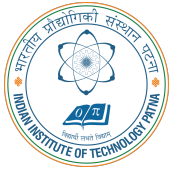
CS1101: Foundations of Programming

Examples on Recursion



Dept. of Computer Science & Engineering

Indian Institute of Technology Patna

Generation of permutation

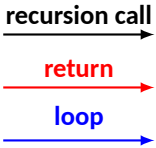
Generation of permutation

```
void permutation(int pending[], int n, int r, int used[], int uCount){
   if(uCount == r){
     for(int i=0; i<r; i++) printf("%d",used[i]);
     printf("\n");
   } else{
     for(int i=uCount; i<n; i++){
       swap(pending, i, uCount);
       used[uCount]=pending[uCount];
       permutation(pending, n, r, used, uCount+1);
       swap(pending, uCount, i);
     }
   }
}
```

```
void swap(int p[], int i, int j){
   int tmp;
   tmp = p[i];
   p[i] = p[j];
   p[j] = tmp;
}
```

```
int main(){
   int used[10], size=3, r=3, uCount=0;
   int pending[]={1,2,3};
   permutation(pending, size, r, used, uCount);
   return 0;
}
```

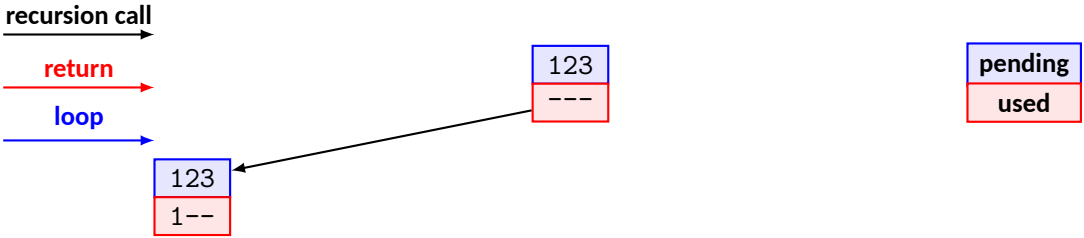
Generation of permutation



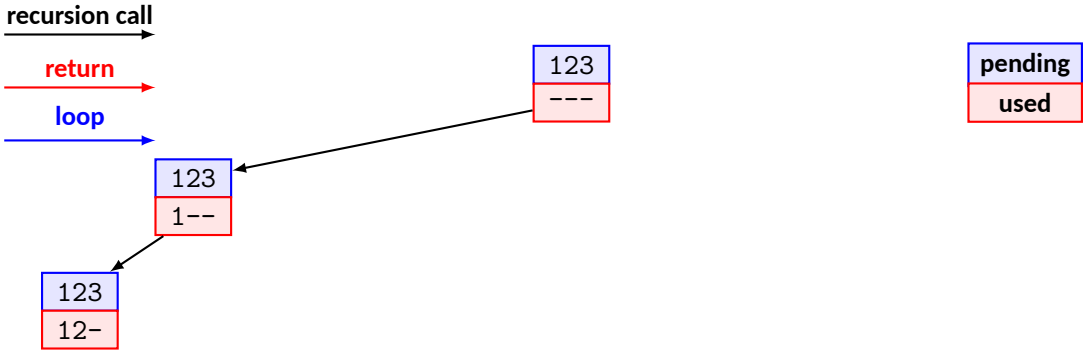
123

pending
used

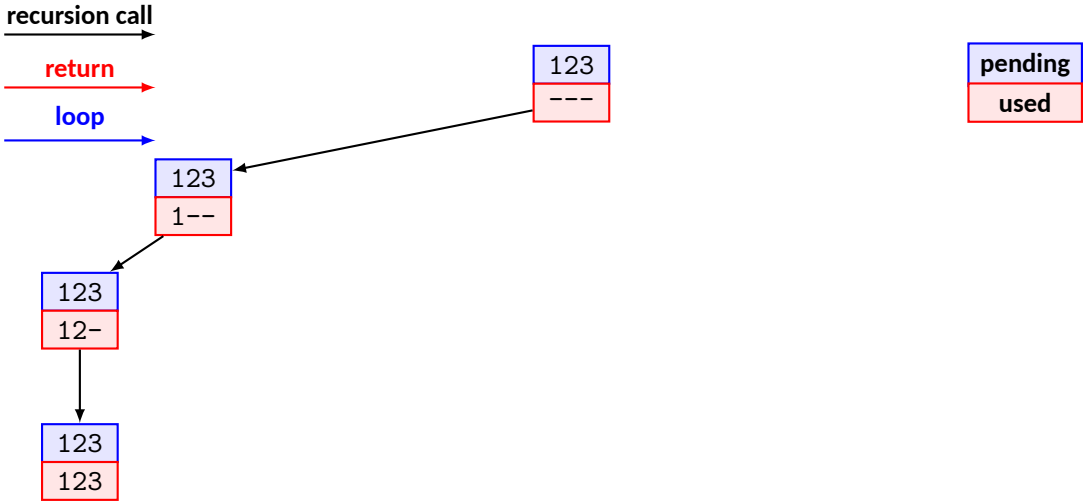
Generation of permutation



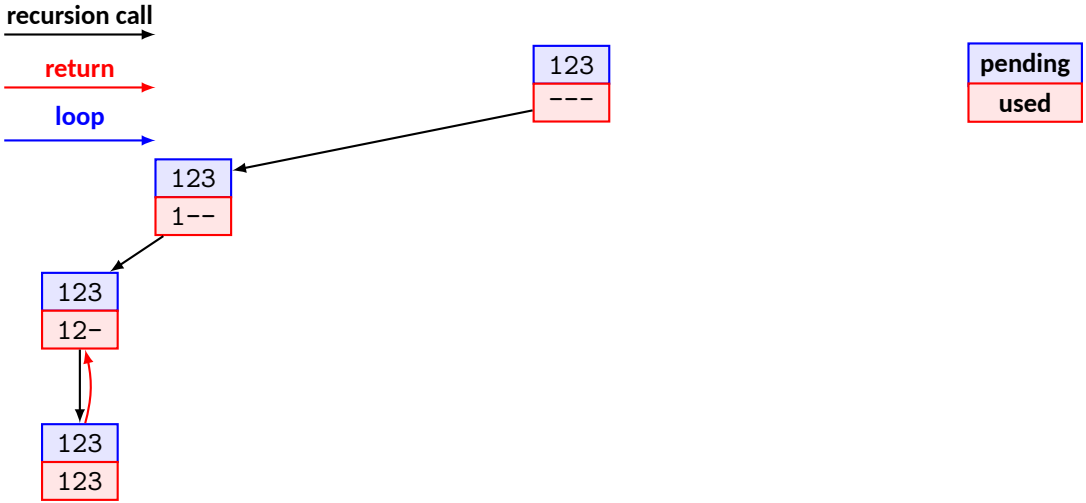
Generation of permutation



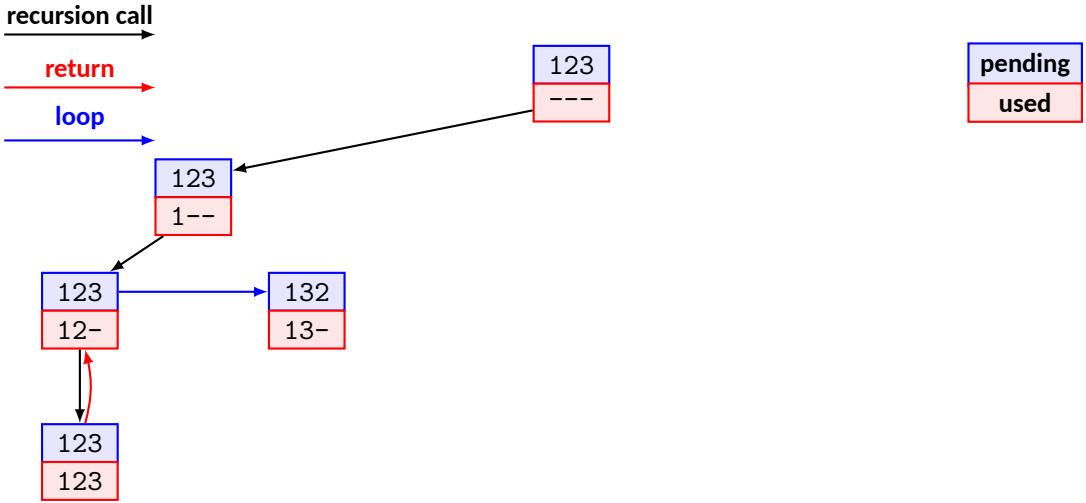
Generation of permutation



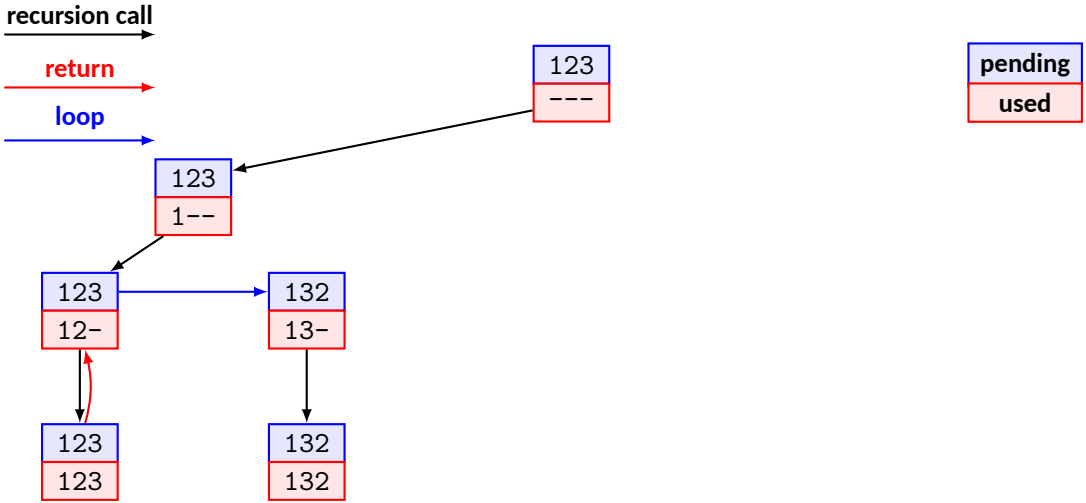
Generation of permutation



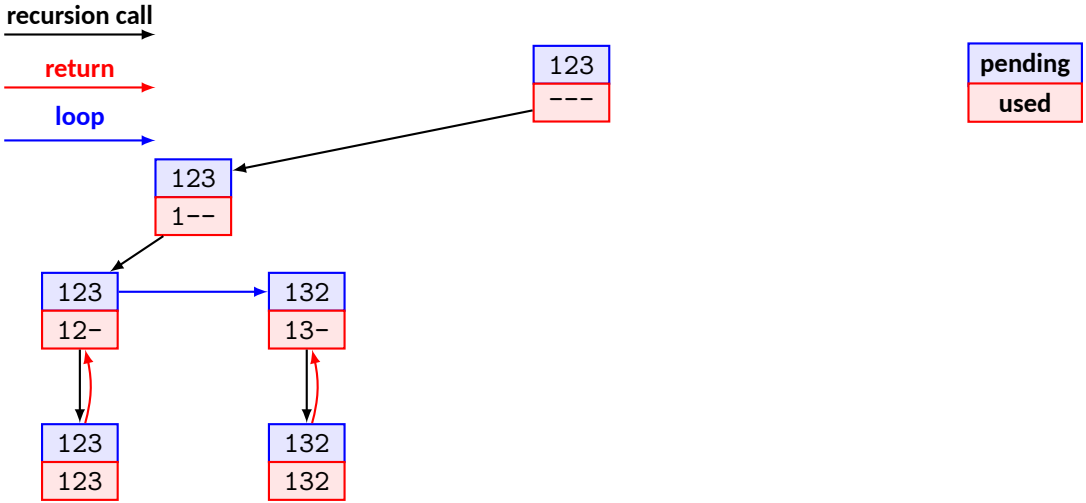
Generation of permutation



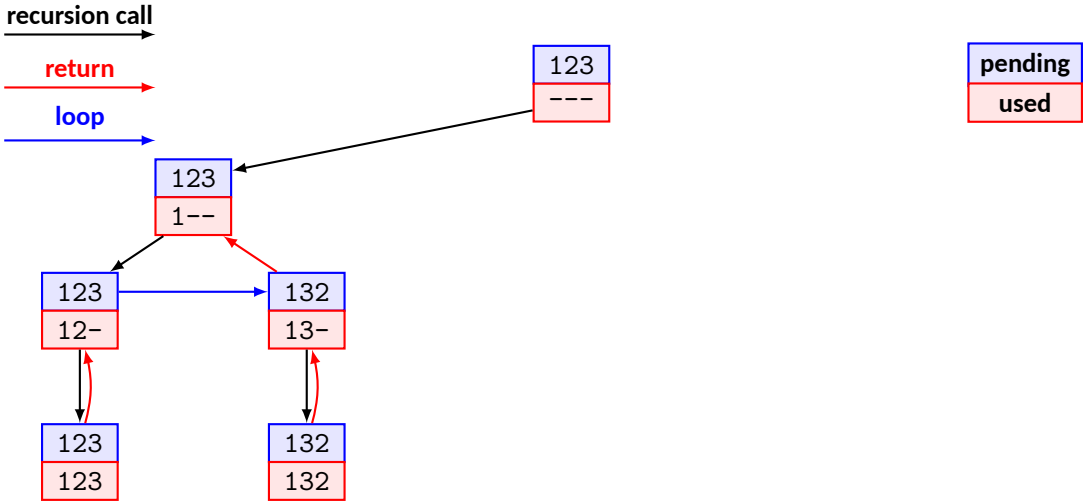
Generation of permutation



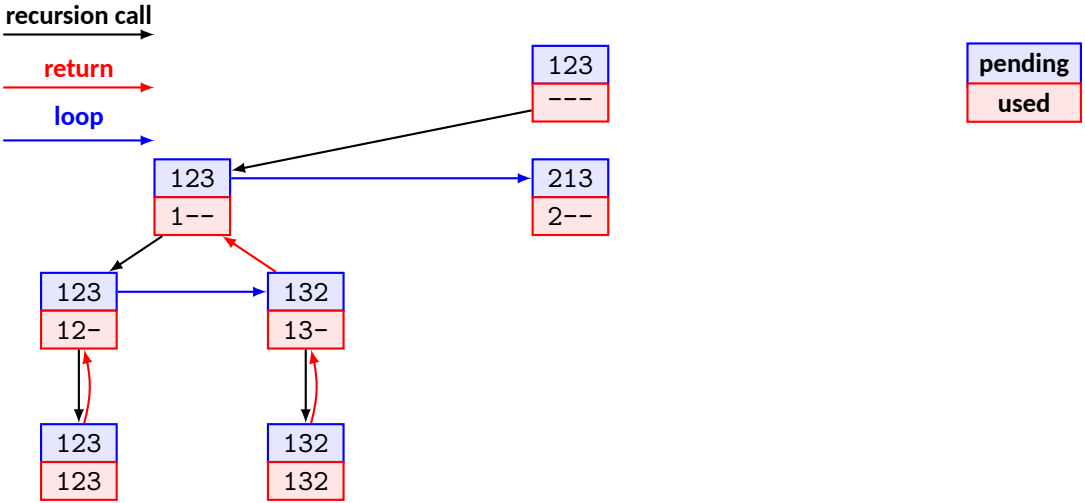
Generation of permutation



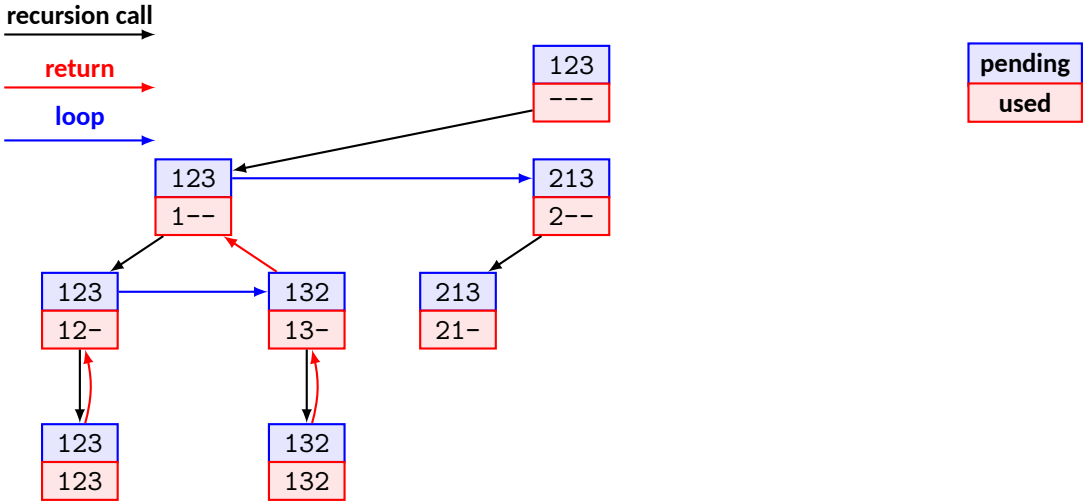
Generation of permutation



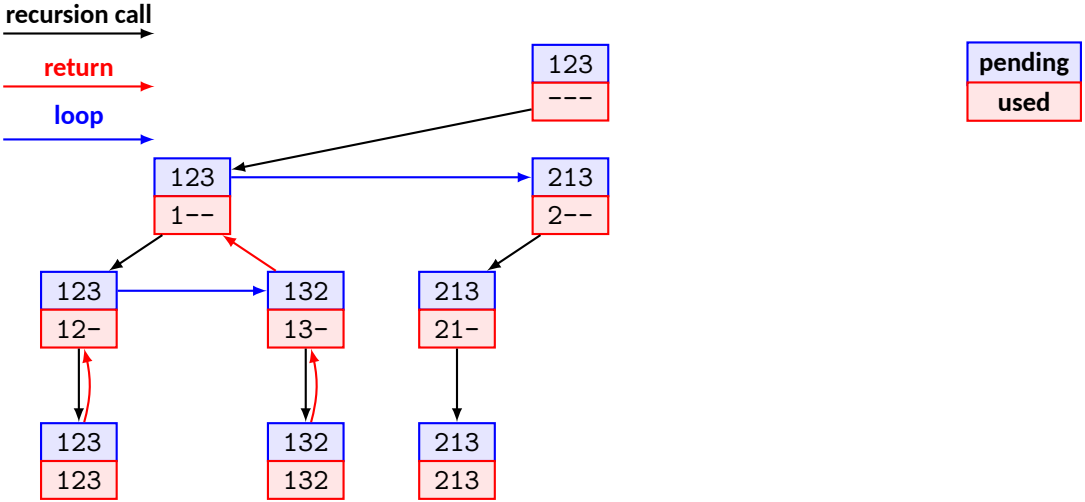
