به نام خدا

داک آموزشی رویداد گلابی

مرحله دوم

دانشکده مهندسی کامپیوتر

دانشگاه صنعتی شریف

نیم سال اول ۰۱ـ۰۰

دبیر رویداد:
محمدطه جهانی نژاد

مسئول مرحله دوم:
حسین گلی

طراحان داک آموزشی مرحله دوم:
علی مرادی شهمیری
علی پاشا منتصری
دادبه توانایی

ویراستاران داک آموزشی مرحله دوم:
علی مرادی شهمیری
علی پاشا منتصری
دادبه توانایی
ناصر کاظمی

مسئول لتک داک:
ناصر کاظمی

فهرست

مطالب ۲
بخش ۱. جستجوی بازگشتی و روش پس‌گرد (Backtracking) ۲
بخش ۲. جست‌وجوی پهنا نخست یا BFS ۶
بخش ۳. جست‌وجوی ژرفا نخست یا DFS ۱۰

مطالب

بخش ۱. جستجوی بازگشتی و روش پس گرد (*Backtracking*)

روش **پس‌گرد** یا **Backtracking** روشی کلی است که از آن می‌توان در حل مسائل جستجو استفاده کرد. به طور کلی در این روش سعی می‌کنیم که جواب مورد نظر مسئله را مرحله به مرحله و به صورت افزایشی بسازیم بدین شکل که ابتدا با یک کاندید خالی برای جواب شروع می‌کنیم و در هر مرحله سعی می‌کنیم عنصری به این کاندید بیفزاییم و آن را کامل‌تر کنیم تا در نهایت به جوابی برای مسئله تبدیل شود.

ممکن است در طی ساختن جواب در یک مرحله عنصری به کاندید ناتمام جواب اضافه کنیم که باعث شود کاندید شرایط مورد نظر برای جواب مسئله را از دست بدهد. در این صورت متوجه می‌شویم که این کاندید توانایی تبدیل شدن به جوابی برای مسئله را ندارد و آن را رها می‌کنیم.

می‌توان این فرایند را به صورت یک درخت تصور کرد که راس‌های آن کاندیدهای جواب هستند و هر یک از فرزندان هر کاندید هم کاندیدهای دیگری برای جواب هستند که با افزودن یک عنصر به کاندید پدر به دست می‌آیند و برگ‌های درخت نیز یا کاندیدهایی هستند که توانایی کامل‌تر شدن و گسترش یافتن به جواب را ندارند و یا جواب مسئله هستند.

همانطور که گفته شد از این روش می‌توان برای یافتن جواب مطلوبی برای مسائل جستجو استفاده کرد. مسئله n وزیر در این زمینه مثال مشهوری است. در این مسئله هدف این است که چینشی برای n وزیر در یک جدول $n \times n$ پیدا کنیم به گونه‌ای که هیچ دو وزیری یکدیگر را تهدید نکنند.

برای حل این مسئله به روش بک‌ترک می‌توانیم از یک جدول خالی شروع کنیم و با شروع از سطر اول در هر مرحله در خانه‌ای از سطری که روی آن قرار داریم یک وزیر بگذاریم به گونه‌ای که وزیرهایی که قبلاً گذاشته‌ایم را تهدید نکند و سپس به سطر بعدی برویم و مسئله را به شکل بازگشتی حل کنیم. در صورتی که وزیر را در هیچ خانه‌ای از سطر کنونی نتوانیم قرار دهیم یعنی این چینش کاندید مناسبی برای جواب نیست و آن را رها می‌کنیم. در صورتی که در همه‌ی سطرها وزیر قرار دهیم نیز به پاسخ مسئله رسیده‌ایم.

