ACM/ICPC CheatSheet

Puzzles

Contents

1 STL Useful Tips **1**
- 1.1 Common libraries . . . 1
- 1.2 String . . . 2
- 1.3 Sort . . . 2
- 1.4 Permutations . . . 2
- 1.5 Searching . . . 2
- 1.6 Random algorithm . . . 2

2 Number Theory **3**
- 2.1 Max or min . . . 3
- 2.2 Greatest common divisor — GCD . . . 3
- 2.3 Least common multiple — LCM . . . 3
- 2.4 If prime number . . . 3
- 2.5 Leap year . . . 3
- 2.6 Factorial `mod` . . . 3
- 2.7 a^b `mod` p . . . 4
- 2.8 Generate combinations . . . 4

3 Searching Algorithms **4**
- 3.1 Depth first search — DFS . . . 4
- 3.2 Breath first search – BFS . . . 4
- 3.3 Find rank k in array . . . 4

4 Dynamic Programming **4**
- 4.1 Knapsack problems . . . 4
- 4.2 Longest common subsequence . . . 4
- 4.3 Maximum submatrix . . . 4

5 Trees **4**
- 5.1 Tree representation in array . . . 4
- 5.2 Tree traversal . . . 4

6 Graph Theory **4**
- 6.1 Flood fill algorithm . . . 4
- 6.2 SPFA — shortest path . . . 5
- 6.3 Floyd-Warshall algorithm – shortest path of all pairs . . . 5
- 6.4 Prim — minimum spanning tree . . . 5
- 6.5 Eulerian path . . . 5
- 6.6 Topological sort . . . 5

1 STL Useful Tips

1.1 Common libraries

```
#include<iostream>
#include<cstdio>
#include<cmath>
#include<string>
#include<queue>
#include<stack>
#include<vector>
#include<deque> // double ended queue
#include<priority_queue> // priority queue
#include <functional> // for hash
#include<algorithm>
```

```
#include<cstdlib> // random
#include<ctime>
#include<sstream>
```

1.2 String

```
// Searching
unsigned int find(const string &s2, unsigned int pos1 = 0);
unsigned int rfind(const string &s2, unsigned int pos1 = end);
unsigned int find_first_of(const string &s2, unsigned int pos1 = 0);
unsigned int find_last_of(const string &s2, unsigned int pos1 = end);
unsigned int find_first_not_of(const string &s2, unsigned int pos1 = 0);
unsigned int find_last_not_of(const string &s2, unsigned int pos1 = end);
// Insert, Erase, Replace
string& insert(unsigned int pos1, const string &s2);
string& insert(unsigned int pos1, unsigned int repetitions, char c);
string& erase(unsigned int pos = 0, unsigned int len = npos);
string& replace(unsigned int pos1, unsigned int len1, const string &s2);
string& replace(unsigned int pos1, unsigned int len1, unsigned int repetitions, char c);
// String streams
stringstream s1;
int i = 22;
s1 << "Hello world! " << i;
cout << s1.str() << endl;
```

1.3 Sort

```
void sort(iterator first, iterator last);
void sort(iterator first, iterator last, LessThanFunction comp);
void stable_sort(iterator first, iterator last);
void stable_sort(iterator first, iterator last, LessThanFunction comp);
void partial_sort(iterator first, iterator middle, iterator last);
void partial_sort(iterator first, iterator middle, iterator last, LessThanFunction comp);
bool is_sorted(iterator first, iterator last);
bool is_sorted(iterator first, iterator last, LessThanOrEqualFunction comp);
// example for sort, if have array x, start_index, end_index;
sort(x+start_index, x+end_index);
```

1.4 Permutations

```
bool next_permutation(iterator first, iterator last);
bool next_permutation(iterator first, iterator last, LessThanOrEqualFunction comp);
bool prev_permutation(iterator first, iterator last);
bool prev_permutation(iterator first, iterator last, LessThanOrEqualFunction comp);
```

1.5 Searching

```
iterator find(iterator first, iterator last, const T &value);
iterator find_if(iterator first, iterator last, const T &value, TestFunction test);
bool binary_search(iterator first, iterator last, const T &value);
bool binary_search(iterator first, iterator last, const T &value, LessThanOrEqualFunction comp);
```

1.6 Random algorithm

```
srand(time(NULL));
// generate random numbers between [a,b)
rand() % (b - a) + a;
// generate random numbers between [0,b)
rand() % b;
// generate random permutations
random_permutation(anArray, anArray + 10);
random_permutation(aVector, aVector + 10);
```

2 Number Theory

2.1 Max or min

```
int max(int a, int b) { return a>b ? a:b; }
int min(int a, int b) { return a<b ? a:b; }
```

2.2 Greatest common divisor — GCD

```
int gcd(int a, int b)
{
  if (b==0) return a;
  else return gcd(b, a%b);
}
```

2.3 Least common multiple — LCM

```
int lcm(int a, int b)
{
  return a*b/gcd(a,b);
}
```

2.4 If prime number

```
bool prime(int n)
{
  for (int i=2;i*i<=n;i++)
    if (n%i==0) return false;
  return true;
}
```

2.5 Leap year

```
bool isLeap(int n)
{
  if (n%100==0)
    if (n%400==0) return true;
    else return false;

  if (n%4==0) return true;
  else return false;
}
```

2.6 Factorial mod

```
//n! mod p
int factmod (int n, int p) {
  long long res = 1;
  while (n > 1) {
    res = (res * powmod (p-1, n/p, p)) % p;
    for (int i=2; i<=n%p; ++i)
      res=(res*i) %p;
    n /= p;
  }
  return int (res % p);
}
```

2.7 a^b mod p

2.8 Generate combinations

3 Searching Algorithms

3.1 Depth first search — DFS

3.2 Breath first search – BFS

3.3 Find rank k in array

4 Dynamic Programming

4.1 Knapsack problems

4.2 Longest common subsequence

4.3 Maximum submatrix

5 Trees

5.1 Tree representation in array

5.2 Tree traversal

6 Graph Theory

6.1 Flood fill algorithm

```
//component(i) denotes the
//component that node i is in
void flood_fill(new_component)
  do
    num_visited = 0
    for all nodes i
      if component(i) = -2
      num_visited = num_visited + 1
      component(i) = new_component

    for all neighbors j of node i
      if component(j) = nil
        component(j) = -2
  until num_visited = 0

void find_components()
```

```
  num_components = 0
  for all nodes i
    component(node i) = nil
  for all nodes i
    if component(node i) is nil
      num_components = num_components + 1
      component(i) = -2
      flood_fill(component num_components)
```

6.2 SPFA — shortest path

6.3 Floyd-Warshall algorithm – shortest path of all pairs

```
// map[i][j]=infinity at start
void floyd()
{
  for (int k=1; k<=n; k++)
    for (int i=1; i<=n; i++)
      for (int j=1; j<=n; j++)
        if (i!=j && j!=k && i!=k)
          if (map[i][k]+map[k][j]<map[i][j])
            map[i][j]=map[i][k]+map[k][j];
}
```

6.4 Prim — minimum spanning tree

6.5 Eulerian path

6.6 Topological sort