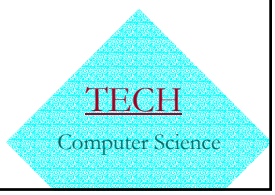
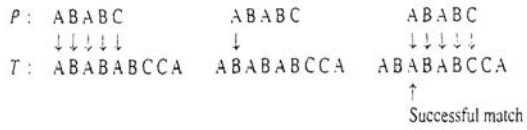
String Matching

- **detecting the occurrence of a particular substring (pattern) in another string (text)**
- A straightforward Solution
- The Knuth-Morris-Pratt Algorithm
- The Boyer-Moore Algorithm



Straightforward solution

- **Algorithm: Simple string matching**
- **Input**: P and T, the pattern and text strings; m, the length of P. The pattern is assumed to be nonempty.
- **Output**: The return value is the index in T where a copy of P begins, or -1 if no match for P is found.



int simpleScan(char[] P,char[] T,int m)

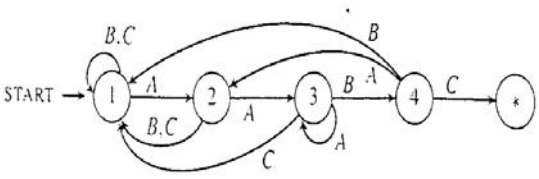
```
int match  //value to return.
int i,j,k;
match = -1;
j=1;k=1; i=j;
while(endText(T,j)==false)
 if( k>m )
   match = i; //match found.
   break;
 if(t_j == p_k)
     j++;  k++;
 else
   //Back up over matched characters.
   int backup=k-1;
   j = j-backup;
   k = k-backup;
   //Slide pattern forward,start over.
   j++;  i=j;
return match;
```

Analysis

- Worst-case complexity is in θ(mn)
- Need to back up.
- Works quite well on average for natural language.

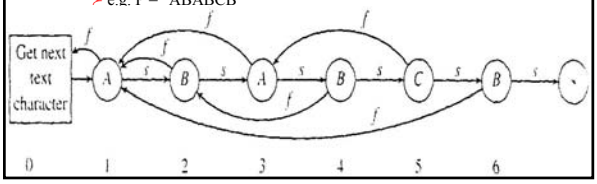
The Knuth-Morris-Pratt Algorithm

- Pattern Matching with Finite Automata
  - **e.g. P = "AABC"**



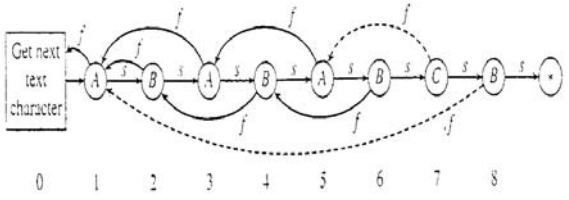
The Knuth-Morris-Pratt Flowchart

- Character labels are inside the nodes
- Each node has two arrows out to other nodes: success link, or fail link
- next character is read only after a success link
- A special node, node 0, called "get next char" which read in next text character.
  - e.g. P = "ABABCB"



Construction of the KMP Flowchart

- **Definition:Fail links**
  - **We define fail[k] as the largest r (with r<k) such that $p_{1,..}p_{r-1}$ matches $p_{k-r+1}...p_{k-1}$.That is the (r-1) character prefix of P is identical to the one (r-1) character substring ending at index k-1. Thus the fail links are determined by repetition within P itself.**



Algorithm: KMP flowchart construction

- **Input**: P,a string of characters;m,the length of P.
- **Output**: fail,the array of failure links,defined for indexes 1,...,m.The array is passed in and the algorithm fills it.
- **Step:**
- void kmpSetup(char[] P, int m, int[] fail)
- int k,s
- 1. fail[1]=0;
- 2. for(k=2;k<=m;k++)
- 3. s=fail[k-1];
- 4. while(s>=1)
- 5. if(p_s==p_{k-1})
- 6. break;
- 7. s=fail[s];
- 8. fail[k]=s+1;

The Knuth-Morris-Pratt Scan Algorithm

- int kmpScan(char[] P,char[] T,int m,int[] fail)
- int match, j,k;
- match= -1;
- j=1; k=1;
- while(endText(T_j)==false)
- if(k>m)
- match = j-m;
- break;
- if(k==0)
- j++; k=1;
- else if(t_j==p_k)
- j++; k++;
- else
- //Follow fail arrow.
- k=fail[k];
- //continue loop.
- return match;

Analysis

- KMP Flowchart Construction require 2m – 3 character comparisons in the worst case
- The scan algorithm requires 2n character comparisons in the worst case
- Overall: Worst case complexity is θ(n+m)

The Boyer-Moore Algorithm

- The new idea
  - **first heuristic**
    - e.g. scan from right to left, jump forward …
  - **Find "`must`" in**
    - **`If you wish to understand you must…`**

- **`must`**
- **`1 1 1 1 1 111 1 1 1211`**
- **`If you wish to understand you must…`**

Algorithm:Computing Jumps for the Boyer-Morre Algorithm

- Input:Pattern string P:m the length of P;alphabet size alpha=|Σ|
- Output:Array charJump,defined on indexes 0,....,alpha-1.The array is passed in and the algorithm fills it.
- void computeJumps(char[] P,int m,int alpha,int[] charJump)
- char ch; int k;
- for (ch=0;ch<alpha;ch++)
- charJump[ch]=m;
- for (k=1;k<=m;k++)
- charJump[p_k]=m-k;

If you wish to understand you must

- ...