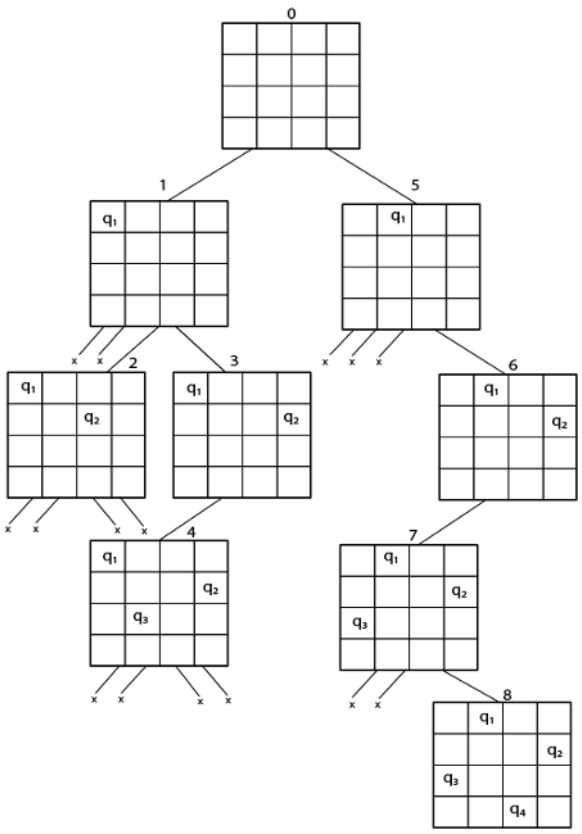
در ادامه کدی که این الگوریتم را پیاده‌سازی می‌کند مشاهده می‌کنیم:

```c
#include <stdbool.h>
#include <stdio.h>

#define N 20

void printBoard(int state[], int n) {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            if (state[i] == j)
                printf("1 ");
            else
                printf("0 ");
        }
        printf("\n");
    }
}

// check if the queen in the ith row is threatened by the queens in the rows above
bool checkIthQueen(int state[], int i) {
    for (int j = 0; j < i; j++) {
        // check vertically
        if (state[j] == state[i]) return false;
        // check diagonally
        if (state[j] == state[i] - (i - j) || state[j] == state[i] + (i - j)) return false;
    }
    return true;
}

// returns true if reaches a valid placement of queens and false otherwise
bool solve(int state[], int n, int row) {
    if (row == n) return true;
    for (int col = 0; col < n; col++) {
        state[row] = col;
        if (checkIthQueen(state, row) && solve(state, n, row + 1))
    return true;
    }
    return false;
}

int state[N];

```

```
41 int main() {
42     int n;
43     scanf("%d", &n);
44     bool result = solve(state, n, 0);
45     if (result)
46         printBoard(state, n);
47     else
48         printf("No solution exists!\n");
49 }
```

در کد فرض کردیم که n ای که در ورودی داده می‌شود حداکثر ۲۰ است. همچنین برای ذخیره کردن کاندید جواب از آرایه‌ی state استفاده کردیم که state[i] شخص می‌کند که وزیر مستقر در سطر I ام در ستون چندم قرار دارد (سطرها و ستون‌ها را با شروع از صفر شماره‌گذاری می‌کنیم).

همچنین درصورتی که n برابر ۴ باشد درخت جستجوی پس‌گرد به شکل زیر خواهد بود:



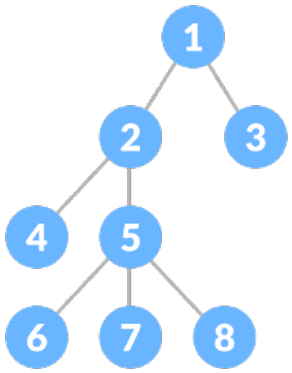
مسئله دیگری که می‌توان با روش پس‌گرد حل کرد پیدا کردن زیرمجموعه‌ای از اعداد داده شده با مجموعی مشخص است. صورت مسئله این است که اعداد طبیعی $a_1, a_2, ..., a_n$ و عدد s در ورودی داده شده‌اند و می‌خواهیم زیرمجموعه‌ای از این اعداد پیدا کنیم که مجموع اعداد حاضر در آن برابر با s باشد یا بگوییم که چنین زیرمجموعه‌ای وجود ندارد. این مسئله را می‌توانید با روشی مشابه با روش مطرح شده برای n وزیر حل کنید. مسائل دیگری از جمله پر کردن جدول سودوکو و ... نیز با همین روش قابل حل‌اند.