Generation of permutation



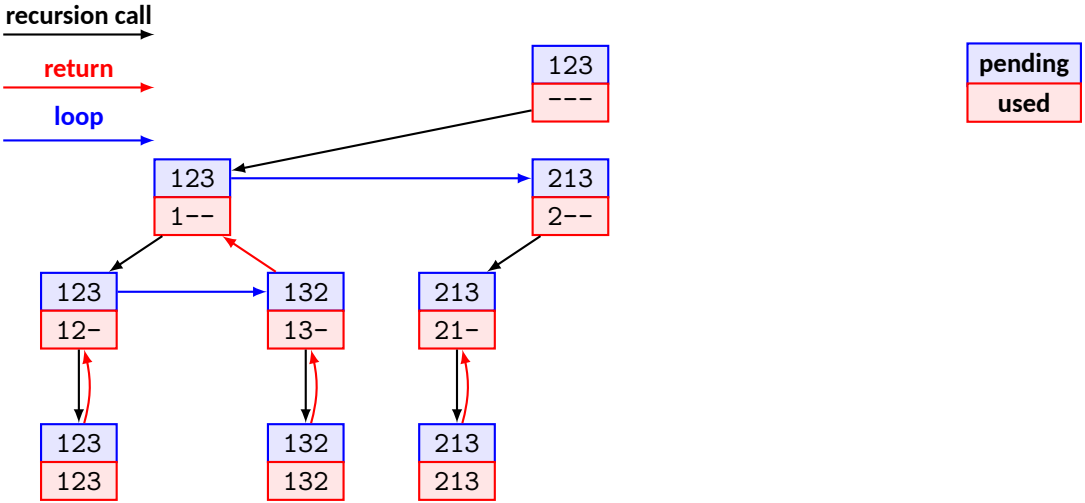
Generation of permutation



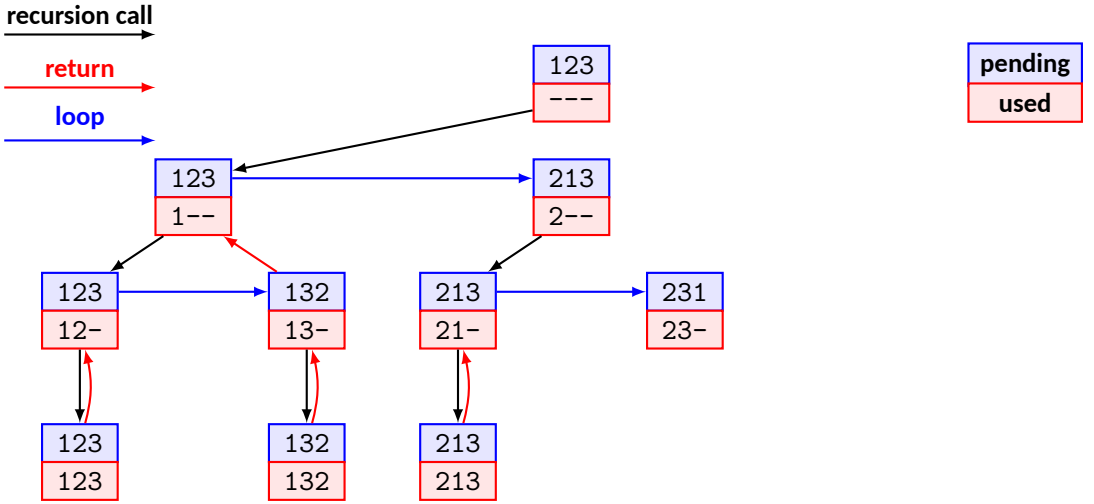
Generation of permutation



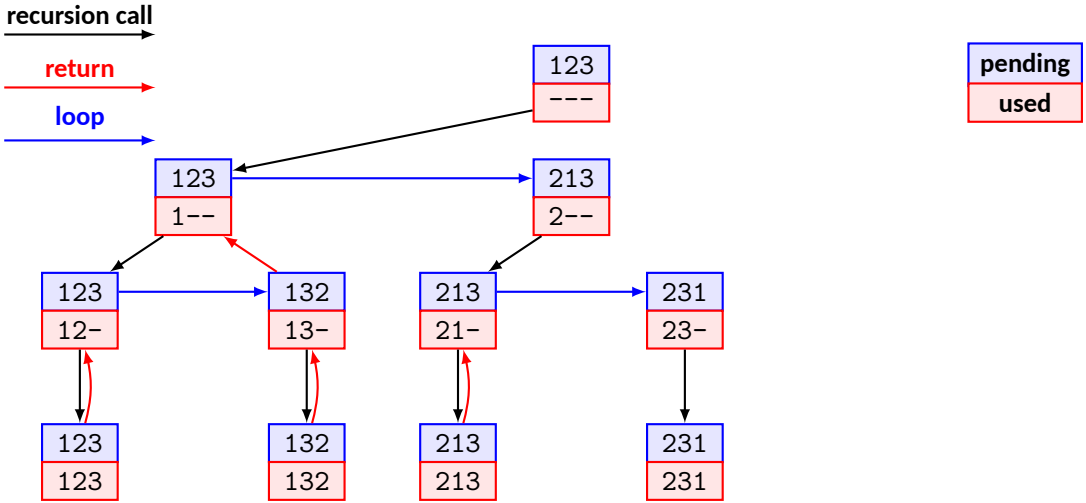
Generation of permutation



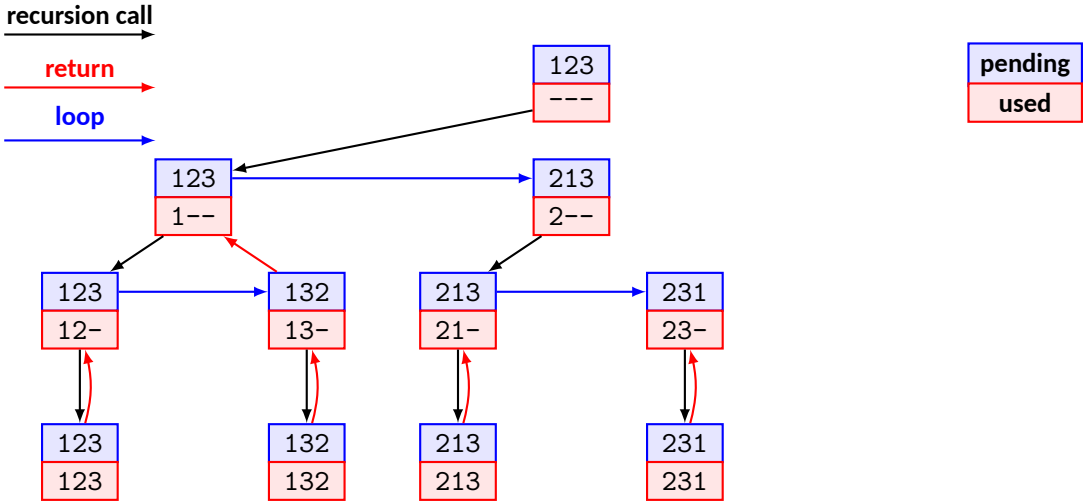
Generation of permutation



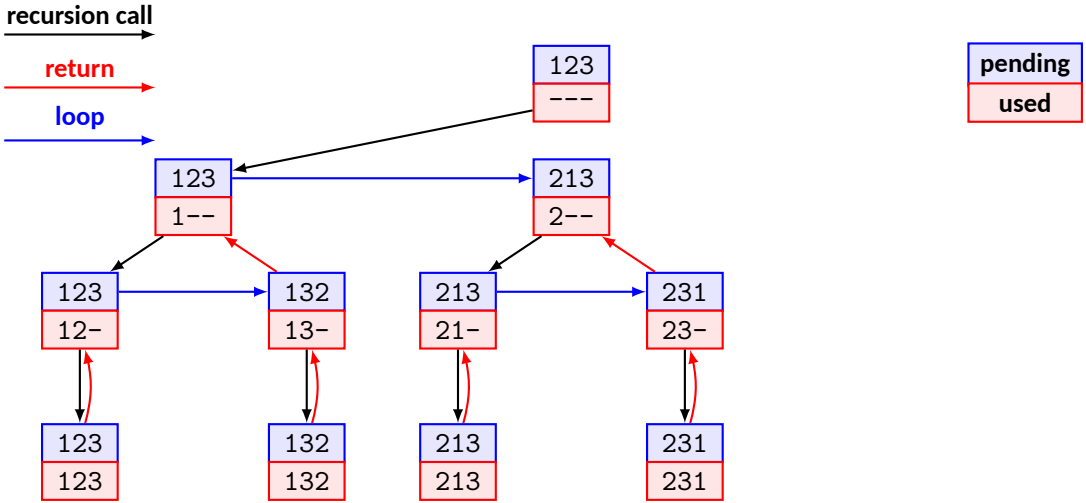
Generation of permutation



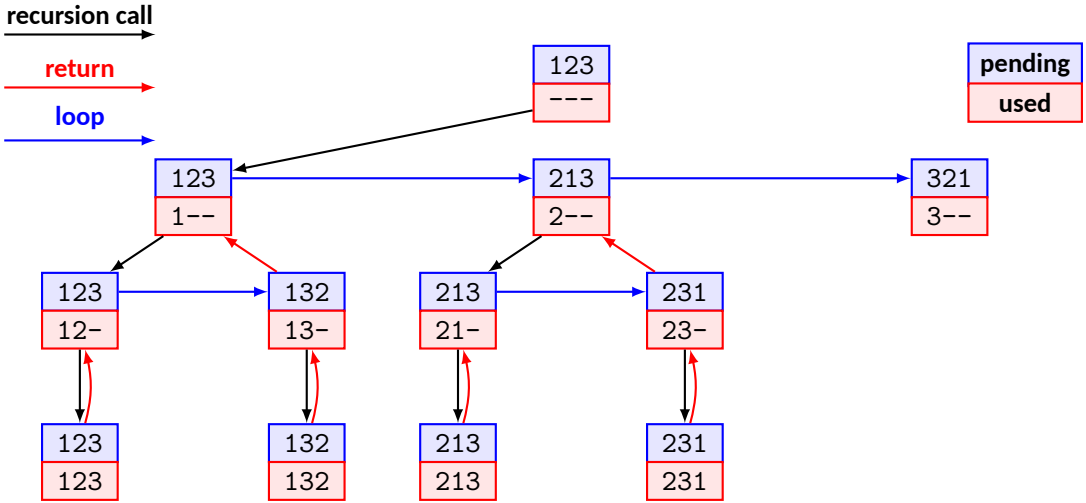
Generation of permutation



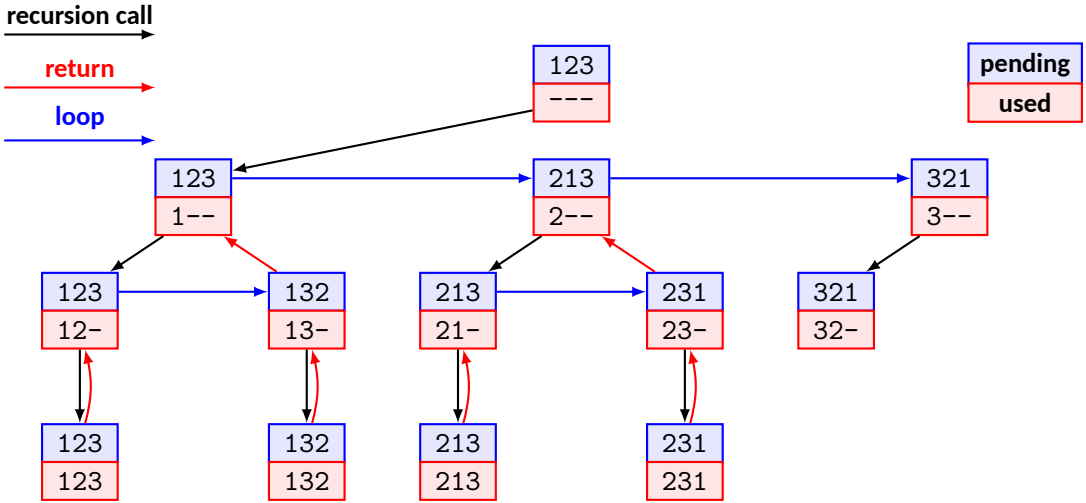
Generation of permutation



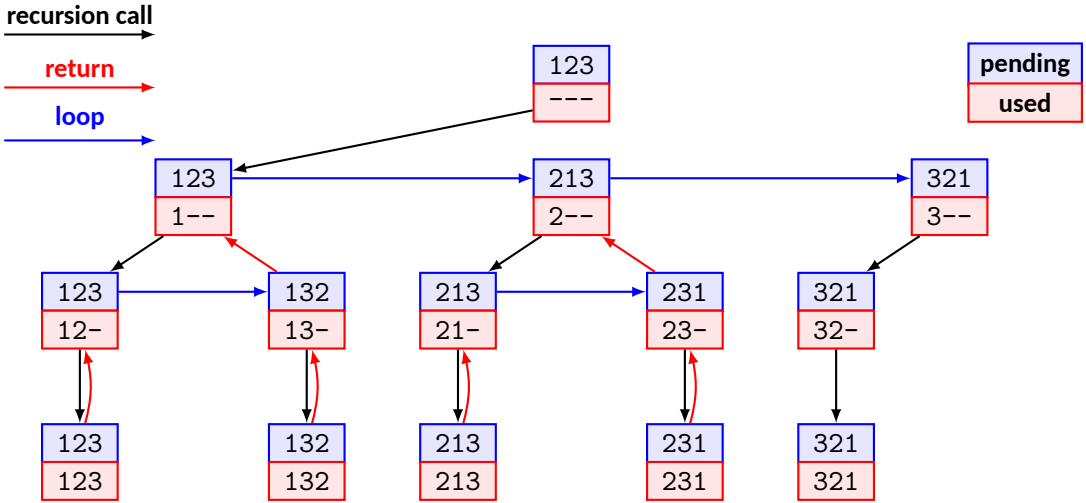
Generation of permutation



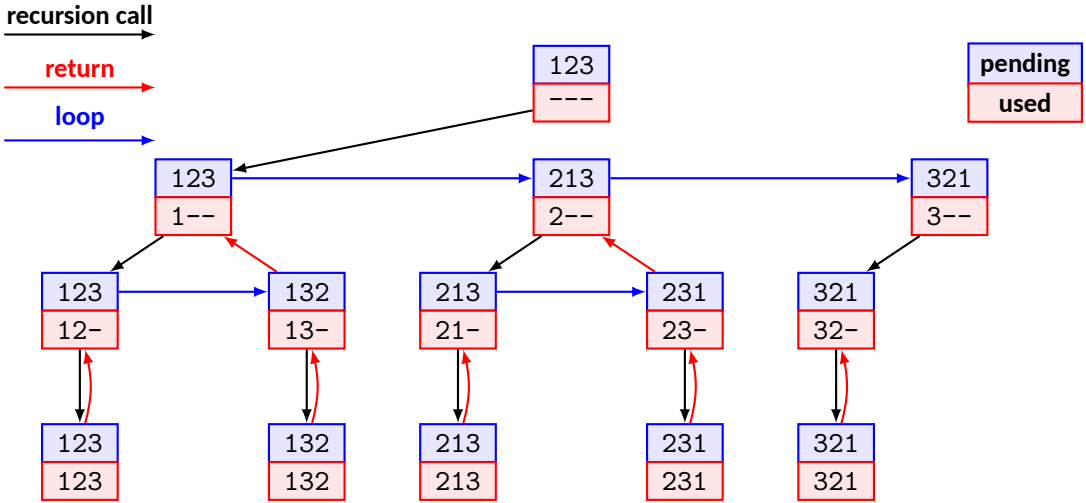
Generation of permutation



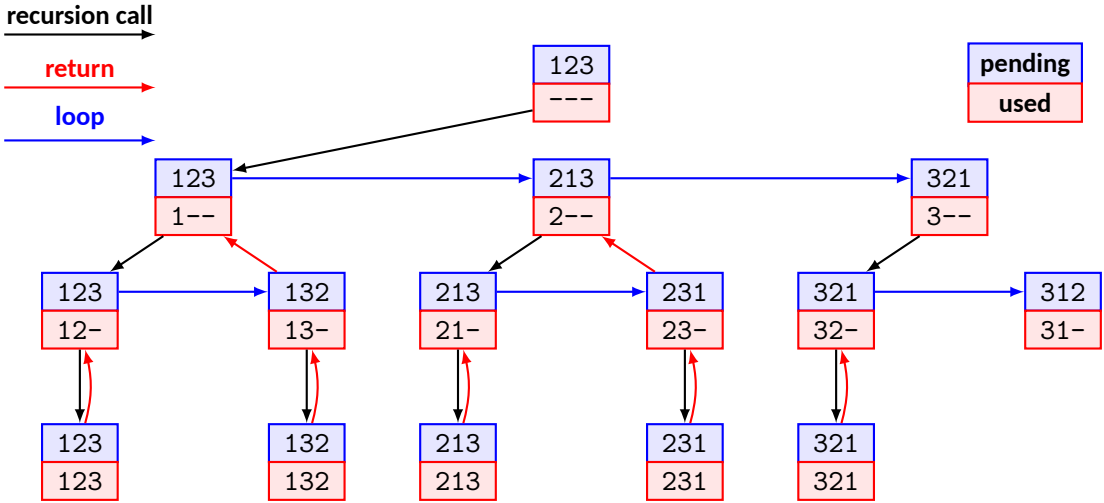
Generation of permutation



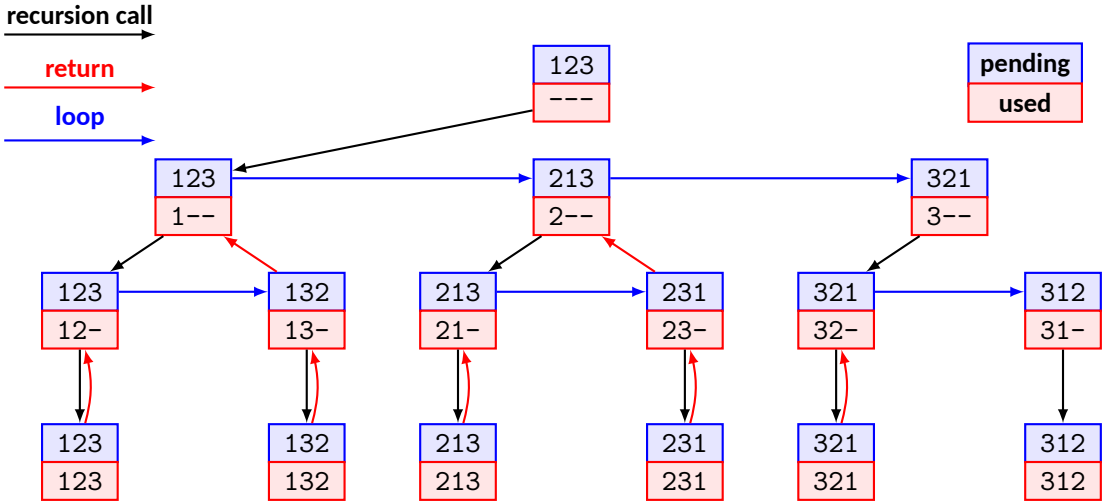
Generation of permutation



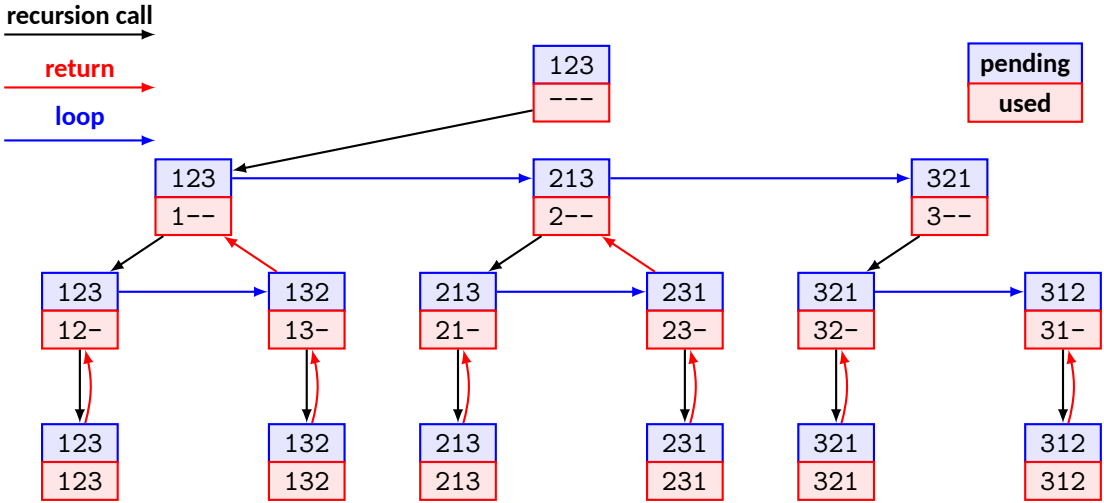
