

শাফায়েতের ব্লগ

প্রোগ্রামিং, অ্যালগরিদম, ব্যাকগ্র্যান্ড ইঞ্জিনিয়ারিং

[Home](#)

অ্যালগরিদম নিয়ে যত লেখা!

আমার সম্পর্কে...

স্ট্রিং ম্যাচিং: নুথ-মরিসন-প্র্যাট (কেএমপি) অ্যালগরিদম

জুন ১১, ২০১৮ by [Shafaet Ashraf](#)

আজকে আমরা শিখবো কেএমপি (KMP) অ্যালগরিদম ব্যবহার করে স্ট্রিং ম্যাচিং করা। কেএমপি শব্দটি এসেছে ৩জন কম্পিউটার বিজ্ঞানী Donald Knuth, James H. Morris এবং Vaughan Pratt এর নাম থেকে।

আমাদের প্রবলেম হলো একটা দুটি স্ট্রিং text এবং pattern দেয়া আছে, আমাদের বলতে হবে text এর ভিতর pattern স্ট্রিংটি সাবস্ট্রিং হিসাবে আছি কিনা। যেমন ধরো টেক্স্টটি হলো "MOD", এই স্ট্রিংটার খোঝার পদ্ধতি আছে "M", "O", "D", "MO", "OD" এবং "MOD", এখন যদি pattern = "MO" খুজতে বলে আমরা true রিটুর্ন করবো।

ক্রটফোর্স অ্যালগরিদম ব্যবহার করে স্ট্রিং ম্যাচিং করার টাইম কমপ্লেক্সিটি $O(n * m)$, যেখানে n এবং m হলো টেক্স্ট ও প্যাটার্নের দৈর্ঘ্য। কিন্তু কেএমপি ব্যবহার করে $O(n + m)$ কমপ্লেক্সিটিতে প্যাটার্ন খুজে বের করা যায়।

আমরা প্রথমে ক্রটফোর্স অ্যালগরিদমটা শিখবো এবং শিখতে গিয়ে দেখবো যে আমরা কিছু কাজ বারবার করছি যেটা একটু বুদ্ধিমানের মত চিন্তা করলে করার দরকার নেই। সেখান থেকে আমরা কেএমপি অ্যালগরিদম শিখবো। খাতা-কলম বা হোয়াইট-বোর্ড ছাড়া কেএমপি অ্যালগরিদম ব্যাখ্যা করা আমার জন্য একটু কষ্টকর, তাও আমি চেষ্টা করছি, আশা করি লেখাটা দ্রুত পড়ে ফেলার চেষ্টা না করে ধীরে ধীরে মনযোগ দিয়ে পড়বে।

মনে করো আমাদের টেক্স্ট হলো "abababacd" এবং প্যাটার্ন হলো "ababac"। ক্রটফোর্স অ্যালগরিদমে আমরা টেক্স্টের প্রতিটা ইনডেক্সে গিয়ে সেখান থেকে লুপ চালিয়ে প্যাটার্ন খুজে বের করার চেষ্টা করি।

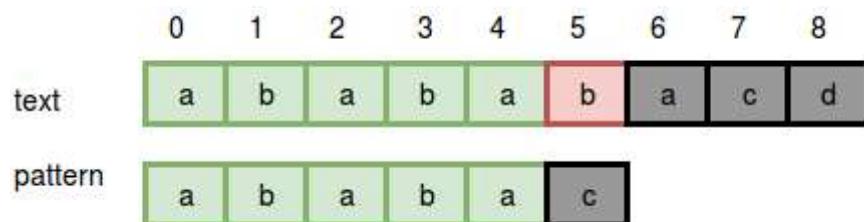
```
bool naive_matching(string text, string pattern) {
    int n = text.size();
```

```

int m = pattern.size();
for(int i = 0; i < n; i++) {
    //for each position i in text, we will try
    //to match text[i, i+1, ..., i+m-1] with pattern[0, 1, ... m-1]
    int j = 0;
    for(j = 0; j < m && i + j < n; j++) {
        if(text[i + j] != pattern[j]) {
            break; // mismatch found, break the inner loop
        }
    }
    if(j == m) {
        return true;
    }
}
return false;
}

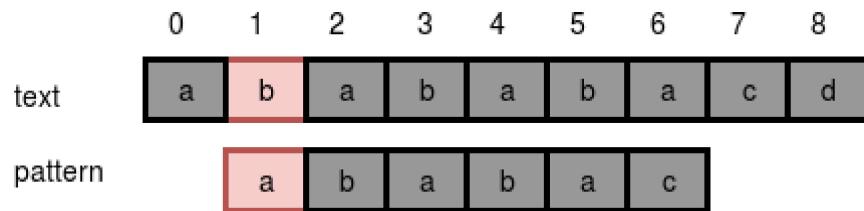
```