بخش ۲. جست‌وجوی پهنا نخست یا *BFS*

BFS (Breadth-First Search) یک الگوریتم برای جست‌وجو در گراف و درخت است. قبل از توضیح این الگوریتم ابتدا با درخت آشنا می‌شویم.

درخت:

درخت به گرافی گفته می‌شود که میان هر دو گره‌ای تنها یک راه وجود داشته باشد و در آن دور یا طوق نیز وجود ندارد. همچنین در درخت گره‌ای وجود دارد که به آن ریشه یا Root گفته می‌شود و در بالاترین سطح قرار می‌گیرد. شکل زیر نمونه‌ای از یک درخت را نشان می‌دهد:

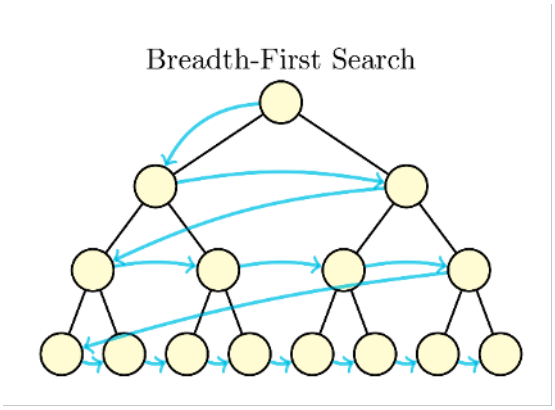


در شکل بالا گره شماره ۱ ریشه است. همچنین به گره‌هایی که در زیر یک گره وجود دارند و به آن متصل‌اند فرزندان آن گره گفته می‌شود. به عنوان مثال گره‌های شماره ۴ و ۵ فرزندان گره ۲ می‌باشند.

الگوریتم BFS:

حال به معرفی الگوریتم جست‌وجوی پهنا نخست یا همان BFS می‌پردازیم. این الگوریتم به منظور یافتن عنصری خاص که مد نظر ما است استفاده می‌شود. بدین گونه که ابتدا با شروع از ریشه بررسی می‌کند که آیا عنصر مد نظر ما در این گره قرار دارد یا خیر. اگر که برابر بود آن را بر می‌گرداند در غیر این صورت به سطح بعدی رفته و گره‌های آن سطح را از چپ به راست بررسی می‌کند و به همین شکل تا زمانی که عنصر مد نظر را پیدا کند ادامه می‌دهد.

در شکل زیر نحوه به کارگیری این الگوریتم به تصویر کشیده شده است:



در تصویر فوق فلش‌های آبی نشان‌گر مسیری هستند که الگوریتم BFS برای بررسی گره‌ها طی می‌کند. پس از آنکه گره‌های هم سطح بررسی شدند نوبت به فرزندان آن‌ها می‌رسد که در سطح بعدی قرار دارند. به عنوان مثال اگر بخواهیم که گره شماره چهار را در شکل اول با استفاده از BFS پیدا کنیم به صورت زیر عمل می‌کنیم:

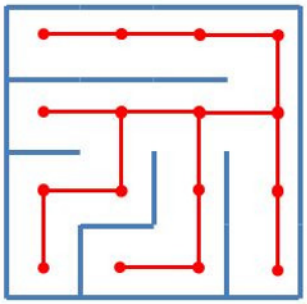
۱. ابتدا از ریشه شروع می‌کنیم. از آنجایی که ریشه گره مورد نظر ما نیست به سطح بعدی می‌رویم.

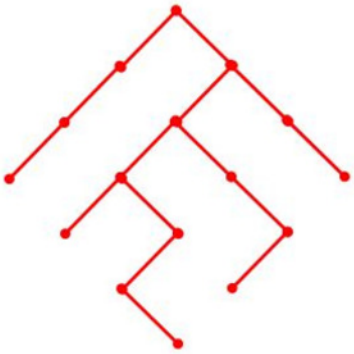
۲. حال گره شماره ۲ را بررسی می‌کنیم و از آنجایی که برابر ۴ نیست به سراغ گره شماره ۳ می‌رویم و دوباره به علت نابرابری از روی آن رد می‌شویم.