Generation of permutation



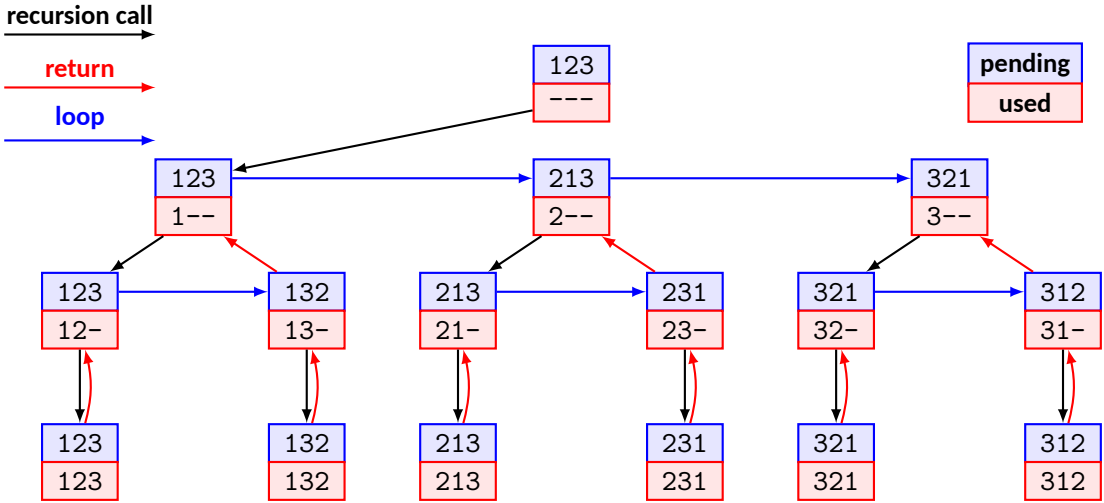
Generation of permutation



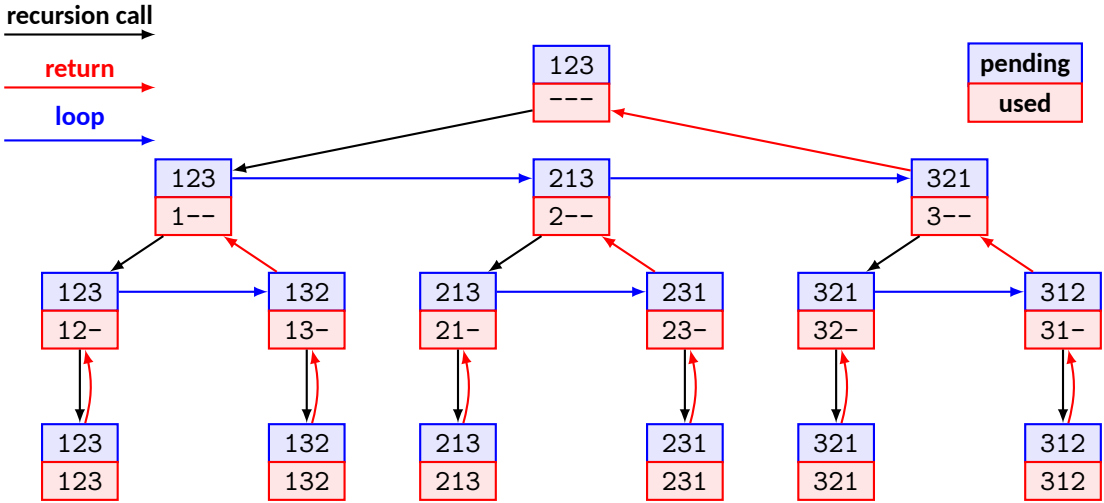
Generation of permutation



Generation of permutation



Generation of permutation



Generation of combination

Generation of combination

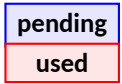
```
void combination(int pending[], int size, int used[], int count, int consumed){
   if(consumed == size){
     for(int i=0; i < count; i++) printf("%d",used[i]);
     printf("\n");
     return ;
   }
   used[count] = pending[consumed];
   combination(pending, size, used, count+1, consumed+1);
   combination(pending, size, used, count, consumed+1);
}
int main(){
   int pending[]={2,5,8}, used[10], consumed=0, size=3, count=0;
   combination(pending, size, used, count, consumed);
   return 0;
}
```

