্রবলেমটা। একটা ডিকশনারি ইনপুট দেয়া থাকবে। প্রতিবার ডিকশনারির ২টা শব্দ কুয়েরি দিবে, বলতে হবে তাদের মধ্যে common prefix এর দৈর্ঘ্য কত। যেমন algo আর algea এর কমন প্রিফিক্স alg, দৈর্ঘ্য ৩। ট্রাইতে ডিকশনারিতে ইনসার্ট করে প্রতি কুয়েরিতে শব্দদুটি এন্ড-মার্ক থেকে LCA(lowest common ancestor) বের করে প্রবলেমটা সলভ করা যায়।

কিছু প্রবলেম:
UVA 10226
(UVA 11362 Phonebook)
UVA 11488 Hyper prefix sets
POJ 2001 Shortest Prefix
POJ 1056

হ্যাপি কোডিং!