Bruteforce

Jarif Rahman

28th January 2022

সূচীপত্র

Competitive Programming এর প্রব্লেম সল্ভিং এর একটা মেথড হলো bruteforce। এর মুল বিষয় হলো সব সম্ভাব্য উত্তর দ্বারা টেস্ট করে দেখা। যেমন ধর তোমাকে n টা point দেয়া হয়েছে, তোমাকে যেকোনো দুইটা point এর মধ্যে সবচেয়ে কম দূরত্ব বের করতে হবে (point বলতে cartesian coordinate এর point বুঝানো হচ্ছে)। এটার bruteforce solution হবে:

```
//sq(x) is equal to $x^2$
double sq(x){
    return x*x;
}

int main(){
    int n; cin >> n;
    vector<pair<double, double>> v(n);
    for(int i = 0; i < n; i++) cin >> v[i].first >> v[i].second;
    double mn = 1e9;

    for(int i = 0; i < n; i++) for(int j = i+1; j < n; j++) mn = min(mn,
        sqrt(sq(v[i].first-v[j].first) + sq(v[i].second-v[j].second)) );

    // distance between 2 point $(x_1,y_1),(x_2,y_2)$ is $\sqrt{(x_1-x_2)^2+(y_1-y_2)^2}$

    cout << mn << "\n";
}
```

এর time complexity $O(n^2)$। Sweep Line ব্যবহার করে এটাকে $O(n \log n)$ এও করা যেত। সাধারণত bruteforce solution অনেক সহজ হয়। কিন্তু এর complexity বেশি হয়। কিন্তু অনেক কঠিন প্রব্লেম এও bruteforce লাগতে পারে। আবার কোনো কোনো প্রব্লেম পুরাপুরি bruteforce না হলেও এর অংশ হিসেবে bruteforce লাগতে পারে।

সাধারণত $O(n^2)$, $O(n^3)$, $O(n^4)$ এরকম ধরণের bruteforce অনেক সহজ হয়। সেগুলা নিয়ে আর আলোচনা করা হলো না। আমরা কয়েকটা গুরুত্বপুর্ণ bruteforce প্রব্লেম নিয়ে আলোচনা করব যেগুলা সাধারণ লুপ দিয়ে করা যায় না।

1 Generating Subsets

প্রব্লেম: তোমাকে n সাইজ এর একটা সেট দেয়া আছে, তোমাকে এর সব সাবসেট প্রিন্ট করা লাগবে।

আমরা এটা সল্ভ করার জন্য $\{0, 1, 2, \cdots, n-1\}$ সেট এর কথা চিন্তা করতে পারি। অন্য সেট এর ক্ষেত্রেও একই ভাবে সল্ভ করা যাবে। আমরা এটাকে recursively সল্ভ করতে পারব। আমাদেরকে আগে ঠিক করা লাগবে 0 থাকবে কি থাকবে না। 0 থাকা আর না থাকা এই দুইটা কেইস এর জন্য ঠিক করা লাগবে 1 থাকবে কি থাকবে। প্রত্যেকটা কেইস এর জন্য আবার ঠিক করা লাগবে 2 থাকবে কি থাকবে না। এভাবে n বার করা লাগবে। কোড:

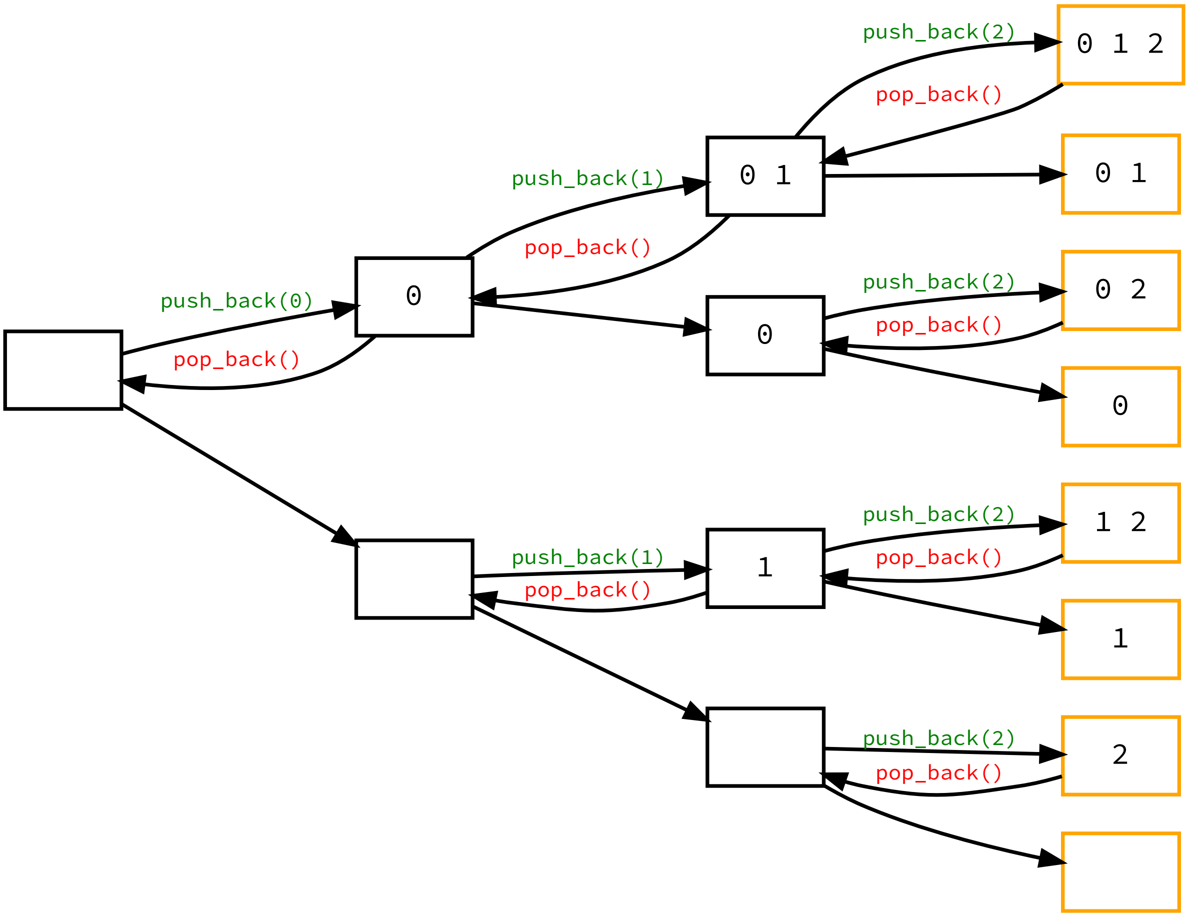
```
void bf(int k){
    if(k == n){
        for(int x: ss) cout << x << " ";
        cout << "\n";
        //do something else with ss
```

```
        return;
    }

    ss.pb(k);
    bf(k+1);
    ss.pop_back();
    bf(k+1);
}

int main(){
    n = 3;
    bf(0);
    /*output:
    0 1 2
    0 1
    0 2
    0
    1 2
    1
    2
        //this one is empty subset
    */
}
```

এই কোডটা কিভাবে কাজ করে তা নিচের ছবিটা দেখলে পরিষ্কার হওয়ার কথা।



চিত্র 1: Generating all subsets

এর time complexity $O(2^n)$

আরেকভাবে subsets generate করা যায়। আমরা একটা সাবসেটকে n সাইজের binary string আকারে প্রকাশ করতে পারি। যদি এখানে i তম সংখ্যা থাকে তাহলে binary string এর i তম bit 1 হবে, নাহলে 0 হবে। $\{0, 1, 2, 3, 4, 5\}$ এর সাবসেট $\{0, 1, 3, 5\}$ কে 110101 আকরে প্রকাশ করা যায়। সাবসেটকে থেকে যেমন binary string বের করতে পারব, তেমনই binary string বা কোনো এক সংখ্যার binary form থেকেও সাবসেট বের করতে পারব। তোমরা যদি binary number সম্পর্কে ভালো ভাবে জেনে থাক তাহলে বুঝবা 0 থেকে $2^n - 1$ এর মধ্যকার সংখ্যাগুলার binary representaiton এ n সাইজ এর সকল binary string থাকে। আমাদের শুধু 0 থেকে $2^n - 1$ পর্জন্ত লুপ চালালেই হবে। নিচের কোডে `1<<n` এর মান হলো 2^n আর যদি কোনো সংখ্যা x এর binary representaiton এর iতম bit 0 হয় তাহলে `x&(1<<i)` 0 হবে, নাহলে `1<<i` বা 2^i হবে। এগুলা সম্পর্কে বিটমাস্ক ক্লাসে বিস্তারিত আলোচনা করা হবে।

```
int n = 4;

for(int i = 0; i < (1<<n); i++){
    vector<int> ss;
    for(int j = 0; j < n; j++) if((i&(1<<j)) != 0) ss.push_back(j);

    for(int x: ss) cout << x << " "; cout << "\n";
    //do something else with ss
}
/*output:
    //this one is empty subset
0
1
0 1
2
0 2
1 2
0 1 2
3
0 3
1 3
0 1 3
2 3
0 2 3
1 2 3
0 1 2 3
*/
```