আমরা এই কোডটাকে সিমুলেট করার চেষ্টা করি। শুরুতে $i = 0$, আমরা টেক্স্টের প্রথম ক্যারেক্টার থেকে শুরু করে প্যাটার্ন এবং টেক্স্টের একটি একটি করে ক্যারেক্টার মিলাতে থাকবো। যদি সবগুলো ক্যারেক্টার মিলে যায় তাহলে আমরা প্যাটার্ন পেয়ে গেছি। যদি কোনো এক জায়গায় গিয়ে না মিলে তাহলে লুপ ব্ৰেক করে দিবো। নিচের ছবি দেখ:



চিত্র ১.০

5 নম্বর ক্যারেক্টারে গিয়ে আমরা একটি মিসম্যাচ পেয়েছি। ব্রুট ফোর্স অ্যালগরিদমের ভিতরের লুপটা ৬ নম্বর লাইনে ব্ৰেক কৰবে, এৱপৰ ইনডেক্স $i = 1$ এ গিয়ে আবাৰ খুজতে থাকবে।



চিত্র ১.২

এভাবে text এর প্রতিটি ইনডেক্সে গিয়ে লুপ চালিয়ে প্যাটার্ন খুজতে হবে, এজন্য এই অ্যাপ্রোচের টাইম কমপ্লেক্সিটি $O(n * m)$ । কোনোভাবে কি আমরা প্রতিটা ইনডেক্সে গিয়ে লুপ চালানো এড়াতে পাৰি? তাৰ আগে আমাদেৱ জানতে হবে সাফিক্স এবং প্রিফিক্স কি।



top

প্রিফিক্স: একটা স্ট্রিং শেষ থেকে শূন্য বা তার বেশি সংখ্যক ক্যারেক্টার ফেলে দিলে যা বাকি থাকে সেটাই একটা স্ট্রিং এর প্রিফিক্স। যেমন "ABC" স্ট্রিং টার প্রিফিক্স হলো "A", "AB" এবং "ABC"। এর মধ্যে "A" এবং "AB" হলো **প্রোপার (Proper)** প্রিফিক্স কারণ এগুলো মূল স্ট্রিংটার সমান না।

সাফিক্স: একটা স্ট্রিং শুরু থেকে শূন্য বা তার বেশি সংখ্যক ক্যারেক্টার ফেলে দিলে যা বাকি থাকে সেটাই একটা স্ট্রিং এর সাফিক্স। যেমন "ABC" স্ট্রিং টার সাফিক্স হলো "C", "BC" এবং "ABC"। এর মধ্যে "C" এবং "BC" হলো **প্রোপার (Proper) সাফিক্স** কারণ এগুলো মূল স্ট্রিংটার সমান না।

মনে করো আমরা যে প্যাটার্নটা খুজছি সেটা হলো abxyabcd। এবার নিচের (চিত্র ১.৩) ছবিটা দেখ:

????	A	B	X	Y	A	B	?	?	?	????
A	B	X	Y	A	B	C	D			

চিত্র ১.৩

ছবিতে (চিত্র ১.৩) আমরা টেক্স্টের সাথে প্যাটার্ন মিলাতে মিলাতে এক জায়গায় মিসম্যাচ পেয়েছি। কোন মিসম্যাচ হওয়া ক্যারেক্টারটা কি অথবা তার পরের ক্যারেক্টারগুলো কি সেটা নিয়ে আপাতত চিন্তা করা দরকার নাই। এখন আমরা যদি ব্র্যান্ডফোর্সের মতো প্যাটার্নকে একঘর বামে শিফট করে মিলাতে চেষ্টা করি তাহলে আদৌ কি কোন লাভ আছে?

????	A	B	X	Y	A	B	?	?	?	????
A	B	X	Y	A	B	C	D			

চিত্র ১.৪

১ ঘর যদি শিফট করি তাহলে প্রশ্নবোধক চিহ্নের জায়গাগুলোয় যাই থাকুক না কেন কোনো লাভ নাই। কত ঘর শিফট করলে লাভ হতেও পারে সেটা কিসের উপর নির্ভর করে? সেটা নির্ভর করে প্যাটার্নের যতটুকু প্রিফিক্স টেক্স্টের সাথে ম্যাচ করেছে সেটার উপর, এক্ষেত্রে সেই প্রিফিক্সটা হলো "ABXYAB"। আমরা যদি নিচের মত করে শিফট করি তাহলে ম্যাচ পেলেও পেতে পারি:

????	A	B	X	Y	A	B	?	?	?	?	?	?	????
A	B	X	Y	A	B	C	D						

চিত্র ১.৫

তারমানে আমাদেরকে প্যাটার্নটা এমনভাবে শিফট করতে হবে যাতে প্যাটার্নের প্রিফিক্সের সাথে প্যাটার্নেরই সাফিক্সের 'আংশিক' (partial) ম্যাচিং পাই। তাহলে যেটা ঘটবে, আমরা প্যাটার্নের প্রিফিক্সের সাথে ইনপুট টেক্স্টের partial ম্যাচ পাবো এবং এরপর আমরা আবার সামনে গিয়ে ক্যারেক্টার বাই ক্যারেক্টার মিলিয়ে দেখবো পুরো টেক্স্টা ম্যাচ কা নাকি।



top

আরেকটা উদাহরণ দেখলে পরিষ্কার হবে। মনে করো এবার প্যাটার্নটা হলো “ABABAC” এবং আমরা নিচের মতো আংশিক ম্যাচিং পেয়েছি:

????	A	B	A	B	A	?	?	?	?	?	?	?	?	?	????
	A	B	A	B	A	C									

চিত্র ১.৬

এবার আমরা কতটুকু শিফট করবো সেটা নির্ভর করবে ম্যাচ করা প্রিফিক্স “ABABA” এর উপর। নিচের ছবিটা দেখো:

????	A	B	A	B	A	?	?	?	?	?	?	?	?	?	?	????
	A	B	A	B	A	C										

চিত্র ১.৭

চিত্র ১.৭ এ প্যাটার্নটা ডানে ২ ঘর শিফট করেছি। এতে করে আমরা প্যাটার্নের প্রিফিক্সের সাথে প্যাটার্নেরই সাফিক্সের আংশিক ম্যাচ পাবো। এক্ষেত্রে প্যাটার্নের প্রথম ৩ ক্যারেক্টারের সাথে শেষ ৩ ক্যারেক্টার ম্যাচ করছে। তারমানে টেক্সটের সাথেও প্যাটার্নের প্রথম ৩ ক্যারেক্টার ম্যাচ করবে। এরপর আমরা আবার সামনে গিয়ে বাকি ক্যারেক্টারগুলো মিলিয়ে দেখবো।

এখন ধরো দৃঙ্গাগ্যক্রমে আমরা আবার মিসম্যাচ পেলাম:

????	A	B	A	B	A	?	?	?	?	?	?	?	?	?	?	????
	A	B	A	B	A	C										

চিত্র ১.৮

এখন কতখানি শিফট করবো? সেটা নির্ভর করে “ABA” এর উপর। আমাদেরকে এমনভাবে ABA কে এমনভাবে ডানে শিফট করতে হবে যেন শিফট করার পর ABA এর প্রিফিক্সের সাথে ABA এর সাফিক্স আংশিক ম্যাচ করে। এক্ষেত্রে চিত্র ১.৯ এর মত করে শিফট করতে হবে:

????	A	B	A	B	A	?	?	?	?	?	?	?	?	?	?	????
	A	B	A	B	A	C	D									

চিত্র ১.৯

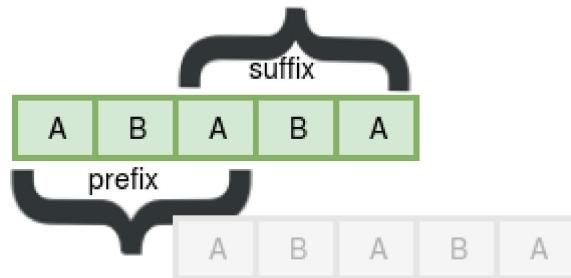
এতগুলো উদাহরণ দেয়ার উদ্দেশ্য একটা জিনিস পরিষ্কার করা, প্যাটার্নের কতখানি প্রিফিক্স টেক্সটের সাথে ম্যাচ করেছে তার উপর নির্ভর করবে প্যাটার্ন কয়েকটা শিফট করবো তার উপর নির্ভর করবে প্যাটার্ন কয়েকটা শিফট ক



top

মনে করো যতখানি প্রিফিক্স ম্যাচ করেছে সেই স্ট্রিংটুকুর নাম $P \mid P$ কতবর শিফট করলে আমরা “যে টেক্সটটুকু অলরেডি ম্যাচ করেছে তার সাফিক্সের সাথে” আংশিক ম্যাচ পাবো?

সেটা জানতে আমাদের বের করতে হবে P স্ট্রিংটার সবথেকে বড় প্রিফিক্স যেটা একই সাথে P এর একটা সাফিক্স ও। এই লাইনটা শুনলে আমার নিজেরই তালগোল পাকিয়ে যায়, তাই আসো আরেকটা ছবি দেখি:



Longest prefix that is also a suffix

চিত্র ১.১০

আশা করি ছবি দেখে পরিষ্কার হয়েছে লাইনটির মানে। সর্বোচ্চ কতখানি সাফিক্স প্রিফিক্সের সাথে মিলে যায় সেটা বের করে শিফট করতে হবে। “ABABAC” স্ট্রিং এর প্রিফিক্স আছে ৭টা:

length	prefix
0	""
1	A
2	AB
3	ABA
4	ABAB
5	ABABA
6	ABABAC

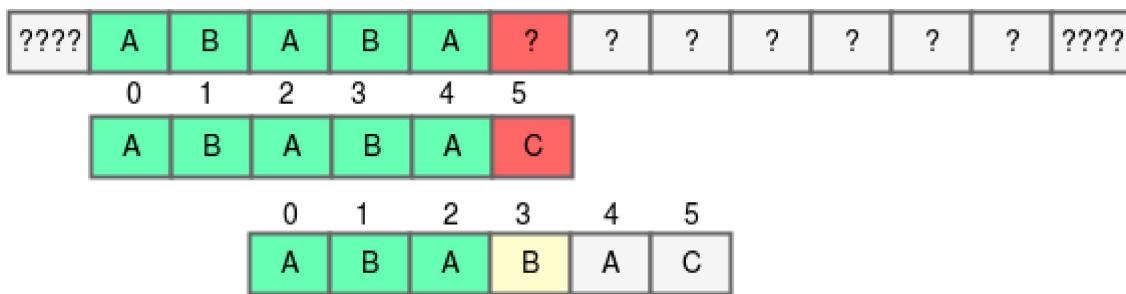
চিত্র ১.১১

এবার সবগুলো স্ট্রিং এর জন্য সবথেকে বড় প্রিফিক্সের দৈর্ঘ্য বের করবো যেটা একই সাথে একটা সাফিক্স। এক্ষেত্রে যেহেতু আমাদের আংশিক ম্যাচ দরকার, আমরা শুধুমাত্র প্রোপার সাফিক্স ও প্রিফিক্স নিয়ে চিন্তা করবো (অর্থাৎ সাফিক্স/প্রিফিক্সের দৈর্ঘ্য হবে স্ট্রিংটার থেকে কম)।

length	prefix	longest prefix length is also a suffix
0	""	0
1	A	0
2	AB	0
3	ABA	1
4	ABAB	2
5	ABABA	3
6	ABABAC	0

চিত্র ১.১২

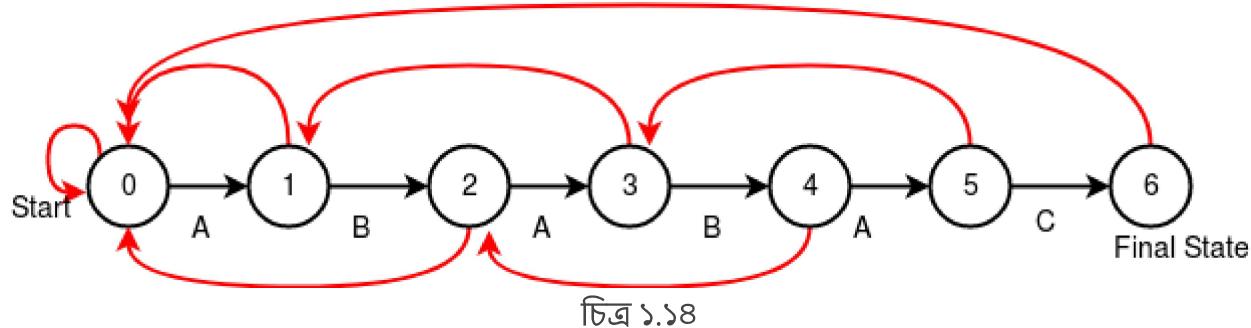
এই টেবিলটির একটা নাম আছে, এটাকে বলা হয় Failure table (ফেইলর টেবিল)। এই টেবিল দেখে আমরা বলে দিতে পারি প্যাটার্নের কতখানি প্রিফিক্স ম্যাচ করার পর মিসম্যাচ পাওয়া গেলে আবার কোথা থেকে ম্যাচিং শুরু করতে হবে।



চিত্র ১.১৩

যেমন উপরের ABABA পর্যন্ত ম্যাচ করার পর একটা ফেইলড ম্যাচ পেয়েছি। চিত্র ১.১২ এর টেবিলের ৬নম্বর রো থেকে পাই যে প্যাটার্নের প্রথম ৩ ক্যারেক্টার নিয়ে যে প্রিফিক্স হয় সেটা প্যাটার্নের শেষ ৩ক্যারেক্টার নিয়ে যে সাফিক্স হয় তার সমান। তারমানে প্যাটার্নের প্রথম ৩ ক্যারেক্টার “যে টেক্সটটুকু অলরেডি ম্যাচ করেছে” তার সাফিক্সের সমান। তাহলে আমরা প্যাটার্নের প্রথম ৩ক্যারেক্টার বাদ দিয়ে পরের ক্যারেক্টার থেকে আবার ম্যাচিং শুরু করবো।

তুমি যদি থিওরি অফ কম্পিউটেশনের কোনো কোর্স করে থাকো তাহলে বুঝতে পারছো ফেইলর টেবিলটা আসলে এক ধরনের “ফাইনাইট সেট অটোমাটা” (না পড়ে থাকলেও চিন্তার কিছু নেই)। অটোমেশন হলো একটা “সেট অফ সেট” (set of states) এবং এক সেট থেকে অন্য সেট এ কিভাবে যেতে হবে সেরকম কিছু নিয়ম। আমাদের ক্ষেত্রে সেট হলো প্যাটার্নের কতখানি ম্যাচ করেছে সেট। আর এক সেট থেকে অন্য সেটটো কিভাবে যাবো সেটা নির্ভর করে টেক্সট এবং প্যাটার্নের পরবর্তী ক্যারেক্টার ম্যাচ করছে নাকি তার উপর। যদি ম্যাচ করে তাহলেতো সহজ, পরের ক্যারেক্টারে গিয়ে আবার ম্যাচ করার চেষ্টা করবো। আর যদি ম্যাচ না করে তাহলে কতখানি সাফিক্স প্রিফিক্সের সাথে মিলে গিয়েছে (চিত্র ১.১০) সেটা বের করবো ফেইলর টেবিল দেখে। সহজে বোঝার জন্য আমরা ফেইলর টেবিলটাকে নিচের মত করে আকতে পারি:



চিত্র 1.18 এ আমরা ABABAC স্ট্রিং এর জন্য অটোমেশনটাকে দেখতে পাচ্ছি। Start স্টেট হলো যখন কোনো ক্যারেক্টার ম্যাচ করেনি। প্রতিবার একটা করে ক্যারেক্টার ম্যাচ করলে আমরা কালো এজ দিয়ে পরের স্টেটে যাবো, ফাইনাল স্টেটে চলে গেলে কাজ শেষ। লাল তীর চিহ্ন দিয়ে দেখানো হয়েছে মিসম্যাচ পেলে কোন স্টেট এ যাবো। চিত্র 1.12 এর টেবিলের সাথে মিলালে বুঝাবে ফেইলর টেবিল আর উপরের গ্রাফটি আসলে একই জিনিস বুঝাচ্ছে।

এখন আমাদের হাতে ২টি সমস্যা, প্রথমটা হলো ফেইলর টেবিলটা তৈরি করা, ২য়টি হলো সেটা ব্যবহার করে ম্যাচিং করা।

প্রথমে আমরা ফেইলর টেবিলটা তৈরি করি। আমরা এমন একটি ফাংশন লিখবো যেটা *failure[]* নামের m সাইজের একটা অ্যারে তৈরি করবে, “ABABAC” স্ট্রিং এর জন্য অ্যারেটা হবে এরকম:

```

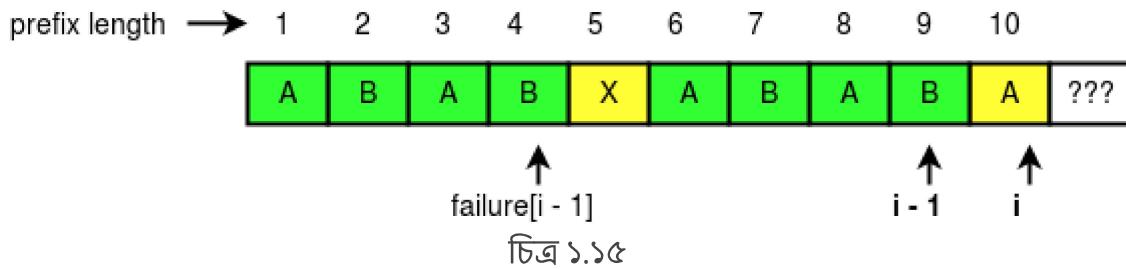
failure[0] = 0
failure[1] = 0
failure[2] = 0
failure[3] = 1
failure[4] = 2
failure[5] = 3
failure[6] = 0

```

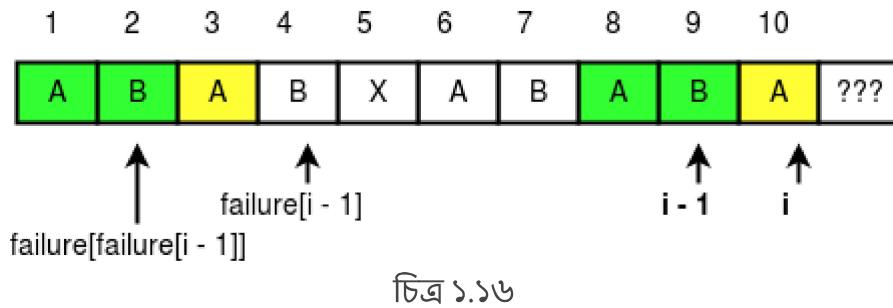
এখানে ইনডেক্সিং নিয়ে একটু সাবধানে থাকতে হবে। *failure[i]* দিয়ে আমরা স্ট্রিং এর i নম্বর ইনডেক্সের কথা বুঝাচ্ছি না, বরং i দৈর্ঘ্যের প্রিফিক্সের কথা বুঝাচ্ছি।

এখন মনে করো i দৈর্ঘ্যের প্রিফিক্সের জন্য *failure[i]* এর মান আমি জানি না কিন্তু $0 \leq i \leq \text{length}$ পর্যন্ত সব দৈর্ঘ্যের জন্য আমি *failure[length]* এর মান আগেই কোনোভাবে বের করে ফেলেছি। এখন আমি দেখবো $i-1$ দৈর্ঘ্যের স্ট্রিং এর জন্য কতখানি ম্যাচ পেয়েছি এবং বর্তমানে যে ক্যারেক্টারে আছি সেটা ব্যবহার করে ম্যাচটাকে লম্বা করা যায় নাকি।

চিত্র 1.15 এ একটা উদাহরণ দেখানো হয়েছে:



চিত্র ১.১৫ এ আমরা $failure[10]$ এর মান বের করার চেষ্টা করছি। আমরা আগের ক্যারেক্টারের গিয়ে $failure[i-1]$ এর মান দেখে বুঝবো কতটুকু প্রিফিক্স-সাফিক্স অলরেডি ম্যাচ করেছে। এরপর চেষ্টা করবো পরের ক্যারেক্টারের সাথে বর্তমান ক্যারেক্টারকে মিলানোর (চিত্র ১.১৫ এ হলুদ রঙ এর ঘর)। যদি মিলে যেত তাহলে $failure[10]$ এর মান হতো $failure[9] + 1 = 4 + 1 = 5$ । কিন্তু যেহেতু মিলেনি, আমরা পরবর্তি সেরা ম্যাচিং এ চলে যাবো এবং সেটা পাবো $failure[failure[i-1]]$ এ:



এবার হলুদ ঘরের ক্যারেক্টার দুটো মিলে গেছে, তারমানে $i = 10$ এর জন্য "ABA" সাফিক্স এবং প্রিফিক্স ম্যাচ করে যার দৈর্ঘ্য 3, তাই $failure[10]$ এর মান হবে 3।

আমাদের বেস কেস হবে, $failure[0] = failure[1] = 0$ । বাকি $2 \leq i \leq length$ এর জন্য $failure[i]$ মান আমরা আগেরগুলো দেখে দেখে বের করতে পারবো। নিচের কোডটা দেখো:

```

1 //https://www.topcoder.com/community/data-science/data-science-tutorials/introduction-to-string-
2 #define MAX 100000
3 int failure[MAX];
4
5 void build_failure_function(string pattern, int m) {
6     failure[0] = 0
7     failure[1] = 0; //base case
8
9     for(int i = 2; i <= m; i++) { //i is length of the prefix we are dealing with
10        // j is the index of the largest next partial match
11        // (the largest suffix/prefix) of the string under index i - 1
12        int j = failure[i - 1];
13        while(true) {
14            // check if the last character of prefix of length i "expands" the current "candidate"
15            if(pattern[j] == pattern[i - 1]) {
16                failure[i] = j + 1;
17                break;
18            }
19            // if we cannot "expand" even the empty string
20            if(j == 0) {
21                failure[i] = 0;
22                break;
23        }
24    }
25}
```

```

23     }
24     // else go to the next best "candidate" partial match
25     j = failure[j];
26   }
27 }
28 }
```

ফেইলর টেবিল জেনারেট করা হয়ে গেলে আমরা স্ট্রিং ম্যাচিং শুরু করতে পারি। কোড লেখার আগে তুমি একবার অটোমেশন গ্রাফটা একে হাতে কলমে সিমুলেট করো। আমাদের দুইটা পয়েন্টার থাকবে i এবং j । i দিয়ে বুঝাবো আমরা অটোমেশনের কোন স্টেট এ আছি, অর্থাৎ কতখানি প্রিফিক্স অলরেডি ম্যাচ করেছে এবং j দিয়ে বুঝাবো টেক্সটের কোন ক্যারেক্টারটার সাথে ম্যাচ করছি। যদি $text[j] == pattern[i]$ হয় তাহলে আমরা i এবং j এর মান ইনক্রিমেন্ট করে আবার ম্যাচ করার চেষ্টা করবো, অর্থাৎ অটোমেশনের কালো তীরচিহ্ন ধরে আগবো (চিত্র ১.১৪)। কিন্তু যদি $text[j]! = pattern[i]$ হয় তাহলে আমরা ফেইলর টেবিল ব্যবহার করে যতটুকু সাফিক্স-প্রিফিক্স ম্যাচ করেছে সেখান থেকে আবার ম্যাচিং শুরু করবো, অর্থাৎ $i = failure[i]$ হয়ে যাবে। আর যদি দেখি $i = 0$ হয়ে গেছে কিন্তু আমরা এখনোও ম্যাচ করতে পারছি না তাহলে j এর মান ইনক্রিমেন্ট করে টেক্সটের পরের ক্যারেক্টার থেকে আবার ম্যাচিং শুরু করতে হবে।

```

1 bool kmp(string text, string pattern)
2 {
3     int n = text.size();
4     int m = pattern.size();
5     build_failure_function(pattern, m);
6
7     int i = 0; // the initial state of the automaton is
8             // the empty string
9
10    int j = 0; // the first character of the text
11
12    while(true) {
13        if(j == n) {
14            return false; //reached the end of the text
15        }
16
17        // character matched
18        if(text[j] == pattern[i]) {
19            i++; // change the state of the automaton
20            j++; // get the next character from the text
21            if(i == m) {
22                return true;
23            }
24        } else {
25            if (i == 0) {
26                // if we reached the empty string and failed to
27                // "expand" even it; we go to the next
28                // character from the text, the state of the
29                // automaton remains zero
30                j++;
31            }
32            else {
33                //we try to "expand" the next best (largest) match
34                i = failure[i];
35            }
36        }
37    }
}
```

```

38     return false;
39 }
```

