INSTITUTE OF INFORMATION TECHNOLOGY
University of Dhaka

Assignment on
Backtracking

Course: CSE 201

Submitted by :

Kamruzzaman Asif

Roll: 1217

Submitted to :

Dr. Sumon Ahmed

Associate Professor

IIT DU

Task -1:

Propose an algorithm that will print all the permutations of a given string. Explain the working methodology of your algorithm in detail.

Solution:

We can generate all the permutation of string using backtracking. An algorithm to print all the permutation of a string which can also handle duplicate characters in a string is given below:

Algorithm:

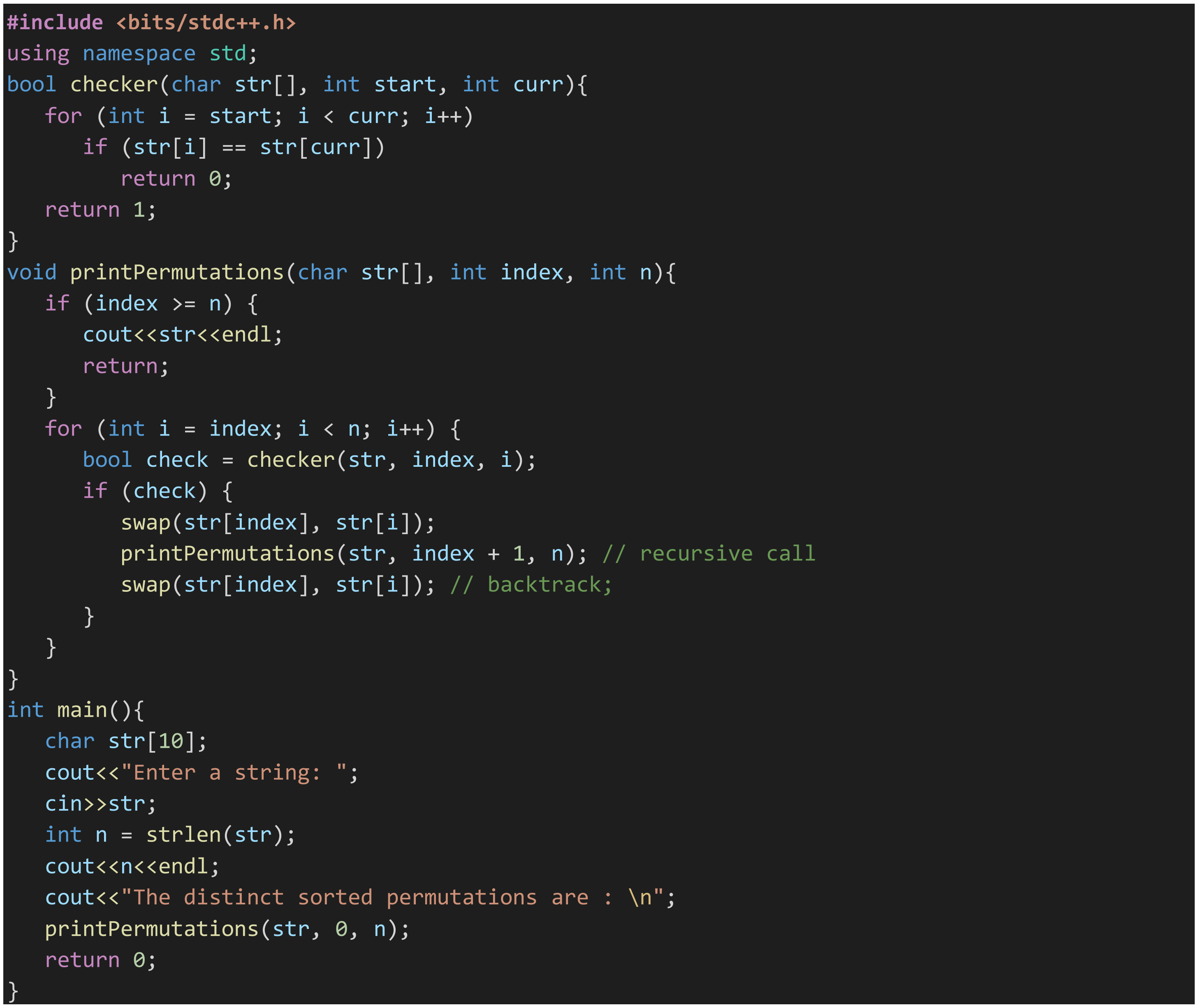
checker (str, start, current):

```
1.  for i = start upto current
2.      if (str[i] == str[current])
3.          return 0;
4.  return 1;
```

printPermutations (str, index, n):

```
1.  if ( index >= n)
2.      print str;
3.      return;
4.  else
5.      for i = index upto n
6.          bool check = checker (str, index, i);
7.          if (check)
8.              swap(str[index],str[i]);
9.              printPermutations (str, index+1, n);  // recursive call
10.             swap(str[index],str[i]);  // backtrack
```