Practice Problem: Apple Division

CSES এর এই প্রব্লেমের কথা চিন্তা কর। তোমাকে n টা আপেলের ওজন দেওয়া আছে। তোমাকে এদের দুইটা ভাগে ভাগ করা লাগবে যাতে এই দুই ভাগের আপেলের ওজনের সমষ্টির পার্থক্য সর্বনিম্ন হয়।

আমরা এটাকে bruteforce ব্যবহার করে সল্ভ করতে পারব। আমরা আপেলগুলার সব সাবসেট বের করে নিব। প্রত্যেকটা সাবসেটের জন্য চিন্তা করব যদি এটা ভাগ ১ হয় তাহলে ভাগ ১ আর ভাগ ২ এর সমষ্টির পার্থক্য কত। আমাদের উত্তর হবে এই সব পার্থক্যের সর্বনিম্ন সংখ্যা।

কোড (আগে নিজে চেষ্টা কর)

2 Generating Permutaions

প্রব্লেম: তোমাকে একটা সংখ্যা n দেয়া হয়েছে, তোমাকে $\{0, 1, 2, \cdots n-1\}$ এর সকল Permutaion বের করা লাগবে। যেমন: $n = 3$ হলে উত্তর হবে $\{\{0, 1, 2\}, \{0, 2, 1\}, \{1, 0, 2\}, \{1, 2, 0\}, \{2, 0, 1\}, \{2, 1, 0\}\}$।

আমরা আগের মত করে permutaions বের করতে পারব। আমরা n টা সংখ্যার মধ্যে থেকে প্রথমটা ঠিক করব। আমাদের n টা choice আছে। প্রত্যেকটা choice এর জন্য আমদের দ্বিতীয় সংখ্যার জন্য $n-1$ টা choice আছে। এভাবে n বার করে আমার সব permutaion বের করতে পারব।

কিন্তু C++ এ এরচেয়ে ভালো উপায় আছে। `next_permutaion` নামের একটা ফাংকশান আছে। এটা কোনো এক অ্যারে (অথবা এরকম কোনো container) এর lexicograhic order এ পরবর্তী সবচেয়ে বড় permutaion generate করে। যদি এটাই lexicograhically সবচেয়ে বড় permutaion হয় তাহলে এটা false রিটার্ন করে। নাহলে true রিটার্ন করে (`next_permutaion` কোনো অ্যারে রিটার্ন করে না, `std::sort`, `std::reverse` এর মতো মুল অ্যারেকেই মডিফাই করে দেয়)।

যেমন:

- permutaion $\{0, 1, 2, 3, 4\}$ এর `next_permutaion` $\{0, 1, 2, 4, 3\}$
- permutaion $\{0, 1, 2, 4, 3\}$ এর `next_permutaion` $\{0, 1, 3, 2, 4\}$
- permutaion $\{0, 2, 1, 4, 3\}$ এর `next_permutaion` $\{0, 2, 3, 1, 4\}$
- permutaion $\{4, 3, 2, 1, 0\}$ এর `next_permutaion` নাই।

আবার আমরা $\{0, 1, 2\}$ কে `next_permutaion` false না হয়া পর্জন্ত `next_permutaion` করলে পাব। $\{\{0, 1, 2\}, \{0, 2, 1\}, \{1, 0, 2\}, \{1, 2, 0\}, \{2, 0, 1\}, \{2, 1, 0\}\}$। এখানে এর সব permutaion এ আছে। এভাবে আমরা সব permutaion generate করতে পারব।

```
do{
    for(int x: v) cout << x << " "; cout << "\n";
    // do something else with v
}while(next_permutation(v.begin(), v.end()));

/* output:
0 1 2
0 2 1
1 0 2
1 2 0
2 0 1
2 1 0
*/
```

যেহেতু `next_permutaion` এর time complexity $O(n)$ তাই এর time complexity $O(n! \cdot n)$।

উল্যেখ্য যে অ্যারেতে সব উপাদান আলাদা না হলেও `next_permutaion` কাজ করবে। যেমন:

```
vector<int> v = {1, 1, 2, 10};
```

```
do{
    for(int x: v) cout << x << " "; cout << "\n";
    // do something else with v
}while(next_permutation(v.begin(), v.end()));

/* output:
1 1 2 10
1 1 10 2
1 2 1 10
1 2 10 1
1 10 1 2
1 10 2 1
2 1 1 10
2 1 10 1
2 10 1 1
10 1 1 2
10 1 2 1
10 2 1 1
*/
```