Generation of combination

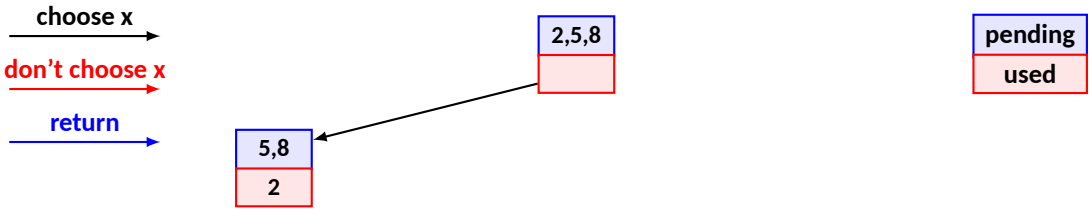
choose x

don't choose x

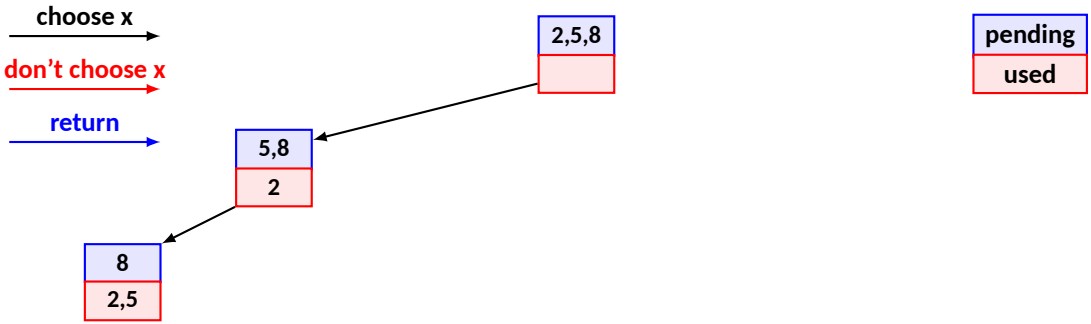
return



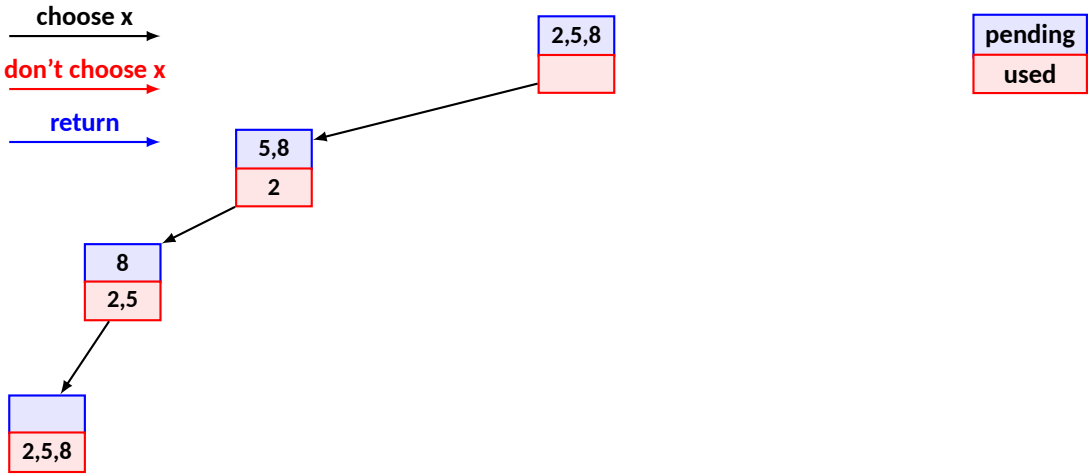
Generation of combination



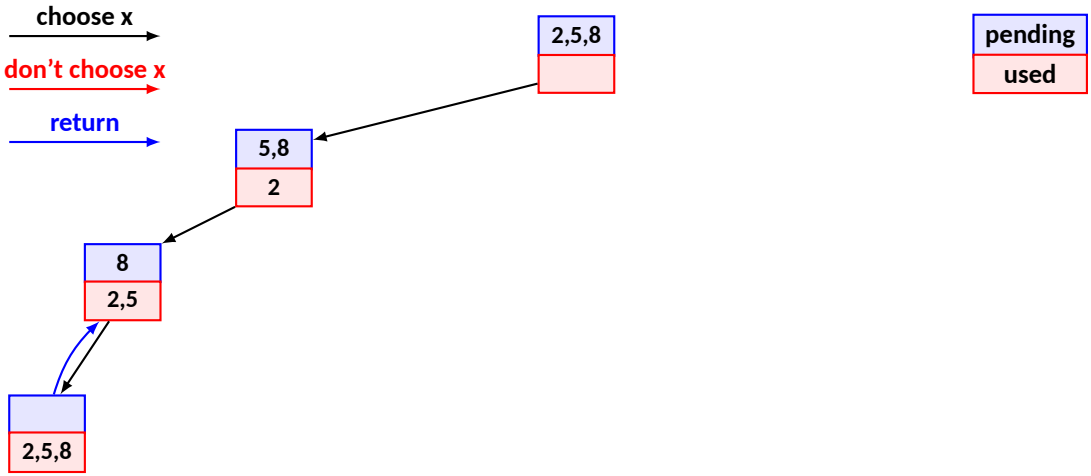
Generation of combination



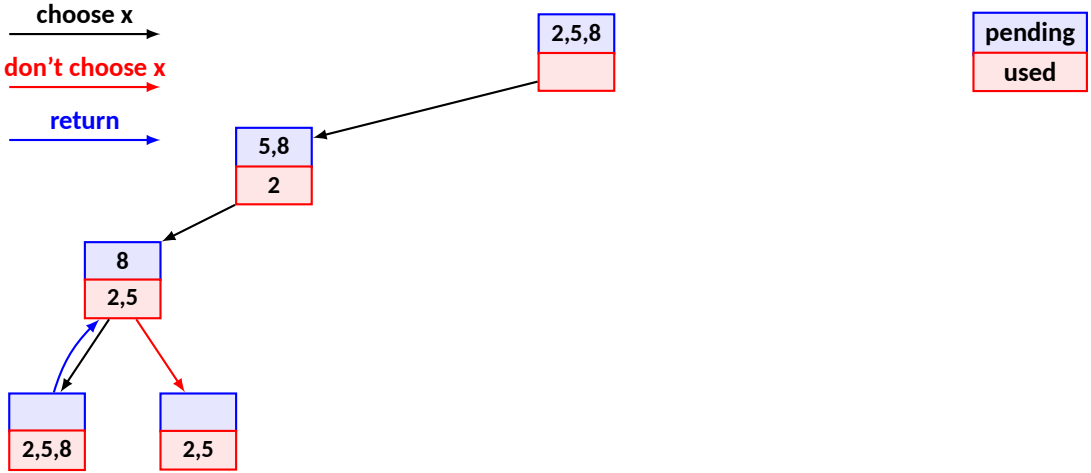
Generation of combination



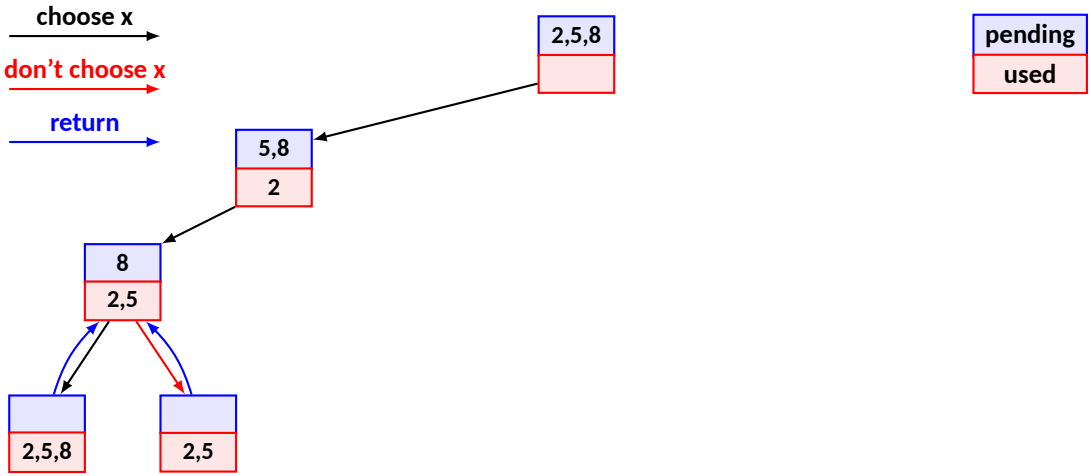
Generation of combination



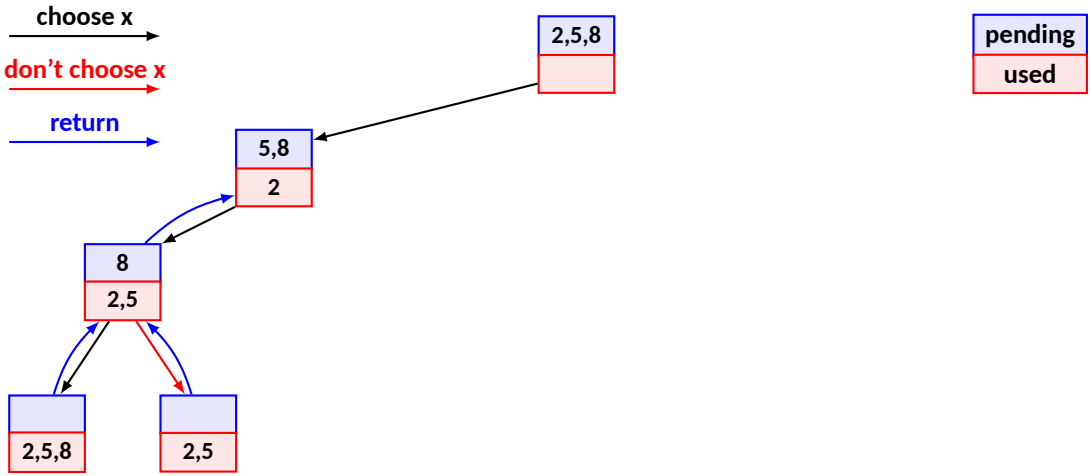
Generation of combination



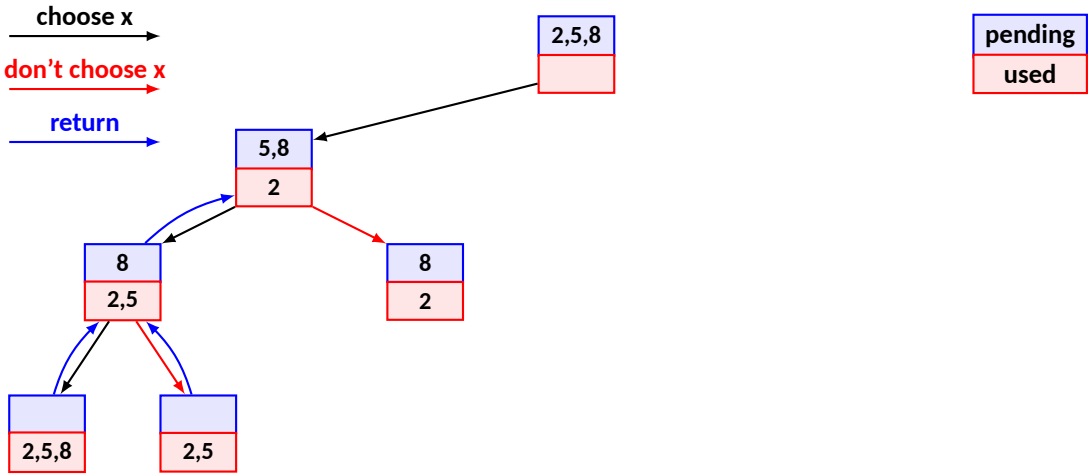
Generation of combination



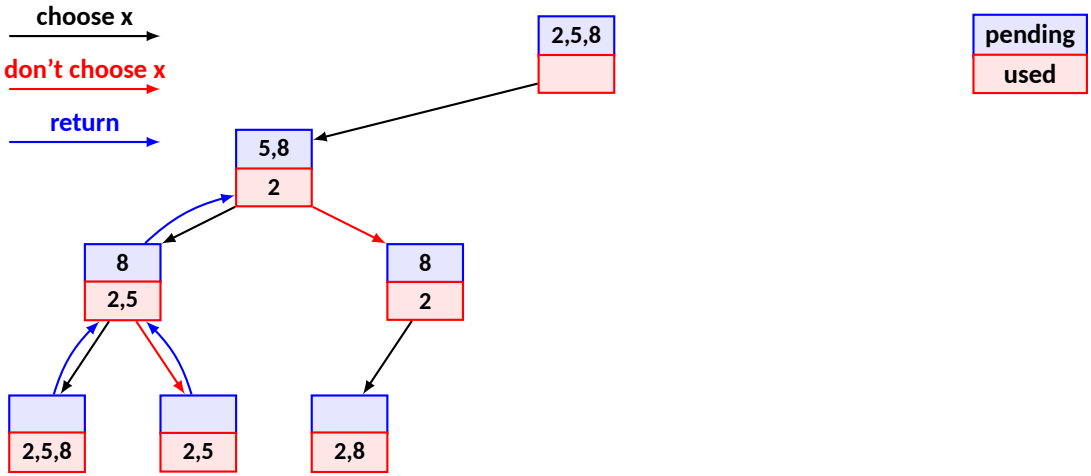
Generation of combination



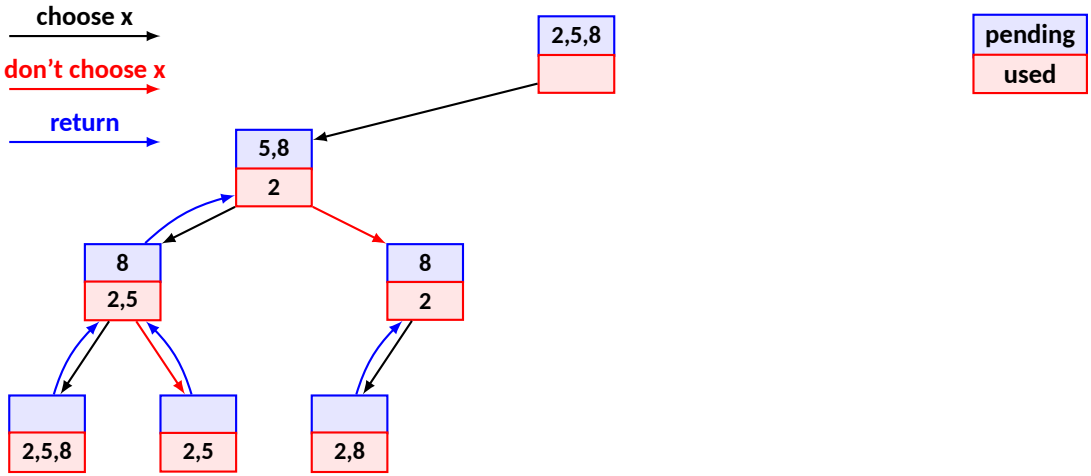
Generation of combination



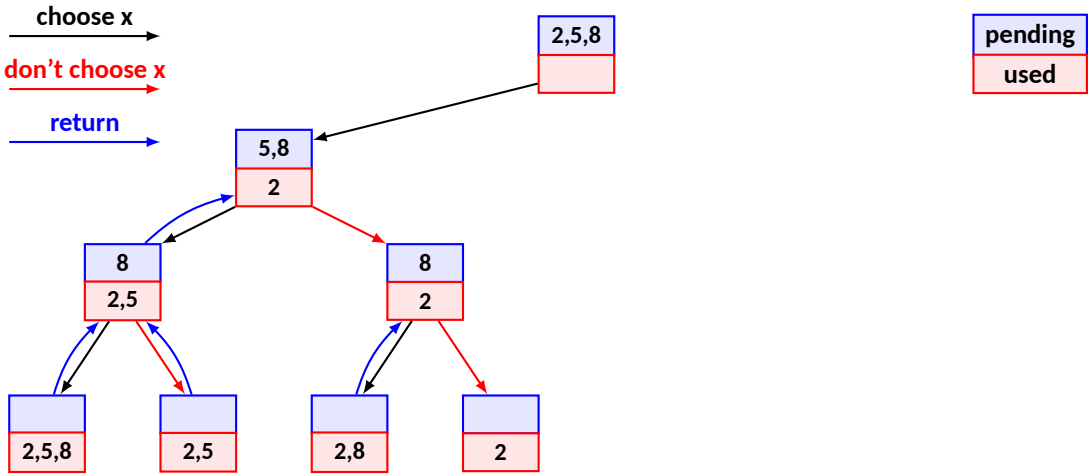
Generation of combination



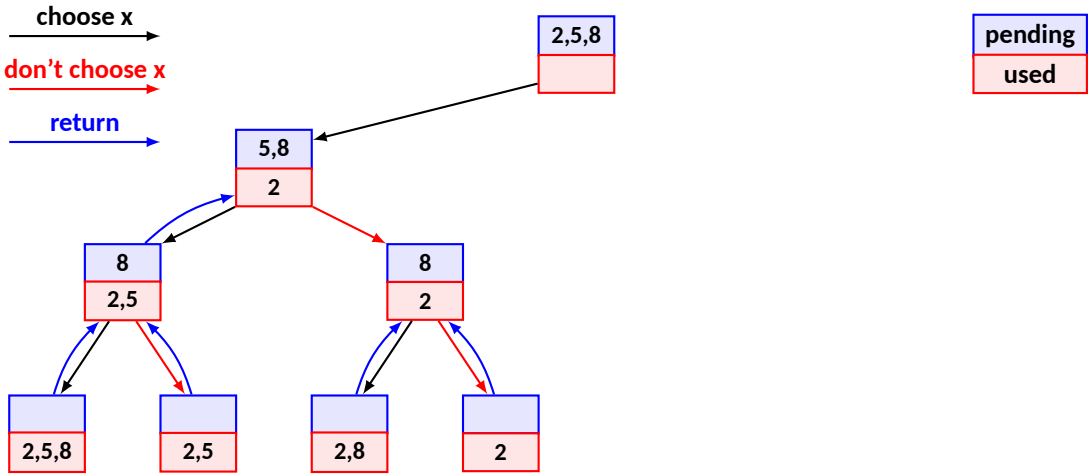
Generation of combination



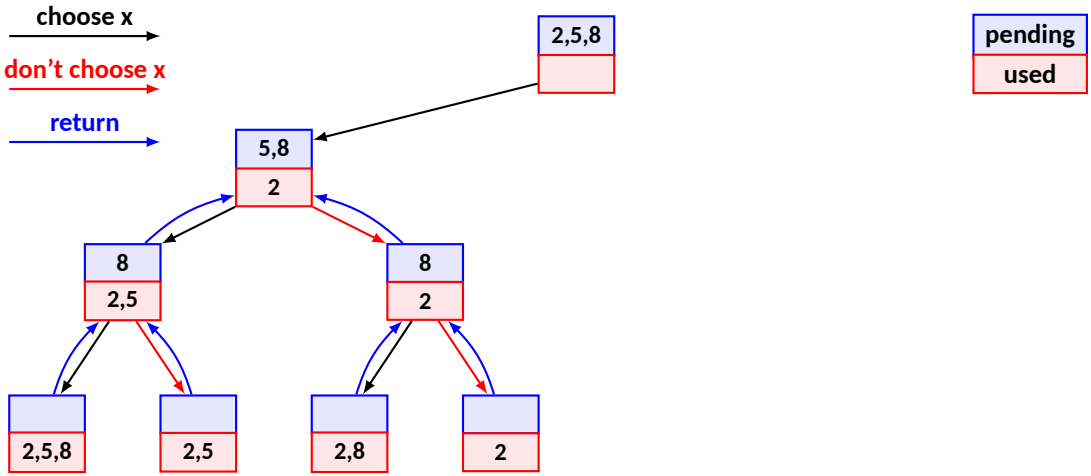
Generation of combination