Code:



```cpp
#include <bits/stdc++.h>
using namespace std;
bool checker(char str[], int start, int curr){
   for (int i = start; i < curr; i++)
      if (str[i] == str[curr])
         return 0;
   return 1;
}
void printPermutations(char str[], int index, int n){
   if (index >= n) {
      cout<<str<<endl;
      return;
   }
   for (int i = index; i < n; i++) {
      bool check = checker(str, index, i);
      if (check) {
         swap(str[index], str[i]);
         printPermutations(str, index + 1, n); // recursive call
         swap(str[index], str[i]); // backtrack;
      }
   }
}
int main(){
   char str[10];
   cout<<"Enter a string: ";
   cin>>str;
   int n = strlen(str);
   cout<<n<<endl;
   cout<<"The distinct sorted permutations are : \n";
   printPermutations(str, 0, n);
   return 0;
}
```

Now let’s explain the working methodology of the above algorithm:

In a short, the printpermutations function prints the permutations of the given string and the checker function checks for swaping duplicate characters. Inside the printpermutations function we used the recursive call to this function and backtracking. We will now understand the working method more clearly by the given examples of two cases.

Case 1: No duplicate characters present in string

Index = 0

Start = 0 end =2

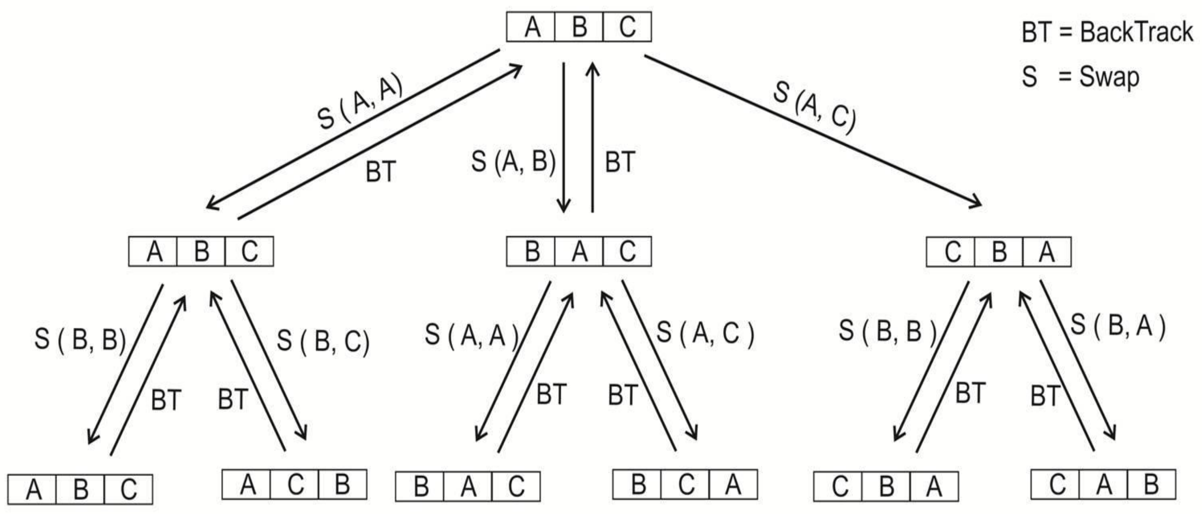
Index = 1

Start = 1 end = 2

Index = 2

Start = 2 end = 2

Gives output



Backtrack Tree of Permutation

Case 2: duplicate characters present in string

Index = 0

Start = 0 end =2

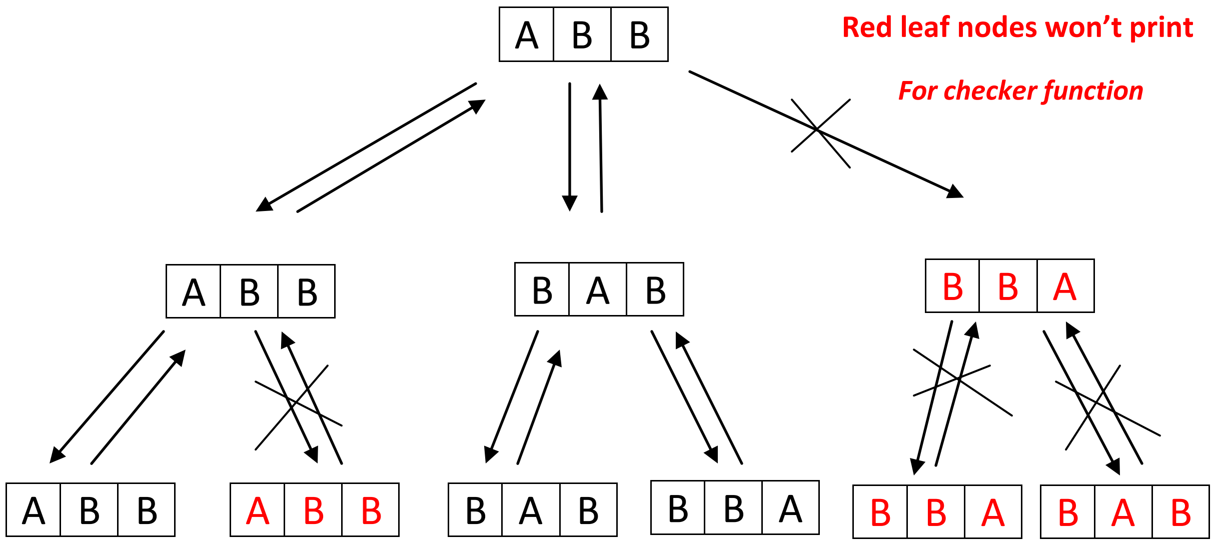
Index = 1

Start = 1 end = 2

Index = 2

Start = 2 end = 2

Gives output



Backtrack Tree of Permutation

Firstly, in the printpermutaions function we check if the current index is greater or equal of the string size, if so then we print out the string or go to the for loop. The for loop starts from the current index and runs until the end(size of string). We have a checker function here which checks for duplicates characters for swaping if duplicates found then the loop continues else the if statements executes. In the if statements first swap the two chars of index and current index of loop then a recursive call with increasing index by one and then again swap(backtrack).

Task -2:

What modifications do you need to do in the above-proposed algorithm to handle combinations if the number of selected items k is given.

Solution:

To print all the combination of a string with k selected items we can also use backtracking and recursion. The below algorithm prints the combination of a string (note: this algorithm can also handle duplicate characters):

<u>Algorithm:</u>

sort():

at first we sort the string.

printCombination(char str[], int index): // call by an empty string and index 0

```
1.  declaring a character arry a and an integer variable k
2.  if ((k=strlen(str))<key) follow steps 3 to 14  else print str
    // key- number of selected items k
3.          strcpy(a,str);
4.          for int i=index upto strlen(s) follow step 5 to 14    // s main input string