কিন্তু যেমন তেমন অ্যারের সব permutaion বের করতে **অবশ্যই আগে অ্যারেটাকে সর্ট করে নিতে হবে**।

3 Meet in the middle

CSES এর এই প্রব্লেমের কথা চিন্তা কর। তোমাকে n টা সংখ্যার একটা অ্যারে দেয়া আছে, তোমাকে বের করতে হবে এমন কতগুলা সাবসেট আছে যাদের সমষ্টি x। এটা তো খুবই সহজ। আমরা প্রত্যেকটা সাবসেট বের করে তাদের সমষ্টি x হয় কি না দেখলেই হয়। কিন্তু সমস্যা হলো এখানে $n \leq 40$ আর $2^{40} = 1099511627776$ যেটাতে TLE হওয়া প্রায় নিশ্চিত।

আমরা এই ধরনের প্রব্লেম এ meet in the middle ব্যবহার করে থাকি যেখানে n প্রায় 40 এর কাছাকাছি হয়। আমরা এই অ্যারেকে দুইটা সমান (বিজোড় হলে একটাতে ১ টা সংখ্যা বেশি) আরেতে ভাগ করব। ধর অ্যারে দুইটা a আর b। যতগুলা সাবসেট যাদের সমষ্টি x তাদের প্রত্যেকের একটা অংশ a তে অ্যারে আরেকটা অংশ b তে থাকবে (শুধু a তে থাকলে b এর অংশকে ফাঁকা সেট ধরে নিব, একই কাজ শুধু b তে থাকলেও করব)। a আর b উভয়েই প্রায় $\frac{n}{2}$ টা সংখ্যা আছে। আমরা আগে শিখানো উপায় ব্যবহার করে a আর b এর সকল সাবসেটের সমষ্টি বের করে নিতে পারব। ধর S_a আর S_b হলো যথাক্রমে a আর b এর সকল সাবসেটের এর সমষ্টি। যেমন: $a = \{1, 2, 4\}$ হলে a এর সব সাবসেট হবে $\{\{\}, \{1\}, \{2\}, \{4\}, \{1, 2\}, \{1, 4\}, \{2, 4\}, \{1, 2, 4\}\}$ এবং S_a হবে $\{0, 1, 2, 4, 3, 5, 6, 7\}$। এখন আমাদের শুধু count করা লাগবে এমন কত গুলা (i, j) আছে যাতে $S_{a_i} + S_{b_j} = x$ হয়। S_a আর S_b উভয়েই প্রায় $2^{\frac{n}{2}}$ টা সংখ্যা আছে এবং প্রত্যেকটার সাথে প্রত্যেকটার তুলনা করতে গেলে আমরা $2^{\frac{n}{2}} \cdot 2^{\frac{n}{2}} = 2^n$ টা তুলনা করে বসব এবং আমাদের complexity যা ছিলো তাই থেকে যাবে।

এবার পালা meet in the middle এর মুল বিষয়ের। আমরা কিন্তু শুধু $O(2^{\frac{n}{2}} \log 2^{\frac{n}{2}})$ (বা $O(\frac{n}{2} \cdot 2^{\frac{n}{2}}$ যেহেতু log এর base 2) এ count করতে পারব। $S_{a_i} + S_{b_j} = x$ হলে $S_{b_j} = x - S_{a_i}$। আমাদের তাই প্রত্যেক S_{a_i} জন্য আমাদের count করতে হবে S_b তে কয়টা সংখ্যা আছে যাদের মান $x - S_{a_i}$ এর সমান। এটা সর্ট করে binary search দিয়ে করা যাবে।

আমরা meet in the middle এ আমরা এই কাজই করে থাকি। আমরা অ্যারেটাকে দুইভাগে ভাগ করি এবং তাদের binary search বা map বা similiar কিছু দিয়ে combine করি।

উপরের উদাহরণের জন্য কোডটা হবে এরকম।

উল্লেখ্য যে এই প্রব্লেমটাকে শুধু $O(2^{\frac{n}{2}})$ তেও সল্ভ করা যেত merge algorithm ব্যবহার করে। কিন্তু ওই idea তা একটু complex আর এটা AC করার জন্য $O(2^{\frac{n}{2}} \log 2^{\frac{n}{2}})$ ই যথেষ্ট।

Practice Problem: এই প্রব্লেমটা নিজে নিজে সল্ভ করার চেষ্টা কর।