কেএমপি অ্যালগরিদমের টাইম কমপ্লেক্সিটি $O(n + m)$ । লিনিয়ার টাইম স্ট্রিং ম্যাচিং করার আরেকটা অ্যালগরিদম **রবিন-কার্প** নিয়ে আগে লিখেছিলাম। কিন্তু হ্যাশ কলিশনের জন্য রবিন-কার্পের পারফরমেন্স অনেক ক্ষেত্রেই খারাপ হয়ে যেতে পারে, বেশিভাগ ক্ষেত্রেই কেএমপি ব্যবহার করে ম্যাচিং করা সুবিধাজনক।

প্রোগ্রামিং কনটেস্টে সরাসরি কেএমপি ব্যবহার করে সমাধান করতে হয় এমন সমস্যা খুব বেশি পাবে না, কিন্তু ফেইলের ফাংশনের প্রোপার্টি ব্যবহার করে সমাধান করতে হয় এমন সমস্যা প্রায়ই পাওয়া যায়। পরবর্তি কোনো একটা লেখায় সেরকম কিছু সমস্যা নিয়ে আলোচনা করবো। আপাতত এই পর্যন্তই, হ্যাপি কোডিং!

প্র্যাকটিস প্রবলেম:

http://www.lightoj.com/volume_showproblem.php?problem=1255

http://www.lightoj.com/volume_showproblem.php?problem=1258

রিসোর্স:

<https://www.topcoder.com/community/data-science/data-science-tutorials/introduction-to-string-searching-algorithms/>

ফেসবুকে মন্তব্য

0 comments

top

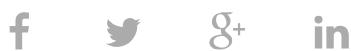
0 comments

Sort by **Oldest**

Add a comment...



Facebook Comments plugin

Powered by **Facebook Comments**

► Posted in অ্যালগরিদম/প্রবলেম সলভিং, প্রোগ্রামিং ? Tagged কেএমপি, স্ট্রিং, স্ট্রিং ম্যাচিং

27,305 times read (excluding bots)

◀ অয়লার ট্যুর (ফ্লিরি এবং হেয়ারহজলার অ্যালগরিদম)

ট্রাভেলোকা এবং আমার সফটওয়্যার ইঞ্জিনিয়ারিং এ
হাতেখাড়ি ►

Leave a Reply

Connect with:

Secured by [OneAll Social Login](#)

Your email address will not be published. Required fields are marked *

Comment



Name *

top

Email *

Website

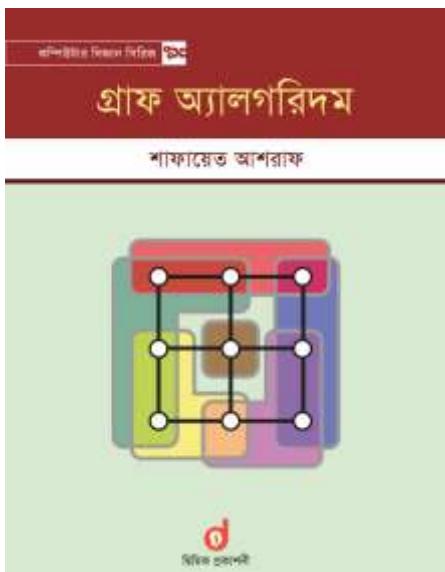
Save my name,
email, and
website in this
browser for the
next time I
comment.

[Post Comment](#)

phonetic probhat english

আমার সম্পর্কে

শাফায়েত, সফটওয়্যার ইঞ্জিনিয়ার @ **Meta** (বিস্তারিত...)





ফায়েত আশর

Follow Page

Contact Us

প্রোগ্রামিং কনটেস্ট এবং অ্যালগরিদম

অনুপ্রেরণা(৪):

কেন আমি প্রোগ্রামিং শিখবো?

কম্পিউটার বিজ্ঞান কেন পড়বো?

প্রোগ্রামিং কনটেস্ট এবং অনলাইন জাজে হাতেখড়ি

কনফিউজড প্রোগ্রামার

ব্যাকএন্ড ইঞ্জিনিয়ারিং:

ডিস্ট্রিবিউটেড ফাইল সিস্টেম

ব্লুম ফিল্টার

লগ-স্ট্রাকচার্ড-ট্রি

কনসিস্টেন্ট হ্যাশিং(নতুন)

ক্যাশিং অ্যালগরিদম: এল-আর-ইউ ক্যাশ

অ্যালগরিদম বেসিক(৬):

বিগ "O" নোটেশন

কমপ্লেক্সিটি ক্লাস(P-NP, টুরিং মেশিন ইত্যাদি)

হাল্টিং প্রবলেম

বাইনারি সার্চ - ১

বাইনারি সার্চ - ২(বাইসেকশন)

ফ্লয়েড সাইকেল ফাইন্ডিং অ্যালগরিদম

ডাটা স্ট্রাকচার(১৩):

লিংকড লিস্ট

স্ট্যাক

কিউ+সার্কুলার কিউ

প্লাইডিং রেঞ্জ মিনিমাম কুয়েরি (ডিকিউ)