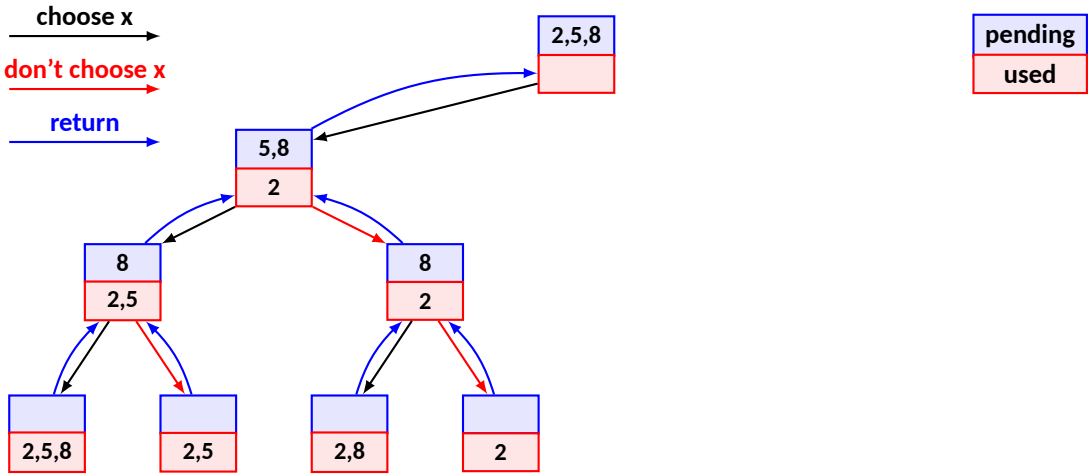
Generation of combination



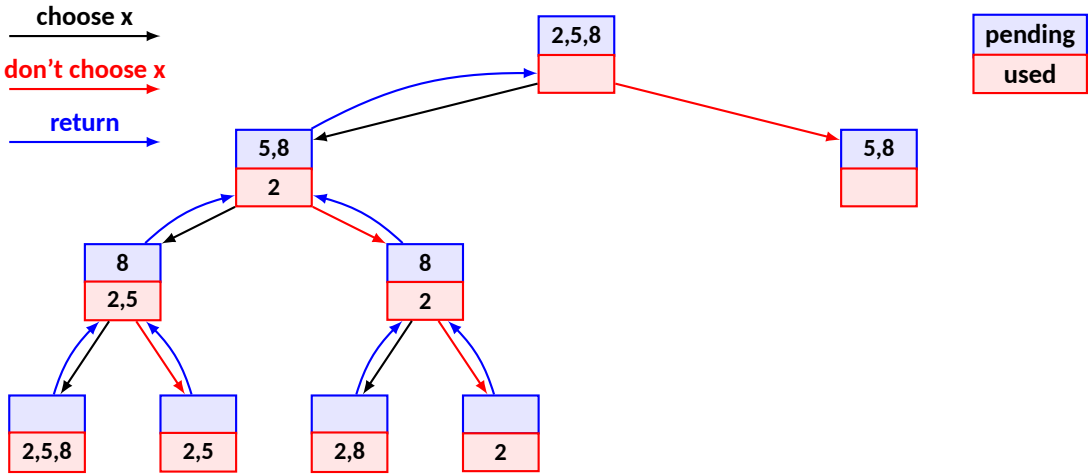
Generation of combination



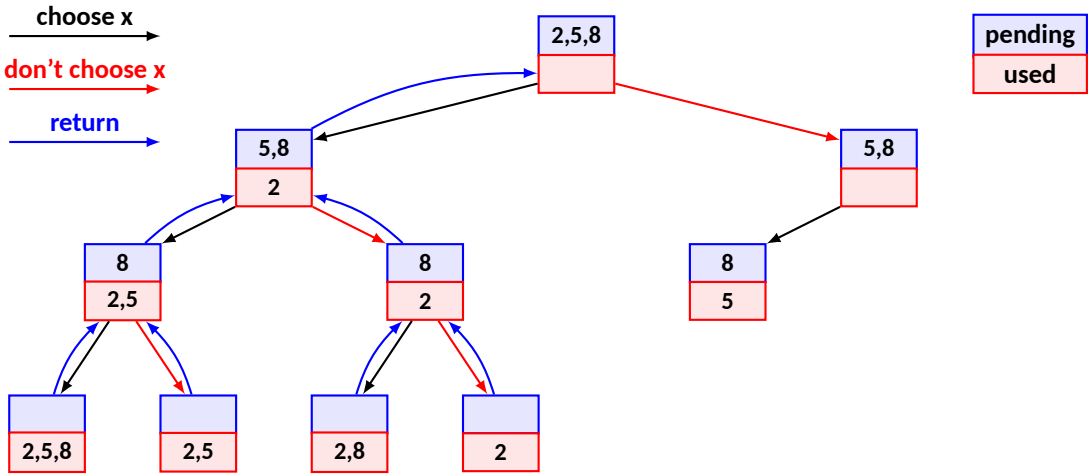
Generation of combination



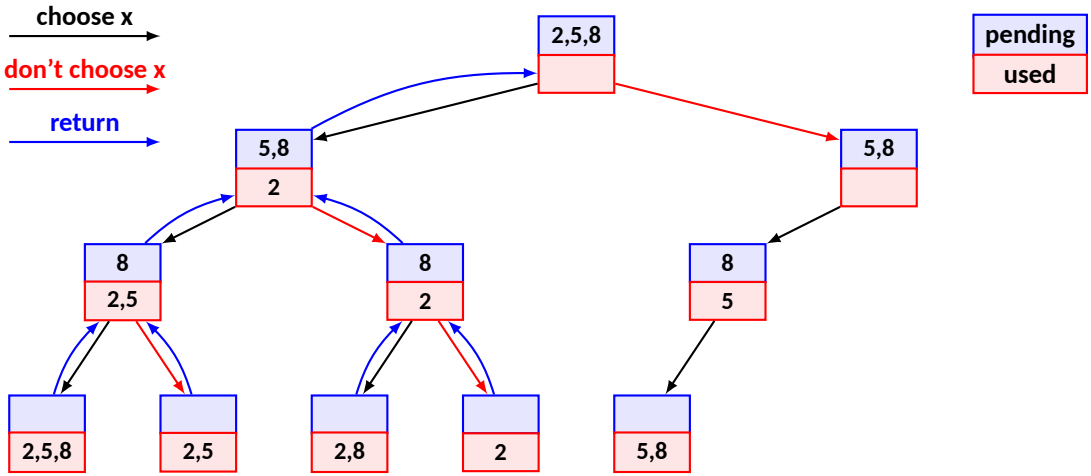
Generation of combination



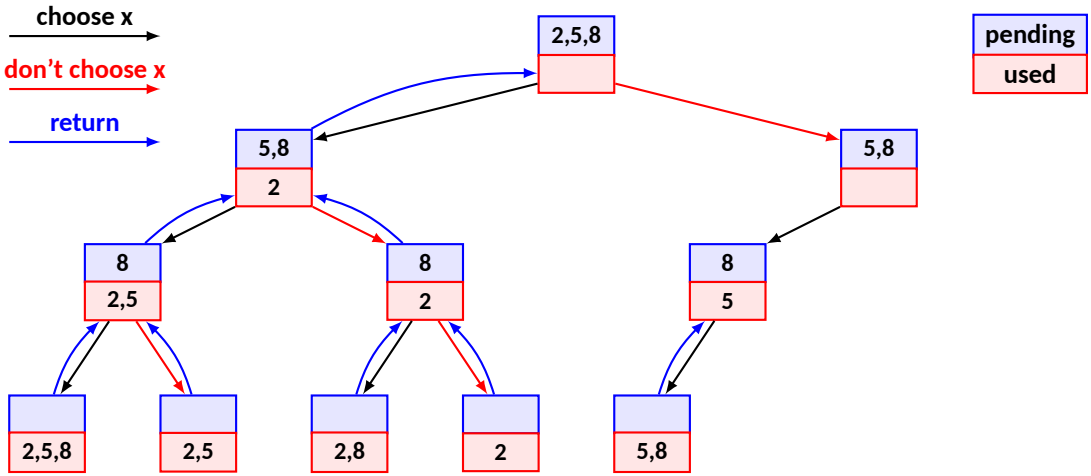
Generation of combination



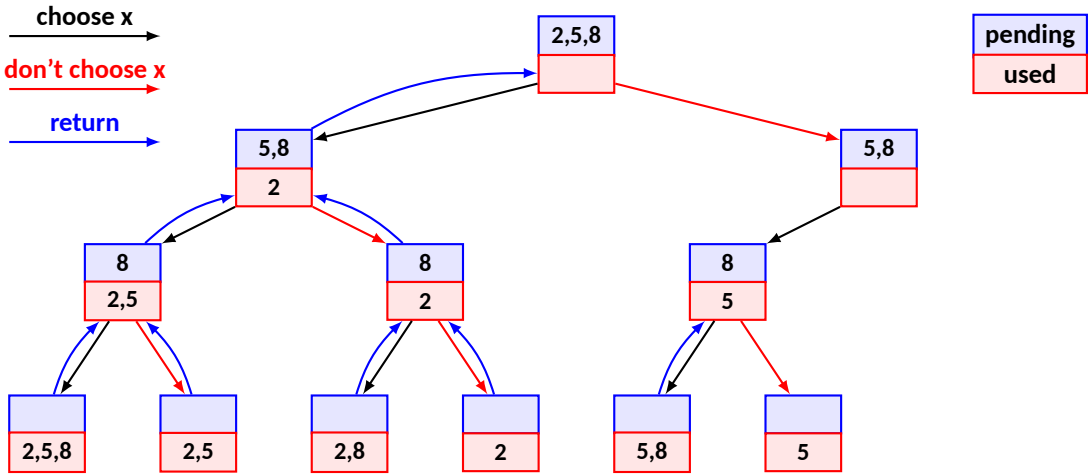
Generation of combination



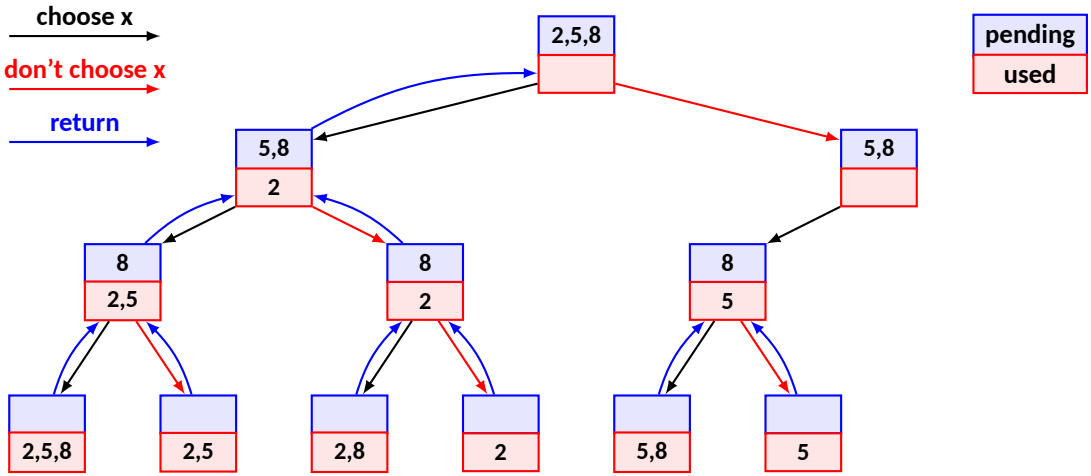
Generation of combination



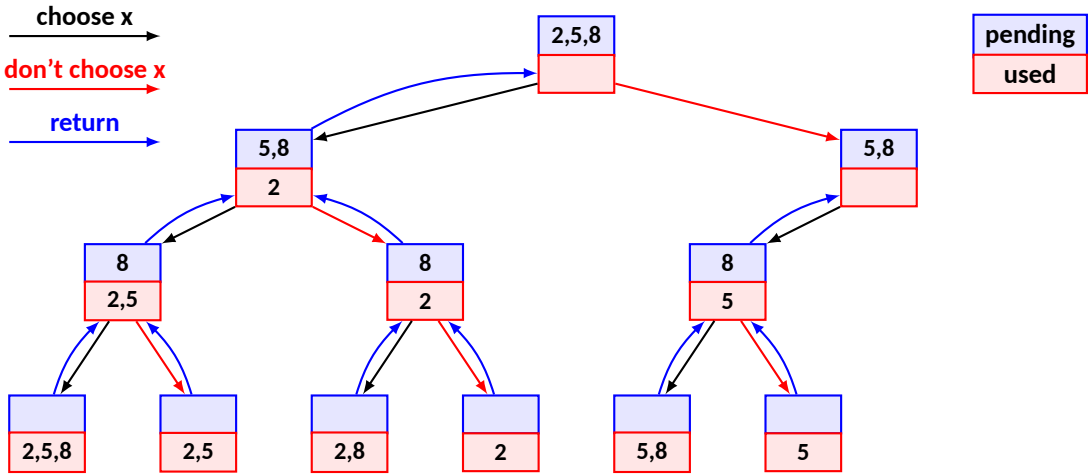
Generation of combination



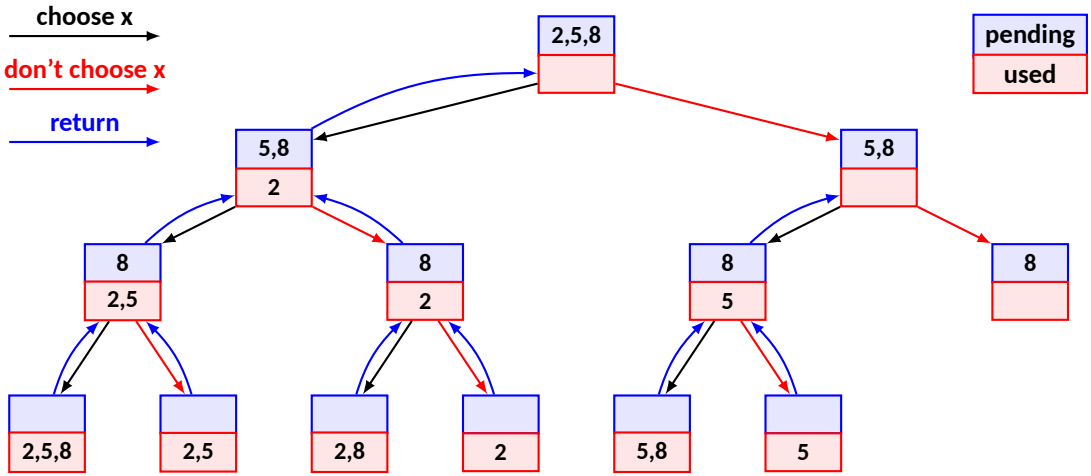
Generation of combination



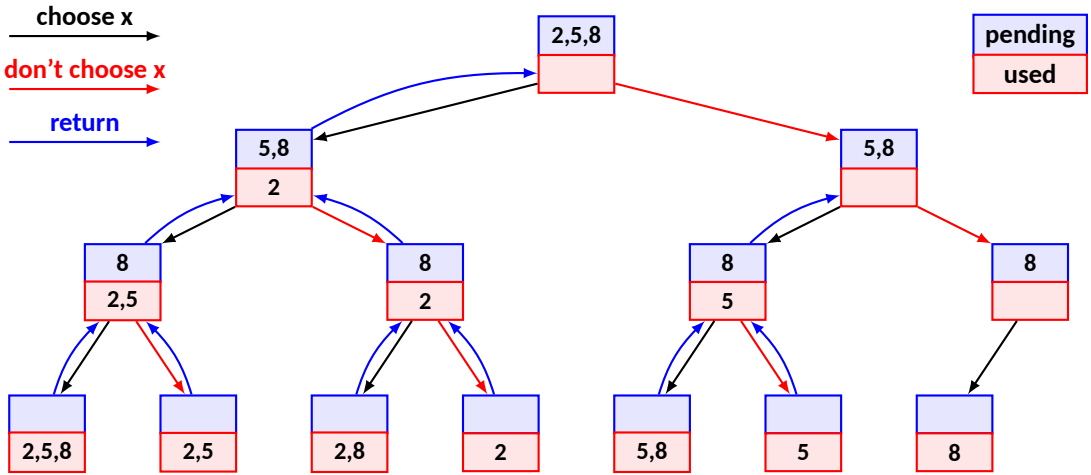
Generation of combination



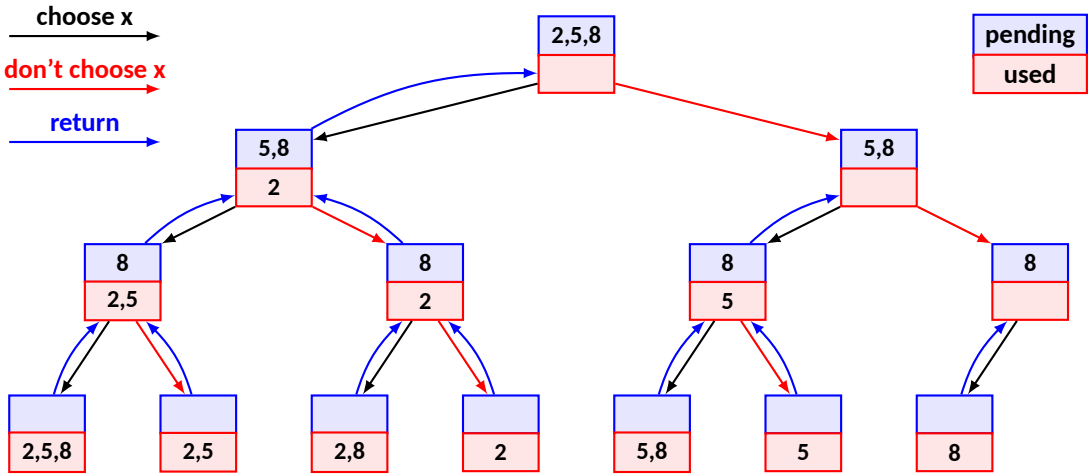
Generation of combination



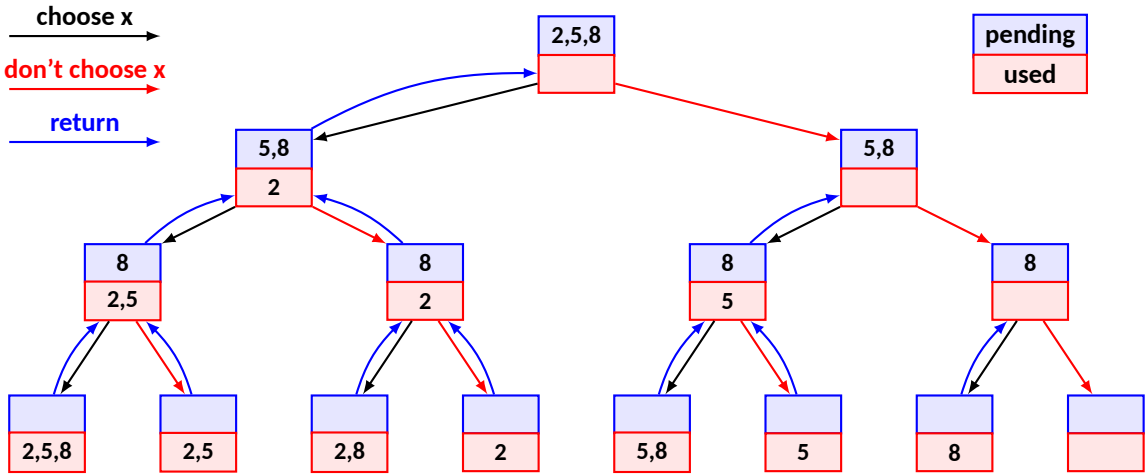
Generation of combination



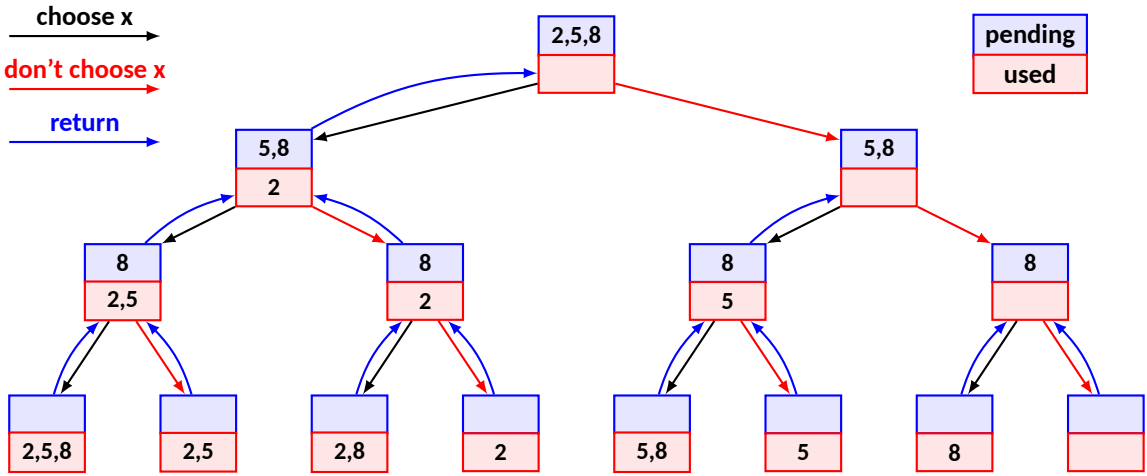
Generation of combination



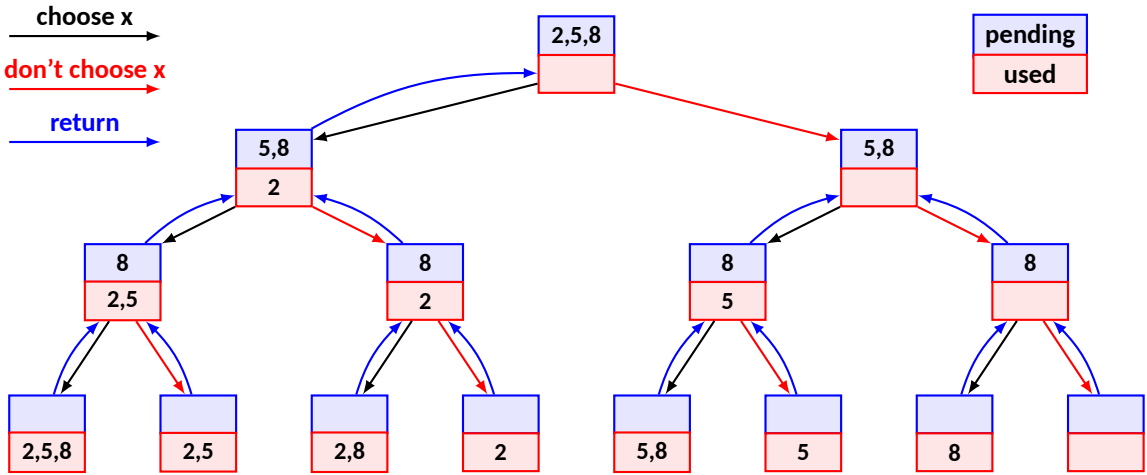