5.                  int f = 0;
6.                  for int j=index upto i     repeat steps 6 to 7
7.                          if(s[i]==s[j])
8.                                  f=1;
9.                  if(f==1)     go through steps 10 & 11
10.                         i++;
11.                         continue;
12.                 a[k] = s[i++];
13.                 a[k+1] = '\0';
14.                 printCombination(a,i);
```

Code:

```cpp
#include <bits/stdc++.h>
using namespace std;

char s[20];
int key;

void printCombination(char str[],int index)
{
    char a[20];
    int k;
    if((k=strlen(str))<key){     // checking size of string
            strcpy(a,str);
        for(int i=index;i<strlen(s); ){      // loop to backtrack
                int f=0;
            for(int j=index;j<i;j++){         // checking repeated characters
                if(s[i]==s[j])
                    f=1;
            }
            if(f==1){
                i++;
                continue;       // if repeated cahr found then continue
            }

            a[k]=s[i++];
            a[k+1]='\0';
            printCombination(a,i);    // recursive call
        }
    }
    else
        cout<<str<<endl;
}


int main()
{
    cout << "Enter String: " << endl;
    cin>>s;
    cout<<"\nEnter combination key: "<<endl;
    cin>>key;
    sort(s,s+strlen(s));
    char temp[20]={"\0"};
    printCombination(temp, 0);

    return 0;
}
```

Now let's explain working methodology:

We sort the string first which helps to handle duplicate characters. Then we call the printCombination function with an empty string and index 0. We set tree root to the first character of string and make tree with it's following distinct characters then print the sub-trees length equal to key. Then we backtrack to main string and check for duplicates if so then go to next char else make tree setting this char as root and then print the sub-trees length equal to key. This process continues.

Case-1: No duplicate character present

main string = ABC   key = 2

For every <u>distinct roots</u> tree made with <u>it's following distinct characters</u>

Print all the sub-tree <u>length equal to key</u>

root -> A

$$\begin{tikzcd}
& A \arrow[dl] \arrow[dr] & \\
B & & C
\end{tikzcd}$$

Output: AB   AC

root -> B

$$\begin{tikzcd}
B \arrow[d] \\
C
\end{tikzcd}$$

BC

root -> C

$$\begin{tikzcd}
C
\end{tikzcd}$$

none

Case-2: duplicate character present

main string = ABBCD   key = 3

For every <u>distinct roots</u> tree made with <u>it's following distinct characters</u>

Print all the sub-tree <u>length equal to key</u>

root -> A

$$\begin{tikzcd}
& & A \arrow[dl] \arrow[dr] & \\
& B \arrow[dl] \arrow[d] \arrow[dr] & & C \arrow[d] \\
B & C & D & D
\end{tikzcd}$$

Output: ABB ABC ABD ACD

root -> B

$$\begin{tikzcd}
& B \arrow[dl] \arrow[dr] & \\
B \arrow[d] \arrow[dr] & & C \arrow[d] \\
C & D & D
\end{tikzcd}$$

BBC BBD BCD

root -> C

$$\begin{tikzcd}
C \arrow[d] \\
D
\end{tikzcd}$$

none

root -> D

$$\begin{tikzcd}
D
\end{tikzcd}$$

none