۳. حال که سطح دوم را بررسی کردیم به سطح سه می‌رویم. از آنجایی که اولین گره ای که بررسی می‌کنیم گره شماره ۴ و مد نظر ما است در نتیجه آن را برگردانده و برنامه را پایان می‌دهیم.

نمونه ای کاربردی:

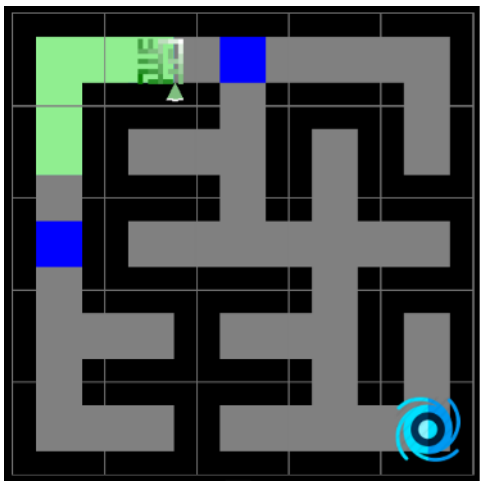
یکی از زمینه‌هایی که در آن می‌توان از BFS استفاده کرد حل ماز می‌باشد. از آنجایی که در مازها بین هر خانه تنها یک راه وجود دارد در نتیجه می‌توان مازها را مانند یک درخت دید:

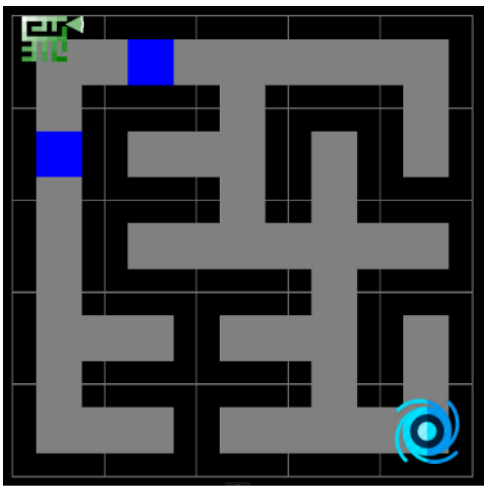


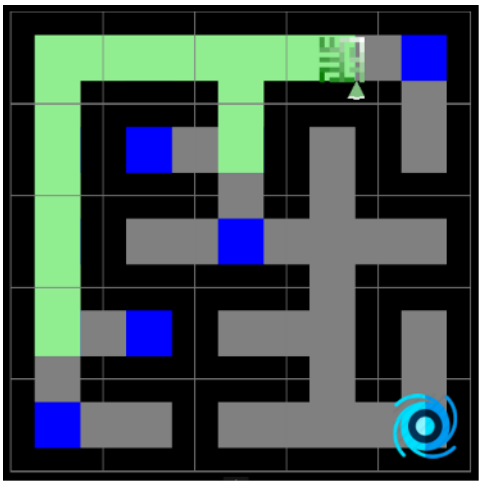


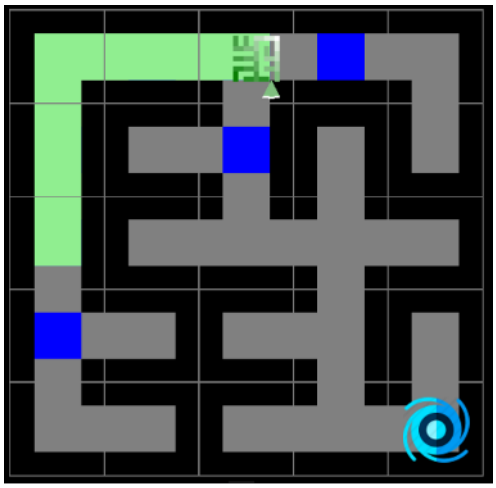
همان‌گونه که در شکل مشخص است اکنون ماز مورد نظر به یک درخت معادل تبدیل شده است. در نتیجه برای رسیدن از نقطه ورود به نقطه خروج با استفاده از BFS سطوح درخت را به ترتیب بررسی می‌کنیم اگر که گره بررسی شده برابر با نقطه خروج باشد آنگاه به صورت بازگشتی به نقطه ورود بازگشته و راه خروج از ماز را می‌نویسیم.