Generation of combination



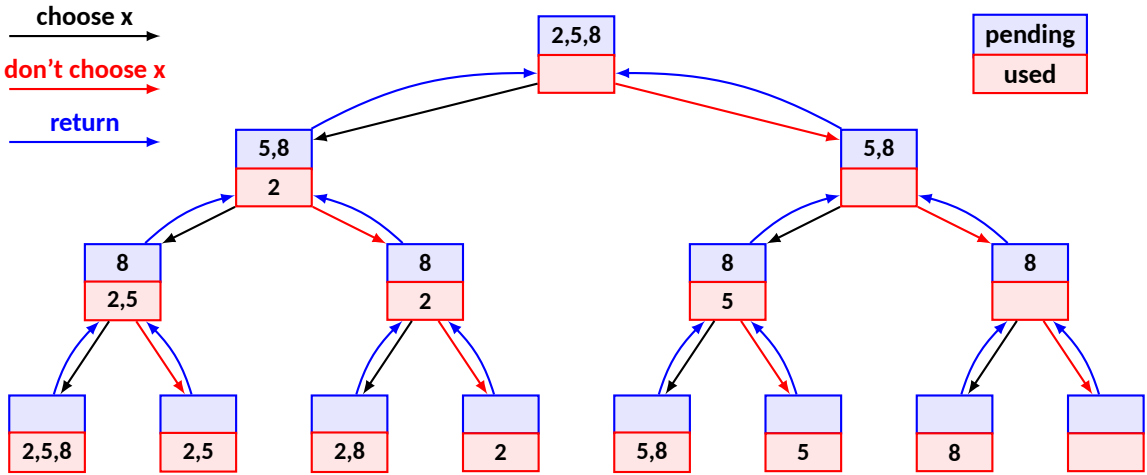
Generation of combination



Generation of combination



Generation of combination



Coin change

- A country has coins of denomination 3, 5, and 10 respectively
- Write a function coinChange(k) that returns -1 if it is not possible to pay a value of k using these coins, otherwise returns the minimum number of coins rerequired for k
- For example coinChange(7) will return -1
- For coinChange(14) will return 4 because 14 can be paid as 3+3+3+5 and there is no other way
- For coinChange(15) it should return 2 as 5+10 though there are other options like 3+3+3+3+3, 5+5+5

Coin change

```
int coinChange(int v){
  int a;
  if(v == 0) return 0;
  if((v == 3) || (v == 5) || (v == 10)) return 1;
  if(v < 3) return -1;
  a = coinChange(v-10); if(a > 0) return a+1;
  a = coinChange(v-5); if(a > 0) return a+1;
  a = coinChange(v-3); if(a > 0) return a+1;
  return -1;
}
```

Coin change

```
int coinChange(int v){
  int a;
  if(v == 0) return 0;
  if((v == 3) || (v == 5) || (v == 10)) return 1;
  if(v < 3) return -1;
  a = coinChange(v-10); if(a > 0) return a+1;
  a = coinChange(v-5); if(a > 0) return a+1;
  a = coinChange(v-3); if(a > 0) return a+1;
  return -1;
}
```

What will happen if the denomination of the coins are: 3, 8, and 10?

Paying with minimum coins

- Let v be the value and c_i are the coin denominations, there are n different coins
- It can be implemented using the following recursion idea

Paying with minimum coins

- Let v be the value and c_i are the coin denominations, there are n different coins
- It can be implemented using the following recursion idea

$$\textbf{minCoins}(\{c_i\}, v) = \begin{cases} 0, & \text{if } v = 0 \\ \min\limits_{1 \le i \le n} \{1 + \textbf{minCoins}(\{c_i\}, v - c_i)\} & \text{if } v > 0 \end{cases}$$

This has been left as an exercise.