Recursive Function Reminder

Arash Rafiey

August 25, 2016

All subsets of a set of size n

```
# include < iostream >
using namespace std;
void Allsubset( int *Array, int i, int n) {
if (i==n) {
  cout<<" new  subset =";
  for (int j=0; j < n; j++)
   if (Array[j] ==1)
     cout<< j + 1 <<",";
  cout<< endl;
  return;
}

Array[i]=0;
Allsubset(Array,i+1,n);
Array[i]=1;
Allsubset(Array,i+1,n);
}
```

The Needleman Wunsch algorithm is an algorithm used in bioinformatics to align protein or nucleotide sequences.

The Needleman Wunsch algorithm is an algorithm used in bioinformatics to align protein or nucleotide sequences.

It was one of the first applications of dynamic programming to compare biological sequences.

The Needleman Wunsch algorithm is an algorithm used in bioinformatics to align protein or nucleotide sequences.

It was one of the first applications of dynamic programming to compare biological sequences.

The algorithm was developed by Saul B. Needleman and Christian D. Wunsch and published in 1970.

```
int main() {
cout<<" hello" << endl;
int n;
cout<<" enter a number";
cin >> n;
int *Array=new int [n];
Allsubset(Array,0,n);
cout <<" to exit enter a number";
int t;
cin >> t;
}
```

all m-subsets of an n-set

```
# include < iostream >
using namespace std;
void msubset(int *Array, int i, int n, int m) {
if (m==0) {
  cout<<" new  subset =";
  for (int j=0; j < i; j++)
   if (Array[j] ==1)
     cout<< j + 1 <<",";
  cout<< endl;
  return;
}
if ( i > n − m) return; // not enough ones is Array
Array[i]=1;
msubset(Array,i+1,n,m-1);
Array[i]=0;
msubset(Array,i+1,n,m);
}
```

```
int main() {
cout<<" hello" << endl;
int n;
cout<<" enter a number for array size";
cin >> n;
int *Array=new int [n];
int m;
cout<< "enter a number for subset size";
cin >> m;
if (m < n)
  msubset(Array,0,n,m);
else cout<< "error" << endl;
cout<< endl;
cout <<" to exit enter a number";
int t;
cin >> t;
}
```

```
void permute(int *Array,int i, int n) {
if (i==n) {
   cout<<" new  permutation =";
   for (int j=0; j < n; j++)
     cout<< Array[j] <<",";
   cout<< endl;
   return;
}
else {
   int temp; int t;
   for (t=i; t < n; t++) {
     // exchange Array[i],Array[t]
     temp=Array[i];   Array[i]=Array[t];   Array[t]=temp;
     permute(Array,i+1,n);
     // exchange back Array[t],Array[i]
     temp=Array[i];   Array[i]=Array[t];   Array[t]=temp;
   } } }
```

```
int main() {
cout<<" hello" << endl;
int n;
cout<<" enter a number for array size";
cin >> n;
int *Array=new int [n];
int j;
for (j=0; j < n; j++) Array[j]=j+1;
permute(Array,0,n);
cout<< endl;
cout <<" to exit enter a number";
int t;
cin >> t;
}
```

Definition : We say a sequence S of $0, 1$ is **nice** if the number of ones and the number of zeros are the same and

in every prefix of S the number of ones is not less than the number of zero.

Problem : Write a program to print-out all the nice sequences of $0, 1$ with length n
$x + y^2 + \sqrt{z}$.

```
void nice-string( int *Array, int i, int difference, int n) {
if (i==n) {
  if ( difference == 0) {
    cout<<" new  string =";
    for (int j=0; j < n; j++)
      cout<< A[j] <<",";
    cout<< endl;
  }
  return;
}
if ( difference < 0 ) return;
if ( difference > n - i) return;
Array[i]=0;
nice-string (Array,i+1,difference-1,n);
Array[i]=1;
nice-string (Array,i+1,difference+1,n);
}
```

```
int main() {
cout<<" hello" << endl;
int n;
cout<<" enter an even number";
cin >> n;
int *Array=new int [n];
subset(Array,0,0,n);
cout<< endl;
cout <<" to exit enter a number";
int t;
cin >> t;
}
```

Ford-Fulkerson Max-Flow Algorithm

Max-Flow($(D = (V, E)$)

1. Define flow f for every edge e by setting $f(e) = 0$.
2. Repeat :
3. Apply BFS-f-augmenting to find an f-augmenting path p
4. Let $\Delta_p = \min_{e \in p} \Delta_e$
5. **for** each edge $e \in p$
6. **if** e is a forward edge
7. $f(e) := f(e) + \Delta_p$.
8. **else** (e is a backward edge)
9. $f(e) := f(e) - \Delta_p$.
10. Until no f-augmenting path p can be found.
11. Return f.