یک نمونه از حل ماز با استفاده از BFS در زیر آمده است:

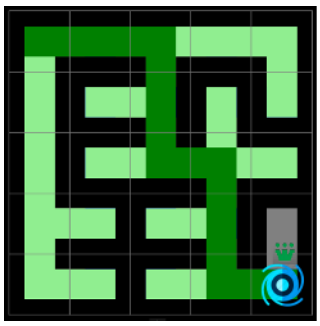








در نهایت با ادامه این روند خواهیم داشت:



بخش ۳. جست‌وجوی ژرفا نخست یا *DFS*

الگوریتم dfs یکی از پایه‌ای‌ترین روش‌های پیمایش گراف است. برای اینکه الگوریتم را توضیح بدهیم با یک مثال شروع می‌کنیم.

فرض کنید در یک هزارتو گیر کرده‌اید که به صورت یک گراف است. یعنی در هر راس گراف یک اتاق قرار دارد و هر یال نشان‌دهنده یک راهرو بین دو اتاق است. همچنین حافظه شما به قدری قوی است که می‌توانید اگر به یک اتاق تکراری رفتید تشخیص بدهید که این اتاق تکراری است و هنگامی که در یک اتاق هستید تنها می‌توانید راهروهای مجاور آن را ببینید. همچنین یک نخ به همراه دارید که یک سر آن به اتاقی که اول کار در آن قرار دارید بسته شده است و سر دیگر در دستان شماست. در یکی از راس‌های گراف گنجی قرار دارد. هدف شما این است که گنج را بیابید. چگونه این کار را انجام می‌دهید؟

پیدا کردن گنج به سادگی انجام الگوریتم زیر است. تا زمانی که به گنج نرسیدیم الگوریتم زیر را انجام دهید :

- اگر همه اتاق‌های مجاور تکراری بودند به اتاقی برو که برای اولین بار از آن به اتاق فعلی آمده‌ای. (کافی است نخی که دستمان است را دنبال کنیم).
- در غیر این صورت به یکی از اتاق‌های مجاور که تکراری نیست برو.

چرا این الگوریتم مسئله ما را حل می‌کند؟ نکته اینجاست که زمانی که ما برای اولین بار در یک اتاق قرار می‌گیریم تمام تلاشمان را می‌کنیم که از آن اتاق مسیری به گنج پیدا کنیم. در نتیجه وقتی که همه اتاق‌های مجاور تکراری می‌شوند و ما نخ را دنبال کرده و بر می‌گردیم می‌توان نتیجه گرفت که هیچ مسیری از آن اتاق به گنج وجود ندارد. در نتیجه هیچ گاه دیگر نباید وارد این اتاق شویم. (و این منطق که نباید وارد اتاق تکراری شویم نیز از همینجا ناشی می‌شود).

حال آنچه در این قسمت بررسی می‌کنیم تصویری کلی از الگوریتم dfs است. فرض کنید آرایه mark نشان می‌دهد که چه راس‌هایی قبلا دیده شده‌اند و در ابتدای کار تمام خانه‌های آن false است و همینطور g[x] هم لیستی از راس‌هایی است که با x مجاور هستند. حالا الگوریتم ما به این صورت خواهد بود.

```
void dfs(int u){
    mark[u] = true;
    for(int y : g[u]) {
        if(mark[y] == false) {
            dfs(y);
        }
    }
}
```

مثلا برای مثالی که زدیم زمانی که dfs به خانه‌ای می‌رسد که در آن گنج بوده می‌توانیم آن را گزارش دهیم.

برای اطلاعات بیشتر هم می‌توانید این لینک را بررسی کنید.