ডিসজয়েন্ট সেট(ইউনিয়ন ফাইন্ড)

ট্রাই(প্রিফিক্স ট্রি/রেডিক্স ট্রি)

সেগমেন্ট ট্রি-১

সেগমেন্ট ট্রি-২(লেজি প্রপাগেশন)



top

অ্যারে কমপ্রেশন/ম্যাপিং
 লোয়েস্ট কমন অ্যানসেস্টর
 বাইনারি ইনডেক্সড ট্রি
 স্কয়ার-র্ভট ডিকম্পোজিশন

গ্রাফ থিওরি(২০):
 গ্রাফ থিওরিতে হাতেখড়ি

অ্যাডজেসেন্সি ম্যাট্রিক্স
 অ্যাডজেসেন্সি লিস্ট

ব্রেথড ফার্স্ট সার্চ (বিএফএস)

মিনিমাম স্প্যানিং ট্রি ১ (প্রিমস অ্যালগোরিদম)
 মিনিমাম স্প্যানিং ট্রি ২ (ক্ৰুসকাল অ্যালগোরিদম)

টপোলজিকাল সর্ট
 ডেপথ ফার্স্ট সার্চ এবং আবারো টপোলজিকাল সর্ট

ডায়াক্রস্ট্রা
 ফ্লয়েড ওয়ার্শন

বেলম্যান ফোর্ড
 আটিকুলেশন পয়েন্ট এবং ব্রিজ

স্ট্রংলি কানেক্টেড কম্পোনেন্ট
 ম্যাক্সিমাম ফ্লো-১

ম্যাক্সিমাম ফ্লো-২
 স্টেবল ম্যারেজ প্রবলেম

অয়লার টুয়ার ট্রি এবং ডায়ামিটার নির্ণয়
 লংগেস্ট পাথ প্রবলেম

অ্যালগরিদম গেম থিওরি(৩):

গেম থিওরি-১
 গেম থিওরি-২
 গেম থিওরি-৩

ডাইনামিক প্রোগ্রামিং:

ডাইনামিক প্রোগ্রামিং ১ (ফিবোনাচ্চি)
 ডাইনামিক প্রোগ্রামিং ২ (শর্টেস্ট পাথ)
 ডাইনামিক প্রোগ্রামিং ৩ (LIS এবং পাথ প্রিন্টিং)
 ডাইনামিক প্রোগ্রামিং ৪ (লংগেস্ট কমন সাবসিকোয়েন্স)
 ডাইনামিক প্রোগ্রামিং ৫ (কয়েন চেঞ্চ/০-১ ন্যাপস্যাক)
 ডাইনামিক প্রোগ্রামিং ৬ (সাবসেট সাম, কম্বিনেটরিক্স, ডিসিশন প্রবলেম)



top

ডাইনামিক প্রোগ্রামিং ৭ (ম্যাট্রিক্স চেইন মাল্টিপ্লিকেশন)

ডাইনামিক প্রোগ্রামিং ৮ ((বিটমাস্ক, ট্রাভেলিং সেলসম্যান)

ডাইনামিক প্রোগ্রামিং ৯ (ট্রি, মিনিমাম ভারটেক্স কভার)

ব্যাকট্র্যাকিং:

ব্যকট্র্যাকিং বেসিক এবং পারমুটেশন জেনারেটর

নাস্তার থিওরি/গণিত:

মডুলার অ্যারিথমেটিক

প্রাইম জেনারেটর (Sieve of Eratosthenes)

বিটওয়াইজ সিভ

ডিরেঞ্জমেন্ট

প্রোবাবিলিটি: এক্সপেক্টেড ভ্যালু

স্ট্রিং ম্যাচিং(২):

রবিন-কাপ্র

কেএমপি (KMP)

অন্যান্য(৩) :

ডিরেকশন অ্যারে

মিট ইন দ্যা মিডল

টেইল-কল রিকার্শন অপটিমাইজেশন(নতুন)

কোয়ান্টাম কম্পিউটার

কোয়ান্টাম কম্পিউটার কী?

কোয়ান্টাম কম্পিউটারের শক্তি এবং সীমাবদ্ধতা



English Blog



top

নতুন লেখা

[ডেটা স্ট্রাকচার: XOR লিংকড লিস্ট](#)

[ব্যাকএন্ড ইঞ্জিনিয়ারিং: ডিস্ট্রিবিউটেড ফাইল সিস্টেম](#)

[অ্যালগরিদম কমপ্লেক্সিটি\(বিগ “O” নোটেশন\)](#)

[ব্যাকএন্ড ইঞ্জিনিয়ারিং: কনসিস্টেন্ট হ্যাশিং](#)

[ব্যাকএন্ড ইঞ্জিনিয়ারিং: লগ-স্ট্রাকচার্ড-ট্রি](#)



This work is licensed under a [Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License](#).

Disclaimer: The advertisement shown in this site is automatically inserted by Google Adsense based on individual user's interest. It doesn't reflect the interest or ideology of the site owner.

AccessPress Staple | WordPress Theme: AccessPress Staple by AccessPress Themes

